Management Services Application Programming Interface

Document Version: 1.5

Document Status: DRAFT

Document Revision Date: 9 July 1990

Status of this Memo:

This draft document will be submitted to the RFC editor as a protocol specification. Distribution of this memo is unlimited. Please note that this document is an interim draft and will change as a result of further activities in the MSI working group. Please send comments to the author.

Issued by:

Oscar Newkerk Digital Equipment Corporation
newkerk@decwet.enet.dec.com

Thanks for their efforts in review and contibutions to:

- Bart Berger/3COM
- Brian Handspicker/DEC
- Dave Perkins/3COM
- Sudhanshu Verma/HP co-chair MSI Working Group

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 Intended Audience 1

1.2 Goals 1

1.3 Related Documents 2

CHAPTER 2 OVERVIEW 3

2.1 Association Services 3

2.2 Management Directive Services 3

2.3 Event Services 3

CHAPTER 3 MODEL IMPLEMENTATION 5

3.1 Structure 5

3.1.1 Management Application 5

3.1.2 Association Services 6

3.1.3 Directive Services 6

3.1.4 Event Subscription Services 6

3.1.5 Protocol Modules 7

CHAPTER 4 ASSOCIATION SERVICES 9

4.1 The Association Object 9

4.2 Operations on the Association Object 10

4.2.1 msi_create_association_object 10

4.2.2 msi_delete_association_object 10

CHAPTER 5 MANAGEMENT DIRECTIVE SERVICES 11

5.1 Directive Operations 12

5.1.1 msi_create_instance 12

5.1.2 msi_delete_instance 12

5.1.3 msi_get_attributes 12

5.1.4 msi_getnext_attribute 12

5.1.5 msi_set_attributes 12

5.1.6 msi_invoke_action 13

5.1.7 msi_read_reply 13

5.1.8 msi_send_reply 13

CHAPTER 6 EVENT SERVICES 15

6.1 Event Subscription Services 15
6.1.1 msi_create_subscription 15
6.1.2 msi_delete_subscription 16
6.1.3 msi_read_event 16

6.2 Event Reporting Services 16
6.2.1 msi_report_event 16

APPENDIX A ROUTINE DESCRIPTIONS FOR MANAGEMENT SERVICES API 17

A.1 Association Routines 18
A.1.1 msi_create_association_object 18
A.1.1.1 Description 18
A.1.1.2 Parameters 18
A.1.1.3 Association Attributes 18
A.1.1.4 Return Values 19
A.1.2 msi_delete_association_object 20
A.1.2.1 Description 20
A.1.2.2 Parameters 20
A.1.2.3 Return Values 20

A.2 Directive Routines 21
A.2.1 msi_create_instance 21
A.2.1.1 Description 21
A.2.1.2 Parameters 21
A.2.1.3 Return Values 22
A.2.2 msi_delete_instance 23
A.2.2.1 Description 23
A.2.2.2 Parameters 23
A.2.2.3 Return Values 24
A.2.3 msi_get_attributes 25
A.2.3.1 Description 25
A.2.3.2 Parameters 25
A.2.3.3 Return Values 26
A.2.4 msi_getnext_attribute 27
A.2.4.1 Description 27
A.2.4.2 Parameters 27
A.2.4.3 Return Values 28
A.2.5 msi_set_attribute 29
A.2.5.1 Description 29
A.2.5.2 Parameters 29
A.2.5.3 Return Values 30
A.2.6 msi_invoke_action 31
A.2.6.1 Description 31
A.2.6.2 Parameters 31
A.2.6.3 Return Values 32

A.2.7 msi_read_reply . . . 33
A.2.7.1 Description . . . 33
A.2.7.1.1 Values for the reply_type . . . 34
A.2.7.1.2 Status Returns . . . 34
A.2.7.2 Return Values . . . 36
A.2.8 msi_send_reply . . . 37
A.2.8.1 Description . . . 37
A.2.8.2 Parameters . . . 37
A.2.8.3 Return Values . . . 38
A.3 Event Routines . . . 39
A.3.1 msi_create_subscription . . . 39
A.3.1.1 Description . . . 39
A.3.1.2 Parameters . . . 39
A.3.1.3 Event Subscription Attributes . . . 39
A.3.1.4 Return Values . . . 40
A.3.2 msi_delete_subscription . . . 41
A.3.2.1 Description . . . 41
A.3.2.2 Parameters . . . 41
A.3.2.3 Return Values . . . 41
A.3.3 msi_read_event . . . 42
A.3.3.1 Description . . . 42
A.3.3.2 Parameters . . . 42
A.3.3.3 Return Values . . . 42
A.3.4 msi_report_event . . . 43
A.3.4.1 Description . . . 43
A.3.4.2 Parameters . . . 43
A.3.4.3 Return Values . . . 43
APPENDIX B DATA TYPES AND UTILITY ROUTINES . . . 45
B.1 Octet String Type . . . 46
B.2 ISO Scope Type . . . 47
B.3 Target Name . . . 48
B.4 Access Control Structure . . . 49
B.5 Object Identifier Type and Routines . . . 50
B.5.1 moss_create_oid . . . 51
B.5.1.1 Description . . . 51
B.5.1.2 Parameters . . . 51
B.5.2 moss_free_oid . . . 52
B.5.2.1 Description . . . 52
B.5.2.2 Parameters . . . 52
B.5.2.3 Return Values . . . 52
B.5.3 moss_get_oid_len . . . 53
B.5.3.1 Description . . . 53
B.5.3.2 Parameters . . . 53

B.5.4 moss_parse_oid 54
B.5.4.1 Description 54
B.5.4.2 Parameters 54
B.5.5 moss_compare_oid 55
B.5.5.1 Description 55
B.5.5.2 Parameters 55
B.5.5.3 Return Values 55
B.5.6 moss_oid_to_text 56
B.5.6.1 Description 56
B.5.6.2 Parameters 56
B.5.7 moss_text_to_oid 57
B.5.7.1 Description 57
B.5.7.2 Parameters 57
B.5.8 moss_concat_oids 58
B.5.8.1 Description 58
B.5.8.2 Parameters 58
B.6 Instance Name Type and Routines 59
B.6.1 moss_create_nonspec_name 60
B.6.1.1 Description 60
B.6.1.2 Parameters 60
B.6.2 moss_create_dist_name 61
B.6.2.1 Description 61
B.6.2.2 Parameters 61
B.6.3 moss_rnd_start_set 62
B.6.3.1 Description 62
B.6.3.2 Parameters 62
B.6.3.3 Return Values 62
B.6.4 moss_rdn_set_end 63
B.6.4.1 Description 63
B.6.4.2 Parameters 63
B.6.4.3 Return Values 63
B.6.5 moss_end_distinguished_name 64
B.6.5.1 Description 64
B.6.5.2 Parameters 64
B.6.5.3 Return Values 64
B.7 Directive Filter Type and Routines 65
B.7.1 moss_init_cmise_filter 66
B.7.1.1 Description 66
B.7.1.2 Parameters 66
B.7.2 moss_finish_filter 67
B.7.2.1 Description 67
B.7.2.2 Parameters 67

B.7.3 moss_free_cmise_filter 68
B.7.3.1 Description 68
B.7.3.2 Parameters 68
B.7.3.3 Return Values 68
B.7.4 moss_add_cmise_filter_element 69
B.7.4.1 Description 69
B.7.4.2 Parameters 69
B.7.4.3 Return Values 69
B.7.5 moss_start_cmise_or_set 70
B.7.5.1 Description 70
B.7.5.2 Parameters 70
B.7.5.3 Return Values 70
B.7.6 moss_start_cmise_and_set 71
B.7.6.1 Description 71
B.7.6.2 Parameters 71
B.7.6.3 Return Values 71
B.7.7 moss_start_cmise_not_set 72
B.7.7.1 Description 72
B.7.7.2 Parameters 72
B.7.7.3 Return Values 72
B.7.8 moss_end_cmise_andornot_set 73
B.7.8.1 Description 73
B.7.8.2 Parameters 73
B.7.8.3 Return Values 73

APPENDIX C AVL UTILITY ROUTINES 75

C.1 Introduction 75
C.1.1 AVL functions 76

C.2 AVL Utility Routine Descriptions 76
C.2.1 moss_avl_init 76
C.2.1.1 Description 76
C.2.1.2 Parameters 76
C.2.1.3 Return Values 76
C.2.2 moss_avl_free 77
C.2.2.1 Description 77
C.2.2.2 Parameters 77
C.2.2.3 Return Values 77
C.2.3 moss_avl_add 78
C.2.3.1 Description 78
C.2.3.2 Parameters 78
C.2.3.3 Return Values 78
C.2.4 moss_avl_reset 79
C.2.4.1 Description 79
C.2.4.2 Parameters 79
C.2.4.3 Return Values 79

C.2.5 moss_avl_point . . . 80
C.2.5.1 Description . . . 80
C.2.5.2 Parameters . . . 80
C.2.5.3 Return Values . . . 80
C.2.6 moss_avl_start_construct . . . 81
C.2.6.1 Description . . . 81
C.2.6.2 Parameters . . . 81
C.2.6.3 Return Values . . . 81
C.2.7 moss_avl_end_construct . . . 82
C.2.7.1 Description . . . 82
C.2.7.2 Parameters . . . 82
C.2.7.3 Return Values . . . 82
C.2.8 moss_avl_add_cons_field . . . 83
C.2.8.1 Description . . . 83
C.2.8.2 Parameters . . . 83
C.2.8.3 Return Values . . . 83
C.2.9 moss_avl_to_buf . . . 84
C.2.9.1 Description . . . 84
C.2.9.2 Parameters . . . 84
C.2.9.3 Return Values . . . 84
C.2.10 moss_avl_from_buf . . . 85
C.2.10.1 Description . . . 85
C.2.10.2 Parameters . . . 85
C.2.10.3 Return Values . . . 85

APPENDIX D EVENT MODEL REFERENCE . . . 87

D.1 Overview of the Model . . . 87

D.2 Definitions . . . 88

D.3 Managed System Requirements . . . 89

D.4 Event Reports . . . 89

D.5 Event Steams . . . 89

D.6 EVD events . . . 90

D.7 Sinks . . . 90

D.8 Filters . . . 90
D.8.1 Filter Structure and Searching . . . 91

D.9 Flow of Events . . . 92

GLOSSARY . . . 93

INDEX

FIGURES

1 Functional Diagram 5
2 Event Model Functional Diagram 6
3 Event Model Diagram 88

TABLES

1 Return Status 60
2 Return Status 61
3 Return Status 66
4 Return Status 67
5 AVL Tag Layout 75
6 AVL Routines 76

CHAPTER 1

INTRODUCTION

The Management Services API defines application programming interfaces that provide a set of services managing objects in a heterogeneous, multi–vendor distributed computing environment.

The Management Services API is designed to allow for the development of portable management applications that insulate management application developers from the details of the management protocol and from the transport services used to route the management directives to the managed objects. It provides facilities to manage both local and remote objects in a seamless fashion.

1.1 Intended Audience

This document is intended to provide a general introduction to the services provided by the Management Services API package. The services described are flexible enough to provide for ease of implementation and to provide a valuable starting point for developing a common set of management applications.

1.2 Goals

The goal of the Management Services API is to provide a common set of services for the integrated management of distributed heterogeneous systems. These goals are attained by supplying:

- Integration–providing common facilities for both system and network management
- Portability–insulating client applications from implementation details concerning management protocols, encoding and decoding of management operations, and details of the transport services used to communicate with the target object
- Seamless distributed management–providing the same interfaces for both local and remote management
- Interoperability–providing services that support the management of heterogeneous systems
- Flexibility–providing support for operations in both synchronous and asychronous modes as well as support for operating in a multi–tasking environment
- Extensablity–allow for addition and modification to the interface to support new protocols, manageable objects, and transports

Non-goals:

This document does not attempt to provide exhaustive detail concerning the processing of the services supported. In particular, while the operation of the Management Services API requires some form of directory service to support translation of names to addresses or names to object identifiers, no attempt is made in this document to define that service.

1.3 Related Documents

This document assumes that the reader is familiar with the ISO/OSI management model. This document does not duplicate information presented in the ISO documents or the Internet RFC documents. Related documents include:

- Information Processing Systems - Open Systems Interconnection - Basic Reference Model, ISO 7498-1.
- Information Processing Systems - Open Systems Interconnection - Management Framework, ISO 7498-4.
- Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information, DP 10165.
- Information Processing Systems - Open Systems Interconnection - Common Management Information Service Definition, IS 9595:1989.
- Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), ISO 8824.
- Information Processing Systems - Open Systems Interconnection - Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), ISO 8825.
- Information Processing Systems - Open Systems Interconnection - Common Management Information Protocol (CMIP) Specification, IS 9596:1989.
- The Common Management Information Services over TCP/IP (CMOT), RFC 1095.
- A Simple Network Management Protocol (SNMP), RFC 1098.
- Management Information Base (MIB) for Network Management of TCP/IP-based Internets, RFC-1066.
- Structure and Identification of Management Information for TCP/IP-based Internets, RFC-1065.
- Tutorial on OSI Event Management, Alarm Reporting, and Log Control for TCP/IP Networks, Internet Draft, Lee LaBarre for the OIM Working Group.

