The ATM Forum Technical Committee

Network Management

M4 Network View Interface CORBA Specification

af-nm-0166.000

August 2001

CORBA Specification for M4 Interface: Network View

AF-NM-0166.000

The ATM Forum
Presidio of San Francisco
P.O. Box 29920 (mail)
572B Ruger Street (surface)
San Francisco, CA 94129-0920
Tel:+1-415.561.6275
Fax: +1-415.561.6120

Acknowledgements

The following people participated in the development of the M4 Network View Interface CORBA Specification:

Rajesh Abbi
Keith Allen
Karen Armington
Beau Atwater
Hani Hawari
An-Ni Huynh
Annette Ihrke
Patrice Lamy
Thomas Meserole
Todd Schumacher
Anirban Sharma
Pawan Saxena
Brian Thorstad
Homayoun Torab
Zhenxin Wang
Kin Wong

Weijing Chen, C. Anthony Cooper, Andrew J. Mayer, Atahan Tuzel, Editors
Roger Kosak, former Chairman of the Network Management Group
Atahan Tuzel, Chairman of the Network Management Group

Table of Contents

Table of Figures and Tables

1 Introduction

This document specifies a CORBA-based ATM network management interface definition that provides a formal representation of the information exchanged between a managed system and a managing system. The interface specification was defined specifically to meet the criteria set forth by ATM Forum "M4 Interface Requirements and Logical MIB: ATM Network View" version 2 (af-nm-0058.001)[1].

1.1 Scope

This document was developed using the following principles:

- **Directly define CORBA IDL from "M4 Interface Requirement and Logical MIB: ATM Network View (version 2)" [1] using existing CMIP MIB as reference when need. The based CORBA specification is revision 2.2 February 1998 [3].**
- **This interface specification not only defines CORBA IDL, but also specifies the recommended usage of CORBA Common Object Services [4][5][6][7] that would impact IDL definition and system interoperability.**

The requirements specified in the Appendix A and IDL definitions specified in the Appendix B are essential parts of the "M4 Network View Interface CORBA Specification" and shall have same mandatory status as the main text of this document. The IDL given in this document has been successfully tested with two commercial compilers. Note that other work in progress may be relevant, specifically work in T1M1.5 and work in ITU-T Q.19/4.

1.2 Definitions

This section provides for any specific definitions needed in this document.

CORBA Name:
A *CORBA name* is an ordered sequence of CORBA name components. Each component except the last is used to name a CORBA naming context. The last component denotes the bound IDL object.

CORBA Name Binding:
A CORBA name to IDL object association is called a *CORBA name binding*. A CORBA name binding is always defined relative to a CORBA naming context.

CORBA Name Component:
A *CORBA name component* consists of two attributes: the ID attribute and the kind attribute. Both the ID attribute and the kind attribute are represented as IDL strings. Appendix D provides guidelines for the syntax of the ID attribute of the name component. The kind attribute adds descriptive power to names in a syntax-independent way, although the naming system does not interpret, assign, or manage kind value in any way.

CORBA Naming Context:
A *CORBA naming context* is a CORBA object that contains a set of CORBA name bindings in which each CORBA name is unique. Different CORBA names can be bound to a CORBA object in the same or different contexts at the same time.

IDL Object Interface (IOI):
An *IDL object interface* is a description of a set of attributes and possible operations that a client may request of an IDL object in a CORBA system. An IDL object satisfies an interface if it can be specified as the target object in each potential request described by the interface.

IDL Object (IO):
An *IDL object* is an identifiable, encapsulated entity that provides one or more services that can be requested by a client in a CORBA system.

IDL(or Interoperable) Object Reference (IOR):
An IOR is a sequence of object-specific protocol profiles, plus a type ID, which unambiguously identifies a managed object. (See Section 11.6.2 and Glossary of [3]).

Managed Object Class (MOC):
The class of all Managed Objects of the same type, e.g., the class of all Connection Termination Points.

Managed Object (MO):
A specific entity for which information may be exchanged over the CORBA interface between the managed system and the managing system.

1.3 Grain-neutral Approach

In the past CORBA implementations were known to have problems scaling up to very large numbers of objects. This was one of the problems addressed by the OMG's CORBA 2.2 release. Even though there are some CORBA 2.2-compliant ORBs available now, it may take other suppliers awhile to implement the new capabilities. To accommodate the network management system implementers that base their systems on these suppliers, a few information modeling conventions are proposed that will enable these systems to instantiate only a small number of objects. The goal of these conventions is to use what would otherwise be considered a "fine-grained" model, but to make minor changes that will allow managed systems to instantiate only one object per object class. Since the number of classes implemented in a system will be small, typically well under one hundred, the number of object instances will be small and pre-CORBA 2.2 scalability concerns should be alleviated. Implementers that prefer may instantiate many objects per class, making the model *granularity neutral*, at least from a number-of-instances point of view.

The following conventions are proposed for use in making fine-grained models implementable with one object per class:

Name bindings are created for every "fine-grained" instance. That is, there will be no difference in the number of name bindings for a system that is implemented with "one instance-per-object" and one that is implemented with "one-instance-per-class." In the one-instance-per-class implementation, all of the name bindings for objects of the same class will be bound to the same object reference.

Every object operation includes a parameter passing in the name of the object. In systems implemented with one instance per object this value will be redundant since that object doesn't need to be told its name. In one-object-per-class systems, however, this parameter will identify the actual target for the operation.

Wherever an object reference would be passed across the interface instead a structure containing both the reference and the object's name should be passed. In one-instance-per-class systems the reference alone does not really identify an object. The name is needed, too.

The conventions above are a compromise, not a perfect solution. Putting potentially millions of name bindings in a Name Service will raise scalability concerns with some, and submitting names to every operation and returning structures with both names and references is somewhat clumsy. This is, however, a good compromise. It will enable the volumes of work done to date on network management information modeling to be more easily re-used, speeding the introduction of CORBA-based network management while addressing the scalability concerns of some CORBA ORB implementations.

2 CORBA Modeling and IDL Definition Guidelines

Within the context of this document, recommendations are presented as either *requirements* denoted by **(R),** *conditional requirements* denoted by **(CR),** or *objectives* denoted by **(O).** In this document, *requirements* are considered functions that are necessary for operational compatibility; while *objectives* are considered features that are viewed to be desirable but not essential for management system to work. *Conditional requirements* are functions that are necessary for operational compatibility of an optional feature such as asynchronous method invocation (i.e., If the feature is supported by the managed system, then the CR is a requirement).

Generally, CORBA modeling and IDL definitions for network management shall follow the guidelines listed below:

(R) GUIDE-1 There shall be one IDL object interface (IOI) per managed object class (MOC).

(R) GUIDE-2 Shall use CORBA Naming Service to represent containment relationship of managed objects. There shall be one unique CORBA name per managed object. This CORBA name shall bind to an IDL object of this managed object's IDL object interface.

(R) GUIDE-3 Each operation method of an IDL object interface shall include a CORBA name relative to the root context as the first parameter, i.e. `op(name, parameters)` to unique identify the target managed object.

(R) GUIDE-4 A structure packed with IDL object reference (IOR) and CORBA name relative to the root context shall be used in place of pure IDL object reference in *most case*, i.e. for generic IDL object:

```
struct ObjectID {Object ref; CosNaming::Name name;};
```

and for interface specific IDL object:

```
struct <interface name>ID {<interface name> ref; CosNaming::Name name;};
```

The `ref` shall not be null IDL object reference in *any situation*. In few case such as bulk operation `getAll...()` the CORBA name only could be used instead of above structure. The choice shall be based on performance.

(R) GUIDE-5 The numbers of IDL objects (IO) in a managed system is implementation dependent as long as all the IDL object interfaces are accessible by the manager system. IDL definition shall be the same regardless a particular implementation("grain-neutral").

(R) GUIDE-6 Shall use semantics of values to represent optional attributes unless it can not be done so, ex. `short` value, then use `union` to represent optional.

(R) GUIDE-7 Shall use `NotSupported` CORBA exception to indicate the optional operation. If managed system does not implement an optional operation, it shall emit the exception back to requester.

(R) GUIDE-8 Shall use `const` to declare the constant values unless the set of values is quite stable and will not change in foreseeable future, then shall use `enum` to declare.

2.1 Use of the OMG Messaging Service

The approach used in this document requires implementation of the OMG Asynchronous Invocation Method Messaging Service [7].

3 Summary of Required Object Classes

Table 3-1 summarizes the set of CORBA IDL object interface that is needed to meet the requirements specified in ATM Forum "M4 Interface Requirements and Logical MIB: ATM Network View" version 2 (af-nm-0058.001)[1]. The first column of this table lists the logical managed entities defined [1]. The second column lists attributes and operations associated with the corresponding logical managed entity in column one. The third column lists the IDL object interface that corresponds to the logical managed entity in column one. The fourth column lists the attributes and operations associated with the corresponding IDL object interface in column three. Where appropriate, comments are provided in the fifth column. This table is not intended to exhaustively list every IDL attribute and operation given in Section 5 of this document.

Table 3-1. M4 Network View Logical MIB to CORBA IDL Mapping Table

M4 Logical MIB Managed Entity	**M4 Logical MIB** Attribute/**Operation**	**CORBA IDL Object**	**CORBA IDL** Attribute/**Operation**	**Comment**
Network	Network ID	**Network**	Not Supported	
	Relationship with managed resources		*Containment Relationship*	
	query network for contained Managed Entities		**NO CORBA OPERATION**	
vcLayerNetworkDomain	atmLNDId	**AtmLND**	`AtmLNDAllAttr`	
	Signal Identification		`AtmLNDAllAttr` **`getCharacteristicInfo`**	
	User Label		`AtmLNDAllAttr` **`getUserLabel`** **`setUserLabel`**	
	NO LOGICAL MIB ATTRIBUTE MATCH		`lndSystemTitle` **`getSystemTitle`**	
	Relationship with vcTTP		*Containment Relationship*	
	Relationship with vcTrail		*NO CORBA RELATIONSHIP*	
	Relationship with vcSubnetwork		*Containment Relationship*	
	Relationship with vcTrailRequest		*NO CORBA RELATIONSHIP*	
	query vcLayerNetworkDomain for Delimiting vcTTPs		**NO CORBA OPERATION**	
	query vpLayerNetworkDomain for existing vcTrails		**NO CORBA OPERATION**	
	query vcLayerNetworkDomain for component vcSubnetwork		**NO CORBA OPERATION**	
	setup vcTrail		**NO CORBA OPERATION**	
	setup vcTrailRequest		**NO CORBA OPERATION**	
	addTps to Multipoint Trail		**NO CORBA OPERATION**	
	release vcTrail		**NO CORBA OPERATION**	
	make external vcLinkEnd		**`makeLinkEnd`**	
	remove external vcLinkEnd		**NO CORBA OPERATION**	
	setup vcTopologicalLink		**NO CORBA OPERATION**	
	release vcTopologicalLink		**NO CORBA OPERATION**	
vcLinkConnection	vcLinkConnection ID	**AtmLinkConn**		
	Signal Identification			
	Directionality			
	User Label			
	availability Status			
	Administrative State			

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/**Operation**	CORBA IDL Object	CORBA IDL Attribute/**Operation**	Comment
	retainedResource			
	Relationship With vcTopologicalLink		*NO CORBA RELATIONSHIP*	
	Relationship With vcNetworkCTPs		*NO CORBA RELATIONSHIP*	
	query vcLinkConnection for Containing vcTopologicalLink		**NO CORBA OPERATION**	
	query vcLinkConnection for terminating vcNetworkCTPs		**NO CORBA OPERATION**	
vcLinkEnd	vcLinkEnd ID	**AtmLinkEnd**	`AtmLinkEndAllAttr`	
	Administrative State		`AtmLinkEndAllAttr` **`getAdminState` `setAdminState`**	
	Availability Status		`AtmLinkEndAllAttr` **`getAvailabilityStatus`**	
	Egress Maximum Assignable Bandwidth		`AtmLinkEndAllAttr` **`getEgressMaxAssignBW`**	
	Ingress Maximum Assignable Bandwidth		`AtmLinkEndAllAttr` **`getIngressMaxAssignBW`**	
	Egress available Bandwidth		`AtmLinkEndAllAttr` **`getEgressAvailableBW`**	
	Ingress available Bandwidth		`AtmLinkEndAll` **`getIngressAvailableBW`**	
	User Label		`AtmLinkEndAllAttr` **`getUserLabel` `setUserLabel`**	
	Link TP Type		`AtmLinkEndAllAttr` **`getLinkEndType` `setLinkEndType`**	
	NO LOGICAL MIB ATTRIBUTE MATCH		`AtmLinkEndAllAttr` **`getCharacteristicInfo`**	
	Relationship With vcTopologicalLink		`AtmLinkEndAllAttr` **`getSupportedLink`**	
	Relationship With vcLogicallLinkTP		*NO CORBA RELATIONSHIP*	
	Relationship With vcSubnetwork		*NO CORBA RELATIONSHIP*	
	Relationship With serverTTPs		`AtmLinkEndAllAttr` **`getServerTTP` `setServerTTP`**	
	Relationship With vcNetworkAccessProfile		`AtmLinkEndAllAttr` **`getNetworkAccessProfile` `setNetworkAccessProfile`**	
	Relationship With vcNetworkCTP: existing Connection Termination Points		`AtmLinkEndAllAttr` **`getSupportedCTPs` `addSupportedCTP` `removeSupportedCTP`**	
	query vcLinkEnd for Terminated vcTopologicalLink		`AtmLinkEndAllAttr` **`getSupportedLink`**	
	query vcLinkEnd for delineated vcSubnetwork		**NO CORBA OPERATION**	
	query vcLinkEnd for associated vpTTP		`AtmLinkEndAllAttr` **`getServerTTP`**	
	associate vcLinkEnd with supporting vpTTP		**`setServerTTP`**	
	vcLinkEnd PVC Trace		`linkPVCTrace`	
vcLogicalLinkTP	vcLogicalLinkTP ID	**NO M4 IDL OBJECT**		
	Egress Maximum Assignable Bandwidth			
	Ingress Maximum Assignable Bandwidth			
	Egress available Bandwidth			
	Ingress available Bandwidth			
	VCI Range			
	User Label			

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/**Operation**	CORBA IDL Object	CORBA IDL Attribute/**Operation**	Comment
	Relationship With vcTopologicalLink			
	Relationship With vcLinkEnd			
	Relationship With vcSubnetwork			
	Relationship With vcNetworkAccessProfile			
	Relationship With vcNetworkCTP: Existing Connection Termination Points			
	query vcLogicalLinkTP for terminated vcTopologicalLink			
	query vcLogicalLinkTP for delineated vcSubnetwork			
	query vcLogicalLinkTP for associated vcLinkEnds			
	associate vcLogicalLinkTP with supporting vcLinkEnd			
	vcLogicalLinkTP PVC Trace			
vcNetworkAccessProfile	vcNetworkAccessProfile ID	**AtmNetworkAccess Profile** **AtmNetworkAccess ProfileFactory**	`AtmNetworkAccessProfileAllAttr`	
	total Egress Bandwidth		`AtmNetworkAccessProfileAllAttr` **`getTotalEgressBW`** **`setTotalEgressBW`**	
	total Ingress Bandwidth		`AtmNetworkAccessProfileAllAttr` **`getTotalIngressBW`** **`setTotalIngressBW`**	
	maximum Number of Active Connection Allowed		`AtmNetworkAccessProfileAllAttr` **`getMaxNumActiveVcConn`** **`setMaxNumActiveVcConn`**	
	VPI/VCI Range		`AtmNetworkAccessProfileAllAttr` **`getVciRange`** **`setVciRange`**	
vcRoutingProfile	vcRoutingProfile ID	**AtmRoutingProfile**		
	connectionTypeSupported			
	routeDescriptionList			
	maxHops			
	Relationship With vcSubnetwork		*Containment Relationship*	
	setup vcRoutingProfile		**NO CORBA OPERATION**	

vcSubnetwork	Subnetwork ID	**AtmSubnetwork**	`AtmSubnetworkAllAttr`
			`AtmSubnetworkTopo`
			`getSubnetworkID`	

	Signal Identification		`AtmSubnetworkAllAttr` **`getCharacteristicInfo`**	
	user Label		`AtmSubnetworkAllAttr` **`getUserLabel`** **`setUserLabel`**	
	availability Status		`AtmSubnetworkAllAttr` **`getAvailabilityStatus`**	

	supported by Object List		`AtmSubnetworkAllAttr`
			`AtmSubnetworkTopo`
			`getSupportedByObjectList`
			`addSupportedByObjects`
			`removeSupportedByObjects`
			`replaceSupportedByObject List`	

	NO LOGICAL MIB ATTRIBUTE MATCH		`AtmSubnetworkAllAttr` **`getSystemTitle`**	
	Relationship With vcSubnetworkConnection		`AtmSubnetworkAllAttr` **`getContainedSNCs`**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	Relationship With vcSubnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentSubnetworks** **addComponentSubnetwork** **removeComponentSubnetwork**	
	Relationship With vcTopologicalLink		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentLinks** **addComponentLinks** **removeComponentLinks**	
	Relationship With vcLogicalLinkTP		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedLinkTPs** **addSupportedLinkTP** **removeSupportedLinkTP**	
	Relationship With vcLinkEnd		AtmSubnetworkAllAttr AtmSubnetworkTopo	
	query vcSubnetwork for existing vcSubnetworkConnections		AtmSubnetworkAllAttr **getContainedSNCs**	
	query vcSubnetwork for component vcSubnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentSubnetworks** **addComponentSubnetwork** **removeComponentSubnetwork**	
	query vcSubnetwork for vcTopologicalLinks between its component subnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentLinks** **addComponentLinks** **removeComponentLinks**	
	query vcSubnetwork for Connecting vcLinkEnds or vcLogicalLinkTPs		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedLinkTPs** **addSupportedLinkTP** **removeSupportedLinkTP**	
	setup vcSubnetworkConnection		**setupPtToPtSNCWithCTP** **setupPtToPtSNCWithLinkTP** **setupPtToMultiSNCWithCTP** **setupPtToMultiSNCWithLinkTP**	
	modify vcSubnetworkConenction		**NO CORBA OPERATION**	
	addTPs to SunetworkConnection		**addTpToMultiSNCWithCTP** **addTpToMultiSNCWithLinkTP** **removeTpFromMultiSNC**	
	release vcSubnetworkConnection		**releaseSNC**	
vcSubnetworkConnection	vcSubnetworkConnectionID	**AtmSNC**	AtmSNCAllAttr	
	Directionality		**NO CORBA OPERATION**	
	availability Status		AtmSNCAllAttr **getAvailabilityStatus**	
	Administrative Status		AtmSNCAllAttr **getAdminState** **setAdminState**	
	User Label		AtmSNCAllAttr **getUserLabel** **setUserLabel**	
	restorableIndicator		AtmSNCAllAttr **getRestorableIndicator** **setRestorableIndicator**	
	retainedResource		**NO CORBA OPERATION**	
	provisionType		AtmSNCAllAttr **getProvisionType** **setProvisionType**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmSNCAllAttr **getCharacteristicInfo** **getOwnershipName** **setOwnershipName** **getConnectionType**	
	Relationship With networkCTPs		AtmSNCAllAttr **getAtpInstance** **getZtpList**	
	Relationship With subnetworkConnections		AtmSNCAllAttr **getComponentSNCList**	
	Relationship With vcLinkConnections		AtmSNCAllAttr **getComponentLinkConnList**	
	Relationship With routingProfiles		AtmSNCAllAttr **getRoutingProfile** **setRoutingProfile**	
	query subnetworkConnection for terminating networkCTPs		AtmSNCAllAttr **getAtpInstance** **getZtpList**	
	query vcSubnetworkConnection for Component vcSubnetworkConnections		AtmSNCAllAttr **getComponentSNCList**	
	vcSubnetworkConnection Connection Trace		traceSNC	
vcTopologicalLink	vcTopologicalLink ID	**AtmLink**	AtmLinkAllAttr	
	Signal Identification		AtmLinkAllAttr **getCharacteristicInfo**	
	Directionality		**NO CORBA OPERATION**	
	Operational State		AtmLinkAllAttr **getAvailabilityStatus**	
	provisioned Bandwidth		**NO CORBA OPERATION**	
	available Bandwidth		**NO CORBA OPERATION**	
	restorationMode		AtmLinkAllAttr **getRestorationMode** **setRestorationMode**	
	Customer Identification		AtmLinkAllAttr **getCustomerID** **setCustomerID**	
	Weight		AtmLinkAllAttr **getLinkWeight** **setLinkWeight**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmLinkAllAttr **getAdminState** **setAdminState**	
	Relationship With linkConnections		AtmLinkAllAttr **getContainedLinkConns**	
	Relationship With logicalLinkTP		*NO CORBA RELATIONSHIP*	
	Relationship With linkEnd		AtmLinkAllAttr **getALinkEnd** **getZLinkEnd**	
	Relationship With subnetwork		AtmLinkAllAttr getLinkedSubnetworks	
	Relationship With vcNetworkAccessProfile		AtmLinkAllAttr **getNetworkAccessProfile** **setNetworkAccessProfile**	
	query vcTopologicalLink for Contained vcLinkConnections		AtmLinkAllAttr **getContainedLinkConns**	
	query vcTopologicalLink For Terminating vcLinkEnds or vcLogicalLinkTPs		AtmLinkAllAttr **getALinkEnd** **getZLinkEnd**	
	query vcTopologicalLink For Delineated vcSubnetworks		AtmLinkAllAttr **getLinkedSubnetworks**	
	set up vcLinkConnection		**setupLinkConnWithCTP** **setupLinkConnOnLink**	
	modify vcLinkConnection			
	release vcLinkConnection		**releaseLinkConn**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	vcTopologicalLink PVC Trace			
vcTrail	vcTrail ID	**NO M4 IDL OBJECT**		
	signal Identification			
	Directionality			
	User Label			
	Administrative State			
	availability State			
	restorable Indicator			
	retainedResource			
	Relationship With vcNetworkTTP			
	query vcTrail for terminatingTTPs			
	vcTrail Connection Trace			
vcTrailRequest	vpTrailrequest ID	**NO M4 IDL OBJECT**		
	Request Status			
	requestType			
	requestCommittedTime			
	Relationship With vcTrail			
vpLayerNetworkDomain	Signal Identification	**AtmLND**	`AtmLNDAllAttr`	
	NO LOGICAL MIB ATTRIBUTE MATCH		`AtmLNDAllAttr` **`GetCharacteristicInfo`**	
	User Label		`AtmLNDAllAttr` **`getUserLabel`** **`setUserLabel`**	
	NO LOGICAL MIB ATTRIBUTE MATCH		`lndSystemTitle` **`getSystemTitle`**	
	Relationship With vpTTP		*Containment Relationship*	
	Relationship With vpTrail		*NO CORBA RELATIONSHIP*	
	Relationship With vpSubnetwork		*Containment Relationship*	
	Relationship With vpTrailRequest		*NO CORBA RELATIONSHIP*	
	query vpLayerNetworkDomain for Delimiting vpTTPs		*Containment Relationship*	
	query vpLayerNetworkDomain for existing vpTrails		**NO CORBA OPERATION**	
	query vpLayerNetworkDomain for component vpSubnetwork		*Containment Relationship*	
	setup vpTrail		**NO CORBA OPERATION**	
	setup vpTrailRequest		**NO CORBA OPERATION**	
	addTps To Multipoint Trail		**NO CORBA OPERATION**	
	release vpTrail		**NO CORBA OPERATION**	
	Make External vpLinkEnd		**makeLinkEnd**	
	Remove External vpLinkEnd		**NO CORBA OPERATION**	
	setup vpTopologicalLink		**NO CORBA OPERATION**	
	release vpTopologicalLink		**NO CORBA OPERATION**	
vpLinkConnection	vpLinkConnection ID	**AtmLinkConn**		
	Signal Identification			
	Directionality			
	User Label			
	availability Status			
	Administrative State			
	retainedResource			
	Relationship With vpTopologicalLink			
	Relationship With vpNetworkCTPs			
	query vpLinkConnection for containing vpTopologicalLink			

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	query vpLinkConnection For terminating vpNetworkCTPs			
vpLinkEnd	vpLinkEnd ID	**AtmLinkEnd**	AtmLinkEndAllAttr	
	Administrative State		AtmLinkEndAllAttr **getAdminState** **setAdminState**	
	Availability Status		AtmLinkEndAllAttr **getAvailabilityStatus**	
	Egress Maximum Assignable Bandwidth		AtmLinkEndAllAttr **getEgressMaxAssignBW**	
	Ingress Maximum Assignable Bandwidth		AtmLinkEndAllAttr **getIngressMaxAssignBW**	
	Egress available Bandwidth		AtmLinkEndAllAttr **getEgressAvailableBW**	
	Ingress available Bandwidth		AtmLinkEndAllAttr **getIngressAvailableBW**	
	User Label		AtmLinkEndAllAttr **getUserLabel** **setUserLabel**	
	Link TP Type		AtmLinkEndAllAttr **getLinkEndType** **setLinkEndType**	
	Loopback Location Identifier		AtmLinkEndAllAttr **getLoopbackLocID** **setLoopbackLocID**	
	ILMI Virtual Identifier		AtmLinkEndAllAttr **getIlmiVpiVci** **setIlmiVpiVci**	
	Supporting NE Location		AtmLinkEndAllAttr **getSupportingNeLoc** **setSupportingNeLoc**	
	Supporting Circuit Pack Location		AtmLinkEndAllAttr **getSupportingPortID** **setSupportingPortID**	
	Server TTP Name		AtmLinkEndAllAttr **getServerTTP** **setServerTTP**	
	Server TTP Characteristic Information Type		AtmLinkEndAllAttr **getServerTTPCharInfo**	
	Server TTP Port Id		AtmLinkEndAllAttr **getServerTTPPortID** **setServerTTPPortID**	
	Server TTP Operational State		AtmLinkEndAllAttr **getServerTTPOpState**	
	Server TTP Technology Specific Additional Information		AtmLinkEndAllAttr **getVendorProfile** **addVendorProfile** **removeVendorProfile**	
	Cell Scrambling Enable		AtmLinkEndAllAttr **getCellScramblingEnabled** **setCellScramblingEnabled**	
	Subscriber Address		AtmLinkEndAllAttr **getSubscriberAddressList** **addSubscriberAddress** **removeSubscriberAddress**	
	Prefered Carrier		AtmLinkEndAllAttr **getPreferredCarrierList** **addPreferredCarrier** **removePreferredCarrier**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmLinkEndAllAttr **getCharacteristicInfo** **getIlmiEstabConnectivityPollInterval** **setIlmiEstabConnectivityPollInterval** **getIlmiCheckConnectivityPollInterval** **setIlmiCheckConnectivityPollInterval** **getIlmiConnectivityPollFactor** **setIlmiConnectivityPollFactor**	
	Relationship With vpTopologicalLink		AtmLinkEndAllAttr **getSupportedLink**	
	Relationship With vpLogicalLinkTP		*NO CORBA RELATIONSHIP*	
	Relationship With vpSubnetwork		*NO CORBA RELATIONSHIP*	
	Relationship With serverTTPs		AtmLinkEndAllAttr **getServerTTP** **setServerTTP**	
	Relationship With vpNetworkAccessProfile		AtmLinkEndAllAttr **getNetworkAccessProfile** **setNetworkAccessProfile**	
	Relationship With vpNetworkCTP: Existing Connection Termination Points		AtmLinkEndAllAttr **getSupportedCTPs** **addSupportedCTP** **removeSupportedCTP**	
	query vpLinkEnd forTerminated vpTopologicalLink		AtmLinkEndAllAttr **getSupportedLink**	
	query vpLinkEnd for delineated vpSubnetwork		**NO CORBA OPERATION**	
	query vpLinkEnd For associated serverTTP		AtmLinkEndAllAttr **getServerTTP**	
	associate vpLinkEnd with supporting serverTTP		**setServerTTP**	
	vpLinkEnd PVC Trace		linkPVCTrace	
vpLogicalLinkTP	vpLogicalLinkTP ID	**NO M4 IDL OBJECT**		
	Egress Maximum Assignable Bandwidth			
	Ingress Maximum Assignable Bandwidth			
	Egress available Bandwidth			
	Ingress available Bandwidth			
	VPI Range			
	User Label			
	Relationship With vpTopologicalLink			
	Relationship With vpLinkEnd			
	Relationship With vpSubnetwork			
	Relationship With vpNetworkAccessProfile			
	Relationship With vpNetworkCTP: Existing Connection Termination Points			
	query vpLogicalLinkTP for terminated vpTopologicalLink			
	query vpLogicalLinkTP for delineated vpSubnetwork			
	query vpLogicalLinkTP for associated vpLinkEnds			
	associate vpLogicalLinkTP with supporting vpLinkEnd			
	vpLogicalLinkTP PVC Trace			

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/**Operation**	CORBA IDL Object	CORBA IDL Attribute/**Operation**	Comment
vpNetworkAccessProfile	vpNetworkAccessProfile ID	**AtmNetworkAccess Profile** **AtmNetworkAccess ProfileFactory**	AtmNetworkAccessProfileAllAttr	
	total Egress Bandwidth		AtmNetworkAccessProfileAllAttr **getTotalEgressBW** **setTotalEgressBW**	
	total Ingress Bandwidth		AtmNetworkAccessProfileAllAttr **getTotalIngressBW** **setTotalIngressBW**	
	maximum Number of Active Connection Allowed		AtmNetworkAccessProfileAllAttr **getMaxNumActiveVcConn** **setMaxNumActiveVcConn** **getMaxNumActiveVpConn** **setMaxNumActiveVpConn**	
	VPI/VCI Range		AtmNetworkAccessProfileAllAttr **getVpiRange** **setVpiRange**	
vpRoutingProfile	vpRoutingProfile ID	**AtmRoutingProfile**		
	connectionTypeSupported		**NO CORBA OPERATION**	
	routeDescriptionList		**NO CORBA OPERATION**	
	maxHops		**NO CORBA OPERATION**	
	Relationship With vpSubnetwork		**NO CORBA OPERATION**	
	setup vpRoutingProfile		**NO CORBA OPERATION**	
vpSubnetwork	Subnetwork ID	**AtmSubnetwork**	AtmSubnetworkAllAttr AtmSubnetworkTopo **getSubnetworkID**	
	Signal Identification		AtmSubnetworkAllAttr **getCharacteristicInfo**	
	user Label		AtmSubnetworkAllAttr **getUserLabel** **setUserLabel**	
	availability Status		AtmSubnetworkAllAttr **getAvailabilityStatus**	
	supported by Object List		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedByObjectList** **addSupportedByObjects** **removeSupportedByObjects** **replaceSupportedByObjectList**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmSubnetworkAllAttr **getSystemTitle**	
	Relationship with vpSubnetworkConnection		AtmSubnetworkAllAttr **getContainedSNCs**	
	Relationship with vpSubnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentSubnetworks** **addComponentSubnetwork** **removeComponentSubnetwork**	
	Relationship with vpTopologicalLink		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentLinks** **addComponentLinks** **removeComponentLinks**	
	Relationship with vpLogicalLinkTP		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedLinkTPs** **addSupportedLinkTP** **removeSupportedLinkTP**	
	Relationship with vpLinkEnd		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedLinkTPs** **addSupportedLinkTP** **removeSupportedLinkTP**	
	query vpSubnetwork for delimiting vpNetworkCTPs		AtmSubnetworkAllAttr AtmSubnetworkTopo	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	query vpSubnetwork for existing vpSubnetworkConnections		AtmSubnetworkAllAttr **getContainedSNCs**	
	query vpSubnetwork for component vpSubnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentSubnetworks** **addComponentSubnetwork** **removeComponentSubnetwork**	
	query vpSubnetwork for vpTopologicalLinks between its component subnetworks		AtmSubnetworkAllAttr AtmSubnetworkTopo **getComponentLinks** **addComponentLinks** **removeComponentLinks**	
	query vpSubnetwork For Connecting vpLinkEnds or vpLogicalLinkTPs		AtmSubnetworkAllAttr AtmSubnetworkTopo **getSupportedLinkTPs** **addSupportedLinkTP** **removeSupportedLinkTP**	
	setup vpSubnetworkConnection		**setupPtToPtSNCWithCTP** **setupPtToPtSNCWithLinkTP** **setupPtToMultiSNCWithCTP** **setupPtToMultiSNCWithLinkTP**	
	modify vpSubnetworkConenction		**NO CORBA OPERATION**	
	addTPs to SunetworkConnection		**addTpToMultiSNCWithCTP** **addTpToMultiSNCWithLinkTP** **removeTpFromMultiSNC**	
	release vpSubnetworkConnection		**releaseSNC**	
vpSubnetworkConnection	Directionality	**AtmSNC**	AtmSNCAllAttr	
	availability Status		**NO CORBA OPERATION**	
	Administrative Status		AtmSNCAllAttr **getAvailabilityStatus**	
	User Label		AtmSNCAllAttr **getAdminState** **setAdminState**	
	restorableIndicator		AtmSNCAllAttr **getUserLabel** **setUserLabel**	
	retainedResource		AtmSNCAllAttr **getRestorableIndicator** **setRestorableIndicator**	
	provisionType		AtmSNCAllAttr **getProvisionType** **setProvisionType**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmSNCAllAttr **getCharacteristicInfo** **getOwnershipName** **setOwnershipName** **getConnectionType**	
	Relationship with networkCTPs		AtmSNCAllAttr **getAtpInstance** **getZtpList**	
	Relationship with subnetworkConnections		AtmSNCAllAttr **getComponentSNCList**	
	Relationship with vpLinkConnections		AtmSNCAllAttr **getComponentLinkConnList**	
	Relationship with routingProfiles		AtmSNCAllAttr **getRoutingProfile** **setRoutingProfile**	
	query subnetworkConnection for terminating networkCTPs		AtmSNCAllAttr **getAtpInstance** **getZtpList**	
	query vpSubnetworkConnection for Component vpSubnetworkConnections		AtmSNCAllAttr **getComponentSNCList**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	vpSubnetworkConnection Connection Trace		traceSNC	
vpTopologicalLink	vpTopologicalLink ID	**AtmLink**	AtmLinkAllAttr	
	Signal Identification		AtmLinkAllAttr **getCharacteristicInfo**	
	Directionality		**NO CORBA OPERATION**	
	Operational State		AtmLinkAllAttr **getAvailabilityStatus**	
	provisioned Bandwidth		**NO CORBA OPERATION**	
	available Bandwidth		**NO CORBA OPERATION**	
	restorationMode		AtmLinkAllAttr **getRestorationMode setRestorationMode**	
	Customer Identification		AtmLinkAllAttr **getCustomerID setCustomerID**	
	Weight		AtmLinkAllAttr **getLinkWeight setLinkWeight**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmLinkAllAttr **getAdminState setAdminState**	
	Relationship With linkConnections		AtmLinkAllAttr **getContainedLinkConns**	
	Relationship with logicalLinkTP		AtmLinkAllAttr **getALinkEnd getZLinkEnd**	
	Relationship with linkEnd		AtmLinkAllAttr **getLinkedSubnetworks**	
	Relationship with subnetwork		AtmLinkAllAttr **getNetworkAccessProfile setNetworkAccessProfile**	
	Relationship with vpNetworkAccessProfile		AtmLinkAllAttr **getContainedLinkConns**	
	query vpTopologicalLink for contained vpLinkConnections		AtmLinkAllAttr **getALinkEnd getZLinkEnd**	
	query vpTopologicalLink for Terminating vpLinkEnds or vpLogicalLinkTPs		AtmLinkAllAttr **getLinkedSubnetworks**	
	query vpTopologicalLink for delineated vpSubnetworks		**setupLinkConnWithCTP setupLinkConnOnLink**	
	set up vpLinkConnection		**NO CORBA OPERATION**	
	modify vpLinkConnection		**releaseLinkConn**	
	release vpLinkConnection		**NO CORBA OPERATION**	
	vpTopologicalLink PVC Trace		**NO CORBA OPERATION**	
vpTrail	vpTrail ID	**NO M4 IDL OBJECT**		
	signal Identification			
	Directionality			
	User Label			
	Administrative State			
	availability State			
	restorable Indicator			
	retained Resource			
	Relationship with vpNetworkTTP			
	query vpTrail for terminatingTTPs			
	vpTrail Connection Trace			
	vpTrailrequest ID			
vpTrailRequest	Request Status	**NO M4 IDL OBJECT**		
	requestType			
	requestCommittedTime			
	Relationship With vpTrail			
vcNetworkCTP	vcCTP ID	**AtmNetworkCTP**	AtmNetworkCTPAllAttr	
	VPI/VCI Value		AtmNetworkCTPAllAttr **getNetworkCTPVpiVci**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	User Label		AtmNetworkCTPAllAttr **getUserLabel** **setUserLabel**	
	segment endpoint		AtmNetworkCTPAllAttr **getSegmentEndpoint** **setSegmentEndpoint**	
	Ingress Tagging Indicator		AtmNetworkCTPAllAttr **getIngressTaggingInd** **setIngressTaggingInd**	
	Egress Tagging Indicator		AtmNetworkCTPAllAttr **getEgressTaggingInd** **setEgressTaggingInd**	
	PM OAM Method		AtmNetworkCTPAllAttr **getPmOamMethod** **setPmOamMethod**	
	PM OAM Direction		AtmNetworkCTPAllAttr **getPmOamDirection** **setPmOamDirection**	
	PM OAM block size		AtmNetworkCTPAllAttr **getPmOamBlockSize** **setPmOamBlockSize**	
	PM OAM Forward Active		AtmNetworkCTPAllAttr **getPmOamForwardActive** **setPmOamForwardActive**	
	PM OAM Backward Active		AtmNetworkCTPAllAttr **getPmOamBackwardActive** **setPmOamBackwardActive**	
	NO LOGICAL MIB ATTRIBUTE MATCH		**getCharacteristicInfo** **getAlarmSeverityAssignmentProfile** **setAlarmSeverityAssignmentProfile** **getCurrentProblemList**	
	Relationship with vcNetworkTTP		AtmNetworkCTPAllAttr **getRelatedAtmTTP** **setRelatedAtmTTP**	
	Relationship with subnetworkConnection		AtmNetworkCTPAllAttr **getAssociatedSNCs**	
	Relationship with trafficDescriptorProfile		AtmNetworkCTPAllAttr **getEgressTrafficDescProfile** **getIngressTrafficDescProfile** **setTrafficDescProfile**	
	associate vcNetworkCTP with vcNetworkTTP		**NO CORBA OPERATION**	
	query vcNetworkCTP for associated vcNetworkTTP		AtmNetworkCTPAllAttr	
	query vcNetworkCTP for associated subnetworkConnections		AtmNetworkCTPAllAttr	
	lookback vcTrail at vcNetworkCTP		**loopbackOamCell**	
vcNetworkTTP	vcTTP ID	**AtmNetworkTTP**	AtmNetworkTTPAllAttr	
	availability Status		AtmNetworkTTPAllAttr **getAvailabilityStatus**	
	PM OAM Method		AtmNetworkTTPAllAttr **getPmOamMethod** **setPmOamMethod**	
	PM OAM Direction		AtmNetworkTTPAllAttr **getPmOamDirection** **setPmOamDirection**	
	PM OAM block size		AtmNetworkTTPAllAttr **getPmOamBlockSize** **setPmOamBlockSize**	
	PM OAM Forward Active		AtmNetworkTTPAllAttr **getPmOamForwardActive** **setPmOamForwardActive**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	PM OAM Backward Active		AtmNetworkTTPAllAttr **getPmOamBackwardActive** **setPmOamBackwardActive**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmNetworkTTPAllAttr **getCharacteristicInfo** **getNetworkCTPVpiVci** AlarmSeverityAssignmentProfile **setAlarmSeverityAssignmentProfile** **getCurrentProblemList**	
	Relationship with vcNetworkCTP		AtmNetworkTTPAllAttr	
	Relationship with vcTrail		AtmNetworkTTPAllAttr **getAssociatedTrail**	
	Relationship with AAL Profile		*NO CORBA RELATIONSHIP*	
	Relationship with Service Profile		*NO CORBA RELATIONSHIP*	
	query vcNetworkTTP for associated vcNetworkCTP		**getRelatedAtmCTP** **setRelatedAtmCTP**	
	query vcTTP For terminated vcTrail		**getAssociatedTrail**	
	loopback vpTrail at vpTTP		**loopbackOamCell**	
vpNetworkCTP	vpCTP ID	**AtmNetworkCTP**	AtmNetworkCTPAllAttr	
	VPI Value		AtmNetworkCTPAllAttr **getNetworkCTPVpiVci**	
	User Label		AtmNetworkCTPAllAttr **getUserLabel** **setUserLabel**	
	segment endpoint		AtmNetworkCTPAllAttr **getSegmentEndpoint** **setSegmentEndpoint**	
	Ingress Tagging Indicator		AtmNetworkCTPAllAttr **getIngressTaggingInd** **setIngressTaggingInd**	
	Egress Tagging Indicator		AtmNetworkCTPAllAttr **getEgressTaggingInd** **setEgressTaggingInd**	
	PM OAM Method		AtmNetworkCTPAllAttr **getPmOamMethod** **setPmOamMethod**	
	PM OAM Direction		AtmNetworkCTPAllAttr **getPmOamDirection** **setPmOamDirection**	
	PM OAM block size		AtmNetworkCTPAllAttr **getPmOamBlockSize** **setPmOamBlockSize**	
	PM OAM Forward Active		AtmNetworkCTPAllAttr **getPmOamForwardActive** **setPmOamForwardActive**	
	PM OAM Backward Active		AtmNetworkCTPAllAttr **getPmOamBackwardActive** **setPmOamBackwardActive**	
	NO LOGICAL MIB ATTRIBUTE MATCH		**getCharacteristicInfo** **getAlarmSeverityAssignmentProfile** **setAlarmSeverityAssignmentProfile** **getCurrentProblemList**	
	Relationship with vpNetworkTTP		AtmNetworkCTPAllAttr **getRelatedAtmTTP** **setRelatedAtmTTP**	
	Relationship with subnetworkConnection		AtmNetworkCTPAllAttr **getAssociatedSNCs**	
	Relationship with trafficDescriptorProfile		AtmNetworkCTPAllAttr **getEgressTrafficDescProfile** **getIngressTrafficDescProfile** **setTrafficDescProfile**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
	associate vpNetworkCTP with vpNetworkTTP		**NO CORBA OPERATION**	
	query vpNetworkCTP for associated vpNetworkTTP		AtmNetworkCTPAllAttr	
	query vpNetworkCTP for associated subnetworkConnections		AtmNetworkCTPAllAttr	
	lookback vpTrail at vpNetworkCTP		**loopbackOamCell**	
vpNetworkTTP	vpTTP ID	**AtmNetworkTTP**	AtmNetworkTTPAllAttr	
	availability Status		AtmNetworkTTPAllAttr **getAvailabilityStatus**	
	PM OAM Method		AtmNetworkTTPAllAttr **getPmOamMethod setPmOamMethod**	
	PM OAM Direction		AtmNetworkTTPAllAttr **getPmOamDirection setPmOamDirection**	
	PM OAM block size		AtmNetworkTTPAllAttr **getPmOamBlockSize setPmOamBlockSize**	
	PM OAM Forward Active		AtmNetworkTTPAllAttr **getPmOamForwardActive setPmOamForwardActive**	
	PM OAM Backward Active		AtmNetworkTTPAllAttr **getPmOamBackwardActive setPmOamBackwardActive**	
	NO LOGICAL MIB ATTRIBUTE MATCH		AtmNetworkTTPAllAttr **getCharacteristicInfo getNetworkCTPVpiVci** AlarmSeverityAssignmentProfile **setAlarmSeverityAssignmentProfile getCurrentProblemList**	
	Relationship with vpNetworkCTP		AtmNetworkTTPAllAttr	
	Relationship with vpTrail		AtmNetworkTTPAllAttr **getAssociatedTrail**	
	query vpNetworkTTP for associated vpNetworkCTP		**getRelatedAtmCTP setRelatedAtmCTP**	
	query vpTTP For terminated vpTrail		**getAssociatedTrail**	
	loopback vpTrail at vpTTP		**loopbackOamCell**	
aal1Profile		**NO M4 IDL OBJECT**		
aal3/4Profile		**NO M4 IDL OBJECT**		
aal5Profile		**NO M4 IDL OBJECT**		
alarmRecord				See Appendix C, Table C-1
alarmSeverityAssignmentProfile		**AlarmSeverityAssignmentProfile**	AlarmSeverityAssignmentSetType **getAlarmSeverityAssignmentList addAlarmSeverityAssignments removeAlarmSeverityAssignments setAlarmSeverityAssignmentList**	
atmCellProtocolMonitoringLogRecord		**NO M4 IDL OBJECT**		
cesServiceProfile		**NO M4 IDL OBJECT**		
eventForwardingDiscriminator		**NO M4 IDL OBJECT**		
latestOccurrenceLog		**NO M4 IDL OBJECT**		
log				See Appendix C, Table C-1
trafficDescriptor	Managed Entity ID	**AtmTrafficDesc**		AtmTrafficDesc is uninstantiable. See Note 2 for Table 3-1.
	Profile Name		AtmTrafficDescABRAllAttr AtmTrafficDescCBRAllAttr AtmTrafficDescVBRAllAttr AtmTrafficDescUBRAllAttr **getProfileName**	

