Developing CORBA Applications

Solstice Enterprise Manager™ 4.1

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

Part No. 806-7977-10
October 2001, Revision A

Please
Recycle

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

The *Developing CORBA Applications* guide provides information required to develop CORBA based network manager applications that interact with the SEM CORBA Gateway. It contains procedures, guidelines and examples for various functions that can be carried out through these interactions.

Who Should Use This Book

This document is intended for SEM CORBA applications developers, which includes architects, designers and coders. No prior experience with Solstice Enterprise Manager (Solstice EM) is assumed. However, if you are not familiar with Solstice EM, see Section “Related Books” on page -xv for a listing of books to refer.

Before You Read This Book

Read through the Solstice EM documentation so that you have an understanding of the programming context, because many parts of this book refer to concepts that are covered in documents listed in Section “Related Books” on page -xv.

It is assumed that you have a basic understanding of CORBA application development as well as Network Management concepts. It is also recommended that you go through the JIDM document, *Inter-Domain Management: Specification Translation and Interaction Translation*.

How This Book Is Organized

This book contains the following chapters:

Chapter 1 "**Introduction to SEM CORBA Development Environment**" provides an introduction to SEM CORBA development environment.

Chapter 2 "**Interacting With SEM CORBA Gateway**" introduces the various interfaces that SEM CORBA Gateway implements and explains how to interact with the SEM CORBA Gateway.

Chapter 3 "**Managing Networks With SEM CORBA Gateway**" explains SEM CORBA Gateway from the network management perspective.

Chapter 4 "**Handling Events With SEM CORBA Gateway**" explains event handling in detail.

Chapter 5 "**Translating Data**" describes various data formats used in SEM CORBA Gateway and how to transform from one to the other.

Chapter 6 "**Accessing Information Through Metadata Gateway**" describes Metadata Gateway functionality and how to interact with Metadata Gateway.

Chapter 7 "**Managing Agents**" describes how to manage different kinds of agents (SNMP/CMIP) using SEM CORBA Gateway.

Chapter 8 "**Interoperating SEM CORBA Gateway**" considers scenarios where the manager application is developed using different ORBs other than the ORB used for SEM CORBA Gateway.

Appendix A "**IDLs Used by SEM CORBA Gateway**" provides a listing of IDLs used by SEM CORBA Gateway.

Appendix B "**Programming Techniques**" details a few programming tips for CORBA application development.

Appendix C "**criteria for ProxyAgents**" details `criteria` for `ProxyAgents`.

Related Books

Following is a list of related books:

- *Developing C++ Applications*
- *Management Information Server (MIS) Guide*
- *Managing Your Network*
- *Customizing Guide*
- *CORBA Gateway Administration Guide*

What Typographic Changes Mean

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

TABLE P-1 Typographic Conventions

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

Shell Prompts in Command Examples

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

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Accessing Sun Documentation Online

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

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

Sun Welcomes Your Comments

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

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

CHAPTER 1

Introduction to SEM CORBA Development Environment

You can use the SEM CORBA development environment to extend the functionality of Solstice Enterprise Manager (Solstice EM) by developing custom applications to meet your particular network management needs. These applications will interact with the SEM CORBA Gateway and provide a means for a comprehensive CORBA-based north-bound interface. Your applications can be truly independent of the platform (they can be on a different machine), the ORB and the language of implementation.

In order to set up the SEM CORBA development environment you should install and execute configuring scripts of the SEM CORBA ToolKit. You also need to build the SEM CORBA Gateway executables using the ToolKit. For more information refer to Chapter 2 “SEM CORBA ToolKit” in the *CORBA Gateway Administration Guide*.

This chapter introduces the SEM CORBA ToolKit development environment, provides an overview of the SEM CORBA architecture, and introduces you to related technical terms and resources. It describes the following topics:

1.1 Overview of SEM CORBA Architecture

The SEM CORBA Gateway translates CORBA manager requests in Interface Description Language (IDL) to Solstice EM Portable Management Interface (PMI) or equivalent requests. CORBA Gateways also translate Solstice EM PMI responses to IDL or Internet Inter-ORB Protocol (IIOP) responses, and PMI events to CORBA events. This product supports the manager-side interfaces to Solstice EM based on the CORBA/Telecommunications Management Network (TMN) Interworking

standard. Solstice EM also provides an Abstract Syntax Notation 1 (ASN1) metadata server, which enables client applications to access and traverse ASN1 type information for a given attribute of a managed object or notification.