CHAPTER 2

OVERVIEW

The Management Services API routines are divided into three functional areas. These areas are Association Services, Management Directive Services, and Event Services. The functionality provided by each of these service is described briefly below.

2.1 Association Services

The Association Services provide a procedural interface that is used to establish the behavior of the service to be provided by the underlying management services implementation. The attributes assigned to the association at the time that it is created and the values assigned to those attributes, define the specifics of the subsequent interactions between the management application and the management services implementation.

Associations are modeled as objects whose attributes are set according to the characteristics of the services to be provided. These attributes are set by the management application when the association is created. The association may then be used to invoke any procedure supported by the management services implementation or by the managed object. When the association is no longer needed it is deactivated. The Association Services are described in Chapter 4.

2.2 Management Directive Services

The Management Directive Services are used to send management directives to managed objects and receive the responses to those directives. An association to the management services implmentation must be created before these services are used.

The set of operations provided by the Management Directive Services is defined by the documents that define the Common Management Information Services, or CMIS, interface. The operations that are supported depend on the capabilities of the managed object, the management services implementation, and the underlying protocol that is used to convey the operation request to the target.

However, if the management services implementation does not extend the basic services supported by the managed object or the specific protocol that is used to transmit the operation, then an error indicating the operation is not supported will be returned by either the management services implementation or by the managed object, as appropriate. The details of the Management Directive Services are described in Chapter 5.

2.3 Event Services

The Event Services are used to provide support for the management application to subscribe to event reports from managed objects. A brief overview of the model is also presented in Chapter 6 along with a description of the services.

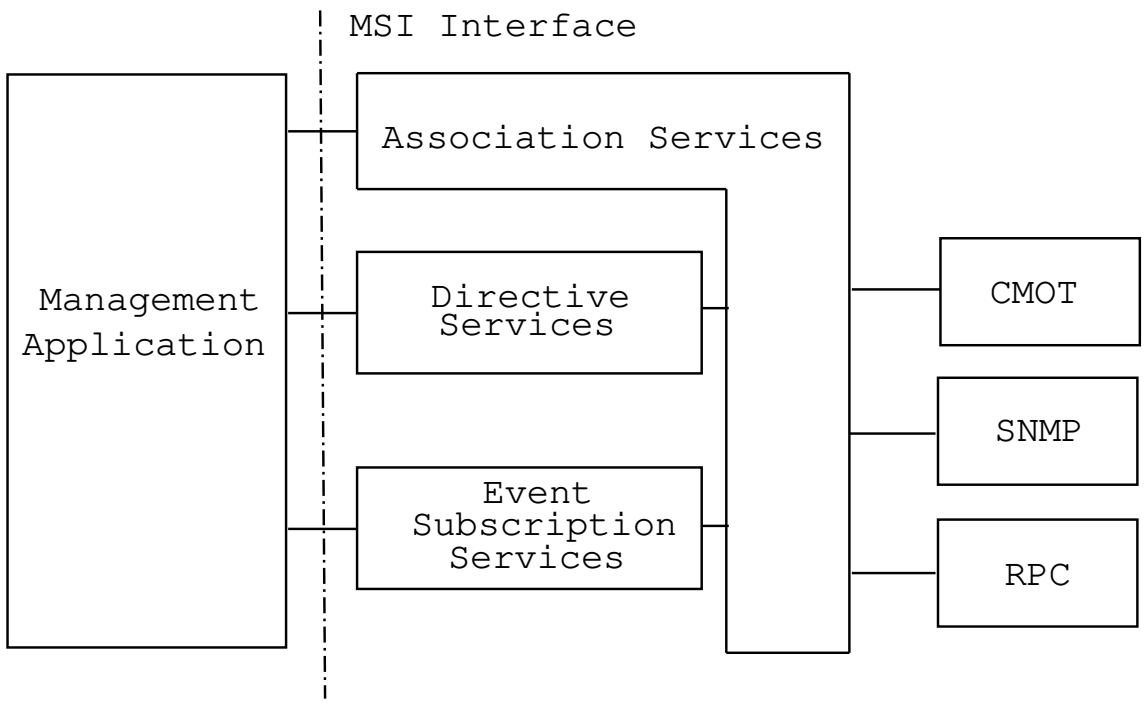
CHAPTER 3

MODEL IMPLEMENTATION

3.1 Structure

This chapter describes a model implementation of the Management Services API. This is only intended to be an example of one potential implementation. The figure shown in Figure 1 is a functional diagram of an implementation of the Management Services API.

Figure 1: Functional Diagram



3.1.1 Management Application

The management application is any client of the services provided by the Management Services API implementation. The interaction between the management application and the management services implementation is provided by the use of Association Services, Directive Services, and Event Subscription Services. These services allow the management application to interact with managed objects in a protocol and transport independent manner.

3.1.2 Association Services

The Association Services component is used to configure the type of service that the management application wishes the management services implementation to provide. Some of the optional attributes are used to specify default values that the management application can override by providing explicit values in subsequent management operations. The aspects of the interaction that can be controlled using the association services are:

- The default protocol to be used to communicate with the managed object.
- Whether the processing of directives on an association should be synchronous or asynchronous. If asynchronous support is required, then the underlying system must support a tasking system, such as threads, to enable this style of interaction.
- The default address for the managed objects. This name must be resolved by the Management Services API implementation to a valid application address for each protocol and transport that is to be used to send management requests to the target.
- The optional access control information needed to validate the association.

For details of the association services see Chapter 4; for the detailed routine descriptions see Section A.1 in Appendix A.

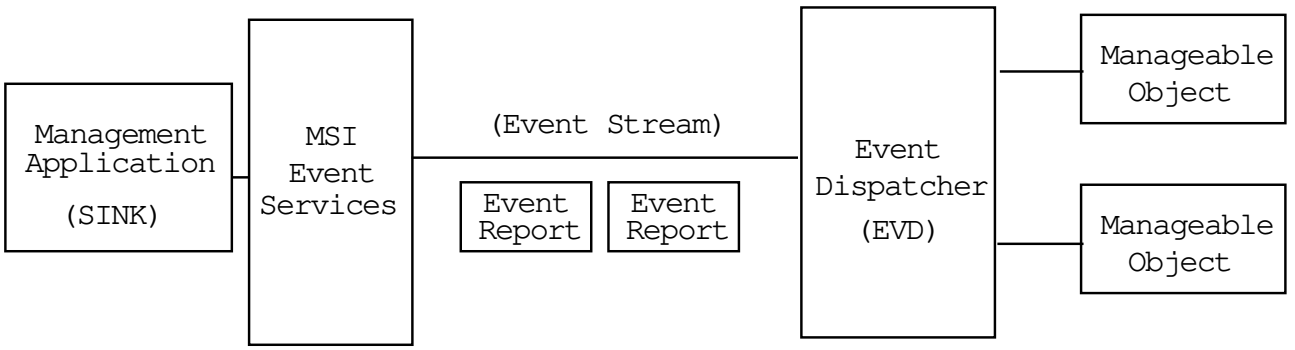
3.1.3 Directive Services

The Directive Services component is used by the management application to request the execution of management directives by managed objects. The directive is executed using the context stored in the association that is specified as one of the parameters to the directive routine. This context may include the location of the managed object, the protocol and transport to use to send the directive to the managed object, and the style of reply is specified by the association used in processing the directive. In addition, the management application can override the default values stored in the association by providing explicit values in the individual directive calls. For the details of the directive services see Chapter 5; for the detailed routine descriptions see Section A.2 in Appendix A.

3.1.4 Event Subscription Services

The event subscription services component allows the management application to act as a sink for event reports from managed objects. These services allow the management application to configure a set of managed objects from which to receive event notifications. Figure 2 shows a functional diagram of the event model.

Figure 2: Event Model Functional Diagram



For further details of the event subscriptions services, refer to Chapter 6; for the detailed routine definitions see Section A.3 in Appendix A.

6 Model Implementation

3.1.5 Protocol Modules

The protocol modules in Figure 2 (CMOT, SNMP, and RPC) are used by the association services to support the interaction between the management application and the managed object. These modules are not directly accessible through the Management Services API but are shown for completeness. This list of protocols is not intended to be exhaustive as this model allows for the addition of other protocols.

CHAPTER 4

ASSOCIATION SERVICES

The Association Services provide client applications with routines to create associations with the management services implementation by creating *Association Objects*. The attributes and the values of the attributes assigned to the association object define the type of services that the application wants to have provided by the management services implementation. The information in the association object provides additional context for the execution of management directives by the management services implementation.

4.1 The Association Object

All association objects have the following attributes:

- *msi_target_name*–a name that can be resolved to the address or addresses of the managed object.
- *msi_access_control*–access control information needed to validate the association
- *msi_protocol*–the distribution layer protocol module used to communicate with the targeted server
  - *msi_protocol_any*–indicates that the management services implementation should choose the protocol used to communicate between the client and targeted server
  - *msi_protocol_cmip*–use the CMIP management protocol
  - *msi_protocol_cmot*–use the CMOT management protocol
  - *msi_protocol_snmp*–use the SNMP management protocol
  - *msi_protocol_rpc*–use RPC
- *msi_style*–the type of interaction style that is requested by the management application for this association
  - *msi_style_callback* - The directive reply invokes a specified callback procedure for each reply. This style requires the use of a tasking system to support waiting for the reply and the invocation of the callback procedure.
  - *msi_style_nonblocking* - The management application will poll for directive replies using the *read_reply* routine to get successive replies. This style also requires support from a tasking system or a threads package to wait for the reply and place the reply data into a local cache.
  - *msi_style_blocking* - The directive operation will block until all replies are received. The routine then returns with a linked list that contains all of the replies to the operation.
- *msi_timeout* —If the type of interaction style that is requested by the application for this association is *msi_style_blocking*, then this attribute controls the amount of time in seconds that the directive will block waiting for the replies. A value of zero indicates no limit.

The steps in using an association are:

1. Create an Association Object. The *msi_create_association_object* routine returns a handle to a newly created association object with the requested attributes. The association object represents an active session with the management services implementation layer. If the optional *msi_target_name* parameter was supplied, then that target address will be the default for directives issued using this association. However, the management application can override the default by supplying an explicit target address as one of the parameters of the directive call. See Chapter 5 for details.
2. Issue Directives. Once the association is created, it can be used to issue management directives through the management services implementation.
3. Delete the Association Object. When the association object is no longer needed, the *msi_delete_association_object* routine unbinds from the the management services implementation and deletes the association object.

4.2 Operations on the Association Object

All of the calls listed below include an association object handle as an argument.

4.2.1 msi_create_association_object

The *msi_create_association_object* creates a new instance of the association object, binds it to the management services implementation, and returns a handle to be used in subsequent management operations. The attributes that are supplied to this routine control the processing of the management operations.

4.2.2 msi_delete_association_object

The *msi_delete_association_object* deletes the association object specified by the association object handle argument. The association will be unbound from the management services implementation.

CHAPTER 5

MANAGEMENT DIRECTIVE SERVICES

Management applications use the directive services of the Management Services API to query and/or modify attributes of manageable objects and to invoke operations on manageable objects.

The steps in the interaction between management applications and manageable objects are:

1. The management application creates an association with the management services implementation. This is done using the association services.
2. The management application then issues management operations to some managed object. The management services implementation uses the context supplied by the association and the parameters to the operation itself to forward the directive to the target managed object.

   - Manageable object creation or deletion operations such as *msi_create_instance*, or *msi_delete_instance*
   - Query or modification operations on manageable object attributes such as *msi_get_attributes*, *msi_set_attributes*, or *msi_getnext_attribute*
   - Manageable object specific actions such as *msi_invoke_action*

   The operations provided by the directive services component are aligned with the CMIS (Common Management Information Services (IS 9596:1989)) operations with the addition of the operations required to support SNMP as defined in RFC 1098. For specific information about the behavior of these operations, refer to the either the ISO document or RFC 1098.
3. Get the result of the service request.

   If the *msi_style* attribute of the association was set to *msi_style_callback*, then the directive routine returns to the management application immediately. When the reply to the directive is returned from the target system, then the callback routine for that association is invoked and issues a *msi_read_reply* operation to receive the result of the service requested. The specific reply that is received is determined by examining the *invoke_identifier* that is returned. Note that the use of this style requires the support of either some form of tasking system or a multi-threaded environment.

   If the *msi_style* attribute of the association was set to *msi_style_nonblocking*, then the directive routine returns immediately to the caller. To receive the reply to the operation the management application calls the *msi_read_reply* routine passing the *invoke_identifier* to identify the directive whose response is required. If the reply is not available, then the routine returns a status of *MAN_NO_REPLY_AVAILABLE* and the management application must try later. If reply data is available, it is returned to the management application with the *more_replies* parameter set to indicate if there are more replies to this operation to be read. These additional replies are read using additional calls to *msi_read_reply*. Note that the use of this style requires the support of either some form of tasking system or a multi-threaded environment.

