NISTIR 4866

Network Management Support for OSI Systems (NeMaSOS) Version 2.0 Programmer's Reference Manual

Kevin G. Brady
James F. Fox
Robert Aronoff

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

NIST

NISTIR 4866

Network Management Support for OSI Systems (NeMaSOS) Version 2.0 Programmer's Reference Manual

Kevin G. Brady
James F. Fox
Robert Aronoff

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

July 1992

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin, Secretary

TECHNOLOGY ADMINISTRATION
Robert M. White, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

Preface

The work reported herein was sponsored jointly (under the NIST Joint Sponsors of the Network Management Program) by the U.S. Air Force (TIC/TISC, Scott AFB), the U.S. Army (USAISEC, Ft. Huachuca, AZ), the Defense Information Systems Agency (JTC3A, Ft. Monmouth, NJ), and the National Institute of Standards and Technology. For information on the program, contact Ms. Fran Nielsen, Manager, Network Management Group, NIST, Technology Building Room B217, Gaithersburg, Maryland 20899.

Table of Contents

1. Introduction

1.1. Organization of Programmer's Reference Manual

This Programmer's Reference Manual is intended to provide support to application programmers using the interface library of functions to CMIS/CMIP provided. The manual is divided into two basic sections; the first describes the interface to the ACSE/ROSE protocol machine provided with this release, and the second describes the CMIS/P interface.

The ACSE/ROSE interface provides a straight-forward asynchronous interface which allows manipulation of certain parameters while assuming default values for others at the ACSE/ROSE service interface. Tables 3, 4 and 5 in this section provide a quick reference for the programmer to determine which ACSE/ROSE service primitives are supported, which ISODE routines are associated with those primitives, and which interface library routines are called to perform the services. The code for these routines can be used "as is", with only a few simple modifications required to customize for the local system. Alternatively, users desiring a more comprehensive implementation, in which additional parameters can be controlled and default values are not assumed, can use the software provided as a basis for a more extensive implementation and apply modifications to the code as necessary. This ACSE/ROSE section is divided into two subsections; the first covering routines to be invoked by the user, and the second subsection covering those routines that are invoked by the user-invoked routines described in the first sub section. All routines are described in detail, thus providing the necessary insight required for the implementor to modify the code for his/her particular application. The implementation provided was used to test the CMIS/P interface and is, therefore, only as complete as necessary to accomplish that purpose. Consequently, this implementation does not provide a complete exercise of the ACSE/ROSE services (e.g., default values were used wherever possible).

The second part of this document provides a detailed description of the CMIS/P interface. The routines provided allow a user to send and receive CMIP PDUs. Tables 13, 14, 15, 16, and 17 at the beginning of this section provide a quick reference for the programmer to determine which CMIS service primitives are supported, which parameters are associated with those primitives, and which interface library routines are to be used to fill in or extract information from those parameters in the given service primitive. Also in this section, each CMIS message type and its associated data structure are described in detail. Since these data structures are manipulated by the interface library routines to hide their complexity from the rest of the user program, the complex structures representing these messages are provided in the documentation for information purposes only (e.g., for users wishing to modify the existing code). The final part of the CMIS/P interface section includes descriptions of each of the routines used to fill in information, or extract information from, the data structures representing each CMIS/P message type.

Version 2.0 of NeMaSOS implements the kernel, multiple object selection, cancel get, linked reply, and filter functional units of the CMIP/CMIS version 1 IS standards, with the restrictions imposed by OIW NMSIG Phase I implementor's agreements. This version of NeMaSOS allows for the passage of the access control parameter without providing interface support for filling or extracting the elements of this data structure.

1.2. Required Software Support

Certain software is required to support the ACSE/ROSE and CMIS/P service interface library routines. The "ISO Development Environment" (ISODE) was used to generate the routines to encode and decode the message structures, to provide the ACSE and ROSE functionality, and to provide the upper OSI layer functionality. Consequently, ISODE must be installed prior to use of this interface. ISODE 6.0 is publicly available and can be anonymous FTP'd across the Internet from uu.psi.com [136.161.128.3]. The file isode/isode-6.tar should be retrieved in BINARY mode. This is a 10.5MB tar image. The file isode/isode-6.tar.Z (3.5MB) is the compressed tar image. To install ISODE, follow the installation procedures provided as part of the distribution software. Certain configuration files must be customized for the systems involved. Information assisting the user in customizing these configuration file entries is discussed in the "NSAP addressing" sub section of the ACSE/ROSE section of this manual.

Just as ISODE provides session layer, presentation layer, and part of the application layer functionality, the lower OSI layer functionality (i.e., transport layer (TP4) and below) is provided by SUNLINK OSI for this software release. To support this configuration, SUNLINK OSI must be properly installed and configured to work with the ISODE software. Alternative approaches can be pursued but must be properly configured to work with ISODE. The ISODE installation guide and reference manuals provide guidance to possible alternative lower layer support options.

1.3. Overview of the Service Interface Routines

A user wishing to implement a basic network management system would need to:

(1) send association (ACSE) request/response messages,

(2) receive association (ACSE) request/response messages,

(3) send CMIS/P request/response messages,

(4) receive CMIS/P request/response messages,

(5) access a MIB to retrieve the requested information, and,

(6) display information contained in each message.

The interface provided listens for messages. When new connections or messages on existing connections are received, they are put into one of two queues. All messages dealing with association control (ACSE) are added to the ACSE message queue. Messages dealing with either ROSE or CMIS/P are added to the CMIP message queue. To process these messages, the user retrieves messages from the queues, extracts the necessary information from the messages, and then takes the appropriate actions based on the content of the messages. The existence of two queues enables the implementor to establish a mechanism to allow for prioritized processing of messages, if so desired.

In order to use this implementation "as is", the user need only provide in addition, the means (5 above) for accessing the MIB (i.e., the managed objects themselves) and retrieving the requested information. The interface code provided :

(1) allows the association requests and responses to be sent and received,

(2) allows the CMIS/P and ROSE messages to be sent and received,

(3) provides the means to fill the complicated structures representing these messages with no knowledge of the data structures used, and,

(4) provides the means to extract information from these messages.

Depending upon the implementor's requirements, some screen interface (6 above) might be desired to assist in input and output. The implementor can develop his own screen interface, if desired, or use the one provided. The screen interface provided in this distribution was used to test the CMIS/P implementation and was written for the SUNVIEW environment. The code which is included with this release needs to be modified by the implementor to display any pertinent information contained in the messages specific to the actual management information being conveyed.

2. ACSE/ROSE Interface

The ACSE/ROSE routines provide an example of a straight-forward use of ACSE, exercising some, but not all, options of ACSE. To simplify the association establishment negotiation process, default values where used where reasonable rather than requiring the user to fill in the parameter information. These routines can be used "as is" for those users requiring only this level of functionality. This interface was used for testing the CMIS/P interface and can be used by an implementor as an easily customizable interface to ACSE and ROSE. For those users desiring a more complete implementation exercising additional ACSE options, source code is provided which can be easily modified for those purposes. To assist the user in this process, brief descriptions of each routine follow and the source code is commented to indicate where and what kind of modifications are required.

Section 2.1 discusses the necessary entries to the locally resident directory data base required to accomplish management associativity between peer management entities, including an explanation of the syntax and semantics of the relevant network addressing. Section 2.2 provides an explanation of the queuing mechanism used to manage two queues of received messages (one for ACSE messages and one for CMIP/ROSE messages). In conjunction with that discussion, the two different data structures are shown which are used to represent the two general types of messages, either ACSE or CMIP/ROSE, and which can be stored in the respective queues. Section 2.3 presents a sample skeleton program segment representing the necessary ACSE and ROSE related function calls to establish associations and send and receive management information over those associations. The subsections within section 2.3 describe each of the interface library routines, indicating their parameters and return values, which are referenced or implied by this sample program segment. Section 2.4 presents descriptions of those library functions which can be useful in the processing of messages from these queues. And finally, section 2.5 describes the lower level functions used to manage the message queues.

2.1. Required Addressing Information

2.1.1. The ISODE Entities Database

The following is a brief overview of the use and structure of the isoentities database which is required for proper functioning of the ACSE interface. A more detailed description of the ISODE Entities Database can be found in The ISODE Entities Database, Chapter 7, Volume 1 of the ISODE manual. The isoentities file is normally stored in the /usr/local/etc directory.

To establish an association, a call is made to the *make_association()* function (defined in section 1.3.1 of this document), passing the following two parameters in the call:

(1) sd - An integer pointer (output parameter) which will contain the file descriptor for the association as assigned by the system, and

(2) host - A character string (input parameter) containing the name of the host with which the association is to be established.

The *make_association()* routine performs a lookup in the isoentities database in order to get:

(1) the calling application entity information which identifies the application process (and the application entity within that application process) that is initiating the association,

(2) the Presentation Service Access Pointer (PSAP) address of the association initiator,

(3) the called application entity information which identifies the intended responding application process (and the application entity within that application process), and

(4) the PSAP address of the responder.

The isoentities database stores the system addressing information needed to establish an association between two hosts. As an example, the following information must be available in the isoentities database for use in association establishment when one host, "mgmt", wants to establish an association with a host named "mgmt3". In order for the *make_association()* routine to be able to fetch the necessary information required to perform the ASCE A-ASSOCIATE request, the following entries must be put into the isoentities database:

```
mgmt    "network management"    1.17.4.0.13 \
                       #2/NS+47000400030003080020O1d82100

mgmt3   "network management"    1.17.4.0.13 \
                       #2/NS+470004000300030800200 6a1b200
```

Table 1 indicates the meanings for these information fields.

Information	Example
Hostname	mgmt
The service to be provided over the association	network management
Object Identifier definition of AET	1.17.4.0.13
TSEL used by osi.netd (Instructs tsapd to use SUNLINK OSI)	#2/
Presentation address of host expressed in string format	NS+470004000300030800200 1d82100
Authority and Format Identifier(AFI)	47
SNPA	0004
Subnet	00030003
SNPA	08002001d821
NSEL	00
Hostname	mgmt3
The service to be provided over the association	network management
Object Identifier definition of AET	1.17.4.0.13
TSEL used by osi.netd (Instructs tsapd to use SUNLINK OSI)	#2/
Presentation address of host expressed in string format	NS+4700040003000308002006a1b200
Authority and Format Identifier(AFI)	47
SNPA	0004
Subnet	00030003
SNPA	08002006a1b2
NSEL	00

Table 1

Before making the call to establish an association between host "mgmt" and host "mgmt3", the user:

(1) assigns the string "mgmt3" to the variable, host, and

(2) passes the address of the integer variable, sd.

When invoked, the *make_association()* routine uses the parameter, host, to perform a lookup of the called host in the isoentities database and retrieve the necessary information needed to complete the association request. The name of the system on which the initiating application is running (in this case, "mgmt") is used as the calling host name. The system name is discovered by the make_association() routine and, therefore, does not have to be passed as a parameter to the routine.

2.1.2. NSAP Addresses

To help clarify how addressing is represented, the following table shows the entries from the SUNLINK hosts file and the corresponding entry in the isoentities database. The corresponding fields of the NSAP address are outlined in the table. The SunLink OSI file /etc/sunlink/osi/hosts has an entry defining the service for localhost called CLIENT. Note that this entry is mandatory if you are running SunLink OSI release 5.2 or greater.

From the "/etc/sunlink/osi/hosts" file:

```
localhost    { [(osinet)3,3; (802.osinet)08:00:20:01:d8:21; 0] \
         [(tsel)0] [(ssel)"NULL"]                  \     /* Transport & Session selectors */
       } CLIENT                                          /* Service (uses FTAM CLIENT )   */
```

From the "/usr/local/etc/isoentities" file:

```
mgmt    "network management"    1.17.4.0.13 \
                               #2/NS+4700040003000308002001d82100
```

SUNLINK vs. ISODE NSAP Addressing							
	Initial Domain Part (IDP)		Domain Specific Part (DSP)				
Host	Authority and Format Identifier (AFI)	Initial Domain Identifier (IDI)	Organization ID	Subnet Number	Field Identifier	MAC Address	NSEL
SUNLINK							
mgmt	(osinet)		3	3	(802.osinet)	08:00:20:01:d8:21	0
ISODE							
mgmt	47	0004	0003	0003		08002001d821	00

Table 2

2.2. Message Data Structures

Two generic message types are used for all messages in this implementation; one for CMIP/S messages, the other for ACSE messages. The messages are stored in one of two queues, depending upon which of the two above mentioned types the received message is. The queues are implemented as linked lists of messages. To retrieve messages from these queues, the user need only check for the existence of the head of the list. Routines, to be described later, when invoked, automatically extract messages from the queues until each queue is empty. This scheme allows the implementor to develop a priority based system if desired (i.e., different priorities can be assigned to each of the two message queues). Both of the queues hold all information received with the message, so all information is available to the user.

CMIP/S message structure:

```
struct message_list_type {
  int association_id;           /* Association-descriptor this message was received on   */
  int invoke_id;                /* Invoke ID of this message                  */
  int link_id;                  /* Linked Id of this message                  */
  int nolinked;                 /* non-zero if no linked ID present           */
  int operation;                /* CMIP operation number                      */
  PE  args;                     /* Encoded User Data received with message (Presentation Element)*/
  char *message_pointer;           /* Pointer to the message structure        */
  int rose_operation;            /* Rose operation number                     */
  struct message_list_type *next;  /* Pointer to the next message             */
}
```

ACSE message structure:

```
struct acse_message_list_type {
  int operation;                /* ACSE operation number         */
  int association_id;           /* Association-descriptor this message was received on   */
  int start_indication_dis_type;  /* ACSE message type one of:          */
#define                         START      0
#define                         INDICATION 1
#define                         DISCONNECT 2
#define                         CONNECT    3
#define                         FINISH     4
#define                         ACSAP_RELEASE 5
  union acse_type {
    struct AcSAPstart *start;
    struct AcSAPindication *indication;
    struct TSAPdisconnect *disconnect;
    struct AcSAPconnect *connect;
    struct RoSAPindication *roi;
    struct AcSAPrelease *release_response;

  } *start_indication;          /* pointer to the message based on above type      */

 struct acse_message_list_type *next; /* Pointer to the next message              */
}
```

2.3. ACSE/ROSE interface library of functions

This section describes the functions that comprise the ACSE/ROSE interface. The ACSE/ROSE service interface for all ACSE/ROSE operations is composed of a library of functions called by the user, and modifiable routines called by those routines on behalf of the user. Please note that this interface is provided for the novice user not wishing to implement all options of ACSE/ROSE. Users wishing a more detailed implementation should consult the ISODE reference manuals directly.

Using this simplified ACSE interface, the following program segment demonstrates all calls a user must make to:

(1) initialize variables,

(2) establish an association,

(3) check for messages received,

(4) process messages received, and

(5) send messages.

```
main (argc, argv)
  int      argc;
  char    **argv;
{
  int sd;

  initialize (argc, argv);       /* (1) Must be called first */

  make_association("mgmt3",&sd)  /* (2) */

  for (;;)
  {
    check_iserver ();            /* (3) */
    check_messages ();            /* (4) */
    send_any_message (sd);         /* (5) */
  }
}
```

Each of these functions is described in detail in the following pages. Calls to the ACSE and ROSE routines are also described to assist users interested in modifying the code to achieve greater control of the ACSE and ROSE parameters. The function *check_iserver()* is a provided routine that 1) checks for activity on any association-descriptors, and 2) calls the designated routine (described later) to process the received message and add it to the appropriate queue. The function *check_messages()* is a user-modifiable routine which sends a response message or an error message to each request message received from the remote systems. The skeleton of this function is provided in section 2.6, demonstrating the different actions to be taken based on the type of message received. The function *send_any_message()*, a user-provided routine, sends request messages to remote systems by making calls to the *request()* routine described later. Prior to sending, the appropriate request message must have been filled in with valid information (described in the CMIS/P section). This routine must provide the means for sending all valid CMIS/P request messages. The implementor must, therefore: 1) make the appropriate calls to the fill functions described later, and 2) call the *request()* function with the appropriate message type and the pointer to the newly created request message.

The following tables (tables 3, 4 and 5) provide a quick reference for the programmer to determine which ACSE/ROSE service primitives are supported, which ISODE routines are associated with those primitives, and which NeMaSOS interface library routines are called to perform the services.

Association Control (ACSE)

NeMaSOS Function	ISODE function called	ACSE primitive represented	Description of work performed
make_association()	AcAsynAssocRequest()	A-ASSOCIATE.REQUEST	Create an association
associate_retry()	AcAsynRetryRequest()	A-ASSOCIATE.REQUEST	Check for asynchronous Request
accept_association()	AcAssocResponse()	A-ASSOCIATE.RESPONSE	Respond to an association request
release()	AcRelRequest()	A-RELEASE.REQUEST	Release an association
release_retry()	AcRelRetryRequest()	A-RELEASE.REQUEST	Check for asynchronous release request
release_response()	AcRelResponse()	A-RELEASE.RESPONSE	Respond to a release request
cmip_abort()	AcUAbortRequest()	A-ABORT.REQUEST	Abort an association

Table 3

Message Control (ROSE)

NeMaSOS Function	ISODE function called	ACSE primitive represented	Description of work performed
accept_message()	RyWait()	None	Receive an incoming message
request_message_received() * called by RyWait() *	None	RO-INVOKE.INDICATION	Process a received CMIP request message
error_message_received() * called by RyWait() *	None	RO-ERROR.INDICATION RO-U-REJECT.INDICATION RO-P-REJECT.INDICATION	Process a received CMIP error message Process a received CMIP user reject message Process a received CMIP provider reject message
result_message_received() * called by RyWait() *	None	RO-RESULT.INDICATION	Process a received CMIP result message
send_error()	RyDsError()	RO-ERROR.REQUEST	Send a CMIP error message
reject()	RyDsUReject()	RO-UREJECT.REQUEST	Reject an invocation
request()	RyStub()	RO-INVOKE.REQUEST	Send a CMIP Request message
response	RyDsResult()	RO-RESULT.REQUEST	Send a CMIP result message
add_operations()	RoSetService()	None	Uses Presentation as the underlying service
delete_operations()	RyDispatch()	None	Remove operations from an association descriptor
ros_init	AcInit()	None	Process an association request
ros_work	None	None	Process associate retry requests, release retry requests, or accept a request message.
ros_lose()	RyLose()	None	Clear all knowledge of an association from ISODE

Table 4

Utilities

NeMaSOS Function	ISODE function called	Description of work performed
add_acse_message()	None	Add an ACSE message to the ACSE message queue
extract_acse_message()	None	Extract an ACSE message from the ACSE message queue
add_cmip_message()	None	Add a CMIS/P message to the CMIP message queue
extract_cmip_message()	None	Extract a CMIS/P message from the CMIP message queue
check_iserver()	iserver_wait()	Listen for incoming messages
check_messages()	None	Check the queues for messages
clear_sd()	RyLose()	Clear an association descriptor
initialize()	iserver_init()	Initialize server
iserver_init()	TNetListen()	Initialize listening facility
iserver_wait()	TNetAccept()	Listen for incoming messages
map_operation()	None	Convert message type into array index
print_aci()	None	Print an ISODE AcSAPindication structure
print_roi()	None	Print an ISODE RoSAPindication structure
acse_free(id)	None	None
check_request_operation()	None	Verify that the request message to be sent has all required fields filled
check_response_operation()	None	Verify that the response message to be sent has all required fields filled

Table 5

Association Control

2.3.1. initialize

```
int initialize (argc, argv)
   int        argc;
   char     **argv;
```

Description The *initialize()* function is called to fill in the application entity title of the system on which the process is currently running. It looks up the address from the isoentities database so that future calls to *iserver_wait()* will listen for incoming messages. The function makes a call to iserver_init (see page 46, Volume 1 of the ISODE manuals) to perform the initialization. This function must be called before any message can be received.

Parameters

int argc Input argc to main routine (Number of input parameters).

char **argv Input argv to main routine (Vector of input parameters).

NOTE: See the READ_ME file in the directory /NEMASOS for a detailed explanation of parameter options.

Association Establishment

The following two routines deal with association establishment, allowing the user to create and accept association requests.

2.3.2. make_association

```
int  make_association( sd, host )

     int *sd;
     char *host;
```

Description The *make_association()* routine is used to establish an association between an initiator (manager or agent) and a responder (agent or manager). This routine checks the acceptability of the input parameters, and uses the address of the system upon which the application is currently running as the address of the initiator. The address of the host to be connected to is looked up in the isoentities database and the routine calls *AcAsynAssocRequest()* to establish the association (see page 33, Volume 1 of the ISODE manuals). If the call to *make_association()* is successful, a success indication (OK or CONNECTING_2) is returned and the sd (association id) parameter is assigned the value that references this association. Otherwise, NOTOK is returned.

Parameters

int *sd This parameter is assigned the association-descriptor that references the newly created association. Any time an operation is to be performed over this association, this association-descriptor should be used. (Output)

char *host This parameter is a character pointer to a string containing the host name of the intended responder of the association. This parameter is used to locate the called application entity information which identifies the intended responding application process (and the application entity within that application process). (Input)

Returns

OK The response message has been added to the ACSE message queue. The association is established, the sd parameter contains the association-descriptor.

NOTOK An error occurred, the association is not established, retry the call.

CONNECTING_2 The response to this asynchronous request was not received in the allotted time. The sd parameter contains the association-descriptor that will be used when the association is established. The response message completing the association establishment will be added to the ACSE message queue to notify the user when it is received.

DONE The association was rejected for one of the reasons listed in table 8. An ACSE message is added to the ACSE message queue with an AcSAPindication structure containing this information.

2.3.3. accept_association

```
accept_association (acs)
   struct AcSAPstart *acs;
```

Description The *accept_association()* function is used to send an association response over a designated association. After receiving an association request, the data field of the ACSE message contains the AcSAPstart structure containing the association information. This data can be examined and modified by the user to set any association parameters desired. This function uses the data in this structure when responding. The function calls *AcAssocResponse()* (see page 28, Volume 1 of the ISODE manual) to respond to the association request, and will (by default) accept the request.

Parameters

struct AcSAPstart *acs A pointer to the AcSAPstart structure containing the association information. This structure is stored in the received ACSE request message. (Input)

Returns

OK The association was accepted and the association now exists.

NOTOK An error occurred while sending the message. The error message is also printed to the output window of the screen interface.

Association Termination

The following three routines deal with association termination, allowing the user to release and abort associations.

2.3.4. release

```
int release ( sd )
  int     sd;
```

Description The *release()* function is used to send a release request over a designated association-descriptor. The function *AcRelRequest()* (see page 35, Volume 1 of the ISODE manual) is called to send the release request.

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

Returns

OK The release request was sent and responded to. An ACSE message has been added to the ACSE message queue. The message added to the queue will either be a RELEASE_REJECT (the release request was rejected) or a RELEASE_RESPONSE (the release was accepted and the association terminated).

NOTOK An error occurred while sending the message. An ACSE_ABORT message is added to the ACSE message queue containing the reason for the error.

2.3.5. release_response

```
int release_response (sd)
   int        sd;
```

Description The *release_response()* function is used to send a release response over a designated association. This function calls *AcRelResponse()* (see page 38, Volume 1 of the ISODE manual) to send the release response message.

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

Returns

OK The release response was sent and the association is terminated.

NOTOK An error occurred while sending the message. The error message is also printed to the output window of the screen interface.