M4 Logical MIB Managed Entity	**M4 Logical MIB** Attribute/**Operation**	**CORBA IDL Object**	**CORBA IDL** Attribute/**Operation**	**Comment**
	Service Category		AtmTrafficDescABRAllAttr AtmTrafficDescCBRAllAttr AtmTrafficDescVBRAllAttr AtmTrafficDescUBRAllAttr **getServiceCategory**	
	Conformance Definition		AtmTrafficDescABRAllAttr AtmTrafficDescCBRAllAttr AtmTrafficDescVBRAllAttr AtmTrafficDescUBRAllAttr **getConformanceDefinition**	
	Peak Cell Rate - Ingress and Egress	**AtmTrafficDescABR**	AtmTrafficDescABRAllAttr **getAllAttrABR** **getPeakCellRate**	See Note 2
		AtmTrafficDescCBR	AtmTrafficDescCBRAllAttr **getAllAttrCBR** **getPeakCellRate**	See Note 2
		AtmTrafficDescVBR	AtmTrafficDescVBRAllAttr **getAllAttrVBR** **getPeakCellRate**	See Note 2
		AtmTrafficDescUBR	AtmTrafficDescUBRAllAttr **getAllAttrUBR** **getPeakCellRate**	See Note 2
	Cell Delay Variation Tolerance in relation to the PCR - Ingress and Egress	**AtmTrafficDescABR**	AtmTrafficDescABRAllAttr **getAllAttrABR** **getCDVTolerancePCR**	
		AtmTrafficDescCBR	AtmTrafficDescCBRAllAttr **getAllAttrCBR** **getCDVTolerancePCR**	
		AtmTrafficDescVBR	AtmTrafficDescVBRAllAttr **getAllAttrVBR** **getCDVTolerancePCR**	
		AtmTrafficDescUBR	AtmTrafficDescUBRAllAttr **getAllAttrUBR** **getCDVTolerancePCR**	
		AtmTrafficDescVBR	AtmTrafficDescVBRAllAttr **getAllAttrVBR** **getCDVToleranceSCR**	CDVT-SCR for VBR if I.371 is supported
	Sustainable Cell Rate - Ingress and Egress	**AtmTrafficDescVBR**	AtmTrafficDescVBRAllAttr **getAllAttrVB** **getSustainableCellRate**	
	Maximum Burst Size - Ingress and Egress	**AtmTrafficDescVBR**	AtmTrafficDescVBRAllAttr **getAllAttrVBR** **getMaxBurstSize**	
	Minimum Cell Rate - Ingress and Egress	**AtmTrafficDescABR**	AtmTrafficDescABRAllAttr **getAllAttrABR** **getMinCellRate**	
	Initial Cell Rate - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getInitialCellRate**	
	Transient Buffer Exposure - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getTransientBufferExposure**	
	Rate Decrease Factor - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getRateDecreaseFactor**	
	Rate Increase Factor - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getRateIncreaseFactor**	
	Fixed Round Trip Time		AtmTrafficDescABRAllAttr **getAllAttrABR** **getFixedRoundTripTime**	
	Nrm - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getABRNrm**	
	Trm - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getABRTrm**	
	CDF - Ingress and Egress		AtmTrafficDescABRAllAttr **getAllAttrABR** **getABRCDF**	

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/**Operation**	CORBA IDL Object	CORBA IDL Attribute/**Operation**	Comment
	ADTF - Ingress and Egress		`AtmTrafficDescABRAllAttr` **`getAllAttrABR`** **`getABRADTF`**	
	CLR - Ingress and Egress	**AtmTrafficDescCBR**	`AtmTrafficDescCBRAllAttr` **`getAllAttrCBR`** **`getCLR`**	
		AtmTrafficDescVBR	`AtmTrafficDescVBRAllAttr` **`getAllAttrVBR`** **`getCLR`**	
NOTE: The following logical managed entities and CORBA IDL objects pertain to submodule ATMF_M4NW_PM.				
atmCellProtocolMonitoringCurrentData	Managed Entity ID	**CellProtocolMonCurrentData**	`CurrentDataID`	Attributes from structs CurrentDataAttributes and CurrentIntervalData defined in module CORBA_PM
	Administrative State		`AdministrativeState`	
	Suspect Flag		`SuspectFlag`	
	Elapsed Time		`ElapsedTime`	
	Threshold Data ID		`ThresholdDataID`	
	Number of Suppressed Intervals		`suppressionIndicator`	
	No M4 attribute; from Q.822.		`OperationalState`	
	No M4 attribute; from Q.822.		`GranularityPeriod`	
	Discarded Cells due to Protocol Errors		`NumberDiscCellsProtErr`	Determined by appropriate value of PerfParameter
	Received OAM Cells		`Number RecvOAMCells`	
	No actions have been defined.		**`setAdministrativeState`**	Methods inherited from CurrentData of module CORBA_PM
			`setHistoryRetention`	
			`setThresholdDataID`	
			`getCurrentDataAttributes`	
			`getCurrentIntervalData`	
		AtmPMBulkOperations	**`getCurrentPMBulkData`**	Method inherited from PMBulkOperations of module CORBA_PM
atmCellProtocolMonitoringHistoryData	Managed Entity ID	**CellProtocolMonHistoryData**	`HistoryDataID`	Attributes from structs HistoryDataAttributes and HistoryIntervalData defined in module CORBA_PM
	Period End Time		`PeriodEndTime`	
	Suspect Flag		`SuspectFlag`	
	Number of Suppressed Intervals		`NumIntervals`	
	No M4 attribute; from Q.822.		`Granularityperiod`	
	Discarded Cells due to Protocol Errors		`NumberDiscCellsProtErr`	Determined by appropriate value of PerfParameter
	Received OAM Cells		`NumberRecvOAMCells`	
	No actions have been defined.		**`getHistoryDataAttributes`**	Methods inherited from HistoryData of module CORBA_PM
			`getHistoryIntervalData`	
		AtmPMBulkOperations	**`getHistoryPMBulkData`**	Method inherited from PMBulkOperations of module CORBA_PM
atmTrafficLoadCurrentData	Managed Entity ID	**AtmTrafficLoadCurrentData**	`CurrentDataID`	Attributes from structs CurrentDataAttributes and CurrentIntervalData as defined in module CORBA_PM
	Administrative State		`AdministrativeState`	
	Suspect Flag		`SuspectFlag`	
	Elapsed Time		`ElapsedTime`	
	Threshold Data ID		`ThresholdDataID`	
	Number of Suppressed Intervals		`suppressionIndicator`	
	No M4 attribute; from Q.822.		`OperationalState`	
	No M4 attribute; from Q.822.		`GranularityPeriod`	
	Cells Received		`NumberCellsRecvd`	Determined by appropriate value of PerfParameter
	Cells Transmitted		`NumberCellsTrnsd`	
	No actions have been defined.		**`setAdministrativeState`**	Methods inherited from CurrentData of module CORBA_PM
			`setHistoryRetention`	
			`setThresholdDataID`	
			`getCurrentDataAttributes`	

M4 Logical MIB Managed Entity	**M4 Logical MIB** Attribute/**Operation**	**CORBA IDL Object**	**CORBA IDL** Attribute/**Operation**	**Comment**
			getCurrentIntervalData	
		AtmPMBulkOperations	**getCurrentPMBulkData**	Method inherited from PMBulkOperations of module CORBA_PM
atmTrafficLoadHistoryData	Managed Entity ID	**AtmTrafficLoadHistoryData**	HistoryDataID	Attributes from structs HistoryDataAttributes and HistoryIntervalData defined in module CORBA_PM
	Period End Time		PeriodEndTime	
	Suspect Flag		SuspectFlag	
	Number of Suppressed Intervals		NumIntervals	
	No M4 attribute; from Q.822.		GranularityPeriod	
	Cells Received		NumberCellsRecvd	Determined by appropriate value of PerfParameter
	Cells Transmitted		NumberCellsTrnsd	
	No actions have been defined.		**getHistoryDataAttributes**	Methods inherited from HistoryData of module CORBA_PM
			getHistoryIntervalData	
		AtmPMBulkOperations	**getHistoryPMBulkData**	Method inherited from PMBulkOpertaions of module CORBA_PM
congestionDiscardCurrentData	Managed Entity ID	**CongDiscardCurrentData**	CurrentDataID	Attributes from structs CurrentDataAttributes and CurrentIntervalData defined in module CORBA_PM
	Administrative State		AdministrativeState	
	Suspect Flag		SuspectFlag	
	Elapsed Time		ElapsedTime	
	Threshold Data ID		ThresholdDataID	
	Number of Supprressed Intervals		suppressionIndicator	
	No M4 attribute; from Q.822.		OperationalState	
	No M4 attribute; from Q.822.		GranularityPeriod	
	All Cells Discarded		TcountAllCellsDisc	Determined by appropriate value of PerfParameter
	Priority Cells Discarded		TcountPriorityCellsDisc	
	No actions have been defined.		**setAdministrativeState**	Methods inherited from CurrentData of module CORBA_PM
			setHistoryRetention	
			setThresholdDataID	
			getCurrentDataAttributes	
			getCurrentIntervalData	
		AtmPMBulkOperations	**getCurrentPMBulkData**	Method inherited from PMBulkOperations of module CORBA_PM
congestionDiscardHistoryData	Managed Entity ID	**CongDiscardHistoryData**	HistoryDataID	Attributes from structs HistoryDataAttributes and HistoryIntervalData defined in module CORBA_PM
	Period End Time		PeriodEndTime	
	Suspect Flag		SuspectFlag	
	Number of Suppressed Intervals		NumIntervals	
	No M4 attribute; from Q.822.		GranularityPeriod	
	All Cells Discarded		TcountAllCellsDisc	Determined by appropriate value of PerfParameter
	Priority Cells Discarded		TcountPriorityCellsDisc	
	No actions have been defined.		**getHistoryDataAttributes**	Methods inherited from HistoryData of module CORBA_PM
			getHistoryIntervalData	
		AtmPMBulkOperations	**getHistoryPMBulkData**	Method inherited from PMBulkOperations of module CORBA_PM
tcAdaptorProtocolMonitoringCurrentData	Managed Entity ID	**TcAdaptProtMonCurrentData**	CurrentDataID	Attributes from structs CurrentDataAttributes and
	Administrative State		AdministrativeState	
	Suspect Flag		SuspectFlag	
	Elapsed Time		ElapsedTime	

M4 Logical MIB Managed Entity	**M4 Logical MIB** Attribute/**Operation**	**CORBA IDL Object**	**CORBA IDL** Attribute/**Operation**	**Comment**
	Threshold Data ID		`ThresholdDataID`	CurrentIntervalData defined in module CORBA_PM
	Number of Suppressed Intervals		`suppressionIndicator`	
	No M4 attribute; from Q.822.		`OperationalState`	
	No M4 attribute; from Q.822.		`GranularityPeriod`	
	Discarded Cells due to HEC violation		`NumberDiscCellsHECViolat`	Determined by appropriate value of PerfParameter
	No actions have been defined.		**`setAdministrativeState`**	Methods inherited from CurrentData of module CORBA_PM
			`setGranularityPeriod`	
			`setThresholdDataID`	
			`getCurrentDataAttributes`	
			`getCurrentIntervalData`	
		AtmPMBulkOperations	**`getCurrentPMBulkData`**	Method inherited from PMBulkOperations of module CORBA_PM
tcAdaptorProtocolMonitoringHistoryData	Managed Entity ID	**TcAdaptProtMonHistoryData**	`CurrentDataID`	Attributes from structs HistoryDataAttributes and HistoryIntervalData defined in module CORBA_PM
	Period End Time		`PeriodEndTime`	
	Suspect Flag		`SuspectFlag`	
	Number of Suppressed Intervals		`NumIntervals`	
	No M4 attribute; from Q.822.		`GranularityPeriod`	
	Discarded Cells due to HEC viiolation		`NumberDiscCellsHECViolat`	Determined by appropriate value of PerfParameter
	No actions have been defined.		**`getHistoryDataAttributes`**	Methods inherited from HistoryData of module CORBA_PM
			`getHistoryIntervalData`	
		AtmPMBulkOperations	**`getHistoryPMBulkData`**	Method inherited from PMBulkOperations of module CORBA_PM
thresholdData	Managed Entity ID	**AtmThresholdData**	`ThresholdDataID`	Attributes from structs ThresholdDataID and PerfThreshold defined in module CORBA_PM
	Performance Parameter and Threshold Value		`PerfParameter`	
			`ThresholdValue`	
	No actions have been defined.		**`getThresholdData`** **`setThresholdData`**	Methods inherited from ThresholdData of module CORBA_PM
		AtmPMBulkOperations	**`getAllThresholdDataIDs`** **`getThresholdBulkData`** **`setThresholdBulkData`**	Methods inherited from PMBulkOperations of module CORBA_PM
No Logical MIB counterpart.		**AtmCurrentDataFactory**		IDL-specific object.

NOTES for Table 3-1:
1. All attributes are shown in a non-emphasized type font, and `CORBA IDL attributes are shown in Section 5 of this document.` **Operations are shown in bold type. Logical MIBmanaged Entities and CORBA IDL Objects are shown in bold type.** *Relationships are shown in italics.*

2. The trafficDescriptor logical MIB managed entity covers the full range of Service Categories as defined in the ATM Forum's TM Specification 4.0. In this document's IDL, one interface is used for each Service Category. This approach is taken because CORBA IDL does not permit a convenient method of describing conditionality. Each of these specialized traffic descriptor interfaces inherits from an uninstantiable parent interface named AtmTrafficDesc that contains common methods related to objectID, profileName, serviceCategory and conformanceDefinition.

4 Containment and Inheritance Diagrams

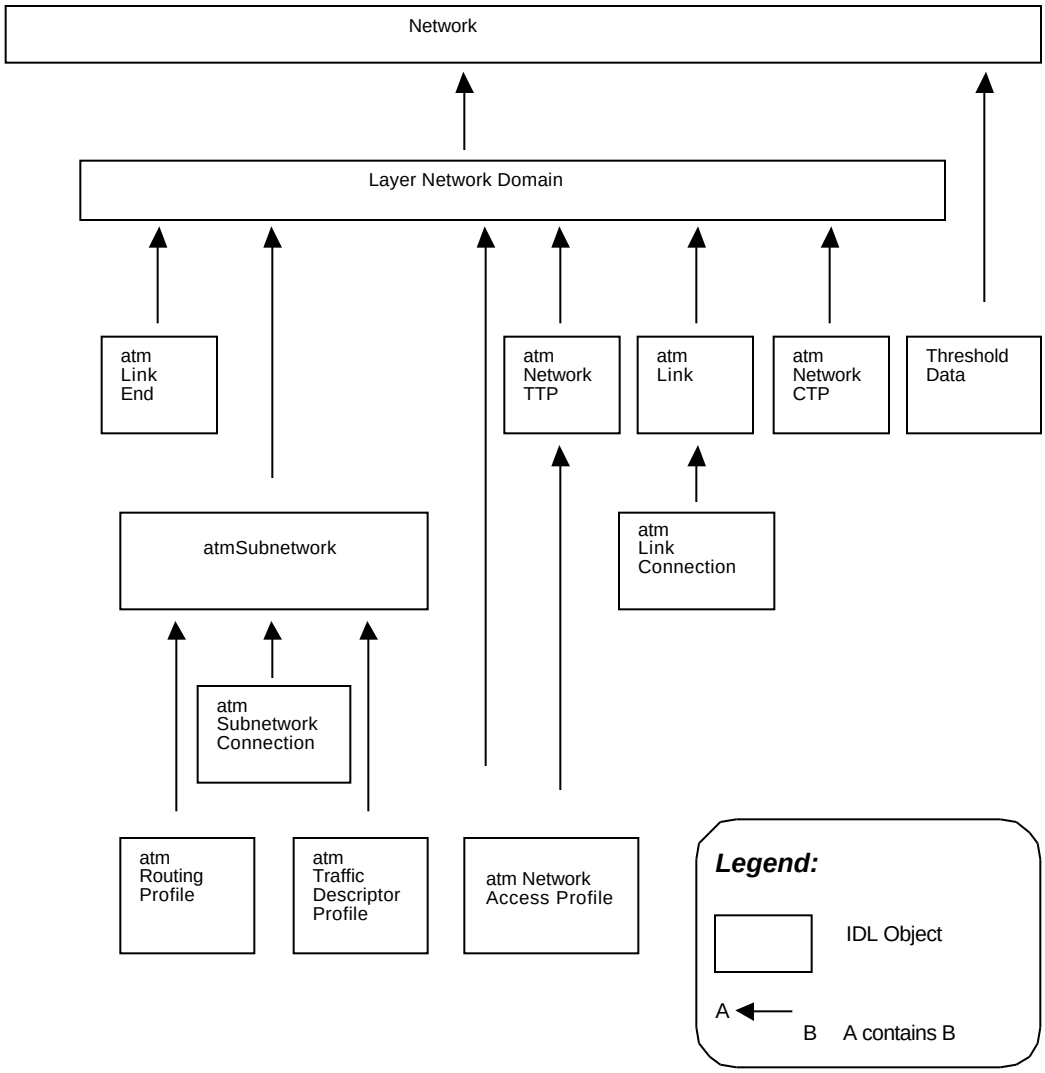

Figure 4-1. Containment Diagram

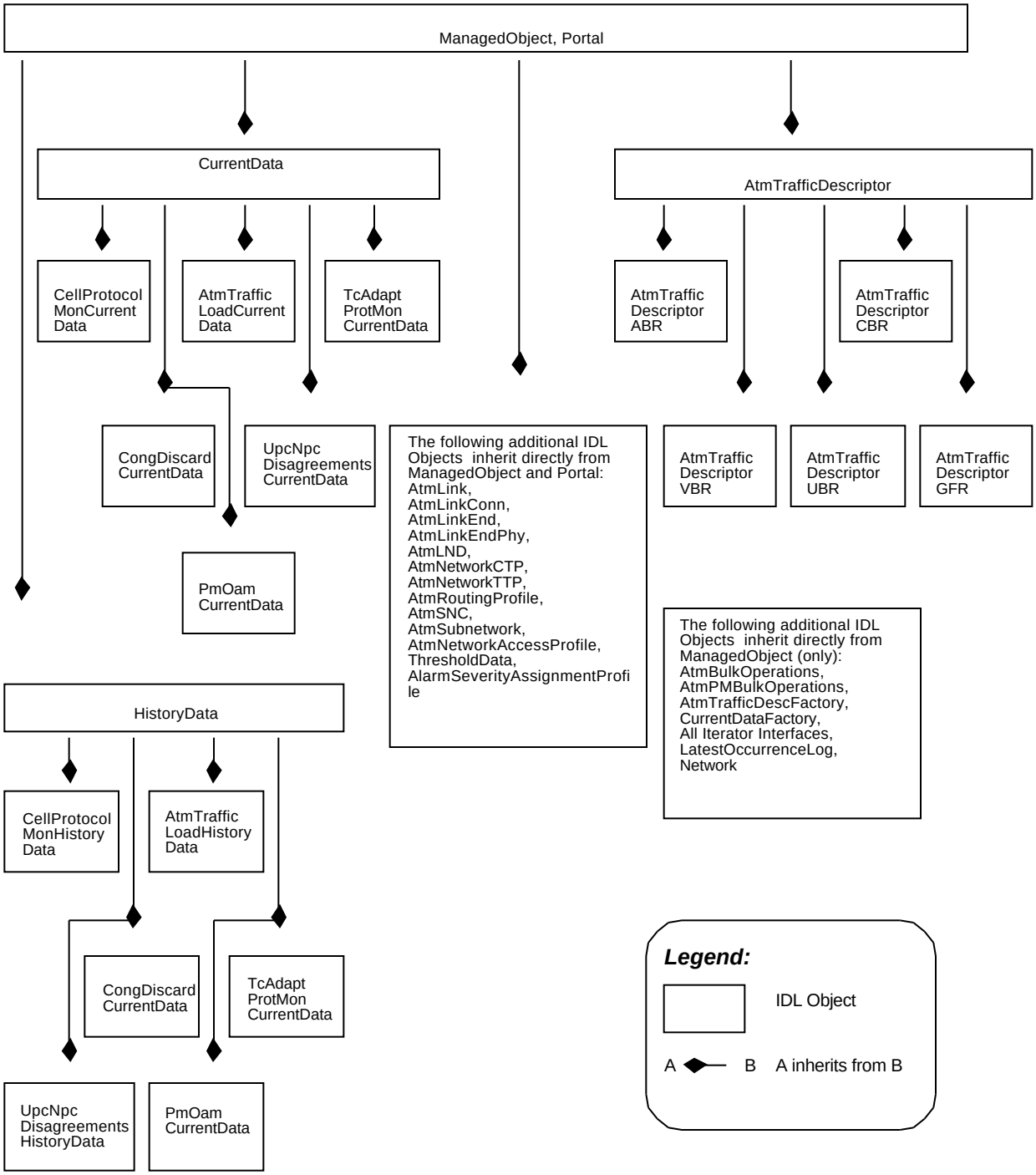

Figure 4-2. Inheritance Diagram

```
/**
```

5 CORBA IDL Definitions

```
*/

#ifndef _atmf_m4nw_idl_
#define _atmf_m4nw_idl_

#include "NetMgmt.idl"

/**
This IDL code is intended to be stored in a file named "atmf_m4nw.idl" located in the search
path of your IDL compiler.
```

5.1 Module ATMF_M4NW

```
This IDL provides a set of IDL interfaces for managing an ATM network using the ATM Forum M4
Network View requirements and logical MIB found in AF-NM-0058.001.

All performance management aspects are grouped under a separate sub-module, atmf_m4nw_pm (see
Section 5.5), to facilitate extensions of this IDL.
*/

module atmf_m4nw
{
const string moduleName = "atmf_m4nw";

#ifndef _atm_probable_cause_const_idl_
#define _atm_probable_cause_const_idl_

module atm_probable_cause_const
{
       const string moduleName = "atm_probable_cause_const";
       const short lCD = 1;  // Loss of Cell Delineation
       const short pLCPLOF = 2;     // PLCP Loss of Frame for DS3
       const short pLCPFE = 3;      // PLCP Far End Alarm for DS3

/**
This sub-module contains the constant values defined for the ATM Specific ProbableCause UIDs
*/

}; // end of module atm_probable_cause_const
#endif // _atm_probable_cause_const_idl_

/**
```

5.2 Imports and Forward Declarations

```
IMPORTS

Types imported from NetMgmt
*/
       typedef NetMgmt::AdministrativeState       AdministrativeState;
       typedef NetMgmt::MOID              MOID;
       typedef NetMgmt::MOIDList                  MOIDList;
       typedef NetMgmt::Name              NameType;
       typedef NetMgmt::OperationalState          OperationalState;
       typedef NetMgmt::UID               UID;
       typedef NetMgmt::ProbableCause             ProbableCauseType;
       typedef NetMgmt::GeneralizedTime           GeneralizedTime;

/**
Exceptions imported from NetMgmt are DuplicateItem, DuplicateName, ItemNotFound,
NotSupported, ObjectFailure, OutOfRange, and InvalidID.
```

```
Interfaces imported from NetMgmt are ManagedObject, ManagedObjectFactory, Portal, and
NameIterator.


FORWARD DECLARATIONS
*/
       interface AlarmSeverityAssignmentProfile;
       interface AlarmSeverityAssignmentProfileFactory;
       interface AtmBulkOperations;
       interface AtmLink;
       interface AtmLinkConn;
       interface AtmLinkEnd;
       interface AtmLinkEndPhy;
       interface AtmLND;
       interface AtmNetworkAccessProfile;
       interface AtmNetworkAccessProfileFactory;
       interface AtmNetworkCTP;
       interface AtmNetworkTTP;
       interface AtmRoutingProfile;
       interface AtmSubnetwork;
       interface AtmSNC;
       interface AtmTrafficDesc;
       interface AtmTrafficDescABR;
       interface AtmTrafficDescCBR;
       interface AtmTrafficDescGFR;
       interface AtmTrafficDescUBR;
       interface AtmTrafficDescVBR;
       interface AtmTrafficDescFactory;
       interface Network;
       interface LatestOccurrenceLog;
       interface SNCAllIterator;
       interface TTPIDIterator;
       interface TTPAllIterator;
       interface LinkIDIterator;
       interface LinkAllIterator;
       interface LinkEndIDIterator;
       interface LinkEndAllIterator;
       interface NetworkCTPAllIterator;
       interface NetworkAccessProfileIDIterator;
       interface NetworkAccessProfileAllIterator;
       interface TrafficDescIDIterator;
       interface TrafficDescAllIterator;
       interface LinkEndPhyIDIterator;
       interface LinkEndPhyAllIterator;
       interface RoutingProfileIDIterator;
       interface RoutingProfileAllIterator;
```

```
/**
```