If the *msi_style* attribute of the association was set to *msi_style_blocking*, then the directive routine blocks until all of the replies to the operation are returned from the target or until the time out set by the attribute of the association expires. The replies are returned in a linked list structure in the *reply* parameter of the directive.

4. Repeat steps 2 & 3 until the association is no longer needed.
5. Delete the association between the management application and the management services implementation.

5.1 Directive Operations

Some targets support *scoping* and *filtering*. Through scoping and filtering, management applications can qualify which classes and instances of manageable objects should process the directive. Generally, these targets are those that support either the CMIP or CMOT management protocols.

Scoping allows a management application to specify groups of related manageable objects based on the containment hierarchy of the object classes and their instances. Filtering allows a management application to constrain the instances of the manageable object class that are targets of the directive based on comparison operations on their attribute values.

For a complete description of the behavior of scoping and filtering, refer to the listed ISO documents and RFC's in Section 1.3.

5.1.1 msi_create_instance

The *msi_create_instance* routine is used to request that a new instance of a manageable object class be created. There are no scoping or filtering operations allowed with this routine. However, the request can include initial values for some or all of the new instance's attributes and/or the name of an already existing instance to use as a template.

5.1.2 msi_delete_instance

The *msi_delete_instance* routine is used to delete the specified manageable object instance. The operation of this directive can be modified by the use of the scoping and filters.

5.1.3 msi_get_attributes

The *msi_get_attributes* operation reads the values of the specified attributes of manageable objects. If scoping and filtering are supported, then this operation can be applied selectively to multiple manageable objects using scoping and, if desired, qualify that selection further using filtering.

5.1.4 msi_getnext_attribute

The *msi_getnext_attribute* procedure implements the GET_NEXT operation of the SNMP protocol. For details of the behavior of this operation refer to RFC 1098.

5.1.5 msi_set_attributes

The *msi_set_attributes* operation is used by the management application to modify the specified attributes of the manageable object. If scoping and filtering are supported, then this operation can be applied selectively to multiple objects using scoping and, if desired, qualify that selection using filtering.

5.1.6 msi_invoke_action

The *msi_invoke_action* routine requests the manageable object to perform an object class specific operation. If the protocol in use supports scoping and filtering, then this operation may be specified on multiple classes and instances of manageable objects using scoping and, if desired, select only a subset to operate on via the filter parameter.

5.1.7 msi_read_reply

The *msi_read_reply* routine is used by the management application to request the result of an operation. This routine is used to read the return data from an operation.

5.1.8 msi_send_reply

The *msi_send_reply* routine is used to return the result of an operation. For example, this routine can be used by the management application to return a response to a confirmed event report.

CHAPTER 6

EVENT SERVICES

Event Services are used to process asynchronous notifications from managed objects. The event services provide event subscription services for a management application that wishes to act as an event sink. Event Services also provide event posting services for the event dispatcher to use in sending event reports to event sinks.

The event reports from the managed objects are posted to the Event Dispatcher object that resides on the same system as the object generating the event report. The Event Dispatcher is responsible for queuing these event reports internally and then forwarding them to anyone that has subscribed to event reports from that system.

6.1 Event Subscription Services

The event subscription services are provided by the *msi_create_subscription* procedure. The user who wants to subscribe to event reports creates an event subscription object that is configured to receive event reports from the desired systems and or managed objects. Once the subscription object has been configured and activated, the event reports that arrive in the subscription object are queued internally until the management application reads them using the *msi_read_event* routine.

6.1.1 msi_create_subscription

The *msi_create_subscription* routine causes the management services implementation to create an event receiver that is configured to accept event reports from managed objects. The input parameters to this routine are a set of attributes that control behavior of the event subscription. These attributes include the name of the event dispatcher to receive event reports from, a set of event forwarding discriminators that control the type of events accepted, an optional access control parameter, the protocol used to transmit the event reports, and the style of use of the event receiver.

- *msi_target_name*–the name of the targeted system.
- *msi_access_control*–the access control information for the subscription.
- *msi_protocol*–the distribution layer protocol module used to communicate with the event dispatcher
  - *msi_protocol_any* - indicates that the management services should choose the protocol module used to communicate between the client and the event dispatcher
  - *msi_protocol_cmip* - use the CMIP management protocol
  - *msi_protocol_cmot* - use the CMOT management protocol
  - *msi_protocol_snmp* - use the SNMP management protocol
  - *msi_protocol_rpc* - use RPC

- *msi_style*–the type of interaction style that is requested by the management application for this subscription
  - *msi_style_callback* - The directive reply invokes a specified callback procedure for each received event.
  - *msi_style_nonblocking* - The management application will poll for events using the *msi_read_event* routine to get event reports.
- *msi_event_discriminator*–the event forwarding discriminator to apply to event reports on this subscription. For details of the discriminator and its operations see the section in Appendix A on directive filters.
- *msi_event_begin_time*–this attibute sets the begin time attribute for the event forwarding descriminator. If it is not present the begin time is set to 00:00.
- *msi_event_end_time*–this attibute sets the end time attribute for the event forwarding descriminator. If it is not present the begin time is set to 24:00.
- *msi_event_maxevents*–the maximum number of events that can be forwarded in the period set by the *msi_event_windowtime* attribute. If there is no value supplied or the value is zero, then the flow control option is inactive.
- *msi_event_windowtime*–the time interval over which the rate of events is to be monitored for the purpose of flow control. If there is no value supplied or the value is zero, then the flow control option is inactive.

6.1.2 msi_delete_subscription

The *msi_delete_subscription* routine is used by a management application to terminate event subscription.

6.1.3 msi_read_event

The *msi_read_event* operation returns the next event report in the event sink queue.

6.2 Event Reporting Services

The event dispatcher uses association services to create an association between the event dispatcher and the event sink. These associations are called *event streams*. Once the association is created, the event reports from the managed objects that pass the forwarding discriminator for the stream are sent to the sink using the *msi_report_event* routine.

6.2.1 msi_report_event

The *msi_report_event* routine is used by the event dispatcher to send an event report to the sink.

APPENDIX A

ROUTINE DESCRIPTIONS FOR MANAGEMENT SERVICES API

A.1 Association Routines

A.1.1 msi_create_association_object

```
man_status
msi_create_association_object(
   association       *assoc,
   avl               *attribute_list,
   callback_proc     *callback,
   void              *data,
   stream            *stream
);
```

A.1.1.1 Description

The *msi_create_association_object* routine creates a new association object. The characteristics of the association can be specified using the *attribute_list* parameter. The association is used to configure the relationship between the management application and the management services implementation. After the association is successfully created, then the association handle is used in subsequent directive operations to provide context for the processing of the directives and directive replies.

A.1.1.2 Parameters

Parameter	Direction	Status	Description
assoc	Output	Required	This is the handle that represents the association that is created.
attribute_list	Input	Required	This is a list of attribute values to be applied to the created association.
callback	Input	Optional[1]	This is the callback procedure to be used to process directive replies on this association. This routine is called with three parameters, the *data* pointer from the association, a pointer to the association handle itself and the *invoke_identifier* of the directive that generated the reply.
data	Input	Optional[1]	This is the pointer to user defined data that is to be passed when the user supplied callback procedure is invoked.
stream	Output	Required	This is a pointer to the stream that represents this association.[2]

[1]This parameter is used only if the style of the association is set to local_callback.

[2]This output parameter is used to allow the management application to manage I/O on the association using its own mechanisms.

A.1.1.3 Association Attributes

Parameter	Status	Description
msi_target_name	Optional	This attribute identifies the default target to which directives using this association should be sent. If this parameter is not provided and an explicit target address is not supplied in the parameters to the directive, then the management services implementation will determine the destination of the request based on the object class and object instance information.
msi_protocol	Required	This attribute indicates the protocol to use to communicate with the target. It can be one of *msi_protocol_any*, *msi_protocol_cmip*, *msi_protocol_cmot*, *msi_protocol_snmp*, or *msi_protocol_rpc*.
msi_style	Required	This attribute controls the style of processing that the management application wishes to use for directives issued using this association. This attribute can be set to *msi_style_callback*, *msi_style_nonblocking*, or *msi_style_blocking*.
msi_timeout	Optional	This attribute controls the amount of time in seconds that the management services implementation will wait for directive replies before returning an error. A value of zero indicates no limit and is the default. The *msi_operation_timeout* parameter to the directive routines will override this value.

A.1.1.4 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_UNKNOWN_ATTRIBUTE	One of the attributes supplied was unknown.
MAN_C_BAD_ATTRBUTE_VALUE	One of the attribute values supplied was incorrect or not supported.
MAN_C_UNKNOWN_TARGET	The target specified in the association attribute could not be found.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_ACCESS_REJECT	The access control information supplied was rejected by the target system.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

A.1.2 msi_delete_association_object

```
man_status
msi_delete_association_object(
   association      *assoc
);
```

A.1.2.1 Description

The *msi_delete_association_object* call is used to delete an association object. If there are any directive replies outstanding, then they will be discarded.

A.1.2.2 Parameters

Parameter	Direction	Status	Description
assoc	Input	Required	The handle that represents the association that is to be deleted.

A.1.2.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_STILL_ACTIVE	The association was shut down but had outstanding requests.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

A.2 Directive Routines

A.2.1 msi_create_instance

```
man_status
msi_create_instance(
                    association         *assoc,
                    object_id           *object_class ,
                    avl                 *object_instance ,
                    long int            *invoke_identifier,
                    avl                 *superior_instance,
                    access_info         *access_control,
                    avl                 *attribute_value_list,
                    avl                 *reference_instance,
                    avl                 *target,
                    int32               *operation_timeout
);
```

A.2.1.1 Description

The *msi_create_instance* routine is used to create a new instance of a class. There are no scoping or filtering operations allowed with the routine. However, the request can include initial values for some or all of the new instance's attributes and/or a template instance name from which attribute values can be taken.

A.2.1.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association identification for the connection to the target.
object_class	Input	Required	Class specification for the target managed object
object_instance	Input	Optional	Name for the target managed object's particular occurrence
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
invoke_identifier	Output	Required	ID associated with the request
superior_instance	Input	Optional	The name of the an instance of the superior class that is the parent of the new instance to be created. If this parameter is supplied, then *object_instance* is not supplied.
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.
attribute_value_list	Input	Optional	A pointer to a list of attribute values to be assigned to the newly created instance.
reference_instance	Input	Optional	The name of an already existing instance from which attribute values are to be taken. If both a reference instance and a attribute value list are supplied, then the values in the attribute list supersede the values from the reference instance.

Parameter	Direction	Status	Description
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero indicates unlimited time.

A.2.1.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by the target, or protocol being used.

A.2.2 msi_delete_instance

```
man_status
msi_delete_instance(
                    association         *assoc,
                    object_id           *object_class ,
                    avl                 *object_instance ,
                    reply_list          *reply,
                    long int            *invoke_identifier ,
                    iso_scope           *scope,
                    avl                 *filter,
                    access_info         *access_control,
                    int_32              *synchronization,
                    avl                 *target,
                    int_32              *operation_timeout
);
```

A.2.2.1 Description

The *msi_delete_instance* routine sends a delete operation to the target target for the specified class and instance.

A.2.2.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
object_class	Input	Required	Class specification for the target managed object
object_instance	Input	Required	Name for the target managed object's particular occurrence
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
invoke_identifier	Output	Required	ID associated with the request
scope	Input	Optional	A pointer to an *iso_scope* structure, which indicates the subtree, rooted at the base managed object which is to be searched by the object manager to decide what object(s) the operation is to be carried out on. The levels of search that may be performed are: (a) the base object alone, (b) the Nth level subordinates of the base object, (c) the base object and all off its subordinates down to and including the Nth level, (d) the base object and all of its subordinates. The default scope is base object alone.
filter	Input	Optional	A pointer to a filter data structure which specifies the set of assertions that comprise the filter test to be applied to the scoped managed object(s).
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.

Parameter	Direction	Status	Description
synchronization	Input	Optional	Management operation to be best effort or atomic. The default is best effort.
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero or NULL indicates unlimited time.

A.2.2.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by the target, or protocol being used.

A.2.3 msi_get_attributes

```
man_status
msi_get_attributes(
                    association          *assoc,
                    object_id            *object_class ,
                    avl                  *object_instance ,
                    reply_list           *reply,
                    long int             *invoke_identifier ,
                    iso_scope            *scope,
                    avl                  *filter,
                    access_info          *access_control,
                    int_32               *synchronization,
                    avl                  *attribute_id_list,
                    avl                  *target,
                    int_32               *operation_timeout
);
```

A.2.3.1 Description

This operation queries the specified classes for the attribute values of an instance. It can specify this operation selectively on multiple classes and instances using scoping and filtering.