2.3.6. cmip_abort

```
int cmip_abort (sd)
   int        sd;
```

Description The *cmip_abort()* function is used to send an abort request over a designated association. This function calls *AcUAbortRequest()* (see page 39, Volume 1 of the ISODE manual) to send the abort message.

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

Message Queue Management

The following four routines deal with queue management, more specifically with inserting and extracting messages from both the ACSE and CMIP message queues.

2.3.7. check_iserver

int check_iserver ()

Description The *check_iserver()* function is called with no parameters. It makes a call to *iserver_wait()* (see page 47, Volume 1 of the ISODE manual) which: 1) checks for activity on any association-descriptors, and 2) calls the appropriate routine (described later) to process the received message and add it to the appropriate queue. This function should be called as often as possible to receive messages.

Returns NONE

2.3.8. check_messages

int check_messages ()

Description The *check_messages()* function is a skeleton routine that the user can employ to extract messages from both the CMIP message queue and the ACSE message queue in any order the user desires. This routine requires modification by the user to fill in the necessary code to process the messages received. The messages should be processed by the user code according to the message type and the appropriate responses should be taken. The following code segment is a sample for extracting the messages:

```
if ((result = extract_acse_message ( &association_id, &operation, &type, &acse_ptr )) != NULL)
{
  switch (operation)
  {
  case ACSE_RELEASE:
  /* ACSE Release request  occurred, data field contains a (struct AcSAPindication *) data;
     Process message....
     break;
  case ACSE_ERROR:
  /* ACSE error occurred, data field contains a (struct TSAPdisconnect *) data; */
     Process message....
     break;
  case ACSE_ABORT:
  /* ACSE ABORT occurred, data field contains a (struct AcSAPindication *) data; */
     Process message....
     break;
  case RELEASE_REJECT:
  /* ACSE RELEASE REJECT  occurred, data field contains a (struct AcSAPrelease *) data; */
     Process message....
     break;
  case ASSOCIATE_INDICATION:
  /* ACSE ASSOCIATE  REQUEST  occurred, data field contains a (struct AcSAPstart *) data; */
     Process message....
     break;
  case ASSOCIATE_RESPONSE:
  /* ACSE ASSOCIATE  RESPONSE  occurred, data field contains a (struct AcSAPconnect *) data; */
     Process message....
     break;

  case RELEASE_RESPONSE:
  /* ACSE RELEASE  RESPONSE  occurred, data field contains a (struct AcSAPrelease *) data; */
     Process message....
     break;

  case RELEASE_FINISH:
  case RELEASE_END:
  /* ACSE RELEASE  RESPONSE (old style)  occurred, data field contains a (struct RoSAPindication
     Process message....
     break;
}
```

```
if ((result = extract_cmip_message(&association_id, &invoke_id, &link_id,
                                   &nolinked, &mode, &rose_op, &cmip_op, &msg_ptr )) != NULL)
{
  case ROI_INVOKE:
    /* RO-INVOKE REQUEST occurred, use cmip_op to process type  */
     Process message....
     break;
  case ROI_RESULT:
    /* RO-RESULT occurred, use cmip_op to process type  */
     Process message....
     break;
  case ROI_ERROR:
    /* RO-ERROR occurred, use cmip_op and rose_op to process type  */
     Process message....
     break;
  case ROI_UREJECT:
    /* RO-UREJECT occurred, use cmip_op and rose_op to process type  */
     Process message....
     break;
  case ROI_PREJECT:
    /* RO-PREJECT occurred, use cmip_op and rose_op to process type  */
     Process message....
     break;
}
```

Returns As modified by the user.

2.3.9. extract_acse_message

```
int extract_acse_message ( association_id, operation, type, acse_ptr )
   int      *association_id;
   int      *operation;
   int      *type;
   struct acse_type **acse_ptr;
```

Description The *extract_acse_message()* function is used to retrieve ACSE messages from the ACSE message queue. This function is called from within the check_messages() routine, or an equivalent user routine, to check for messages in the ACSE message queue. These messages are stored in the ACSE message queue in the structure referenced in section 2.2. If this function returns "NULL", the message list is either empty (on the first call), or exhausted (on subsequent calls).

Parameters

int *association_id Association-descriptor identifying the association on which the message was received. (Output)

int *operation The ACSE operation type of the message received. (Output)
One of: ACSE_RELEASE, ACSE_ERROR, ACSE_ABORT, RELEASE_REJECT, ASSOCIATE_INDICATION, ASSOCIATE_RESPONSE, RELEASE_RESPONSE, RELEASE_FINISH, RELEASE_END.

int *type The type of the C structure for the message contained in the acse_ptr union. (Output)
One of: ACSAP_START, ACSAP_INDICATION, ACI_FINISH, ACI_ABORT ACSAP_DISCONNECT, ACSAP_CONNECT, ACSAP_FINISH, ACSAP_RELEASE.

struct acse_type **acse_ptr A pointer to the structure defined by type.(Output)

Returns

SUCCESS A message existed in the queue and information was extracted from it and stored in the parameters.

NULL The message list is empty.

2.3.10. extract_cmip_message

```
int extract_cmip_message ( association_id, invoke_id, link_id, nolinked, mode,
                           rose_op, cmip_op, msg_ptr )

  int     *association_id;
  int     *invoke_id;
  int     *link_id;
  int     *nolinked;
  int     *mode;
  int     *rose_op;
  int     *cmip_op;
  char    **msg_ptr;
```

Description The *extract_cmip_message()* function is used to retrieve CMIP messages from the CMIP message queue. This function is called from within the check_messages() routine, or an equivalent user routine, to check for messages in the CMIP message queue. Messages are stored in the CMIP message queue in the structure referenced in section 2.2. If the function returns "NULL", the message list is either empty (on the first call), or exhausted (on subsequent calls).

Parameters

int *association_id Association-descriptor identifying the association on which the message was received. (Output)

int *invoke_id The invoke ID of the message. (Output)

int *link_id The linked Id of the message. (Output)

int *nolinked Present if no linked ID exists. (Output)

int *mode The mode of the CMIP operation (Confirmed or Unconfirmed). (Output)

int *rose_op A number representing the ROSE operation type. (Output)

int *cmip_op A number representing the CMIP operation type. (Output)

char **msg_ptr A pointer to the CMIP message. (Output)

Returns

SUCCESS A message existed in the queue and information was extracted from it and stored in the parameters.

NULL The message list is empty.

NO_SUCH_RO_OP The ROSE operation to be stored in the parameter "rose_op" is not one of the known ROSE operations.

NO_SUCH_MSG_TYPE The CMIP message to be stored in the parameter "cmip_op" is not one of the known CMIP messages.

Sending CMIP/ROSE Messages

The following four routines allow the user to send requests, responses, errors and reject invocations.

2.3.11. request

```
int request (sd, msg_type, mode, in)
   int      sd;
   int      msg_type;
   int      mode;
   caddr_t   in;
```

Description The *request()* function sends a CMIP request message over the association designated by the association-descriptor, sd. The "out" parameter is a pointer to the request message which should have been previously allocated and filled in via calls to the appropriate parameter fill routines. A call to *RyStub()* (see page 97, Volume 4 of the ISODE manual) is made to send the request message.

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

int msg_type The integer representation of the CMIP request message type (see table 12) (Input)

int *mode The mode of the message (Confirmed(1) or Unconfirmed(0)). (Input)

caddr_t in A pointer to the C structure containing the CMIP operation's argument. (Input)

Returns

OK The request message was sent.

NOTOK An error occurred while sending the message. The error message is also printed to the output window of the screen interface.

REQUEST_INCOMPLETE One of the required fields was not filled in for this request message.

2.3.12. response

```
int response (sd, id, msg_type, out, priority, linked)
   int       sd;
   int       id;
   int       msg_type;
   caddr_t   out;
   int       priority;
   int       linked;
```

Description The *response()* function sends a CMIP response message over the association designated by the association-descriptor, sd. The "out" parameter is a pointer to the response message which should have been previously allocated and filled in via calls to the appropriate parameter fill routines. A call to *RyDsResult()* (see page 102, Volume 4 of the ISODE manual) is made to send the message. The linked parameter specifies if this response is to be sent as part of a linked reply (set to 1), or if it is to be sent as a single result (set to 0).

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

int id ID of the ROS operation invocation being responded to. (Input)

msg_type CMIS operation type

caddr_t out A pointer to the C structure containing the operation's result. (Input)

int priority The priority of the response (use ROS_NOPRIO, if undetermined). (Input)

int linked Set to 1 if this response is to be sent as a linked-reply, set to 0 if this is a single response. (Input)

Returns

OK The response message was sent.

NOTOK An error occurred while sending the message. The error message is also printed to the output window of the screen interface.

RESPONSE_INCOMPLETE A required field was not filled in for this response message.

2.3.13. send_error

```
int send_error ( sd, id, err, out, priority, linked )
  int      sd;
  int      id;
  int      err;
  caddr_t  out;
  int      priority;
  int      linked;
```

Description The *send_error()* function sends a CMIP error message over a designated association as indicated by the association-descriptor, sd. The "out" parameter is a pointer to the error message which should have been previously allocated and filled in via calls to the appropriate parameter fill routines. A call is made to *RyDsError()* (see page 102, Volume 4 of the ISODE manual) to send the message. The linked parameter specifies if this response is to be sent as part of a linked reply (set to 1), or if it is to be sent as a single result (set to 0).

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

int id ID of the ROS operation invocation being responded to. (Input)

int err The integer representation of the error code being returned. (Input)

caddr_t out A pointer to the C structure containing the error parameter, if any. Since some errors do not have any paramenters, this can be a NULL pointer. (Input)

int priority The priority of the response (use ROS_NOPRIO, if undetermined). (Input)

int linked Set to 1 if this response is to be sent as a linked-reply, set to 0 if this is a single response. (Input)

Returns

OK The error message was sent.

NOTOK An error occurred while sending the message. The error message is printed to the output window of the screen interface.

2.3.14. reject

```
int      reject ( sd, id, reason, priority )
   int      sd;
   int      id;
   int   reason;
   int priority;
```

Description The *reject()* function is used to reject an invocation. The input parameter, reason, is filled with one of the reasons listed in table 9. A call to *RyDsReject()* (see page 103, Volume 4 of the ISODE reference manual) is made to send the rejection.

Parameters

int sd Association-descriptor designating the particular association over which the message is to be sent. (Input)

int id ID of the ROS operation request invocation being rejected. (Input)

int reason The reason for the rejection (see table 9). (Input)

int priority The priority of the response (use ROS_NOPRIO, if undetermined). (Input)

Returns

OK The reject message was sent.

NOTOK An error occurred while sending the message. The error message is printed to the output window of the screen interface.

2.4. Utilities

The following utility routines are included to assist in the processing of messages.

2.4.1. print_aci

```
print_aci (aci, additional_message, sd)
   struct AcSAPindication *aci;
   char       additional_message[];
   int        sd;
```

Description The *print_aci()* function prints an AcSAPindication structure to stdout, given the pointer to the structure. The function uses internal information to determine the type of indication the message contains (see table 6) and prints the appropriate information. The additional message parameter allows the user to print additional information, if necessary. Such additional information is optional.

Parameters

struct AcSAPindication *aci Pointer to the AcSAPindication structure that is to be printed. The table below identifies the different types of indications and the reasons contained in these types. (Input)

char additional_message[] Array of additional character data to be printed along with structure information. (Input)

int sd Association-descriptor identifying the association on which the message was received. (Input)

Returns NONE

AcSAPindication structure	
Type	**Reason**
ACI_FINISH	ACF_URGENT ACF_USERDEFINED
ACI_ABORT	ACA_USER ACA_PROVIDER ACA_LOCAL ACS_ACCEPT ACS_REJECT ACS_PERMANENT ACS_TRANSIENT

Table 6

2.4.2. clear_sd

```
int      clear_sd (sd)
  int      sd;
```

Description The *clear_sd()* function is used to clear a file descriptor when an error occurs on an association. This function is automatically called for associations established by this implementation. However, the function is included here for users who wish to create their own associations. Use of this function prevents the function *TNetAccept()* from listening for activity on a file descriptor designating an association that has been aborted due to the occurrence of a fatal error.

Parameters

int sd Association-descriptor designating the particular association to be cleared. (Input)

Returns NONE

2.4.3. acse_free

```
int acse_free (id)
   struct acse_type *id;
```

Description The *acse_free()* function frees an ACSAP structure given the pointer to that structure. This function uses internal information to determine the structure type (see table 7) and then frees all memory associated with that structure.

Parameters

struct acse_type *id Pointer to the AcSAP structure to be freed. This is the data field of an ACSE message. (Input)

Returns NONE

Internal ACSE Message Types	
Type	Structure Released
ACSAP_START	AcSAPstart
ACSAP_INDICATION	AcSAPindication
ACI_FINISH	AcSAPfinish
ACI_ABORT	AcSAPabort
ACSAP_DISCONNECT	TSAPdisconnect
ACSAP_CONNECT	AcSAPconnect
ACSAP_FINISH	RoSAPindication
ACSAP_RELEASE	AcSAPrelease

Table 7

2.4.4. map_operation

```
map_operation (rose_operation, cmip_operation)
   int        rose_operation;
   int        cmip_operation;
```

Description The *map_operation()* function provides access to a table which stores local identifiers for both the range of CMIP request/response operation types, and the range of CMIP error types. Since this single table contains the integer representation for the errors and the request/response types, and since these values overlap, this function is needed to do the appropriate table lookup and provide the mapping function which returns the correct message type (i.e., index into the array) (see table 12).

Parameters

int rose_operation The ROSE operation number - listed in table 11 (Input).

int cmip_operation The CMIP operation number - listed in table 11 (Input).

Returns The index element of the array (see table 12) representing the CMIP message to be processed.

2.5. Routines

The following routines are called by the functions described in the previous section. Although the following functions are not called directly by the user of the sample implementation provided with NeMaSOS, the descriptions of these functions are provided here to assist those implementors who may desire direct access to these functions or who may need to modify these functions to suit their application.

Association Establishment

The following two routines deal with association establishment.

2.5.1. associate_retry

```
int associate_retry (sd)
  int     sd;
```

Description The *associate_retry()* function is called by ros_work to process an association retry request message. An association retry will occur when a previously attempted association request is answered. The function *AcAsynRetryRequest()* (see page 34, Volume 1 of the ISODE manuals) is automatically called and the result is added to the ACSE message queue.

Parameters

int sd Association-descriptor designating the particular association on which the activity occurred. (Input)

Returns

OK The association is established. The sd parameter contains the association-descriptor.

NOTOK An error occurred. The association is not established. Retry the call.

CONNECTING_2 The response was not received in the time allotted for this asynchronous request. The sd parameter is updated with the association-descriptor that will be used, and a future response message will be added to the ACSE message queue to notify the user when the association is established.

DONE The association was rejected for one of the reasons listed in table 8. An ACSE message will be added to the ACSE message queue with an AcSAPindication structure containing this information.

2.5.2. add_operations

```
static void add_operations (sd)
   int       sd;
```

Description The *add_operations()* function is used in conjunction with the *RyWait()* routine (see page 104, Volume 4 of the ISODE manual) to designate the operations that are allowed to be performed over an association (you can only receive operations you know about). Operations allowed include all CMIS operations (i.e., GET, SET, DELETE...).

Parameters

int sd Association-descriptor of the association to which operations are to be added. (Input)

Returns NONE

Association Termination

The following three routines deal with association termination.

2.5.3. release_retry

```
int       release_retry (sd)
   int       sd;
```

Description The *release_retry()* function is called by ros_work to process a release retry request message. A release retry will occur when a previously attempted release request is answered. The function *AcRelRetryRequest()* (see page 37, Volume 1 of the ISODE manuals) is automatically called and the result is added to the ACSE message queue.

Parameters

int sd Association-descriptor designating the particular association on which the activity occurred. (Input)

Returns

OK The message was received and the association terminated.

2.5.4. delete_operations

```
static void delete_operations (sd)
   int        sd;
```

Description The *delete_operations()* function is used in conjunction with the *RyDispatch()* routine (see page 100, Volume 4 of the ISODE manual). This function removes operations allowed to be performed over an association (you can only receive operations you know about). Operations allowed include all CMIS operations (i.e., GET, SET, DELETE...).

Parameters

int sd Association-descriptor designating the particular association from which to delete the operations. (Input)

2.5.5. rose_lose

```
static int ros_lose (td)
   struct TSAPdisconnect *td;
```

Description The *rose_lose()* function is called by the initiator of an association (e.g., the manager) when an association is terminated abnormally by the responder on the association (e.g., the agent). It clears all knowledge of the association from memory on the initiator's system.

Parameters

struct TSAPdisconnect *td The TSAPdisconnect structure returned from TNetAccept routine. (Input)

Returns NONE

Message Queue Management

The following four routines deal with queue management for both the ACSE and CMIP message queues.

2.5.6. add_acse_message

```
static void add_acse_message (association_id, operation, data)
   int      *association_id;
   int       operation;
   char     *data;
```

Description The *add_acse_message()* function adds an ACSE message to the ACSE message queue. Table 10 lists the possible operation types, the associated C defined constant designators, and the ISODE structures for storing the operation information. This function is called by numerous routines any time an ACSE message is received.

Parameters

int *association_id Association-descriptor identifying the association on which the message was received. (Input)

int operation The numeric value of the ACSE operation (see table 10). (Input)

char *data A character pointer to the message received. (Actual viewing of the message requires type-casting to the appropriate specific message type.) (Input)

Returns NONE

2.5.7. add_cmip_message

```
static void add_cmip_message (sd, id, value, rose_operation, cmis_operation)
   int       sd;
   int       id;
   caddr_t   value;
   int       rose_operation;
   int       cmis_operation;
```

Description The *add_cmip_message()* function adds a message to the CMIP message queue. The association-descriptor the message was received on, the invoke Id, and a character pointer to the message are stored. Table 11 lists the various CMIS operations and ROSE operations that are stored.

Parameters

int sd Association-descriptor identifying the association on which the message was received. (Input)

int id The invoke ID of the message. (Input)

caddr_t value A pointer to the CMIP message to be added to the CMIP message queue and stored. (Input)

int rose_operation A number representing the ROSE operation type of the CMIP message being added to the CMIP message queue. (Input)

int cmis_operation A number representing the CMIS operation type of the CMIP message being added to the CMIP message queue. (Input)

Returns NONE

2.5.8. ros_init

```
int ros_init (vecp, vec)
  int      vecp;
  char   **vec;
```

Description The *ros_init()* function is used in conjunction with the *iserver_wait()* function, (see page 47, Volume 1 of the ISODE manual). It is called any time new activity occurs on a file descriptor, signifying a new association request. The parameter *vec* contains the association data and *vecp* contains the length of vec. The function processes the data and adds an ACSE message to the ACSE queue for the association request. This function is registered with iserver_wait by using the function *iserver_init()* (see page 46 , Volume 1 of the ISODE manuals).

Parameters

int vecp The length of the initialization vector, vec. (Input)

char **vec The initialization vector containing the association information. (Input)

Returns

NOTOK An error occurred processing the message. An ACSE message was added to the ACSE message queue containing the error.

2.5.9. ros_work

```
int ros_work (sd)
  int      sd;
```

Description The *ros_work()* function is used in conjunction with the *iserver_wait()* function, (see page 47, Volume 1 of the ISODE manual). It is called any time activity occurs on a file descriptor, signifying a request/response on an association-descriptor. The parameter *sd* contains the association-descriptor of the association on which the activity occurred. This function determines whether the request is 1) an association retry request, 2) a release retry request, or 3) a CMIP message. It then calls the appropriate routine to process the message.

Parameters

int sd Association-descriptor designating the particular association on which the activity occurred. (Input)

Returns

OK The message was received and processed.

Receiving CMIP/ROSE Messages

The following four routines allow the user to receive requests, responses, errors and reject invocations.

2.5.10. request_message_received

```
int request_message_received (sd, ryo, rox, in, roi)
   int       sd;
   struct RyOperation *ryo;
   struct RoSAPinvoke *rox;
   caddr_t   in;
   struct RoSAPindication *roi;
```

Description The *request_message_received()* function is automatically called by *RyWait()* (see page 104, Volume 4 of the ISODE manuals), it signals the receipt of a CMIP request message and adds the message to the CMIP message queue. In order to be automatically called by *RyWait()* at the appropriate time, this function is registered by using the function *RyDispatch* (see page 100, Volume 4 of the ISODE manuals).

Parameters

int sd Association-descriptor identifying the association on which the message was received. (Input)

struct RyOperation *ryo The associated RyOperation structure. (see page 97, Volume 4 of the ISODE manual) (Input)

struct RoSAPinvoke *rox The associated RoSAPinvoke structure, containing the invoke ID and the operation type. (see page 97, Volume 4 of the ISODE manual) (Input)

caddr_t in A character pointer to the message received. (Input)

struct RoSAPindication *roi A pointer to the RoSAPindication structure. (see page 97, Volume 4 of the ISODE manual) (Input)

Returns

DONE Request message was received and enqueued.

2.5.11. error_message_received

```
int error_message_received (sd, id, reason, value, roi)
   int       sd;
   int       id;
   int       reason;
   caddr_t   value;
   struct RoSAPindication *roi;
```

Description The *error_message_received()* function is automatically called by *RyWait()* (see page 104, Volume 4 of the ISODE manuals). It signals the receipt of a CMIP error message and adds the message to the CMIP message queue. In order to be automatically called by *RyWait()* at the appropriate time, this function is registered by using the call *RyDispatch()* (see page 100 , Volume 4 of the ISODE manuals).

Parameters

int sd Association-descriptor identifying the association on which the message was received. (Input)

int id The invoke Id of the message. (Input)

int reason The reason for the error (see table 9). (Input)

caddr_t value A character pointer to the error message.

struct RoSAPindication *roi A pointer to a RoSAPindication structure that is updated only if the call fails (see page 58, Volume 1 of the ISODE reference manual). (Input)

Returns

DONE Error message was received and enqueued.

2.5.12. result_message_received

```
int result_message_received (sd, id, reason, value, roi)
   int       sd;
   int       id;
   int       reason;
   caddr_t   value;
   struct RoSAPindication *roi;
```

Description The *result_message_received()* function is automatically called by *RyWait()* (see page 104, Volume 4 of the ISODE manuals). It signals the receipt of a CMIP response message and adds the message to the CMIP message queue. In order to be automatically called by *RyWait()* at the appropriate time, this function is registered by using the function *RyDispatch()* (see page 100 , Volume 4 of the ISODE manuals).

Parameters

int sd Association-descriptor identifying the association on which the message was received. (Input)

int id The invoke Id of the message. (Input)

int reason Identifies the type of the result message received. Values are either RY_RESULT or RY_REJECT. (Input)

caddr_t value A character pointer to the result message.

struct RoSAPindication *roi A pointer to a RoSAPindication structure that is updated only if the call fails (see page 58, Volume 1 of the ISODE reference manual). (Input)

Returns

DONE Result message was received and enqueued.

2.5.13. accept_message

```
int       accept_message (sd)
   int       sd;
```

Description The *accept_message()* function is called by ros_work to process a CMIP message and add it to the CMIP message queue. This function calls *RyWait()* (see page 104, Volume 4 of the ISODE manuals) to receive the message.

Parameters

int sd Association-descriptor designating the particular association on which the activity occurred. (Input)

Returns

OK The message was received and processed.