5.3 Structures and Typedefs

```
*/
       typedef UID ProblemCause;
       typedef UID CharacteristicInfo;

/**
ABR Nrm
*/
       enum ABRNrm
       {
              noNrm,         // no Nrm
              nrm2,          //  2
              nrm4 ,         // 4
              nrm8,          // 8
              nrm16,         // 16
              nrm32,         // 32
              nrm64,         // 64
              nrm128,// 128
              nrm256 // 256
       };

/**
ABR Trm
*/
       enum ABRTrm
       {
              noTrm, // no Trm
              trm1,  // 100 ms
              trm2,  // 100 * 2^(-1) = 50 ms
              trm3,  // 100 * 2^(-2) = 25 ms
              trm4,  // 100 * 2^(-3) = 12.5 ms
              trm5,  // 100 * 2^(-4) = 6.25 ms
              trm6,  // 100 * 2^(-5) = 3.125 ms
              trm7,  // 100 * 2^(-6) = 1.5626 ms
              trm8   // 100 * 2^(-7) = 0.78125ms
       };

/**
ABR CDF
*/
       enum ABRCDF
       {
              noCDF,         // no CDF
              cDF0,          // 0
              cDF1over64,    // 1/64
              cDF1over32,    // 1/32
              cDF1over16,    // 1/16
              cDF1over8,     // 1/8
              cDF1over4,     // 1/4
              cDF1over2,     // 1/2
              cDF1           // 1
       };

/**
GFR1 or GFR2
*/
       enum GFR1or2
       {
              GFR1,
              GFR2
       };
```

```
/**
Alarm Severity Code.
*/
       enum AlarmSeverityCodeType
       {
              alarmSeverityCodeNonalarmed,
              alarmSeverityCodeMinor,
              alarmSeverityCodeMajor,
              alarmSeverityCodeCritical,
              alarmSeverityCodeWarning
       };

       union AlarmSeverityCodeTypeOpt switch (boolean)
       {
              case TRUE:
                     AlarmSeverityCodeType val;
       };

/**
Alarm Severity Assignment.  Each alarm severity assignment structure
identifies a particular problem (with a Unique ID) and then provides the
alarm severity code assigned if that problem is service affecting, not service
affecting, or service independent.  This structure is usually part of
an AlarmSeverityAssignmentList.
*/
       struct AlarmSeverityAssignmentType
       {
              ProbableCauseType    problem;
              AlarmSeverityCodeTypeOpt severityAssignedServiceAffecting;
              AlarmSeverityCodeTypeOpt severityAssignedNonServiceAffecting;
              AlarmSeverityCodeTypeOpt severityAssignedServiceIndependent;
       };

/**
Alarm Severity Assignment Lists provide a listing of all abnormal
conditions that may exist in instances of an object class, and show the
assigned alarm severity information (minor, major etc.) for each condition.
*/
       typedef sequence<AlarmSeverityAssignmentType>
              AlarmSeverityAssignmentSetType;


/**
Managed objects supporting the AlarmSeverityAssignmentProfile
interface specify the alarm severity assignment for other managed
objects. Instances of this interface are referenced by the
alarmSeverityAssignmentProfilePointer attribute in the managed objects.
*/
       struct AlarmSeverityAssignmentProfileAllAttr
       {
              AlarmSeverityAssignmentSetType
                     alarmSeverityAssignmentList;
                     // alarmSeverityAssignmentProfilePackage
                     // GET-REPLACE, ADD-REMOVE
       }; // struct AlarmSeverityAssignmentProfileAllAttr

/** Alarm Status indicates the occurrence of an abnormal condition relating to
an object. Attributes of this type may also function as a summary indicator
of alarm conditions associated with a specific resource. It is used to
indicate the existence of an alarm condition, a pending alarm condition such
as threshold situations, or (when used as a summary indicator) the highest
severity of active alarm conditions.  When used as a summary indicator, the
order of severity (from highest to lowest) is: activeReportable-Critical
activeReportable-Major activeReportable-Minor activeReportable-Indeterminate
activeReportable-Warning activePending cleared.
*/
```

```
        enum AlarmStatus
        {cleared, activeReportableIndeterminate,
                activeReportableWarning, activeReportableMinor,
                activeReportableMajor, activeReportableCritical, activePending};


/** Avalibility Type is used in a sequence to indicate the availability
of a resource.  Zero or more of these conditions may be indicated.
*/
        enum AvailabilityType
                {inTest, failed, powerOff, offLine, offDuty, dependency,
                degraded, notInstalled, logFull};

/** Availability status is used to indicate the availability of a resource.
It is represented as a sequence of enums because several of the enumerated
conditions may exist at once.
*/
        typedef sequence<AvailabilityType> AvailabilityStatus;


/** The current problem structure identifies an existing problem with an
object.  It is typically a component of a Current Problem List.
*/
        struct CurrentProblem
        {
                ProbableCauseType     problem;
                AlarmStatus           alarmStatus;
        };

/** Current Problem Lists identify the current existing problems, with
severity, associated with a managed object.
*/
        typedef sequence<CurrentProblem> CurrentProblemList;

        struct LatestOccurrenceLogID
        {
                NameType              name;
                LatestOccurrenceLog   ref;
        };

        struct AtmLinkID
        {
                NameType      name;
                AtmLinkref;
        };

        typedef sequence<AtmLinkID> AtmLinkIDList;

        struct AtmLinkConnID
        {
                NameType      name;
                AtmLinkConn   ref;
        };

        typedef sequence<AtmLinkConnID> AtmLinkConnIDList;

        struct AtmLinkEndID
        {
                NameType      name;
                AtmLinkEnd    ref;
        };

        typedef sequence<AtmLinkEndID> AtmLinkEndIDList;

        struct AtmLinkEndPhyID
        {
                NameType              name;
                AtmLinkEndPhy         ref;
```

```
	};

	typedef sequence<AtmLinkEndPhyID> AtmLinkEndPhyIDList;




/**
Link ID or Link TP (Link End) ID
*/
	union LinkOrLinkTP switch (boolean)
	{
		case TRUE:
			AtmLinkID	atmLinkID;
		default:
			AtmLinkEndID	atmLinkEndID;
	};

	struct AtmLNDID
	{
		NameType	name;
		AtmLND		ref;
	};

	typedef sequence<AtmLNDID> AtmLNDIDList;

	struct AtmNetworkAccessProfileID
	{
		NameType			name;
		AtmNetworkAccessProfile		ref;
	};

	typedef sequence<AtmNetworkAccessProfileID>
		AtmNetworkAccessProfileIDList;

	struct AtmNetworkAccessProfileFactoryID
	{
		NameType				name;
		AtmNetworkAccessProfileFactory		ref;
	};

	struct AtmNetworkCTPID
	{
		NameType		name;
		AtmNetworkCTP		ref;
	};

	typedef sequence<AtmNetworkCTPID> AtmNetworkCTPIDList;

	struct AtmNetworkTTPID
	{
		NameType		name;
		AtmNetworkTTP		ref;
	};

	typedef sequence<AtmNetworkTTPID> AtmNetworkTTPIDList;

	struct AtmRoutingProfileID
	{
		NameType		name;
		AtmRoutingProfile	ref;
	};

	typedef sequence<AtmRoutingProfileID> AtmRoutingProfileIDList;

	struct AtmSNCID
```

```
        {
                NameType        name;
                AtmSNC          ref;
        };

        typedef sequence<AtmSNCID> AtmSNCIDList;

        struct AtmSubnetworkID
        {
                NameType                name;
                AtmSubnetwork           ref;
        };

        typedef sequence<AtmSubnetworkID> AtmSubnetworkIDList;

        struct AtmTrafficDescID
        {
                NameType                name;
                AtmTrafficDescref;
        };

        typedef sequence<AtmTrafficDescID> AtmTrafficDescIDList;

        struct AtmTrafficDescFactoryID
        {
                NameType                        name;
                AtmTrafficDescFactory ref;
        };

/**
TM 4.1 Conformance Definition
*/
        enum ConformanceDefinition
        {
                other,
                cBR1,
                vBR1,
                vBR2,
                vBR3,
                uBR1,
                uBR2,
                aBR,
                gFR
        };

/**
A connection type may be broadcast (point-to-multipoint),
merge (multipoint-to-point), composite (root-to-leaves & leaves-to-root),
multipoint (multipoint-to-multipoint), or pointToPoint (point-to-point)
*/
        enum ConnectionType
        {
                broadcast,
                merge,
                composite,
                multipoint,
                pointToPoint
        };

/**
Virtual ID - VPI value and VCI value
*/
        struct VirtualID
        {
                unsigned long  vpi;
                unsigned long  vci;
        };
```

```
/**
Element resulting from a connection trace
*/
       struct ConnTraceElement
       {
              LinkOrLinkTP   linkOrLinkTP;
              VirtualID      virtualID;
       };

/**
List of connection trace results
*/
       typedef sequence<ConnTraceElement> ConnTraceList;

/**
ILMI connectivity state
*/
       enum IlmiConnectivityState
       {
              unknown,
              connected,
              notConnected
       };

/**
Link End type
*/
       enum LinkEndType
       {
              uni,
              intraNNI,
              interNNI,
              unconfigured
       };

/**
Type of Link trace request
*/
       enum LinkTraceType
       {
              allSubnets,
              allInVPLnd,
              allInVCLnd,
              selectedSubnets
       };

/**
Results of Link trace request for a subnetwork
*/
       struct LinkTraceSubnetConn
       {
              AtmSubnetworkID       subnetwork;
              AtmSNCIDList          atmSubnetConns;
       };

/**
Results of loopback request
*/
       struct LoopbackCellReply
       {
              boolean        loopbackSuccessful;
              ProblemCause          problemCause;
       };

/**
Loopback location code
*/
```

```
	typedef sequence<octet>	LoopbackLocationCode;

/**
Loopback Location
*/
	struct LoopbackLoc
	{
		boolean			endPoint;
		LoopbackLocationCode	loopbackLocationCode;
	};

/**
An optional admin state may be locked, unlocked, or not specified
*/
	enum OptAdministrativeState
	{
		locked,
		unlocked,
		noAdminState
	};

/**
Optional Boolean
*/
	enum OptBoolean
	{
		false,
		true,
		notSpecified
	};

/**
An optional restorable type may be restorable, nonrestorable, or not specified
*/
	enum OptRestorableType
	{
		restorable,
		nonrestorable,
		notSpecifiedResType
	};

/**
Provision type may be manual or automatic
*/
	enum ProvisionType
	{
		manual,
		automatic
	};

/**
PM OAM block size
*/
	enum PmOamBlockSize
	{
		bs128,
		bs256,
		bs512,
		bs1024
	};

/**
PM OAM cell type for loopback
*/
	enum PmOamCellType
	{
		segment,
		endToEnd
```

```
	};

/**
PM OAM direction
*/
	enum PmOamDirection
	{
		receive,
		transmit,
		both
	};

/**
PM OAM method
*/
	enum PmOamMethod
	{
		tMN,
		oAM,
		pmOamNotSupported
	};

/**
Port ID, managed element and port required, others optional
*/
	struct PortID
	{
		string managedElement;
		string bay;
		string shelf;
		string drawer;
		string slot;
		string port;
	};

/**
ABR Rate Change Factor
*/
	enum RateChangeFactor
	{
		rCF1over32768,// 1/32768
		rCF1over16384,// 1/16384
		rCF1over8192,         // 1/8192
		rCF1over4096,         // 1/4096
		rCF1over2048,         // 1/2048
		rCF1over1024,         // 1/1024
		rCF1over512,          // 1/512
		rCF1over256,          // 1/256
		rCF1over128,          // 1/128
		rCF1over64,           // 1/64
		rCF1over32,           // 1/32
		rCF1over16,           // 1/16
		rCF1over8,            // 1/8
		rCF1over4,            // 1/4
		rCF1over2,            // 1/2
		rCF1                  // 1
	};

/**
restoration mode
*/
	enum RestorationMode
	{
		unavailable,
		availRoutingOnly,
		availReRoutingOnly,
		availRoutingAndReRouting
	};
```

```
/**
Traffic Service Category
*/
       enum ServiceCategory
       {
              otherSc,
              cBRSc,
              rtVBRSc,
              nrtVBRSc,
              aBRSc,
              uBRSc,
              gFRSc
       };

/**
List of Link trace results
*/
       typedef sequence<LinkTraceSubnetConn> SNCsBySubnetList;

/**
List of strings
*/
       typedef sequence<string> StringList;

/**
VPI or VCI range
*/
       struct VpiOrVciRange
       {
              long   lowVID;
              long   highVID;
       };

/**
Description of Z-end TP (CTP) for multipoint request
*/
       struct ZtpCompositeCtp
       {
              AtmNetworkCTPID       zTp;
              boolean       zTpTrailEndPointInd;
              AtmTrafficDescID      zEgressTrafficDescProfile;
       };

/**
List of Z-end TP (CTP) descriptions for multipoint request
*/
       typedef sequence<ZtpCompositeCtp> ZtpCompositeCtpList;

/**
Description of Z-end TP (LinkEnd) for multipoint request
*/
       struct ZtpCompositeLinkEnd
       {
              AtmLinkEndID          zTp;
              VirtualID             zTpVirtualID;
              boolean       zTpTrailEndPointInd;
              AtmTrafficDescID      zEgressTrafficDescProfile;
       };

/**
List of Z-end TP (LinkEnd) descriptions for multipoint request
*/
       typedef sequence<ZtpCompositeLinkEnd> ZtpCompositeLinkEndList;

/**
Latest Occurrence Log Definitions
*/
```

```
        enum CellHeaderAbnormalityType
        {
                unassignedVpiVciValue,
                outOfRangeVpiVciValue
        };

/**
Latest Occurrence Log Entry
*/
        struct LatestOccurrenceLogEntry
        {
                AtmLinkEndID linkEndId;
                VirtualID virtualID;
                CellHeaderAbnormalityType abnormalityType;
                GeneralizedTime timeStamp;
        };

/**
List of Latest Occurrence Log Entries
*/
        typedef sequence<LatestOccurrenceLogEntry> LatestOccLogList;


/**
All attributes of ATM Link
*/
        struct AtmLinkAllAttr
        {
                AtmLinkID linkID;
                CharacteristicInfo linkCharacteristicInfo;
                AvailabilityStatus linkAvailabilityStatus;
                AdministrativeState linkAdminState;
                string linkCustomerID;
                AtmLinkEndID linkALinkEnd;
                AtmLinkEndID linkZLinkEnd;
                AtmNetworkAccessProfileID linkNetworkAccessProfile;
                RestorationMode linkRestorationMode;
                long linkWeight;
                AtmLinkConnIDList linkContainedLinkConns;
                AtmSubnetworkIDList linkLinkedSubnetworks;
        };

        typedef sequence<AtmLinkAllAttr> AtmLinkAllAttrList;

/**
All attributes of ATM Link End
*/
        struct AtmLinkEndAllAttr
        {
                AtmLinkEndID leID;
                CharacteristicInfo leCharacteristicInfo;
                AvailabilityStatus leAvailabilityStatus;
                AdministrativeState leAdminState;
                string leUserLabel;
                AtmNetworkAccessProfileID leNetworkAccessProfile;
                LinkEndType leType;
                long leIngressMaxAssignBW;
                long leEgressMaxAssignBW;
                long leIngressAvailableBW;
                long leEgressAvailableBW;
                AtmLinkID leSupportedLink;
                MOID leServerTTP;
                AtmNetworkCTPIDList leSupportedCTPs;
                LoopbackLocationCode leLoopbackLocID;
                VirtualID leIlmiVpiVci;
                long leIlmiEstabConnectivityPollInterval;
                long leIlmiCheckConnectivityPollInterval;
```

```
                long leIlmiConnectivityPollFactor;
                string leSupportingNeLoc;
                PortID leSupportingPortID;
                CharacteristicInfo leServerTTPCharInfo;
                PortID leServerTTPPortID;
                OperationalState leServerTTPOpState;
                boolean leCellScramblingEnabled;
                StringList leSubscriberAddressList;
                StringList lePreferredCarrierList;
                MOID leVendorProfile;
        };

        typedef sequence<AtmLinkEndAllAttr> AtmLinkEndAllAttrList;


/**
All Attributes of ATM Link End Physical
*/
        struct AtmLinkEndPhyAllAttr
        {
                AtmLinkEndPhyID atmLinkEndPhyID;
                CharacteristicInfo leCharacteristicInfo;
                AvailabilityStatus leAvailabilityStatus;
                AdministrativeState leAdminState;
                string leUserLabel;
                AtmNetworkAccessProfileID leNetworkAccessProfile;
                LinkEndType leType;
                long leIngressMaxAssignBW;
                long leEgressMaxAssignBW;
                long leIngressAvailableBW;
                long leEgressAvailableBW;
                AtmLinkID leSupportedLink;
                MOID leServerTTP;
                AtmNetworkCTPIDList leSupportedCTPs;
                LoopbackLocationCode leLoopbackLocID;
                VirtualID leIlmiVpiVci;
                long leIlmiEstabConnectivityPollInterval;
                long leIlmiCheckConnectivityPollInterval;
                long leIlmiConnectivityPollFactor;
                string leSupportingNeLoc;
                PortID leSupportingPortID;
                CharacteristicInfo leServerTTPCharInfo;
                PortID leServerTTPPortID;
                OperationalState leServerTTPOpState;
                boolean leCellScramblingEnabled;
                StringList leSubscriberAddressList;
                StringList lePreferredCarrierList;
                MOID leVendorProfile;
        };

        typedef sequence<AtmLinkEndPhyAllAttr> AtmLinkEndPhyAllAttrList;

/**
All attributes of ATM Layer Network Domain
*/
        struct AtmLNDAllAttr
        {
                AtmLNDID lndID;
                string lndSystemTitle;
                CharacteristicInfo lndCharacteristicInfo;
                string lndUserLabel;
        };

/**
All attributes of ATM Network Access Profile
*/
        struct AtmNetworkAccessProfileAllAttr
        {
                AtmNetworkAccessProfileID napID;
```

```
                long napTotalIngressBW;
                long napTotalEgressBW;
                long napMaxNumActiveVcConn;
                long napMaxNumActiveVpConn;
                VpiOrVciRange napVpiRange;
                VpiOrVciRange napVciRange;
        };

        typedef sequence<AtmNetworkAccessProfileAllAttr>
                AtmNetworkAccessProfileAllAttrList;

/**
All attributes of ATM Network CTP
*/
        struct AtmNetworkCTPAllAttr
        {
                AtmNetworkCTPID nctpID;
                CharacteristicInfo nctpCharacteristicInfo;
                string nctpUserLabel;
                VirtualID nctpNetworkCTPVpiVci;
                boolean nctpSegmentEndpoint;
                AtmTrafficDescID nctpEgressTrafficDescProfile;
                AtmTrafficDescID nctpIngressTrafficDescProfile;
                AtmNetworkTTPID nctpRelatedAtmTTP;
                AtmSNCIDList nctpAssociatedSNCs;
                AlarmSeverityAssignmentProfile nctpAlarmProfile;
                CurrentProblemList nctpCurrentProblemList;
                OptBoolean nctpIngressTaggingInd;
                OptBoolean nctpEgressTaggingInd;
                PmOamMethod nctpPmOamMethod;
                PmOamDirection nctpPmOamDirection;
                PmOamBlockSize nctpPmOamBlockSize;
                OptBoolean nctpPmOamForwardActive;
                OptBoolean nctpPmOamBackwardActive;
        };

        typedef sequence<AtmNetworkCTPAllAttr> AtmNetworkCTPAllAttrList;

/**
All attributes of ATM Network TTP
*/
        struct AtmNetworkTTPAllAttr
        {
                AtmNetworkCTPID nttpID;
                CharacteristicInfo nttpCharacteristicInfo;
                VirtualID nttpNetworkCTPVpiVci;
                AvailabilityStatus nttpAvailabilityStatus;
                AtmNetworkCTPID nttpRelatedAtmCTP;
                MOID nttpAssociatedTrail;
                AlarmSeverityAssignmentProfile nttpAlarmProfile;
                CurrentProblemList nttpCurrentProblemList;
                PmOamMethod nttpPmOamMethod;
                PmOamDirection nttpPmOamDirection;
                PmOamBlockSize nttpPmOamBlockSize;
                OptBoolean nttpPmOamForwardActive;
                OptBoolean nttpPmOamBackwardActive;
        };

        typedef sequence<AtmNetworkTTPAllAttr> AtmNetworkTTPAllAttrList;

/**
All Attributes of ATM Routing Profile
*/
        struct AtmRoutingProfileAllAttr
        {
                AtmRoutingProfileID rpID;
                ConnectionType connectionType;
                string routeDescriptionList;
```

```
                unsigned short maxHops;
        };

        typedef sequence<AtmRoutingProfileAllAttr> AtmRoutingProfileAllAttrList;


/**
All attributes of ATM Subnetwork Connection
*/
        struct AtmSNCAllAttr
        {
                AtmSNCID sncID;
                CharacteristicInfo sncCharacteristicInfo;
                AvailabilityStatus sncAvailabilityStatus;
                AdministrativeState sncAdminState;
                string sncUserLabel;
                string sncOwnershipName;
                AtmNetworkCTPID sncAtpInstance;
                AtmNetworkCTPIDList sncZtpList;
                ConnectionType sncConnectionType;
                OptBoolean sncRestorableIndicator;
                AtmSNCIDList sncComponentSNCList;
                AtmLinkConnIDList sncComponentLinkConnList;
                ProvisionType sncProvisionType;
                MOID sncRoutingProfile;
        };

        typedef sequence<AtmSNCAllAttr> AtmSNCAllAttrList;

/**
All attributes of ATM Subnetwork
*/
        struct AtmSubnetworkAllAttr
        {
                AtmSubnetworkID subnetID;
                string subnetSystemTitle;
                CharacteristicInfo subnetCharacteristicInfo;
                AvailabilityStatus subnetAvailabilityStatus;
                string subnetUserLabel;
                MOIDList subnetSupportedByObjectList;
                AtmSNCIDList subnetContainedSNCs;
                AtmSubnetworkIDList subnetComponentSubnetworks;
                AtmLinkIDList subnetComponentLinks;
                AtmLinkEndIDList subnetSupportedLinkTPs;
        };

/**
Topology specific attributes of ATM Subnetwork
*/
        struct AtmSubnetworkTopo
        {
                AtmSubnetworkID subnettID;
                MOIDList subnettSupportedByObjectList;
                AtmSubnetworkIDList subnettComponentSubnetworks;
                AtmLinkIDList subnettComponentLinks;
                AtmLinkEndIDList subnettSupportedLinkTPs;
        };

/**
All attributes of ATM ABR Traffic Descriptor
*/
        struct AtmTrafficDescABRAllAttr
        {
                string aBRProfileName;
                ServiceCategory aBRServiceCategory;
                ConformanceDefinition aBRconformanceDefinition;
                long aBRPeakCellRate;
                long aBRCDVTolerancePCR;
                long aBRMinCellRate;
```

```
                long aBRInitialCellRate;
                long aBRTransientBufferExposure;
                RateChangeFactor aBRRateDecreaseFactor;
                RateChangeFactor aBRRateIncreaseFactor;
                long aBRFixedRoundTripTime;
                ABRNrm aBRNrm;
                ABRTrm aBRTrm;
                ABRCDF aBRCDF;
                long aBRADTF; // ZERO if not supported
        };

/**
All attributes of ATM CBR Traffic Descriptor
*/
        struct AtmTrafficDescCBRAllAttr
        {
                string cBRprofileName;
                ServiceCategory cBRServiceCategory;
                ConformanceDefinition cBRConformanceDefinition;
                long cBRPeakCellRate;
                long cBRCDVTolerancePCR;
                long cBRCLR;
        };

/**
All attributes of ATM VBR Traffic Descriptor
*/
        struct AtmTrafficDescVBRAllAttr
        {
                string vBRProfileName;
                ServiceCategory vBRServiceCategory;
                ConformanceDefinition vBRConformanceDefinition;
                long vBRPeakCellRate;
                long vBRCDVTolerancePCR;
                long vBRCDVToleranceSCR;
                        // negative if I.371 not supported
                long vBRCLR;
                long vBRSustainableCellRate;
                long vBRMaxBurstSize;
        };

/**
All attributes of ATM UBR Traffic Descriptor
*/
        struct AtmTrafficDescUBRAllAttr
        {
                string uBRProfileName;
                ServiceCategory uBRServiceCategory;
                ConformanceDefinition uBRConformanceDefinition;
                long uBREPeakCellRate;
                long uBRECDVTolerancePCR;
        };

/**
All attributes of ATM GFR Traffic Descriptor
*/
        struct AtmTrafficDescGFRAllAttr
        {
                string gFRProfileName;
                ServiceCategory gFRServiceCategory;
                ConformanceDefinition gFRConformanceDefinition;
                long peakCellRate;
                long cDVTolerancePCR;
                long maxFrameSize;
                long minCellRate;     // for CLP=0
                long maxBurstSize;
                GFR1or2 gfrOneOrTwo;
        };
```

```
/**
All attributes of ATM Traffic Descriptor Profile
The choice of union has been made based upon consideration of near-term implementation needs.
The ideal solution is to use the CORBA 2.3 valuetype for inheritance data structure.
However, this document is based upon an earlier CORBA version for which valuetype is not
available, but for which the union provides the closest available imitation of data structure
inheritance.
*/
       union AtmTrafficDescAllAttr switch (short)
       {
              case 1:
                     AtmTrafficDescABRAllAttr     trafficDescABRAll;
              case 2:
                     AtmTrafficDescCBRAllAttr     trafficDescCBRAll;
              case 3:
                     AtmTrafficDescUBRAllAttr     trafficDescUBRAll;
              case 4:
                     AtmTrafficDescVBRAllAttr     trafficDescVBRAll;
              case 5: AtmTrafficDescGFRAllAttr    trafficDescGFRAll;
       };

       typedef sequence<AtmTrafficDescAllAttr> AtmTrafficDescAllAttrList;
```

```
/**
```

5.4 Interfaces

```
INTERFACES
```

5.4.1 AtmBulkOperations

```
Operations with names ending in AllAttrlist return a sequence of all attributes for
each of the objects identified by input parameter …IDList.  The parameter howMany
specifies the maximum number of instances for which attributes can be returned in a
single message.  The returned parameter …Iterator provides an iterator for recovering
instances in excess of howMany.
*/

       interface AtmBulkOperations : NetMgmt::ManagedObject
       {
       void getSNCAllAttrList
              (in AtmSNCIDList sncIDList, // empty list implies all AtmSNCs
              in unsigned long howMany,
              out AtmSNCAllAttrList sncAllList,
              out SNCAllIterator sncAllIter)
              raises (NetMgmt::ObjectFailure);

/**
Operations with names including ID return a list of all object IDs in each identified
object class.  The parameters howMany and . . .iterator are used to regulate the size of
returned messages.  See elsewhere in this IDL for methods that return
AtmNetworkCTPIDList (under AtmLinkEnd) and AtmSNCIDList (under AtmSubnetwork).
*/

       void getNetworkTTPIDList
              (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
              in unsigned long howMany,
              out AtmNetworkTTPIDList ttpIDList,
              out TTPIDIterator ttpIDIter)
              raises (NetMgmt::ObjectFailure);

       void getNetworkTTPAllAttrList
              (in AtmNetworkTTPIDList ttpIDList, //empty list implies all AtmNetworkTTP
              in unsigned long howMany,
              out AtmNetworkTTPAllAttrList ttpAllList,
              out TTPAllIterator ttpAllIter)
              raises (NetMgmt::ObjectFailure);

       void getLinkIDList
              (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
              in unsigned long howMany,
              out AtmLinkIDList linkIDList,
              out LinkIDIterator linkIDIter)
              raises (NetMgmt::ObjectFailure);

       void getLinkAllAttrList
              (in AtmLinkIDList linkIDList, // empty list implies all AtmLinks
              in unsigned long howMany,
              out AtmLinkAllAttrList linkAllList,
              out LinkAllIterator linkAllIter)
              raises (NetMgmt::ObjectFailure);

       void getLinkEndIDList
              (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
              in unsigned long howMany,
              out AtmLinkEndIDList linkEndIDList,
              out LinkEndIDIterator linkEndIDIter)
              raises (NetMgmt::ObjectFailure);

       void getLinkEndAllAttrList
              (in AtmLinkEndIDList linkEndIDList, // empty list implies all AtmLinkEnds
```

```
            in unsigned long howMany,
            out AtmLinkEndAllAttrList linkEndAllList,
            out LinkEndAllIterator linkEndAllIter)
            raises (NetMgmt::ObjectFailure);

        void getNetworkCTPAllAttrList
            (in AtmNetworkCTPIDList ctpIDList, //empty list implies all AtmNetworkCTP
            in unsigned long howMany,
            out AtmNetworkCTPAllAttrList netCTPAllList,
            out NetworkCTPAllIterator netCTPAllIter)
            raises (NetMgmt::ObjectFailure);

        void getNetworkAccessProfileIDList
            (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
            in unsigned long howMany,
            out AtmNetworkAccessProfileIDList accProIDList,
            out NetworkAccessProfileIDIterator accProIDIter)
            raises (NetMgmt::ObjectFailure);

        void getNetworkAccessProfileAllAttrList
            (in AtmNetworkAccessProfileIDList accProIDList, // empty list implies all
            in unsigned long howMany,
            out AtmNetworkAccessProfileAllAttrList accProAllList,
            out NetworkAccessProfileAllIterator accProAllIter)
            raises (NetMgmt::ObjectFailure);

        void getTrafficDescIDList
            (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
            in unsigned long howMany,
            out AtmTrafficDescIDList trafDescIDList,
            out TrafficDescIDIterator trafDescIDIter)
            raises (NetMgmt::ObjectFailure);

        void getTrafficDescAllAttrList
            (in AtmTrafficDescIDList trafDescIDList, // empty list implies all
            in unsigned long howMany,
            out AtmTrafficDescAllAttrList trafDescAllList,
            out TrafficDescAllIterator trafDescAllIter)
            raises (NetMgmt::ObjectFailure);

        void getLinkEndPhyIDList
            (in unsigned long howMany,
            out AtmLinkEndPhyIDList linkEndPhyIDList, // all AtmLinkEndPhy instances
            out LinkEndPhyIDIterator linkEndPhyIDIter)
            raises (NetMgmt::ObjectFailure);

        void getLinkEndPhyAllAttrList
            (in AtmLinkEndPhyIDList linkEndPhyIDList, // empty list implies all
            in unsigned long howMany,
            out AtmLinkEndPhyAllAttrList linkEndPhyAllList,
            out LinkEndPhyAllIterator linkEndPhyAllIter)
            raises (NetMgmt::ObjectFailure);

        void getRoutingProfileIDList
            (in AtmLNDIDList lndIDList, // empty list implies all ATM LNDs
            in unsigned long howMany,
            out AtmRoutingProfileIDList routProIDList,
            out RoutingProfileIDIterator routProIDIter)
            raises (NetMgmt::ObjectFailure);

        void getRoutingProfileAllAttrList
            (in AtmRoutingProfileIDList routProIDList, // empty list implies all
            in unsigned long howMany,
            out AtmRoutingProfileAllAttrList routProAllList,
            out RoutingProfileAllIterator routProAllIter)
            raises (NetMgmt::ObjectFailure);

        };     // end of interface AtmBulkOperations
```

```
/**
```

5.4.2 Supporting Iterator Interfaces

```
The following iterator interfaces provide a means for limiting the length of messages
returned by methods in the AtmBulkOperations interface defined above.  Each iterator
interface would invoke its inherited destroy () method for the purpose of resetting or
reinitialization.
*/
       interface SNCAllIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmSNCAllAttrList sncAllList);
       };
       interface TTPIDIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmNetworkTTPIDList netTTPIDList);
       };
       interface TTPAllIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmNetworkTTPAllAttrList netTTPAllList);
       };
       interface LinkIDIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmLinkIDList linkIDList);
       };
       interface LinkAllIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmLinkAllAttrList linkAllList);
       };
       interface LinkEndIDIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmLinkEndIDList linkEndIDList);
       };
       interface LinkEndAllIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmLinkEndAllAttrList linkEndAllList);
       };
       interface NetworkCTPAllIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmNetworkCTPAllAttrList netCTPAllList);
       };
       interface NetworkAccessProfileIDIterator : NetMgmt::ManagedObject
       {
              boolean nextN
                     (in unsigned long howMany,
                     out AtmNetworkAccessProfileIDList accProIDList);
       };
```

```
	interface NetworkAccessProfileAllIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmNetworkAccessProfileAllAttrList accProAllList);
	};
	interface TrafficDescIDIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmTrafficDescIDList trafDescIDList);
	};
	interface TrafficDescAllIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmTrafficDescAllAttrList trafDescAllList);
	};
	interface LinkEndPhyIDIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmLinkEndPhyIDList linkEndPhyIDList);
	};
	interface LinkEndPhyAllIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmLinkEndPhyAllAttrList linkEndPhyAllList);
	};
	interface RoutingProfileIDIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmRoutingProfileIDList routProIDList);
	};
	interface RoutingProfileAllIterator : NetMgmt::ManagedObject
	{
		boolean nextN
			(in unsigned long howMany,
			out AtmRoutingProfileAllAttrList routProAllList);
	};


/**
```

5.4.3 AlarmSeverityAssignmentProfile

```
**/
	interface AlarmSeverityAssignmentProfile: NetMgmt::ManagedObject, NetMgmt::Portal
	{
/**
This method is used to retrieve the object's Alarm Severity Assignment List.
*/
		AlarmSeverityAssignmentSetType getAlarmSeverityAssignmentList
			(in NameType profileName)
			raises (NetMgmt::ObjectFailure);

/**
This method is used to add an alarm to the object's Alarm Severity
Assignment List.  An Attribute Value Change notification will be sent if the
object supports it.  If an exception is thrown, the object is not changed.
*/
		void addAlarmSeverityAssignments
			(in NameType profileName,
			in AlarmSeverityAssignmentSetType
				alarmSeverityAssignmentList)
			raises (NetMgmt::ObjectFailure);
```

```
/**
This method is used to remove entries from the object's Alarm Severity
Assignment List.  An Attribute Value Change notification will be sent if the
object supports it.  If an exception is thrown, the object is not changed.
*/
              void removeAlarmSeverityAssignments
                     (in NameType profileName,
                     in AlarmSeverityAssignmentSetType
                            alarmSeverityAssignmentList)
                     raises (NetMgmt::ObjectFailure);

/**
This method is used to replace all the entries in the object's Alarm
Severity Assignment List with the submitted list.  An Attribute Value Change
notification will be sent if the object supports it.  If an exception is
thrown, the object is not changed.
*/
              void setAlarmSeverityAssignmentList
                     (in NameType profileName,
                     in AlarmSeverityAssignmentSetType
                            alarmSeverityAssignmentList)
                     raises (NetMgmt::ObjectFailure);

       }; // interface AlarmSeverityAssignmentProfile

       interface AlarmSeverityAssignmentProfileFactory: NetMgmt::ManagedObject
       {
              AlarmSeverityAssignmentProfile create
                     (in NameType profileName,
                     inout string name,    // auto naming if null
                     in AlarmSeverityAssignmentSetType list)
                            // alarmSeverityAssignmentProfilePackage
                            // GET-REPLACE, ADD-REMOVE
                     raises (NetMgmt::ObjectFailure);

       }; // interface AlarmSeverityAssignmentProfileFactory
```

**/ * *

5.4.4 AtmLink

```
*/
       interface AtmLink : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect Creation by the EMS.

An atmLink is a topological component used to describe a fixed relationship
between two atmSubnetworks (through the contained atmLinkTP instances) and
represents a topological association along with capacity.  Many atmLinks may
exist between a pair of atmSubnetworks, although an atmLink may not exist
between a composite atmSubnetwork and any of its component atmSubnetworks.
An atmLink is terminated by two atmLinkTPs, one in each atmSubnetwork.  These
atmLinkTP instances may exist before an instance of atmLink may be created,
otherwise they are created as a result of the setupLinkAction.  An instance of
atmLink is created by the managed system or by using the setupLink ACTION.
Overlapping links (and address ranges) are not allowed.

If the availabilityStatus is failed or degraded, the atmLink object shall not
allow new atmLinkConnections to be established.

Supported values for the availabilityStatus are:
- Failed:  The atmLink cannot function.  All underlying transport connections
have failed.
- Degraded:  The atmLink is degraded in some respect.  For instance, the
atmLink cannot perform the function of establishing new atmLinkConnections
while it can still accept ACTIONS to tear down existing connections.
- Empty SET (none of the availableStatus conditions exist).
```

```
The administrativeState is used for administratively locking and unlocking the
atmLink.  When unlocked, the atmLink functions normally.  When in the locked
state, the atmLink is prohibited from the set-up, modification, or release of
link connections, thus any of these actions shall be rejected.  Locking an
atmLink does not automatically lock the contained atmLinkConnections.

The characteristicInformation attribute describes the format of the
characteristic information that the resource carries.  The attribute value
is set to vcCI (I.751) for VC Layer atmLinks and vpCI (I.751) for VP Layer
atmLinks.

Note that the related atmNetworkAccessProfile information is also in the
NE-view atmAccessProfile object contained in the tcAdaptorTTPBidirectional or
in the vpTTPBidirectional object.  The characteristics described by the
atmNetworkAccessProfile associated with an atmLink shall be consistent with
the atmNetworkAccessProfile of the related atmLinkTPs.

The setupLinkConnection ACTION sets up a point-to-point connection between two
non-connected subnetworkTPs in the each of the linked atmSubnetworks.

The modifyLinkConnection ACTION modifies the QOS and traffic descriptors of a
point-to-point connection between two connected subnetworkTPs in the two
linked atmSubnetworks.

The releaseLinkConnection ACTION releases a point-to-point connection between
subnetworkTPs in each of the linked atmSubnetwork.

Notifications Supported
Attribute Value Change: This notification is used to report changes of the
bandwidth values.

State Change:  This notification is used to report changes to the State
attributes of this managed entity.  The notification shall identify the state
attribute that changed, its old value, and its new value.

Managed Entity Creation:  This notification is used to report the creation of
an instance of this Managed Entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this Managed Entity.
*/
              AtmLinkAllAttr getAllAttrLink
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute, set at creation, describes the signal that
is transferred across the link: VP or VC.
*/
              CharacteristicInfo getCharacteristicInfo
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute identifies whether or not this instance of
the link managed entity is capable of  performing its normal
function (i.e., transport ATM cells).
*/
              AvailabilityStatus getAvailabilityStatus
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure);
              AdministrativeState getAdminState
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              void setAdminState
                     (in NameType atmLinkName,
                     in AdministrativeState adminState )
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
```

```
/**
This string identifies the customer who may use a private link.  If
the value of this attribute is set to NULL, then the link may be
assumed to be a non-private link.  Only connections of the customer
identified by the customerID attribute shall be established across a
private link.
*/
               string getCustomerID
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setCustomerID
                      (in NameType atmLinkName,
                      in string customerID)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
A Link that represents a unary link has two logical end points, one
on each subnetwork that it is linking.
*/
               AtmLinkEndID getALinkEnd
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure);
               AtmLinkEndID getZLinkEnd
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure);

/**
Each Link may use one atmNetworkAccessProfile.
*/
               AtmNetworkAccessProfileID getNetworkAccessProfile
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setNetworkAccessProfile
                      (in NameType atmLinkName,
                      in AtmNetworkAccessProfileID networkAccessProfile)
                      raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID,
                             NetMgmt::NotSupported);

/**
This read/write attribute is used to configure the restoration mode
of a link as: unavailable for routing and re-routing, available for
routing and not re-routing; available for re-routing and not
routing; or available for both routing and rerouting.
*/
               RestorationMode getRestorationMode
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setRestorationMode
                      (in NameType atmLinkName,
                      in RestorationMode restorationMode)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This integer value describes the relative weight of using the link.
The specific value of this attribute is determined by the manager
who sets the linkWeight parameter.  This attributed takes on a ZERO
value in cases where the link is not assigned a specific weight.
*/
               long getLinkWeight
                      (in NameType atmLinkName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setLinkWeight
                      (in NameType atmLinkName,
                      in long provisionedBW )
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
```

```
A Link is a group of link connections sharing the same extremities.
This relationship involves one and only one instance of the Link
managed entity, and zero or more instances of the linkConnection
managed entity.
*/
              AtmLinkConnIDList getContainedLinkConns
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
One Link has a relationship with the two and only two subnetworks
that it is linking. A Link cannot exists without the subnetworks
being identified.
*/
              AtmSubnetworkIDList getLinkedSubnetworks
                     (in NameType atmLinkName)
                     raises (NetMgmt::ObjectFailure);

/**
This operation allows the requester to set-up a
linkConnection between two non-connected networkCTPs of two
subnetworks.
*/
              void setupLinkConnWithCTP
                     (in NameType atmLinkName,
                     in string userLabel,
                     in OptRestorableType restorableType,
                     inout OptAdministrativeState adminState,
                     in AtmNetworkCTPID aNetworkCTP,
                     in boolean aTrailEndPointInd,
                     in AtmTrafficDescID aTozTrafficDescProfile,
                     in AtmNetworkCTPID zNetworkCTP,
                     in boolean zTrailEndPointInd,
                     in AtmTrafficDescID zToaTrafficDescProfile,
                     in AtmRoutingProfileID routingProfile,
                     out AtmLinkConnID newLinkConn )
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::InvalidID,
                            NetMgmt::ItemNotFound);

/**
This operation allows the requester to set-up a
linkConnection across the Link between two subnetworks.
*/
              void setupLinkConnOnLink
                     (in NameType atmLinkName,
                     in string userLabel,
                     in OptRestorableType restorableType,
                     inout OptAdministrativeState adminState,
                     inout VirtualID aVirtualID,
                     in boolean aTrailEndPointInd,
                     in AtmTrafficDescID aTozTrafficDescProfile,
                     inout VirtualID zVirtualID,
                     in boolean zTrailEndPointInd,
                     in AtmTrafficDescID zToaTrafficDescProfile,
                     in AtmRoutingProfileID routingProfile,
                     out AtmLinkConnID newLinkConn,
                     out AtmNetworkCTPID aNetworkCTP,
                     out AtmNetworkCTPID zNetworkCTP )
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::InvalidID,
                            NetMgmt::ItemNotFound);

/**
This operation allows for the release of a linkConnection between two
connected networkCTPs of the two different subnetworks, the linkConnection or
```

```
the networkCTPs involved identified directly.
*/
              void releaseLinkConn
                     (in NameType atmLinkName,
                     in MOID connectionID) // networkCTP or LinkConn
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::InvalidID,
                            NetMgmt::ItemNotFound);

       }; // interface AtmLink

/**
```

5.4.5 AtmLinkConn

```
*/
       interface AtmLinkConn : NetMgmt::ManagedObject, NetMgmt::Portal
       {
       }; // interface AtmLinkConn
```

```
/**
```