A.2.3.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
object_class	Input	Required	Class specification for the target managed object
object_instance	Input	Required	Name for the target managed object's particular occurrence
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
invoke_identifier	Output	Required	ID associated with the request
scope	Input	Optional	A pointer to an *iso_scope* structure, which indicates the subtree, rooted at the base managed object which is to be searched by the object manager to decide what object(s) the operation is to be carried out on. The levels of search that may be performed are: (a) the base object alone, (b) the Nth level subordinates of the base object, (c) the base object and all off its subordinates down to and including the Nth level, (d) the base object and all of its subordinates. The default scope is base object alone.
filter	Input	Optional	A pointer to a filter data structure which specifies the set of assertions that comprise the filter test to be applied to the scoped managed object(s).
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.

Parameter	Direction	Status	Description
synchronization	Input	Optional	Management operation to be best effort or atomic. The default is best effort.
attribute_id_list	Input	Optional	A pointer to a list of attribute ids. If no value is specified, the default is that all attribute identifiers are returned.
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero or NULL indicates unlimited time.

A.2.3.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by the target, or protocol being used.

A.2.4 msi_getnext_attribute

```
man_status
msi_getnext_attribute(
                    association         *assoc,
                    avl                 *var_bind_list,
                    long int            *invoke_identifier
                    reply_list          *reply,
                    access_info         *access_control,
                    avl                 *target,
                    int_32              *operation_timeout

);
```

A.2.4.1 Description

This operation is used to support the SNMP GET_NEXT operation.

A.2.4.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
var_bind_list	Input	Required	A pointer to a list of SNMP variable bindings, each of which is a pair of object names (ids) and values. This list specifies the object names for which values are to be retrieved as per the GET NEXT operation. The format of these object names is specified in the SNMP specification.
invoke_identifier	Output	Required	The *invoke_identifier* is used in the directive replies to identify the particular directive that caused the replies to be generated.
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero or NULL indicates unlimited time.

A.2.4.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by this protocol.

A.2.5 msi_set_attribute

```
man_status
msi_set_attribute(
            association          *assoc,
            object_id            *object_class,
            avl                  *object_instance,
            int                   mode,
            avl                  *modification_list,
            long int             *reply,
            long int             *invoke_identifier,
            iso_scope            *scope,
            avl                  *filter,
            access_info          *access_control,
            int_32               *synchronization,
            avl                  *target,
            int_32               *operation_timeout
);
```

A.2.5.1 Description

The *msi_set_attribute* operation specifies attributes of a managed object to be modified. It can also specify this operation selectively on multiple classes using scoping and, if desired, qualify their selection via the filter argument.

A.2.5.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
object_class	Input	Required	The class specification for the target managed object.
object_instance	Input	Required	The particular instance of an object identified by object_class.
mode	Input	Required	This parameter controls whether a response is required. It is one of *CONFIRMED* or *NON_CONFIRMED*.
modification_list	Input	Required	A pointer to a list of attribute identifiers, values, and the operation to be performed on the attribute (replace, set to default, add or remove).
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
invoke_identifier	Output	Required	The *invoke_identifier* is used in the directive replies to identify the particular directive that caused the replies to be generated.

Parameter	Direction	Status	Description
scope	Input	Optional	A pointer to an *iso_scope* structure, which indicates the subtree, rooted at the base managed object which is to be searched by the object manager to decide what object(s) the operation is to be carried out on. The levels of search that may be performed are: (a) the base object alone, (b) the Nth level subordinates of the base object, (c) the base object and all off its subordinates down to and including the Nth level, (d) the base object and all of its subordinates. The default scope is base object alone.
filter	Input	Optional	A pointer to a filter data structure which specifies the set of assertions that comprise the filter test to be applied to the scoped managed object(s).
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.
synchronization	Input	Optional	Management operation to be best effort or atomic. The default is best effort.
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero or NULL indicates unlimited time.

A.2.5.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by the target, or protocol being used.

A.2.6 msi_invoke_action

```
man_status
msi_invoke_action(
            association          *assoc,
            object_id            *object_class,
            avl                  *object_instance,
            int                   mode,
            object_id            *action_type,
            reply_list           *reply,
            long int             *invoke_identifier,
            iso_scope            *scope,
            avl                  *filter,
            access_info          *access_control,
            int_32               *synchronization,
            avl                  *action_info,
            avl                  *target,
            int_32               *operation_timeout
);
```

A.2.6.1 Description

The *msi_invoke_action* operation requests that the managed object perform the action specified. This operation may be carried out on multiple classes using scoping and, if desired, select only a subset to operate on via the filter argument.

A.2.6.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
object_class	Input	Required	The class specification for the target managed object.
object_instance	Input	Required	The particular instance of an object identified by object_class.
mode	Input	Required	This parameter controls whether a response is required. It is one of *CONFIRMED* or *NON_CONFIRMED*.
action_type	Input	Required	The particular action that is to be performed.
reply	Output	Optional	This parameter is a pointer to a linked list of replies to this operation. If the *msi_style* attribute of the association is set to *msi_style_blocking*, then this parameter is used to return the responses to the operation.
invoke_identifier	Output	Required	The *invoke_identifier* is used in the directive replies to identify the particular directive that caused the replies to be generated.

Parameter	Direction	Status	Description
scope	Input	Optional	A pointer to an *iso_scope* structure, which indicates the subtree, rooted at the base managed object which is to be searched by the object manager to decide what object(s) the operation is to be carried out on. The levels of search that may be performed are: (a) the base object alone, (b) the Nth level subordinates of the base object, (c) the base object and all off its subordinates down to and including the Nth level, (d) the base object and all of its subordinates. The default scope is base object alone.
filter	Input	Optional	A pointer to a filter data structure which specifies the set of assertions that comprise the filter test to be applied to the scoped managed object(s).
access_control	Input	Optional	A pointer to the access control information to be used for this particular call.
synchronization	Input	Optional	Management operation to be best effort or atomic. The default is best effort.
action_info	Input	Optional	A pointer to the arguments to this action call.
target	Input	Optional	This is a pointer to a structure which contains an address specifying where the request should be sent. If this parameter is not provided then the management services implementation will determine the destination of the request based on the default target specified in the association. If no target address was specified for the association, then the destination is determined from the object class and object instance information.
operation_timeout	Input	Optional	The number of seconds to wait for the operation to complete. A value of zero or NULL indicates unlimited time.

A.2.6.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NOT_SUPPORTED	This operation is not supported by the target, or protocol being used.

A.2.7 msi_read_reply

The *msi_read_reply* routine is used by the management application to request the result of an operation. The *invoke_identifier*, *assoc*, and the optional *reply_data* parameters are used to identify the operation whose result is being requested.

```
man_status
msi_read_reply(
                association          *assoc,
                long int             *invoke_identifier,
                reply_list           *reply_data ,
                reply_type           *reply,
                reply_status         *status,
                object_id            *object_class,
                avl                  *object_instance,
                time                 *operation_time,
                avl                  *reply_information,
                boolean              *more_replies
);
```

A.2.7.1 Description

This call is used to return the response to a directive. If the *msi_style* of the association that is being used is set to *msi_style_callback*, then this routine is called from the callback routine with the *assoc* and *invoke_identifier* used to identify the particular reply that is available.

If the *msi_style* is set to *msi_style_nonblocking* then this routine is called any time after the directive is issued with the *assoc* and *invoke_identifier* parameter indicating the directive whose response is required.

if the *msi_style* of the association that is being used is set to *msi_style_blocking*, then this routine is called after the directive routine returns. In this case, the *reply_data* returned from the original directive is also passed as an input parameter.

In all cases, the *read_reply* routine returns the reply data in the output parameters of the routine. Note that the interpretation of the returned data depends on the value returned in the *status* parameter. If the operation returned an error, then the return parameters are used to return error information. Also for all the styles, if the *more_replies* parameter is returned as non–zero, then successive calls to *read_reply* are used to return the data for the next reply. If the reply type is *MAN_C__GETNEXT_REPLY*, then the returned varbindlist is passed back in the *reply_information* parameter.

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
invoke_identifier	Input	Required	The *invoke_identifier* of the directive that caused the reply to be generated.
reply_data	Input	Optional	The reply list returned from the operation call. This is used with the blocking style of operation.
reply	Output	Required	The type of the reply returned.
status	Output	Required	The status returned from the particular operation. See Section A.2.7.1.2 for a list of the possible status returns for the directives.
object_class	Output	Required	The class specification for the target managed object.

Parameter	Direction	Status	Description
object_instance	Output	Required	The particular instance of an object identified by object_class.
operation_time	Output	Required	The time the target completed execution of the directive.
reply_information	Output	Required	The list of returned data from the directive. The actual contents of this list is determined by the reply parameter.
more_replies	Output	Required	This argument indicates whether or not there are more replies for the given invoke_identifier.

A.2.7.1.1 Values for the reply_type

The values returned in the *reply* parameter indicate the type of response to the directive that the managed object returned. The contents and interpretation of the other return parameters is based on this value. Possible reply types include:

- MAN_C__GET_REPLY
- MAN_C__SET_REPLY
- MAN_C__CREATE_REPLY
- MAN_C__DELETE_REPLY
- MAN_C__ACTION_REPLY
- MAN_C__GETNEXT_REPLY

A.2.7.1.2 Status Returns

The values for the *status* parameter for each of the operation type is listed below.

- Get Status Returns.
  - MAN_C__NORMAL_REPLY
  - MAN_C__ACCESS_DENIED
  - MAN_C__CLASS_INSTANCE_CONFLICT
  - MAN_C__COMPLEXITY_LIMITATION
  - MAN_C__GET_LIST_ERROR
  - MAN_C__INSUFFICIENT_RESOURCES
  - MAN_C__INVALID_FILTER
  - MAN_C__INVALID_SCOPE
  - MAN_C__NO_SUCH_CLASS
  - MAN_C__NO_SUCH_OBJECT_INSTANCE
  - MAN_C__PROCESSING_FAILURE
  - MAN_C__SYNC_NOT_SUPPORTED
  - MAN_C__RESOURCE_LIMITATION
- GET_NEXT Status Returns.
  - MAN_C__NORMAL_REPLY

    - MAN_C__TOO_BIG
    - MAN_C__NO_SUCH_NAME
    - MAN_C__BAD_VALUE
    - MAN_C__READ_ONLY
    - MAN_C__GEN_ERR
- Set Status Returns
    - MAN_C__NORMAL_REPLY
    - MAN_C__ACCESS_DENIED
    - MAN_C__CLASS_INSTANCE_CONFLICT
    - MAN_C__COMPLEXITY_LIMITATION
    - MAN_C__INVALID_FILTER
    - MAN_C__INVALID_SCOPE
    - MAN_C__NO_SUCH_CLASS
    - MAN_C__NO_SUCH_OBJECT_INSTANCE
    - MAN_C__PROCESSING_FAILURE
    - MAN_C__SET_LIST_ERROR
    - MAN_C__SYNC_NOT_SUPPORTED
    - MAN_C__RESOURCE_LIMITATION
- Create Status Returns
    - MAN_C__NORMAL_REPLY
    - MAN_C__ACCESS_DENIED
    - MAN_C__CLASS_INSTANCE_CONFLICT
    - MAN_C__COMPLEXITY_LIMITATION
    - MAN_C__DUPLICATE_M_O_INSTANCE
    - MAN_C__INVALID_ATTRIBUTE_VALUE
    - MAN_C__INVALID_OBJECT_INSTANCE
    - MAN_C__MISSING_ATTRIBUTE_VALUE
    - MAN_C__NO_SUCH_ATTRIBUTE_ID
    - MAN_C__NO_SUCH_CLASS
    - MAN_C__NO_SUCH_OBJECT_INSTANCE
    - MAN_C__NO_SUCH_REFERENCE_OBJECT
    - MAN_C__PROCESSING_FAILURE
    - MAN_C__RESOURCE_LIMITATION
- Delete Status Returns
    - MAN_C__NORMAL_REPLY
    - MAN_C__ACCESS_DENIED

  - MAN_C__CLASS_INSTANCE_CONFLICT
  - MAN_C__COMPLEXITY_LIMITATION
  - MAN_C__INVALID_FILTER
  - MAN_C__INVALID_SCOPE
  - MAN_C__NO_SUCH_CLASS
  - MAN_C__NO_SUCH_OBJECT_INSTANCE
  - MAN_C__PROCESSING_FAILURE
  - MAN_C__SYNC_NOT_SUPPORTED
  - MAN_C__RESOURCE_LIMITATION
- Action Status Returns
  - MAN_C__NORMAL_REPLY
  - MAN_C__ACCESS_DENIED
  - MAN_C__CLASS_INSTANCE_CONFLICT
  - MAN_C__COMPLEXITY_LIMITATION
  - MAN_C__DUPLICATE_ARGUMENT
  - MAN_C__INVALID_ARGUMENT_VALUE
  - MAN_C__INVALID_FILTER
  - MAN_C__INVALID_SCOPE
  - MAN_C__NO_SUCH_ACTION
  - MAN_C__NO_SUCH_ARGUMENT
  - MAN_C__NO_SUCH_CLASS
  - MAN_C__NO_SUCH_OBJECT_INSTANCE
  - MAN_C__PROCESSING_FAILURE
  - MAN_C__SYNC_NOT_SUPPORTED
  - MAN_C__RESOURCE_LIMITATION

A.2.7.2 Return Values

Return Code	Description
MAN_C_SUCCESS	The reply was returned.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NO_REPLY	The reply for the requested operation was not available.