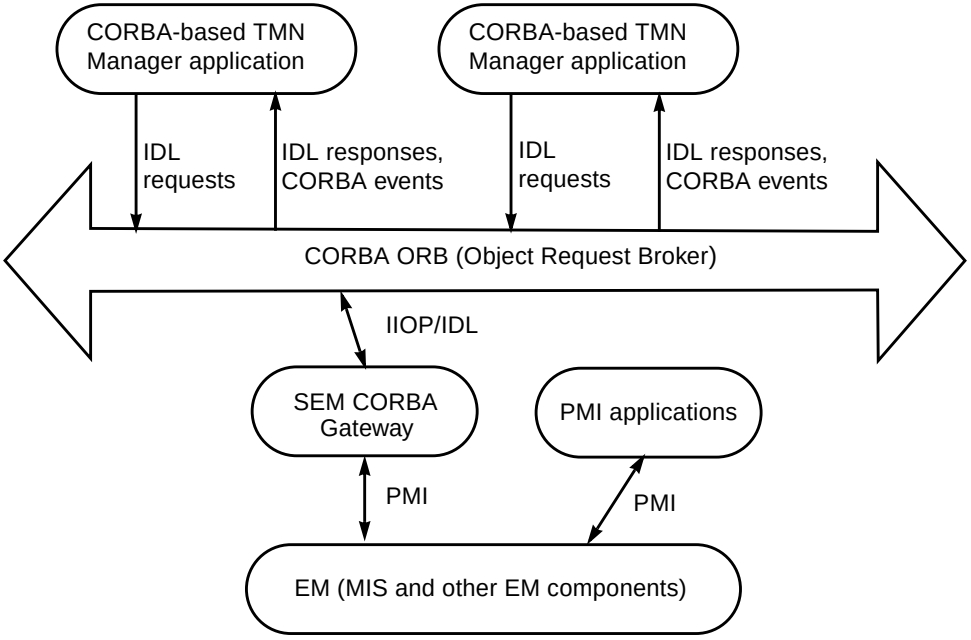


FIGURE 1-1 SEM CORBA Gateway Architecture

For more information about the CORBA/TMN Interworking standard, refer to the proposed standards at the *Object Management Group* (OMG) website listed in Section 1.3 “References” on page 1-4.

1.2 SEM CORBA ToolKit Development Environment

The development environment is provided by the ToolKit. The ToolKit is ORB specific, which means the configuration files and Makefiles that get installed are dependent on the ORB you choose while installing Solstice EM.The machine on which you install the ToolKit will be your development machine.

The ToolKit consists of a set of IDLs, source files, header files, Makefiles and configuration files. You must execute the `Makefile` in the home directory of the ToolKit to build, package and optionally install the SEM CORBA Gateway on the development machine. The runtime package created can be used to deploy SEM CORBA Gateway on any other machine.