2.6. Tables

Association rejection Reasons		
Error	**Return Value**	**Meaning**
Provider-Initiated Aborts (FATAL)	ACS_ADDRESS	Address unknown
	ACS_REFUSED	Connect request refused on this network connection
	ACS_CONGEST	Local limit exceeded
	ACS_PRESENTATION	Presentation disconnect
	ACS_PROTOCOL	Protocol error
	ACS_RESPONDING	Rejected by responding ACPM
	ACS_ABORT	Peer aborted association
User-Initiated Rejections (FATAL)	ACS_PERMANENT	Permanent
	ACS_TRANSIENT	Transient
Interface Errors (NON-FATAL)	ACS_REJECT	Release rejected
	ACS_PARAMETER	Invalid parameter
	ACS_OPERATION	Invalid operation

Table 8

Message rejection Reasons

Error	Return Value	Meaning
Provider-Initiated Aborts (FATAL)	ROS_ADDRESS	Address unknown
	ROS_REFUSED	Connect request refused on this network connection
	ROS_SESSION	Session Disconnect
	ROS_PRESENTATION	Presentation disconnect
	ROS_PROTOCOL	Protocol error
	ROS_CONGEST	Congestion at RoSAP
	ROS_REMOTE	Remote system problem
	ROS_DONE	Association done via async handler
	ROS_ABORTED	Peer aborted association
	ROS_RTS	RTS disconnect
	ROS_ACS	ACS disconnect
User-Initiated Rejections (FATAL)	ROS_VALIDATE	Authentication failure
	ROS_BUSY	Busy
Provider-Initiated Rejects (NON-FATAL)	ROS_GP_UNRECOG	Unrecognized APDU
	ROS_GP_MISTYPED	Mistyped APDU
	ROS_GP_STRUCT	Badly structured APDU

Table 9

ACSE Message Types

Operation	# define	ISODE structure
ASSOCIATE_INDICATION	ACSAP_START	(struct AcSAPstart *) data
ACSE_RELEASE	ACSAP_INDICATION	(struct AcSAPindication *) data
ACSE_ABORT	ACSAP_INDICATION	(struct AcSAPindication *) data
RELEASE_END	ACSAP_FINISH	(struct RoSAPindication *) data
RELEASE_FINISH	ACSAP_FINISH	(struct RoSAPindication *) data
ACSE_ERROR	ACSAP_DISCONNECT	(struct TSAPdisconnect *) data
ASSOCIATE_RESPONSE	ACSAP_CONNECT	(struct AcSAPconnect *) data
RELEASE_REJECT	ACSAP_RELEASE	(struct AcSAPrelease *) data
RELEASE_RESPONSE	ACSAP_RELEASE	(struct AcSAPrelease *) data;

Table 10

CMIP and ROSE Operation Types

Rose Operation	CMIP Operation Name (assigned by ISODE)	Integer Rep
ROI_INVOKE ROI_RESULT	operation_CMIP_m__EventReport	0
	operation_CMIP_m__EventReport__Confirmed	1
	operation_CMIP_m__Linked__Reply	2
	operation_CMIP_m__Get	3
	operation_CMIP_m__Set	4
	operation_CMIP_m__Set__Confirmed	5
	operation_CMIP_m__Action	6
	operation_CMIP_m__Action__Confirmed	7
	operation_CMIP_m__Create	8
	operation_CMIP_m__Delete	9
	operation_CMIP_m__CancelGet	10
ROI_ERROR	error_CMIP_noSuchObjectClass	0
	error_CMIP_noSuchObjectInstance	1
	error_CMIP_accessDenied	2
	error_CMIP_syncNotSupported	3
	error_CMIP_invalidFilter	4
	error_CMIP_noSuchAttribute	5
	error_CMIP_invalidAttributeValue	6
	error_CMIP_getListError	7
	error_CMIP_setListError	8
	error_CMIP_noSuchAction	9
	error_CMIP_processingFailure	10
	error_CMIP_duplicateManagedObjectInstance	11
	error_CMIP_noSuchReferenceObject	12
	error_CMIP_noSuchEventType	13
	error_CMIP_noSuchArgument	14
	error_CMIP_invalidArgumentValue	15
	error_CMIP_invalidScope	16
	error_CMIP_invalidObjectInstance	17
	error_CMIP_missingAttributeValue	18
	error_CMIP_classInstanceConflict	19
	error_CMIP_complexityLimitation	20
	error_CMIP_mistypedOperation	21
	error_CMIP_noSuchInvokeId	22
	error_CMIP_operationCancelled	23
ROI_PREJECT	None	None
ROI_UREJECT	None	None
ROI_FINISH	None	None

Table 11

CMIP Message Types (struct fill_table table_CMIP_fills[])

Array Element	Message Type	ISODE value	CMIP value
0	NO_SUCH_OBJECT_CLASS	error_CMIP_noSuchObjectClass	0
1	NO_SUCH_OBJECT_INSTANCE	error_CMIP_noSuchObjectInstance	1
8	ACCESSDENIED	error_CMIP_accessDenied	2
4	SYNC_NOT_SUPPORTED	error_CMIP_syncNotSupported	3
4	INVALID_FILTER	error_CMIP_invalidFilter	4
1	NO_SUCH_ATTRIBUTE	error_CMIP_noSuchAttribute	5
4	INVALID_ATTRIBUTE_VALUE	error_CMIP_invalidAttributeValue	8
9	GET_LIST_ERROR	error_CMIP_getListError	8
8	SET_LIST_ERROR	error_CMIP_setListError	8
9	NO_SUCH_ACTION	error_CMIP_noSuchAction	8
10	PROCESSING_FAILURE	error_CMIP_processingFailure	10
11	DUPLICATE_MANAGED OBJECT_INSTANCE	error_CMIP_duplicateManaged ObjectInstance	11
12	NO_SUCH_REFERENCE_OBJECT	error_CMIP_noSuchReferenceObject	12
10	NO_SUCH_EVENT_TYPE	error_CMIP_noSuchEventType	13
10	NO_SUCH_ARGUMENT	error_CMIP_noSuchArgument	14
19	INVALID_ARGUMENT_VALUE	error_CMIP_invalidArgumentValue	15
10	INVALID_SCOPE	error_CMIP_invalidScope	18
10	INVALID_OBJECT_INSTANCE	error_CMIP_invalidObjectInstance	17
19	MISSING_ATTRIBUTE_VALUE	error_CMIP_missingAttributeValue	18
19	CLASS_INSTANCE_CONFLICT	error_CMIP_classInstanceConflict	19
10	COMPLEXITY_LIMITATION	error_CMIP_complexityLimitation	21
21	MISTYPED_OPERATION	error_CMIP_mistypedOperation	21
22	NO_SUCH_INVOKEID	error_CMIP_noSuchInvokeId	22
23	OPERATION_CANCELLED	error_CMIP_operationCancelled	23

CMIP Message Types (struct fill_table table_CMIP_fills[]) (Continued)			
Array Element	Message Type	ISODE value	CMIP value
24 24 26 27	SET_REQ SET_IND SET_RSP SET_CNF	operation_CMIP_m__Set or operation_CMIP_m__Set__Confirmed	4 5
28 29 30 31	GET_REQ GET_IND GET_RSP GET_CNF	operation_CMIP_m__Get	3
32 33 34 35	EVENT_REQ EVENT_IND EVENT_RSP EVENT_CNF	operation_CMIP_m__EventReport or operation_CMIP_m__EventReport__Confirmed	0 1
36 37 38 39	ACTION_REQ ACTION_IND ACTION_RSP ACTION_CNF	operation_CMIP_m__Action or operation_CMIP_m__Action__Confirmed	6 7
40 41 42 43	CREATE_REQ CREATE_IND CREATE_RSP CREATE_CNF	operation_CMIP_m__Create	9
44 45 46 47	DELETE_REQ DELETE_IND DELETE_RSP DELETE_CNF	operation_CMIP_m__Delete	9
48 49	CANCEL_REQ CANCEL_IND	operation_CMIP_m__CancelGet	10
50 51	ACTION_ERR DELETE_ERR	None None	N/A N/A
52	LINKED_REPLY	operation_CMIP_m__Linked__Reply	2

Table 12

3. CMIS Operations

3.1. Introduction

The CMIS service interface is comprised of a set of library functions residing in cmislib.a. These functions, forming the CMIS service interface, provide the means for the user, when sending PDUs, to allocate and initialize data structures representing CMIP operation PDUs, to appropriately fill in the parameter fields of these structures (using the "fill" functions), and to encode and send out the PDUs to the peer management entity. When receiving CMIP PDUs, these functions enable the user to retrieve incoming PDUs, recognize the operation type of the PDU, decode the PDUs, and extract the CMIS parameter information from the PDU (using the "extract" functions). The CMIS services supported by these library functions include: M-GET, M-SET, M-ACTION, M-EVENT-REPORT, M-CREATE, M-DELETE, and CMIS errors.

These library functions are organized into what could loosely be considered to be three basic types of functions. The first category of library functions are used when sending request and response PDUs and include the init_operation_struct(), request(), or the response() functions. The init function is intended to be the first function called when initiating a CMIS service because it allocates and initializes the basic data structure for the service message. The request() and response() functions are intended to be the last functions called when a request or response is to be sent. The request() and response() function, after checking that critical data structure fields are non-null, initiates the encoding and sending of the message (PDU).

The second and third categories of functions are the mandatory and optional function calls, respectively, used to fill in parameter information. For each of the CMIS request and response services, the list of functions are marked as mandatory or optional. A mandatory function is one that should be called because for that CMIS service the CMIS standard mandates that that parameter shall be provided. If a mandatory function is not called, or if the user tries to fill this parameter by other means, the encoder may fail when attempting to encode this parameter. An optional function, on the other hand, is one that deals with a parameter that the CMIS standard does not mandate, but rather makes optional. The user may call optional functions to fill in optional parameters when it is desired to pass information in these parameters. Unlike the mandatory case, it is acceptable to not invoke the optional functions and thus leave those corresponding message fields null or as otherwise initialized by the init_operation_struct() function.

When operating in the receive mode, to process CMIS indications and confirms, the sequence of function calls is somewhat different from that used to send PDUs. First, the user invokes the extract_cmip_message() function to retrieve the message from the CMIP message queue and determine the CMIS message type. Then, the appropriate "extract" functions are called, according to the message type, to retrieve any desired parameter information from the message. And finally, the free_operation_struct() function is called to delete the message structure when it is no longer needed.

Also included in the CMIS library of functions are routines that enable the CMIS user to fill and extract information when it it necessary to send or receive a CMIS error.

For each CMIP operation and CMIS service primitive supporting that operation, tables 13-17 list the interface functions used to fill or extract information from the relevant parameters. These tables are organized to show the relevant parameters for each operation primitive and the appropriate fill and extract interface function to use in processing those parameters. The tables do not include the repetitive statement of the common functions mentioned above which are to be used for all primitives to initialize and send the PDUs or to determine the type of received CMIS message and then to delete the message when no longer needed.

3.1.1. Object Identifier (OID)

An **object identifier** is a sequence of non-negative integer values that represent a path in a tree. The tree consists of a root connected to a number of labeled nodes via edges. Each label consists of a non-negative integer value and possibly a brief textual description. Each node may, in turn, have child nodes of its own, termed subordinates, which are also labeled. This process may be repeated to an arbitrary depth.

For all functions in the CMIS/CMIP service interface library, when an **object identifier** is to be passed as a parameter, a character string is used to represent the object identifier. The character string should contain the integer values which specify the path through the tree, starting at the root and proceeding to the object in question. The integer values are separated by a period (dot).

3.1.2. Presentation Element (PE)

A **presentation element** is a data structure which is used to represent data in a machine-independent form. The typedef PE is a pointer to a PElement structure. The structure contains several elements, most of which are uninteresting to the user of the NeMaSOS library. Please reference volume 1, page 124 of the ISODE manuals for a complete description of a presentation element.

There are several routines which can be used to translate between the machine-independent representation of the element and machine-specific objects such as integers, strings, and the like. It is extremely important that programs use these routines to perform the translation between objects. They have been carefully coded to present a simple, uniform interface between machine-specifics and the machine-independent encoding protocol, please reference volume 1, page 125 of the ISODE manuals for a list of available functions.

Most presentation elements used in the fill and extract functions should be created by the encode and decode routines provided by PEPY. The user need only call the encode functions passing a pointer to a presentation element (PE) and the structure to be encoded. Then the user calls the appropriate fill routine passing the pointer to the PE returned containing the encoded data. For the decoding functions, call the appropriate extract routine and then call the correct decode function with the returned PE.

M-GET Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-GET Request	baseManagedObjectClass baseManagedObjectInstance accessControl synchronization scope filter attribute identifier List	fill_baseManagedObjectClass fill_baseManagedObjectInstance fill_accessControl fill_synchronization fill_scope fill_filter fill_attributeIdlist	extract_baseManagedObjectClass extract_baseManagedObjectInstance extract_accessControl extract_synchronization extract_scope extract_filter extract_attributeIdlist
M-GET Result	managedObjectClass managedObjectInstance currentTime attributeList	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_attributeList	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_attributeList
M-GET Errors	accessDenied getListError noSuchObjectClass syncNotSupported	classInstanceConflict invalidFilter noSuchObjectInstance operationCancelled	complexityLimitation invalidScope processingFailure

M-SET Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-SET Request	baseManagedObjectClass baseManagedObjectInstance accessControl synchronization scope filter modification List	fill_baseManagedObjectClass fill_baseManagedObjectInstance fill_accessControl fill_synchronization fill_scope fill_filter fill_modificationList	extract_baseManagedObjectClass extract_baseManagedObjectInstance extract_accessControl extract_synchronization extract_scope extract_filter extract_modificationlist
M-SET Result	managedObjectClass managedObjectInstance currentTime attributeList	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_attributeList	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_attributeList
M-SET Errors	accessDenied invalidFilter noSuchObjectInstance syncNotSupported	classInstanceConflict invalidScope processingFailure	complexityLimitation noSuchObjectClass setListError

Table 13

M-ACTION Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-ACTION Request	baseManagedObjectClass baseManagedObjectInstance accessControl synchronization scope filter action Information	fill_baseManagedObjectClass fill_baseManagedObjectInstance fill_accessControl fill_synchronization fill_scope fill_filter fill_actionInfo	extract_baseManagedObjectClass extract_baseManagedObjectInstance extract_accessControl extract_synchronization extract_scope extract_filter extract_actionInfo
M-ACTION Result	managedObjectClass managedObjectInstance currentTime actionReply	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_actionReply	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_actionReply
M-ACTION Errors	accessDenied invalidScope noSuchAction noSuchObjectInstance	classInstanceConflict invalidArgumentValue noSuchArgument processingFailure	complexityLimitation invalidFilter noSuchObjectClass syncNotSupported

M-EVENT-REPORT Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-EVENT-REPORT Request	managedObjectClass managedObjectInstance eventTime eventType eventInfo	fill_managedObjectClass fill_managedObjectInstance fill_eventTime fill_eventType fill_eventInfo	extract_managedObjectClass extract_managedObjectInstance extract_eventTime extract_eventType extract_eventInfo
M-EVENT-REPORT Result	managedObjectClass managedObjectInstance currentTime eventReply	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_eventReply	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_eventReply
M-EVENT-REPORT Errors	invalidArgumentValue noSuchObjectClass	noSuchArgument noSuchObjectInstance	noSuchEventType processingFailure

Table 14

M-CREATE Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-CREATE Request	managedObjectClass	fill_managedObjectClass	extract_managedObjectClass
	managedObjectInstance	fill_managedObjectInstance	
	superiorObjectInstance	fill_superiorObjectInstance	extract_createObjectInstance
	accessControl	fill_accessControl	extract_accessControl
	referenceObjectInstance	fill_referenceObjectInstance	extract_referenceObjectInstance
	attribute List	fill_attributeList	extract_attributeList
M-CREATE Result	managedObjectClass	fill_managedObjectClass	extract_managedObjectClass
	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
	currentTime	fill_currentTime	extract_currentTime
	attributeList	fill_attributeList	extract_attributeList
M-CREATE Errors	accessDenied	classInstanceConflict	duplicateManagedObjectInstance
	invalidAttributeValue	invalidObjectInstance	missingAttributeValue
	noSuchAttribute	noSuchObjectClass	noSuchObjectInstance
	noSuchReferenceObject	processingFailure	

M-DELETE Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-DELETE Request	baseManagedObjectClass	fill_baseManagedObjectClass	extract_baseManagedObjectClass
	baseManagedObjectInstance	fill_baseManagedObjectInstance	extract_baseManagedObjectInstance
	accessControl	fill_accessControl	extract_accessControl
	synchronization	fill_synchronization	extract_synchronization
	scope	fill_scope	extract_scope
	filter	fill_filter	extract_filter
M-DELETE Result	managedObjectClass	fill_managedObjectClass	extract_managedObjectClass
	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
	currentTime	fill_currentTime	extract_currentTime
M-DELETE Errors	accessDenied	classInstanceConflict	complexityLimitation
	invalidFilter	invalidScope	noSuchObjectClass
	noSuchObjectInstance	processingFailure	syncNotSupported

Table 15

M-CANCELGET Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-CANCELGET Request	InvokeId	fill_invoke_id	extract_invoke_id
M-CANCELGET Result	None	None	None
M-CANCELGET Errors	mistypedOperation	noSuchInvokeId	processingFailure

M-LINKEDREPLY Operation

CMIS Service	CMIS Parameter	NeMaSOS Fill Function	NeMaSOS Extract Function
M-LINKEDREPLY Request	See Section 3.9	See Section 3.9	See Section 3.9
M-LINKEDREPLY Result	See Section 3.9	See Section 3.9	See Section 3.9
M-LINKEDREPLY Errors	mistypedOperation	noSuchInvokeId	processingFailure

Table 16

ERRORS			
CMIS Error	**CMIS Parameter**	**NeMaSOS Fill Function**	**NeMaSOS Extract Function**
accessDenied	NONE	NONE	NONE
classInstanceConflict	baseManagedObjectClass baseManagedObjectInstance	fill_baseManagedObjectClass fill_baseManagedObjectInstance	extract_baseManagedObjectClass extract_baseManagedObjectInstanc
complexityLimitation	scope filter synchronization	fill_scope fill_filter fill_synchronization	extract_scope extract_filter extract_synchronization
duplicateManagedObject Instance	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
getListError	managedObjectClass managedObjectInstance currentTime getInfoList	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_getInfoStatus	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_getInfoStatus
invalidArgumentValue	actionValue or eventValue	fill_actionValue or fill_eventValue	extract_value extract_value
invalidAttributeValue	Attribute	fill_attribute	extract_attribute
invalidFilter	filter	fill_filter	extract_filter
invalidScope	scope	fill_scope	extract_scope
invalidObjectInstance	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
missingAttributeValue	AttributeId	fill_attributeId	extract_attributeId
noSuchAction	managedObjectClass actionType	fill_managedObjectClass fill_actionType	extract_managedObjectClass extract_actionType

Table 17

ERRORS			
CMIS Error	**CMIS Parameter**	**NeMaSOS Fill Function**	**NeMaSOS Extract Function**
noSuchArgument	actionId or eventId	fill_actionId fill_eventId	extract_Id extract_Id
noSuchAttribute	AttributeId	fill_attributeId	extract_attributeId
noSuchEventType	managedObjectClass eventType	fill_managedObjectClass fill_eventType	extract_managedObjectClass extract_eventType
noSuchObjectClass	managedObjectClass	fill_managedObjectClass	extract_managedObjectClass
noSuchObjectInstance	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
noSuchReferenceObject	managedObjectInstance	fill_managedObjectInstance	extract_managedObjectInstance
processingFailure	managedObjectClass managedObjectInstance specificErrorInfo	fill_managedObjectClass fill_managedObjectInstance fill_specificErrorInfo	extract_managedObjectClass extract_managedObjectInstance extract_specificErrorInfo
setListError	managedObjectClass managedObjectInstance currentTime setInfoList	fill_managedObjectClass fill_managedObjectInstance fill_currentTime fill_setInfoStatus	extract_managedObjectClass extract_managedObjectInstance extract_currentTime extract_setInfoStatus
syncNotSupported	synchronization	fill_synchronization	extract_synchronization
mistypedOperation	NONE	NONE	NONE
noSuchInvokeId	NONE	NONE	NONE
operationCancelled	NONE	NONE	NONE

Table 18

3.2. CMIS M-SET operation

The CMIS M-SET operation is used to modify attribute values of managed objects by setting them to the values specified in the M-SET request. There are four services associated with this operation in the confirmed mode: M-SET request, M-SET indication, M-SET response, and M-SET confirm. In the unconfirmed mode, only the request and indication services are used. These four services are described in the following subsections.

Please note that in the confirmed mode, the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.2.1. CMIS M-SET request

The CMIS M-SET request service enables the user to issue a request for the M-SET operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-SET request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-SET request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_baseManagedObjectClass()	- mandatory function call.
fill_baseManagedObjectInstance()	- mandatory function call.
fill_accessControl()	- optional function call.
fill_synchronization()	- optional function call.
fill_scope()	- optional function call.
fill_filter()	- optional function call.
fill_modificationList()	- mandatory function call.
request()	- mandatory function call (must be last).

3.2.2. CMIS M-SET indication

The CMIS M-SET indication signals the receipt of an M-SET request and contains the information passed in the M-SET request PDU. The functions listed below allow the user to extract the information from the M-SET indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_baseManagedObjectClass()
extract_baseManagedObjectInstance()
extract_accessControl()
extract_synchronization ()
extract_scope()
extract_filter()
extract_modificationlist()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_SetArgument) contains the CMIP parameter information for both an M-SET request and an M-SET indication:

```
struct type_CMIP_SetArgument {
    struct type_CMIP_ObjectClass *baseManagedObjectClass;
    struct type_CMIP_ObjectInstance *baseManagedObjectInstance;
    struct type_CMIP_AccessControl *accessControl;
    struct type_CMIP_CMISSync *synchronization;
    struct type_CMIP_Scope *scope;
    struct type_CMIP_CMISFilter *filter;
    struct member_CMIP_7 {
            struct element_CMIP_11
            {
                    struct type_CMIP_ModifyOperator *modifyOperator;
                 struct type_CMIP_AttributeId *attributeId;
                    PE      attributeValue;
            } *member_CMIP_8;
         struct member_CMIP_7 *next;
    } *modificationList;
```

3.2.3. CMIS M-SET response

The CMIS M-SET response operation is used to respond to an M-SET request after having performed the requested M-SET operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-SET response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-SET response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
fill_attributeList()	- optional function call.
response()	- mandatory function call (must be last).

3.2.4. CMIS M-SET confirm

The CMIS M-SET confirm signals the receipt of an M-SET response and contains the information passed in the M-SET response PDU. The functions listed below allow the user to extract the information from the CMIP M-SET confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_attributeList()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_SetResult) contains the CMIP parameter information for an M-SET response and an M-SET confirm:

```
struct type_CMIP_SetResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct member_CMIP_10 {
            struct type_CMIP_Attribute *Attribute;
            struct member_CMIP_10 *next;
    } *attributeList;
};
```

3.3. CMIS M-GET operation

The CMIS M-GET operation is used by a CMISE-service-user to retrieve attribute values from a peer CMISE-service-user. There are four services associated with this operation: M-GET request, M-GET indication, M-GET response, and M-GET confirm. In accordance with the standard, the CMIS M-GET service is only provided in the confirmed mode. These four services are described in the following subsections.