A.2.8 msi_send_reply

```
man_status
msi_send_reply(
                association          *assoc,
                long int             *invoke_identifier,
                reply_type            reply,
                reply_status         *status,
                object_id            *object_class,
                avl                  *object_instance,
                time                 *operation_time,
                avl                  *attribute_list,
                boolean              *more_replies
);
```

A.2.8.1 Description

The *msi_send_reply* routine is used to return the result of an operation. Note that, while this routine is defined to be general purpose enough to respond to management directives, the only use that the management application currently makes of this routine is to return the response to a confirmed event report. The *invoke_identifier* parameter is used to identify the operation whose result is being sent.

A.2.8.2 Parameters

Parameter	Direction	Status	Description
association	Input	Required	The association handle.
invoke_identifier	Output	Required	The *invoke_identifier* of the directive that caused the reply to be generated.
reply	Output	Required	The type of the reply returned.
status	Output	Required	The status to be returned from the particular operation. See Section A.2.7.1.2 for a list of the possible status returns for the directives.
object_class	Input	Required	The class specification for the target managed object.
object_instance	Input	Required	The particular instance of an object identified by object_class.
operation_time	Output	Required	The time the target completed execution of the directive.
attribute_list	Output	Required	The list of returned data from the directive.
more_replies	Output	Required	This argument indicates whether or not there are more replies for the given invoke_identifier.

The possible values for the *reply* parameter are the same as those listed in the *msi_read_reply* routine.

A.2.8.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.
MAN_C_NO_REPLY	The reply for the requested operation was not available.

A.3 Event Routines

A.3.1 msi_create_subscription

```
man_status
msi_create_subscription(
   event_subscription   *event_handle,
   avl                  *attribute_list,
   callback_proc        *callback,
   void                 *data,
   stream               *stream
);
```

A.3.1.1 Description

The *msi_create_subscription* routine is used to subscribe to event reports from the management services implementation. The routine creates an event report stream from the target system to an event receiver on the subscriber's system. The various attributes of the subscription object control the functioning of this event stream.

Once the subscription object has been created, the event stream from the target system is connected to the event receiver by using the directive services to configure the event forwarding discriminator on the target system. Then the *event_subscription* handle is used in the *msi_read_event* call to receive event reports.

A.3.1.2 Parameters

Parameter	Direction	Status	Description
event_handle	Output	Required	The handle that represents the event stream that is created.
attribute_list	Input	Required	The attribute values to be applied to the created event subscription.
callback	Input	Optional	This is the callback procedure to be used to process event reports if the style of the event subscription was set to *local_callback*.
data	Input	Optional	This is the pointer to user defined data that is to be passed when the user supplied callback procedure is invoked.
stream	Output	Required	This is a pointer to the stream that represents this subscription. This output parameter is used to allow the management application to manage I/O on the association using it's own mechanisms.

A.3.1.3 Event Subscription Attributes

Attribute	Status	Description
msi_target_name	Required	The name of the target that will be the source of the event reports.

Attribute	Status	Description
msi_access_control	Optional	The access control to be used to create the event subscription.
msi_protocol	Required	This attribute indicates the protocol to use to communicate with the target. It can be one of *msi_protocol_any*, *msi_protocol_cmip*, *msi_protocol_cmot*, *msi_protocol_snmp*, or *msi_protocol_rpc*.
msi_style	Required	This attribute controls the style of processing that the management application wishes to use for event reports received on this subscription. This attribute can be set to *msi_style_callback* or *msi_style_blocking*.
msi_event_descriminator	Required	This attribute is used to configure the event forwarding discriminator construct on the target system.
msi_event_begin_time	Optional	This attibute sets the begin time attribute for the event forwarding descriminator. If it is not present the begin time is set to 00:00.
msi_event_end_time	Optional	This attibute sets the end time attribute for the event forwarding descriminator. If it is not present the begin time is set to 24:00.
msi_event_maxevents	Optional	The maximum number of events that can be forwarded in the period set by the *msi_event_windowtime* attribute. If there is no value supplied or the value is zero, then the flow control option is inactive.
msi_event_windowtime	Optional	The time interval over which the rate of events is to be monitored for the purpose of flow control. If there is no value supplied or the value is zero, then the flow control option is inactive.

A.3.1.4 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_UNKNOWN_ATTRIBUTE	One of the attribute values supplied was unknown.
MAN_C_BAD_ATTRBUTE_VALUE	One of the attribute values supplied was incorrect or not supported.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_DEST_UNREACHABLE	The target did not respond to the subscription request.
MAN_C_ACCESS_REJECT	The access control information supplied was rejected by the target system.
MAN_C_PROT_NOT_SUPPORTED	The protocol requested is not supported by the target system.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

A.3.2 msi_delete_subscription

```
man_status
msi_delete_subscription(
   event_subscription   *event_handle
);
```

A.3.2.1 Description

The *msi_delete_subscription* routine deletes the event forwarding discriminator on the target system to disable the event report stream that is used to send event reports to the management application. This deactivates this event subscription.

A.3.2.2 Parameters

Parameter	Direction	Status	Description
event_handle	Input	Required	The handle obtained from *msi_create_subscription* routine.

A.3.2.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

A.3.3 msi_read_event

```
man_status
msi_read_event(
                event_subscription   *event_subscription,
                object_id            *object_class,
                avl                  *object_instance,
                time                 *event_time,
                object_id            *event_type,
                avl                  *event_parameters
);
```

A.3.3.1 Description

The *msi_read_event* operation reads an event report from the event queue of the specified subscription object and formats the information.

A.3.3.2 Parameters

Parameter	Direction	Status	Description
event_subscription	Input	Required	The handle obtained from *msi_subscribe_event* routine.
object_class	Output	Required	The manageable object class that generated the event report.
object_instance	Output	Required	The instance that generated the event report.
event_time	Output	Required	The time that the event occurred.
event_type	Output	Required	The event type object identifier.
event_parameters	Output	Required	The event parameters if any.

A.3.3.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_NO_RESOURCES	There was a lack of resources on the local system to support this request.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

A.3.4 msi_report_event

```
man_status
msi_report_event(
                association             *event_receiver,
                object_id               *object_class,
                avl                     *object_instance,
                time                    *event_time,
                object_id               *event_type,
                avl                     *event_parameters
);
```

A.3.4.1 Description

The *msi_report_event* routine is used by the event dispatcher to send an event report to the event sink that has subscribed to such reports.

A.3.4.2 Parameters

Parameter	Direction	Status	Description
event_receiver	Input	Required	The handle obtained from *msi_create_association* call.
object_class	Input	Required	The managed object class that is generating the event report.
object_instance	Input	Required	The managed object instance that is generating the event report.
event_time	Input	Required	The time that the event occurred in binary format.
event_type	Input	Required	The event type identifier.
event_parameters	Input	Optional	The event parameters.

A.3.4.3 Return Values

Return Code	Description
MAN_C_SUCCESS	The operation completed successfully.
MAN_C_LOCAL_SHUTDOWN	The local system is in a shutdown state and will not process this request.
MAN_C_INVALID_HANDLE	The handle passed was not valid for this operation.

APPENDIX B

DATA TYPES AND UTILITY ROUTINES

The data types used in the parameters of the management services API routines are defined in this appendix. Most of those parameters are defined by data types which are simple types or simple structures. However, some of the parameters have types that are complex and of variable size. One of these complex parameters is the type used for object identifiers and a set of utility routines are defined to allow them to be created and manipulated. The object identifier type and the utility routines that manipulate it are defined in Section B.5.

Other complex parameters are encoded using a type called an *AVL* or Attribute Value List type. The *AVL* is defined as a opaque type. This means that the *AVL* itself is referred to only by an opaque handle and all creation and modification of parameters that are defined to be based on the *AVL* type is done through procedures. The *AVL* type and the procedures used to manipulate it are documented in Appendix C.

B.1 Octet String Type

The octet string is used for passing data to be encoded in the AVL type. The *local_tag* field in the structure is used to indicated the local representation of the data.

```
typedef struct {
    uns_int16   length ;
    uns_int32      local_tag ;
    void       *string ;
} octet_string ;
```

In this structure definition *uns_int16* is an unsigned integer who's size is 16 bits and *int32* is a signed integer who's size is 32 bits.

The *local_tag* field in the *octet_string* is used to indicate the local data type that is pointed to by the *string* field. The following types are defined for this field.

NOTE

This list may need to be expanded.

- MAN_LC_BOOLEAN
- MAN_LC_INTEGER
- MAN_LC_BITSTRING
- MAN_LC_OCTET_STRING
- MAN_LC_NULL
- MAN_LC_OBJECT_ID

B.2 ISO Scope Type

The ISO scope structure is used to encode the *iso_scope* parameter for the directive routines.

```
typedef struct {
    uns_int16   choice;
    int32      scope ;
} iso_scope ;
```

Where the *choice* field is one of *msi_scope_integer*, *msi_scope_individual_levels*, or *msi_scope_base_to_nth_level*. If *choice* is *msi_scope_integer*, then the *scope* field is one of *msi_scope_int_baseobject*, *msi_scope_int_firstlevel* or *msi_scope_int_wholesubtree*.

NOTE

Expand this explanation.

In this structure definition *uns_int16* is an unsigned integer who's size is 16 bits and *int32* is a signed integer who's size is 32 bits.

B.3 Target Name

The target name data type is used to identify the agent that is the target of the association and/or directive. It is defined by the following ASN.1 template;

```
System-title ::= CHOICE { ddn [0] IMPLICIT DirectoryDN,
             oid [1] IMPLICIT OBJECT IDENTIFIER}

DirectoryDN ::= {joint-iso-ccitt ds(5) modules(1)
            informationFramework(1).DistinguishedName
```

The *DirectoryDN* is the form that is used by this interface.

For the purposes of the MSI interface, the *AVL* data type will be used to pass this information.

NOTE

The exact contents of the DirectoryDN needs to be the subject of implementor's agreements. There is currently work being done in the OIM working group to define the contents of the DirectoryDN to be used for identifying agents.

B.4 Access Control Structure

The access control information needed for directives and events is defined to be an ASN.1 EXTERNAL type. The definition of the encoding for the ASN.1 EXTERNAL type is found in ISO 8824. The form chosen for this interface is shown in the following ASN.1 macro;

```
AccessInfo ::= [UNIVERSAL 8] IMPLICIT SEQUENCE {
                  AccessMethod            OBJECT IDENTIFIER,
                  CHOICE {
                      AccessInformation       [1] IMPLICIT OCTET STRING
                         }
                  };
```

The information for the access control function is passed through the MSI interface using this structure.

```
typedef struct {
    object_id      *access_method;
    avl            *access_information ;
    } access_info ;
```

NOTE

The definitions of the methods and the data to be passed for each will need to be the topic of implementor's agreements.

B.5 Object Identifier Type and Routines

The data type used for object identifiers is:

```
typedef struct {
    uns_int16           count ;
    uns_int32          *(value[]) ;
} object_id ;
```

In this structure definition *uns_int16* is an unsigned integer who's size is 16 bits and *uns_int32* is an unsigned integer who's size is 32 bits.

In order to simplify the processing of object identifiers, the following utility routines are proposed.

B.5.1 moss_create_oid

```
object_id *
moss_create_oid(
    int16            count,
    uns_int32       *values,
    );
```

B.5.1.1 Description

This routine takes an array of integer values and creates a variable of type *object_id* from it. If there was an error, the returned pointer will be NULL.

B.5.1.2 Parameters

Parameter	Direction	Status	Description
values	Input	Required	A pointer to an array of integers that are the elements of the object identifier.
count	Input	Required	The number of entries in the values array.
oid	Output	Required	The pointer to the *object_id* that is created.

B.5.2 moss_free_oid

```
man_status
moss_free_oid(
    object_id   *oid
    );
```

B.5.2.1 Description

This routine frees the storage for a variable of type *object_id*.

B.5.2.2 Parameters

Parameter	Direction	Status	Description
oid	Input	Required	The pointer to the *object_id* to be released.

B.5.2.3 Return Values

Return Codes	Description
MAN_C_SUCCESS	The operation completed successfully.

B.5.3 moss_get_oid_len

```
int
moss_get_oid_len(
    object_id   *oid,
    );
```

B.5.3.1 Description

This routine takes a pointer to an *object_id* and returns the "length", that is, the count of the number of elements in it.

B.5.3.2 Parameters

Parameter	Direction	Status	Description
oid	Input	Required	A pointer to an *object_id*.
length	Output	Required	The number of elements in the object identifier.

B.5.4 moss_parse_oid

```
uns_int32
moss_parse_oid(
    object_id       *oid,
    int16              element_number,
    );
```

B.5.4.1 Description

This routine takes a pointer to an *object_id* and returns the value of one of the elements in the object identifier. The element number to be returned is passed in the *element_number* parameter and the value of that element is returned as the return value of the function.