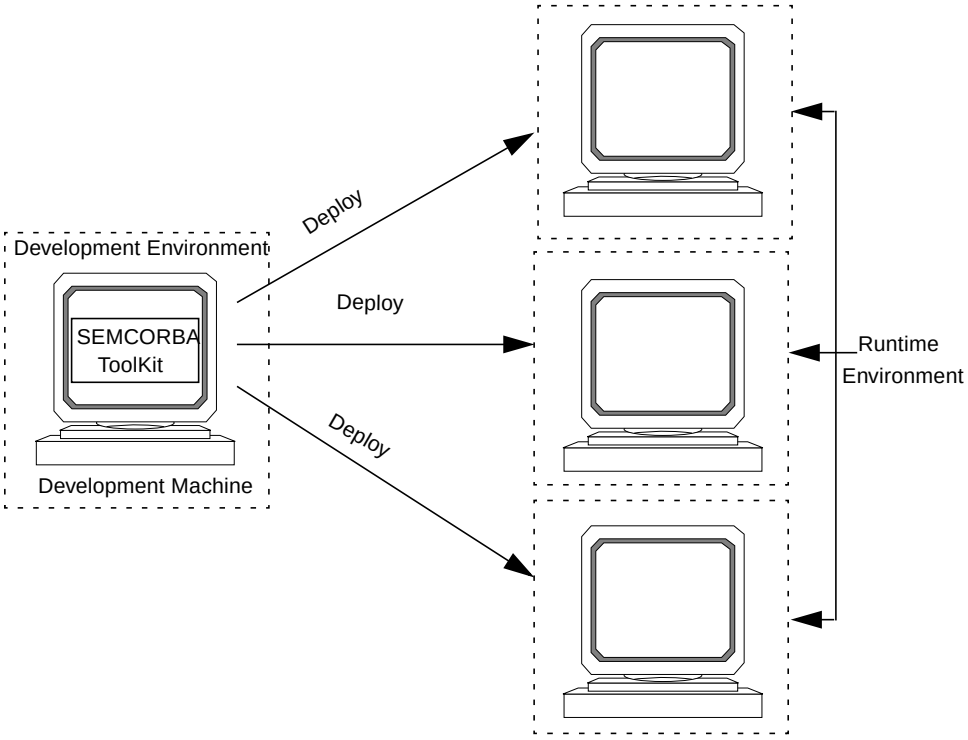


FIGURE 1-2 SEM CORBA ToolKit Development Environment

For more information refer to Chapter 2 “SEM CORBA ToolKit” in the *CORBA Gateway Administration Guide*.

1.3 References

1. *CORBA/TMN Interworking* (also known as *JIDM Interaction Translation*) Standard - final submission to OMG's CORBA/TMN Interworking RFP. Refer to the OMG web site at:

   `http://www.omg.org/cgi-bin/doc?telecom/98-10-10`

2. *CORBA Standards Specifications: CORBA 2.3* by the Object Management Group.

3. Your ORB vendor's documentation.

CHAPTER 2

Interacting With SEM CORBA Gateway

The CORBA applications need to interact with SEM CORBA Gateway to perform network management functions. You should know the interfaces and methods exposed by the SEM CORBA Gateway in order to develop these applications which will interact with SEM CORBA Gateway.

This chapter describes the following topics:

The SEM CORBA Gateway is compliant with Joint Inter-Domain Management (JIDM) standards and provides a set of *interfaces* based on these standards. The SEM CORBA Gateway is designed to work with standard management reference models (such as SNMP/IP and CMIP/OSI). The interfaces defined and implemented by the SEM CORBA Gateway define the interaction between it and applications. Hence, it is critical for developing efficient applications to understand these interfaces.

Note – You are assumed to have installed the SEM CORBA ToolKit and setup the development environment. All examples cited henceforth are more appropriate in a scenario in which the ToolKit has been installed and configured.

2.1 JIDM Interfaces

The JIDM Standards specify interfaces that define a basic set of services, of which the SEM CORBA Gateway implements a subset. This section lists the implemented interfaces sorted into different categories.

2.1.1 Interfaces Required for Manager Applications

The following interfaces are required for manager (client) applications:

- `ProxyAgent` interface
- `ProxyAgentController` interface
- `ProxyAgentFinder` interface
- `EventPort` interface
- `EventPortFactory` interface

Note – Throughout this document, the terms “manager application” and “client” are used interchangeably.

The `ProxyAgent` interface implements the following CMIS commands required for a client application to interact with the SEM CORBA Gateway:

- cmis_get
- cmis_set
- cmis_create
- cmis_create_sync
- cmis_delete
- cmis_action

2.1.2 Interfaces Required for Agent Applications

The following interface is required for agent applications:

- `EventPortFinder` interface

2.1.3 OSI Management Interfaces

The Open System Interconnection (OSI) management interface definitions provided by JIDM standards (`OSIMgmt.idl`) include features that are specific to the OSI management model and support the following functionality:

- Scope and filtering
- Naming of objects according to OSI management principles
- Creation and deletion of objects according to OSI principles
- Creation of `DomainPort`s and `EventPort`s associated with OSI `AE-titles`

The SEM CORBA Gateway implements the following OSI management interfaces which are a subset of the JIDM standards:

- `ProxyAgent` interface derived from `JIDM::ProxyAgent`
- `LinkedReplyHandler` and `EndOfRepliesHandler` interfaces

The `ProxyAgent` interface provides all the OSI network management functionality.