Please note that the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.3.1. CMIS M-GET request

The CMIS M-GET request service enables the user to issue a request for the M-GET operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-GET request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-GET request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_baseManagedObjectClass()	- mandatory function call.
fill_baseManagedObjectInstance()	- mandatory function call.
fill_accessControl()	- optional function call.
fill_synchronization()	- optional function call.
fill_scope()	- optional function call.
fill_filter()	- optional function call.
fill_attributeIdlist()	- mandatory function call.
request()	- mandatory function call (must be last).

3.3.2. CMIS M-GET indication

The CMIS M-GET indication signals the receipt of an M-GET request and contains the information passed in the M-GET request PDU. The functions listed below allow the user to extract the information from the M-GET indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_baseManagedObjectClass()
extract_baseManagedObjectInstance()
extract_accessControl()
extract_synchronization()
extract_scope()
extract_filter()
extract_attributeIdlist()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_GetArgument) contains the CMIP parameter information for both an M-GET request and an M-GET indication:

```
struct type_CMIP_GetArgument {
    struct type_CMIP_ObjectClass *baseManagedObjectClass;
    struct type_CMIP_ObjectInstance *baseManagedObjectInstance;
    struct type_CMIP_AccessControl *accessControl;
    struct type_CMIP_CMISSync *synchronization;
    struct type_CMIP_Scope *scope;
    struct type_CMIP_CMISFilter *filter;
    struct member_CMIP_4 {
            struct type_CMIP_AttributeId *AttributeId;
            struct member_CMIP_4 *next;
    } *attributeList;
};
```

3.3.3. CMIS M-GET response

The CMIS M-GET response operation is used to respond to an M-GET request after having performed the requested M-GET operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-GET response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-GET response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
fill_attributeList()	- optional function call.
response()	- mandatory function call (must be last).

3.3.4. CMIS M-GET confirm

The CMIS M-GET confirm signals the receipt of an M-GET response and contains the information passed in the M-GET response PDU. The functions listed below allow the user to extract the information from the CMIP M-GET confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_attributeList()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_GetResult) contains the CMIP parameter information for both an M-GET response and an M-GET confirm:

```
struct type_CMIP_GetResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct member_CMIP_6 {
        struct type_CMIP_Attribute *Attribute;
        struct member_CMIP_6 *next;
    } *attributeList;
};
```

3.4. CMIS M-ACTION operation

The CMIS M-ACTION operation is used by a CMISE-service-user to request a peer CMISE-service-user to perform an action on managed object(s). There are four services associated with this operation in the confirmed mode: M-ACTION request, M-ACTION indication, M-ACTION response, and M-ACTION confirm. In the unconfirmed mode, only the request and indication services are used. These four services are described in the following subsections.

Please note that in the confirmed mode, the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.4.1. CMIS M-ACTION request

The CMIS M-ACTION request service enables the user to issue a request for the M-ACTION operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-ACTION request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-ACTION request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_baseManagedObjectClass()	- mandatory function call.
fill_baseManagedObjectInstance()	- mandatory function call.
fill_accessControl()	- optional function call.
fill_synchronization()	- optional function call.
fill_scope()	- optional function call.
fill_filter()	- optional function call.
fill_actionInfo()	- mandatory function call.
request()	- mandatory function call (must be last).

3.4.2. CMIS M-ACTION indication

The CMIS M-ACTION indication signals the receipt of an M-ACTION request and contains the information passed in the M-ACTION request PDU. The functions listed below allow the user to extract the information from the M-ACTION indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_baseManagedObjectClass()
extract_baseManagedObjectInstance()
extract_accessControl()
extract_synchronization()
extract_scope()
extract_filter()
extract_actionInfo()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_ActionArgument) contains the CMIP parameter information for both an M-ACTION request and an M-ACTION indication:

```
struct type_CMIP_ActionArgument {
    struct type_CMIP_ObjectClass *baseManagedObjectClass;
    struct type_CMIP_ObjectInstance *baseManagedObjectInstance;
    struct type_CMIP_AccessControl *accessControl;
    struct type_CMIP_CMISSync *synchronization;
    struct type_CMIP_Scope *scope;
    struct type_CMIP_CMISFilter *filter;
    struct type_CMIP_ActionInfo *actionInfo;
};
```

3.4.3. CMIS M-ACTION response

The CMIS M-ACTION response operation is used to respond to an M-ACTION request after having performed the requested M-ACTION operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-ACTION response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-ACTION response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
fill_actionReply()	- optional function call.
response()	- mandatory function call (must be last).

3.4.4. CMIS M-ACTION confirm

The CMIS M-ACTION confirm signals the receipt of an M-ACTION response and contains the information passed in the M-ACTION response PDU. The functions listed below allow the user to extract the information from the CMIP M-ACTION confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_actionReply()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_ActionResult) contains the CMIP parameter information for both an M-ACTION response and an M-ACTION confirm:

```
struct type_CMIP_ActionResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct type_CMIP_ActionReply *actionReply;
};
```

3.5. CMIS M-DELETE operation

The CMIS M-DELETE operation is used by an invoking CMISE-service-user to request a peer CMISE-service-user to delete a managed object instance and to de-register its identification. There are four services associated with this operation: M-DELETE request, M-DELETE indication, M-DELETE response, and M-DELETE confirm. In accordance with the standard, the CMIS M-DELETE service is only provided in the confirmed mode. These four services are described in the following subsections.

Please note that the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.5.1. CMIS M-DELETE request

The CMIS M-DELETE request service enables the user to issue a request for the M-DELETE operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-DELETE request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be the init_operation_struct(); the last function called must be the request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-DELETE request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_baseManagedObjectClass()	- mandatory function call.
fill_baseManagedObjectInstance()	- mandatory function call.
fill_accessControl()	- optional function call.
fill_synchronization ()	- optional function call.
fill_scope()	- optional function call.
fill_filter()	- optional function call.
request()	- mandatory function call (must be last).

3.5.2. CMIS M-DELETE indication

The CMIS M-DELETE indication signals the receipt of an M-DELETE request and contains the information passed in the M-DELETE request PDU. The functions listed below allow the user to extract the information from the M-DELETE indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_baseManagedObjectClass()
extract_baseManagedObjectInstance()
extract_accessControl()
extract_synchronization ()
extract_scope()
extract_filter()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_DeleteArgument) contains the CMIP parameter information for both an M-DELETE request and an M-DELETE indication:

```
struct type_CMIP_DeleteArgument {
    struct type_CMIP_ObjectClass *baseManagedObjectClass;
    struct type_CMIP_ObjectInstance *baseManagedObjectInstance;
    struct type_CMIP_AccessControl *accessControl;
    struct type_CMIP_CMISSync *synchronization;
    struct type_CMIP_Scope *scope;
    struct type_CMIP_CMISFilter *filter;
};
```

3.5.3. CMIS M-DELETE response

The CMIS M-DELETE response operation is used to respond to an M-DELETE request after having performed the requested M-DELETE operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-DELETE response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-DELETE response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
response()	- mandatory function call (must be last).

3.5.4. CMIS M-DELETE confirm

The CMIS M-DELETE confirm signals the receipt of an M-DELETE response and contains the information passed in the M-DELETE response PDU. The functions listed below allow the user to extract the information from the CMIP M-DELETE confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_DeleteArgument) contains the CMIP parameter information for an M-DELETE response and M-DELETE confirm:

```
struct type_CMIP_DeleteResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
};
```

3.6. CMIS M-CREATE operation

The CMIS M-CREATE operation is used by an invoking CMISE-service-user to request a peer CMISE-service-user to create and register a local identifier for a new managed object instance, complete with its identification and values for associated management information. There are four services associated with this operation: M-CREATE request, M-CREATE indication, M-CREATE response, and M-CREATE confirm. In accordance with the standard, the CMIS M-GET service is only provided in the confirmed mode. These four services are described in the following subsections.

Please note that in the confirmed mode, the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.6.1. CMIS M-CREATE request

The CMIS M-CREATE request service enables the user to issue a request for the M-CREATE operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-CREATE request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-CREATE request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- mandatory function call.
fill_superiorObjectInstance() or fill_managedObjectInstance()	- mandatory function call.
fill_accessControl()	- optional function call.
fill_referenceObjectInstance()	- mandatory function call.
fill_attributeList()	- mandatory function call.
request()	- mandatory function call (must be last).

3.6.2. CMIS M-CREATE indication

The CMIS M-CREATE indication signals the receipt of an M-CREATE request and contains the information passed in the M-CREATE request PDU. The functions listed below allow the user to extract the information from the M-CREATE indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_createObjectInstance ()
extract_accessControl()
extract_referenceObjectInstance ()
extract_attributeList()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_CreateArgument) contains the CMIP parameter information for both an M-CREATE request and an M-CREATE indication:

```
    struct type_CMIP_CreateArgument {
        struct type_CMIP_ObjectClass *managedObjectClass;
        struct choice_CMIP_1 {
            int     offset;
#define     choice_CMIP_1_distinguishedName          1
#define     choice_CMIP_1_nonSpecificForm            2
#define     choice_CMIP_1_localDistinguishedName     3
#define     choice_CMIP_1_superiorObjectInstance     4
            union {
                struct type_CMIP_DistinguishedName *distinguishedName;
                struct qbuf *nonSpecificForm;
                struct type_CMIP_RDNSequence *localDistinguishedName;
                struct type_CMIP_ObjectInstance *superiorObjectInstance;
            }       un;
        } *element_CMIP_2;
        struct type_CMIP_AccessControl *accessControl;
        struct type_CMIP_ObjectInstance *referenceObjectInstance;
        struct member_CMIP_2 {
            struct type_CMIP_Attribute *Attribute;
            struct member_CMIP_2 *next;
        } *attributeList;
    };
```

3.6.3. CMIS M-CREATE response

The CMIS M-CREATE response operation is used to respond to an M-CREATE request after having performed the requested M-CREATE operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-CREATE response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-CREATE response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
fill_attributeList()	- optional function call.
response()	- mandatory function call (must be last).

3.6.4. CMIS M-CREATE confirm

The CMIS M-CREATE confirm signals the receipt of an M-CREATE response and contains the information passed in the M-CREATE response PDU. The functions listed below allow the user to extract the information from the CMIP M-CREATE confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currenttime()
extract_attributeList()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_CreateResult) contains the CMIP parameter information for both an M-CREATE response and an M-CREATE confirm:

```
struct type_CMIP_CreateResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct member_CMIP_3 {
            struct type_CMIP_Attribute *Attribute;
            struct member_CMIP_3 *next;
    } *attributeList;
};
```

3.7. CMIS M-EVENT operation

The CMIS M-EVENT operation is used by a CMISE-service-user to report an event to a peer CMISE-service-user. There are four services associated with this operation in the confirmed mode: M-EVENT request, M-EVENT indication, M-EVENT response, and M-EVENT confirm. In the unconfirmed mode, only the request and indication services are used. These four services are described in the following subsections.

Please note that in the confirmed mode, the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.7.1. CMIS M-EVENT request

The CMIS M-EVENT request service enables the user to issue a request for the M-EVENT operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-EVENT request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-EVENT request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- mandatory function call.
fill_managedObjectInstance()	- mandatory function call.
fill_eventTime()	- optional function call.
fill_eventType()	- optional function call.
fill_eventInfo()	- optional function call.
request()	- mandatory function call (must be last).

3.7.2. CMIS M-EVENT indication

The CMIS M-EVENT indication signals the receipt of an M-EVENT request and contains the information passed in the M-EVENT request PDU. The functions listed below allow the user to extract the information from the M-EVENT indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_eventTime()
extract_eventType()
extract_eventInfo()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_EventReportArgument) contains the CMIP parameter information for both an M-EVENT request and an M-EVENT indication:

```
struct type_CMIP_EventReportArgument {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *eventTime;
    struct type_CMIP_EventTypeId *eventType;
    PE      eventInfo;
};
```

3.7.3. CMIS M-EVENT response

The CMIS M-EVENT response operation is used to respond to an M-EVENT request after having performed the requested M-EVENT operation, and to convey information associated with the successful result of that operation. Several functions comprise the support for the M-EVENT response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-EVENT response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_managedObjectClass()	- optional function call.
fill_managedObjectInstance()	- optional function call.
fill_currentTime()	- optional function call.
fill_eventReply()	- optional function call.
response()	- mandatory function call (must be last).

3.7.4. CMIS M-EVENT confirm

The CMIS M-EVENT confirm signals the receipt of an M-EVENT response and contains the information passed in the M-EVENT response PDU. The functions listed below allow the user to extract the information from the CMIP M-EVENT confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_eventReply()
free_operation_struct()
```

Data Structure

The following data structure (i.e., type_CMIP_EventReportResult) contains the CMIP parameter information for both an M-EVENT response and an M-EVENT confirm:

```
struct type_CMIP_EventReportResult {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct type_CMIP_EventReply *eventReply;
};
```

3.8. CMIS M-CANCELGET operation

The CMIS M-CANCELGET operation is used to halt the current execution of a previously issued M-GET request by specifying the invoke identifier of the M-GET in the M-CANCELGET request. There are four services associated with this operation in the confirmed mode: M-CANCELGET request, M-CANCELGET indication, M-CANCELGET response, and M-CANCELGET confirm. These four services are described in the following subsections.

Please note that in the confirmed mode, the response and confirm services are only used when the operation has been fully successful. When the operation has been only partially successful or unsuccessful, responses take the form of errors which are returned using the appropriate error services and functions described in this document.

3.8.1. CMIS M-CANCELGET request

The CMIS M-CANCELGET request service enables the user to issue a request for the M-CANCELGET operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-CANCELGET request service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be request(). The following list designates those library functions available to the CMIS user to formulate and execute an M-CANCELGET request. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
fill_invoke_id()	- mandatory function call.
request()	- mandatory function call (must be last).

3.8.2. CMIS M-CANCELGET indication

The CMIS M-CANCELGET indication signals the receipt of an M-CANCELGET request and contains the information passed in the M-CANCELGET request PDU. The functions listed below allow the user to extract the information from the M-CANCELGET indication message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

extract_cmip_message()
extract_invoke_id()
free_operation_struct()

Data Structure

The following data structure (i.e., type_CMIP_InvokeIDType) contains the CMIP parameter information for both an M-CANCELGET request and an M-CANCELGET indication:

```
struct type_CMIP_InvokeIDType {
    integer    parm;
};
```

3.8.3. CMIS M-CANCELGET response

The CMIS M-CANCELGET response operation is used to respond to an M-CANCELGET request after having performed the requested M-CANCELGET operation. Several functions comprise the support for the M-CANCELGET response service. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute an M-CANCELGET response. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- mandatory function call (must be first).
response()	- mandatory function call (must be last).

3.8.4. CMIS M-CANCELGET confirm

The CMIS M-CANCELGET confirm signals the receipt of an M-CANCELGET response and contains the information passed in the M-CANCELGET response PDU. The functions listed below allow the user to extract the information from the CMIP M-CANCELGET confirm message and place it in local data structures. The order in which these functions are invoked by the user is not critical other than that the extract_cmip_message() function must be the first function called because all other functions act on the message structure returned by this function call. Naturally, the free_operation_struct() function should not be called until the message is no longer needed, since it deallocates the message structure.

extract_cmip_message()
free_operation_struct()

Data Structure

No data structure is associated with this response.

3.9. CMIS M-LINKEDREPLY operation

The CMIS M-LINKEDREPLY operation is used to send a linked response. There are two services associated with this operation: M-LINKEDREPLY request, and M-LINKEDREPLY indication. These two services are described in the following subsections.

3.9.1. CMIS M-LINKEDREPLY request

The CMIS M-LINKEDREPLY request service enables the user to issue a request for the M-LINKEDREPLY operation to be performed and enables the user to pass the information necessary to support the performance of this operation. Several functions comprise the support for the M-LINKEDREPLY request service. Depending on which type of linked reply the user wishes to send will determine the functions to fill this request. In the table below is a list of the different types of linked replies that can be sent, along with the corresponding section in this document that specifies the functions to be called to fill the information for that type.

linked reply type	section for fill functions decriptions
getResult	3.3.3
getListError	4.5
setResult	3.2.3
setListError	4.20
actionResult	3.4.3
processingFailure	4.19
deleteResult	3.5.3
actionError	3.9.3
deleteError	3.9.4

After filling the parameter information for this linked reply the user needs to call the response() function to send this linked reply. When the user calls the response function they must set the linked parameter to a 1 to specify that this is a linked reply.

3.9.2. CMIS M-LINKEDREPLY indication

The CMIS M-LINKEDREPLY indication signals the receipt of an M-LINKEDREPLY request and contains the information passed in the M-LINKEDREPLY request PDU. Depending on which type of linked reply the user received, will determine the functions to extract the information from this request. In the table below is a list of the different types of linked replies that can be received, along with the corresponding section in this document that specifies the functions to be called to extract the information for that type.

linked reply type	section for fill functions decriptions
getResult	3.3.2
getListError	4.5
setResult	3.2.2
setListError	4.20
actionResult	3.4.2
processingFailure	4.19
deleteResult	3.5.2
actionError	3.9.3
deleteError	3.9.4

Data Structure

The following data structure (i.e., type_CMIP_LinkedReplyArgument) contains the CMIP parameter information for an M-LINKEDREPLY request and an M-LINKEDREPLY indication:

```
struct type_CMIP_LinkedReplyArgument {
   int      offset;
#define type_CMIP_LinkedReplyArgument_getResult           1
#define type_CMIP_LinkedReplyArgument_getListError    2
#define type_CMIP_LinkedReplyArgument_setResult           3
#define type_CMIP_LinkedReplyArgument_setListError    4
#define type_CMIP_LinkedReplyArgument_actionResult    5
#define type_CMIP_LinkedReplyArgument_processingFailure       6
#define type_CMIP_LinkedReplyArgument_deleteResult    7
#define type_CMIP_LinkedReplyArgument_actionError     8
#define type_CMIP_LinkedReplyArgument_deleteError     9

   union {
         struct type_CMIP_GetResult *getResult;
         struct type_CMIP_GetListError *getListError;
         struct type_CMIP_SetResult *setResult;
         struct type_CMIP_SetListError *setListError;
         struct type_CMIP_ActionResult *actionResult;
         struct type_CMIP_ProcessingFailure *processingFailure;
         struct type_CMIP_DeleteResult *deleteResult;
         struct type_CMIP_ActionError *actionError;
         struct type_CMIP_DeleteError *deleteError;
   }     un;
};
```

So the sequence to filling the above data structure to send a M-LINKEDREPLY request is: first determine the type of linked reply you are sending (i.e, getResult), then fill the structure (i.e, getResult) by calling the functions in the appropriate section of this document (i.e, 3.3.3), then finally to send the M-LINKEDREPLY request call the request() function with the linked parameter set to 1. Likewise to extract the information from the above data structure to receive a M-LINKEDREPLY request: first determine the type of linked reply you are receiveing (i.e, getResult), and then extract the structure (i.e, getResult) information by calling the functions in the appropriate section of this document (i.e, 3.3.2).

3.9.3. CMIS delete error for M-LINKEDREPLY request and indication

The CMIS delete error is used to indicate that the requested linked operation was not performed because acess was denied. Several functions comprise the support for the CMIS delete error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS delete error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this linked reply message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_managedObjectInstance()
fill_currenttime()
fill_deleteErrorInfo()
response()
```

To receive this linked reply message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currenttime()
extract_deleteErrorInfo()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS delete error:

```
     struct type_CMIP_DeleteError {
        struct type_CMIP_ObjectClass *managedObjectClass;

        struct type_CMIP_ObjectInstance *managedObjectInstance;

        struct type_UNIV_GeneralizedTime *currentTime;

        integer     deleteErrorInfo;
#define int_CMIP_deleteErrorInfo_accessDenied   2
     };
```

3.9.4. CMIS action error for M-LINKEDREPLY request and indication

The CMIS action error is used to indicate that the requested linked operation was not performed because of one of the following reasons: access denied, no such action, no such argument, invalid argument value. Several functions comprise the support for the CMIS action error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be response(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS action error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this linked reply message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_managedObjectInstance()
fill_currenttime()
fill_actionErrorInfo()
response()
```

To receive this linked reply message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currenttime()
extract_actionErrorInfo()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS action error:

```
struct type_CMIP_ActionError {
    struct type_CMIP_ObjectClass *managedObjectClass;

    struct type_CMIP_ObjectInstance *managedObjectInstance;

    struct type_UNIV_GeneralizedTime *currentTime;

    struct type_CMIP_ActionErrorInfo *actionErrorInfo;
};
```

4. CMIS Errors

When a CMIS confirmed operation is requested, a response is expected in return. Previous sections of this programmer's reference manual discussed appropriate responses and response procedures when the confirmed operation was performed successfully. This section discusses the alternative situation when a confirmed operation was not successfully accomplished. In particular, when a CMIS confirmed operation was either partially or fully unsuccessful, the response takes the form of an error. The CMIS user is responsible both for determining if an error has occurred and for sending the appropriate error response for the given CMIS operation, as specified in the CMIS standard. The mechanisms used to send and receive CMIS errors are discussed below.

After determining the appropriate error message to be sent, the user should formulate and send the message in the following way: 1) initialize the message data structure using the init_operation_struct() function, 2) fill in the message structure with any appropriate parameter information using the "fill_.." functions, and 3) call the send_error() function to send the error PDU. Since the same basic set of functions is used for all errors, the "msg_type" parameter provides an input to each of these functions to indicate the particular error type that is to be initialized, filled, or sent.

For receiving error messages, the following mechanisms are provided to the CMIS user. The user first calls the extract_cmip_message() function to retrieve a pending message from the CMIS queue. By looking at the message type of this message, the user should know the nature of the contents of that message. Based on this knowledge of the message type, the user should then call the appropriate "extract_...()" function(s) to obtain the message information. After retrieving all the information from the message, when the message structure is no longer needed, the user should delete the message using the free_operation_struct() function to free the previously allocated space.

What follows is a listing of all possible CMIS errors, along with the functions used to send and receive these errors. As mentioned above, only an error from the set of errors appropriate for a given CMIS operation (as specified in the CMIS standard) should be sent in response to a partially or completely failed operation.

4.1. CMIS access denied error

The CMIS access denied error is used to indicate that the requested operation was not performed for reasons related to the security of the open system. Two functions comprise the support for the CMIS access denied error. The first function called must be init_operation_struct(); the last function called must be send_error(). Because no parameter information is passed with this error, no "fill_.." functions are needed. The following list designates those library functions available to the CMIS user to formulate and execute an CMIS access denied error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- (must be first).
send_error()	- (must be last).

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
free_operation_struct()

Data Structure

No data structure is associated with this error.

4.2. CMIS class instance conflict error

The CMIS class instance conflict error is used to signify that the requested operation was not performed because the specified managed object instance is not a member of the specified base managed object class. Several functions comprise the support for the CMIS class instance conflict error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS class instance conflict error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_baseManagedObjectClass()
fill_baseManagedObjectInstance()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_baseManagedObjectClass()
extract_baseManagedObjectInstance()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS class instance conflict error:

```
struct type_CMIP_BaseManagedObjectId {
    struct type_CMIP_ObjectClass *baseManagedObjectClass;
    struct type_CMIP_ObjectInstance *baseManagedObjectInstance;
};
```

4.3. CMIS complexity limitation error

The CMIS complexity limitation error is used to indicate that the requested operation was not performed because a parameter was too complex. Several functions comprise the support for the CMIS complexity limitation error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS complexity limitation error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_filter()
fill_scope()
fill_synchronization()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_filter()
extract_scope()
extract_synchronization()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS complexity limitation error:

```
struct type_CMIP_ComplexityLimitation {
    struct type_CMIP_Scope *scope;
    struct type_CMIP_CMISFilter *filter;
    struct type_CMIP_CMISSync *sync;
};
```

4.4. CMIS duplicate managed object instance error

The CMIS duplicate managed object instance error is used to indicate that the requested operation was not performed because the specified managed object instance is not a member of the specified class. Several functions comprise the support for the CMIS duplicate managed object instance error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS duplicate managed object instance error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectInstance()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectInstance()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS duplicate managed object instance error:

```
    struct type_CMIP_ObjectInstance {
        int     offset;
#define     type_CMIP_ObjectInstance_distinguishedName      1
#define     type_CMIP_ObjectInstance_nonSpecificForm        2
#define     type_CMIP_ObjectInstance_localDistinguishedName 3
        union {
            struct type_CMIP_DistinguishedName *distinguishedName;
            struct qbuf *nonSpecificForm;
            struct type_CMIP_RDNSequence *localDistinguishedName;
        }   un;
    };
```

4.5. CMIS get list error

The CMIS get list error is used to indicate that one or more attribute values were not read for one of the following two reasons: 1) access denied -- i.e., the requested operation was not performed for reasons pertinent to the security of the open system; or 2) no such attribute -- i.e, the identifier for the specified attribute or attribute group was not recognized. The attribute values that could be read are returned along with the error indication for those that could not be read. Several functions comprise the support for the CMIS get list error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS get list error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_managedObjectInstance()
fill_currentTime()
fill_getInfoStatus()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_getInfoStatus()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS get list error:

```
struct type_CMIP_GetListError {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct member_CMIP_5 {
          struct type_CMIP_GetInfoStatus *GetInfoStatus;
          struct member_CMIP_5 *next;
    } *getInfoList;
};
```

4.6. CMIS invalid argument value error

The CMIS invalid argument value error is used to indicate that the information value specified in the operation request was out of range, or otherwise inappropriate. This error can be sent for one of two reasons; this was either an inappropriate action value, or an inappropriate event value. Depending upon the reason for which this error is being sent, the CMIS user will need to call the appropriate fill function: either fill_actionValue() or fill_eventValue. Several functions comprise the support for the CMIS invalid argument value error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS invalid argument value error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_actionValue() or fill_eventValue()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_value()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS invalid argument value error:

```
    struct type_CMIP_InvalidArgumentValue {
        int     offset;
#define     type_CMIP_InvalidArgumentValue_actionValue      1
#define     type_CMIP_InvalidArgumentValue_eventValue       2
        union {
              struct type_CMIP_ActionInfo *actionValue;
              struct element_CMIP_8 {
                 struct type_CMIP_EventTypeId *eventType;
                 PE      eventInfo;
              } *eventValue;
        }       un;
    };
```

4.7. CMIS invalid attribute value error

The CMIS invalid attribute value error is used to indicate that an attribute value specified in the operation request was out of range, or otherwise inappropriate. Several functions comprise the support for the CMIS invalid attribute value error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS invalid attribute value error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_attribute()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_attribute()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS invalid attribute value error:

```
struct type_CMIP_Attribute {
    struct type_CMIP_AttributeId *attributeId;
    PE      attributeValue;
};
```

4.8. CMIS invalid filter error

The CMIS invalid filter error is used to indicate that the filter parameter contains an invalid assertion or an unrecognized logical operator. Several functions comprise the support for the CMIS invalid filter error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS invalid filter error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_filter()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_filter()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS invalid filter error:

```
      struct type_CMIP_CMISFilter {
          int     offset;
#define       type_CMIP_CMISFilter_item       1
#define       type_CMIP_CMISFilter_and        2
#define       type_CMIP_CMISFilter_or         3
#define       type_CMIP_CMISFilter_not        4
          union {
                struct type_CMIP_FilterItem *item;
                struct member_CMIP_0 {
                      struct type_CMIP_CMISFilter *CMISFilter;
                      struct member_CMIP_0 *next;
                } *and;
                struct member_CMIP_1 {
                      struct type_CMIP_CMISFilter *CMISFilter;
                      struct member_CMIP_1 *next;
                } *or;
                struct type_CMIP_CMISFilter *not;
          }      un;
      };
```

4.9. CMIS invalid scope error

The CMIS invalid scope error is used to indicate that the value of the scope parameter is invalid. Several functions comprise the support for the CMIS invalid scope error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS invalid scope error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_scope()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_scope()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS invalid scope error:

```
    struct type_CMIP_Scope {
        int     offset;
#define     type_CMIP_Scope_1                        1
#define     type_CMIP_Scope_individualLevels         2
#define     type_CMIP_Scope_baseToNthLevel           3
        union {
            integer     choice_CMIP_3;
#define     int_CMIP_choice_CMIP_3_baseObject        0
#define     int_CMIP_choice_CMIP_3_firstLevelOnly    1
#define     int_CMIP_choice_CMIP_3_wholeSubtree      2
            integer     individualLevels;
            integer     baseToNthLevel;
        }       un;
    };
```

4.10. CMIS invalid object instance error

The CMIS invalid object instance error is used to indicate that the object instance name specified implied a violation of the naming rules. Several functions comprise the support for the CMIS invalid object instance error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS invalid object instance error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectInstance()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectInstance()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS invalid object instance error:

```
        struct type_CMIP_ObjectInstance {
            int     offset;
#define      type_CMIP_ObjectInstance_distinguishedName       1
#define      type_CMIP_ObjectInstance_nonSpecificForm         2
#define      type_CMIP_ObjectInstance_localDistinguishedName  3
            union {
                struct type_CMIP_DistinguishedName *distinguishedName;
                struct qbuf *nonSpecificForm;
                struct type_CMIP_RDNSequence *localDistinguishedName;
            }       un;
        };
```

4.11. CMIS missing attribute value error

The CMIS missing attribute value error is used to indicate that a required attribute value was not supplied, and a default value was not available. Several functions comprise the support for the CMIS missing attribute value error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS missing attribute value error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_attributeId()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_attributeId()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS missing attribute value error:

```
    struct type_CMIP_Pseudo__missingAttributeValue {
            struct type_CMIP_AttributeId *AttributeId;
            struct type_CMIP_Pseudo__missingAttributeValue *next;
    };
           struct type_CMIP_AttributeId {
               int      offset;
#define         type_CMIP_AttributeId_globalForm       1
#define         type_CMIP_AttributeId_localForm        2
               union {
                      OID      globalForm;
                      integer    localForm;
               }       un;
           };
```

4.12. CMIS no such action error

The CMIS no such action error is used to indicate that the requested operation was not performed because the specified managed object instance is not a member of the specified managed object class. Several functions comprise the support for the CMIS no such action error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such action error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_actionType()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_actionType()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such action error:

```
struct type_CMIP_NoSuchAction {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ActionTypeId *actionType;
};
```

4.13. CMIS no such argument error

The CMIS no such argument error is used to indicate that either the event information specified was not recognized or the action information specified was not supported. Several functions comprise the support for the CMIS no such argument error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such argument error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_actionId(), or
fill_eventId()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_Id()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such argument error:

```
   struct type_CMIP_NoSuchArgument {
      int     offset;
#define      type_CMIP_NoSuchArgument_actionId     1
#define      type_CMIP_NoSuchArgument_eventId      2
      union {
              struct element_CMIP_9 {
                 struct type_CMIP_ObjectClass *managedObjectClass;
                 struct type_CMIP_ActionTypeId *actionType;
              } *actionId;
              struct element_CMIP_10 {
                 struct type_CMIP_ObjectClass *managedObjectClass;
                 struct type_CMIP_EventTypeId *eventType;
              } *eventId;
      }       un;
   };
```

4.14. CMIS no such attribute error

The CMIS no such attribute error is used to indicate that an attribute specified in the operation request was not recognized. Several functions comprise the support for the CMIS no such attribute error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such attribute error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_attributeId()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_attributeId()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such attribute error:

```
        struct type_CMIP_AttributeId {
            int     offset;
#define         type_CMIP_AttributeId_globalForm      1
#define         type_CMIP_AttributeId_localForm       2
            union {
                    OID     globalForm;
                    integer    localForm;
            }       un;
        };
```

4.15. CMIS no such event type error

The CMIS no such event type error is used to indicate that the requested event type was not recognized. Several functions comprise the support for the CMIS no such event type error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such event type error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_eventType()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_eventType()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such event type error:

```
struct type_CMIP_NoSuchEventType {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_EventTypeId *eventType;
};
```

4.16. CMIS no such object class error

The CMIS no such object class error is used to indicate that the class of the specified managed object was not recognized. Several functions comprise the support for the CMIS no such object class error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such object class error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such object class error:

```
        struct type_CMIP_ObjectClass {
            int      offset;
#define       type_CMIP_ObjectClass_globalForm       1
#define       type_CMIP_ObjectClass_localForm        2
            union {
                    OID      globalForm;
                    integer    localForm;
            }        un;
        };
```

4.17. CMIS no such object instance error

The CMIS no such object class error is used to indicate that the specified managed object instance was not recognized. Several functions comprise the support for the CMIS no such object instance error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such object instance error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectInstance()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectInstance()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such object instance error:

```
     struct type_CMIP_ObjectInstance {
        int     offset;
#define      type_CMIP_ObjectInstance_distinguishedName        1
#define      type_CMIP_ObjectInstance_nonSpecificForm          2
#define      type_CMIP_ObjectInstance_localDistinguishedName  3
        union {
             struct type_CMIP_DistinguishedName *distinguishedName;
             struct qbuf *nonSpecificForm;
             struct type_CMIP_RDNSequence *localDistinguishedName;
        }      un;
     };
```

4.18. CMIS no such reference object error

The CMIS no such reference object error is used to indicate that the reference object instance was not recognized. Several functions comprise the support for the CMIS no such reference object error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS no such reference object error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectInstance()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectInstance()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS no such reference object error:

```
    struct type_CMIP_ObjectInstance {
        int     offset;
#define     type_CMIP_ObjectInstance_distinguishedName      1
#define     type_CMIP_ObjectInstance_nonSpecificForm        2
#define     type_CMIP_ObjectInstance_localDistinguishedName 3
        union {
            struct type_CMIP_DistinguishedName *distinguishedName;
            struct qbuf *nonSpecificForm;
            struct type_CMIP_RDNSequence *localDistinguishedName;
        }     un;
    };
```

4.19. CMIS processing failure error

The CMIS processing failure error is used to indicate that a general failure in processing the operation was encountered. Several functions comprise the support for the CMIS processing failure error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS processing failure error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

```
init_operation_struct()
fill_managedObjectClass()
fill_managedObjectInstance()
fill_specificErrorInfo()
send_error()
```

To receive this error message and extract the information from it, use the following functions:

```
extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_specificErrorInfo()
free_operation_struct()
```

Data Structure

The following data structure holds the CMIP parameter information for a CMIS processing failure error:

```
struct type_CMIP_ProcessingFailure {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    PE      specificErrorInfo;
};
```

4.20. CMIS set list error

The CMIS set list error is used to indicate that one or more attribute values were not modified for one of the following three reasons: 1) access denied -- i.e, the requested operation was not performed for reasons pertinent to the security of the open system; 2) invalid attribute value -- i.e, the attribute value specified was out of range or otherwise inappropriate; or 3) no such attribute -- i.e, the identifier for the specified attribute or attribute group was not recognized. The attribute values that could be modified, were modified. Several functions comprise the support for the CMIS set list error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS set list error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_managedObjectClass()
fill_managedObjectInstance()
fill_currentTime()
fill_setInfoStatus()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_managedObjectClass()
extract_managedObjectInstance()
extract_currentTime()
extract_setInfoStatus()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS set list error:

```
struct type_CMIP_SetListError {
    struct type_CMIP_ObjectClass *managedObjectClass;
    struct type_CMIP_ObjectInstance *managedObjectInstance;
    struct type_UNIV_GeneralizedTime *currentTime;
    struct setInfoList {
            struct type_CMIP_SetInfoStatus *SetInfoStatus;
            struct setInfoList *next;
    } *setInfoList;
};
```

4.21. CMIS synchronization not supported error

The CMIS synchronization not supported error is used to indicate that the requested operation was not performed because the type of synchronization specified is not supported. Several functions comprise the support for the CMIS synchronization not supported error. Except for the first and last of these functions, the order in which they are called is not critical. The first function called must be init_operation_struct(); the last function called must be send_error(). The following list designates those library functions available to the CMIS user to formulate and execute a CMIS synchronization not supported error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual. To send this error message, use the following functions:

init_operation_struct()
fill_synchronization()
send_error()

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
extract_synchronization()
free_operation_struct()

Data Structure

The following data structure holds the CMIP parameter information for a CMIS synchronization not supported error:

```
        struct type_CMIP_CMISSync {
            integer     parm;
#define        int_CMIP_CMISSync_bestEffort        0
#define        int_CMIP_CMISSync_atomic            1
        };
```

4.22. CMIS mistyped operation error

The CMIS mistyped operation error is used to indicate that the requested cancel-get operation was not performed because the invoke identifier specified did not correspond to a GET request. Two functions comprise the support for the CMIS mistyped operation error. The first function called must be init_operation_struct(); the last function called must be send_error(). Because no parameter information is passed with this error, no "fill_.." functions are needed. The following list designates those library functions available to the CMIS user to formulate and execute a CMIS mistyped operation error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- (must be first).
send_error()	- (must be last).

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
free_operation_struct()

Data Structure

No data structure is associated with this error.

4.23. CMIS no such invokeid error

The CMIS no such invokeid error is used to indicate that the requested cancel GET operation was not performed beacause the invokid specified did not exist. Two functions comprise the support for the CMIS no such invokeid error. The first function called must be init_operation_struct(); the last function called must be send_error(). Because no parameter information is passed with this error, no "fill_.." functions are needed. The following list designates those library functions available to the CMIS user to formulate and execute an CMIS no such invokeid error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- (must be first).
send_error()	- (must be last).

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
free_operation_struct()

Data Structure

No data structure is associated with this error.

4.24. CMIS operation cancelled error

The CMIS operation cancelled error is used to indicate that the requested cancel GET operation was perfomed. This error is sent back with the invokid of the original get request, the cancel-get request is responded to with a cancel-get response to that invokeid. Two functions comprise the support for the CMIS operation cancelled error. The first function called must be init_operation_struct(); the last function called must be send_error(). Because no parameter information is passed with this error, no "fill_.." functions are needed. The following list designates those library functions available to the CMIS user to formulate and execute an CMIS operation cancelled error. Detailed descriptions of these functions, along with the function parameters, are provided later in this manual.

init_operation_struct()	- (must be first).
send_error()	- (must be last).

To receive this error message and extract the information from it, use the following functions:

extract_cmip_message()
free_operation_struct()

Data Structure

No data structure is associated with this error.

5. CMIS Parameter Fill Functions

This section provides a description of all the parameter fill functions contained in the CMIS interface. These functions are contained in "cmislib.a". The descriptions that follow contain descriptive overviews, input and output parameters, and parameter value and ranges, where appropriate. The following two structures are used extensively throughout the fill and extract routines, and are defined here for brevity.

```
union ID
{
   int        local_Form;
   char     *global_Form;
};
```

```
union Instance
{
   struct distinguishedName
   {
       char  *type;
       PE    value;
       int   RDN_flag;
   } DistinguishedName;
   struct qbuf *nonSpecificForm;
};
```

5.1. init_operation_struct

```
int init_operation_struct(msg_type, msg_ptr)
        int msg_type;
        char **msg_ptr;
```

Description

The *init_operation_struct()* routine allocates and initializes the outermost data structure used for sending CMIP messages (in particular, requests, responses, and errors). Since different CMIP operations require different information, and therefore different data structures, this function allocates the appropriate data structure based on the operation type indicated by the value of the input parameter "msg_type". Initialization of this data structure includes setting the pointers to contained structures for each of the CMIS parameters to NULL. In addition, the msg_ptr is set to point to this newly allocated data structure. Upon return from this function, this pointer is used as an input parameter to associated functions for this operation in order to point to the structure in which new information is to be inserted for the CMIP operation message.

Parameters

msg_type CMIS operation type

Range of Values

NO_SUCH_OBJECT_CLASS, NO_SUCH_OBJECT_INSTANCE, ACCESS_DENIED, SYNC_NOT_SUPPORTED, INVALID_FILTER, NO_SUCH_ATTRIBUTE, INVALID_ATTRIBUTE_VALUE, GET_LIST_ERROR, SET_LIST_ERROR, NO_SUCH_ACTION, PROCESSING_FAILURE, DUPLICATE_MANAGED_OBJECT_INSTANCE, NO_SUCH_REFERENCE_OBJECT, NO_SUCH_EVENT_TYPE, NO_SUCH_ARGUMENT, INVALID_ARGUMENT_VALUE, INVALID_SCOPE, INVALID_OBJECT_INSTANCE, MISSING_ATTRIBUTE_VALUE, CLASS_INSTANCE_CONFLICT, COMPLEXITY_LIMITATION, SET_REQ, SET_RSP, GET_REQ, GET_RSP, EVENT_REQ, EVENT_RSP, ACTION_REQ, ACTION_RSP, CREATE_REQ, CREATE_RSP, DELETE_REQ, DELETE_RSP.

msg_ptr This function sets this pointer to point to the newly allocated CMIP operation structure so that upon return from this function, the msg_ptr can be the pointer to the CMIP operation data structure.

Return Values SUCCESS, NO_MEM, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST

5.2. fill_baseManagedObjectClass

```
int fill_baseManagedObjectClass( id_type, id, msg_type, msg_ptr)
      int id_type;
      union ID id;
      int msg_type;
      char **msg_ptr;
```

Description

The *fill_baseManagedObjectClass()* routine fills the CMIS field for the basemanagedObjectClass identifier. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the class designation and sets the baseManagedObjectClass pointer in the CMIP operation structure to point to this newly allocated structure. Then the managed object class structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the Managed Object Class designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the Managed Object Class identifier in local form or a Character string containing the object class identifier in global form. (see section 3.1.1 for explanation of treatment of object identifiers by these interface functions)

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 through 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

msg_type CMIS operation type.

Range of Values

SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ, CLASS_INSTANCE_CONFLICT.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST,
NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.3. fill_managedObjectClass

```
int fill_managedObjectClass( id_type, id, msg_type, msg_ptr)
    int id_type;
    union ID id;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_managedObjectClass()* routine fills the CMIS field for the managedObjectClass identifier. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the class designation and sets the managedObjectClass pointer in the CMIP operation structure to point to this newly allocated structure. Then the managed object class structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether Managed Object Class designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the Managed Object Class identifier in local form, or a Character string containing the object class identifier in global form. (see section 3.1.1 for explanation of treatment of object identifiers by these interface functions)

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 through 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

msg_type CMIS operation type.

Range of Values

CREATE_REQ, CREATE_RSP, SET_RSP, GET_RSP, ACTION_RSP, DELETE_RSP, EVENT_REQ, EVENT_RSP, NO_SUCH_OBJECT_CLASS, GET_LIST_ERROR, SET_LIST_ERROR, NO_SUCH_ACTION, PROCESSING_FAILURE, NO_SUCH_EVENT_TYPE, DELETE_ERR, ACTION_ERR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.4. fill_attributeId

```
int fill_attributeId( id_type, id, msg_type, msg_ptr)
    int id_type;
    union ID id;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_attributeId()* routine fills the CMIS field for the attribute identifier. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the attribute ID designation and sets the attributeId pointer in the CMIP operation structure to point to this newly allocated structure. Then the attribute ID structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a Character string containing the attribute identifier in global form. (see section 3.1.1 for explanation of treatment of object identifiers by these interface functions)

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 through 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string)

msg_type CMIS operation type.

Range of Values

NO_SUCH_ATTRIBUTE , MISSING_ATTRIBUTE_VALUE.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST,
NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.5. fill_baseManagedObjectInstance

```
int fill_baseManagedObjectInstance ( instance_type, instance msg_type, msg_ptr)
    int      instance_type;
    union Instance instance;
    int      msg_type;
    char    **msg_ptr;
```

Description

The *fill_baseManagedObjectInstance()* routine fills the CMIP PDU field for the basemanagedObjectInstance name. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the name designation and sets the baseManagedObjectInstance pointer in the CMIP operation structure to point to this newly allocated structure. Then the managed object instance structure is filled with either the distinguishedname, localdistinguishedname, or nonspecificform, depending upon which name type was passed in by the CMIS user. NOTE: For this phase, if the user wishes to pass a nonspecificform, he may do so by filling it in himself and taking responsibility for the appropriate filling of this parameter. If this parameter is not filled correctly the program could fail.

The name structure is a distinguished name and is constituted as follows: a Distinguished Name (DN) is a sequence of one or more Relative Distinguished Names (RDNs). For this reason, a DN is also referred to as an RDN sequence. Each RDN, in turn, is a sequence of one or more Attribute Value Assertions (AVAs). And finally, an AVA is a pairing of an attribute type and an attribute value. Because of the complex nature of the name structure, this function may have to be called several times to build the entire name. The name can potentially consist of a linked list of linked lists of name elements. That is, the DN can be a linked list of RDNs which can, in turn, be linked lists of AVAs.

The following sequence of calls to this function should be followed to properly build the name structure. When the CMIS user is beginning the creation of a new distinguished name (DN), the RDN_flag in the union instance should be set to BEGIN_RDNSEQUENCE. After the initial step, when beginning a new RDN, set this flag to ADD_RELATIVDISTINGUISHEDNAME. When the CMIS user is adding an attribute value assertion (AVA) to an already existing RDN, this flag should be set to ADD_ATTRIBUTEVALUEASSERTION.

This function returns a SUCCESS indication. If any errors are detected prior to a successful completion of each function, the function is terminated at that point with the appropriate error indication. As can be seen from the sequence of calls to this function, the name structure must be filled in starting with the root of the DN naming tree and then filling in each AVA of the RDN at a given node in the path before progressing to the next RDN in the pathname. This procedure is reiterated until all RDNs in the RDN sequence have been filled in.

Parameters

instance_type Indicates if the user is filling a distinguishedName, nonSpecificForm, or localDistinguishedName.

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME.

instance This union supplies the necessary information to fill either the distinguishedname, localdistinguishedname, or nonspecificform. For the distinguishedname and local distinguishedname forms, the union contains the AVA as well as a flag indicating the current stage in building the name. This flag can take on one of three values:

Range of Values

BEGIN_RDNSEQUENCE, ADD_RELATIVEDISTINGUISHEDNAME,
or ADD_ATTRIBUTEVALUEASSERTION

msg_type CMIS operation type.

Range of Values

SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ, CLASS_INSTANCE_CONFLICT

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_NAME_TYPE, GLOB_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NEW_DN_RANGE, NO_MEM, FIELD_ALREADY_FILLED, BAD_FORM

5.6. fill_managedObjectInstance