B.5.4.2 Parameters

Parameter	Direction	Status	Description
oid	Input	Required	A pointer to an *object_id*.
element_number	Input	Required	The element number of the object identifier to be returned.

B.5.5 moss_compare_oid

```
int
moss_compare_oid(
    object_id   *oid1,
    object_id   *oid2
    );
```

B.5.5.1 Description

This routine takes two object identifiers and compares them for equality. The result of the comparison is the return value from the routine.

B.5.5.2 Parameters

Parameter	Direction	Status	Description
oid1	Input	Required	The pointer to the first *object_id*.
oid2	Input	Required	The pointer to the second *object_id*.

B.5.5.3 Return Values

Return Codes	Description
MAN_C_EQUAL	The two object identifiers are equal.
MAN_C_NOTEQUAL	The two object identifiers are not equal.

B.5.6 moss_oid_to_text

```
char *
moss_oid_to_text(
    object_id   *oid,
    char        *sep,
    char        *prefix,
    char        *suffix,
    );
```

B.5.6.1 Description

This routine takes an object identifier and creates a printable string version of it. The form of the returned string is *prefix ele1 sep ele2 sep ele3 suffix*. Where *prefix* is the prefix string, *ele#* stands for one element of the object identifier, *sep* is the printable character that is used as the element separator, and *suffix* is the suffix string. For example if the prefix string were "< ", the separator were " >< ", and the suffix string were " >", then an output from this routine would look like *< 1 >< 23 >< 45 >*. In case of error, a NULL pointer is returned. The memory to hold the string is allocated by the routine and must be released by the user.

B.5.6.2 Parameters

Parameter	Direction	Status	Description
oid	Input	Required	The pointer to the *object_id*.
sep	Input	Optional	The printable character or characters that are used as the element separator. If this is NULL or points to a non-printable character, then the separator will be a period.
prefix	Input	Optional	The printable character or characters that are used as a prefix to the printable version of the object identifier. If this is NULL or points to a non-printable character, then the prefix will be a space.
suffix	Input	Optional	The printable character or characters that are used as a suffix for the printable version of the object identifier. If this is NULL or points to a non-printable character, then the suffix will be a space.

B.5.7 moss_text_to_oid

```
object_id *
moss_text_to_oid(
    char        *text,
    );
```

B.5.7.1 Description

This routine takes a null terminated string and translates it to an object identifier. If there is an error, then a NULL pointer is returned.

B.5.7.2 Parameters

Parameter	Direction	Status	Description
text	Input	Required	A pointer to a null terminated string version of the object identifier.

B.5.8 moss_concat_oids

```
object_id *
moss_concat_oids(
    object_id   *oid1,
    object_id   *oid2,
    );
```

B.5.8.1 Description

This routine takes two object identifiers and produces a third that is the concatenation of the first two. That is oid1 + oid2 = oid3. If there is an error, then a NULL pointer is returned.

B.5.8.2 Parameters

Parameter	Direction	Status	Description
oid1	Input	Required	The pointer to the first *object_id*.
oid2	Input	Required	The pointer to the second *object_id*.

B.6 Instance Name Type and Routines

These routines are used to create and manipulate instance names. An instance name is stored in an AVL structure and can, in fact, be built using standard AVL manipulation routines. These routines simply provide a somewhat shorted for of creating these commonly used parameters. Note that each of these routines assume that the AVL handle they are passed has already been initialized using *moss_avl_init*.

B.6.1 moss_create_nonspec_name

```
man_status
moss_create_nonspec_name(
                        avl             *name,
                        octet_string    *value
                ) ;
```

B.6.1.1 Description

This routine creates a name in the AVL that is encoded as the non–specific form.

B.6.1.2 Parameters

Parameter	Direction	Description
name	Input	The address of an allocated variable of type *avl*.
value	Input	An octet_string that contains the value of the nonspecific name.

Table 1: Return Status

Return Values	Description
MAN_C_BAD_PARAMETER	The name pointer is NULL.
MAN_C_ALREADY_INITIALIZED	The name has already been initialized.
MAN_C_SUCCESS	The routine was successful.

B.6.2 moss_create_dist_name

```
man_status
moss_create_dist_name(
                        avl             *name,
                ) ;
```

B.6.2.1 Description

This routine creates a name in the AVL that is to be encoded in the distinguished name form.

B.6.2.2 Parameters

Parameter	Direction	Description
name	Input	The address of an allocated variable of type *avl*.

Table 2: Return Status

Return Values	Description
MAN_C_BAD_PARAMETER	The name pointer is NULL.
MAN_C_ALREADY_INITIALIZED	The name has already been initialized.
MAN_C_SUCCESS	The routine was successful.

B.6.3 moss_rnd_start_set

```
man_status
moss_rdn_start_set(
                        avl                 *name
                )
```

B.6.3.1 Description

This routine marks the variable pointed to by *name* to indicate the start of a relative distinguished name. The actual elements of the name are added using the appropriate *moss_avl* routines and the name is marked as finished by using the *moss_rdn_end_set* routine. The variable pointed to by the *name* parameter must have already been initialized by a call to *moss_avl_init*.

B.6.3.2 Parameters

Parameter	Direction	Description
name	Input	The address of an initialized variable of type *avl*.

B.6.3.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The name pointer is NULL.
MAN_C_NOT_INITIALIZED	The name parameter has has not yet been initialized.
MAN_C_SUCCESS	The routine was successful.

B.6.4 moss_rdn_set_end

```
man_status
moss_rdn_set_end(
                        avl                 *name
                )
```

B.6.4.1 Description

This routine marks the end of a relative distinguished name set. The distinguished name, of which this set is a part, can then be further defined by adding another relative distinguished name or ended using the *moss_end_distinguished_name* routine.

B.6.4.2 Parameters

Parameter	Direction	Description
name	Input	The address of an initialized variable of type *avl*.

B.6.4.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The name pointer is NULL.
MAN_C_NOT_INITIALIZED	The name parameter has has not yet been initialized.
MAN_C_SUCCESS	The routine was successful.

B.6.5 moss_end_distinguished_name

```
man_status
moss_end_distinguished_name(
                        avl                 *name
                )
```

B.6.5.1 Description

This routine marks the end of a distinguished name pointed to by *name*.

B.6.5.2 Parameters

Parameter	Direction	Description
name	Input	The address of an initialized variable of type *avl*.

B.6.5.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The name pointer is NULL.
MAN_C_NOT_INITIALIZED	The name pointer has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The name pointer has not been prepared with a call to *moss_start_distinguished_name*.
MAN_C_SUCCESS	The routine was successful.

B.7 Directive Filter Type and Routines

When a management operation is issued to a target managed object, the application of the operation can be restricted by the use of a *filter*. This filter evaluates to a boolean expression. It is evaluated for each instance that initially qualifies for the management operation. The management operation shall be executed by the managed object if and only if, the filter expression evaluates to TRUE. A simple filter consists of an attribute ID, the attribute value, and a relational operator. The relational operators allowed in the *CMIS_filter_level1* parameter are:

CMIS_equality
CMIS_substring
CMIS_gte
CMIS_lte
CMIS_present
CMIS_subset
CMIS_superset
CMIS_intersection

If the relational operator is *CMIS_substring*, then the *CMIS_filter_level2* parameter is one of:

CMIS_initial_string
CMIS_any_string
CMIS_final_string

A more complex filter consists of multiple sets of attribute ID, attribute value and relational operators, tied together by logical operators. These logical operators are:

AND
OR
NOT

The following routines are used by the management application to build a *filter*:

moss_init_cmise_filter
moss_finish_cmise_filter
moss_free_cmise_filter
moss_add_cmise_filter_item
moss_start_cmise_or_set
moss_start_cmise_and_set
moss_add_cmise_not
moss_end_cmise_andornot_set

B.7.1 moss_init_cmise_filter

```
man_status
moss_init_cmise_filter(
                avl    *filter
               ) ;
```

B.7.1.1 Description

This routine initializes a CMISE filter.

B.7.1.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated AVL.

Table 3: Return Status

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_ALREADY_INITIALIZED	The filter has already been initialized.
MAN_C_SUCCESS	The routine was successful.

B.7.2 moss_finish_filter

```
man_status
moss_finish_cmise_filter(
                 avl    *filter
                )
```

B.7.2.1 Description

The *moss_finish_cmise_filter* routine marks the end of a filter. This routine must be called to properly balance the filter construction.

B.7.2.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated AVL.

Table 4: Return Status

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter is not under construction.
MAN_C_SUCCESS	The routine was successful.

B.7.3 moss_free_cmise_filter

```
man_status
moss_free_cmise_filter(
                avl    *filter
               )
```

B.7.3.1 Description

This routine frees the storage associated with a CMISE filter.

B.7.3.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated filter data structure.

B.7.3.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_SUCCESS	The routine was successful.

B.7.4 moss_add_cmise_filter_element

```
man_status
moss_add_cmise_filter_item(
                avl    *filter,
                object_id *attribute_id,
                int32   tag,
                octet_string octet,
                cmis_filter_level1 operation1,
                cmis_filter_level2 operation2
               )
```

B.7.4.1 Description

This routine adds a filter element to a filter. The element consists of an attribute ID, an optional attribute value, and a comparison operation to apply between the attribute value specified in the filter element and the current value of the attribute.

B.7.4.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated AVL.
attribute_id	Input	A pointer to the attribute ID that will be checked.
tag	Input	The ASN.1 data type of the attribute's value.
octet	Input	A pointer to the reference value which the comparison is to evaluate.
operation1	Input	The first level operation defined by the ISO FilterItem.
operation2	Input	The second level operation defined by ISO FilterItem (only interpreted if operation1 is *CMIS_substring*).

B.7.4.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter element is not under construction.
MAN_C_SUCCESS	The routine was successful.

B.7.5 moss_start_cmise_or_set

```
man_status
moss_start_cmise_or_set(
                avl    *filter
               ) ;
```

B.7.5.1 Description

The *moss_start_cmise_or_set* routine indicates that what follows will be evaluated and logically OR'd together. Once the OR set has been completed, a call to *moss_end_cmise_andornot_set* must be performed.

B.7.5.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated filter.

B.7.5.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter element is not under construction.
MAN_C_SUCCESS	The routine was successful.

B.7.6 moss_start_cmise_and_set

```
man_status
moss_start_cmise_and_set(
                 avl     *filter
               )
```

B.7.6.1 Description

The *moss_start_cmise_and_set* routine indicates that what follows will be evaluated and logically AND'ed together. Once the AND set has been completed, a call to *moss_end_cmise_andornot_set* must be performed.

B.7.6.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated filter.

B.7.6.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter element is not under construction.
MAN_C_SUCCESS	The routine was successful.

B.7.7 moss_start_cmise_not_set

```
man_status
moss_start_cmise_not_set(
                avl    *filter
               ) ;
```

B.7.7.1 Description

The *moss_add_cmise_not* routine indicates that what follows will be evaluated and the result will be negated. Once the not set has been completed, a call to *moss_end_cmise_andornot_set* must be performed.

B.7.7.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated filter.

B.7.7.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter element is not under construction.
MAN_C_SUCCESS	The routine was successful.

B.7.8 moss_end_cmise_andornot_set

```
man_status
moss_end_cmise_andornot_set(
                 avl    *filter
                ) ;
```

B.7.8.1 Description

This routine is used to indicate the end of either an OR, AND, or NOT set.

B.7.8.2 Parameters

Parameter	Direction	Description
filter	Input	The address of an allocated filter.

B.7.8.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The filter pointer is NULL.
MAN_C_NOT_INITIALIZED	The filter has has not yet been initialized.
MAN_C_NOT_CONSTRUCTED	The filter element is not under construction.
MAN_C_SUCCESS	The routine was successful.

APPENDIX C

AVL UTILITY ROUTINES

C.1 Introduction

The *AVL* type is used to represent a series of management values. The *AVL* data type will be used to support the routines in management services API. It is the data type which actually appears within the procedure call arguments to the management services implementation. This mechanism permits the programmer to supply to and receive from these routines a list of values whose types are not known until runtime (runtime binding vs. compile-time binding). This requirement for a general-purpose structure arises from the need for management software to perform operations upon objects about which the software has no compile-time knowledge.

AVLs are an opaque type. This means that an AVL is referred to only by an opaque reference, called a handle, and all creation and modification of AVL's is done using the AVL MOSS routines. The actual opaque handle can be allocated on the stack, or by allocating memory.

In order to move AVL lists between address spaces, you can use functions which convert between an AVL and a flat buffer representation. The buffer representation can then be processed using any of the traditional methods for exchanging data between processes or exchanging data between user and kernel address space.

The basic AVL consists of a 4-tuple, (object ID, modifier, tag, octet string). The value represented by the 4-tuple can be any of the simple data types encoded as an octet string format. For more complicated data types a *construction* is performed. That is to say, the complex records must be broken down into the basic octet string representation before being added to the AVL. The *moss_avl_start_construct* and *moss_avl_end_construct* routines support these operations. Basically the construction routines allow the programmer to insert the equivalent of a field start and field end marker in the AVL stream.