The `LinkedReplyHandler` and `EndOfRepliesHandler` interfaces facilitate implementation of the asynchronous model of communication.

Note – Appendix A provides more information on the IDL files and interfaces implemented by the SEM CORBA Gateway.

FIGURE 2-1 shows the interfaces exposed by the SEM CORBA Gateway components that are the basis on which applications can be developed.

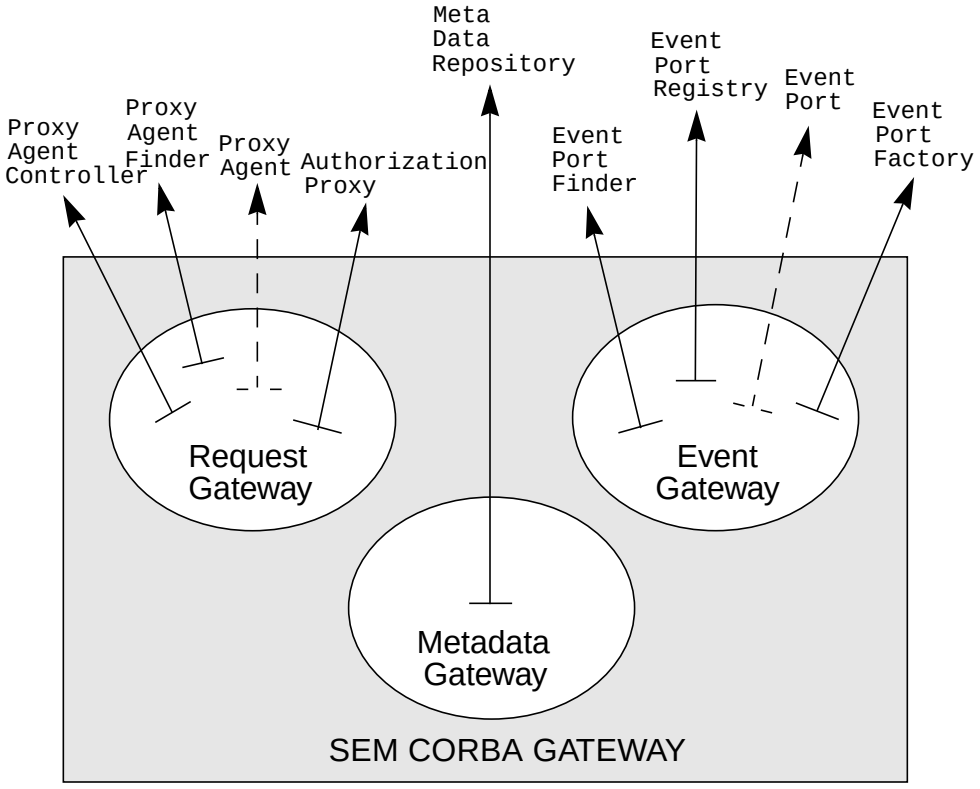


FIGURE 2-1 Interfaces Exposed by SEM CORBA Gateway Components[1]

2.2 Interacting With Solstice EM CORBA Request Gateway

The Request Gateway (RGW) is primarily designed to handle the initial CORBA client connections to the gateway and CMIS *requests* or *responses*. Hence, it implements the interfaces that support the following functionality:

- Connecting clients for the first time
- Authenticating clients
- Accessing the managed object domain and creating `JIDM::ProxyAgent`
- Handling CMIS requests and responses

1. The interfaces shown with dotted lines are created dynamically.

2.2.1 Connecting Clients for the First Time

A client application (for example a CORBA manager application) must be connected to the SEM CORBA Gateway before you can carry out any manager functions. The steps involved in connecting to the Gateway are:

- Getting the `ProxyAgentFinder` object
- Creating an `AuthenticationClient`
- Encrypting the user profile
- Finding/Creating the `ProxyAgent`

A detailed explanation of each step, and a code example follow.