5.4.6 AtmLinkEnd

```
*/
       interface AtmLinkEnd : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect Creation by the EMS.

This managed entity is used to represent the termination of a pure topological
Link in an ATM network.  In the VP LND, a Link End represents an ATM interface
associated with the underlying transport facility.

In addition, interface and server trail related information may be represented
in the ATM Link End.  That is, the Link End may be used to represent the
appropriate server trail TP information, removing the need to represent the
server trail TPs across the M4 network view interface.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this managed entity.

Attribute Value Change:  This notification is used to report changes to the
attribute changes of this managed entity.  The notification shall identify the
attribute that changed, its old value, and its new value.  This is used to
provide changes is Availability Status, and serverTTPOperationalState

State Change:  This notification is used to report changes to the State
attributes of this managed entity.  The notification shall identify the state
attribute that changed, its old value, and its new value.
*/

              AtmLinkEndAllAttr getAllAttrLinkEnd
                     (in NameType atmLinkEndName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              CharacteristicInfo getCharacteristicInfo
                     (in NameType atmLinkEndName)
                     raises (NetMgmt::ObjectFailure);

/**
This read-only attribute describes the operational status (working,
degraded, not-working) of the ATM Interface represented by the
LinkEnd.
*/
              AvailabilityStatus getAvailabilityStatus
                     (in NameType atmLinkEndName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This settable attribute allows for the configuration of the
administrative state of the ATM Interface represented by the
LinkEnd.
*/
              AdministrativeState getAdminState
                     (in NameType atmLinkEndName)
                     raises (NetMgmt::ObjectFailure);
              void setAdminState
                     (in NameType atmLinkEndName,
                     in AdministrativeState adminState )
                     raises (NetMgmt::ObjectFailure);

/**
This string may be used to describe additional information about the
atmLinkEnd, such as a circuit identifier.
*/
```

```
                string getUserLabel
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);
                void setUserLabel
                        (in NameType atmLinkEndName,
                        in string customerID)
                        raises (NetMgmt::ObjectFailure);

/**
Each LinkEnd may use one atmNetworkAccessProfile.
*/
                AtmNetworkAccessProfileID getNetworkAccessProfile
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setNetworkAccessProfile
                        (in NameType atmLinkEndName,
                        in AtmNetworkAccessProfileID networkAccessProfile)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID,
                                NetMgmt::NotSupported);

/**
Describes the interface type that the atmLinkEnd supports:
UNI, inter-NNI, intra-NNI, or unconfigured.
*/
                LinkEndType getLinkEndType
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);
                void setLinkEndType
                        (in NameType atmLinkEndName,
                        in LinkEndType linkEndType )
                        raises (NetMgmt::ObjectFailure);

/**
This read only attribute identifies the maximum amount of bandwidth
assignable on the link in the Ingress direction (inbound or towards
the ATM NE).
*/
                long getIngressMaxAssignBW
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);

/**
This read only attribute identifies the maximum amount of bandwidth
assignable on the link in the Egress direction (outbound or away
from the ATM NE).
*/
                long getEgressMaxAssignBW
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);

/**
This read-only attribute identifies the amount of bandwidth left on the link
in the Ingress direction (inbound or towards the ATM NE).
*/
                long getIngressAvailableBW
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);

/**
This read-only attribute identifies the amount of bandwidth left on
the link in the Egress direction (outbound or away from the ATM NE).
*/
                long getEgressAvailableBW
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);

/**
Each TopologicalLink may be terminated by two instances of the
```

```
LinkEnd managed entity.
*/
               AtmLinkID getSupportedLink
                      (in NameType atmLinkEndName)
                      raises (NetMgmt::ObjectFailure);

/**
Each LinkEnd may be supported by one instance of a TTP managed
entity in the serverLayer.
*/
               MOID getServerTTP
                      (in NameType atmLinkEndName)
                      raises (NetMgmt::ObjectFailure);
               void setServerTTP
                      (in NameType atmLinkEndName,
                      in MOID serverTTP)
                      raises (NetMgmt::ObjectFailure);

/**
A list of CTPs within the same layer network domain that are
supported by the LinkEnd.  This provides the association between the
LinkEnd (and underlying server trail) and the same layer network
domain CTPs supported at the LinkEnd.  That is, a VP vpLinkEnd
identifies the VP CTPs supported at the interface point. In cases
where VP and VC are managed together, this attribute may be used to
list both VP and VC Layer networkCTPs.
*/
               AtmNetworkCTPIDList getSupportedCTPs
                      (in NameType atmLinkEndName)
                      raises (NetMgmt::ObjectFailure);
               void addSupportedCTP
                      (in NameType atmLinkEndName,
                      in AtmNetworkCTPID supportedCTP )
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                             NetMgmt::InvalidID, NetMgmt::DuplicateItem);
               void removeSupportedCTP
                      (in NameType atmLinkEndName,
                      in AtmNetworkCTPID supportedCTP )
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                             NetMgmt::InvalidID, NetMgmt::ItemNotFound);

/**
A code used for OAM cell loopback purposes.  Incoming OAM Loopback
cells with a Loopback Location field value that matches the value of
the loopbackLocationIdentifier attribute shall be looped-back over
the interface.
*/
               LoopbackLocationCode getLoopbackLocID
                      (in NameType atmLinkEndName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setLoopbackLocID
                      (in NameType atmLinkEndName,
                      in LoopbackLocationCode loopbackloc)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute identifies the VPI/VCI value used over the
UNI to support ILMI.
*/
               VirtualID getIlmiVpiVci
                      (in NameType atmLinkEndName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setIlmiVpiVci
                      (in NameType atmLinkEndName,
                      in VirtualID ilmiVpiVci)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               long getIlmiEstabConnectivityPollInterval
                      (in NameType atmLinkEndName)
```

```
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setIlmiEstabConnectivityPollInterval
                        (in NameType atmLinkEndName,
                        in long ilmiEstabInt)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                long getIlmiCheckConnectivityPollInterval
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setIlmiCheckConnectivityPollInterval
                        (in NameType atmLinkEndName,
                        in long ilmiCheckInt)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                long getIlmiConnectivityPollFactor
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setIlmiConnectivityPollFactor
                        (in NameType atmLinkEndName,
                        in long ilmiPollFactor)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This parameter identifies the location identifier of the NE that
supports the LinkEnd.
*/
                string getSupportingNeLoc
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setSupportingNeLoc
                        (in NameType atmLinkEndName,
                        in string supportingNeLoc)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This parameter identifies the location identifier of the circuit pack that
supports the LinkEnd.
*/
                PortID getSupportingPortID
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setSupportingPortID
                        (in NameType atmLinkEndName,
                        in PortID supportingPortID )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the type of the Server TTP that is
represented by the vpLinkEnd.
*/
                CharacteristicInfo getServerTTPCharInfo
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);

/**
This optional attribute indicates port id of the Server TTP that is
represented by the vpLinkEnd.
*/
                PortID getServerTTPPortID
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setServerTTPPortID
                        (in NameType atmLinkEndName,
                        in PortID serverTTPPortID )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates current operational state of the
Server TTP that is represented by the vpLinkEnd.
*/
```

```
                OperationalState getServerTTPOpState
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute allows cell scrambling to be activated or
deactivated on the ATM Interface represented by the vpLinkEnd.
*/
                boolean getCellScramblingEnabled
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setCellScramblingEnabled
                        (in NameType atmLinkEndName,
                        in boolean cellScramblingEnabled)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional read/write attribute describes the subscriber address
associated with the vpLinkEnd.
*/
                StringList getSubscriberAddressList
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void addSubscriberAddress
                        (in NameType atmLinkEndName,
                        in string subscriberAddress )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeSubscriberAddress
                        (in NameType atmLinkEndName,
                        in string subscriberAddress )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);

/**
This optional read/write attribute provides and identification of
the preferred carrier if it is directly assigned to the vpLinkEnd.
*/
                StringList getPreferredCarrierList
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void addPreferredCarrier
                        (in NameType atmLinkEndName,
                        in string subscriberAddress )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removePreferredCarrier
                        (in NameType atmLinkEndName,
                        in string subscriberAddress )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);

/**
Optional pointer to a vendor specific profile.
*/
                MOID getVendorProfile
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void addVendorProfile
                        (in NameType atmLinkEndName,
                        in MOID vendorProfile )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeVendorProfile
                        (in NameType atmLinkEndName,
                        in MOID vendorProfile )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);
```

```
                void linkPVCTrace
                       (in NameType atmLinkEndName,
                       in LinkTraceType linkTraceType,
                       in AtmSubnetworkIDList selectedSubnets,
                       out SNCsBySubnetList traceResults)
                       raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                              NetMgmt::InvalidID, NetMgmt::ItemNotFound);

        }; // interface AtmLinkEnd
```

5.4.7 AtmLinkEndPhy

```
/**
*/
        interface AtmLinkEndPhy : AtmLinkEnd
        {
/**
Expect Creation by the EMS.

This subclass of ATM Link End is used to represent the termination of a VP
topological Link in an ATM network and its supporting transport termination.

Server trail related information is represented in the ATM Link End Phy.  That
is, the Link End Phy. is used to represent the appropriate server trail TP
information, removing the need to represent the server trail TPs across the
M4 network view interface.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this managed entity.

Attribute Value Change:  This notification is used to report changes to the
attribute changes of this managed entity.  The notification shall identify the
attribute that changed, its old value, and its new value.  This is used to
provide changes is Availability Status, and serverTTPOperationalState

State Change:  This notification is used to report changes to the State
attributes of this managed entity.  The notification shall identify the state
attribute that changed, its old value, and its new value.

Alarm:  This message is used to notify the management system when a failure
has been detected or cleared.  The following parameters shall be supplied with
this notification:
        -       The Nature of the Alarm (i.e., see generic trouble list)
        -       Specific Problems (optional)
        -       The ID of the Managed Entity Reporting the Alarm
        -       The Failed Switch Component or List of Failed
                (or Possibly Failed) Components
        -       Back-up Status (optional)
                This is a Boolean indication as to whether or not the failed
                entity has been backed-up.
        -       Back-up Entity (optional)
                This is the ID of the managed entity providing back-up services
        to the failed entity.  This parameter shall be NULL when the
                value of the "Back-up Status" parameter is false.
        -       Severity of Failure (critical, major, minor, warning,
                indeterminate, and cleared)
        -       Additional Information (optional)
        -       Proposed Repair Actions (optional)
        -       Time and Date Failure was Detected
*/
```

```
                AlarmSeverityAssignmentProfile
                        getAlarmSeverityAssignmentProfile
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure);
                void setAlarmSeverityAssignmentProfile
                        (in NameType atmLinkEndName,
                        in AlarmSeverityAssignmentProfile profile)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

                CurrentProblemList getCurrentProblemList
                        (in NameType atmLinkEndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

        }; // interface AtmLinkEndPhy


/**
```

5.4.8 AtmLND

```
*/
        interface AtmLND : NetMgmt::ManagedObject, NetMgmt::Portal
        {
/**
Expect Creation by the EMS.

The atmLayerNetworkDomain object class represents the part of the VC or VP
Layer which is available to a managing system through the M4 interface.
The atmLayerNetworkDomain corresponds to an administration.  An
atmLayerNetworkDomain is defined to support the requirement for independent
layer management of either the VC Layer or the VP Layer. The
atmLayerNetworkDomain object represents part of an administration's portion of
the VC or VP Layer which is available to a managing system through the M4
interface.  In this model, an ATM Layer Network Domain is associated with one
and only one top subnetwork, which can be further decomposed. There may be
several Layer Network Domains within a single Network.

An atmLayerNetworkDomain is defined to support the requirement for independent
layer management of the VC and VP Layers.

The userLabel may be used to represent additional information about the layer
network domain.  In cases where the vcLayerNetworkDomain is managed by a
different system the systemTitle may be used.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this managed entity.

Attribute Value Change: This notification is used to report changes of the
user label.
*/
                AtmLNDAllAttr getAllAttrLND
                        (in NameType atmLndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                string getSystemTitle
                        (in NameType atmLndName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                CharacteristicInfo getCharacteristicInfo
                        (in NameType atmLndName)
                        raises (NetMgmt::ObjectFailure);

/**
This read/write attribute identifies the managing organization.
*/
                string getUserLabel
```

```
                        (in NameType atmLndName)
                        raises (NetMgmt::ObjectFailure);
                void setUserLabel
                        (in NameType atmLndName,
                        in string userLabel)
                        raises (NetMgmt::ObjectFailure);

/**
Make Link TP
*/
                AtmLinkEndID makeLinkEnd
                        (in NameType atmLndName,
                        in NameType linkEndName, // auto named if null
                        in MOID serverTTP,  // server TTP or
                        in MOID serverInterface, // server interface
                        in long vpiValue, // if making VC Link TP
                        in AtmNetworkAccessProfileID accessProfile,
                        in string userLabel,
                        in MOID vendorProfile)
                        raises (NetMgmt::ObjectFailure,
                                NetMgmt::NotSupported,
                                NetMgmt::InvalidID,
                                NetMgmt::ItemNotFound);

// Setup Link Method FFS     //
// Remove Link Method FFS    //
// Create CTP/TTP Method FFS //
// Remove CTP/TTP Method FFS //

        }; // interface AtmLND
```

```
/**
```

5.4.9 AtmNetworkCTP

```
*/
       interface AtmNetworkCTP : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect creation using AtmSubnetwork: setupPtToPtSNCWithLinkTP,
creation using atmVcLND: createNetworkCTP (for VC CTPs), or
creation using atmVpLND: createNetworkCTP (for VP CTPs)

Expect deletion using AtmSubnetwork: releaseSNC

his managed entity is used to represent the termination of a VP or VC
connections on an ATM subnetwork.  An instance of the SubnetworkConnection or
of a linkConnection managed entity may be used to relate two instances of the
Network Connection Termination Point managed entity (i.e., for point-to-point
cross connection).

The relatedAtmTTP attribute is used to associate the final CTP of a VCC or VPC
with the same layer Trail Termination Point.  Other attributes reflect the
VCI/VPI (depending on network layer), traffic descriptors, and quality of
service class.

The relatedTrafficDescriptors attribute may be used to point to the traffic
descriptor profile at points where ingress and/or egress UPC/NPC functions
are performed or when the relatedAtmTTP attribute points to an instance of the
atmNetworkTTP object class.  The object pointed to by the
relatedTrafficDescriptors attribute may also contain QOS information.

The tmnCommunicationsAlarmInformation allows the reporting of communications
alarms associated with the atmNetworkCTP.  When an AIS or RDI failure is
detected and alarm reporting is supported, the atmNetworkCTP object shall
generate a communicationsAlarm notification with the probableCause parameter
value set equal to aIS or farEndReceiverFailure, respectively.

The conditional oamCellLoopback provides the method used to request the
termination point to insert an OAM cell for downstream loopback and to report
whether or not the cell was returned within the required time.

Notifications Supported
Alarm:  This message is used to notify the management system when a failure
has been detected or cleared.  The following parameters shall be supplied with
this notification:
       -      The Nature of the Alarm (i.e., see generic trouble list)
       -      Specific Problems (optional)
       -      The ID of the Managed Entity Reporting the Alarm
       -      The Failed Switch Component or List of Failed (or Possibly
              Failed) Components
       -      Back-up Status (optional)
              This is a Boolean indication as to whether or not the failed
              entity has been backed-up.
       -      Back-up Entity (optional)
              This is the ID of the managed entity providing back-up services
       to the failed entity.  This parameter shall be NULL when the
              value of the "Back-up Status" parameter is false.
       -      Severity of Failure (critical, major, minor, warning,
              indeterminate, and cleared)
       -      Additional Information (optional)
       -      Proposed Repair Actions (optional)
       -      Time and Date Failure was Detected

Attribute Value Change:  This notification is used to report changes to the
attributes of this managed entity.  The notification shall identify the
attribute that changed, its old value, and its new value.

Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
```

```
Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this managed entity.
*/
               AtmNetworkCTPAllAttr getAllAttrNetworkCTP
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               CharacteristicInfo getCharacteristicInfo
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
User Label may be used to provide a master connection name for each connection
end-point at the edge of the network. This field may identify the
administrative name used by the adjacent carrier
*/
               string getUserLabel
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setUserLabel
                      (in NameType atmNetworkCTPName,
                      in string userLabel)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute identifies the VPI/VCI value associated
with the connection being terminated
*/
               VirtualID getNetworkCTPVpiVci
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure);

/**
This boolean attribute indicates whether the NetworkCTP object
instance has been configured to represent an end-point of a VCC or
VPC Segment
*/
               boolean getSegmentEndpoint
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure);
               void setSegmentEndpoint
                      (in NameType atmNetworkCTPName,
                      in boolean segmentEndpoint)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
one instance of the Traffic Descriptor managed entity may
characterize the CTP.
*/
               AtmTrafficDescID getEgressTrafficDescProfile
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               AtmTrafficDescID getIngressTrafficDescProfile
                      (in NameType atmNetworkCTPName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setTrafficDescProfile
                      (in NameType atmNetworkCTPName,
                      in AtmTrafficDescID egressTrafficDescProfile,
                      in AtmTrafficDescID ingressTrafficDescProfile)
                      raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID,
                             NetMgmt::NotSupported);

/**
Zero or one instance of the NetworkTTP managed entity may exist for
each instance of a CTP managed entity, depending on if the trail
termination coincides with the CTP
*/
               AtmNetworkTTPID getRelatedAtmTTP
```

```
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setRelatedAtmTTP
                        (in NameType atmNetworkCTPName,
                        in AtmNetworkTTPID relatedAtmTTP)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID,
                                NetMgmt::NotSupported);

/**
Zero or more of the NetworkCTP managed entity may exist for each
instance of a  subnetworkConnection managed entity
*/
                AtmSNCIDList getAssociatedSNCs
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This operation is used to request that the NetworkCTP insert a
loopback OAM cell into the ATM cell stream, verify its return, and
report the results of the loopback (i.e., passed or failed) back to
the management system. Along with each request will be the location
where the inserted OAM cell shall loop-back and an indication as to
whether a segment or end-to-end OAM cell shall be used.  The
Loopback Location Code which indicates where the loopback is to take
place may be used to identify the loopback location.  Additionally,
a globally unique default value (e.g., "end-point") may also be used
to perform a loopback at the other end of a VCC or VPC.
*/
                LoopbackCellReply loopbackOamCell
                        (in NameType atmNetworkCTPName,
                        in LoopbackLoc loopbackLoc,
                        in PmOamCellType oamCellType )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AlarmSeverityAssignmentProfile
                        getAlarmSeverityAssignmentProfile
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure);
                void setAlarmSeverityAssignmentProfile
                        (in NameType atmNetworkCTPName,
                        in AlarmSeverityAssignmentProfile profile)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);
                CurrentProblemList getCurrentProblemList
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This boolean attribute specifies if tagging is being used
on the receive side of an ATM VPC or VCC.  A value of true indicates
that tagging is being used.  A value of false indicates it is not being used.
*/
                boolean getIngressTaggingInd
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setIngressTaggingInd
                        (in NameType atmNetworkCTPName,
                        in boolean ingressTagInd)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This boolean attribute specifies if tagging is being used
on the transmit side of an ATM VPC or VCC.  A value of true
indicates that tagging is being used.  A value of false indicates it
is not being used.
*/
                boolean getEgressTaggingInd
                        (in NameType atmNetworkCTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setEgressTaggingInd
```

```
                    (in NameType atmNetworkCTPName,
                    in boolean egressTagInd)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the method used to setup and
terminate the PM OAM monitoring activity.  Valid values are TMN,
OAM, or notSupported.   If the value is notSupported, then PM OAM is
not supported on the endpoint
*/
             PmOamMethod getPmOamMethod
                    (in NameType atmNetworkCTPName)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
             void setPmOamMethod
                    (in NameType atmNetworkCTPName,
                    in PmOamMethod pmOamMethod )
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the desired direction(s) of
transmission to monitor PM OAM.  Valid directions are:  away from
activator (transmit), towards activator (receive), or both
*/
             PmOamDirection getPmOamDirection
                    (in NameType atmNetworkCTPName)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
             void setPmOamDirection
                    (in NameType atmNetworkCTPName,
                    in PmOamDirection pmOamDirection )
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the PM OAM nominal block size
choice for both the receive and transmit dirctions
*/
             PmOamBlockSize getPmOamBlockSize
                    (in NameType atmNetworkCTPName)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
             void setPmOamBlockSize
                    (in NameType atmNetworkCTPName,
                    in PmOamBlockSize pmBlockSize)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional boolean attribute is used to initiate generation of PM
OAM cells in the forward direction by setting the value to true
*/
             boolean getPmOamForwardActive
                    (in NameType atmNetworkCTPName)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
             void setPmOamForwardActive
                    (in NameType atmNetworkCTPName,
                    in boolean pmForwardActive )
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional boolean attribute is used to initiate generation of PM
OAM cells in the backward direction by setting the value to true
*/
             boolean getPmOamBackwardActive
                    (in NameType atmNetworkCTPName)
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
             void setPmOamBackwardActive
                    (in NameType atmNetworkCTPName,
                    in boolean pmBackwardActive )
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

       }; // interface AtmNetworkCTP
```

```
/**
```

5.4.10 AtmNetworkTTP

```
*/
       interface AtmNetworkTTP : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect creation using AtmSubnetwork: setupPtToPtSNCWithCTP or
AtmSubnetwork: setupPtToPtSNCWithLinkTP

Expect deletion using AtmSubnetwork: releaseSNC

The atmNetworkTTP object class is used when the Network View only is provided.

The relatedAtmCTP attribute is used to associate the final CTP of a VCC or VPC
with the Trail Termination Point.

The optional method is provided to request the termination point to insert an
OAM cell for downstream loopback and to report whether or not the cell was
returned within the required time.

The availabilityStatus may be used to indicate the availability of the
atmNetworkTTP.  Changes in the availabilityStatus are reported using the
attributeValueChangeNotification.

Supported values for the availabilityStatus are:
- Failed:  The atmLinkConnection cannot function
- Empty SET (none of the availableStatus conditions exist).

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this managed entity.

Attribute Value Change Notification:  This notification is used to report
changes to the availability status attribute of this managed entity.
*/
              AtmNetworkTTPAllAttr getAllAttrNetworkTTP
                     (in NameType atmNetworkTTPName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              CharacteristicInfo getCharacteristicInfo
                     (in NameType atmNetworkTTPName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute identifies the VPI/VCI value associated
with the trail being terminated.
*/
              VirtualID getNetworkCTPVpiVci
                     (in NameType atmNetworkTTPName)
                     raises (NetMgmt::ObjectFailure);

/**
This read-only attribute identifies whether or not the managed
entity is capable of performing its normal functions (Failed or no
unavailability condition existing).
*/
              AvailabilityStatus getAvailabilityStatus
                     (in NameType atmNetworkTTPName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
Zero or one instance of the NetworkTTP managed entity may
exist for each instance of a NetworkCTP managed entity MO.
*/
              AtmNetworkCTPID getRelatedAtmCTP
```

```
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setRelatedAtmCTP
                        (in NameType atmNetworkTTPName,
                        in AtmNetworkCTPID relatedAtmCTP)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID,
                                NetMgmt::NotSupported);

/**
A Trail is terminated by two vpTTPs
*/
                MOID getAssociatedTrail
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                LoopbackCellReply loopbackOamCell
                        (in NameType atmNetworkTTPName,
                        in LoopbackLoc loopbackLoc,
                        in PmOamCellType oamCellType )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AlarmSeverityAssignmentProfile
                        getAlarmSeverityAssignmentProfile
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure);
                void setAlarmSeverityAssignmentProfile
                        (in NameType atmNetworkTTPName,
                        in AlarmSeverityAssignmentProfile profile)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);
                CurrentProblemList getCurrentProblemList
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the method used to setup and
terminate the PM OAM monitoring activity.  Valid values are TMN,
OAM, or notSupported.   If the value is notSupported, then PM OAM is
not supported on the endpoint.
*/
                PmOamMethod getPmOamMethod
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setPmOamMethod
                        (in NameType atmNetworkTTPName,
                        in PmOamMethod pmOamMethod )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the desired direction(s) of
transmission to monitor PM OAM.  Valid directions are:  away from
activator (transmit), towards activator (receive), or both.
*/
                PmOamDirection getPmOamDirection
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setPmOamDirection
                        (in NameType atmNetworkTTPName,
                        in PmOamDirection pmOamDirection )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional attribute indicates the PM OAM nominal block size
choice for both the receive and transmit dirctions.
*/
                PmOamBlockSize getPmOamBlockSize
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setPmOamBlockSize
                        (in NameType atmNetworkTTPName,
                        in PmOamBlockSize pmOamBlockSize)
```

```
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional boolean attribute is used to initiate generation of PM
OAM cells in the forward direction by setting the value to true.
*/
                boolean getPmOamForwardActive
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setPmOamForwardActive
                        (in NameType atmNetworkTTPName,
                        in boolean pmForwardActive )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This optional boolean attribute is used to initiate generation of PM
OAM cells in the backward direction by setting the value to true.
*/
                boolean getPmOamBackwardActive
                        (in NameType atmNetworkTTPName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setPmOamBackwardActive
                        (in NameType atmNetworkTTPName,
                        in boolean pmBackwardActive )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

        }; // interface AtmNetworkTTP

/**
```

5.4.11 AtmRoutingProfile

```
*/
        interface AtmRoutingProfile : NetMgmt::ManagedObject, NetMgmt::Portal
        {
        }; // interface AtmRoutingProfile

/**
```

5.4.12 AtmSNC

```
*/
        interface AtmSNC : NetMgmt::ManagedObject, NetMgmt::Portal
        {
/**
Created using AtmSubnetwork: setupPtToPtSNCWithCTP or
AtmSubnetwork: setupPtToPtSNCWithLinkTP

Deleted using AtmSubnetwork: releaseSNC

An atmSubnetworkConnection represents a connection across a subnetwork.  An
atmSubnetworkConnection is responsible for transporting cells across a
subnetwork.  It is always bidirectional.  An instance of
atmSubnetworkConnection is terminated by atmNetworkCTPs.

An instance of this object is created by the managed system or by an action on
the atmSubnetwork object.  An atmSubnetworkConnection in a composite
subnetwork is made up of a series of atmSubnetworkConnections and
atmLinkConnections.  An atmSubnetworkConnection cannot be created between a
composite subnetwork and one of its component subnetworks.

Supported values for the availabilityStatus are:
- Failed:  The atmSubnetworkConnection cannot function
- Empty SET (none of the availableStatus conditions exist).

The administrativeState is used for administratively locking and unlocking the
atmSubnetworkConnection.  When unlocked, the atmSubnetworkConnection functions
normally.  When in the locked state, the atmSubnetworkConnection is prohibited
from the transport of characteristic information.

For point to point Subnetwork Connections the a-TPInstance and a single entry
```

```
in the z-TPList are used to indicate the endpoints.  Multiple entries in the
z-TPList and the a-TPInstance are used to represent the end points of broadcast
(point-to-multipoint), merge (multipoint-to-point), and composite connections.
The a-TPInstance identifies the primary endpoint.  Only the z-TPList is used
identify all end points in a multipoint-to-multipoint connection (there is no
primary end point).  In this case the a-TPInstance shall be NULL.

Notifications Supported
State Change:  This notification is used to report changes to the State
attributes of this managed entity.  The notification shall identify the state
attribute that changed, its old value, and its new value.

Attribute Value Change: This notification is used to report changes of the
user label.

Managed Entity Creation: This notification is used to report the creation of
an instance of this Managed Entity.

Managed Entity Deletion: This notification is used to report the deletion of
an instance of this Managed Entity.
*/
               AtmSNCAllAttr getAllAttrSNC
                      (in NameType atmSNCName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               CharacteristicInfo getCharacteristicInfo
                      (in NameType atmSNCName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute identifies whether or not the managed
entity is capable of performing its normal functions (Failed or no
unavailability condition existing).
*/
               AvailabilityStatus getAvailabilityStatus
                      (in NameType atmSNCName)
                      raises (NetMgmt::ObjectFailure);

/**
This read/write attribute is used to lock and unlock cell flow
through the subnetwork connection.
*/
               AdministrativeState getAdminState
                      (in NameType atmSNCName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setAdminState
                      (in NameType atmSNCName,
                      in AdministrativeState adminState )
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read/write attribute can be used as an administrative handle (e.g.,
circuit ID) for the end-to-end connection
*/
               string getUserLabel
                      (in NameType atmSNCName)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
               void setUserLabel
                      (in NameType atmSNCName,
                      in string userLabel)
                      raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read/write attribute can identify ownership of a particular connection.
This ownership field can be used for administration specific use such as
customer, organization, department or people names.  This field is useful for
associating multiple connections to a particular customer or organization.
*/
               string getOwnershipName
```

```
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setOwnershipName
                        (in NameType atmSNCName,
                        in string ownershipName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AtmNetworkCTPID getAtpInstance
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure);
                AtmNetworkCTPIDList getZtpList
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure);
                ConnectionType getConnectionType
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure);

/**
This read/write attribute is used to configure the connection as
restorable or non-restorable
*/
                boolean getRestorableIndicator
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setRestorableIndicator
                        (in NameType atmSNCName,
                        in boolean restorableIndicator)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
A composite subnetworkConnection is made of multiple linkConnections
(at least one) and inner subnetworkConnections (at least two)
*/
                AtmSNCIDList getComponentSNCList
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AtmLinkConnIDList getComponentLinkConnList
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read/write attribute indicates whether the route for the
associated subnetworkConnection is specified by the administrator
(manual) or determined by the system (automatic) that may include
managing and managed entities of the subnetwork
*/
                ProvisionType getProvisionType
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure);
                void setProvisionType
                        (in NameType atmSNCName,
                        in ProvisionType provisionType)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
A subnetwork connection may be constrained by a routingProfile
*/
                AtmRoutingProfileID getRoutingProfile
                        (in NameType atmSNCName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setRoutingProfile
                        (in NameType atmSNCName,
                        in AtmRoutingProfileID routingProfile)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This operation determines the path of the SubnetworkConnection and
returns the path at the lowest possible level supported by the
managed system.  For example, at the lowest level of subnetwork
```

```
partitioning.  The connection trace returns the virtual id (VPI for
VP LND connections or VPI/VCI for VC LND connections) for each
atmLink or external interface point (linkTP) of the connection.  For
VC connections, the trace should be examined at both the VC level as
well as the VP Level, if both LNDs are under the purview of the
managed system.  In cases of multipoint connection, the results
should be returned in a breadth first fashion.
*/
              ConnTraceList traceSNC
                     (in NameType atmSNCName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

       }; // interface AtmSNC


/**
```

5.4.13 AtmSubnetwork

```
*/
       interface AtmSubnetwork : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect Creation by the EMS.

Explicit creation by NMS requires makeSubnetwork method on atmLnd.

An atmSubnetwork is a topological component used for carrying characteristic
information(ATM cells within a layer network).  The atmSubnetwork is
delineated by ATM LinkEnds. Subnetworks are used for making subnetwork
connections.  An instance of atmSubnetwork may be specific to the VC or VP
layer and is contained in the appropriate vcLayerNetworkDomain or
vpLayerNetworkDomain.  A point subnetwork does not contain any visible
subnetwork connections.

The atmSubnetwork object provides an abstraction that allows the establishment
and removal of connections across the atmSubnetwork.

characteristicInformation describes the format of the characteristic
information that the resource carries.  This is set to vcCI (I.751) for VC
Layer atmSubnetworks and vpCI (I.751) for VP Layer atmSubnetworks.  The
characteristicInformation, where present, for dependent objects shall match
this attribute.

The userLabel may be used to describe the managing organization.  In cases
where the atmSubnetwork is managed by a different system the inherited
systemTitle may be used.

The supportedByObjectList  points to managed elements that support the
subnetwork.  (specific information about these elements is available through
the M4 NE view).

Supported values for the availabilityStatus are:
- Degraded:  The atmSubnetwork is degraded in some respect.  For instance,
the atmSubnetwork cannot perform the function of establishing new
atmSubnetworkConnections while it can still accept ACTIONS to tear down
existing connections.
- Empty SET (none of the availableStatus conditions exist).

Notifications Supported
Attribute Value Change: This notification is used to report changes of the
user label, and changes to the availability status attribute of this managed
entity.

Managed Entity Creation:  This notification is used to report the creation of
an instance of this Managed Entity.

Managed Entity Deletion:  This notification is used to report the deletion of
an instance of this Managed Entity.
*/
```

```
                AtmSubnetworkAllAttr getAllAttrSubnetwork
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AtmSubnetworkTopo getSubnetworkTopology
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                AtmSubnetworkID getSubnetworkID
                        ()
                        raises (NetMgmt::ObjectFailure);
                        // returns the name of the object
                string getSystemTitle
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

                CharacteristicInfo getCharacteristicInfo
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read-only attribute identifies whether or not the managed
entity is capable of performing its normal functions (Failed,
degraded, or no unavailability condition existing).
*/
                AvailabilityStatus getAvailabilityStatus
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);

/**
This read/write attribute identifies the managing organization.
*/
                string getUserLabel
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setUserLabel
                        (in NameType atmSubnetName,
                        in string userLabel)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
The supportedByObjectList  points to managed elements that support
the subnetwork.
*/
                MOIDList getSupportedByObjectList
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);
                void addSupportedByObjects
                        (in NameType atmSubnetName,
                        in MOIDList supportingObjects)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeSupportedByObjects
                        (in NameType atmSubnetName,
                        in MOIDList supportingObject)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);
                void replaceSupportedByObjectList
                        (in NameType atmSubnetName,
                        in MOIDList supportingObjects)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
A subnetwork contains zero or more subnetwork connections.
*/
                AtmSNCIDList getContainedSNCs
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);

/**
```

```
A subnetwork may be partitioned into one or more subnetworks
*/
                AtmSubnetworkIDList getComponentSubnetworks
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);
                void addComponentSubnetwork
                        (in NameType atmSubnetName,
                        in AtmSubnetworkID componentSubnetwork)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeComponentSubnetwork
                        (in NameType atmSubnetName,
                        in AtmSubnetworkID componentSubnetwork)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);

/**
A composite vcSubnetwork contains vcTopologicalLinks between its
component subnetworks.
*/
                AtmLinkIDList getComponentLinks
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);
                void addComponentLink
                        (in NameType atmSubnetName,
                        in AtmLinkID componentLink)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeComponentLink
                        (in NameType atmSubnetName,
                        in AtmLinkID componentLink)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);

/**
A subnetwork is delineated by zero or more LinkTPs.
*/
                AtmLinkEndIDList getSupportedLinkTPs
                        (in NameType atmSubnetName)
                        raises (NetMgmt::ObjectFailure);
                void addSupportedLinkTP
                        (in NameType atmSubnetName,
                        in AtmLinkEndID supportedLinkTP)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::DuplicateItem);
                void removeSupportedLinkTP
                        (in NameType atmSubnetName,
                        in AtmLinkEndID supportedLinkTP)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::ItemNotFound);

/**
The setupPtToPtSNCWithCTP method sets up a point-to-point
connection between non-connected CTPs in the atmSubnetwork.
*/
                void setupPtToPtSNCWithCTP
                        (in NameType atmSubnetName,
                        in string userLabel,
                        in OptRestorableType restorableType,
                        inout OptAdministrativeState adminState,
                        in AtmNetworkCTPID aNetworkCTP,
                        in boolean aTrailEndPointInd,
                        in AtmTrafficDescID aTozTrafficDescProfile,
                        in AtmNetworkCTPID zNetworkCTP,
                        in boolean zTrailEndPointInd,
                        in AtmTrafficDescID zToaTrafficDescProfile,
                        in AtmRoutingProfileID routingProfile,
                        out AtmSNCID newSNC)
```

```
                    raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                           NetMgmt::InvalidID, NetMgmt::ItemNotFound);

/**
The setupPtToPtSNCWithLinkTP method sets up a point-to-point
connection between LinkEnds in the atmSubnetwork.
*/
              void setupPtToPtSNCWithLinkTP
                    (in NameType atmSubnetName,
                    in string userLabel,
                    in OptRestorableType restorableType,
                    inout OptAdministrativeState adminState,
                    in AtmLinkEndID aLinkTP,
                    inout VirtualID aVirtualID,
                    in boolean aTrailEndPointInd,
                    in AtmTrafficDescID aTozTrafficDescProfile,
                    in AtmLinkEndID zLinkTP,
                    inout VirtualID zVirtualID,
                    in boolean zTrailEndPointInd,
                    in AtmTrafficDescID zToaTrafficDescProfile,
                    in AtmRoutingProfileID routingProfile,
                    out AtmSNCID newSNC,
                    out AtmNetworkCTPID aNetworkCTP,
                    out AtmNetworkCTPID zNetworkCTP)
                    raises (NetMgmt::ObjectFailure,
                           NetMgmt::NotSupported,
                           NetMgmt::InvalidID,
                           NetMgmt::ItemNotFound);

/**
The setupPtToMultiSNCWithCTP method sets up a composite point-to-
multipoint connection between non-connected CTPs in the atmSubnetwork.
*/
              void setupPtToMultiSNCWithCTP
                    (in NameType atmSubnetName,
                    in string userLabel,
                    in OptRestorableType restorableType,
                    inout OptAdministrativeState adminState,
                    in AtmNetworkCTPID aNetworkCTP,
                    in boolean aTrailEndPointInd,
                    in AtmTrafficDescID aTozTrafficDescProfile,
                    in AtmTrafficDescID aIngressTrafficDescProfile,
                    in ZtpCompositeCtpList ztpCompositeCtpList,
                    in AtmRoutingProfileID routingProfile,
                    out AtmSNCID newSNC)
                    raises (NetMgmt::ObjectFailure,
                           NetMgmt::NotSupported,
                           NetMgmt::InvalidID,
                           NetMgmt::ItemNotFound);

/**
The setupPtToMultiSNCWithLinkTP method sets up a composite point-to-
multipoint connection between LinkEnds in the atmSubnetwork.
*/
              void setupPtToMultiSNCWithLinkTP
                    (in NameType atmSubnetName,
                    in string userLabel,
                    in OptRestorableType restorableType,
                    inout OptAdministrativeState adminState,
                    in AtmLinkEndID aLinkTP,
                    inout VirtualID aVirtualID,
                    in boolean aTrailEndPointInd,
                    in AtmTrafficDescID aTozTrafficDescProfile,
                    in AtmTrafficDescID aIngressTrafficDescProfile,
                    inout ZtpCompositeLinkEndList ztpCompositeLinkEndList,
                    in AtmRoutingProfileID routingProfile,
                    out AtmSNCID newSNC,
                    out AtmNetworkCTPID aNetworkCTP,
```

```
                        out AtmNetworkCTPID zNetworkCTP )
                        raises (NetMgmt::ObjectFailure,
                               NetMgmt::NotSupported,
                               NetMgmt::InvalidID,
                               NetMgmt::ItemNotFound);

/**
The addTpToMultiSNCWithCTP method adds an endpoint to a composite
point-to-multipoint connection in the atmSubnetwork.
*/
                void addTpToMultiSNCWithCTP
                        (in NameType atmSubnetName,
                        in AtmSNCID modifiedSNC,
                        in AtmTrafficDescID aTozTrafficDescProfile,
                        in ZtpCompositeCtp ztpCompositeCtp,
                        in AtmRoutingProfileID routingProfile)
                        raises (NetMgmt::ObjectFailure,
                               NetMgmt::NotSupported,
                               NetMgmt::InvalidID,
                               NetMgmt::ItemNotFound);

/**
The addTpToMultiSNCWithLinkTP method adds an endpoint to a
composite point-to-multipoint connection in the atmSubnetwork.
*/
                void addTpToMultiSNCWithLinkTP
                        (in NameType atmSubnetName,
                        in AtmSNCID modifiedSNC,
                        in AtmTrafficDescID aTozTrafficDescProfile,
                        inout ZtpCompositeLinkEnd ztpCompositeLinkEnd,
                        in AtmRoutingProfileID routingProfile,
                        out AtmNetworkCTPID zNetworkCTP)
                        raises (NetMgmt::ObjectFailure,
                               NetMgmt::NotSupported,
                               NetMgmt::InvalidID,
                               NetMgmt::ItemNotFound);

/**
The removeTpFromMultiSNC method removes a TP from a composite
point-to-multipoint connection in the atmSubnetwork.
*/
                void removeTpFromMultiSNC
                        (in NameType atmSubnetName,
                        in AtmSNCID modifiedSNC,
                        in AtmNetworkCTPID ztpRemoved )
                        raises (NetMgmt::ObjectFailure,
                               NetMgmt::NotSupported,
                               NetMgmt::InvalidID,
                               NetMgmt::ItemNotFound);

/**
The releaseSNC method releases a point-to-point or a multipoint connection
between CTPs in the atmSubnetwork.
*/
                void releaseSNC
                        (in NameType atmSubnetName,
                        in AtmSNCID connectionID )
                        raises (NetMgmt::ObjectFailure,
                               NetMgmt::NotSupported,
                               NetMgmt::InvalidID,
                               NetMgmt::ItemNotFound);

        }; // interface AtmSubnetwork
```

```
/**
```