```
int fill_managedObjectInstance ( instance_type, instance msg_type, msg_ptr)
    int       instance_type;
    union Instance instance;
    int       msg_type;
    char      **msg_ptr;
```

Description

The *fill_managedObjectInstance()* routine fills the CMIP PDU field for the managedObjectInstance name. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the name designation and sets the managedObjectInstance pointer in the CMIP operation structure to point to this newly allocated structure. Then the managed object instance structure is filled with either the distinguishedname, localdistinguishedname, or nonspecificform, depending upon which name type was passed in by the CMIS user. NOTE: For this phase if the user wishes to pass a nonspecificform, he may do so by filling it in himself and taking responsibility for the appropriate filling of this parameter. If this parameter is not filled correctly the program could fail.

The name structure is a distinguished name and is constituted as follows: a Distinguished Name (DN) is a sequence of one or more Relative Distinguished Names (RDNs). For this reason, a DN is also referred to as an RDN sequence. Each RDN, in turn, is a sequence of one or more Attribute Value Assertions (AVAs). And finally, an AVA is a pairing of an attribute type and an attribute value. Because of the complex nature of the name structure, this function may have to be called several times to build the entire name. The name can potentially consist of linked lists of linked lists of name elements. That is, the DN can be a linked list of RDNs which can, in turn, be a linked list of AVAs.

The following sequence of calls to this function should be followed to properly build the name structure. When the CMIS user is beginning the creation of a new distinguished name (DN), the RDN_flag in the union instance should be set to BEGIN_RDNSEQUENCE. After the initial step, when beginning a new RDN, set this flag to ADD_RELATIVDISTINGUISHEDNAME. When the CMIS user is adding an attribute value assertion (AVA) to an already existing RDN, this flag should be set to ADD_ATTRIBUTEVALUEASSERTION.

This function returns a SUCCESS indication. If any errors are detected prior to a successful completion of each function, the function is terminated at that point with the appropriate error indication. As can be seen from the sequence of calls to this function, the name structure must be filled in starting with the root of the DN naming tree and then filling in each AVA of the RDN at a given node in the path before progressing to the next RDN in the pathname. This procedure is reiterated until all RDNs in the RDN sequence have been filled in.

Parameters

instance_type Indicates if the user is filling a distinguishedName, nonSpecificForm, or localDistinguishedName.

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME.

instance This union supplies the necessary information to fill either the distinguishedname, localdistinguishedname, or nonspecificform. For the distinguishedname and local distinguishedname forms, the union contains the AVA as well as a flag indicating the current stage in building the name. This flag can take on one of three values:

Range of Values

BEGIN_RDNSEQUENCE, ADD_RELATIVEDISTINGUISHEDNAME,
or ADD_ATTRIBUTEVALUEASSERTION

msg_type CMIS operation type.

Range of Values

CREATE_REQ, CREATE_RSP, SET_RSP, GET_RSP, ACTION_RSP, DELETE_RSP, EVENT_REQ, EVENT_RSP, GET_LIST_ERROR, SET_LIST_ERROR, NO_SUCH_OBJECT_INSTANCE, NO_SUCH_REFERENCE_OBJECT, INVALID_OBJECT_INSTANCE, PROCESSING_FAILURE, DUPLICATE_MANAGED_OBJECT_INSTANCE, DELETE_ERR, ACTION_ERR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_NAME_TYPE, GLOB_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NEW_DN_RANGE, NO_MEM, FIELD_ALREADY_FILLED, BAD_FORM

5.7. fill_superiorObjectInstance

```
int fill_superiorObjectInstance ( instance_type, instance, msg_type, msg_ptr)
    int            instance_type;
    union Instance instance;
    int            msg_type;
    char          **msg_ptr;
```

Description

The *fill_superiorObjectInstance()* routine fills the CMIP PDU field for the superiorObjectInstance name. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the name designation and sets the superiorObjectInstance pointer in the CMIP operation structure to point to this newly allocated structure. Then the superior object instance structure is filled with either the distinguishedname, localdistinguishedname, or nonspecificform, depending upon which name type was passed in by the CMIS user. NOTE: For this phase, if the user wishes to pass a nonspecificform, he may do so by filling it in himself and taking responsibility for the appropriate filling of this parameter. If this parameter is not filled correctly the program could fail.

The name structure is a distinguished name and is constituted as follows: a Distinguished Name (DN) is a sequence of one or more Relative Distinguished Names (RDNs). For this reason, a DN is also referred to as an RDN sequence. Each RDN, in turn, is a sequence of one or more Attribute Value Assertions (AVAs). And finally, an AVA is a pairing of an attribute type and an attribute value. Because of the complex nature of the name structure, this function may have to be called several times to build the entire name. The name can potentially consist of a linked list of linked lists of name elements. That is, the DN can be a linked list of RDNs which can, in turn, be linked lists of AVAs.

The following sequence of calls to this function should be followed to properly build the name structure. When the CMIS user is beginning the creation of a new distinguished name (DN), the RDN_flag in the union instance should be set to BEGIN_RDNSEQUENCE. After the initial step, when beginning a new RDN, set this flag to ADD_RELATIVEDISTINGUISHEDNAME. When the CMIS user is adding an attribute value assertion (AVA) to an already existing RDN, this flag should be set to ADD_ATTRIBUTEVALUEASSERTION.

This function returns a SUCCESS indication. If any errors are detected prior to a successful completion of each function, the function is terminated at that point with the appropriate error indication. As can be seen from the sequence of calls to this function, the name structure must be filled in starting with the root of the DN naming tree and then filling in each AVA of the RDN at a given node in the path before progressing to the next RDN in the pathname. This procedure is reiterated until all RDNs in the RDN sequence have been filled in.

Parameters

instance_type Indicates if the user is filling a distinguishedName, nonSpecificForm, or localDistinguishedName.

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME.

instance This union supplies the necessary information to fill either the distinguishedname, localdistinguishedname, or nonspecificform. For the distinguishedname and local distinguishedname forms, the union contains the AVA as well as a flag indicating the current stage in building the name. This flag can take on one of three values:

Range of Values

BEGIN_RDNSEQUENCE, ADD_RELATIVEDISTINGUISHEDNAME,
or ADD_ATTRIBUTEVALUEASSERTION

msg_type CMIS operation type.

Range of Values

CREATE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_NAME_TYPE, GLOB_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NEW_DN_RANGE, NO_MEM, FIELD_ALREADY_FILLED, BAD_FORM

5.8. fill_referenceObjectInstance

```
int fill_referenceObjectInstance ( instance_type, instance msg_type, msg_ptr)
    int       instance_type;
    union Instance instance;
    int       msg_type;
    char    **msg_ptr;
```

Description

The *fill_referenceObjectInstance()* routine fills the CMIP PDU field for the referenceObjectInstance name. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the name designation and sets the referenceObjectInstance pointer in the CMIP operation structure to point to this newly allocated structure. Then the reference object instance structure is filled with either the distinguishedname, localdistinguishedname, or nonspecificform, depending upon which name type was passed in by the CMIS user. NOTE: For this phase if the user wishes to pass a nonspecificform, he may do so by filling it in himself and taking responsibility for the appropriate filling of this parameter. If this parameter is not filled correctly the program could fail.

The name structure is a distinguished name and is constituted as follows: a Distinguished Name (DN) is a sequence of one or more Relative Distinguished Names (RDNs). For this reason, a DN is also referred to as an RDN sequence. Each RDN, in turn, is a sequence of one or more Attribute Value Assertions (AVAs). And finally, an AVA is a pairing of an attribute type and an attribute value. Because of the complex nature of the name structure, this function may have to be called several times to build the entire name. The name can potentially consist of a linked list of linked lists of name elements. That is, the DN can be a linked list of RDNs which can, in turn, be linked lists of AVAs.

The following sequence of calls to this function should be followed to properly build the name structure. When the CMIS user is beginning the creation of a new distinguished name (DN), the RDN_flag in the union instance should be set to BEGIN_RDNSEQUENCE. After the initial step, when beginning a new RDN, set this flag to ADD_RELATIVEDISTINGUISHEDNAME. When the CMIS user is adding an attribute value assertion (AVA) to an already existing RDN, this flag should be set to ADD_ATTRIBUTEVALUEASSERTION.

This function returns a SUCCESS indication. If any errors are detected prior to a successful completion of each function, the function is terminated at that point with the appropriate error indication. As can be seen from the sequence of calls to this function, the name structure must be filled in starting with the root of the DN naming tree and then filling in each AVA of the RDN at a given node in the path before progressing to the next RDN in the pathname. This procedure is reiterated until all RDNs in the RDN sequence have been filled in.

Parameters

instance_type Indicates if the user is filling a distinguishedName, nonSpecificForm, or localDistinguishedName.

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME.

instance This union supplies the necessary information to fill either the distinguishedname, localdistinguishedname, or nonspecificform. For the distinguishedname and local distinguishedname forms, the union contains the AVA as well as a flag indicating the current stage in building the name. This flag can take on one of three values:

Range of Values

BEGIN_RDNSEQUENCE, ADD_RELATIVEDISTINGUISHEDNAME,
or ADD_ATTRIBUTEVALUEASSERTION

msg_type CMIS operation type.

Range of Values
CREATE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_NAME_TYPE, GLOB_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NEW_DN_RANGE, NO_MEM, FIELD_ALREADY_FILLED, BAD_FORM

5.9. fill_accessControl

```
int fill_accessControl( access, msg_type, msg_ptr)
    int access;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the access control field of the CMIP PDU. The *fill_accessControl()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the access control information and sets the accessControl pointer in the CMIP operation structure to point to this newly allocated structure. Then the access control field is filled with the access control information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: For this version of the implementation, since no agreements have been reached concerning the nature of access control information, a default version of the information (a single integer value) will be filled in by this function and the input parameters will be disregarded. In later versions, this function will be upgraded to allow for passing of actual access control information.

Parameters

access The integer value for access control.

Range of Values Integer values between 1 and 2(32)-1

Sample Values 22

msg_type CMIS operation type.

Range of Values
SET_REQ, GET_REQ, ACTION_REQ,
CREATE_REQ, DELETE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NO_MEM, FIELD_ALREADY_FILLED

5.10. fill_synchronization

```
int fill_synchronization( sync, msg_type, msg_ptr)
    int sync;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the synchronization field of the CMIP PDU. The *fill_synchronization()* routine checks the acceptability of the input parameters. If they are within range the function allocates the data structure to hold the synchronization information and sets the synchronization pointer in the CMIP operation structure to point to this newly allocated structure. Then the synchronization field is filled with the synchronization information (i.e., BESTEFFORT or ATOMIC) passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: If the function is not called, nothing will be put into the CMIP operation structure and upon receipt, this NULL field will indicate to the other CMIS provider that the default case of best effort should be used for synchronization.

Parameters

sync Either besteffort or atomic.

Range of Values BESTEFFORT or ATOMIC

msg_type CMIS operation type.

Range of Values

SYNC_NOT_SUPPORTED, COMPLEXITY_LIMITATION,
SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NOT_SUPPORTED_SYNC, NO_MEM, FIELD_ALREADY_FILLED,
NULL_MSG_PTR

5.11. fill_scope

```
int fill_scope( scope_type, scope_value, msg_type, msg_ptr)
    int scope_type;
    int scope_value;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the scope field of the CMIP PDU. The *fill_scope()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the scope information and sets the scope pointer in the CMIP operation structure to point to this newly allocated structure. Then the scope field is filled with the scope level information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: If this function is not called, nothing will be put into the CMIP operation structure and upon receipt, this NULL field will indicate to the peer CMIS provider that the default case of base object alone is to be used for scoping.

Parameters

scope_type Indicates type of scoping to be performed: baseObject (the base object alone), firstLevelOnly (the first level subordinates of the base object), wholeSubtree (the base object and all of its subordinates), individualLevels (the Nth level subordinates of the base object), or baseToNthLevel (the base object and all of its subordinates down to and including the Nth level.

Range of Values

One of: BASEOBJECT, FIRSTLEVELONLY, WHOLESUBTREE, INDIVIDUALLEVELS, or BASETONTHLEVEL.

scope_value Indicates the value for either the individualLevel that is to be scoped, or the Nth level to stop at when using baseToNthLevel scoping. If scoping is done with baseobject, firstlevelonly or wholeSubtree, this value should be set to 0.

Range of Values NULL, or integer value from 1 to 2(32) - 1.

msg_type CMIS operation type.

Range of Values

INVALID_SCOPE, COMPLEXITY_LIMITATION, SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, SCOPE_TYPE_OUT_OF_RANGE, SCOPE_VALUE_OUT_OF_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NO_MEM, FIELD_ALREADY_FILLED, NULL_MSG_PTR

5.12. fill_filter

```
int fill_filter(operator_type, item_type1, item_type2, not_flag1,
              not_flag2, id_type1, id_type2, id1, id2, att_val1,
                    att_val2, substring_type1, substring_type2, msg_type, msg_ptr)
    int        operator_type;
    int        item_type1;
    int        item_type2;
    int        not_flag1;
    int        not_flag2;
    int        id_type1;
    int        id_type2;
    union      id *id1;
    union      id *id2;
    PE           att_val1;
    PE           att_val2;
    int        substring_type1;
    int        substring_type2;
    int        msg_type;
    char       **msg_ptr;
```

Description

The *fill_filter()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the filter information and sets the filter pointer in the CMIP operation structure to point to this newly allocated structure. Then the filter field is filled with the filter information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

operator_type This parameter is limited to the following 5 different filter constructions: Not (item_type1 And item_type2) (NAND), Not (item_type1 Or item_type2) (NOR), item_type1 And item_type2 (AND), item_type1 Or item_type2 (OR), item_type1 (NULL).

Range of Values NAND, NOR, AND, OR, NULL

item_type1 This parameter specifies what the user would like to filter on for the first Item.

Range of Values
EQUALITY, GREATEROREQUAL, LESSOREQUAL, PRESENT, SUBSTRINGS, SUBSETOF, SUPERSETOF, NONNULLSETINTERSECTION

item_type2 This parameter specifies what the user would like to filter on for the second Item. Set to NULL if operator_type = NULL.

Range of Values
EQUALITY, GREATEROREQUAL, LESSOREQUAL, PRESENT, SUBSTRINGS, SUBSETOF, SUPERSETOF, NONNULLSETINTERSECTION

not_flag1 If the user chooses to Not the first Item then set this to TRUE.

Range of Values TRUE or FALSE

not_flag2 If the user chooses to Not the second Item then set this to TRUE.

Range of Values TRUE or FALSE. Set to FALSE if operator_type = NULL.

id_type1 Indicates whether attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id1 union. This is the OID type for item_type1.

Range of Values LOCAL or GLOBAL

id1 Either an integer that specifies the attribute identifier in local form, or a Character string containing the attribute identifier in global form. (see section 3.1.1 for explanation of treatment of object identifiers by these interface functions)

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 through 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string) This is the OID for item_type1.

id_type2 Same as parameter id_type1 except that this is the OID type for item_type2.

id2 Same as parameter id1 except that this is the OID for item_type2.

att_val1 Pointer to the PE containing the encoded information for item_type1.

att_val2 Pointer to the PE containing the encoded information for item_type2. Set this to NULL if operator_type = NULL.

substring_type1 If item_type1 is set to SUBSTRINGS, then the user must indicate (by setting this parameter) what part of the string they wish to filter on. Set to NULL if item_type1 does not equal SUBSTRINGS.

Range of Values INITIALSTRING, ANYSTRING, FINALSTRING

substring_type2 If item_type2 is set to SUBSTRINGS, then the user must indicate (by setting this parameter) what part of the string they wish to filter on. Set to NULL if item_type1 does not equal SUBSTRINGS or if operator_type equals NULL.

Range of Values INITIALSTRING, ANYSTRING, FINALSTRING

msg_type CMIS operation type.

Range of Values

INVALID_FILTER, COMPLEXITY_LIMITATION, SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, OPERATOR_TYPE_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, BAD_FORM, NO_MEM, ITEM_TYPE_RANGE, FIELD_ALREADY_FILLED

5.13. fill_modificationList

```
int fill_modificationList(modify_operation, id_type, id, att_value, msg_type, msg_ptr)
    int modify_operation;
    int id_type;
    union ID id;
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the modification list of the CMIP PDU. If the input paramaters are within range, the function allocates the data structure to hold the modification list information and sets the modification list pointer in the CMIP operation structure to point to this newly allocated structure. Then the modification list structure is filled with the modify_operation, attribute value and either the globalForm identifier or the localForm identifier. The attribute value is passed in by the user in the form of a presentation element (PE). It is the responsibility of the CMIS user to create the PE by calling the appropriate encode function for that attribute value. The function then returns with a SUCCESS indication. If more than one attribute is to be included in the modification list, the CMIS user should call the *fill_modificationList()* function one time for each of the attributes. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

modify_operation Indicates which of the four possible types of modification is to be performed on the attribute.

Range of Values int_CMIP_ModifyOperator_replace, int_CMIP_ModifyOperator_removeValues, int_CMIP_ModifyOperator_addValues, int_CMIP_ModifyOperator_setToDefault

id_type Indicates whether the attribute id, as passed in by the CMIS user, is in the local or global form. This parameter indicates which of the two forms represented by the id union is to be used.

Range of Values type_CMIP_ObjectClass_globalForm, type_CMIP_ObjectClass_localForm

id Either an integer that specifies the attribute identifier in local form or a Character string containing the identifier in global form.

Range of Values For global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.). For local form, an integer between 0 and 2^32 - 1.

Sample Values Global Form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.) Local Form = 35

att_value The encoded attribute value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values SET_REQ

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, INVALID_MSG_TYPE,
FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NO_MEM,
UNABLE_TO_COPY_PE

5.14. fill_attributeList

```
int fill_attributeList( id_type, id, att_value, msg_type, msg_ptr)
    int id_type;
    union ID id;
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the attribute list of the CMIP PDU. The *fill_attributeList()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the attribute list information and sets the attribute list pointer in the CMIP operation structure to point to this newly allocated structure. Then the attribute list structure is filled with the attribute value and either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The attribute value is passed in by the user in the form of a presentation element (PE). It is the responsibility of the CMIS user to create the PE by calling the appropriate encode function for that attribute value. The function then returns with a SUCCESS indication. If more than one attribute is to be included in the attribute list, the CMIS user should call the *fill_attributeList()* function one time for each of the attributes. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the attribute id, as passed in by the CMIS user, is in the local or global form. This parameter indicates which of the two forms represented by the id union is to be used.

Range of Values LOCAL or GLOBAL.

id Either an integer that specifies the attribute identifier in local form or a Character string containing the identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded attribute value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values

SET_REQ, SET_RSP, GET_RSP, CREATE_REQ, CREATE_RSP.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE,
FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NO_MEM

5.15. fill_attributeIdlist

```
int fill_attributeIdlist( id_type, id, msg_type, msg_ptr)
    int id_type;
    union ID id;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the attribute list of the CMIP PDU. The *fill_attributeIdlist()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the attribute ID list information and sets the attributeIdList pointer in the CMIP operation structure to point to this newly allocated structure. Then the attribute ID list structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function then returns with a SUCCESS indication. If more than one attribute id is to be included in the attribute list, the CMIS user should call the *fill_attributeIdlist()* function one time for each of the attribute ids. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the attribute id, as passed in by the CMIS user, is in the local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL.

id Either an integer that specifies the attribute identifier in local form or a character string containing the identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string)

msg_type CMIS operation type.

Range of Values GET_REQ,

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE,
FIELD_DOES_NOT_EXIST, NO_MEM

5.16. fill_currentTime

```
int fill_currentTime( currenttime, msg_type, msg_ptr)
    char *currenttime;
    int msg_type;
    char **msg_ptr;
```

Description

This function fills the currentTime field of the CMIP PDU. The *fill_currentTime()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the currentTime information and sets the currentTime pointer in the CMIP operation structure to point to this newly allocated structure. Then the currentTime field is filled with the currentTime information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

currenttime A string that represents the time at which the operation occurred.

Sample Values The string 19890613123012.333-0500 represents a local time of 12:30:12 (and 333 msecs) on 13th June 1989, in a time zone which is 5 hours behind GMT.

msg_type CMIS operation type.

Range of Values

GET_LIST_ERROR, SET_LIST_ERROR, SET_RSP,
GET_RSP, EVENT_REQ, EVENT_RSP,
ACTION_RSP, CREATE_RSP, DELETE_RSP.
DELETE_ERR, ACTION_ERR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST,
NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.17. fill_eventTime

```
int fill_eventTime( currenttime, msg_type, msg_ptr)
    char *currenttime;
    int  msg_type;
    char **msg_ptr;
```

Description

This function fills the eventTime field of the CMIP PDU. The *fill_eventTime()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the eventTime information and sets the eventTime pointer in the CMIP operation structure to point to this newly allocated structure. Then the eventTime field is filled with the eventTime information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error condition.

Parameters

currenttime A string that represents the time at which the event occurred.

Sample Values The string 19890613123012.333-0500 represents a local time of 12:30:12 (and 333 msecs) on 13th June 1989, in a time zone which is 5 hours behind GMT.

msg_type CMIS operation type.

Range of Values

GET_LIST_ERROR, SET_LIST_ERROR, SET_RSP,
GET_RSP, EVENT_REQ, EVENT_RSP,
ACTION_RSP, CREATE_RSP, DELETE_RSP.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST,
NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.18. fill_getInfoStatus

```
int fill_getInfoStatus( error_type, error_status, id_type, id, att_value, msg_type, msg_ptr)
    int error_type;
    int error_status;
    int id_type;
    union ID id;
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_getInfoStatus()* routine fills the CMIS field for the get info status. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the get info status designation and sets the GetInfoStatus pointer in the CMIP operation structure to point to this newly allocated structure. Then the get info status structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The *fill_getInfoStatus()* function should be called one time for each attribute that is included in a get list error. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

error_type Indicates whether an error occurred on this particular attribute.

Range of Values ERROR or NOERROR

error_status Error that occurred. If no error, set to NULL.

Range of Values STATUS_ACCESSDENIED, STATUS_NOSUCHATTRIBUTE, NULL

id_type Indicates whether the attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form or a character string containing the attribute identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value If, for this attribute, the GET operation was successful, this att_value is the encoded attribute value that is returned from the encode routine that the user calls. If, for this attribute, the GET operation was unsuccessful, this att_value should be set to NULL because there is no value that was sent.

msg_type CMIS operation type.