1. **Get the `ProxyAgentFinder` object**

   The first step is to get a reference to the `ProxyAgentFinder`, an interface implemented by the RGW. This enables client programs to get access to a given managed object domain.

   The client application must initialize the Object Request Broker (ORB), initialize the Object Adapter, and resolve the *root naming context* before the Proxy Agent Finder can be resolved.

2. **Create an `AuthenticationClient`**

   The second step in connecting to the Gateway is to create an `AuthenticationClient`. The Solstice EM provides UNIX CLEAR *encryption* and *decryption* for authenticating users, but you can implement an authentication class and integrate it with Solstice EM (see Section 2.9 “Providing an Extra Layer of Authentication” on page 2-17).

3. **Encrypt the user profile**

   The third step is to specify the user profile in the `JIDM::Criteria` structure. The authentication client created in Step 2, provides a method for encrypting the user profile. The encrypted user profile is specified in the `JIDM:Criteria` structure.

4. **Find/Create the `ProxyAgent`**

   The user profile provided as part of `criteria` is validated before finding or creating the `ProxyAgent`.

   The `ProxyAgentFinder` has a method called `access_domain` that locates the `ProxyAgent`, for the `key` and the `criteria` that are passed as parameters. `criteria` is the attribute that is specified in Step 3, with the encrypted user profile.

The following code segment shows how to get connected to the SEM CORBA Gateway.

For detailed working examples, refer to `/opt/SUNWconn/em/src/corba_gateway/requests` directory of the Solstice EM package.

CODE EXAMPLE 2-1 Connecting to the SEM CORBA Gateway (for VisiBroker)

```
// Example to show how to get connected to SEM CORBA gateway.
//
#include <corba.h>
#include <stdlib.h>
#include <cos/CosNaming_c.hh>

int
main(
    int argc,
    char **argv
) {
    CORBA::ORB_var orb;
    PortableServer::POA_var root_poa;
    PortableServer::POAManager_var poa_manager;
    PortableServer::POA_var cgwPOA;


    // CORBA Orb intialization
    orb = CORBA::ORB_init(argc, argv);
    root_poa =
    PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

    // Get rootnaming context
    CosNaming::NamingContext_var root_nc;
    CORBA::Object_var object = orb-
>resolve_initial_references("NameService");
    root_nc = CosNaming::NamingContext::_narrow(object);

    if(CORBA::is_nil(root_nc)) {
    cerr << "FAILED: Unable to obtain root naming context\n" <<
flush;
      exit(0);
    }

    // STEP NO 1: Find proxy agent finder
    CosNaming::Name name;
    name.length(1);
    JIDM::ProxyAgentFinder_var proxy_agent_finder;
    name[0].id = CORBA::string_dup("JIDM::ProxyAgentFinder");
```

CODE EXAMPLE 2-1 Connecting to the SEM CORBA Gateway (for VisiBroker) *(Continued)*

```
    name[0].kind = CORBA::string_dup("");
    CORBA::Object_var object = root_nc->resolve(name);
    proxy_agent_finder = JIDM::ProxyAgentFinder::_narrow(object);
    if (CORBA::is_nil(proxy_agent_finder)) {
    cerr << "FAILED:Unable to obtain correct Proxy Agent Finder\n"
<< flush;
        exit(1);
    }
    if(proxy_agent_finder->_non_existent()) {
        cerr << "FAILED: Proxy Agent Finder does not exist\n" <<
flush;
        exit(1);
    }

    // Get use name and password user
    cout << "Enter User Name: " << flush;
    char user_name[128];
    cin >> user_name;
    const char* password_prompt = "Enter Password: ";
    // getpassphrase return a pointer to static data which should
not be
    // deleted
    char* raw_password = getpassphrase(password_prompt);
    cout << "Got your password will authenticate..\n " << flush;

    // STEP NO 2:  create authentication client
    AuthenticationClient* ac = new
AuthenticationClientHandle("UNIX_CLEAR");

    // STEP NO 3: ENcrypt the user profile
    JIDM::Criteria a_criteria;
    a_criteria[1].name = CGWGlobal::JIDM_USER_PROFILE;
    cout << "JIDM user profile done ..\n" << flush ;
    a_criteria[1].value = *(ac->encrypt_user_profile(
        user_name,
        raw_password,
        NULL));
    f(a_criteria[1].value.type()->kind() == CORBA::tk_null) {
      cerr << "FAILED: Unable to obtain encrypted user profile\n"
<< flush;
        exit(3);
    }

    // STEP NO 4: Find the proxy agent
    JIDM::ProxyAgent_var proxy_agent;
```