5.4.14 AtmNetworkAccessProfile

```
*/
       interface AtmNetworkAccessProfile : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect Creation by the EMS or by atmLND interface.

An atmNetworkAccessProfile contains information that describe the maximum
ingress and egress bandwidth, along with the range of VPI or VCI values that
are applied to the atmLink or atmLinkEnd object instances that point to it.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
*/
              AtmNetworkAccessProfileAllAttr getAllAttrNetworkAccessProfile
                     (in NameType atmNetworkAccessProfileName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
This read/write attribute identifies the total aggregate ingress
bandwidth for a link or a linkTP (linkEnd or logicalLinkTP).
*/
              long getTotalIngressBW
                     (in NameType atmNetworkAccessProfileName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              void setTotalIngressBW
                     (in NameType atmNetworkAccessProfileName,
                     in long totalIngressBW)
                     raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                            NetMgmt::NotSupported);

/**
This read/write attribute identifies the total aggregate egress
bandwidth for a link or a linkTP (linkEnd or logicalLinkTP).
*/
              long getTotalEgressBW
                     (in NameType atmNetworkAccessProfileName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              void setTotalEgressBW
                     (in NameType atmNetworkAccessProfileName,
                     in long totalEgressBW)
                     raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                            NetMgmt::NotSupported);

/**
This read/write attribute identifies the maximum number of
concurrently active VC connections that a link or a linkTP (linkEnd
or logicalLinkTP) may support.
*/
              long getMaxNumActiveVcConn
                     (in NameType atmNetworkAccessProfileName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
              void setMaxNumActiveVcConn
                     (in NameType atmNetworkAccessProfileName,
                     in long maxNumActiveVcConn)
                     raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                            NetMgmt::NotSupported);

/**
This read/write attribute identifies the maximum number of
concurrently active VP connections that a link or a linkTP (linkEnd
or logicalLinkTP) may support.
*/
              long getMaxNumActiveVpConn
                     (in NameType atmNetworkAccessProfileName)
```

```
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
                void setMaxNumActiveVpConn
                        (in NameType atmNetworkAccessProfileName,
                        in long maxNumActiveVpConn)
                        raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                                NetMgmt::NotSupported);

/**
This read/write attribute describes the virtual ID range (VPIs) that
may be used for Connections associated with a link or LinkTP.
*/
                VpiOrVciRange getVpiRange
                        (in NameType atmNetworkAccessProfileName)
                        raises (NetMgmt::ObjectFailure);
                void setVpiRange
                        (in NameType atmNetworkAccessProfileName,
                        in VpiOrVciRange vpiRange)
                        raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                                NetMgmt::NotSupported);

/**
This read/write attribute describes the virtual ID range (VCIs) that
may be used for Connections associated with a link or LinkTP.
*/
                VpiOrVciRange getVciRange
                        (in NameType atmNetworkAccessProfileName)
                        raises (NetMgmt::ObjectFailure);
                void setVciRange
                        (in NameType atmNetworkAccessProfileName,
                        in VpiOrVciRange vciRange)
                        raises (NetMgmt::ObjectFailure, NetMgmt::OutOfRange,
                                NetMgmt::NotSupported);

        }; // interface AtmNetworkAccessProfile


        interface AtmNetworkAccessProfileFactory : NetMgmt::ManagedObject
        {
/**
Expect Creation by the EMS.

Used to create instances of ATM Network Access Profile.
*/

                AtmNetworkAccessProfileFactoryID
                        getAtmNetworkAccessProfileFactoryID
                        ()
                        raises (NetMgmt::ObjectFailure);
                        // returns the name of the object

/**
Create ATM Network Access Profile.
*/
                AtmNetworkAccessProfileID makeAtmNetworkAccessProfile
                        (in NameType networkAccessProfileName,
                        in long totalIngressBW,
                        in long totalEgressBW,
                        in long maxNumActiveVcConn,
                        in long maxNumActiveVpConn,
                        in VpiOrVciRange vpiRange,
                        in VpiOrVciRange vciRange )
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::InvalidID);

        }; // interface AtmNetworkAccessProfileFactory
```

```
/**
```

5.4.15 AtmTraficDesc

```
*/
       interface AtmTrafficDesc : NetMgmt::ManagedObject, NetMgmt::Portal
       {
/**
Expect Creation by the EMS or by Traffic Desc factory interface.

This interface supports ATM Forum TM 4.1.

The interface AtmTrafficDesc is uninstantiable.

An atmTrafficDescriptor provides a superclass for the category specific
traffic parameter subclasses.

Notifications Supported
Managed Entity Creation: This notification is used to report the creation of
an instance of this managed entity.

If profile name is provided,
*/
              string getProfileName
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              ServiceCategory getServiceCategory
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

              ConformanceDefinition getConformanceDefinition
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

       }; // interface AtmTrafficDesc
```

```
       interface AtmTrafficDescABR : AtmTrafficDesc
       {
/**
Expect Creation by the EMS or by Traffic Desc factory interface.
This interface supports ATM Forum TM 4.1.

An atmTrafficDescriptor contains information that describes the ingress and
egress Traffic descriptors for the ABR category.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
*/

              AtmTrafficDescABRAllAttr getAllAttrABR
                     (in NameType atmABRTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getPeakCellRate
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

// if policing is performed
              long getCDVTolerancePCR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getMinCellRate
                     (in NameType atmTrafficDescName)  // ABR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getInitialCellRate
                     (in NameType atmTrafficDescName)  // ABR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getTransientBufferExposure
                     (in NameType atmTrafficDescName)  // ABR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              RateChangeFactor getRateDecreaseFactor
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              RateChangeFactor getRateIncreaseFactor
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getFixedRoundTripTime
                     (in NameType atmTrafficDescName)  // ABR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              ABRNrm getABRNrm
                     (in NameType atmTrafficDescName)  // ABR-optional
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              ABRTrm getABRTrm
                     (in NameType atmTrafficDescName)  // ABR-optional
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              ABRCDF getABRCDF
                     (in NameType atmTrafficDescName)  // ABR-optional
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getABRADTF
                     (in NameType atmTrafficDescName)  // ABR-optional
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
```

```
       }; // interface AtmTrafficDescABR



       interface AtmTrafficDescCBR : AtmTrafficDesc
       {

/**
Expect Creation by the EMS or by Traffic Desc factory interface
This interface supports ATM Forum TM 4.1

An atmTrafficDescriptor contains information that describes the ingress and
egress Traffic descriptors for the CBR category.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
*/

              AtmTrafficDescCBRAllAttr getAllAttrCBR
                     (in NameType atmCBRTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getPeakCellRate
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

              long getCLR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

// if policing is performed
              long getCDVTolerancePCR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

       }; // interface AtmTrafficDescCBR
```

```
       interface AtmTrafficDescVBR : AtmTrafficDesc
       {
/**
Expect Creation by the EMS or by Traffic Desc factory interface.
This interface supports ATM Forum TM 4.1.

An atmTrafficDescriptor contains information that describes the ingress and
egress Traffic descriptors for the VBR category.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
*/

              AtmTrafficDescVBRAllAttr getAllAttrVBR
                     (in NameType atmVBRTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getPeakCellRate
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

              long getCLR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getSustainableCellRate
                     (in NameType atmTrafficDescName) // VBR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getMaxBurstSize
                     (in NameType atmTrafficDescName)  // VBR
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
if policing is performed
*/
              long getCDVTolerancePCR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

/**
if policing is performed and If I.371 is supported
*/
              long getCDVToleranceSCR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

       }; // interface AtmTrafficDescVBR
```

```
       interface AtmTrafficDescUBR : AtmTrafficDesc
       {
/**
Expect Creation by the EMS or by Traffic Desc factory interface.
This interface supports ATM Forum TM 4.1.

An atmTrafficDescriptor contains information that describes the ingress and
egress Traffic descriptors for the UBR category.

Notifications Supported
Managed Entity Creation:  This notification is used to report the creation of
an instance of this managed entity.
*/

              AtmTrafficDescUBRAllAttr getAllAttrUBR
                     (in NameType atmUBRTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getPeakCellRate
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure);

// if policing is performed
              long getCDVTolerancePCR
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

       }; // interface AtmTrafficDescUBR


       interface AtmTrafficDescGFR : AtmTrafficDesc
       {
/**
Expect creation by the EMS or by TrafficDescFactory interface.
This interface supports ATM Forum TM 4.1.

An atmTrafficDescriptor contains information that describes the ingress and egress traffic
descriptors for the GFR category.

Notifications Supported
Managed Entity Creation: This notification is used to report the creation of an instance of
this managed entity.
*/
              AtmTrafficDescGFRAllAttr getAllAttrGFR
                     (in NameType atmGFR1TrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getPeakCellRate
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getCDVTolerancePCR            //if policing is performed
                     (in NameType atmTrafficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getMaxFrameSize
                     (in NameType atmTraficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getMinCellRate
                     (in NameType atmTraficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

              long getMaxBurstSize
                     (in NameType atmTraficDescName)
                     raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);
```

```
};  // interface AtmTrafficDescGFR
```

```
       interface AtmTrafficDescFactory : NetMgmt::ManagedObject
       {
/**
Expect Creation by the EMS.

Used to create instances of ATM Traffic Desc.
*/

              AtmTrafficDescFactoryID getTrafficDescFactoryID
                     ()
                     raises (NetMgmt::ObjectFailure);
                     // returns the name of the object

/**
Create UBR ATM Traffic Descriptor.
*/
              AtmTrafficDescID makeUBRAtmTrafficDesc
                     (in MOID containerObject,
                     in string profileName,
                     in ServiceCategory serviceCategory,
                     in ConformanceDefinition conformanceDefinition,
                     in long peakCellRate,
                     in long cDVTolerancePCR )
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::OutOfRange,
                            NetMgmt::ItemNotFound);

/**
Create CBR ATM Traffic Descriptor.
*/
              AtmTrafficDescID makeCBRAtmTrafficDesc
                     (in MOID containerObject,
                     in string profileName,
                     in ServiceCategory serviceCategory,
                     in ConformanceDefinition conformanceDefinition,
                     in long peakCellRate,
                     in long cDVTolerancePCR,
                     in long cLR)
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::OutOfRange,
                            NetMgmt::ItemNotFound);

/**
Create VBR ATM Traffic Descriptor.
*/
              AtmTrafficDescID makeVBRAtmTrafficDesc
                     (in MOID containerObject,
                     in string profileName,
                     in ServiceCategory serviceCategory,
                     in ConformanceDefinition conformanceDefinition,
                     in long peakCellRate,
                     in long cDVTolerancePCR,
                     in long cDVToleranceSCR,
                            // negative if I.371 not supported
                     in long cLR,
                     in long sustainableCellRate,
                     in long maxBurstSize )
                     raises (NetMgmt::ObjectFailure,
                            NetMgmt::NotSupported,
                            NetMgmt::OutOfRange,
                            NetMgmt::ItemNotFound);

/**
Create ABR ATM Traffic Descriptor.
*/
```

```
                AtmTrafficDescID makeABRAtmTrafficDesc
                        (in MOID containerObject,
                        in string profileName,
                        in ServiceCategory serviceCategory,
                        in ConformanceDefinition conformanceDefinition,
                        in long peakCellRate,
                        in long cDVTolerancePCR,
                        in long minCellRate,
                        in long initialCellRate,
                        in long transientBufferExposure,
                        in RateChangeFactor rateDecreaseFactor,
                        in RateChangeFactor rateIncreaseFactor,
                        in long fixedRoundTripTime,
                        in ABRNrm aBRNrm,
                        in ABRTrm aBRTrm,
                        in ABRCDF aBRCDF,
                        in long aBRADTF )  // ZERO if not supported
                        raises (NetMgmt::ObjectFailure,
                                NetMgmt::NotSupported,
                                NetMgmt::OutOfRange,
                                NetMgmt::ItemNotFound);

/**
Create GFR ATM Traffic Descriptor.
*/
                AtmTrafficDescID makeGBRAtmTrafficDesc
                        (in MOID containerObject,
                        in string profileName,
                        in ServiceCategory serviceCategory,
                        in ConformanceDefinition conformanceDefinition,
                        in long peakCellRate,
                        in long cDVTolerancePCR,
                        in long maxFrameSize,
                        in long minCellRate,  //for CLP=0
                        in long maxBurstSize,
                        in GFR1or2 gFR1or2)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                                NetMgmt::OutOfRange, NetMgmt::ItemNotFound);

        }; // interface AtmTrafficDescFactory

/**
```

5.4.16 Latest Occurrence Log

```
*/
        interface LatestOccurrenceLog : NetMgmt::ManagedObject
        {
                void getAdminState
                        (out AdministrativeState adminState)
                        raises (NetMgmt::ObjectFailure);

                void setAdminState
                        (in AdministrativeState adminState )
                        raises (NetMgmt::ObjectFailure);

                void getLinkEndEntries
                        (in AtmLinkEndID linkEndID,
                        out LatestOccLogList latestOccLogList)
                        raises (NetMgmt::ObjectFailure, NetMgmt::ItemNotFound);
        }; // interface LatestOccurrenceLog

/**
```

5.4.17 Network

```
*/
        interface Network : NetMgmt::ManagedObject
        {
/**
Network can be used to contain other objects.
```

```
Expect creation by the EMS.
Deletion of Network by a client is not allowed.
*/
       }; // interface Network



/**
```

5.5 Module ATMF_M4NW_PM

```
This module defines the interfaces for supporting Performance Management (PM) for ATM.
Further discussion of PM-related managed entities and attributes is provided in Sections
3.17, 3.18, 3.26, 3.26. 3.41, 3.42, 3.90, 3.91 and 3.92 of Reference [2].  All of the
performance management aspects for module atmf_m4nw are grouped into this subordinate module,
atmf_m4nw_pm.   Module atmf_m4nw_pm draws upon the technology independent performance
management module corba_pm and adds extensions that are specific to ATM.
*/

#ifndef _atmf_m4nw_pm_idl_
#define _atmf_m4nw_pm_idl_

#include "corba_pm.idl"

module atmf_m4nw_pm
{
const string moduleName = "atmf_m4nw_pm";

/**
IMPORTS

Types imported from corba_pm
*/
       typedef corba_pm::PerfParameter            PerfParameter;
       typedef corba_pm::PerfDataSet              PerfDataSet;

/**
Interfaces imported from corba_pm are CurrentData, CurrentDataFactory,
CurrentPMBulkDataIterator, HistoryData, HistoryPMBulkDataIterator, ThresholdData,
ThresholdBulkDataIterator, PMBulkOperations, and PerformanceDataFileGenerator.


FORWARD DECLARATIONS
*/
       interface CellProtocolMonCurrentData;
       interface AtmTrafficLoadCurrentData;
       interface CongDiscardCurrentData;
       interface TcAdaptProtMonCurrentData;
       interface UpcNpcDisagreementsCurrentData;
       interface PmOamCurrentData;
       interface AtmCurrentDataFactory;
       interface AtmPMBulkOperations;
       interface AtmCurrentPMBulkDataIterator;
       interface CellProtocolMonHistoryData;
       interface AtmTrafficLoadHistoryData;
       interface CongDiscardHistoryData;
       interface TcAdaptProtMonHistoryData;
       interface UpcNpcDisagreementsHistoryData;
       interface PmOamHistoryData;
       interface AtmHistoryPMBulkDataIterator;
       interface AtmThresholdData;
       interface AtmThresholdPMBulkDataIterator;
       interface AtmPerformanceDataFileGenerator;

/**
Type Definitions
```

```
ATM Performance Data Set (PerfDataSet) Values
*/
       const short cellProtMon = 1;        // Cell Protocol Monitoring
       const short trafLoad = 2;                  // Traffic Load
       const short congDiscard = 3;        // Congestion Discards
       const short tcAdaptProtMon = 4;            // TC Adaptor Protocol Monitoring
       const short UpcNpcDisagreements = 5;       // UPC and NPC Disagreements
       const short PmOam = 6;                     // PM OAM Cell Monitoring

/**
ATM Performance Counter (PerfParameter) Values

cellProtMon counters
*/
       const short numberDiscCellsProtErr = 1;
         // Count of cells discarded due to protocol errors; thresholded count

       const short numberRecvOAMCells = 2;
         // Count of the number of received OAM Cells

/**
trafLoad counters
*/
       const short numberCellsRecvd = 3;                 // Number of cells received

       const short numberCellsTrnsd = 4;                 // Number of cells transmitted

/**
congDiscard counters
*/
       const short numberCellsDiscCong = 5;
         // Count of cells discarded due to congestion; thresholded count

       const short numberCLP0CellsDisc = 6;
         // Count of high prioritycells discarded due to congestion; thresholded count

/**
tcAdaptProtMon counters
*/
       const short numberDiscCellsHECViolat = 7;
         // Count of cells discarded due to HEC Violations; thresholded count

/**
UpcNpcDisagreements counters
*/
       const short numberDiscardUpcNpcCells = 8;
         // Number of cells discarded due to policing (UPC/NPC); thresholded count

       const short numberSuccessfullyPassedUpcNpcCells = 9;
         // Number of cells marked by UPC/NPC that are passed

       const short numberDiscardCLP0UpcNpcCells = 10;
         // Number of CLP=0 cells   discarded due to policing; thresholded count

       const short numberSuccessfullyPassedCLP0UpcNpcCells = 11;
         // Number of CLP=0 cells marked by UPC/NPC that are passed

/**
PmOam counters
*/
       const short numberPmOamLostCells = 12;     // Lost cells measured by PM OAM

       const short numberPmOamMisinsertedCells = 13;
         // Misinserted cells measured by PM OAM

       const short numberPmOamUserCells = 14;     // User cells measured by PM OAM

       const short numberPmOamFarEndLostCells = 15;
```

```
        // Far-End Lost cells measured by PM OAM

      const short numberPmOamFarEndMisinsertedCells = 16;
        // Far End Misinserted cells measured by PM OAM

      const short numberPmOamFarEndUserCells = 17;
        // Far End User cells measured by PM OAM



/**
Interfaces and Methods

All of the interfaces in this submodule use only the methods inherited from the
interfaces imported from module corba_pm.  Such methods are used together with
the ATM-specific values of PerfDataSet and PerfParameter, which were defined earlier
in this module.  These interfaces do not define any additional methods.

CURRENT DATA INTERFACES
*/
      interface CellProtocolMonCurrentData : corba_pm::CurrentData
      {
/**
Retrieves attributes or current counter values for counters within the cellProtMon grouping,
namely, numberDiscCellsProtErr and numberRecvOAMCells, as indicated by the
appropriate values of PerfDataSet and PerfParameter.  Additional inherited methods are
provide for the setting of AdministrativeState, associating threshold data with a
current data instance, resetting counters, activating or deactivating the suppression
of counters having all-zero counts, and activating or deactivating history retention.

A Threshold Crossing Alert is used to notify the management system when any of the
performance parameters exceeds a pre-set threshold described in the associated
ThresholdData object.  See the CurrentData interface for a description of the
information supplied in this Threshold Crossing Alert.  Similar considerations apply
to the following interfaces.

The containment relationship between an AtmLinkEnd and its associated
CellProtocolMonCurrentData object can be represented in the CORBA Naming
Service, or it may be expressed in the ID of the name component (as
suggested in Appendix D), or it may be found using methods inherited from
interface Portal.  Similar considerations apply to the following interfaces.
*/
      }; // interface CellProtocolMonCurrentData

      interface AtmTrafficLoadCurrentData : corba_pm::CurrentData
      {
/**
Retrieves attributes or counter values for counters within the trafLoad grouping,
namely, numberCellsRecvd and numberCellsTrnsd, as indicated by the appropriate values of
PerfDataSet and PerfParameter.  Additional inherited methods are provide for supporting
operations as described earlier for CellProtocolMonCurrentData.

The containment relationship between an AtmLinkEnd or AtmNetworkCTP and the associated
AtmTrafficLoadCurrentData object can be represented in the CORBA Naming Service, or it may be
expressed in the ID of the name component (as suggested in Appendix D), or it may be found
using methods inherited from interface Portal.
*/
      }; // interface AtmTrafficLoadCurrentData

      interface CongDiscardCurrentData : corba_pm::CurrentData
      {
/**
Retrieves attributes or current counter values for counters within the congDiscard grouping,
namely, numberCellsDiscCong and numberCLP0CellsDisc, as indicated by the
appropriate values of PerfDataSet and PerfParameter.

The containment relationship between an AtmLinkEnd and its associated CongDiscardCurrentData
object can be represented in the CORBA Naming Service, or it may be expressed in the ID of
```

```
the name component (as suggested in Appendix D), or it may be found using methods inherited
from interface Portal.
*/
       }; // interface CongDiscardCurrentData

       interface TcAdaptProtMonCurrentData : corba_pm::CurrentData
       {
/**
Retrieves attributes or current counter values for counters within the tcAdaptProtMon
grouping, namely, numberDiscCellsHECViolat, as indicated by the appropriate values of
PerfDataSet and PerfParameter.  Additional inherited methods are provide for supporting
operations as described earlier for CellProtocolMonCurrentData.

The containment relationship between an AtmLinkEnd and its associated
TcAdaptProtMonCurrentData object can be represented in the CORBA Naming Service, or it may be
expressed in the ID of the name component (as suggested in Appendix D), or it may be found
using methods inherited from interface Portal.
*/
       }; // interface TcAdaptProtMonCurrentData

       interface UpcNpcDisagreementsCurrentData : corba_pm::CurrentData
       {
/**
Retrieves attributes or current counter values for counters within the UpcNpcDisagreements
grouping, namely, numberCellsDiscCong, numberSuccessfullyPassedUpcNpcCells,
numberDiscardCLP0UpcNpcCells, and numberSuccessfullyPassedCLP0UpcNpcCells, as indicated by
the appropriate values of PerfDataSet and PerfParameter.  Additional inherited methods are
provide for supporting operations as described earlier for CellProtocolMonCurrentData.

The containment relationship between an AtmLinkEnd and its associated
UpcNpcDisagreementsCurrentData object can be represented in the CORBA Naming Service, or it
may be expressed in the ID of the name component (as suggested in Appendix D), or it may be
found using methods inherited from interface Portal.
*/
       }; // interface UpcNpcDisagreementsCurrentData

       interface PmOamCurrentData : CORBA_PM::CurrentData
       {
/**
Retrieves attributes or current counter values for counters within the PmOam grouping,
namely, numberPmOamLostCells, numberPmOamMisinsertedCells, numberPmOamUserCells,
numberPmOamFarEndLostCells, numberPmOamFarEndMisinsertedCells and numberPmOamFarEndUserCells,
as indicated by the appropriate values of PerfDataSet and PerfParameter.  Additional
inherited methods are provide for supporting operations as described earlier for
CellProtocolMonCurrentData.

The containment relationship between an AtmLinkEnd and its associated PmOamCurrentData object
can be represented in the CORBA Naming Service, or it may be expressed in the ID of the name
component (as suggested in Appendix D), or it may be found using methods inherited from
interface Portal.
*/
       }; // interface PmOamCurrentData

       interface AtmCurrentDataFactory : corba_pm::CurrentDataFactory
       {
/**
Expect creation by EMS.  Used to create specific instances of interfaces inheriting from
corba_pm::CurrentData.
*/
       }; // interface AtmCurrentDataFactory

       interface AtmPMBulkOperations : corba_pm::PMBulkOperations
       {
/**
Provides a method for retrieving in bulk all current data of the types indicated by
the appropriate values of PerfDataSet and PerfParameter, and having a containment
```

```
relation with the objects identified by MOIDList.  For specific examples of such performance
parameter types and their containment relationships, see previous comments under CURRENT DATA
INTERFACES.

Under this same interface, a second method is provided for retrieving in bulk all history
data of the types indicated by the appropriate values of PerfDataSet and PerfParameter, and
having a containment relation with the objects identified by MOIDList.  For specific examples
of such performance parameter types and their containment relationships, see following
comments under HISTORY DATA INTERFACES.

Under this same interface, additional methods are provided that provide for the bulk
retrieval of ThresholdDataIDs, for the retrieval of PerfThresholdDataList (a list of
threshold instances complete with threshold data values), and for the bulk setting of
ThresholdData values.
*/
       }; // interface AtmPMBulkOperations

       interface AtmCurrentPMBulkDataIterator : corba_pm::CurrentPMBulkDataIterator
       {
/**
Used to support the method under AtmPMBulkOperations for the bulk retrieval of
current data.
       }; // interface AtmCurrentPMBulkDataIterator

/**
HISTORY DATA INTERFACES
/*

       interface CellProtocolMonHistoryData : corba_pm::HistoryData
       {
/**
Retrieves attributes or counter values for NumberDiscCellsProtErr and NumberRecvOAMCells, as
indicated by the appropriate values of PerfParameter.
Contains counter values for counters within the cellProtMon grouping, namely,
numberDiscCellsProtErr and numberRecvOAMCells, as indicated by the appropriate values of
PerfDataSet and PerfParameter.  The relevant time period is defined by PeriodStartTime and
PeriodEndTime.

The containment relationship between an AtmLinkEnd and its associated
CellProtocolMonHistoryData object can be represented in the CORBA Naming
Service, or it may be expressed in the ID of the name component (as suggested in Appendix D),
or it may be found using methods inherited from interface Portal.  Similar considerations
apply to the following interfaces.
*/
       }; // interface CellProtocolMonHistoryData

       interface AtmTrafficLoadHistoryData : corba_pm::HistoryData
       {
/**
Retrieves attributes or counter values for counters within the trafLoad grouping, namely,
numberCellsRecvd and numberCellsTrnsd, as indicated by the appropriate values of PerfDataSet
and PerfParameter.
The relevant time period is defined by PeriodStartTime and PeriodEndTime.

The containment relationship between an AtmLinkEnd or AtmNetworkCTP and the associated
AtmTrafficLoadHistoryData object can be represented in the CORBA Naming Service, or it may be
expressed in the ID of the name component (as suggested in Appendix D), or it may be found
using methods inherited from interface Portal.
*/
       }; // interface AtmTrafficLoadHistoryData

       interface CongDiscardHistoryData : corba_pm::HistoryData
       {
/*
Retrieves attributes or counter values for counters within the congDiscard grouping,
namely, numberCellsDiscCong and numberCLP0CellsDisc, as indicated by the appropriate values
of PerfDataSet and PerfParameter.
The relevant time period is defined by PeriodStartTime and PeriodEndTime.
```

```
The containment relationship between an AtmLinkEnd and its associated CongDiscardHistoryData
object can be represented in the CORBA Naming Service, or it may be expressed in the ID of
the name component (as suggested in Appendix D), or it may be found using methods inherited
from interface Portal.
*/
       }; // interface CongDiscardHistoryData

       interface TcAdaptProtMonHistoryData : corba_pm::HistoryData
       {
/**
Retrieves attributes or counter values for counters within the tcAdaptProtMon grouping,
namely, numberDiscCellsHECViolat, as indicated by the appropriate values of PerfDataSet and
PerfParameter.
The relevant time period is defined by PeriodStartTime and PeriodEndTime.

The containment relationship between an AtmLinkEnd and its associated
TcAdaptProtMonHistoryData object can be represented in the CORBA Naming Service, or it may be
expressed in the ID of the name component (as suggested in Appendix D), or it may be found
using methods inherited from interface Portal.
*/
       }; // interface TcAdaptProtMonHistoryData

       interface UpcNpcDisagreementsHistoryData : corba_pm::HistoryData
       {
/**
Retrieves attributes or counter values for counters within the UpcNpcDisagreements grouping,
namely, numberCellsDiscCong, numberSuccessfullyPassedUpcNpcCells,
numberDiscardCLP0UpcNpcCells, and numberSuccessfullyPassedCLP0UpcNpcCells,
as indicated by the appropriate values of PerfDataSet and PerfParameter.
The relevant time period is defined by PeriodStartTime and PeriodEndTime.

The containment relationship between an AtmLinkEnd and its associated
UpcNpcDisagreementsHistoryData object can be represented in the CORBA Naming
Service, or it may be expressed in the ID of the name component (as suggested in Appendix D),
or it may be found using methods inherited from interface Portal.
*/
       }; // interface UpcNpcDisagreementsHistoryData

       interface PmOamHistoryData : corba_pm::HistoryData
       {
/**
Retrieves attributes or counter values for counters within the PmOam grouping, namely,
numberPmOamLostCells, numberPmOamMisinsertedCells, numberPmOamUserCells,
numberPmOamFarEndLostCells, numberPmOamFarEndMisinsertedCells and numberPmOamFarEndUserCells,
as indicated by the appropriate values of PerfDataSet and PerfParameter.
The relevant time period is defined by PeriodStartTime and PeriodEndTime.

The containment relationship between an AtmLinkEnd and its associated PmOamHistoryData object
can be represented in the CORBA Naming Service, or it may be expressed in the ID of the name
component (as suggested in Appendix D), or it may be found using methods inherited from
interface Portal.
*/
       }; // interface PmOamHistoryData

/**
Note that the interface AtmCurrentDataFactory includes provision for generating a
HistoryDataIDList at the same time that the corresponding CurrentDataIDList is generated.
Therefore no separate interface for a HistoryDataFactory is provided.
*/

       interface AtmHistoryPMBulkDataIterator : corba_pm::HistoryPMBulkDataIterator
       {
/**
Used to support the method under AtmPMBulkOperations for the bulk retrieval of history
data.  A previous comment under CURRENT DATA has already discussed this method.
*/
       }; // interface AtmHistoryPMBulkDataIterator
```

```
/**
ADDITIONAL INTERFACES
*/

       interface AtmThresholdData : corba_pm::ThresholdData
       {
/**
Provides methods for retrieving and for setting ThresholdData values.
*/
       }; // interface AtmThresholdData

       interface AtmThresholdPMBulkDataIterator : corba_pm::ThresholdPMBulkDataIterator
       {
/**
Used to support methods under AtmPMBulkOperations for the bulk retrieval and the bulk setting
of ThresholdData.  A previous comment under CURRENT DATA has already discussed these methods.
*/
       }; // interface AtmThresholdPMBulkDataIterator

       interface AtmPerformanceDataFileGenerator :
              corba_pm::PerformanceDataFileGenerator
       {
/**
Provides a method for initiating the generation of a bulk file at the server.  The name of
this bulk file is returned to the client, while the bulk file itself can be returned by other
means that may not directly use a CORBA interface.
*/
       }; // interface AtmPerformanceDataFileGenerator

}; // end of module atmf_m4nw_pm
#endif // _atmf_m4nw_pm_idl_

}; // end of module atmf_m4nw
#endif // _atmf_m4nw_idl_
```

```
/**
```