Range of Values GET_LIST_ERROR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, INVALID_ERROR_STATUS, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR

5.19. fill_setInfoStatus

```
int fill_setInfoStatus(error_type, error_status, modify_operation, id_type, id, att_value, msg_type, msg_ptr)
    int error_type;
    int error_status;
    int modify_operation;
    int id_type;
    union ID id;
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_setInfoStatus()* routine fills the CMIS field for the set info status. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the set info status designation and sets the SetInfoStatus pointer in the CMIP operation structure to point to this newly allocated structure. Then the set info status structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The *fill_setInfoStatus()* function should be called one time for each attribute that is included in a set list error. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

error_type Indicates whether an error occurred on this particular attribute.

Range of Values ERROR or NOERROR

error_status Error that occurred. If no error, set to NULL.

Range of Values

STATUS_ACCESSDENIED, STATUS_NOSUCHATTRIBUTE,
STATUS_INVALIDATTRIBUTEVALUE, NULL

modify_operation Specifies one of four ways the to operate on the specified attributes.

Range of Values int_CMIP_ModifyOperation_replace int_CMIP_ModifyOperation_removeValues
int_CMIP_ModifyOperation_addValues
int_CMIP_ModifyOperation_setToDefault

id_type Indicates whether the attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form or a character string containing the attribute identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5"

att_value If, for this attribute, the SET operation was successful, this att_value is the encoded attribute value that is returned from the encode routine that the user calls. If, for this attribute, the SET operation was unsuccessful, this att_value should be set to the value received in the SET request.

msg_type CMIS operation type.

Range of Values SET_LIST_ERROR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, INVALID_ERROR_STATUS, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR

5.20. fill_actionValue

```
int fill_actionValue( id_type, id, att_value, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    PE         att_value;
    int       msg_type;
    char     **msg_ptr;
```

Description

The *fill_actionValue()* routine fills the CMIS field for the action value. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the action value designation and sets the actionValue pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_ActionInfo structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the action ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action identifier in local form or a character string containing the action identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded action value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values ACTION_REQ, INVALID_ARGUMENT_VALUE.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NO_MEM, FIELD_ALREADY_FILLED

5.21. fill_eventValue

```
int fill_eventValue( id_type, id, att_value, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    PE        att_value;
    int       msg_type;
    char      **msg_ptr;
```

Description

The *fill_eventValue()* routine fills the CMIS field for the event value. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the event value designation and sets the eventValue pointer in the CMIP operation structure to point to this newly allocated structure. Then the event structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error condition.

Parameters

id_type Indicates whether the event ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the event identifier in local form or a character string containing the event identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded event value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values INVALID_ARGUMENT_VALUE.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.22. fill_attribute

```
int fill_attribute( id_type, id, att_value, msg_type, msg_ptr)
    int      id_type;
    union ID  id;
    PE        att_value;
    int      msg_type;
    char    **msg_ptr;
```

Description

The *fill_attribute()* routine fills the CMIS field for the attribute. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the attribute designation and sets the attribute pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_Attribute structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form or a character string containing the attribute identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded attribute value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values INVALID_ATTRIBUTE_VALUE.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE,
FIELD_DOES_NOT_EXIST, NO_MEM, FIELD_ALREADY_FILLED

5.23. fill_actionReply

```
int fill_actionReply( id_type, id, att_value, msg_type, msg_ptr)
    int        id_type;
    union ID  id;
    PE          att_value;
    int        msg_type;
    char     **msg_ptr;
```

Description

The *fill_actionReply()* routine fills the CMIS field for the action reply. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the action reply designation and sets the actionReply pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_ActionReply structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the actionreply ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the actionreply identifier in local form or a character string containing the actionreply identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded actionreply (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values ACTION_RSP.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.24. fill_actionInfo

```
int fill_actionInfo( id_type, id, att_value, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    PE        att_value;
    int       msg_type;
    char      **msg_ptr;
```

Description

The *fill_actionInfo()* routine fills the CMIS field for the action info. This function checks the acceptability of the input parameters. If they are all within range, it allocates the data structure to hold the action info designation and sets the actionInfo pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_ActionInfo structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the actioninfo ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the the actioninfo identifier in local form or a character string containing the the actioninfo identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded the actioninfo (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values ACTION_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.25. fill_eventReply

```
int fill_eventReply( id_type, id, att_value, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    PE         att_value;
    int       msg_type;
    char    **msg_ptr;
```

Description

The *fill_eventReply()* routine fills the CMIS field for the event reply. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the event reply designation and sets the eventReply pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_EventReply structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. Also filled in this structure is the att_value. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the eventreply ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the eventreply identifier in local form or a character string containing the eventreply identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

att_value The encoded eventreply (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values EVENT_RSP.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.26. fill_eventType

```
int fill_eventType( id_type, id, msg_type, msg_ptr)
    int        id_type;
    union ID   id;
    int        msg_type;
    char       **msg_ptr;
```

Description

The *fill_eventType()* routine fills the CMIS field for the event Type. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the event Type designation and sets the eventType pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_EventTypeId structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether eventtype ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the eventtype identifier in local form or a character string containing the eventtype identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

msg_type CMIS operation type.

Range of Values EVENT_REQ

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.27. fill_actionType

```
int fill_actionType( id_type, id, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    int       msg_type;
    char      **msg_ptr;
```

Description

The *fill_actionType()* routine fills the CMIS field for the action Type. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the action Type designation and sets the actionType pointer in the CMIP operation structure to point to this newly allocated structure. Then the type_CMIP_ActionTypeId structure is filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether actiontype ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the actiontype identifier in local form or a character string containing the actiontype identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

msg_type CMIS operation type.

Range of Values NO_SUCH_ACTION

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NO_MEM, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, FIELD_ALREADY_FILLED

5.28. fill_eventId

```
int fill_eventId( id_type, id, event_type, event, msg_type, msg_ptr)
    int       id_type;
    union ID  id;
    int       event_type;
    union ID  event;
    int       msg_type;
    char      **msg_ptr;
```

Description

The *fill_eventId()* routine fills the CMIS field for the event Id. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the event Id designation and sets the eventId pointer in the CMIP operation structure to point to this newly allocated structure. Then both the managedObjectClass and eventType in the structure are filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether eventtype ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the eventtype identifier in local form or a character string containing the the managed object class identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

event_type Same as id_type except it specifies the eventid designation.

event Same as id except it specifies the eventid designation.

msg_type CMIS operation type.

Range of Values NO_SUCH_ARGUMENT.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NULL_MSG_PTR, NO_MEM, FIELD_ALREADY_FILLED

5.29. fill_actionId

```
int fill_actionId( id_type, id, action_type, action, msg_type, msg_ptr)
    int        id_type;
    union ID   id;
    int        action_type;
    union ID   action;
    int        msg_type;
    char       **msg_ptr;
```

Description

The *fill_actionId()* routine fills the CMIS field for the action Id. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the action Id designation and sets the actionId pointer in the CMIP operation structure to point to this newly allocated structure. Then both the managedObjectClass and actionType in the structure are filled with either the localForm or globalForm identifier, depending on which one was passed in by the CMIS user. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the managed object class ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the id union.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the managed object class identifier in local form or a character string containing the managed object class identifier in global form.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string.)

action_type Same as id_type except it specifies the actionid designation.

action Same as id except it specifies the actionid designation.

msg_type CMIS operation type.

Range of Values NO_SUCH_ARGUMENT.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, BAD_FORM, GLOB_RANGE, NO_SUCH_MSG_TYPE, NULL_MSG_PTR, NO_MEM, FIELD_ALREADY_FILLED

5.30. fill_eventInfo

```
int fill_eventInfo( att_value, msg_type, msg_ptr)
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_eventInfo()* routine sets the CMIS attribute value field to point to the presentation element (PE) passed in as att_value. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the PE and sets the eventInfo pointer in the CMIP operation structure to point to this newly allocated structure. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

att_value The encoded event value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values EVENT_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NULL_MSG_PTR, NULL

5.31. fill_specificErrorInfo

```
int fill_specificErrorInfo( att_value, msg_type, msg_ptr)
    PE att_value;
    int msg_type;
    char **msg_ptr;
```

Description

The *fill_specificErrorInfo()* routine sets the CMIS attribute value field to point to the presentation element (PE) passed in as att_value. This function checks the acceptability of the input parameters. If they are all within range, the function allocates the data structure to hold the PE and sets the specificErrorInfo pointer in the CMIP operation structure to point to this newly allocated structure. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

att_value The encoded specific error info value (in the form of a PE) as returned from the encode routine that the user calls.

msg_type CMIS operation type.

Range of Values PROCESSING_FAILURE.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR

5.32. fill_actionErrorInfo

```
int fill_actionErrorInfo(error_status, id_type1, ID1, id_type2,
                         ID2, att_value, msg_type, msg_ptr)
        int         error_status;
        int         id_type1;
        union id  ID1;
        int         id_type2;
        union id  ID2;
        PE           att_value;
        int         msg_type;
        char **msg_ptr;
```

Description

The *fill_actionErrorInfo()* routine checks the acceptability of the input parameters. If they are within range, the function allocates the data structure to hold the actionErrorInfo information and sets the msg_ptr pointer in the CMIP operation structure to point to this newly allocated structure. Then the actionErrorInfo structure is filled with the actionErrorInfo information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

errorStatus The user should set this parameter to the appropriate error they want to send.

Range of Values ACCESSDENIED, NO_SUCH_ACTION, NO_SUCH_ARGUMENT, INVALID_ARGUMENT_VALUE

id_type1 Indicates whether attribute ID designation, as passed in by the CMIS user, is in local or global form. This parameter is used by the function to determine what type to use in working with the ID1 union.

Range of Values LOCAL or GLOBAL

ID1 Either an integer that specifies the attribute identifier in local form, or a Character string containing the attribute identifier in global form. (see section 3.1.1 for explanation of treatment of object identifiers by these interface functions)

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 through 39 if the first element is 0 or 1. Subsequent values must be non-negative numbers and each value is separated by a dot(.).

Sample Values local_form = 35, global_form = "1.17.244.5" (The quotes in the global_form signify that it is a character string)

id_type2 Same as parameter id_type1.

ID2 Same as parameter id1 except.

att_value Pointer to the PE containing the encoded information.

msg_type CMIS operation type.

Range of Values INVALID_FILTER, COMPLEXITY_LIMITATION, SET_REQ, GET_REQ, ACTION_REQ, DELETE_REQ.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

Return Values

SUCCESS, GLOBE_RANGE, NO_SUCH_MSG_TYPE, NULL_MSG_PTR, BAD_FORM, NO_MEM INVALID_ERROR_STATUS.

5.33. fill_deleteErrorInfo

```
int fill_deleteErrorInfo(msg_type, msg_ptr, deleteErrorInfo)
          int    msg_type;
          char   **msg_ptr;
          int    deleteErrorInfo;
```

Description

This function fills the deleteErrorInfo field of the CMIP PDU. The *fill_deleteErrorInfo()* routine checks the acceptability of the input parameters. If they are within range, the deleteErrorInfo field is filled with the deleteErrorInfo information passed in by the CMIS user. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

msg_type CMIS operation type.

Range of Values DELETE_ERR, ACTION_ERR.

msg_ptr Pointer to the CMIP message containing all information for this CMIS operation.

deleteErrorInfo The integer value for deleteErrorInfo.

Range of Values Integer values between 1 and 2(32)-1

Return Values SUCCESS, NO_SUCH_MSG_TYPE

6. Extract Functions

This section provides a description of all the parameter extract functions contained in the CMIS interface. These functions are contained in "cmislib.a". The descriptions that follow contain descriptive overviews, input and output parameters, and parameter value and ranges, where appropriate. The following two structures are used extensively throughout the fill and extract routines, and are defined here for brevity.

```
union ID
{
  int       local_Form;
  char     *global_Form;
};

union Instance
{
  struct distinguishedName
  {
     char  *type;
     PE    value;
     int   RDN_flag;
  } DistinguishedName;
  struct qbuf *nonSpecificForm;
};
```

6.1. free_operation_struct

```
int free_operation_struct(msg_type, msg_ptr)
        int msg_type;
        char **msg_ptr;
```

Description

The *free_operation_struct()* routine frees the entire data structure used for sending/receiving CMIP messages (in particular, requests, responses, and errors). Since different CMIP operations require different information, and therefore different data structures, this function calls the appropriate ISODE free routine based on the operation type indicated by the value of the input parameter "msg_type".

Parameters

msg_type CMIS operation type

Range of Values

NO_SUCH_OBJECT_CLASS, NO_SUCH_OBJECT_INSTANCE, ACCESS_DENIED, SYNC_NOT_SUPPORTED, INVALID_FILTER, NO_SUCH_ATTRIBUTE, INVALID_ATTRIBUTE_VALUE, GET_LIST_ERROR, SET_LIST_ERROR, NO_SUCH_ACTION, PROCESSING_FAILURE, DUPLICATE_MANAGED_OBJECT_INSTANCE, NO_SUCH_REFERENCE_OBJECT, NO_SUCH_EVENT_TYPE, NO_SUCH_ARGUMENT, INVALID_ARGUMENT_VALUE, INVALID_SCOPE, INVALID_OBJECT_INSTANCE, MISSING_ATTRIBUTE_VALUE, CLASS_INSTANCE_CONFLICT, COMPLEXITY_LIMITATION, SET_REQ, SET_RSP, GET_REQ, GET_RSP, EVENT_REQ, EVENT_RSP, ACTION_REQ, ACTION_RSP, CREATE_REQ, CREATE_RSP, DELETE_REQ, DELETE_RSP.

msg_ptr This function frees the entire structure pointed to by msg_ptr.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, NULL_MSG_PTR

6.2. extract_baseManagedObjectClass

```
int extract_baseManagedObjectClass( id_type, id, msg_type, msg_ptr)
        int      *id_type;
        union ID *id;
        int      msg_type;
        char     *msg_ptr;
```

Description

The *extract_baseManagedObjectClass()* routine retrieves the CMIS field for the basemanagedObjectClass identifier from the particular CMIS message indicated by the msg_ptr parameter. Since the managed object class identifier can be represented in either a local or global form, this function retrieves the base managed object class identifier in the appropriate form as sent by the peer CMIS user, and indicates in which form the identifier is provided. The function normally returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the Managed Object Class designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the Managed Object Class identifier in local form, or a character string containing the object class identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND, CLASS_INSTANCE_CONFLICT

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, NO_SUCH_MSG_TYPE,
BAD_FORM, NULL_MO_PTR

6.3. extract_managedObjectClass

```
int extract_managedObjectClass( id_type, id, msg_type, msg_ptr)
    int       *id_type;
    union ID *id;
    int msg_type;
    char *msg_ptr;
```

Description

The purpose of the *extract_managedObjectClass()* routine is to extract the CMIS field for the basemanagedObjectClass identifier. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error condition.

Parameters

id_type Indicates whether the Managed Object Class designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the Managed Object Class identifier in local form, or a character string containing the object class identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

CREATE_REQ, CREATE_IND, CREATE_RSP, CREATE_CNF, SET_RSP,
SET_CNF, GET_RSP, GET_CNF, ACTION_CNF, ACTION_RSP,
DELETE_RSP, DELETE_CNF, EVENT_REQ, EVENT_IND, EVENT_RSP,
EVENT_CNF, NO_SUCH_OBJECT_CLASS, GET_LIST_ERROR, SET_LIST_ERROR,
NO_SUCH_ACTION, PROCESSING_FAILURE, NO_SUCH_EVENT_TYPE, DELETE_ERR, ACTIO

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, NO_SUCH_MSG_TYPE

6.4. extract_baseManagedObjectInstance

```
int  extract_baseManagedObjectInstance (instance_type, instance, msg_type, msg_ptr)
   int            *instance_type;
   union Instance *instance;
   int             msg_type;
   char           *msg_ptr;
```

Description

The *extract_baseManagedObjectInstance()* routine retrieves the base managed object instance name from the CMIS message indicated by the msg_ptr argument. Since the base managed object instance can be represented in three different formats (distinguishedName, nonSpecificForm or localDistinguishedName), this function retrieves the base managed object instance in the appropriate form as sent by the peer CMIS user, and indicates in which form the name is provided. The function then returns one of three values: MORE_RDN indicates more relative distinguished names exist; MORE_AVA indicates more attribute value assertions exist; and NO_MORE_RDN signifies the name is complete. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

instance_type Specifies which name form is used (distinguishedName, nonSpecificForm, or localDistinguishedName).

Range of Values DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME

instance This union contains the necessary information to represent the instance name as either the distinguishedname, localdistinguishedname, or nonspecificform, based on the above type.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND, CLASS_INSTANCE_CONFLICT

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NULL_MSG_PTR, NO_SUCH_MSG_TYPE, BAD_FORM,
NO_DISTINGUISHED_NAME, MORE_AVA, MORE_RDN, NO_MORE_RDN

6.5. extract_managedObjectInstance

```
    int extract_managedObjectInstance ( instance_type, instance, msg_type, msg_ptr)
        int             *instance_type;
        union Instance *instance;
        int             msg_type;
        char            *msg_ptr;
```

Description

The *extract_managedObjectInstance()* routine retrieves the managed object instance name from the CMIS message indicated by the msg_ptr argument. Since the managed object instance can be represented in three different formats (distinguishedName, nonSpecificForm or localDistinguishedName), this function retrieves the managed object instance in the appropriate form as sent by the peer CMIS user, and indicates in which form the name is provided. The function then returns one of three values: MORE_RDN indicates more relative distinguished names exist; MORE_AVA indicates more attribute value assertions exist; and NO_MORE_RDN signifies the name is complete. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

instance_type Specifies which name form is used (distinguishedName, nonSpecificForm, or localDistinguishedName).

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME

instance This union contains the necessary information to represent the instance name as either the distinguishedname, localdistinguishedname, or nonspecificform, based on the above type.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

CREATE_REQ, CREATE_IND, CREATE_RSP, CREATE_CNF, SET_RSP,
SET_CNF, GET_RSP, GET_CNF, ACTION_CNF, ACTION_RSP,
DELETE_RSP, DELETE_CNF, EVENT_REQ, EVENT_IND, EVENT_RSP,
EVENT_CNF, GET_LIST_ERROR, SET_LIST_ERROR,
NO_SUCH_OBJECT_INSTANCE, NO_SUCH_REFERENCE_OBJECT,
INVALID_OBJECT_INSTANCE, PROCESSING_FAILURE,
DUPLICATE_MANAGED_OBJECT_INSTANCE, DELETE_ERR, ACTION_ERR.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NULL_MSG_PTR, NO_SUCH_MSG_TYPE, BAD_FORM,
NO_DISTINGUISHED_NAME, MORE_AVA, MORE_RDN, NO_MORE_RDN

6.6. extract_referenceObjectInstance

```
int extract_referenceObjectInstance ( instance_type, instance, msg_type, msg_ptr)
    int          *instance_type;
    union  Instance *instance;
    int           msg_type;
    char          *msg_ptr;
```

Description

The *extract_referenceObjectInstance()* routine retrieves the reference object instance name from the CMIS message indicated by the msg_ptr argument. Since the reference object instance can be represented in three different formats (distinguishedName, nonSpecificForm or localDistinguishedName), this function retrieves the reference object instance in the appropriate form as sent by the peer CMIS user, and indicates in which form the name is provided. The function then returns one of three values: MORE_RDN indicates more relative distinguished names exist; MORE_AVA indicates more attribute value assertions exist; and NO_MORE_RDN signifies the name is complete. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

instance_type Specifies which name form is used (distinguishedName, nonSpecificForm, or localDistinguishedName).

Range of Values

DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHEDNAME

instance This union contains the necessary information to represent the instance name as either the distinguishedname, localdistinguishedname, or nonspecificform, based on the above type.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

CREATE_REQ, CREATE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NULL_MSG_PTR, NO_SUCH_MSG_TYPE, BAD_FORM,
NO_DISTINGUISHED_NAME, MORE_AVA, MORE_RDN, NO_MORE_RDN

6.7. extract_createObjectInstance

```
int   extract_createObjectInstance (object_type, instance_type, instance, msg_type, msg_ptr)
  int              *object_type;
  int              *instance_type;
  union Instance   *instance;
  int               msg_type;
  char             *msg_ptr;
```

Description

The purpose of the *extract_createObjectInstance()* routine is to retrieve the CMIP PDU field for the managed object instance name in the Create Argument structure. The object_type parameter indicates whether the name is a superior object instance (choice_CMIP_1_superiorObjectInstance), or a managed object instance (NULL). Since the superior object instance or managed object instance can be represented in three different formats (distinguishedName, nonSpecificForm or localDistinguishedName), this function retrieves the object instance in the appropriate form as sent by the peer CMIS user, and indicates in which form the name is provided. The function then returns one of three values: MORE_RDN indicates more relative distinguished names exist; MORE_AVA indicates more attribute value assertions exist; and NO_MORE_RDN signifies the name is complete. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

object_type This parameter indicates if the name represents a superior object instance, or a managed object instance.

Range of Values choice_CMIP_1_superiorObjectInstance, NULL

instance_type Specifies which name form is used (distinguishedName, nonSpecificForm, or localDistinguishedName).

Range of Values DISTINGUISHEDNAME, NONSPECIFICFORM, or LOCALDISTINGUISHED-NAME

instance This union contains the necessary information to represent the instance name as either the distinguishedname, localdistinguishedname, or nonspecificform, based on the above type.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values CREATE_REQ, CREATE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST, NULL_MSG_PTR,
NO_DISTINGUISHED_NAME, MORE_AVA, MORE_RDN, NO_MORE_RDN

6.8. extract_currentTime

```
int extract_currentTime (time, msg_type, msg_ptr)
   char    *time;
   int     msg_type;
   char    *msg_ptr;
```

Description

This function extracts the currentTime field of the CMIP PDU. The currentTime field is returned in the first parameter of this function. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

time A string that represents the time at which an operation occurred.

Sample Values The string 19890613123012.333-0500 represents a local time of 12:30:12 (and 333 msecs) on 13th June 1989, in a time zone which is 5 hours behind GMT.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

GET_LIST_ERROR, SET_LIST_ERROR, SET_RSP, SET_CNF, GET_RSP, GET_CNF, EVENT_REQ, EVENT_IND, EVENT_RSP, EVENT_CNF, ACTION_RSP, ACTION_CNF, CREATE_RSP, CREATE_CNF, DELETE_RSP, DELETE_CNF, DELETE_ERR, ACTION_ERR.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE

6.9. extract_modificationlist

```
int extract_modificationlist(modify_operation, id_type, id, att_value, msg_type, msg_ptr)
    int        *modify_operation;
    int        *id_type;
    union ID   *id;
    PE         *att_value;
    int        msg_type;
    char       *msg_ptr;
```

Description

The *extract_modificationlist()* function extracts individual attribute information from the modification list from a CMIP M-SET PDU. The *extract_modificationlist()* function should be called one time for each attribute that is to be extracted from the modification list. The value of the attribute is assigned to the attribute parameter pointed to by att_value. This value is still in the form of an encoded presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value. The function then returns with the number of attributes remaining in the list to be extracted. If NULL(0) is returned, no attributes remain to be extracted. Attributes are extracted from the tail end of the list, reducing the list size by one after each extraction.

Parameters

modify_operation Indicates the type of modify operation the sender wishes the receiver to perform with the object attribute.

Range of Values int_CMIP_ModifyOperator_replace, int_CMIP_ModifyOperator_removeValues, int_CMIP_ModifyOperator_addValues, int_CMIP_ModifyOperator_setToDefault

id_type Indicates whether the attribute designation is in local or global form.

Range of Values type_CMIP_ObjectClass_globalForm, type_CMIP_ObjectClass_localForm

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded attribute value.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values SET_REQ, SET_IND

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, INVALID_MSG_TYPE, NULL_MOD_LIST_PTR, NULL_MO_PTR, BAD_FORM, UNABLE_TO_COPY_PE

6.10. extract_attributeList

```
int extract_attributeList( id_type, id, att_value, msg_type, msg_ptr)
    int        *id_type;
    union ID   *id;
    PE         *att_value;
    int        msg_type;
    char       *msg_ptr;
```

Description

The *extract_attributeList()* function extracts individual attribute information from the attribute list from a CMIP PDU. The *extract_attributeList()* function should be called one time for each attribute that is to be extracted from the attribute list. The value of the attribute is assigned to the attribute parameter pointed to by att_value. This value is still in the form of an encoded presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value. The function then returns with the number of attributes remaining in the list to be extracted. If NULL(0) is returned, no attributes remain to be extracted. Attributes are extracted from the tail end of the list, reducing the list size by one after each extraction.