CODE EXAMPLE 2-1 Connecting to the SEM CORBA Gateway (for VisiBroker) *(Continued)*

```
    proxy_agent = proxy_agent_finder->access_domain(a_key,
a_criteria);
    if (CORBA::is_nil(proxy_agent)) {
     cerr << "FAILED: Unable to obtain correct Proxy Agent" << endl
<< flush;
        exit(4);
    }
    if(proxy_agent->_non_existent()) {
        cerr << "FAILED: Proxy Agent does not exist" << endl <<
flush;
    exit(4);
    }

    // Successfully for the Proxy Agent, hence connected to SEM
CORBA Gateway
}
```

2.2.2 Authenticating Clients

Authentication of clients is done by `ProxyAgentFinder`. The `ProxyAgentFinder` implementation uses internal Solstice EM functionality to check whether the supplied user profile refers to a valid user of Solstice EM and if so, the `ProxyAgentFinder` returns a reference of the `ProxyAgent` to the client.

2.2.3 Accessing the Managed Object Domain and Creating `JIDM::ProxyAgent`

Managed objects are classified into domains. Manager applications that need to interact with a managed object, must first get access to its domain. The SEM CORBA RGW creates a `ProxyAgent` whenever a client gains access to the domain. The RGW searches through its list of `ProxyAgent`s for any pre-existing `ProxyAgent` that matches the `criteria`, and if one is found, it returns a reference for it back to the client application. If a `ProxyAgent` for the required `criteria` is not found, the `ProxyAgentFinder` interface creates a new `ProxyAgent` and returns its reference to the client application.

2.2.4 Handling CMIS Requests and Responses

The RGW is solely responsible for handling requests sent by manager applications and the responses sent back to them. The RGW accepts the manager requests in IDL, translates them into low-level PMI requests, and sends them to the Solstice EM MIS. The Solstice EM MIS responds to the RGW. The RGW then translates these responses back into IDL format and sends them back to the CORBA application.

The RGW implements the OSI management features specified by the JIDM standards and hence supports those specific to OSI management. The OSI management implementation of the RGW includes a `ProxyAgent` interface (derived from `JIDM::ProxyAgent`) and includes methods to support all the OSI management CMIS operations, such as `get`, `set`, and `create`.

2.3 Non JIDM interfaces

The SEM CORBA Gateway provides an extension by means of text based CMIS commands, listed in Section 2.1.1 “Interfaces Required for Manager Applications” on page 2-2.

The following text commands are supported by implementing an interface which is derived from `JIDM::ProxyAgent`.

- `cmis_get_text`
- `cmis_set_text`
- `cmis_create_text`
- `cmis_create_sync_text`
- `cmis_delete_text`
- `cmis_action_text`

For details on the arguments to be passed to these commands, refer to the examples in the `EM_HOME/src/corba_gateway/examples` directory. Also refer to the IDL files `ASN1TypesExt.idl`, `CMIExt.idl`, and `OSIMgmtExt.idl`.

2.4 Interacting With Solstice EM Event Gateway

The Solstice EM Event Gateway (EGW) primarily handles the delivery of CMIS events / notifications to CORBA clients or manager applications. To listen to events, the clients or manager applications created by you must handle the following functionality in the given order:

1. Creating and registering event ports (using the `EventPort` interface)
2. Creating event discriminating filters (EDFs) using `M-create`
3. Issue of `cmis_create()` commands on the `ProxyAgents` implemented by RGW.

The Solstice EM EGW implements the following three interfaces specified by JIDM standards:

- `EventPort` interface
- `EventPortFactory` interface
- `EventPortFinder` interface

Apart from the three JIDM standard interfaces, the Solstice EM EGW provides another interface called `EventPortRegistry`.

These interfaces and instructions for the interfacing of client / manager applications are described in the following sections.