5.6 Module CORBA_PM

```
This is a peer module to the module atmf_m4nw.  It provides technology independent interfaces
for Performance Management (PM).

This IDL code is intended to be stored in a file named "corba_pm.idl" located in the search
path of your IDL compiler.
*/

#ifndef _corba_pm_idl_
#define _corba_pm_idl_
#include "NetMgmt.idl"

module corba_pm
{
const string moduleName = "corba_pm";

/**
IMPORTS

Types imported from NetMgmt
*/
       typedef NetMgmt::MOID                    MOID;
       typedef NetMgmt::MOIDList                MOIDList;
       typedef NetMgmt::Name                    Name;
       typedef NetMgmt::AdministrativeState     AdministrativeState;
       typedef NetMgmt::OperationalState        OperationalState;
       typedef NetMgmt::UID                     UID;
       typedef NetMgmt::GeneralizedTime         GeneralizedTime;
       typedef NetMgmt::ObjectClass          ObjectClass;

/**
Exceptions imported from NetMgmt are DuplicateItem, Duplicatename, ItemNotFound, InvalidID,
NotSupported, ObjectFailure, and MaxMonitorPointsExceeded.

The MaxMonitorPointsExceeded exception indicates that the request could not be serviced
because it exceeded the server's capacity in terms of the maximum number of points that
the server (or its underlying NEs) can simultaneously monitor.

Interfaces imported from NetMgmt are ManagedObject, ManagedObjectFactory, Portal, and NameIterator.


FORWARD DECLARATIONS
*/
       interface CurrentData;
       interface CurrentDataFactory;
       interface CurrentPMBulkDataIterator;
       interface HistoryData;
       interface HistoryPMBulkDataIterator;
       interface ThresholdData;
       interface ThresholdPMBulkDataIterator;
       interface PMBulkOperations;
       interface PerformanceDataFileGenerator;

/**
Type Definitions
*/
       typedef unsigned long          CounterValue;      // Actual counter value
       typedef unsigned long          ThresholdValue;    // Value of threshold
       typedef GeneralizedTime       ElapsedTime; // Elapsed time of an interval
       typedef GeneralizedTime       PeriodStartTime;     // Start time of an interval
       typedef GeneralizedTime       PeriodEndTime;       // Ending time of an interval
       typedef short                 NumIntervals;    // Number of intervals

/**
A UID is a structure that contains a module name and a constant
```

```
value used here to indicate either:
the type of a performance parameter (PerfParameter);
or the type of a grouping of performance parameters (PerfDataSet).
*/
       typedef UID          PerfParameter;
                     // The UID representing the type of performance parameter
       typedef UID          PerfDataSet;
                     // The UID representing the type of a grouping of performance parameters

       typedef sequence<PerfDataSet> PerfDataSetList;
                     // A list of performance parameter groupings
       typedef string       FileName;            // A file name

/**
PM Parameter Definitions
*/
       enum SuspectFlag // Indicates that counters are either reliable or unreliable
       {
              reliable,            // reliable for the interval
              unreliable           // unreliable for the interval
       };

       enum GranularityPeriod // The period over which PM measurements are aggregated
       {
              fifteenMinutes,
              twentyFourHours
       };

       enum RetrievalType // For bulk retrieval, indicates how intervals should be retrieved
       {
              all,                 // All intervals should be retrieved
              mostRecent,          // Only most recent interval should be retrieved
              between
              // Intervals between a specified start time and ending time should be retrieved
       };

/**
PM ID Structures

These structs facilitate grain-neutral operation per this document's Section 1.3
in a generic (technology independent) setting.
*/
       struct CurrentDataID
       {
              NameType                    name;
              CurrentData                 ref;
       };

       typedef sequence<CurrentDataID> CurrentDataIDList;

       struct HistoryDataID
       {
              NameType                    name;
              HistoryData                 ref;
       };

       typedef sequence<HistoryDataID> HistoryDataIDList;

       struct ThresholdDataID
       {
              NameType                    name;
              ThresholdData               ref;
       };

       typedef sequence<ThresholdDataID> ThresholdDataIDList;

/**
PM Data Structures
```

```
*/
       struct PerfCounter
       {
              PerfParameter               perfParameter; // type of PM parameter
              CounterValue                counterValue;  // actual counter value
       };

       typedef sequence<PerfCounter> PerfCounterList;  // list of PM counters

       struct PerfThreshold
       {
              PerfParameter               perfParameter;  // type of PM parameter
              ThresholdValue       thresholdValue; // threshold value
       };

       typedef sequence<PerfThreshold> PerfThresholdList; // list of thresholds

       struct ThresholdBulkData
       {
              ThresholdDataID             tDataId;      // ID of threshold instance
              PerfThresholdList           tData;        // list of thresholds
       };

       typedef sequence<ThresholdBulkData> ThresholdBulkDataList;
              // List of threshold instances complete with threshold data

       struct CurrentDataAttributes // The attributes of a current data instance
       {
              CurrentDataID               cDataId;      // ID of instance
              AdministrativeState         adminState;   // administrative state
              OperationalState            operState;    // operational state
              GranularityPeriod           granPeriod;   // granularity period
              ThresholdDataID             threshId;     // associated threshold data
              boolean                    historyRetention;
                     // historyRetention indicates that history data will be retained
              boolean              suppressionIndicator;
/**
suppressionIndicator "true" indicates that intervals with counter value of all zeros will be
suppressed.
*/
       };

       struct CurrentIntervalData
       {
              SuspectFlag                 suspect;
       // The suspect flag indicates that the counters within the interval are not reliable.
              PeriodStartTime             startTime;    // Start time of period
              ElapsedTime                 elapsedTime;  // Elapsed time of period
              PerfCounterList             perfCounterList;
       // The perfCounterList is a list of performance counter types and associated values.
       };

       struct CurrentPMBulkData
       {
              CurrentDataAttributes cDataAttributes;
              CurrentIntervalData         cData;
       };
/**
CurrentPMBulkData provides information about a current data instance
along with its associated counter types and values
*/

       typedef sequence<CurrentPMBulkData> CurrentPMBulkDataList;
/**
CurrentPMBulkDataList provides information about a number of current data instances.
It is used for bulk retrieval of current data information.
*/
```

```
       struct HistoryDataAttributes // The attributes of a history data instance
       {
              HistoryDataID               hDataId;      // ID of instance
              GranularityPeriod           granPeriod;   // granularity period
       };

       struct HistoryIntervalData
       {
              SuspectFlag                 suspect;
       // The suspect flag indicates that the counters within the interval are not reliable.
              PeriodStartTime             startTime;    // Start time of period
              PeriodEndTime               endTime;      // Ending time of period
              NumIntervals                numPrevSuppressedIntervals;
       // The number of prior intervals suppressed due to zero count suppression.
              PerfCounterList             perfCounterList;
       // List of performance counter types and associated values
       };

       typedef sequence<HistoryIntervalData> HistoryIntervalDataList;
/**
The HistoryIntervalDataList provides information about a number of collection intervals
for a history data instance, including the associated counter values.
*/

       struct HistoryPMBulkData
       {
              HistoryDataAttributes hDataAttributes;
              HistoryIntervalDataList     hIntervalData;
       };
/**
HistoryPMBulkData provides information about a history data instance
along with its associated counter types and values
*/

       typedef sequence<HistoryPMBulkData> HistoryPMBulkDataList;
/**
The HistoryPMBulkDataList provides information about a number of history       data instances.
It is used for bulk retrieval of history data information.
A number of intervals may be represented for each history data instance.

INTERFACES

CurrentData interface
*/
       interface CurrentData : NetMgmt::ManagedObject, NetMgmt::Portal
       {

/**
The interface CurrentData is uninstantiable.  Specific interfaces can inherit methods from
the interface CurrentData.

A threshold crossing alert is used to notify the management system when any of the
performance parameters exceeds a pre-set threshold described in the associated ThresholdData
object.  The following information shall be supplied in the Threshold Crossing Alert:
 - The ID of the object reporting the threshold crossing alert (where the ID is the
   (name, ref) pair used for the grain-neutral approach).
 - The type of performance parameter that exceeded the threshold is identified by
   PerfParameter.
*/

              void setAdministrativeState
              // sets the Administrative state for a current data instance
                     (in CurrentDataID           cDataId,
                     in AdministrativeState      adminState)
                     raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

              void setThresholdDataID
              // associates threshold data with a current data instance
```

```
                        (in CurrentDataID           cDataId,
                        in ThresholdDataID          threshDataId)
                        raises (NetMgmt::ObjectFailure, NetMgmt::ItemNotFound,
                               NetMgmt::InvalidID);

                void resetCounters
                // resets the counter values to Zero for a current data instance
                        (in CurrentDataID           cDataid)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                               NetMgmt::InvalidID);

                void setSuppressionIndicator
                // activates or deactivates suppression for a current data instance
                        (in CurrentDataID           cDataid,
                        in boolean                  suppressionIndicator)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                               NetMgmt::InvalidID);

                void setHistoryRetention
                // activates or deactivates history retention for a current data instance
                        (in CurrentDataID           cDataId,
                        in boolean                  retainHistory,
                        out HistoryDataID           hDataId)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                               NetMgmt::InvalidID);

                void getCurrentDataAttributes
                // retrieve attributes for a current data instance
                        (in CurrentDataID           cDataId,
                        out CurrentDataAttributes   cDataAttributes)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

                void getCurrentIntervalData
                // retrieve counter values for a current data instance
                        (in CurrentDataID           cDataId,
                        out CurrentIntervalData     cData)
                        raises (NetMgmt::ObjectFailure, NetMgmt::ItemNotFound);

        }; // interface CurrentData

/**
CurrentDataFactory interface
*/
        interface CurrentDataFactory : NetMgmt::ManagedObject
        {
/**
Expect creation by the EMS.
Used to create specific instances inheriting from CurrentData.
There should be only a single instance of this object per management system.
*/

                void makeSpecificCurrentData
                        (in MOID                    monitoredObject,
                                // ID of the monitored object
                             in PerfDataSetList     cDataTypes,
                                // the PM data sets to be monitored for the monitored object
                        in boolean                  historyRetention,
                                // indicates whether history data will be retained
                        in GranularityPeriod  granPeriod,
                        out CurrentDataIDList cDataIdList,
                        out HistoryDataIDList hDataIdList)
                        raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                               NetMgmt::InvalidID, NetMgmt::MaxMonitorPointsExceeded);
        }; // interface CurrentDataFactory

/**
HistoryData interface
This interface is uninstantiable.
```

```
*/
        interface HistoryData : NetMgmt::ManagedObject, NetMgmt::Portal
        {
                void getHistoryDataAttributes
                // retrieve attributes for a history data instance
                        (in HistoryDataID           hDataId,
                        out HistoryDataAttributes   hDataAttributes)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

                void getHistoryIntervalData
                // retrieve history interval information for a history data instance
                        (in HistoryDataID           hDataId,
                        in PeriodStartTime          periodStartTime,
                                // no value implies the earliest available time
                        in PeriodEndTime            periodEndTime,
                                // no value implies the latest available time
                        in RetrievalType            retrievalType,
/**
The RetrievalType indicates:
        all – all intervals should be retrieved,
        mostRecent – only the most recent interval should be retrieved, or
        between – intervals between the specified period start time and period end time
                    should be retrieved.
*/
                        out HistoryIntervalDataList hData)
                        raises (NetMgmt::ObjectFailure, NetMgmt::ItemNotFound);

        }; // interface HistoryData

/**
ThresholdData interface
*/
        interface ThresholdData : NetMgmt::ManagedObject, NetMgmt::Portal
        {
/**
Expect creation by the EMS.
*/
                void getThresholdData // retrieve attributes for a threshold data instance
                        (in ThresholdDataID         tDataId,
                        out PerfThresholdList tData)
                        raises (NetMgmt::ObjectFailure);

                void setThresholdData // set attributes for a threshold data instance
                        (in ThresholdDataID         tDataId,
                        in PerfThresholdList  tData)
                        raises (NetMgmt::ObjectFailure);

        }; // interface ThresholdData

/**
PM Bulk Data Iterator interfaces
*/
        interface CurrentPMBulkDataIterator : NetMgmt::ManagedObject
        {
/**
Created as a result of a bulk retrieval operation for current data.
Retrieves next chunk of information for a current data iterator.
Returns boolean "false" when there is no further data to send.
*/
                boolean nextN
                        (in unsigned long           howMany,
                        out CurrentPMBulkDataList   cBulkData);

        }; // interface CurrentPMBulkDataIterator

        interface HistoryPMBulkDataIterator : NetMgmt::ManagedObject
        {
/**
```

```
Created as a result of a bulk retrieval operation for history data.
Retrieves next chunk of information for a history data iterator.
Returns boolean "false" when there is no further data to send.
*/
              boolean nextN
                     (in unsigned long           howMany,
                     out HistoryPMBulkDataList   hBulkData);

       }; // interface HistoryPMBulkDataIterator

       interface ThresholdPMBulkDataIterator : NetMgmt::ManagedObject
       {
/**
Created as a result of a bulk retrieval operation for threshold data.
Retrieves next chunk of information for a threshold data iterator.
Returns boolean "false" when there is no further data to send.
*/
              boolean nextN
                     (in unsigned long           howMany,
                     out ThresholdBulkDataList   tBulkData);

       }; // interface ThresholdPMBulkDataIterator

/**
PM Bulk Operations interface
*/
       interface PMBulkOperations : NetMgmt::ManagedObject
       {
/**
Expect creation by the EMS.
Within each of the get...PMBulkData methods, the MOIDList gives a generic approach
for specifying the (list of) objects that act as containers      for the PM data to be retrieved.
And for each of these methods, the PerfDataSetList indicates the sets of PM data that
are to be retrieved (i.e., delivered to the client) as a result of the request.
*/
              void getCurrentPMBulkData
                     (in MOIDList                moIdList,
                     in PerfDataSetList          dataTypes,
                     in unsigned long            howMany,
                     out CurrentPMBulkDataList   cBulkData,
                     out CurrentPMBulkDataIterator cBulkDataIterator)
                     raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);
/**
Bulk retrieval of current performance counters contained in objects that are
indicated in the moIdList.  Retrieves the counters that are included
within the grouping indicated by dataTypes.
The parameter howMany specifies the maximum number of instances for which
attributes can be returned in a single message.  This method returns the first
chunk of PM information with cBulkData, and it also creates an instance
of the CurrentPMBulkDataIterator interface for subsequent chunks of information.
*/

              void getHistoryPMBulkData
                     (in MOIDList                moIIdList,
                     in PerfDataSetList          dataTypes,
                     in unsigned long            howMany,
                     out HistoryPMBulkDataList   hBulkData,
                     out HistoryPMBulkDataIterator hBulkDataIterator)
                     raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);
/**
Bulk retrieval of history performance counters contained in objects that are
indicated in the moIdList.  Retrieves the counters that are included
within the grouping indicated by dataTypes.
The parameter howMany specifies the maximum number of instances for which
attributes can be returned in a single message.  This method returns the first
chunk of PM information with hBulkData, and it also creates an instance
of the HistoryPMBulkDataIterator interface for subsequent chunks of information.
*/
```

```
                void getAllThresholdDataIDs
                // retrieve IDs for all Threshold Data instances
                        (out ThresholdDataIDList     tDataIdList)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

                void getThresholdBulkData
                        (in ThresholdDataIDList      tDataIdList,
                        in unsigned long             howMany,
                        out ThresholdBulkDataList    tBulkData,
                        out ThresholdPMBulkDataIterator tBulkDataIterator)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);
/**
Bulk retrieval of Threshold Data information for the instances that are
indicated in the tDataIdList.
The parameter howMany specifies the maximum number of instances for which
attributes can be returned in a single message.  This method returns the first chunk of
Threshold Data information with tBulkData, and it also creates an instance
of the ThresholdBulkDataIterator interface for subsequent chunks of information.
*/

                void setThresholdBulkData
                // permits bulk setting of threshold values in Threshold Data instances
                        (in ThresholdBulkDataList    tBulkData)
                        raises (NetMgmt::ObjectFailure, NetMgmt::InvalidID);

        }; // PMBulkOperations interface

/**
Performance Data File Generator interface
*/
        interface PerformanceDataFileGenerator
        {
/**
Expect creation by the EMS.
This method results in the creation of a structured file that
contains bulk performance information.  This file may be retrieved
by a data collection system through a number of file transfer
mechanisms.  The actual file transfer is outside the scope of CORBA.

Within the generateHistoryData File method, the ObjectClass
represents a generic way to describing the object class that acts
as a container for the PM data to be generated.
Also, the PerfDataSetList identifies the list of perfomance data
counter sets that will be generated as a result of the request.

The RetrievalType indicates:
        all – all intervals should be retrieved,
        mostRecent – only the most recent interval should be retrieved, or
        between – intervals between the specified period start time and period end time
                    should be retrieved.
*/
                void generateHistoryDataFile
                        (in ObjectClass              objectType,
                        in PerfDataSetList           dataTypes,
                        in RetrievalType             retrievalType,
                        in PeriodStartTime           periodStartTime,
                                // no value implies the earliest available time
                        in PeriodEndTime             periodEndTime,
                                // no value implies the latest available time
                        out FileName                 dataFileName)
                        raises (NetMgmt::NotSupported, NetMgmt::ObjectFailure);

        }; // PerformanceDataFileGenerator interface

}; // end of module corba_pm
#endif // _corba_pm_idl_
```

6 Scenario Diagrams

The scenario diagrams are provided only as examples of how the some IDL object interfaces can be used in different operational scenarios. This section is for information ONLY.

Figure 6-1 illustrates some representative exchanges of messages between a client (managing system) and a server (managed system). These messages illustrate four selected methods (or operations), that are defined in Section 5.

Synchronous messaging is assumed in these diagrams. Synchronous messaging implies that the server automatically returns the output specified by the client's message invoking a particular method (or else throws a specified exception). Hence with synchronous messaging, there is no need to explicitly illustrate the message that returns a method's output to the client

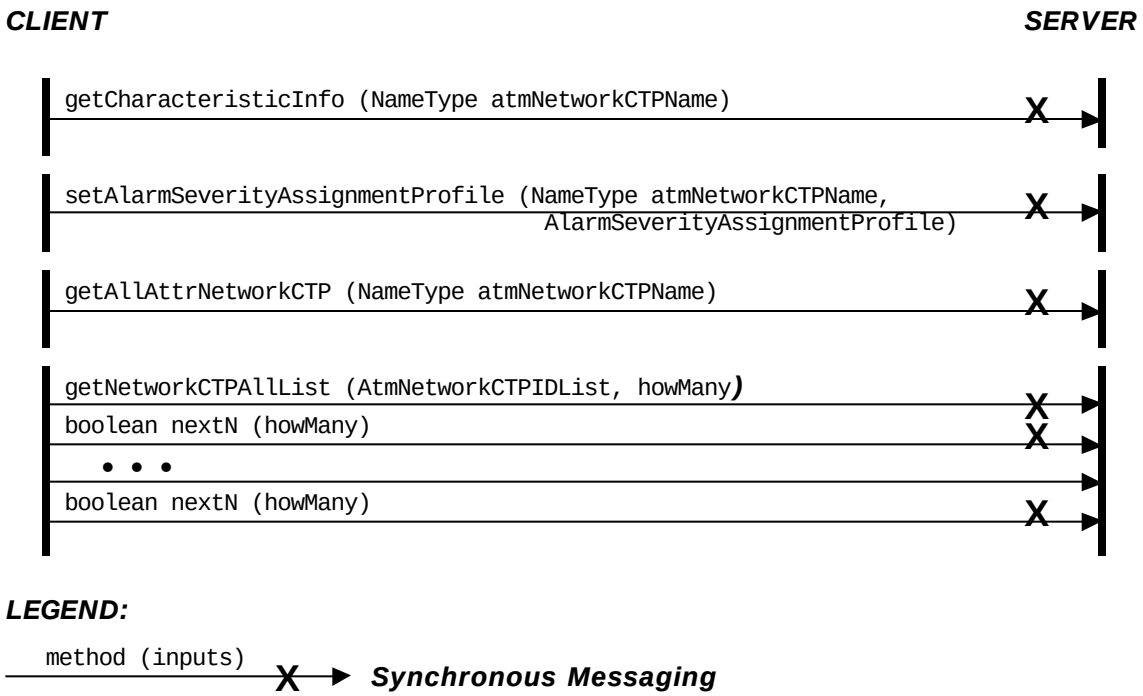

Figure 6-1. Some Synchronous Messaging Examples

In the first method illustrated in Figure 6-1, the client gets (retrieves) a particular attribute, `CharacteristicInfo`, of one managed object, `AtmNetworkCTP`. This object is specified as the input parameter by its name, `NameType atmNetworkCTPName`.

In the second method illustrated in this figure, the client sets (assigns or changes) a particular attribute, `AlarmSeverityAssignmentProfile`, of one managed object, `AtmNetworkCTP`. This object is specified as the input parameter by its name, `NameType atmNetworkCTPName`. The second input parameter is the specified value of `AlarmSeverityAssignmentProfile`.

In the third method illustrated in this figure, the client gets all of the attributes of one managed object, and these attributes are contained in the struct `AtmNetworkCTPAll`. This object is specified as the input parameter by its name, `NameType atmNetworkCTPName`.

In the fourth method illustrated in this figure, the client gets all attributes of each managed object in a list, and these attributes are contained in the sequence of structs, `AtmNetworkCTPAllList`. This list of objects is specified as an input parameter by a sequence of structs, `AtmNetworkCTPIDList`, where each struct in this sequence is an instance of `AtmNetworkCTPID`.

In this fourth method, the client fixes the second input parameter `howMany` to control the length of the returned message. This method also returns an iterator (see Section 5.4.2) which provides the method `boolean nextN` for recovering any additional structs belonging to the sequence `AtmNetworkCTPAllList` that cannot be returned in the server's initial responding message because of the message length limitation established by `howMany`. The client may repeatedly invoke the method `boolean nextN` (with its own input parameter `howMany`) until all elements of `AtmNetworkCTPAllList` have been recovered (which is indicated by a `boolean nextN` invocation returning a "false" value). This scenario is illustrated in Figure 6-1. Alternatively, the client may terminate this process at any time by invoking the `destroy` method for this iterator interface (which is available via inheritance from `ManagedObject`).

References

[1] ATM Forum, “M4 Interface Requirements and Logical MIB: ATM Network View”, version 2, af-nm-0058.001, May 1999.
[2] ATM Forum, “M4 Interface Requirements and Logical MIB: ATM Network Element View,” version 2, af-nm 0020.001, October 1998.
[3] OMG, “The Common Object Request Broker: Architecture and Specification”, Revision 2.2, February 1998.
[4] OMG, “CORBAservices: Common Object Services Specification”, Updated version, December 1998.
[5] OMG, “Notification Service”, OMG TC Document telecom/98-11-01, November 3, 1998.
[6] OMG, “Telecom Log Service”, OMG TC Document telecom/99-05-01, May 1, 1999.
[7] OMG, “CORBA Messaging”, OMG TC Document orbos/98-05-05, May 18, 1998
[8] ATM Forum, "Traffic management Specification," version 4.1, af-tm-0121.000, March 1999.

Appendix A: CORBA Common Object Services Requirements

The CORBA ORB provides basic object-to-object interaction capabilities[3]. Additional capabilities are defined as separate, "Common Object Services."[4]. The CORBA Common Object Services are general purpose, domain-independent services that are fundamental for developing CORBA applications composed of distributed objects. They also provide the basic building blocks for application interoperability. The services are defined with object interfaces and can be combined in many different ways and put to many uses in different applications. In a specific domain, CORBA Common Objects can be used to construct higher-level facilities and object frameworks that can inter-operate across multiple platform environments.

Many of these CORBA Common Object Services have already been implemented and are available as commercial, off-the-shelf software products. Also, developers working in many industries will likely have experience with them in the near future. Re-using these Common Object Services instead of defining new ones strictly for the telecommunications industry or re-implementing the functionality in application-specific code will result in a quicker, more cost-efficient adoption of CORBA for network management.

The following sub-sections specify required CORBA Common Object Services and recommendations on its use to ensure interoperability between different network management systems.

A.1 Naming Service

The OMG Naming Service is the CORBA's directory service, or "white pages"[4]. It allows a client to build a name-to-object association called a *name binding* that other clients can then use to find the object. (CORBA object addresses can be long and difficult for use by humans.) A name binding is always defined relative to a *naming context.* A naming context is an object that contains a set of name bindings in which each name is locally unique. A name binding is a data structure containing two strings and an object reference (address). The ***"ID"*** string is the identifier for the binding and must be unique within a context. A second string, called ***"kind"***, is also part of the data structure. Different names can be bound to an object in the same or different contexts at the same time. The naming context can also be bound to a name in another naming context. Binding contexts in other contexts creates a *naming graph* – a directed graph with nodes and labeled edges where nodes are contexts. Given a context in a naming graph, a sequence of name components (*ID-Kind* pairs) can reference an object. This sequence of structures, called a *compound name*, defines a path in the naming graph that may be navigated to resolve the name and find the object.

There is no requirement that CORBA name bindings represent a containment relationship between objects, but the concept of containment is important in network management and needs to be communicated across network management interfaces. The CORBA Naming Service is the best way to accomplish this. The following paragraphs define a series of requirements on using the CORBA Naming Service to represent the containment relationships among managed object instances.

(R) NAME-1 Every managed object shall have one and only one name (DN). The components of the name may be obtained from multiple federated servers. Although the OMG Naming service supports multiple names per object, this specification restricts a managed object to using a single name. Support for multiple names is outside the scope of the specification.

(R) NAME-2 Since a simple name binding cannot identify an object and also contained objects, each managed object must actually have a corresponding Naming Context. A specially-named binding in each such context will bind the *ID* value "Object" with a reference to the actual managed object. (The *kind* field of this binding will be null.) Other naming contexts, representing contained managed objects, may also be bound to names in this context.

(R) NAME-3 The *ID* field of a name binding for a naming context representing a managed object will be application-dependent, and it may actually have semantic value beyond uniquely identifying a managed object, for a particular class of objects. For example, an ID value of "7" for an equipment holder object representing a slot in a shelf may indicate that this object represents the 7th slot in the shelf. Special semantic value attached to IDs will be documented for each class of managed objects as part of the managed object interface specification. Note that the *ID* field is a string.

(R) NAME-4 The *kind* field of a name binding for a naming context representing a managed object shall be determined by *managed object name binding information*. This is information defined as constants in IDL modules specifically for the purpose of representing possible containment relationships.

The following figure gives an example of name bindings according to the above requirements. In the figure, CORBA Naming Contexts are represented as folders. The contents of the folders are name bindings. The convention for representing a name component as a string with the format <ID>.<kind> is used. (Some example name bindings do not have a pointer shown in the diagram to reduce the complexity of the diagram.) The graph represents a Network object, named "CentralNet," that contains a Managed Element object named "Element9" and a Connection named "R5698."

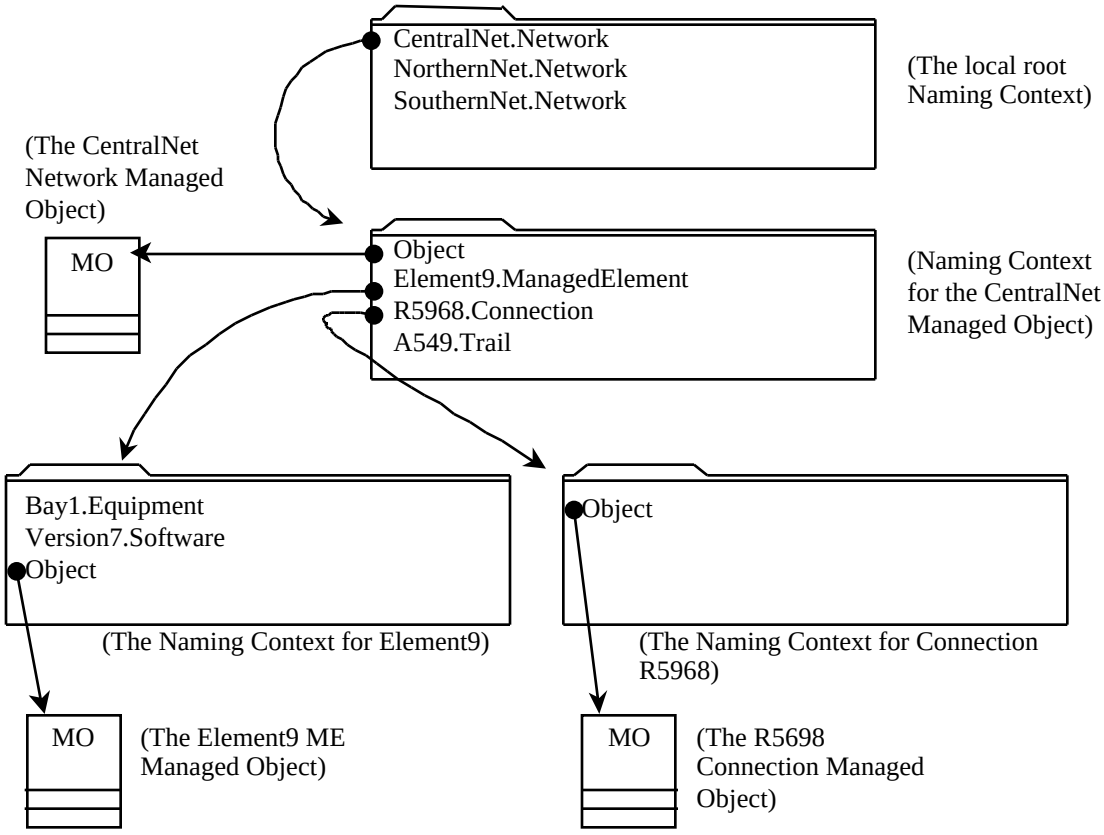

Figure A-1. Example Naming Graph of Managed Objects

(R) NAME-5 Each managed system shall provide at least one local root naming context. Note on the figure above that the top-most naming context is referred to as a "local root" naming context. This is the naming context in which names for the top-most managed objects on the system will be bound, as well as names for certain support service objects.

A managed system may have multiple local root naming contexts. Since managed objects cannot have multiple names, they may be bound under only one local root. Support service objects, however, may have names bound under multiple root naming contexts on the same system. One factor to consider when determining how many local root naming contexts a managed system will have is if the possibility exists that some of the managed objects might sometime have to be moved to another system. Moving an entire tree of managed objects, including the local root naming context, will be simpler than moving a subtree of objects.

(R) NAME-6 A managed system shall provide a local administrative procedure for assigning a CORBA name to each local root naming context on the system. All names exchanged across the managed interface will include the local root context name unless otherwise noted. This includes operation parameters and notifications.

This feature is to enable an administration to make names globally unique. Since the managed system must ensure that all names are unique relative to the local root naming context, by assigning a globally unique name to the local

root naming context an administration can ensure that all names on a managed system are unique. The mechanism used to choose a globally unique name for the local root context is up to the administration. The format of the name will be the same as used by the CORBA Naming Service, *CosNaming::Name*. Multiple components are allowed, but administrations will likely want to keep local root context names short to reduce overhead.

In addition to making names unique, assigning a name to the local root naming context will make it easier for a managing system to resolve names. This is because the managing system can bind the local root naming contexts for all the systems it manages into its own local naming service. The name it uses for this binding will be the same name assigned to the root naming context on the managed system. See Figure A-2 for an example.

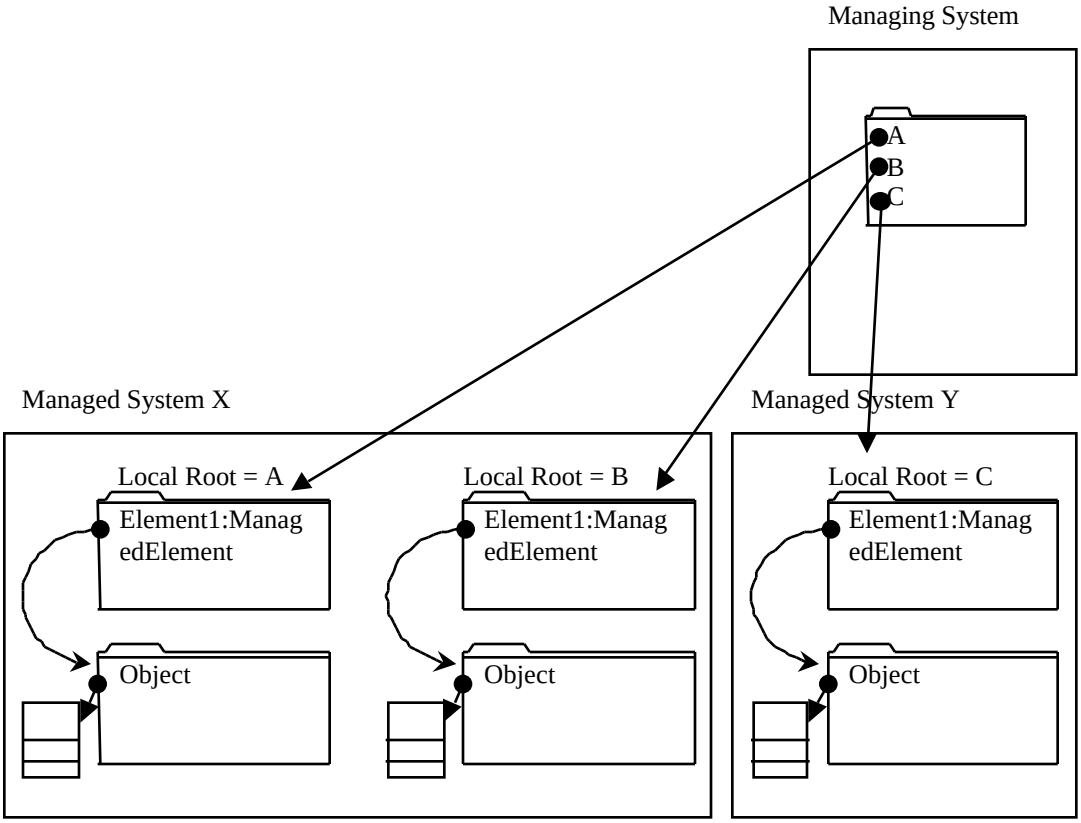

Figure A-2. Assigning Names to Local Root Naming Contexts

The figure shows two element management systems on the bottom. System "X" has two objects of type *ManagedElement*, and System "Y" has 1. Each ManagedElement object belongs to its own local root naming context, which means System X has two local roots and System Y has one. There is also a network management system, and the local root contexts of both EMSs have been bound into the naming service on this system. This administration has chosen to assign the unique names "A" and "B" to the local root contexts on System X, and "C" to the local root context on System Y. References to the local root naming contexts have been bound with these names in the network management System.

Say the System Y emits a notification concerning its *ManagedElement* object. The full name of that object (contained in the notification) will be "C/Element1ManagedElement". Now let's say the NMS wants to retrieve more data from the object. In order to do so, it will have to resolve the name into a CORBA object reference. The NMS can accomplish this by simply performing a resolve operation using the full name on the local context where it bound the EMS local root contexts. Because the NMS' naming service is federated with the EMS naming services, the NMS' naming service can automatically forward the resolve operation to the naming service on the proper EMS, and return the object reference to the NMS application.

It is anticipated that the local root naming context name will be assigned during the initialization of a new system. Once in operation, it will be extremely difficult if not impossible to change.

Once assigned a name, the local root context's CORBA Interoperable Object Reference (IOR) will have to be bound to a naming context on the managing system, since up to now it has no idea the new system exists. This means the managed system will also have to provide a means for accessing the "stringified" IOR of the local root naming context. This value will then be transferred to the managing system by some means other than the management interface (e-mail, ftp, etc.). The managing system will require a way to accept this stringified IOR and bind it to a name on the managing system. As soon as the local root context's IOR is bound to a name on the managing system, the managing system can begin discovering the objects on the new system (using the Multiple Object Operation Service described later) and begin to manage it.

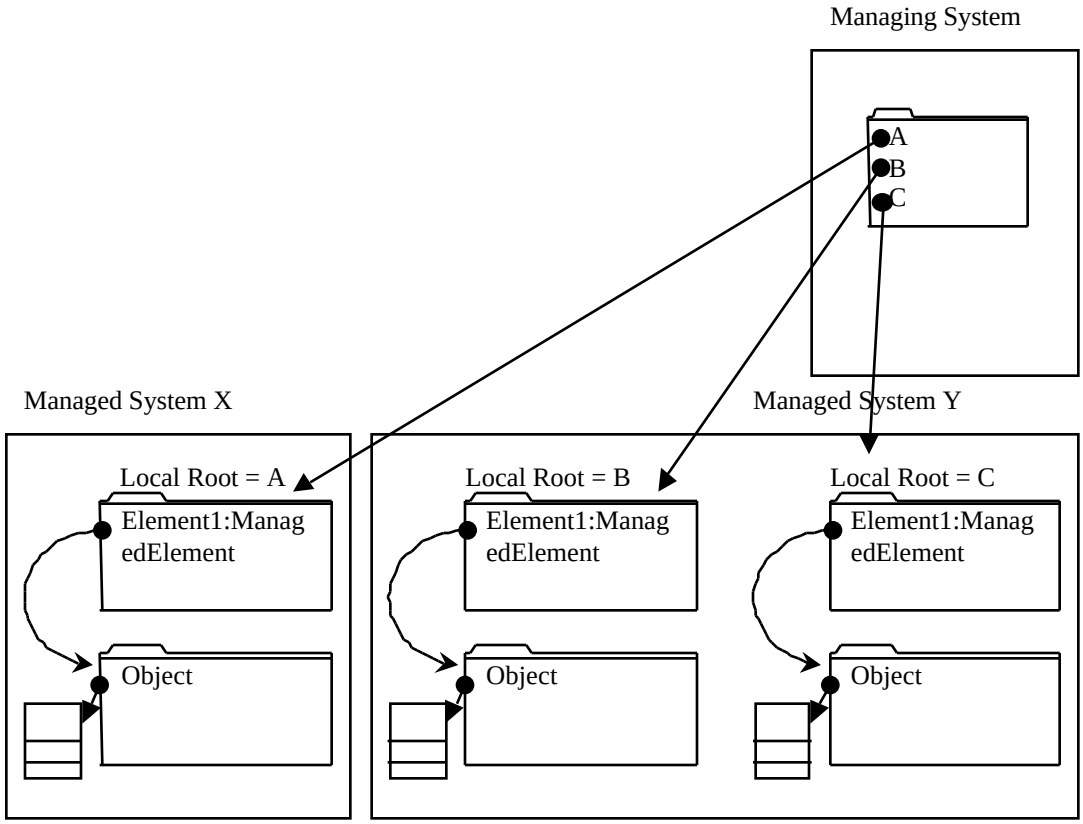

Figure A-3. Moving a Local Root Naming Context and Contained Objects

Figure A-3 shows how a local root naming context and all of the objects contained below it can be moved to another system without changing the names of the objects. The only change that might be required would be to change the object reference bound to the name in the network management system(s). Also, any outstanding references to moved objects would have to be refreshed. Moving only part of a tree contained below a local root naming context would require re-naming those objects.

A.2 Notification Service

The CORBA Notification Service supports the asynchronous exchange of event messages between clients using a subscribe-and-publish paradigm[5]. The Notification Service introduces event channels that broker event messages, notification suppliers that supply event messages, and notification consumers that consume event messages. The CORBA Notification Service preserves all of the semantics specified for the CORBA Event Service, allowing for backward compatibility with Event Service clients. The extended functionality that is important to the network

management domain is the structured event, event filtering, and QoS (Quality of Service). The figure below depicts the general architecture of the Notification Service.

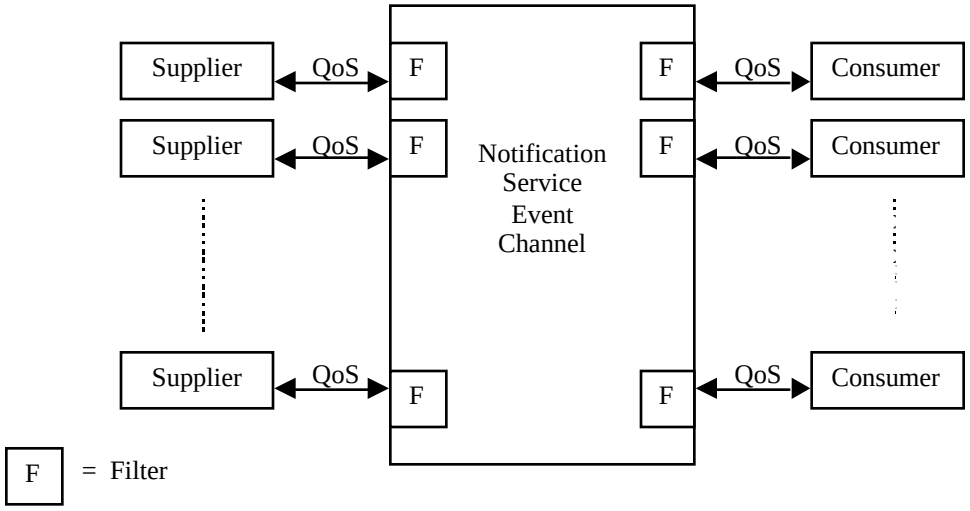

Figure A-4. Architecture of the Notification Service

(R) NOTIF-1 The Notification Service shall support the push interface model. The managed object interface to the event channel shall be a push supplier.

(R) NOTIF-2 The managed system shall instantiate the Notification Service event channel object(s) that it will use. A managed system must instantiate at least one channel and may instantiate more than one. The specification does not support the creation or deletion of event channels across the management interface. Local administrative procedures may be provided for this purpose. (Event channels do, however, support the creation and deletion of filters across the management interface.)

(R) NOTIF-3 The Notification Service shall support structured events. Support for typed events is optional.

(R) NOTIF-4 The form of event messages shall be structured events. The use of typed events is optional. The message interface between suppliers and consumers shall be defined in IDL as if they were using typed events. This is done to enable capturing the notification in IDL (which cannot be done for structured events except with comments) as well as to support typed notifications for applications that wish to use them. Rules for creating structured notifications based on these typed operations are provided below.

The OMG Notification Service definition does define rules for channels to automatically convert typed notifications to structured notifications. If the managed system natively creates typed notifications, but the client wishes to receive structured notifications, these rules shall be followed by the channel. Note, however, that this arrangement is likely less efficient than both systems using typed events. If the managed system natively creates structured notifications, it shall do so according to the rules below.

The structured notifications natively created by a managed system will differ slightly from the structured notifications created by automatic conversion from typed notifications. One reason for this is to make it possible for a managing system to tell the difference, and accept typed notifications if they are supported by the managed system. Another is to more efficiently use structured notifications. Managed systems that natively create structured notifications may exclude optional parameters from those notifications. Because a typed notification is created from a strongly-typed method invocation, a commercial notification channel that translates this to a structured notification will include any null values as name-value pairs in the body of the structured event rather than exclude them. Note that allowing managed systems that natively create structured notifications to exclude optional parameters makes it unlikely that commercial notification channels will be able to support the automatic conversion of structured events to typed events.

To recap, a managed system shall send notifications either as structured events or typed events. If the managed system natively creates structured events, it shall do so according to the rules below. Because, for efficiency, these rules allow managed systems to exclude optional parameters from structured notifications, support for automatic conversion of these structured notifications to typed notifications by commercial notification channels is not expected. Thus, the managing system must accept structured events. If the managed system natively creates typed events, the managing system may rely on the notification channel to automatically convert them to structured events based on the OMG Notification Service's rules. Structured notifications rely upon the heavy use of CORBA "any" data types, however, which can be inefficient. Thus in this case managing system will likely prefer to accept typed notifications.

(R) NOTIF-5 The suppliers and consumers of structured events shall follow these rules for constructing and receiving the structured events. (See the figure below which depicts the Notification Structure and how elements from the IDL notification definition are to be mapped into it):

- The `domain_type` string in the fixed header of the structured event shall be set to "Telecommunications".
- The `type_name` ***string in the fixed header of the structured event shall be set to the scoped name of the operation defining the notification in IDL, for example, "`itut_x780::Notifications::attributeValueChange`".***
- The `event_name` string in the fixed header of the structured event shall be null.
- Optional header fields may be included to support features like Quality of Service as appropriate.
- Each parameter in the operation shall be placed in a name-value pair in the filterable body portion of the structured event. The `fd_name` string of this pair shall be set to the name of the parameter and the type placed in the associated `fd_value` will be the type specified for the parameter. Using as an example the `equipmentAlarm` notification from the IDL presented later in this document, the first `fd_name` string would be set to "`eventTime`" and the first `fd_value` would contain an `ExternalTimeType` data type. Although all notification parameters go in the filterable body of the notification structure, depending on the data type of the parameter it may be difficult or even impossible to create a useful filter utilizing that parameter. Filter "matching rules" are based on the capabilities of the channel.
- Parameters that are denoted "optional" may optionally be excluded from the notification structure. If typed notifications are used, these parameters are included, but will usually have a special null value if not supported. For types for which there is no special null value (such as integers) a special type consisting of a union between the base type (such as integer) and the null type is usually defined. These union types may be excluded from structured notifications when they have a null value, but if they are included, the union type must be used. This is to enable the same filters to be used for both structured and typed notifications.
- The remainder of the body of the structured event (the non-filterable part) shall be null.
- Parameters named "operation" shall be avoided in notification operations to potentially support the use of typed notifications. (When converting typed notifications to structured notifications, the parameters of an operation are automatically placed into a notification structure by the event channel. Unfortunately, the rules developed for doing this state that the name of the operation used to issue the notification goes not in the header of the event, but in the body of the of the structure as the first name-value pair. The `fd_name` string is set to "operation" and the `fd_value` is set to a string containing the name of the operation. Using a parameter named "operation" would then result in a second name-value pair with the name "operation," and the two could be confused.)