The *tag* field in an AVL element is used to indicate the ASN.1 tag that is to be used to encode the data portion of the that element. The *tag* is is of type *unsigned long int* with the bits interpreted in the following manner.

Table 5: AVL Tag Layout

Bits	Meaning
31 - 30	Tag class. One of UNIVERSAL, PRIVATE, CONTEXT_SPECIFIC, or APPLICATION.
29	Bit set indicates that this is a constructed type.
28 - 0	Tag number. Context specific.

The *modifier* field in an AVL element is used in the *msi_set_attributes* operation to indicate the type of set operation that is to be applied to each attribute in the list. In the return data from the *msi_get_attributes* and *msi_set_attributes* operations, it is used to indicate the status of the operation on each individual attribute.

C.1.1 AVL functions

The functions used to construct and manipulate the avl type are:

Table 6: AVL Routines

Routine Name	Purpose
moss_avl_init	Initialize a variable of type AVL.
moss_avl_free	Deallocate the values in the AVL.
moss_avl_add	Add a value to the AVL list.
moss_avl_reset	Prepare to read AVL list.
moss_avl_point	Return a pointer to a value in the AVL list.
moss_avl_start_construct	Start a constructed type.
moss_avl_end_construct	End a constructed type.
moss_avl_to_buf	Create a buffer representation of the AVL.
moss_avl_from_buf	Create an AVL from a buffer representation.

C.2 AVL Utility Routine Descriptions

C.2.1 moss_avl_init

```
man_status
moss_avl_init(
             avl *avl_handle
            )
```

C.2.1.1 Description

In order to utilize an AVL, you must both allocate an *avl_handle* and then initialize it. The *avl_handle* can be allocated like any other structure, by declaring it at compile time or allocating it from dynamic memory at runtime. Once it has been allocated, it is then initialized with this routine.

C.2.1.2 Parameters

Parameter	Direction	Description
avl_handle	Input/Output	The address of the AVL handle.

C.2.1.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_ALREADY_INITIALIZED	The AVL has already been initialized and may contain valid data.
MAN_C_SUCCESS	The call was successful.

C.2.2 moss_avl_free

```
man_status
moss_avl_free(
             avl *avl_handle
            )
```

C.2.2.1 Description

When an AVL is no longer needed, the buffer memory associated with it must be released. The routine *moss_avl_free* properly releases all memory associated with the AVL. This does not affect the AVL handle except that it is marked to indicate that it no longer points to a valid AVL.

C.2.2.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle. The AVL handle is marked as invalid.

C.2.2.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain data to be freed.
MAN_C_SUCCESS	The call was successful.

C.2.3 moss_avl_add

```
man_status
moss_avl_add(
                avl *avl_handle ,
                object_id *oid ,
                unsigned long int modifier ,
                unsigned long int tag ,
                octet_string *octet
            )
```

C.2.3.1 Description

The *moss_avl_add* routine is used to add an element to an avl. This routine *copies* the input value to the list. The AVL must have already been initialized.

C.2.3.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle.
oid	Input	The address of the object id.
modifier	Input	The modifier for this AVL element.
tag	Input	The ASN.1 data type of the added value.
octet	Input	The address of the octet_string. This can be NULL.

C.2.3.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_SUCCESS	The call was successful.

C.2.4 moss_avl_reset

```
man_status
moss_avl_reset(
              avl *avl_handle
             )
```

C.2.4.1 Description

Before reading from the AVL, it is necessary to reset the internal read pointer of the AVL to the beginning, much like resetting a file pointer to point to the beginning of the file before reading. The routine *moss_avl_reset* will adjust this internal pointer to the beginning of the AVL.

C.2.4.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle.

C.2.4.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_SUCCESS	The call was successful.

C.2.5 moss_avl_point

```
man_status
moss_avl_point(
             avl *avl_handle ,
             object_id **oid ,
             unsigned long int *modifier ,
             unsigned long int *tag ,
             octet_string **octet,
             int *last_one
            )
```

C.2.5.1 Description

The *moss_avl_point* routine is used to reference an element of an AVL. This routine does not copy the element out of the AVL. It writes into the pointers passed to the routine. This avoids unnecessary copying of information. The data referenced by the pointers returned from the call must not be written to or the integrity of the AVL may be disrupted.

This routine operates on the current AVL element and advances the internal AVL pointer to the next element.

C.2.5.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle.
oid	Output	The address of a pointer to an object id.
modifier	Output	The address of the element's modifier.
tag	Output	The address of the element's ASN.1 tag.
octet	Output	The address of a pointer to an octet_string.
last_one	Output	The address of the last-one indicator. This is set to TRUE if the current value is the last value or FALSE if there are more elements in the AVL.

C.2.5.3 Return Values

Return Values	Description
MAN_C_NO_ELEMENT	The AVL is already at the end of the list.
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_SUCCESS	The call was successful.

C.2.6 moss_avl_start_construct

```
man_status
moss_avl_start_construct(
               avl *avl_handle ,
               object_id *oid ,
               unsigned long int modifier ,
               unsigned long int tag ,
               octet_string *octet
                       )
```

C.2.6.1 Description

The *moss_avl_start_construct* routine is used to signal the start of a constructed type. The *tag* argument is checked to insure that the constructed bit (bit 29) is set. Note that while the *octet* parameter is present, it is generally not used.

C.2.6.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle.
oid	Input	The address of the object id.
modifier	Input	The modifier for this AVL element.
tag	Input	The ASN.1 data type of the added value.
octet	Input	The address of the octet_string. This can be NULL.

C.2.6.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_CONSTUCTED	The contructed bit in the tag is zero.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_SUCCESS	The call was successful.

C.2.7 moss_avl_end_construct

```
man_status
moss_avl_end_construct(
                        avl *avl_handle
                       )
```

C.2.7.1 Description

The *moss_avl_end_construct* routine is used to signal the end of a constructed type.

C.2.7.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the AVL handle.

C.2.7.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_NOT_CONSTRUCTED	The AVL is not in a construction state.
MAN_C_SUCCESS	The call was successful.

C.2.8 moss_avl_add_cons_field

x>(AVL Utility Routinesmoss_avl_add_cons_field)

```
man_status
moss_avl_add_cons_field(
                         avl *avl_handle ,
                         unsigned long int explicit ,
                         unsigned long int tag ,
                         octet_string *octet
                        )
```

C.2.8.1 Description

Elements within a constructed type, often require the inclusion of a context specific tag. This is most often seen in the cases where the construct contains an ASN.1 CHOICE. The *moss_avl_add_cons_field* routine is used to add this type of element to an AVL.

C.2.8.2 Parameters

Parameter	Direction	Description
avl_handle	Input	Address of the avl handle.
explicit	Input	The supplemental information for this value
tag	Input	The ASN.1 data tag of the added value
octet	Input	The address of the octet_string. This can be NULL.

C.2.8.3 Return Values

Return Values	Description
MAN_C_SUCCESS	The call was successful
MAN_C_BAD_PARAMETER	The AVL handle was NULL
MAN_C_NOT_CONSTRUCTED	The AVL is not in a construction state
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data
MAN_C_PROCESSING_FAILURE	Error with memory allocation

C.2.9 moss_avl_to_buf

```
man_status
moss_avl_to_buf(
               avl *avl_handle ,
               char **buffer ,
               int *len
              )
```

C.2.9.1 Description

The routine *moss_avl_to_buf* creates a *flat* representation of the avl. The purpose of this routine is to create a representation of an AVL that can be passed across address spaces.

This routine allocates the storage necessary to accommodate the flat representation of the AVL. The first sizeof(int) bytes of the return buffer contains the length of the buffer allocated, though not including the space needed to accommodate the length (again, sizeof(int)).

This routine resets the internal read pointer of the AVL.

C.2.9.2 Parameters

Parameter	Direction	Description
avl_handle	Input	Address of the avl handle.
buffer	Output	The address of a pointer for the buffer returned.
len	Output	The address for the length of the buffer.

C.2.9.3 Return Values

Return Values	Description
MAN_C_NO_ELEMENT	The AVL is empty.
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_NOT_CONSTRUCTED	The construction state of the AVL is invalid. (Mismatched begin/end)
MAN_C_SUCCESS	The call was successful.

C.2.10 moss_avl_from_buf

```
man_status
moss_avl_from_buf(
                  avl *avl_handle ,
                  char *buffer
                 )
```

C.2.10.1 Description

The routine *moss_avl_from_buf* creates an AVL structure from a flat representation. This is the inverse function of *moss_avl_to_buf*.

The *avl_handle* must be first initialized by *moss_avl_init*. After the transformation, the internal read pointer of the AVL is reset to the first element. Note that the length of the buffer is not passed explicitly; the length is in the beginning of the buffer.

C.2.10.2 Parameters

Parameter	Direction	Description
avl_handle	Input	The address of the initialized AVL handle.
buffer	Input	Address of the buffer containing the flat AVL.

C.2.10.3 Return Values

Return Values	Description
MAN_C_BAD_PARAMETER	The AVL handle was NULL.
MAN_C_NOT_INITIALIZED	The AVL has not been initialized and does not contain valid data.
MAN_C_SUCCESS	The call was successful.

APPENDIX D

EVENT MODEL REFERENCE

D.1 Overview of the Model

The major components of the model are the event *sources* and *sinks*, the Event Dispatcher (EVD), and the managing systems that configure event management on the managed systems.

Sources are the managed objects that detect the occurrence of their architected events and issue event notifications. Sinks are consumers of event reports, providing arbitrary processing of the information they contain. There can be multiple sinks with arbitrary different functionality. Sinks are named processes that can reside on the same or different nodes as the sources.

The EVD is created as part of the system initialization process and it serves as an intermediary between the source of events; the managed objects and the consumers of events; the sinks. A managed object posts event notifications to the local EVD; the EVD forms event reports from the notifications which are sent to sinks (subject to filtering).

An event stream is an abstraction representing a long-lived relationship between an EVD and a sink, one that survives communication, process, and system failures. The source endpoint for an event stream—the outbound stream—is realized in network management terms by an event forwarding discriminator managed object. When a discriminator is created and enabled, the EVD tries to establish and maintain an application association with the single sink identified in the discriminator. The discriminator contains modifiable attributes to control when to re-attempt association establishment after communications failure, and when to terminate an idle association ("disconnect timer"). Filtering on each outbound stream is controlled by filter attributes of the discriminator. The filtering criteria on one stream are independent of the filtering criteria on any other stream.

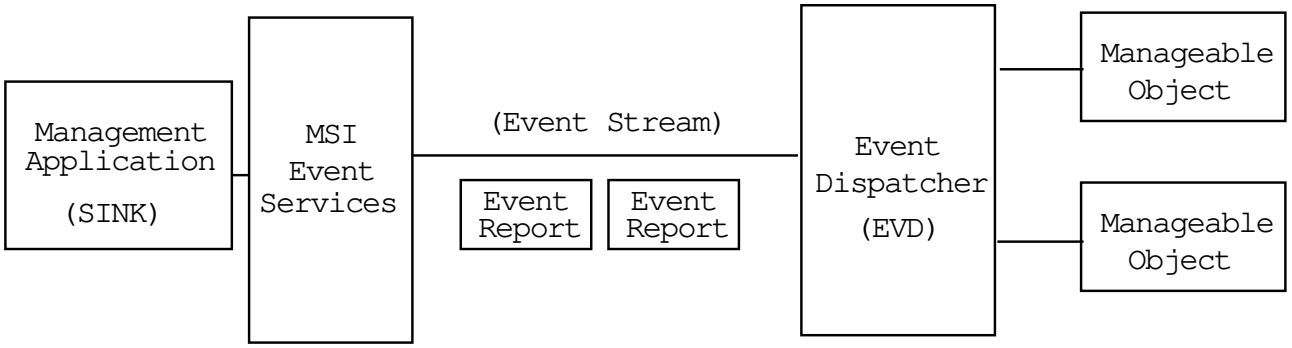
All event reports formed by the EVD from notifications are distributed to each enabled outbound stream where they are filtered and buffered (FIFO) awaiting transmission to a sink. Event reports are transmitted when communication to the sink is possible. If communication to the sink is not possible, the event reports are held locally until it is, or the event forwarding discriminator is deleted. The EVD periodically tries to establish an association to a sink, controlled by the value of the "connect retry timer" discriminator attribute. Event reports are transmitted using the *report_event* routine.

Using management operations, managing systems configure and control the event reporting capabilities of managed systems: they create event forwarding discriminators, the associated filters, and control event report communications. The communications (application associations) used to exchange these management operation requests/responses (Create, Set, etc.) are independent of the communications used to report events; the managing system(s) may be different from those containing either the EVD or sink. Even when the managing system does contain a sink, two separate application associations are used: one to receive event reports, the other for management operations.

Creating or deleting an event forwarding discriminator causes an EVD creation/deletion event to be posted. Changing the value of the "event report destination" attribute of a discriminator has no affect on any existing connection to a sink, only on subsequent connections.

Managing systems can inhibit the reporting of specific event types from managed objects to any sink by manipulating the discriminator filter attributes. Whenever a filter is changed, a "Change Filter" event is issued for that discriminator.