Parameters

id_type Indicates whether the attribute designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded attribute value.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

SET_REQ, SET_IND, SET_RSP, SET_CNF, GET_RSP, GET_CNF,
CREATE_REQ, CREATE_IND, CREATE_RSP, CREATE_CNF.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE,
NULL_ATT_LIST_PTR, NULL_MO_PTR, BAD_FORM

6.11. extract_accessControl

```
int extract_accessControl(access, msg_type, msg_ptr)
    int *access;
    int msg_type;
    char *msg_ptr;
```

Description

This function retrieves the information contained in the access control field of the CMIP PDU. The *extract_accessControl()* routine fills in the access parameter with the access control information retrieved from the message. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: For this version of the implementation, since no agreements have been reached concerning the nature of access control information, a default version of the information (a single integer value) will be retrieved by this function. In later versions, this function will be upgraded to allow for passing of actual access control information.

Parameters

access The integer value for access control.

Range of Values Integer values between 1 and 2(32)-1

Sample Values 22

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, CREATE_REQ, CREATE_IND, DELETE_REQ, DELETE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, NULL_ACCESS_PTR

6.12. extract_synchronization

```
int extract_synchronization( sync, msg_type, msg_ptr)
    int *sync;
    int msg_type;
    char *msg_ptr;
```

Description

This function retrieves the information contained in the synchronization field of the CMIP PDU. The *extract_synchronization()* function fills the sync parameter with the synchronization information contained in the message. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: If the function detects a NULL in this field, it will return the synchronization default value of best effort.

Parameters

sync Either besteffort or atomic.

Range of Values BESTEFFORT or ATOMIC

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

SYNC_NOT_SUPPORTED, COMPLEXITY_LIMITATION, SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NOT_SUPPORTED_SYNC

6.13. extract_scope

```
int extract_scope( scope_type, scope_value, msg_type, msg_ptr)
    int *scope_type;
    int *scope_value;
    int msg_type;
    char *msg_ptr;
```

Description

This function retrieves the information contained in the scope field of the CMIP PDU. The *extract_scope()* routine retrieves the scope level information contained in the CMIP message, and places it in the scope_type and scope_value parameters. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

NOTE: If the function detects a NULL in this field, it will return the scope default value, base object.

Parameters

scope_type Represents the type of scoping specified: baseObject, firstLevelOnly, wholeSubtree, individualLevels, or baseToNthLevel.

Range of Values BASEOBJECT, FIRSTLEVELONLY, WHOLESUBTREE, INDIVIDUALLEVELS, or BASETONTHLEVEL.

scope_value If scoping is done with baseobject, firstlevelonly or wholeSubtree, this value should be set to NULL. Otherwise, if either individualLevel or baseToNthlevel is to be scoped, this value should be a positive integer in the specific range.

Range of Values NULL, or integer value from 1 to 2(32) - 1.

msg_type Type of CMIS service message from which the parameter information is to be extracted. data structure so that it can retrieve the desired information correctly.

Range of Values

INVALID_SCOPE, COMPLEXITY_LIMITATION, SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE

6.14. extract_filter

```
int extract_filter(operator_type, item_type1, item_type2, id_type1, id_type2,
                          not_flag1, not_flag2, id1, id2, substring_type1,
                    substring_type2, att_val1, att_val2, msg_type, msg_ptr)
   int        *operator_type;
   int        *item_type1;
   int        *item_type2;
   int        *id_type1;
   int        *id_type2;
   int        *not_flag1;
   int        *not_flag2;
   union      id *id1;
   union      id *id2;
   int        *substring_type1;
   int        *substring_type2;
   PE          *att_val1;
   PE          *att_val2;
   int         msg_type;
   char        *msg_ptr;
```

Description

The *extract_filter()* routine extracts the filter field of the CMIS message. Upon returning from this function, all of the parameters passed to the function have been filled in with the filter information(except for msg_type and msg_ptr which are inputs to the *extract_filter()* routine). Some of the parameters may have been set to NULL depending on the value of the operator_type parameter. If the value of the parameter "operator_type" is NULL after you make the *extract_filter()* call then all parameters ending with the number "2", such as item_type2, will be NULL. The information contained in each parameter is described in the parameters section below. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

operator_type This parameter will be one of the following 5 different filter constructions: Not (item_type1 And item_type2) (NAND), Not (item_type1 Or item_type2) (NOR), item_type1 And item_type2 (AND), item_type1 Or item_type2 (OR), item_type1 (NULL).

Range of Values NAND, NOR, AND, OR, NULL

item_type1 This parameter shows what to check for, in the filter, for the first Item.

Range of Values
EQUALITY, GREATEROREQUAL, LESSOREQUAL, PRESENT, SUBSTRINGS, SUBSETOF, SUPERSETOF, NONNULLSETINTERSECTION

item_type2 This parameter shows what to check for, in the filter, for the second Item.

Range of Values
EQUALITY, GREATEROREQUAL, LESSOREQUAL, PRESENT, SUBSTRINGS, SUBSETOF, SUPERSETOF, NONNULLSETINTERSECTION

not_flag1 If this parameter is TRUE then this indicates NOT item_type1.

Range of Values TRUE or FALSE

not_flag2 If this parameter is TRUE then this indicates NOT item_type2.

Range of Values TRUE or FALSE.

id_type1 Indicates whether the attribute ID designation, as contained in the message, is in local or global form.

Range of Values LOCAL or GLOBAL

id1 Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

id_type2 Same as parameter id_type1 except that this is the OID type for item_type2.

id2 Same as parameter id1 except that this is the OID for item_type2.

att_val1 This parameter contains the encoded portion of the attribute in the form of a presentation element (PE) for item_type1. It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value.

att_val2 Same as att_val1.

substring_type1 If item_type1 is set to SUBSTRINGS, then this parameter indicates what part of the string to apply the filter to for item_type1. This parameter will be NULL if item_type1 does not equal SUBSTRINGS.

Range of Values INITIALSTRING, ANYSTRING, FINALSTRING

substring_type2 If item_type2 is set to SUBSTRINGS, then this parameter indicates what part of the string to apply the filter to for item_type2. This parameter will be NULL if item_type2 does not equal SUBSTRINGS or if operator_type equals NULL.

Range of Values INITIALSTRING, ANYSTRING, FINALSTRING

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

INVALID_FILTER, COMPLEXITY_LIMITATION, SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, BAD_FORM, NULL_MO_PTR, INVALID_FILTER, OPERATOR_TYPE_RANGE

6.15. extract_attribute

```
int extract_attribute( id_type, id, att_value, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    PE       *att_value;
    int      msg_type;
    char     *msg_ptr;
```

Description

The *extract_attribute()* routine extracts the attribute field of the CMIS message. Upon returning from this function, the id parameter contains the attribute identifier either in localForm or globalForm, depending on which one was contained in the message. Also returned is the att_value, containing the encoded portion of the attribute in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the attribute ID designation, as contained in the message, is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded attribute value.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values INVALID_ATTRIBUTE_VALUE.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.16. extract_attributeId

```
int extract_attributeId( id_type, id, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    int       msg_type;
    char     *msg_ptr;
```

Description

The purpose of the *extract_attributeId()* routine is to extract the attribute identifier field from the CMIS message. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the attribute ID designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values NO_SUCH_ATTRIBUTE, MISSING_ATTRIBUTE_VALUE. Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, NULL_ATT_LIST_PTR, NULL_MO_PTR, BAD_FORM

6.17. extract_attributeIdlist

```
int extract_attributeIdlist( id_type, id, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    int       msg_type;
    char     *msg_ptr;
```

Description

The *extract_attributeIdlist()* function extracts individual attribute IDs from the attribute ID list contained in a CMIP PDU. The *extract_attributeIdlist()* function should be called one time for each attribute ID that is to be extracted from the attribute ID list. The function then returns with the number of attribute IDs remaining in the list to be extracted. If NULL(0) is returned, no attribute IDs remain to be extracted. Attribute IDs are extracted from the tail end of the list, reducing the list size by one after each extraction.

Parameters

id_type Indicates whether the attribute ID designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values GET_REQ, GET_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, NULL_ATT_LIST_PTR, NULL_MO_PTR, BAD_FORM

6.18. extract_setInfoStatus

```
int extract_setInfoStatus(error_type,error_status, modify_operation, id_type, id, att_value, msg_type, msg_ptr)
     int *error_type;
     int *error_status;
        int modify_operation;
     int *id_type;
     union ID *id;
     PE *att_value;
     int msg_type;
     char *msg_ptr;
```

Description

The *extract_setInfoStatus()* routine retrieves set info status information from the CMIS message. This function should be called one time for each attribute that is to be extracted from the attribute list. The value of the attribute is assigned to the attribute parameter pointed to by att_value. This value is still in the form of an encoded presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value. The function then returns with the number of attributes remaining in the list to be extracted. If NULL(0) is returned, no attributes remain to be extracted. Attributes are extracted from the tail end of the list, reducing the list size by one after each extraction.

Parameters

error_type Indicates whether an error occurred on this particular attribute.

Range of Values ERROR or NOERROR

error_status Error that occurred. If no error, it will be set to NULL.

Range of Values
STATUS_ACCESSDENIED, STATUS_NOSUCHATTRIBUTE,
STATUS_INVALIDATTRIBUTEVALUE, NULL

modify_operation Specifies one of four ways the to operate on the specified attributes.

Range of Values int_CMIP_ModifyOperation_replace int_CMIP_ModifyOperation_removeValues
int_CMIP_ModifyOperation_addValues
int_CMIP_ModifyOperation_setToDefault

id_type Indicates whether the attribute designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE that will contain either the original attribute information if the operation failed for this attribute, or the new attribute information if the operation succeeded for this attribute.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values SET_LIST_ERROR.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, INVALID_ERROR_STATUS, NULL_ATT_LIST_PTR, NULL_MO_PTR, BAD_FORM

6.19. extract_getInfoStatus

```
int extract_getInfoStatus( error_type, error_status, id_type, id, att_value, msg_type, msg_ptr)
    int      *error_type;
    int      *error_status;
    int      *id_type;
    union ID *id;
    PE        *att_value;
    int       msg_type;
    char     *msg_ptr;
```

Description

The *extract_getInfoStatus()* routine retrieves get info status information from the CMIS message. This function should be called one time for each attribute that is to be extracted from the attribute list. The value of the attribute is assigned to the attribute parameter pointed to by att_value. This value is still in the form of an encoded presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode function for this attribute value. The function then returns with the number of attributes remaining in the list to be extracted. If NULL(0) is returned, no attributes remain to be extracted. Attributes are extracted from the tail end of the list, reducing the list size by one after each extraction.

Parameters

error_type Indicates whether an error occurred on this particular attribute.

Range of Values ERROR or NOERROR

error_status Error that occurred. If no error, this will be set to NULL.

Range of Values STATUS_ACCESSDENIED, STATUS_NOSUCHATTRIBUTE, NULL

id_type Indicates whether the attribute designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE that will contain either the original attribute information if the operation failed for this attribute, or the new attribute information if the operation succeeded for this attribute.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values GET_LIST_ERROR.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE, NULL_ATT_LIST_PTR, NULL_MO_PTR, BAD_FORM, INVALID_ERROR_STATUS

6.20. extract_actionInfo

```
int extract_actionInfo (id_type, id, att_value, msg_type, msg_ptr)
  int      *id_type;
  union ID *id;
  PE       *att_value;
  int      msg_type;
  char     *msg_ptr;
```

Description

The *extract_actionInfo()* routine extracts the action info field of the CMIS message. The action information is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this action information. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the action designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action identifier in local form, or a character string containing the action identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded action information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values ACTION_REQ, ACTION_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.21. extract_actionReply

```
int extract_actionReply (id_type, ID, att_value, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    PE       *att_value;
    int      msg_type;
    char     *msg_ptr;
```

Description

The *extract_actionReply()* routine extracts the action reply field of the CMIS message. The action reply is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this action reply. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the action designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action identifier in local form, or a character string containing the action identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded action reply information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values ACTION_RSP, ACTION_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.22. extract_actionType

```
int extract_actionType (id_type, id, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    int      msg_type;
    char     *msg_ptr;
```

Description

The *extract_actionType()* routine extracts the action type field of the CMIS message. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether action type designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action identifier in local form, or a character string containing the action identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values NO_SUCH_ACTION

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.23. extract_eventReply

```
int extract_eventReply( id_type, id, att_value, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    PE       *att_value;
    int      *msg_type;
    char     *msg_ptr;
```

Description

The *extract_eventReply()* routine extracts the event reply field of the CMIS message. The event reply is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this event reply. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the event designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the event identifier in local form, or a character string containing the event identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded event reply information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values EVENT_RSP, EVENT_CNF.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.24. extract_eventType

```
int extract_eventType( id_type, id, att_value, msg_type, msg_ptr)
    int      *id_type;
    union ID *id;
    PE       *att_value;
    int      *msg_type;
    char     *msg_ptr;
```

Description

The *extract_eventType()* routine extracts the eventType field of the CMIS message. The event type is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this event type. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

id_type Indicates whether the event designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the event identifier in local form, or a character string containing the event identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded event type information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values EVENT_REQ, EVENT_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST,
NO_SUCH_MSG_TYPE, NULL_MO_PTR, BAD_FORM

6.25. extract_eventInfo

```
int extract_eventInfo (att_value, msg_type, msg_ptr)
PE      *att_value;
int     msg_type;
char    *msg_ptr;
```

Description

This *extract_eventInfo()* routine extracts the eventTime field of the CMIP PDU. The event information is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this event information. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

att_value Pointer to the PE containing the encoded event information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values EVENT_REQ, EVENT_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, NULL_MSG_PTR

6.26. extract_eventTime

```
int extract_eventTime( currenttime, msg_type, msg_ptr)
    int *currenttime;
    int msg_type;
    char *msg_ptr;
```

Description

This *extract_eventTime()* routine extracts the eventTime field of the CMIP PDU. The eventTime field is returned in the first parameter of this function. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

currenttime A string that represents the current time that the operation occurred.

Sample Values The string 19890613123012.333-0500 represents a local time of 12:30:12 (and 333 msecs) on 13th June 1989, in a time zone which is 5 hours behind GMT.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values

GET_LIST_ERROR, SET_LIST_ERROR, SET_RSP, SET_CNF, GET_RSP, GET_CNF, EVENT_REQ, EVENT_IND, EVENT_RSP, EVENT_CNF, ACTION_RSP, ACTION_CNF, CREATE_RSP, CREATE_CNF, DELETE_RSP, DELETE_CNF.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE

6.27. extract_specificErrorInfo

```
int extract_specificErrorInfo( att_value, msg_type, msg_ptr)
    PE *att_value;
    int msg_type;
    char *msg_ptr;
```

Description

The *extract_specificErrorInfo()* routine extracts the specific error information from the CMIS message. The specific error information is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this specific error information. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

att_value Pointer to the PE containing the encoded specific error information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values PROCESSING_FAILURE.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NULL_MSG_PTR, FIELD_DOES_NOT_EXIST, NO_SUCH_MSG_TYPE

6.28. extract_Id

```
int extract_Id (type, bmoc_type, bmoc, id_type, id, msg_type, msg_ptr)
   int       *type;
   int       *bmoc_type;
   union ID  *bmoc;
   int       *id_type;
   union ID  *id;
   int        msg_type;
   char      *msg_ptr;
```

Description

The *extract_Id()* routine extracts the field for the action Id, or event Id, from the CMIS message. The type parameter is set to indicate the form (action Id or event Id) of the ID parameter. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error condition.

Parameters

type Indicates whether the ID parameter is an action Id or event Id.

Range of Values type_CMIP_NoSuchArgument_actionId or type_CMIP_NoSuchArgument_eventId

bmoc_type Indicates whether the Managed Object Class designation is in local or global form.

Range of Values LOCAL or GLOBAL

bmoc Either an integer that specifies the Managed Object Class identifier in local form, or a character string containing the object class identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

id_type Indicates whether the action Id/event Id designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action/event identifier in local form, or a character string containing the action/event identifier in global form, based on the id_type parameter.

Range of Values Same as bmoc above.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values NO_SUCH_ARGUMENT.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE,
FIELD_DOES_NOT_EXIST, NULL_MSG_PTR, NO_MEM

6.29. extract_value

```
int extract_value( action_or_event, id_type, id, att_value, msg_type, msg_ptr)
    int       *action_or_event;
    int       *id_type;
    union ID  *id;
    PE        *att_value;
    int       msg_type;
    char      *msg_ptr;
```

Description

The *extract_value()* routine extracts the invalid argument field from the CMIS message. The information is returned in the form of a presentation element (PE). It is the responsibility of the CMIS user to call the appropriate decode functions to decode this information. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

action_or_event Indicates whether the invalid argument error contained in the CMIS message is for an action or an event.

Range of Values ACTION_ERR, EVENT_ERR

id_type Indicates whether the action/event designation is in local or global form.

Range of Values LOCAL or GLOBAL

id Either an integer that specifies the action/event identifier in local form, or a character string containing the action/event identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

att_value Pointer to the PE containing the encoded information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values INVALID_ARGUMENT_VALUE

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, FIELD_DOES_NOT_EXIST,
NULL_MSG_PTR, NO_MEM

6.30. extract_actionErrorInfo

```
int extract_actionErrorInfo(error_status, id_type1, ID1, id_type2,
                            ID2, att_value, msg_type, msg_ptr)
      int       *error_status;
      int       *id_type1;
      union id  *ID1;
      int       *id_type2;
      union id  *ID2;
      PE         *att_value;
      int        msg_type;
      char       *msg_ptr;
{
```

Description

The *extract_actionErrorInfo()* routine extracts the actionErrorInfo field of the CMIS message. Upon returning from this function, all of the parameters passed to the function have been filled in with the actionErrorInfo information(except for msg_type and msg_ptr which are inputs to the *extract_actionErrorInfo()* routine). The information contained in each parameter is described in the parameters section below. The function then returns a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

errorStatus This parameter will be filled in with the error that was received.

Range of Values ACCESSDENIED, NO_SUCH_ACTION, NO_SUCH_ARGUMENT, INVALID_ARGUMENT_VALUE

id_type1 Indicates whether the attribute ID designation, as contained in the message, is in local or global form.

Range of Values LOCAL or GLOBAL

id1 Either an integer that specifies the attribute identifier in local form, or a character string containing the attribute identifier in global form, based on the id_type parameter.

Range of Values If local form, an integer value between 1 and 2(32)-1; if global form, the first integer value represented in the string must be 0, 1, or 2. The second value must be between 0 and 39, if the first element is 0 or 1. Subsequent values must be non-negative integers.

Sample Values local form 35, global form "1.17.244.5"

id_type2 Same as parameter id_type1.

id2 Same as parameter id1.

att_value Pointer to the PE containing the encoded information.

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values
INVALID_FILTER, COMPLEXITY_LIMITATION, SET_REQ, SET_IND, GET_REQ, GET_IND, ACTION_REQ, ACTION_IND, DELETE_REQ, DELETE_IND.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

Return Values

SUCCESS, NO_SUCH_MSG_TYPE, NULL_MSG_PTR, BAD_FORM, NO_MEM INVALID_ERROR_STATUS.

6.31. extract_deleteErrorInfo

```
int extract_deleteErrorInfo(msg_type, msg_ptr, deleteErrorInfo)
        int       msg_type;
        char      *msg_ptr;
        int       *deleteErrorInfo;
```

Description

This function retrieves the information contained in the deleteErrorInfo field of the CMIP PDU. The *extract_deleteErrorInfo()* routine fills in the deleteErrorInfo parameter with the deleteErrorInfo information retrieved from the message. The function then returns with a SUCCESS indication. If any errors are detected prior to a successful completion of this function, the function is terminated at that point with the appropriate error indication.

Parameters

msg_type Type of CMIS service message from which the parameter information is to be extracted.

Range of Values DELETE_ERR, ACTION_ERR.

msg_ptr Pointer returned from the extract_cmip_message function, which designates the received CMIS message from which the information is to be extracted.

deleteErrorInfo The integer value for deleteErrorInfo.

Range of Values Integer values between 1 and 2(32)-1

Return Values SUCCESS, NO_SUCH_MSG_TYPE

ERROR CODES	
Return Values	**Meaning**
NO_SUCH_MSG_TYPE	Operation value out of range, not a valid CMIS message type.
NO_MEM	Insufficient memory available to allocate necessary data structure.
SCOPE_VALUE_OUT_OF_RANGE	The scope value is out of range.
SCOPE_TYPE_OUT_OF_RANGE	The scope type is out of range.
BAD_FILTER	Certain critical values of filter were set incorrect or NULL.
NULL_MSG_PTR	NULL CMIS message pointer.
BAD_FORM	Name type was not in LOCAL or GLOBAL.
GLOB_RANGE	Object identifier value out of range.
FIELD_DOES_NOT_EXIST	This fill function is not appropriate for the operation you are trying to perform. If this is a return from init_operation_struct() then the fill.h table was corrupted or modified.
FIELD_ALREADY_FILLED	By calling this fill function you are trying to fill something that is already allocated and possibly filled.
REQUEST_INCOMPLETE	A mandatory function for this request operation was not called.
RESPONSE_INCOMPLETE	A mandatory function for this response operation was not called.
BAD_NAME_TYPE	Value out of range.
DN_RANGE	On first call, instance type was not one of distinguished name, local distinguished name or non-specificform. On subsequent calls, instance type was changed while adding RDNs and AVAs.
INVALID_MODE	Mode value was not Confirmed or Unconfirmed.
INVALID_ERROR_STATUS	Error status parameter on get/set Info status was out of range.
NULL_MO_PTR	The managed object class pointer was NULL.
NULL_ATT_LIST_PTR	The attribute list pointer was NULL.
NULL_ACCESS_PTR	The access control pointer was NULL.
NOT_SUPPORTED_SYNC	Synchronization is not supported on this operation.
NO_DISTINGUISHED_NAME	The distinguished name pointer was NULL.

Table 19

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4866

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
JULY 1992

4. TITLE AND SUBTITLE
Network Management Support for OSI Systems
(NeMaSOS) Version 2.0
Programmer's Reference Manual

5. AUTHOR(S)
Kevin Brady Robert Aronoff
Jim Fox

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The NIST Network Management Support for Open Systems (NeMaSOS) project developed a prototype network manager conforming strictly to the CMIS/CMIP IS version 1, and the OSI Workshop Implementors Agreements of December 1991. This document describes libraries of functions implemented to provide the services of CMIS/P, ACSE, and ROSE. The service interface for UNIX systems using ISODE is also described.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Association Control Service Element (ACSE); Common Management Information Services and Protocol (CMIS/CMIP); ISO Development Environment (ISODE); Network Management; OSI Network Management Implementor's Agreements; OSI Network Management Standards; Remote Operations Service Element (ROSE).

13. AVAILABILITY

☒ UNLIMITED

□ FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

□ ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

☒ ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
180

15. PRICE
A09

ELECTRONIC FORM