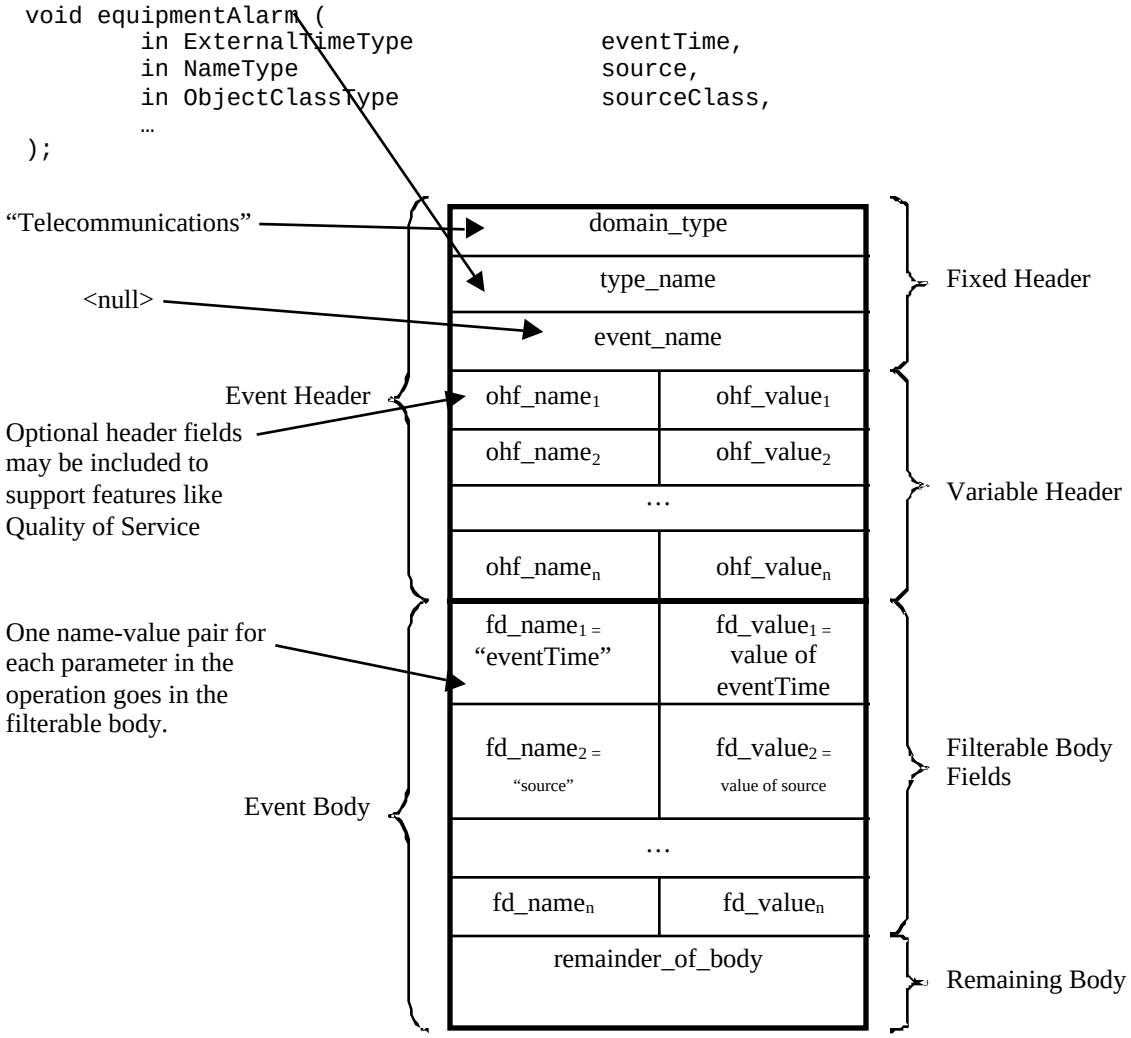

Figure A-5. Mapping Notifications to Structured Events

(R) NOTIF-6 The Notification Service specification supports filter expressions that are used to determine if the event is to be forwarded. It also supports filter expressions that "map" values in the notification to parameters used to impact the operation of the event channel, such as the QoS used in delivering the event. For example, a mapping filter might be used to map a "severity=major" field from an event (which means nothing to an event channel) to a QoS parameter "priority=1" (which does mean something to the channel). The Notification Service shall support event filtering with filter objects that support constraints expressed in the default constraint grammar specified by the OMG. The Notification Service shall also support mapping filters.

(R) NOTIF-7 The Notification Service reliability QoS shall support EventReliability=Persistent & ConnectionReliability=Persistent.

Each event is guaranteed to be delivered to all consumers registered to receive it at the time the event was delivered to the channel, within expiry limits. If the connection between the channel and a consumer is lost for any reason, the channel will persistently store any events destined for that consumer until each event time out due to expiry limits, or the consumer once again becomes available and the channel is subsequently able to deliver the events to all registered consumers. In addition, upon start from a failure the notification channel will automatically re-establish connections to all clients that were connected to it at the time the failure occurred.

(R) NOTIF-8 The Notification Service order policy QoS shall allow the events to be delivered in the order of their arrival, i.e. FIFO. The Notification Service may also optionally support a priority-order QoS in which events could be buffered in priority order, such that higher priority events will be delivered before lower priority events.

(R) NOTIF-9 The Notification Service implementation deployed shall be compliant to the conformance statements of the OMG Notification Service specification with the exception of the pull interface model.

In a distributed computing environment, such as CORBA interfaces can support, it is possible that updates from some clients can be overwritten by concurrent (or near-concurrent) updates from other clients unless suitable safeguards are provided. Even though the Notification Service and the Telecom Log Service provide a basis for making a client aware that its update has been overwritten, they do not provide a locking mechanism to prevent the occurence of such overwrites. Consideration of mandatory responses is necesary if reliance is to be placed solely upon these services.

The OMG Transaction Service [4] provides a comprehensive locking mechanism for preventing the overwriting of one client's update by the near-concurent update of a different client; and this solution is designed for high reliability. However, the OMG Transaction Service may not be required in all applications, and its additional overhead may not be justified.

A.3 Telecom Log Service

The CORBA Telecom Log Service[6] is a CORBA-based log service that fully supports the ITU-T X.735 recommendation. The log is implemented as an Event Service or Notification Service event channel. The Log Service supports the following functionality:

- Writing to the log: Events supplied to the log are persistently stored as log records.
- Forwarding from the log: Events supplied to the log are also forwarded to other logs or to any application that wishes to receive them.
- Log generated events: The log itself will generate events.

Also the Log Service provides functions of log control and management, log record manipulation, log lifecycle management. Figure A-6 gives a graphic representation of the Log Service.

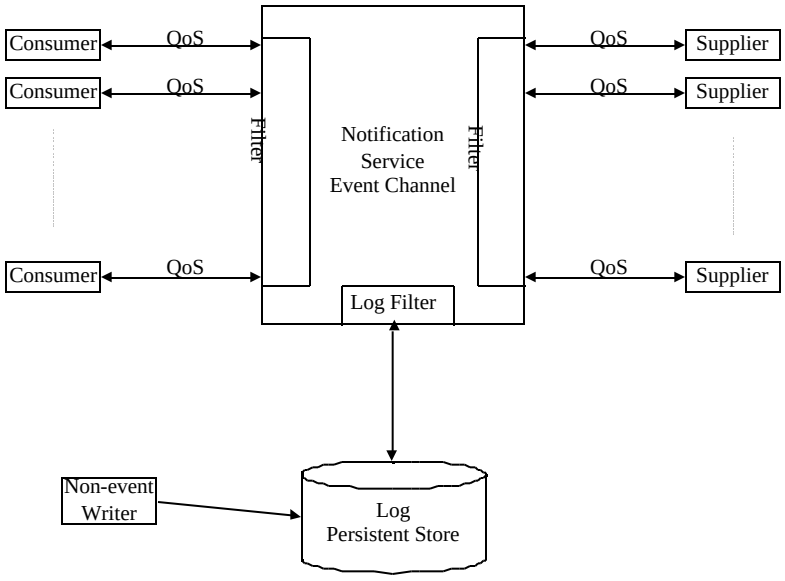

Figure A-6. Telecom Log Service

By manipulating the Log Filter, a managing system is able to control which events are logged and which aren't, in exactly the same way it is able to control which events are forwarded and which aren't. The only exception is the "Non-event Writer," which is an application that writes data directly to the log.

(CR) LOG-1 If a CORBA implementation support Telecom Log Service, the Log Service shall support all the Notification Service requirements.

(R) LOG-2 The Log Record supported by the Log Service shall be the normal `struct LogRecord`. The support of `struct TypedLogRecord` is optional.

(CR) LOG-3 If a CORBA implementation support Telecom Log Service, the Log Service implementation shall be compliant with the required portion of conformance statement in the OMG Telecom Log Service specification with the exception of the pull interface model.

A.4 Messaging Service

The CORBA Messaging Service covers three areas: Asynchronous Method Invocation (AMI), Time Independent Invocation (TII), and Messaging Quality of Service (QoS)[7]. Of the three areas, the AMI has a significant role in the network management domain because it allows clients to make non-blocking requests on a CORBA object. The AMI is treated as a client side language mapping issue only. In most cases, server side implementations are not required to change. In certain situations, such as with a transactional server, the asynchrony of a client does matter and requires server side changes if it is expected to handle transactional asynchronous requests. Transactional requests, however, will not be addressed in this document. Figure A-7 depicts the basic concept of the OMG AMI model.

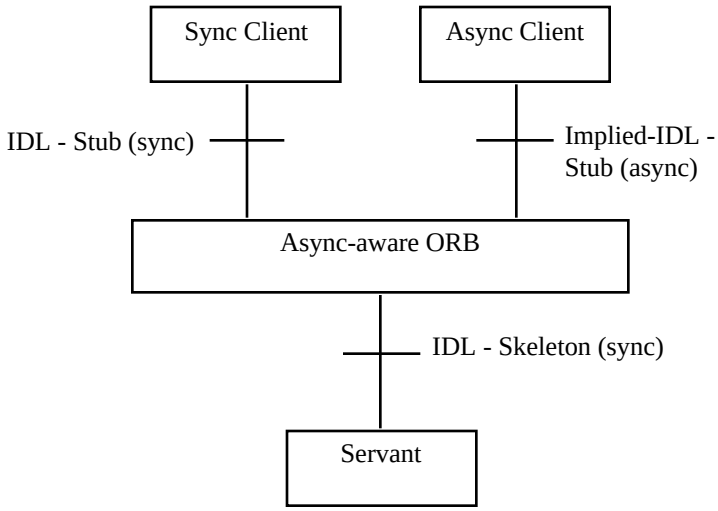

Figure A-7. Asynchronous-aware ORB

The AMI specification provides two models of asynchronous requests: *callback* and *polling*. In the *callback* model, the client passes an object reference for a `ReplyHandler` ***object*** as a parameter when it invokes a two-way asynchronous operation on a server. When the server responds, the client ORB receives the response and dispatches it to the appropriate method on the `ReplyHandler` servant so the client can handle the reply. In other words, the ORB turns the response into a request on the client's `ReplyHandler`. The `ReplyHandler` is a normal CORBA object that is implemented by the programmer as with any object implementation. In the *polling* model, the client makes the request passing in all the parameters needed for the invocation, and is returned a `Poller` object which can be queried to obtain the results of the invocation. This `Poller` is an instance of a `valuetype`, which is a new IDL type introduced by the new *Objects-by-Value* specification. A `valuetype` has

both data members and methods, which when invoked are just local function calls and not distributed CORBA operation invocations.

The value of the Asynchronous Method Invocation capability in network management applications is that it enables managing systems that wish to use asynchronous method calls to inter-operate with managed systems using the same interface definitions as those used by synchronous clients. No changes are required in the interface definition or the implementation of the managed system. The following recommendations are proposed for implementations that that optionally wish to support asynchronous, non-transactional method invocations.

(O) AMI-1 The AMI-aware CORBA implementation shall at least support the callback programming model.

(O) AMI-2 For each operation in an IDL interface, the AMI-aware CORBA implementation shall generate corresponding asynchronous callback method signatures. These signatures are described in implied-IDL which is used to generate language-specific operation signatures.

(O) AMI-3 The AMI-aware CORBA ORB shall pass a type-specific `ExceptionHolder` value instance that contains the marshaled exceptions as its state to the `ReplyHandler` when exception replies are returned from the CORBA object. The AMI-aware IDL compiler would generate a type-specific `ExceptionHolder` for each IDL interface.

(O) AMI-4 The AMI-aware IDL compiler shall generate a type-specific reply handler for each IDL interface. The client will implement and register a reply handler with each asynchronous request and receive a callback when the reply is returned for that request. This reply handler is derived from the generic `Messaging::ReplyHandler`.

A.5 Security Service

A security specification is provided in Reference 4. The possible requirement of a security specification by implementations of this M4 Network View Interface CORBA Specification will be considered in future releases of this document.

Appendix B: Generic Network Management IDL Definitions

B.1 Generic Network Management Constants IDL Definitions (ITU_X721Const.idl)

```
#ifndef _ITU_X721Const_idl_
#define _ITU_X721Const_idl_

module ITU_X721Const {
const string moduleName = "ITU_X721Const";

/** This module contains the constant values defined for the
ProbableCause UID.  These values were borrowed from X.721. */

module ProbableCauseConst {
const string moduleName = "ITU_X721Const::ProbableCauseConst";

        const short indeterminate = 0;
        const short adapterError = 1;
        const short applicationSubsystemFailure = 2;
        const short bandwidthReduced = 3;
        const short callEstablishmentError = 4;
        const short communicationsProtocolError = 5;
        const short communicationsSubsystemFailure = 6;
        const short configurationOrCustomizationError = 7;
        const short congestion = 8;
        const short corruptData = 9;
        const short cpuCyclesLimitExceeded = 10;
        const short dataSetOrModemError = 11;
        const short degradedSignal = 12;
        const short dTE_DCEInterfaceError = 13;
        const short enclosureDoorOpen = 14;
        const short equipmentMalfunction = 15;
        const short excessiveVibration = 16;
        const short fileError = 17;
        const short fireDetected = 18;
        const short floodDetected = 19;
        const short framingError = 20;
        const short heatingOrVentilationOrCoolingSystemProblem = 21;
        const short humidityUnacceptable = 22;
        const short inputOutputDeviceError = 23;
        const short inputDeviceError = 24;
        const short lANError = 25;
        const short leakDetected = 26;
        const short localNodeTransmissionError = 27;
        const short lossOfFrame = 28;
        const short lossOfSignal = 29;
        const short materialSupplyExhausted = 30;
        const short multiplexerProblem = 31;
        const short outOfMemory = 32;
        const short ouputDeviceError = 33;
        const short performanceDegraded = 34;
        const short powerProblem = 35;
        const short pressureUnacceptable = 36;
        const short processorProblem = 37;
        const short pumpFailure = 38;
        const short queueSizeExceeded = 39;
        const short receiveFailure = 40;
        const short receiverFailure = 41;
        const short remoteNodeTransmissionError = 42;
        const short resourceAtOrNearingCapacity = 43;
        const short responseTimeExcessive = 44;
        const short retransmissionRateExcessive = 45;
        const short softwareError = 46;
        const short softwareProgramAbnormallyTerminated = 47;
```

```
        const short softwareProgramError = 48;
        const short storageCapacityProblem = 49;
        const short temperatureUnacceptable = 50;
        const short thresholdCrossed = 51;
        const short timingProblem = 52;
        const short toxicLeakDetected = 53;
        const short transmitFailure = 54;
        const short transmitterFailure = 55;
        const short underlyingResourceUnavailable = 56;
        const short versionMismatch = 57;

}; // end of ProbableCauseConst module


/** This module contains the constant values defined for the
SecurityAlarmCause UID.  These values were borrowed from
X.721. */

module SecurityAlarmCauseConst {
const string moduleName = "ITU_X721Const::SecurityAlarmCauseConst";

        const short authenticationFailure = 1;
        const short breachOfConfidentiality = 2;
        const short cableTamper = 3;
        const short delayedInformation = 4;
        const short denialOfService = 5;
        const short duplicateInformation = 6;
        const short informationMissing = 7;
        const short informationModificationDetected = 8;
        const short informationOutOfSequence = 9;
        const short intrusionDetection = 10;
        const short keyExpired = 11;
        const short nonRepudiationFailure = 12;
        const short outOfHoursActivity = 13;
        const short outOfService = 14;
        const short proceduralError = 15;
        const short unauthorizedAccessAttempt = 16;
        const short unexpectedInformation = 17;
        const short unspecifiedReason = 18;

}; // end of SecurityAlarmCauseConst module


/** This module contains the constant values defined for the
Object Error UID.  These values were borrowed from cmip.asn1
and many may not be needed. */

module ObjectErrorConst {
const string moduleName = "ITU_X721Const::ObjectErrorConst";

        const short accessDenied = 2;
        const short classInstanceConflict = 19;
        const short complexityLimitation = 20;
        const short duplicateManagedObjectInstance = 11;
        const short getListError = 7;
        const short invalidArgumentValue = 15;
        const short invalidAttributeValue = 6;
        const short invalidFilter = 4;
        const short invalidObjectInstance = 17;
        const short invalidOperation = 24;
        const short invalidScope = 16;
        const short missingAttributeValue = 18;
        const short mistypedOperation = 21;
        const short noSuchAction = 9;
        const short noSuchArgument = 14;
        const short noSuchAttribute = 5;
        const short noSuchEventType = 13;
        const short noSuchInvokeID = 22;
```

```
        const short noSuchObjectClass = 0;
        const short noSuchObjectInstance = 1;
        const short noSuchReferenceObject = 12;
        const short operationCancelled = 23;
        const short processingFailure = 10;
        const short setListError = 8;
        const short syncNotSupported = 3;

}; // end of ObjectErrorConst module

}; // end of ITU_X721Const module

#endif // end of ifndef _ITU_X721Const_idl_
```

B.2 Generic Network Management IDL Definitions (NetMgmt.idl)

```
#ifndef _NetMgmt_idl_
#define _NetMgmt_idl_
#include <CosNaming.idl>
#include <CosNotifyChannelAdmin.idl>
#include "ITU_X721Const.idl"
#include "ITU_M3100Const.idl"

/** This IDL code is intended to be stored in a file named "NetMgmt.idl" located in the
search path of your IDL compiler.  Most comments in this file are formatted to be parsed by
an IDL-to-HTML converter such as idldoc or orbacus hidl. This module provides the fundamental
capabilities for implementing network management interfaces and defines the "managed object"
interface. The interfaces below are modeled after the managed object specifications found in
the CMIP specification document X.721. */

module NetMgmt {
const string moduleName = "NetMgmt";

// Types imported from CosNaming
typedef CosNaming::Name Name;

// Types imported from CosNotifyChannelAdmin
typedef CosNotifyChannelAdmin::EventChannel EventChannel;

interface ManagedObject;          // forward declaration
interface Portal;                 // forward declaration
interface NameIterator;                   // forward declaration
interface ManagedObjectFactory;           // forward declaration
interface Notifications;          // forward declaration

/** MO is shorthand for Managed Object.  CORBA uses object references of type
"object" to identify objects.  These are used instead of ASN.1 object
instances.  For network management interfaces, all objects will inherit from
the "ManagedObject" interface.  */

typedef ManagedObject MO;

/** MO List is a list of MO references. */

typedef sequence<MO> MOList;

/** MOListList is a two-dimensional list (list of lists) of managed objects. */

typedef sequence<MOList> MOListList;

/** ScopedName is just a string. */

typedef string ScopedName;

/** Scoped Name Lists are simply lists of Scoped Names. */

typedef sequence<ScopedName> ScopedNameList;
```

```
/** In CORBA, strings containing scoped names are used to identify object
classes (actually, "interfaces"). */

typedef ScopedName ObjectClass;

/** Object Class List is a list of object classes */

typedef sequence<ObjectClass> ObjectClassList;

/** Many times interface specifications need to define standard values to be
passed across the interface.  Also, often the scheme used to define these
values needs to be extensible as new interfaces are subclassed, so
enumerations don't work well.  CMIP uses OIDs, strings of numbers that are
often appended, in standards.  To serve this purpose, the Unique ID is used.
It consists of two parts, a string containing a scoped module name, and an
integer value defined as a constant within that module.  These UIDs, and the
ObjectClass type defined above, replace ASN.1 OIDs.  It is expected that each
module will contain a constant string named "moduleName" that contains the name
of the module for error-free use by the programmer.  A null module name will
indicate a null value for the UID.

Code to interpret a UID might look like the following code snippet:

UID      pc;      // probable cause
...
if (pc.moduleName == ITU_X721::ProbableCauseConst::moduleName) // string compare
         switch (pc.value) {
         case ITU_X721::ProbableCauseConst::adapterError:
         ...
         case ITU_X721::ProbableCauseConst::applicationSubsystemFailure:
         ...
         case ITU_X721::ProbableCauseConst::bandwidthReduced:
         ...
         }
else if (pc.moduleName == BasicNet::ProbableCauseConst::moduleName)
         switch (pc.value) {
         ...
         }

@member moduleName        The scoped module name where values are defined.
@member value             The value defined as a constant within the module.
*/

struct UID {
         string moduleName;        // The scoped module name defining the value
         short value;              // defined as a constant within the module
};
typedef sequence<UID> UIDList;

/** The Managed Object ID is a structure containing both the name of and
reference to a managed object.  It is felt that passing both of these together
across an interface might help to reduce lookups in the name service or calls
to the object to get its name.  More importantly, it enables the definition of
interfaces that can be implemented as either coarse or fine.  In the coarse
implementations, there will be one instance per class but there will be name
bindings in the naming service as if there were fine-grained objects.  All of
the bindings for a single class of objects will reference the same single
instance.  All operations on the objects contain the name of the object, so
that the singleton objects can identify the true object being acted upon.

In fine-grained implementations, the name bindings reference separate objects.
In these cases including the object's name in the operations is redundant, but
a reasonable trade-off.
@member ref      A reference to the object.  Will be null for a null-valued ID.
@member name     The fully-qualified Cos name of the object.  Will be null for
                 a null-valued ID.
*/
```

```
struct MOID {
        MO      ref;
        Name    moName;
};

/** MOIDList is a list of MOIDs. */

typedef sequence<MOID> MOIDList;

/** MOIDListList is a two-dimensional list of MOIDs. */

typedef sequence<MOIDList> MOIDListList;


/* The following state objects are used in many interfaces and parallel the
state objects in CMIP standards. */

/** Administrative State is read/write.  A "locked" object is usually one that
may not be changed or one which is not providing service.  Setting the
Admininstrative State of an object to "shuttingDown" begins the shutdown
process for that object. */

enum AdministrativeState {locked, unlocked, shuttingDown};

/** Operational State is read only.  It simply reports the current capability
of the object to provide service. */

enum OperationalState {disabled, enabled};

/** Usage state is read only.  If "idle," the resource is completely unused.
If "busy," the total capacity of the resource is in use.  "Active" is in
between.  */

enum UsageState {idle, active, busy};

/** Management Extension is a structure for flexibly reporting information.
It is typically used in the Additional Information field of notifications.
@see href="#AdditionalInformation" AdditionalInformation
@member id              identifies the type of information
@member significance    not sure what this is for - from X.721
@member any             contains the actual information, type will depend on
                        the value of the id member.
*/

struct ManagementExtension {
        UID     id;             // identifies the type of info
        boolean significance;   // not sure what this is for
        any     info;           // type will depend on id
};

/** Additional Information is a flexible way to report information that does
not fit into the structure of a notification.  It contains a sequence of a
structure called "Management Extension". */

typedef sequence<ManagementExtension> AdditionalInformation;

/** An Attribute Value structure is used to set or retrieve an attribute value
generically, such as in a batch mode.  This is complicated somewhat by the fact
that none of the network management interfaces are expected to define CORBA
attributes, but instead CORBA operations to get or set attribute values.  So,
a convention similar to the way CORBA attribute accesses are mapped to
programming languages is adopted, where the retrieval of an attribute value
is done with an operation named get<attribute_name> and setting is done with
set<attribute_name>.  In this structure, the string attributeName will contain
the <attribute_name> from the object's attribute access operations.  The value
part is used to convey the attribute's value.
@member attributeName    the name of an operation minus the "get" or "set"
```

```
@member value             containes the value of the attribute, type will depend
                on the attributeName. */

struct AttributeValue {
        string   attributeName;
        any      value;           // type will depend on the attribute
};

/** Attribute Value Lists are used to set or retrieve attributes generically,
in a batch mode. */

typedef sequence<AttributeValue> AttributeList;

/** An Attribute Value Change structure is used in a notification to report an
attribute that has been changed.
@see href="#AttributeValue" AttributeValue
@member attributeName     the name of an operation minus the "get" or "set"
@member oldValue the old value, type will depend on the attributeName
@member newValue the new value, type will depend on the attribute Name.*/

struct AttributeValueChange {
        string           attributeName;
        any              oldValue;        // type will depend on the attribute
        any              newValue;        // type will depend on the attribute
};

/** An Attribute Change List is used to report the attributes that have been
changed in an attribute value change notification. */

typedef sequence<AttributeValueChange> AttributeChangeList;

/** A Correlated Notification is identified by the object that emitted the
notification and the notification id.  Both are included in case the
Notification IDs are not unique across objects.
@member source   Reference to object that emitted the correlated notification
@member notifID  ID of the correlated notification. */

struct CorrelatedNotification {
        MOID             source;
        unsigned long    notifID;
};

/** Correlated Notifications are lists of Correlated Notification structures. */

typedef sequence<CorrelatedNotification> CorrelatedNotifications;

/** Generalized time is a basic ASN.1 type.  It is usually represented as a
string in computing languages but it has certain, parseable formats. The 3
possible forms are:

1.Local time only. "YYYYMMDDHHMMSS.fff", where the optional fff is
accurate to three decimal places.
2.Universal time (UTC time) only. "YYYYMMDDHHMMSS.fffZ".
3.Difference between local and UTC times. "YYYYMMDDHHMMSS.fff+-HHMM".

The options for representing this in IDL seem to be either a string or the UtcT
structure from the CORBA Time Service.  Because UtcT does not seem to make it
possible to differentiate a local time (option 1 above) from a universal time
(option 2 above), a string will be used.  */

typedef string GeneralizedTime;

/** External Time is generalized time. */

typedef GeneralizedTime ExternalTime;

/** PerceivedSeverity reports the severity of an alarm. "Indeterminate" is
used when it is not possible to assign one of the other values */
```

```
enum PerceivedSeverity{indeterminate, critical, major, minor, warning, cleared};

/** ProbableCause, in CMIP standards, may be either an integer or GDMO OID, a
dot-notation string.  The UID type is used instead. */

typedef UID ProbableCause;

/** Proposed Repair Actions are lists of unique identifiers. */

typedef UIDList ProposedRepairActions;

/** Security Alarm Causes are unique identifiers. */

typedef UID SecurityAlarmCause;

/** Security Alarm Detector can indicate either a mechanism or a specific
object.  According to X.721 a choice is made between one or the other, though
it is not clear why.  (Actually, X.721 adds a third choice for an AE-title
which has no equivalent here.)  Unless otherwise indicated, then, at most one
of the members will be non-null.  Two nulls may be sent if the managed system
does not support this property.  May want to consider adding Object Class.
@member mechanismthe scheme or function detecting the alarm, may be null
@member object            the object detecting the alarm, may be null */

struct SecurityAlarmDetector {
        UID             mechanism;       // may be null
        MOID            managedObject;            // may be null
};

/** Service User
@member id       the id of the service user
@member details  details about the service user, type will depend on id  */

struct ServiceUser {
        UID     id;
        any     details; // value will depend on id
};

/** Service Providers share the same representation as Service Users. */

typedef ServiceUser ServiceProvider;

/** Source Indicator is used in many notifications. It identifies whether the
notification is a result of a management operation or something that occurred
on the managed system.  */

enum SourceIndicator {resourceOperation, managementOperation, unknown};

/** Specific Problems are lists of unique identifiers. */

typedef UIDList SpecificProblems;

/** The following three typedefs are used in interface Portal. */

typedef Name NameType;

typedef sequence<NameType> NameSetType;

typedef string KindType;

/** Threshold indication describes if the threshold crossed was an upper
threshold or a lower threshold. */

enum ThresholdIndication {upper, lower};

/** Threshold Information indicates some guage or counter attribute passed a
set threshold. The structure differs from X.721 some to simplify the syntax.
```

```
@member attributeID       identifies the attribute that crossed the threshold.
                          Actually, it is an operation name on an interface minus
                          the "get" or "set".  The interface on which the
                          operation is defined is included elsewhere in the
                          notification as ObjectClass.  A Null value indicates
                          the entire structure is null.
@member observedValue     attributes that are of type integer will be converted
                          to floats
@member indication
@member high              high and low members are for multi-level thresholds.
                          for single-level thresholds they will be equal
@member armTime           may be null */

struct ThresholdInfo {
        string                  attributeID;
        float                   observedValue;
        ThresholdIndication     indication;
        float                   high;
        float                   low;
        ExternalTime            armTIme;
};

/** TrendIndication.  The "unknown" value was added to handle cases where this
parameter is optional. */

enum TrendIndication {lessSevere, noChange, moreSevere, unknownTrend};


/** The Alarm Info structure is used to contain information in Alarm
notifications.
@member eventTime         Managed system's current time.
@member source                  Object emitting notification.
@member sourceClass             Class of source object.
@member notificationIdentifier  A unique identifier for this notification
                                (optional in X.721 but not here)
@member correlatedNotifications List of correlated notifications.  Optional.
                                Null if not supported.
@member probableCause
@member specificProblems  Optional.  Null if not supported.
@member perceivedSeverity
@member backedUpStatus          "True" if backed up (optional in X.721 but not
                                here).  If object is unsure, value should be
                                "false".
@member backUpObject            Will be null if backedUpStatus is "false"
@member trendIndication         Optional.  See type for details.
@member thresholdInfo           Optional.  See type for details.
@member stateChangeDefinition   Optional.  Null if not supported.
@member monitoredAttributes     Optional.  Null if not supported.
@member proposedRepairActions   Optional.  Null if not supported.
@member additionalText          Text message. Optional.  Null if not supported.
@member additionalInfo          Optional. Null if not supported.
*/

struct AlarmInfo {
ExternalTime            eventTime;
MOID                    source;
ObjectClass             sourceClass;
unsigned long           notificationIdentifier;
CorrelatedNotifications correlatedNotifications;
ProbableCause           probableCause;
SpecificProblems specificProblems;
PerceivedSeverityperceivedSeverity;
boolean                 backedUpStatus;
MOID                    backUpObject;
TrendIndication         trendIndication;
ThresholdInfo           thresholdInfo;
AttributeChangeList     stateChangeDefinition;
AttributeList           monitoredAttributes;
```

```
ProposedRepairActions     proposedRepairActions;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** The Attribute Value Change Info structure is used to contain information in
Attribute Value Change notifications. (X.721 includes an attribute identifier
list that does not seem necessary.)
@member eventTime         Managed system's current time
@member source                    Object emitting notification
@member sourceClass               Class of source object
@member notificationIdentifier    A unique identifier for this notification
                                  (optional in X.721 but not here)
@member correlatedNotifications   List of correlated notifications.  Optional.
                                  Null if not supported.
@member sourceIndicator           Cause of event.  Optional.  Use "unknown" if
                                  not supported.
@member attributeChanges  Changed attributes
@member additionalText            Text message.  Optional.  Null if not supported.
@member additionalInfo            Optional.  Null if not supported.
*/

struct AttributeValueChangeInfo {
ExternalTime              eventTime;
MOID                      source;
ObjectClass               sourceClass;
unsigned long             notificationIdentifier;
CorrelatedNotifications correlatedNotifications;
SourceIndicator           sourceIndicator;
AttributeChangeList       attributeChanges;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** The Object Info structure is used to contain information in Object
Creation and Deletion notifications.  In Object Creation notifications the
"source" parameter should be the new object, not the factory.
@member eventTime         Managed system's current time
@member source                    Object emitting notification
@member sourceClass               Class of source object
@member notificationIdentifier    A unique identifier for this notification
                                  (optional in X.721 but not here)
@member correlatedNotifications   List of correlated notifications.  Optional.
                                  Null if not supported.
@member sourceIndicator           Cause of event.  Optional. Use "unknown" if
                                  not supported.
@member attributeList             Attribute values. Optional. Null if not
                                  supported
@member additionalText            Text message.  Optional.  Null if not supported.
@member additionalInfo            Optional.  Null if not supported.
*/

struct ObjectInfo {
ExternalTime              eventTime;
MOID                      source;
ObjectClass               sourceClass;
unsigned long             notificationIdentifier;
CorrelatedNotifications correlatedNotifications;
SourceIndicator           sourceIndicator;
AttributeList             attributeList;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** The Relationship Change Info structure is used to contain information in
Relationship Change notifications.  (X.721 includes an attribute
identifier list that does not seem necessary.)
@member eventTime         Managed system's current time
```

```
@member source                    Object emitting notification
@member sourceClass               Class of source object
@member notificationIdentifier    A unique identifier for this notification
                                  (optional in X.721 but not here)
@member correlatedNotifications   List of correlated notifications.  Optional.
                                  Null if not supported.
@member sourceIndicator           Cause of event.  Optional. Use "unknown" if
                                  not supported.
@member relationshipChanges       Changed relationship attributes
@member additionalText            Text message.  Optional.  Null if not supported.
@member additionalInfo            Optional.  Null if not supported.
*/

struct RelationshipChangeInfo {
ExternalTime              eventTime;
MOID                      source;
ObjectClass               sourceClass;
unsigned long             notificationIdentifier;
CorrelatedNotifications correlatedNotifications;
SourceIndicator           sourceIndicator;
AttributeChangeList       relationshipChanges;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** The Security Alarm Info structure is used to contain information in
Security Alarm notifications.
@member eventTime         Managed system's current time
@member source                    Object emitting notification
@member sourceClass               Class of source object
@member notificationIdentifier    A unique identifier for this notification
                                  (optional in X.721 but not here)
@member correlatedNotifications   List of correlated notifications.  Optional.
                                  Null if not supported.
@member securityAlarmCause
@member securityAlarmSeverity     Clears allowed.  X.721 appears to restrict the
                                  "cleared" value on this alarm but clears
                                  are allowed in this specification.
@member securityAlarmDetector
@member serviceUser
@member serviceProvider
@member additionalText            Text message.  Optional.  Null if not supported.
@member additionalInfo            Optional.  Null if not supported.
*/

struct SecurityAlarmInfo {
ExternalTime              eventTime;
MOID                      source;
ObjectClass               sourceClass;
unsigned long             notificationIdentifier;
CorrelatedNotifications correlatedNotifications;
SecurityAlarmCause        securityAlarmCause;
PerceivedSeveritysecurityAlarmSeverity;
SecurityAlarmDetector     securityAlarmDetector;
ServiceUser               serviceUser;
ServiceProvider           serviceProvider;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** The State Change Info structure is used to contain information in or from
State Change notifications.  (X.721 includes an attribute identifier list that
does not seem necessary.)
@member eventTime         Managed system's current time
@member source                    Object emitting notification
@member sourceClass               Class of source object
@member notificationIdentifier    A unique identifier for this notification
                                  (optional in X.721 but not here)
```

```
@member correlatedNotifications   List of correlated notifications.  Optional.
                                  Null if not supported.
@member sourceIndicator           Cause of event.  Optional. Use "unknown" if
                                  not supported.
@member stateChanges              Changed state attributes
@member additionalText            Text message.  Optional.  Null if not supported.
@member additionalInfo            Optional.  Null if not supported.
*/

struct StateChangeInfo {
ExternalTime              eventTime;
MOID                      source;
ObjectClass               sourceClass;
unsigned long             notificationIdentifier;
CorrelatedNotifications   correlatedNotifications;
SourceIndicator           sourceIndicator;
AttributeChangeList       stateChanges;
string                    additionalText;
AdditionalInformation     additionalInfo;
};

/** Exceptions */

/** Object Error attributes identify the type of error that an object has
experienced and are represented by UIDs. */

typedef UID ObjectError;

/** An ObjectFailure exception means the object implementing the interface
could not process the requested operation.

For now, this exception only returns an error UID and a string explanation.
CMIP standards allow one of the error values that are defined for the error
UID, "processingFailure," to also include additional information.  This
additional information takes the form of an "errorID" along with a parameter
of type any that is defined by the value of the errorID.  I have not found
where these error IDs are defined.  It may be necessary to modify the
ObjectFailure exception to include a structure for this additional information.
Perhaps the "AdditionalInformation" type would work. */

exception ObjectFailure {ObjectError error; string explanation;};

/** A NotSupported exception means the object implementing the interface does
not support the operation. These exception are not throwable on every
operation, only those considered "conditional." */

exception NotSupported {};

/** A ContainedObjects exception means the managed system tried to delete an
object but could not because the object contains other objects and was not
deleted with "deleteContainedObjects" asserted. */

exception ContainedObjects {};

/** A DeleteNotAllowed exception means the managing system tried to delete an
object that it is not allowed to delete. */

exception DeleteNotAllowed {};

/** A DuplicateItem exception means an attempt was made to add a duplicate item
to a list. */

exception DuplicateItem {any item;};

/** An ItemNotFound exception means an attempt was made to access an item that
could not be found on the list.
@param item      the item that could not be found.  Type will depend on the type
of the list submitted to the operation.  */
```

```
exception ItemNotFound {any item;};

/** A DuplicateName exception means the managed system tried to create an
object with a name matching that of an existing object contained by the same
object under which the new object was to be created. */

exception DuplicateName {};

/** An invalid ID exception means the client included an invalid object ID in
an operation.
@param ID        the invalid id */

exception InvalidID {MOID id;};

/** An out of range exception means the client included a parameter with
a value outside the range acceptable for the operation.
*/

exception OutOfRange {};

/**
A MaxMonitorPointsExceeded exception means that the client's request to monitor data at
specified points has exceeded this server's capacity server, in terms of the maximum number
of points that it (or its underlying NEs) can simultaneously monitor.
*/

exception MaxMonitorPointsExceeded {};

/** The Managed Object interface is intended to be the "top" interface from
which all other managed object interfaces inherit.  It is a central place to
specify basic functions which all managed objects are expected to support. */

interface ManagedObject {

/** This method returns the fully-qualified name for the object (interface).
This method is used rather than having a "getID" method defined for each
interface, as is done in CMIP specifications.  This will ensure that objects
have only a single operation to retrieve names when they are sub-classed.

The response is a sequence of name component structures, starting with a "root"
name define for the domain.  (That is, the name of the top-most managed object
on a particular system.)  The client may find the ancestors of this object by
removing components from the tail end of this sequence and performing a
resolve operation on the first part of the name.  */

Name getName()
        raises(ObjectFailure);

/** This method returns a pointer to the Notification Channel used by this
object when it is a producer.  Clients interested in receiving notifications
from this object may then subscribe to this service.

Since it looks like the OMG's Event Logging Service will subclass the
notification service, we probably want to make this a pointer to the event
logging service instead. */

EventChannel getEventChannel(in Name name)
        raises(ObjectFailure);

/** This method returns a list of all the notifications supported by this
instance. It is included to parallel the CMIP capability to retrieve the
packages supported by an instance. */

ScopedNameList getSupportedNotifications (in Name name)
        raises(ObjectFailure);

/** This method may be used to generically get a list of attribute values.
```

```
Any values passed in will be ignored and the values returned will be the
attributes' values.  If the list contains attributes not supported by the
instance those
attributes should be deleted from the list on return. */

void getAttributeList(in Name name, inout AttributeList list)
        raises(ObjectFailure, NotSupported, ItemNotFound);

/** This method may be used to generically get all of the attributes supported
by an instance. */

AttributeList getAllAttributes(in Name name)
        raises(ObjectFailure, NotSupported);

/** This method may be used to generically set a list of attribute values.  The
values passed in will be assigned to the attributes and also returned.  If
the list contains attributes not supported by the instance those attributes
should be deleted from the list on return. */

void setAttributeList(in Name name, inout AttributeList list)
        raises(ObjectFailure, NotSupported, ItemNotFound);

/** This method deletes the object.  If deleteContainedObjects is true, the
contained objects will also be deleted.  If it is not true and there are
contained objects, the ContainedObjects exception will be thrown and the object
will not be deleted. */

void delete(in Name name, in boolean deleteContainedObjects)
        raises(ObjectFailure, DeleteNotAllowed, ContainedObjects);

}; // end of ManagedObject interface


/** This interface defines the generic managed object factory interface.  It is
currently empty but is a place holder for capabilities that may need to be
implemented by all managed object factories.  One example is inheritance from
CosLifeCycle::GenericFactory. */

interface ManagedObjectFactory {

}; // end of ManagedObjectFactory interface


/** This interface contains the definitions of notifications emitted by many
managed objects.

The use of "typed" notifications is done here so that the notifications can be
documented in IDL and to support typed notifications for those manager and
managing systems that wish to use them.  Note that the OMG's Notification
Service supports both structured and typed notifications.  It is not clear if
implementations of the Notification Service will support translation between
them.  It is expected that the implementation agreement between the managing and
managed system will specify the use of structured or typed notifications.

Notification users wishing to use typed notifications need only support the
interfaces below.  Notification publishers and subscribers wishing to use
structured notifications based on the operations defined below should follow
these rules for constructing and reading the notification structure:

The domain_type string in the fixed header of the structure should be set to
"telecommunications".

The event_type string in the fixed header of the structure should be set to
the scoped name of the operation.  For example, for the Attribute Value Change
notification defined below this field would be
"ITU_X721::Notifications::attributeValueChange".

The event_name string in the fixed header of the structure should be null.
```

```
Optional header fields may be included to support features like Quality of
Service as appropriate.

Each parameter in the operation should be placed in a name-value pair in the
filterable body portion of the notification.  The fd_name string of this pair
shall be set to the name of the parameter and the type placed in the associated
fd_value will be the type specified for the parameter.  For example, for the
Attribute Value Change notification defined below a single name-value pair
would be placed in the filterable data portion of the event.  The fd_name
string of this pair would be set to "attributeValueInfo" and fd_value would
contain an AttributeValueInfo structure.

The remainder of the body of the notification (the unfilterable part) should
be null.

Unfortunately, typed notifications are mapped to notification structures
differently, so if one system wants to use typed notifications and the other
structured, the structured notification user must be aware of how the CORBA
Notification Service translates typed notifications to structured
notifications.  See the specification for details.  In short, however, each of
the parameters in the operations below will be converted into a name-value
pair in the filterable data protion of the structured notification.  Also, the
event_type field in the fixed header of the structured notification will be
set to the special value "%TYPED" and the domain_type field will be an empty
string.  Finally, a name-value pair will be added as the first element in the
filterable data portion of the notification with the name "operation".  The
value associated with this name will be a string with the value set to the
scoped name of the operation used to emit the notification
(e.g. ITU_X721::Notifications::attributeValueChange).

Also, structured notification publishers emitting notifications for typed
notifications users must include all of the parameters listed for each
operation in the filterable data portion of the notification.  This is because
if the translation to a typed notification is ambiguous, the notification
channel will not be able to deliver it.  While the translation of some
excluded parameters (such as excluded strings to null strings) may be
possible, others (such as enumerated types) are not.  Thus, all parameters
must be included.

Parameters named "operation" should be avoided in notification operations to
support the use of typed notifications.  While the notification channel should
be able to differentiate the real parameter from the one added based on their
positions in the filterable data list, it could have an impact on filtering as
the default filtering language does not have a way to differentiate parameters
based on position.

Because the scoped operation name is placed in either the event_type string
(when structured notifications are used) or a filterable body name-value pair
with the name "operation" (when typed notifications are used), there is no
"event type" parameter explicitly included in any of the notification
operations defined below.  */

interface Notifications {

/** An Attribute Value Change notification is used to report changes to the
attributes of an object such as addition or deletion of members to one or more
set-valued attributes and replacement of the value of one or more attributes.
@param attributeValueChangeInfo    structure containing the notification info
*/

oneway void attributeValueChange (
in AttributeValueChangeInfo       attributeValueChangeInfo
);

/** A Communications Alarm notification is used to report when an object
detects a communications error.
@param alarmInfo structure containing the notification info
```

```
*/

oneway void communicationsAlarm (
        in AlarmInfo     alarmInfo
);

/** An Environmental Alarm notification is used to report a problem in the
environment.
@param alarmInfo structure containing the notification info
*/

oneway void environmentalAlarm (
        in AlarmInfo     alarmInfo
);

/** An Equipment Alarm notification is used to report a failure in the
equipment.
@param alarmInfo structure containing the notification info
*/

oneway void equipmentAlarm (
        in AlarmInfo     alarmInfo
);

/** An Integrity Violation notifications is used to report that a potential
interruption in information flow has occurred such that information may have
been illegally modified, inserted or deleted.
@param securityAlarmInfo  structure containing the notification info
*/

oneway void integrityViolation (
        in SecurityAlarmInfo      securityAlarmInfo
);

/** An Object Creation notification is used to report the creation of a managed
object to another open system.
@param objectInfostructure containing the notification info
*/

oneway void objectCreation (
        in ObjectInfo    objectInfo
);

/** An Object Deletion notification is used to report the deletion of a managed
object.
@param objectInfostructure containing the notification info
*/

oneway void objectDeletion (
        in ObjectInfo    objectInfo
);

/** An Operational Violation notification is used to report that the provision
of the requested service was not possible due to the unavailability,
malfunction or incorrect invocation of the service.
@param securityAlarmInfo  structure containing the notification info
*/

oneway void operationalViolation (
        in SecurityAlarmInfo      securityAlarmInfo
);

/** A Physical Violation notification is used to report that a physical
resource has been violated in a way that indicates a potential security attack.
@param securityAlarmInfo  structure containing the notification info
*/

oneway void physicalViolation (
```

```
        in SecurityAlarmInfo      securityAlarmInfo
);

/** A Processing Error Alarm notification is used to report a processing
failure in a managed object.
@param alarmInfo structure containing the notification info
*/

oneway void processingErrorAlarm (
        in AlarmInfo     alarmInfo
);

/** A Quality of Service Alarm notification is used to report a failure in the
quality of service of the managed object.
@param alarmInfo structure containing the notification info
*/

oneway void qualityOfServiceAlarm (
        in AlarmInfo     alarmInfo
);

/** A Relationship Change notification is used to report the change in the
value of one or more relationship attributes of a managed object, that result
through either internal operation of the managed object or via management
operation.
@param relationshipChangeInfo     structure containing the notification info
*/

oneway void relationshipChange (
        in RelationshipChangeInfo relationshipChangeInfo
);

/** A Security Service Or Mechanism Violation notification is used to report
that a security attack has been detected by a security service or mechanism.
@param securityAlarmInfo  structure containing the notification info
*/

oneway void securityServiceOrMechanismViolation (
        in SecurityAlarmInfo      securityAlarmInfo
);

/** A State Change notification is used to report the change in the the value
of one or more state attributes of a managed object, that result through either
internal operation of the managed object or via management operation.
@param stateChangeInfo    structure containing the notification info
*/

oneway void stateChange (
        in StateChangeInfo        stateChangeInfo
);

/** A Time Domain Violation notification is used to report that an event has
occurred at an unexpected or prohibited time.
@param securityAlarmInfo  structure containing the notification info
*/

oneway void timeDomainViolation (
        in SecurityAlarmInfo      securityAlarmInfo
);

}; // end of Notifications interface

interface Portal
{
/**
The following defines extra functions needed by portals to accomodate containment
functionality for light objects.
The SuperiorGet operation returns the name of the object containing the current object.  If
```

```
the current object is at the local root naming context, an empty string is returned.
*/
        NameType superiorGet
                (in NameType name)
                raises (ObjectFailure, NotSupported);
/**
The ContainedSubordinatesGet operation returns a list of names of the objects that are
contained within the current object.  The kind parameter is optional.
*/
        NameSetType containedSubordinatesGet
                (in NameType name,
                in unsigned long howMany,
                in KindType kind,
                out NameIterator niterator)
                raises (ObjectFailure, NotSupported);
};

interface NameIterator : NetMgmt::ManagedObject
{
        boolean nextN
                (in unsigned long howMany,
                out NameSetType nameSet);
};

}; // end of NetMgmt module

#endif // end of ifndef _NetMgmt_idl_
```