Figure 3: Event Model Diagram



D.2 Definitions

- Event Dispatcher (EVD)—A local process on a node that accepts event notifications from managed objects on the node, forms event reports from the notifications, duplicates and filters them, establishes communications with event sinks, and transmits the reports to the sinks. In management terms the EVD is a managed object; parent of Event Forwarding Discriminator managed objects.
- Event—The occurrence of a significant normal or abnormal condition, detected by a managed object.
- Event Notification—A local representation of the parameters defined by managed object designers to be associated with each event. Includes event type and parameters specific to each event type. Event notifications are passed by managed objects to the local EVD using a local mechanism, not subject to OSI standardization. Event notifications are mapped with one-to-one correspondence onto externally visible event reports.
- Event Filter—A means of selectively disabling or enabling the reporting of events based on managed object class and event type, or specific managed object instance and event type. Further filtering may be done using event parameter value assertions.
- Event Report—An externally visible data structure that completely describes a single event occurrence. Event reports are formed from event notifications and sent (barring filtering) to event sinks.
- Event Stream—An event stream is an abstraction that represents a long-lived association or relationship between a specific EVD and a specific sink, the events reported on that association, and the communications and buffer resources to support it. Note that in this context, "association" does not mean application layer association; an event stream survives the loss of an underlying application layer association. An event stream has a source endpoint (outbound stream) at the EVD and a sink endpoint (inbound stream) at the event sink.
- Event Type—An event type is a code (object id) that identifies the nature of an event with respect to its managed object definition. Some event types identify generic events and are used by many managed objects; others identify managed object specific events.

- Outbound Stream—The source endpoint for an event stream, where event reports that have passed filtering criteria are queued in FIFO order waiting transmission to the event sink. The concept of a FIFO queue is used to model the property of ordering, such that "later" events are reported after "earlier" events. There is no requirement that an implementation actually structure buffer resources in this way. Any implementation is permitted that preserves the ordering property. In network management terms, the outbound stream is represented by an event forwarding discriminator managed object, a child of the EVD managed object.
- Event Sink—An Event Sink (or sink) is a process that receives, optionally filters, and processes event reports.
- Inbound stream—The sink endpoint for an event stream, where event reports are received.

D.3 Managed System Requirements

All managed systems containing managed objects that define event notifications, are required to support the event management components needed to generate and send event reports, that is, the EVD and event forwarding discriminator objects. Support for event sink-related components is optional. However, a properly configured network requires at least one event sink.

Each managed object class must define the event types it uses, including their parameters and corresponding codes. These may be imported generic event types or managed object specific event types.

At run-time, each managed object instance must:

- Generate a unique id for the object instance when it is created, uniquely identifying this incarnation of the object class
- Timestamp each event occurrence, as close to the true time of occurrence as possible
- Create a unique id for each event occurrence
- Post event notifications to the EVD, including the event parameters, the timestamp, the unique id for the event, and the unique id representing the managed object instance
- Post event notifications (from a single instance of a managed object) in a best-effort temporal order; a managed object specification may require that certain sets of events must always be posted in strict temporal ordering

D.4 Event Reports

Event reports are the data used to describe each occurrence of an event. In addition to the event-specific parameters, they include the managed object id, the timestamp, and the unique ids. They are totally self-describing and do not depend on connection context or other external state. This implies that a single network-wide database could potentially store all event reports, without regard to sinks, discriminators, and so on, with no loss of information.

D.5 Event Steams

The event stream abstraction defines a long-lived management relationship between an EVD and a sink, whose data consists of a sequence of event reports. An event stream and the events associated with the stream (discussed below) also provides information on the absence of event reports and the potential loss of event reports.

Event streams serve to enhance the reliability of event reporting by minimizing the effects of communications failures or sink failures. Event streams persist across these failures and periodically re-attempt communications. They do not disconnect due to lack of communication.

D.6 EVD events

An important goal of the model is to always either:

- Provide all event reports that have not been filtered, or
- Provide information that event reports may have been lost.

In other words, the goal is to detect all possible loss of event reports. To support this goal, a number of EVD events are defined which are entered into an event stream by the EVD. These events differ from "normal" events only in that they represent changes in the event management system itself. For example:

- The "Events Lost" event denotes that the outbound stream is missing some event reports.
- The "Change Filter" event indicates that the filter value has been changed, thereby changing which events will be reported.
- The "Shutdown" event indicates that the outbound stream is trying to close down the association in an orderly fashion. It triggers the sink to terminate, ensuring no events are lost in the shutdown.
- The "Disconnect" event indicates that the association was broken and that events in transit may have been lost.
- The "Enable Stream" event provides a marker indicating when the event forwarding discriminator was enabled, therefore implying that events may have been lost before the discriminator was enabled.

In contrast to normal events, by default, EVD events are filtered out by all discriminators other than the one they apply to. However, they may also be reported on other streams by appropriately manipulating the filters of other discriminators.

Just like normal events, EVD events may be lost due to resource limitations or communications failures. However, an implementation should attempt not to lose the "Events Lost" event due to resource limitations.

D.7 Sinks

Sinks are application processes that process event reports in an arbitrary way. They may write event reports to the system console, or to a log file, or may provide highly tailored operations (ringing alarm bells or flashing red lights!).

D.8 Filters

Event reports may be filtered at the source node. Each event forwarding discriminator contains filter attributes allowing independent management of event forwarding for different sinks.

The outbound stream filters events at the source node. All event notifications posted to the EVD after a discriminator's filter is changed are filtered according to the new value of the filter. Some ordering anomalies may occur if an event with a timestamp prior to the time of the Change Filter event is posted after the filter is changed. To avoid these timing anomalies, managed objects should make their best efforts to post event notifications as soon as possible after detecting the event.

D.8.1 Filter Structure and Searching

The filtering structure needs to be flexible enough to deal with the following cases:

- Block (i.e. discard) all the events from a given class of managed object (that is all instances of that class)
- Pass an event from a specific instance of a managed object, while blocking the same event from all other instances, or vice-versa
- Representation of event parameter value assertions

NOTE

This contribution deals with the first two bullets. It is believed that the scheme presented can be easily extended to include the third bullet by adding some representation of parameter value assertions to the filter data type element.

To support these needs, an event forwarding discriminator contains the following three filter attributes:

- Specific Filter–A set-valued attribute (of complex type) that defines the filtering action for particular instances of managed object classes. An element of this datatype consists of: *{object_instance, SET OF {event_class,filter_action}}* where filter_action = {Pass,Block};
- Global Filter–A set-valued attribute (of complex type) that defines the filtering action based on managed object class. An element of this datatype consists of: *{object_class, SET OF {event_class,filter_action}};*
- Catch-All Filter–A single-valued attribute that defines the filtering action if a search of Specific Filter and Global Filter fails to find a match.

To filter an event, a search algorithm is defined that always returns either Pass or Block. The search rules defined are:

1. First, search the specific filter. If an element matching the managed object instance name and event type is found, use the value (Pass or Block) and terminate the search.
2. Second, search the global filter. If an element matching the managed object class name and event type is found, use the value (Pass or Block) and terminate the search.
3. Finally, if there was no match in either the above filters, the catch-all filter value is used (Pass or Block).

The catch-all filter can be manipulated using the Set management operation. Since the specific and global filters are complex datatypes, specific Action directives are defined which allow them to be manipulated:

- Pass(instance, class, event type)–Pass the event. If the instance name argument is non-null, the operation is applied to the Specific Filter attribute of the discriminator object. If the class name attribute is non-null, the operation is applied to the Global Filter attribute. Both instance name and class name may be non-null, in which case the operation applies to both filters.
- Block(instance, class, event type)–Block the event.
- Ignore(instance, class, event type)–Ignore (remove) the event.
- TestEvent(instance, event type; filter action, filter type)–Returns the current filtering action for the event. Filter action returns Pass or Block; filter type returns Specific, Global or Catch-All.

D.9 Flow of Events

The sequence of steps involved in reporting an event is:

1. The source managed object posts the event notification to the local EVD.
2. The EVD generates an event report from the notification and distributes this locally to all enabled event forwarding discriminators.
3. The event report is filtered on a per event forwarding discriminator basis.
4. If the event report passes the filter, the *report_event* routine is used to send the event report to the sink.

GLOSSARY

Actions: Object operations that directly cause a state transition in the object instance.

Agent: The component of a managed object that performs management functions. It is responsible for routing management directives from a client, received via a management access protocol, to the managed object and returning the result to the client.

Agent address: The information needed by a director to establish communication with the agent's management interface.

ASN.1: Abstract Syntax Notation 1. A method of describing data in a system-independent manner.

Association: An object used to configure and represent a connection with a server.

Attribute: Information about an object instance. An attribute has a value that may be examined and possibly modified.

Authentication: The function of verifying the identity of a person or process. Authentication is done when an association is established.

Authorization: The function of determining whether or not a person or process has the right to perform an operation.

Characteristic: An object attribute that can be changed or modified and that affects the behavior of an entity. Characteristics are never modified as the result of a state change in the object, but can be changed or modified by a management user.

Class name: The name of an object class.

CMIP: Common Management Information Protocol, an ISO/OSI international standard protocol for the exchange of management directives and responses between management systems and managed objects.

Containment: The relationship between an object class and its subordinate objects.

Counter: An attribute that contains information on the number of times an event has occurred since the counter was initialized to zero.

Directive: A request/reply operation between a management application and a managed object agent where the director is the requester of the operation.

Filter: A modifier that is applied to a management directive that controls the object instances to which a directive applies.

Identifiers: Object attributes such as names and/or network addresses that are used to distinguish specific instances of objects.

Management access protocol: An application layer protocol between a management application and an agent.

Management Application: A management system implemented in software that interacts with a management user, initiates management operations on behalf of the management user, coordinates management activities with managed objects, and provides high-level management applications.

Manageable Object: An individual, manageable piece of a system. A manageable object has attributes that describe it and a name that identifies it. Sometimes refered to as a managed object.

Management protocol: See management access protocol.

Object: An identifiable hardware or software element belonging to a system or network environment that is manageable or can be listed for management purposes.

Object Class: A collection of manageable objects that share the same properties. Instances of a class have unique names and share a unique class identifier.

Object Identifier: A label that uniquely defines an object class.

Status: An attribute that reflects the behavior of a manageable object. Status is a reflection, or "snapshot," of an entity's state, which is of interest to management applications. Status attributes cannot be modified by management directives.

INDEX

A

Access Control Structure, 49
Association Object, 9
  Association Operations, 10
    msi_create_association_object, 10
    msi_delete_association_object, 10
  Attributes, 9
    Protocols, 9
Association Routines, 18
  msi_create_association_object, 18
  msi_delete_association_object, 20
Association Services, 3, 9
AVL Utility Routines, 76
  moss_avl_add, 78
  moss_avl_end_construct, 82
  moss_avl_free, 77
  moss_avl_from_buf, 85
  moss_avl_init, 76
  moss_avl_point, 80
  moss_avl_reset, 79
  moss_avl_start_construct, 81
  moss_avl_to_buf, 84

C

CMIS, 3

D

Directive Filter, 65
  moss_add_cmise_filter_element, 69
  moss_end_cmise_andornot_set, 73
  moss_finish_filter, 67
  moss_free_cmise_filter, 68
  moss_init_cmise_filter, 66
  moss_start_cmise_and_set, 71
  moss_start_cmise_not_set, 72
  moss_start_cmise_or_set, 70
Directive Routines, 21
  msi_create_instance, 21
  msi_delete_instance, 23
  msi_getnext_attribute, 27
  msi_get_attributes, 25
  msi_invoke_action, 31
  msi_read_reply, 33
Directive Routines (Cont.)
  msi_send_reply, 37
  msi_set_attribute, 29

E

Event Reporting Services, 16
  msi_report_event, 16
Event Services, 3, 15
Event Subscription, 15
  msi_create_subscription, 15
  msi_delete_subscription, 16
  msi_read_event, 16
Event Subscription Routines, 39
  msi_create_subscription, 39
  msi_delete_subscription, 41
  msi_read_event, 42

G

Goals, 1

I

Instance Name, 59
  moss_create_dist_name, 61
  moss_create_nonspec_name, 60
  moss_end_distinguished_name, 64
  moss_rdn_set_end, 63
  moss_rdn_start_set, 62
Introduction, 1
ISO Scope, 47

M

Management Directive Services, 3
Management Services, 11
  Directive Operations, 12
    msi_create_instance, 12
    msi_delete_instance, 12
    msi_getnext_attribute, 12
    msi_get_attributes, 12
    msi_invoke_action, 13
    msi_read_reply, 13
    msi_send_reply, 13
    msi_set_attributes, 12

O

Object Identifier Routines, 50
moss_compare_oid, 55
moss_concat_oids, 58
moss_create_oid, 51
moss_free_oid, 52
moss_get_oid_len, 53
moss_oid_to_text, 56
moss_parse_oid, 54
moss_text_to_oid, 57
Overview, 3

R

References, 2

T

Target Name Type, 48