B.3 Generic Network Information Model Constant IDL Definitions (ITU_M3100Const.idl)

```
#ifndef _ITU_M3100Const_idl_
#define _ITU_M3100Const_idl_

module ITU_M3100Const {
const string moduleName = "ITU_M3100Const";

/** This module contains constant values identifying information
elements included in the Additional Information parameters of
notifications. */

module AdditionalInformationConst {
const string moduleName = "ITU_M3100::AdditionalInformationConst";

const short alarmEffectOnService = 1;
const short suspectObjectList = 2;
const short userLabel = 3;

}; // end of AdditionalInformationConst module


/** This module contains the constant values defined for the
CharacteristicInfo UID.  These values were borrowed from M.3100. */

module CharacteristicInfoConst {
const string moduleName = "ITU_M3100::CharacteristicInfoConst";

/** opticalSPITTP* object instances with stmLevel attribute = 1 */
const short opticalSTM1SPICI = 1;
/** opticalSPITTP* object instances with stmLevel attribute = 4 */
const short opticalSTM4SPICI = 2;
/** opticalSPITTP* object instances with stmLevel attribute = 16 */
const short opticalSTM16SPICI = 3;
/** electricalSPITTP* object instances with stmLevel attribute = 1 */
const short electricalSTM1SPICI = 4;
/** rsCTP* object instances with stmLevel attribute = 1 */
const short rsSTM1SPICI = 5;
```

```
/** rsCTP* object instances with stmLevel attribute = 4 */
const short rsSTM4SPICI = 6;
/** rsCTP* object instances with stmLevel attribute = 16 */
const short rsSTM16SPICI = 7;
/** msCTP* object instances with stmLevel attribute = 1 */
const short msSTM1SPICI = 8;
/** msCTP* object instances with stmLevel attribute = 4 */
const short msSTM4SPICI = 9;
/** msCTP* object instances with stmLevel attribute = 16 */
const short msSTM16SPICI = 10;
const short au3TU3VC3CI = 11;
const short au4VC4CI = 12;
const short tu11VC11CI = 13;
const short tu12VC12CI = 14;
const short tu2VC2CI = 15;
const short tu12VC11CI = 16;
const short vpCI = 17;
const short vcCI = 18;
const short e0CI = 19;
const short e1CI = 20;
const short e2CI = 21;
const short e3CI = 22;
const short e4CI = 23;

}; // end of CharacteristicInfoConst module


/** This module contains the constant values defined for the General
Error Cause UID.  The values were borrowed from the M.3100 corrigendum
General Error Cause type definition. */

module GeneralErrorCauseConst {
const string moduleName = "ITU_M3100::GeneralErrorCauseConst";

/**  ObjectInIncompatibleState is used to specify that the object
is in a state provided.  */

const short objectInIncompatibleState = 1;

/** NoValidRelatedObject is used to specify related objects that
do not exist in the MIB.  */

const short noValidRelatedObject = 2;

/** InvolvedInOffering is used to identify object(s) that are
already involved in a conflicting service offering. */

const short involvedInOffering = 3;

/** ServiceNotSupported is used to indicate that the operation is
attempting to initiate a service that is not supported by the
equipment.  */

const short serviceNotSupported = 4;

/** ProvisioningOrderConflict is used to identify that a service
is  being provisioned in an order that is not supported by the
equipment.  */

const short provisioningOrderConflict = 5;

/** EquipmentFailure is used to indicate that an equipment failure
as  occured during the operation.  */

const short equipmentFailure = 6;

/** MaxNumberExceeded is used to indicate that requested create
operation cannot be completed as the maximum number of instances
```

```
are reached.  */

const short maxNumberExceeded = 7;

/** ContainedObjects is used to indicate that requested delete
operation cannot be completed as there are contained instances.
*/

const short containedObjects = 8;

}; // end of GeneralErrorCauseConst module


/** This module contains the constant values defined for the Information
Rate UID.  The values were borrowed from the M.3100 InformationRate type
definition, which defines Information Rate as an integer with the
following defined values. */

module InformationRateConst {
const string moduleName = "ITU_M3100::InformationRateConst";

const short ds1sf = 10;
const short ds1esf = 11;
const short zbtsi = 12;
const short tidm = 14;
const short cept1 = 20;
const short ds1c = 25;
const short ds2 = 30;
const short cept2 = 40;
const short ds3async = 50;
const short ds3sync = 51;
const short ds3cbit = 52;
const short ds3pbit = 53;
const short ds4 = 60;
const short ds4e = 65;
const short cept3 = 70;
const short vc11 = 80;
const short vc12 = 85;
const short vc2 = 90;
const short vc3 = 95;
const short vc4 = 100;
const short stm1 = 110;
const short stm4 = 120;
const short stm16 = 130;

}; // end of InformationRateConst module


/** This module contains the constant values defined for the
InformationTransferCapability UID.  These values were borrowed from
M.3100 (M.3100 defines this as an extensible enumerated type, which does
not transfer well to IDL.  So, the "UID" approach is used instead.) */

module InformationTransferCapabilityConst {
const string moduleName = "ITU_M3100::InformationTransferCapabilityConst";

const short speech = 0;
const short audio3pt1 = 1;
const short audio7 = 2;
const short audioComb = 3;
const short digitalRestricted56 = 4;
const short digitalUnrestricted64 = 5;

}; // end of the InformationTransferCapabilityConst module


/** This module contains the constant values defined for the Line Coding
UID.  The values were borrowed from the M.3100 LineCoding type
```

```
definition, which defines Line Coding as an integer with the following
defined values. */

module LineCodingConst {
const string moduleName = "ITU_M3100::LineCodingConst";

const short nrz = 0;
const short rz = 1;
const short diphase = 2;
const short bipolar = 3;
const short b6zs = 4;
const short b8zs = 5;
const short b3zs = 6;
const short ami = 7;
const short amizcs = 8;
const short hdb2 = 9;
const short hdb3 = 10;
const short cchan = 11;

}; // end of LineCodingConst module


/** This module contains the constant values defined for the Media Type
UID.  The values were borrowed from the M.3100 MediaType type
definition, which defines Media Type as an integer with the following
defined values. */

module MediaTypeConst {
const string moduleName = "ITU_M3100::MediaTypeConst";

const short twistedPairCopper = 0;
const short coaxial = 1;
const short singleModeFiber = 2;
const short multiModeFiber = 3;
const short radio = 4;
const short satellite = 5;

}; // end of MediaTypeConst module


/** This module contains the constant values defined for the
ProbableCause UID.  These values were borrowed from M.3100. */

module ProbableCauseConst {
const string moduleName = "ITU_M3100::ProbableCauseConst";

const short indeterminate = 0;

// The following are used with communications alarms.
const short aIS = 1 ;
const short callSetUpFailure = 2;
const short degradedSignal = 3;
const short farEndReceiverFailure = 4;
const short framingError = 5;
const short lossOfFrame = 6;
const short lossOfPointer = 7;
const short lossOfSignal = 8;
const short payloadTypeMismatch = 9;
const short transmissionError = 10;
const short remoteAlarmInterface = 11;
const short excessiveBER = 12;
const short pathTraceMismatch = 13;
const short unavailable = 14;
const short signalLabelMismatch = 15;
const short lossOfMultiFrame = 16;
const short receiveFailure = 17;
const short transmitFailure = 18;
const short modulationFailure = 19;
```

```
const short demodulationFailure = 20;
const short broadcastChannelFailure = 21;
const short connectionEstablishmentError = 22;
const short invalidMessageReceived = 23;
const short localNodeTransmissionError = 24;
const short remoteNodeTransmissionError = 25;
const short routingFailure = 26;

// Values 27-50 are reserved for communications alarm related
// probable causes

// The following are used with equipment alarms.
const short backplaneFailure = 51;
const short dataSetProblem = 52;
const short equipmentIdentifierDuplication = 53;
const short externalIFDeviceProblem = 54;
const short lineCardProblem = 55;
const short multiplexerProblem = 56;
const short neIdentifierDuplication = 57;
const short powerProblem = 58;
const short processorProblem = 59;
const short protectionPathFailure = 60;
const short receiverFailure = 61;
const short replaceableUnitMissing = 62;
const short replaceableUnitTypeMismatch = 63;
const short synchronizationSourceMismatch = 64;
const short terminalProblem = 65;
const short timingProblem = 66;
const short transmitterFailure = 67;
const short trunkCardProblem = 68;
const short replaceableUnitProblem = 69;
/** an equipment alarm to be issued if the system detects that the
real time clock has failed. */
const short realTimeClockFailure = 70;
const short antennaFailure = 71;
const short batteryChargingFailure = 72;
const short diskFailure = 73;
const short frequencyHoppingFailure = 74;
const short iODeviceError = 75;
const short lossOfSynchronisation = 76;
const short lossOfRedundancy = 77;
const short powerSupplyFailure = 78;
const short signalQualityEvaluationFailure = 79;
const short tranceiverFailure = 80;

// Values 81-100 are reserved for equipment alarm related
// probable causes.

// The following are used with environmental alarms.
const short airCompressorFailure = 101;
const short airConditioningFailure = 102;
const short airDryerFailure = 103;
const short batteryDischarging = 104;
const short batteryFailure = 105;
const short commercialPowerFailure = 106;
const short coolingFanFailure = 107;
const short engineFailure = 108;
const short fireDetectorFailure = 109;
const short fuseFailure = 110;
const short generatorFailure = 111;
const short lowBatteryThreshold = 112;
const short pumpFailure = 113;
const short rectifierFailure = 114;
const short rectifierHighVoltage = 115;
const short rectifierLowFVoltage = 116;
const short ventilationsSystemFailure = 117;
const short enclosureDoorOpen = 118;
const short explosiveGas = 119;
```

```
const short fire = 120;
const short flood = 121;
const short highHumidity = 122;
const short highTemperature = 123;
const short highWind = 124;
const short iceBuildUp = 125;
const short intrusionDetection = 126;
const short lowFuel = 127;
const short lowHumidity = 128;
const short lowCablePressure = 129;
const short lowTemperature = 130;
const short lowWater = 131;
const short smoke = 132;
const short toxicGas = 133;
const short coolingSystemFailure = 134;
const short externalEquipmentFailure = 135;
const short externalPointFailure = 136;

// Values 137-150 are reserved for environmental alarm related
// probable causes.

// The following are used with Processing error alarms.
const short storageCapacityProblem = 151;
const short memoryMismatch = 152;
const short corruptData = 153;
const short outOfCPUCycles = 154;
const short sfwrEnvironmentProblem = 155;
const short sfwrDownloadFailure = 156;

/** A processing error alarm to be issued if the system detects
that it has lost the time in the real time clock but  the clock
itself is working. This could happen e.g. during a power cut in a
small NE which does not have battery backup for the real time
clock. */

const short lossOfRealTime = 157;

/** A processing error alarm to be issued after the system has
reinitialised. This will indicate to the management systems that
the view they have of the managed system may no longer be valid.
Usage example: The managed system issues this alarm after a
reinitialization with severity warning to inform the management
system about the event. No clearing notification will be sent. */

const short reinitialized = 158;
const short applicationSubsystemFailure = 159;
const short configurationOrCustomisationError = 160;
const short databaseInconsistency = 161;
const short fileError = 162;
const short outOfMemory = 163;
const short softwareError = 164;
const short timeoutExpired = 165;
const short underlayingResourceUnavailable = 166;
const short versionMismatch = 167;

// Values 168-200 are reserved for processing error alarm related probable
// causes.

const short bandwidthReduced = 201;
const short congestion = 202;
const short excessiveErrorRate = 203;
const short excessiveResponseTime = 204;
const short excessiveRetransmissionRate = 205;
const short reducedLoggingCapability = 206;
const short systemResourcesOverload = 207;

}; // end of ProbableCauseConst module
```

```
/** This module contains the constant values defined for the
ProblemCause UID.  These values were borrowed from M.3100. */

module ProblemCauseConst {
const string moduleName = "ITU_M3100::ProblemCauseConst";

/* An additional value, unknown = -1, that is not in M.3100 was
added here because M.3100 defines problem cause as a choice
between an integer (as above) or null, for unknown.  Instead of
the null choice, unknown problems will be represented by an
integer value of -1. Since UID values are signed short, -1 is
acceptable. */

const short unknown = -1;
const short noSuchTpInstance = 0;
const short noSuchGtpInstance = 1;
const short noSuchTpPoolInstance = 2;
const short mismatchingTpInstance = 3;
const short mismatchingGtpInstance = 4;
const short partOfGtp = 5;
const short involvedInCrossConnection = 6;
const short memberOfTpPool = 7;
const short alreadyMemberOfGtp = 8;
const short noTpInTpPool = 9;
const short noMoreThanOneTpIsAllowed = 10;
const short noMoreThanTwoTpsAreAllowed = 11;

/** alreadyConnected is used to indicate the two termination
points requested to be cross-connected are already cross-connected
versus involvedInCrossConnection is used to indicate one or more
termination points are cross-connected but not to each other. */

const short alreadyConnected = 12;
const short notAlreadyConnected = 13;

}; // end of ProblemCauseConst module


/** This module contains the constant values defined for the
SignallingCapability UID.  These values were borrowed from M.3100
(M.3100 defines this as an extensible enumerated type, which does not
transfer well to IDL.  So, the "UID" approach is used instead.) */

module SignallingCapabilityConst {
const string moduleName = "ITU_M3100:SignallingCapabilityConst";

const short isup = 0;
const short isup92 = 1;
const short ccittNo5 = 2;
const short r2 = 3;
const short ccittNo6 = 4;
const short tup = 5;

}; // end of SignallingCapabilityConst module


}; // end of ITU_M3100Const module

#endif // end of ifndef _ITU_M3100Const_idl_
```

Appendix C: Interim Log Service IDL Definitions

The Telecom Log Service is not fully implemented by some ORB vendors, but its functionality is required for the approach used in this document. Use of the OMG Telecom Log Service is preferred to the use of this Appendix C. IDL is provided in this appendix for a LogManager module that can be used without a fully implemented log service. This IDL is intended for use in conjunction with the IDL provided elsewhere in this document.

```
#ifndef _atmf_logmanager_idl_
#define _atmf_logmanager_idl_

#include "NetMgmt.idl"

module atmf_logmanager
{
const string moduleName = "atmf_logmanager";

/**
Types imported from NetMgmt
*/
       typedef NetMgmt::AdministrativeState       AdministrativeState;
       typedef NetMgmt::MOID              MOID;
       typedef NetMgmt::MOIDList                  MOIDList;
       typedef NetMgmt::Name              NameType;
       typedef NetMgmt::OperationalState          OperationalState;
       typedef NetMgmt::GeneralizedTime           GeneralizedTime;

/**
Exceptions imported from NetMgmt are ItemNotFound, NotSupported, and ObjectFailure.
Interfaces imported from Netmgmt are ManagedObject.

Additional typedefs and structs are provided here.
*/

    typedef unsigned long AtmNEID;
    typedef unsigned short OldStateAttributeValue;
    typedef unsigned short NewStateAttributeValue;
    typedef boolean BackupStatus;
    typedef string SpecificProblems;
    typedef string BackupEntity;
    typedef string AdditionalText;
    typedef string ProposedRepairActions;
    typedef GeneralizedTime LoggingTime;
    typedef GeneralizedTime FirstLoggingTime;
    typedef GeneralizedTime LastLoggingTime;

    enum LogRecordType
    {
       managedEntityCreationLogRecord,
       managedEntityDeletionLogRecord,
       stateChangeLogRecord,
       attributeValueChangeLogRecord,
       alarmRecord
    };

    enum LogFullAction
    {
       wrap_around,
       halt
    };
/**
Enumerated parameter values are defined for the genericTroubleDescription corresponding to
the 38 values found in Table 2-7 of [2].
*/
     enum GenericTroubleDescription
```

```
    {
       gtdais,  //Alarm Indication Signal
       gtdlcd,  //Loss of Cell Delineation
       gtdlof,  //Loss of Frame
       gtdlop,  //Loss of Pointer
       gtdlos,  //Loss of Signal
       gtdptm,  //Payload Type Mismatch
       gtdte,   //Transmission Error
       gtdpath, //Path Trace Mismatch
       gtdrdi,  //Remote Defect indication
       gtdslm,  //Signal Label Mismatch
       gtdsrsu, //Signaling Route Set Unavailable
       gtdbpf,  //Back-plane Failure
       gtdcee,  //Call Establishment Error
       gtdcong, //Congestion
       gtdeidp, //External Interface Device Problem
       gtdlcp,  //Line Card Problem
       gtdmxp,   //Multiplexer Problem
       gtdpower, //Power Problem
       gtdproc,  //Processor Problem
       gtdppf,   //Protection Path Failure
       gtdrecv,  //Receiver Failure
       gtdunit,  //Replaceable Unit Missing
       gtdunitp, //Replaceable Unit Problem
       gtdunitt, //Replaceable Unit Type Mismatch
       gtdtime,  //Timing Problem
       gtdxmit,  //Transmitter failure
       gtdtrk,   //Trunk Card Problem
       gtdstor,  //Storage Capacity Problem
       gtdmemm,  //Memory Mismatch
       gtdcrpt,  //Corrupt Data
       gtdsofte, //Software Environment Problem
       gtdsoftd, //Software Download Failure
       gtdversn, //Version Mismatch
       gtdfan,   //Cooling Fan Failure
       gtddoor,  //Enclosure Door Open
       gtdfuse,  //Fuse Failure
       gtdtemp,  //High Temperature
       gtdven    //Vendor Specific
    };

    enum Severity
    {
       critical,
       major,
       minor,
       warning,
       indeterminate,
       cleared
    };

    enum StateAttributeType
    {
       operational_state,
       administrative_state
    };

    typedef NameType LogID;
    typedef NameType AlarmRecordID;
    typedef NameType ManagedEntityID;
    typedef NameType StateChangeLogRecordID;
    typedef NameType AttributeValueChangeLogRecordID;
    typedef NameType ManagedEntityCreationLogRecordID;
    typedef NameType ManagedEntityDeletionLogRecordID;

    struct Log
    {
       LogID logid;
```

```
      AtmNEID neid;
      AdministrativeState adminstate;
      string discriminatorConstruct;
      LogRecordType logrectype;
      LogFullAction fullaction;
      OperationalState operstate;
   };

   struct AlarmRecord
   {
      AlarmRecordID almrid;
      LoggingTime time;
      ManagedEntityID meid;
      GenericTroubleDescription trbldesc;
      string specificProblems;
      Severity sev;
      BackupStatus bcstatus;
      BackupEntity bcentity;
      AdditionalText addltext;
      ProposedRepairActions repairaction;
   };

   struct StateChangeLogRecord
   {
      StateChangeLogRecordID sclrid;
      LoggingTime time;
      ManagedEntityID meid;
      StateAttributeType attrtype;
      OldStateAttributeValue oldsav;
      NewStateAttributeValue newsav;
   };

   struct AttributeValueChangeLogRecord
   {
      AttributeValueChangeLogRecordID avclrid;
      LoggingTime time;
      ManagedEntityID meid;
      StateAttributeType attrtype;
      OldStateAttributeValue oldsav;
      NewStateAttributeValue newsav;
   };

   struct ManagedEntityCreationLogRecord
   {
      ManagedEntityCreationLogRecordID meclrid;
      LoggingTime time;
      ManagedEntityID meid;
   };

   struct ManagedEntityDeletionLogRecord
   {
      ManagedEntityDeletionLogRecordID medlrid;
      LoggingTime time;
      ManagedEntityID meid;
   };

   typedef sequence<LogID> LogIDList;
   typedef sequence<AlarmRecordID> AlarmRecordIDList;
   typedef sequence<StateChangeLogRecordID>
             StateChangeLogRecordIDList;
   typedef sequence<AttributeValueChangeLogRecordID>
             AttributeValueChangeLogRecordIDList;
   typedef sequence<ManagedEntityCreationLogRecordID>
             ManagedEntityCreationLogRecordIDList;
   typedef sequence<ManagedEntityDeletionLogRecordID>
             ManagedEntityDeletionLogRecordIDList;
   typedef sequence<AlarmRecord> AlarmRecordList;
   typedef sequence<StateChangeLogRecord> StateChangeLogRecordList;
```

```
    typedef sequence<AttributeValueChangeLogRecord>
              AttributeValueChangeLogRecordList;
    typedef sequence<ManagedEntityCreationLogRecord>
              ManagedEntityCreationLogRecordList;
    typedef sequence<ManagedEntityDeletionLogRecord>
              ManagedEntityDeletionLogRecordList;


interface LogManager : NetMgmt::ManagedObject
         {
         void getLogIDs
                 (out LogIDList logIDList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported);

         void getLog
                 (in LogID logID,
                 out Log log,
                 out AlarmRecordIDList arIDLIST,
                 out StateChangeLogRecordIDList sclrIDList,
                 out AttributeValueChangeLogRecordIDList avclrIDList,
                 out ManagedEntityCreationLogRecordIDList meclrIDList,
                 out ManagedEntityDeletionLogRecordIDList medlrIDList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                         NetMgmt::ItemNotFound);
/**
Each of the following five methods returns only those records with LoggingTime between
FirstLoggingTime and LastLoggingTime.  If FirstLoggingTime is not specified, then all records
with LoggingTime prior to LastLoggngTime are returned.  If LastLogginTime is not specified,
then all records with LoggingTime after FirstLoggingTime are returned.
*/
         void getAlarmRecords
                 (in AlarmRecordIDList arIDList,
                 in FirstLoggingTime firstlogtime,
                 in LastLoggingTime lastlogtime,
                 out AlarmRecordList arList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                         NetMgmt::ItemNotFound);

         void getStateChangeLogRecords
                 (in StateChangeLogRecordIDList sclrIDList,
                 in FirstLoggingTime firstlogtime,
                 in LastLoggingTime lastlogtime,
                 out StateChangeLogRecordList sclrList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                         NetMgmt::ItemNotFound);

         void getAttributeValueChangeLogRecords
                 (in AttributeValueChangeLogRecordIDList avclrIDList,
                 in FirstLoggingTime firstlogtime,
                 in LastLoggingTime lastlogtime,
                 out AttributeValueChangeLogRecordList avclrList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                         NetMgmt::ItemNotFound);

         void getManagedEntityCreationLogRecords
                 (in ManagedEntityCreationLogRecordIDList meclrIDList,
                 in FirstLoggingTime firstlogtime,
                 in LastLoggingTime lastlogtime,
                 out ManagedEntityCreationLogRecordList meclrList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
                         NetMgmt::ItemNotFound);

         void getManagedEntityDeletionLogRecords
                 (in ManagedEntityDeletionLogRecordIDList medlrIDList,
                 in FirstLoggingTime firstlogtime,
                 in LastLoggingTime lastlogtime,
                 out ManagedEntityDeletionLogRecordList medlrList)
                 raises (NetMgmt::ObjectFailure, NetMgmt::NotSupported,
```

```
                        NetMgmt::ItemNotFound);

        }; // interface LogManager

}; // end of module atmf_logmanager
#endif // _atmf_logmanager_idl_
```

The following table demonstrates the support of relevant M4 Network View Logical MIB objects, specifically alarmRecord and log, with the IDL objects provided in this appendix. It is similar in nature to Table 3-1.

Table C-1. M4 Network View Logical MIB to CORBA IDL Mapping for Appendix C

M4 Logical MIB Managed Entity	M4 Logical MIB Attribute/Operation	CORBA IDL Object	CORBA IDL Attribute/Operation	Comment
alarmRecord	Managed Entity ID	LogManager	AlarmRecordID	
	Logging Time		LoggingTime	
	Managed Entity		ManagedEntityID	
	Generic Trouble Description		GenericTroubleDescription	
	Specific Problems		specificProblems	
	Severity		Severity	
	Back-up Status		BackupStatus	
	Additional Text		AdditionalText	
	Proposed Repair Actions		ProposedRepairActions	
	No actions have been defined.		getAlarmRecords	
log	Managed Entity ID		LogID	
	No NE view attribute		AtmNEID	Needed for NW view
	Administrative State		AdministrativeState	
	Discriminator Construct		discriminatorConstruct	
	Log Record Types values are: Managed Entity Creation Log Record, Managed Entity Deletion Log Record, State Change Log Record, Attribute Value Change Log Record, Alarm Record		LogRecordTypes	Detailed by the following structs: ManagedEntityCreationLogRecord ManagedEntityDeletionLogRecord StateChangeLogRecord AttributeValueChangeLogRecord AlarmRecord
	Log Full Action		LogFullAction	
	Operational State		OperationalState	
	No actions have been defined.		getLogIDs getLog	

Appendix D: Object Naming Guidelines

This appendix provides suggested object naming guidelines for the M4 Network View CORBA IDL MIB. Such guidelines will promote EMS and NMS interoperability. The naming guidelines, in the table below, provide a name syntax for each CORBA object. The name syntax should be used as the id of the name component of the object.

Syntax:
"text" – text inside quotation should appear as is
| - means OR
<*type*> - identifies a type or category
[optional item] – indicates an optional item
{repetitive item} – indicates an item that may appear zero or more times

Defined Types used:
<*String*> ::= {any_character}
<*VP_or_VC* > ::= "VP" | "VC"
<*NetworkCTPId_or_LinkEndId*> ::= "NetworkCTPId=" | "LinkEndId="
<*Integer-VPI*> ::= <*Integer*>
<*null*> - indicates a null string

Note: asterisk "*" is used as a field delimiter.

Table D-1. Object Naming Guidelines

Object	**Name Syntax** (NameComponent.id for object)
ATMF_M4NW:AtmLink	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*" "Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AorZLinkEnd="<*String*>
ATMF_M4NW:AtmLinkEnd	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*" "Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AtmLinkEndId="<*String*>
ATMF_M4NW:AtmLinkEndPhy	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*

Object	**Name Syntax** (NameComponent.id for object)
	"Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AtmLinkEndId="<*String*>
ATMF_M4NW:AtmLND	"AtmLndLayer="<*VP_or_VC*>
ATMF_M4NW:AtmNetworkCTP	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*" "Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AtmLinkEndId="<*String*>"*" "ATMNetworkCTPId="<*Integer-VPI*>["/"<*Integer-VCI*>]
ATMF_M4NW:AtmNetworkTTP	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*" "Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AtmLinkEndId="<*String*>"*" "ATMNetworkTTPId="<*Integer-VPI*>["/"<*Integer-VCI*>]
ATMF_M4NW:AtmSNC	"AtmLndLayer="<*VP_or_VC*>"*" "ManagedElementId="<*String*>"*" "Bay="<*String*>"*" "Shelf="<*String*>"*" "Slot="<*String*>"*" "Port="<*String*>"*" "AorZLinkEndId="<*String*>"*" "AorZNetworkCTPId="<*Integer-VPI*>["/"<*Integer-VCI*>]
ATMF_M4NW:AtmSubnetwork	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{"Root" | <*String*>}
ATMF_M4NW:AtmNetworkAccessProfile	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmNetworkAccessProfile Factory	"EmsName="<*String*>"*" "FactoryName="<*String*>
ATMF_M4NW:AtmTrafficDesc	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmTrafficDescABR	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmTrafficDescCBR	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmTrafficDescVBR	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmTrafficDescUBR	"AtmLndLayer="<*VP_or_VC*>"*" "SubnetworkId="{<*String*> | <null>}"*" "ProfileName="<*String*>
ATMF_M4NW:AtmTrafficDescFactory	"EmsName="<*String*>"*" "FactoryName="<*String*>
ATMF_M4NW:Network	"NetworkName="<*String*>
ATMF_M4NW: LatestOccurrenceLog	"NetworkName="<*String*>"*" "LatestOccurrenceLogName="<*String*>
ATMF_M4NW:CurrentDataFactory	"EmsName="<*String*>"*" "FactoryName="<*String*>

Object	**Name Syntax** (NameComponent.id for object)
ATMF_M4NW:ThresholdData	"EmsName="<*String*>"*" "ThresholdDataName="<*String*>
ATMF_M4NW: CellProtocolMonCurrentData	"LinkEndId="<*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW: CellProtocolMonHistoryData	"LinkEndId="<*String*>"*" "HistoryDataName="<*String*>
ATMF_M4NW:AtmTrafficLoadCurrentData	<*NetworkCTPId_or_LinkEndId*><*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW:AtmTrafficLoadHistoryData	<*NetworkCTPId_or_LinkEndId*><*String*>"*" "HistoryDataName="<*String*>
ATMF_M4NW: CongDiscardCurrentData	"LinkEndId="<*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW: CongDiscardHistoryData	"LinkEndId="<*String*>"*" "HistoryDataName="<*String*>
ATMF_M4NW: TcAdaptProtMonCurrentData	"LinkEndId="<*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW: TcAdaptProtMonHistoryData	"LinkEndId="<*String*>"*" "HistoryDataName="<*String*>
ATMF_M4NW: UpcNpcDisagreementsCurrentData	"NetworkCTPId="<*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW: UpcNpcDisagreementsHistoryData	"NetworkCTPId="<*String*>"*" "HistoryDataName="<*String*>
ATMF_M4NW: PmOamCurrentData	"NetworkCTPId="<*String*>"*" "CurrentDataName="<*String*>
ATMF_M4NW: PmOamHistoryData	"NetworkCTPId="<*String*>"*" "HistoryDataName="<*String*>