2.4.1 Gaining Access to a Manager or Client Application

A managed object gains access to a manager or a client application, through the `EventPort` so that the event notifications can be forwarded to the manager. This port is created according to certain criteria and contains the title that identifies the manager domain. The Solstice EM event dispatcher (part of MIS) sets up a connection with the `CosEventChannelAdmin::SupplierAdmin` object associated with the `JIDM::EventPort` (implemented by the EGW) with the appropriate title.

2.4.2 Dynamically Creating JIDM::EventPort Objects

The EventPortFactory interface facilitates dynamic creation of JIDM::EventPort objects by providing the create_event_port() method. This interface is implemented by the EGW and basically does the following every time a new manager/client application registers with the EGW:

- Gets a reference to the CosEventChannelAdmin::SupplierAdmin that is to be used to receive events.
- Creates JIDM::EventPort with the criteria supplied by the manager application.

The following code example gives you the steps to follow for implementing a manager application.

CODE EXAMPLE 2-2 Creating a JIDM::EventPort and Registering for Events

```
// Example to show how to create JIDM::EventPort and register for
events
// Assumes ORB is initialized and naming service is resolved.

    CosNaming::Name name;
    name.length(1);
    name[0].id = CORBA::string_dup("EventPortRegistry");
    name[0].kind = CORBA::string_dup("");

    EventPortRegistry_var epr;
    try {
        CORBA::Object_var object = root_nc->resolve(name);
        epr = EventPortRegistry::_narrow(object);
        if (CORBA::is_nil(epr)) {
            cerr << "FAILED: Could not get EventPortRegistry\n";
            exit(3);
        }
    }
    catch (const CORBA::Exception& e) {
      cerr << "FAILED: Could not resolve EventPortRegistry name\n";
        exit(4);
    }
    cout << "PASSED: Accessing the EventPortRegistry\n" << flush;

    JIDM::EventPortFactory_var ep_factory;
    name[0].id = CORBA::string_dup("EventPortFactory");
    try {
        CORBA::Object_var object = root_nc->resolve(name);
        ep_factory = JIDM::EventPortFactory::_narrow(object);
        if (CORBA::is_nil(epr)) {
```

CODE EXAMPLE 2-2 Creating a JIDM::EventPort and Registering for Events *(Continued)*

```
            cerr << "FAILED: Could not get EventPortFactory\n";
            exit(3);
        }
    }
    catch (const CORBA::Exception& e) {
      cerr << "FAILED: Could not resolve EventPortFactory name\n";
        exit(4);
    }
    cout << "PASSED: Accessing the EventPortFactory\n" << flush;

    JIDM::EventPortFinder_var ep_finder;
    name[0].id = CORBA::string_dup("EventPortFinder");
    try {
        CORBA::Object_var object = root_nc->resolve(name);
        ep_finder = JIDM::EventPortFinder::_narrow(object);
        if (CORBA::is_nil(epr)) {
            cerr << "FAILED: Could not get EventPortFinder\n";
            exit(3);
        }
    }
    catch (const CORBA::Exception& e) {
       cerr << "FAILED: Could not resolve EventPortFinder name\n";
        exit(4);
    }
    cout << "PASSED: Accessing the EventPortFinder\n" << flush;


    // Find an EventPort with a right key and criteria

    JIDM::Key key;
    key.length(1);
    key[0].id = CORBA::string_dup("OSI Management");
    key[0].kind = CORBA::string_dup("XSM environment");

    JIDM::Criteria event_criteria;
    event_criteria.length(2);
    event_criteria[0].name = “domain title”;
    event_criteria[0].value <<= CORBA::string_dup(ae_title);

   event_criteria[1].name = CORBA::string_dup("ProxyAgent Access
Criteria");
    event_criteria[1].value <<= criteria;

 //Find/Create Event Port
 try {
        CosEventChannelAdmin::SupplierAdmin_var supplier_admin =
            epr->find_event_port(key, event_criteria);
```

CODE EXAMPLE 2-2 Creating a JIDM::EventPort and Registering for Events *(Continued)*

```
    }
    catch (const JIDM::NoEventPort& e1) {
        cout << "PASSED: Finding EventPort with wrong key.\n" <<
       "\tAs expected, NoEventPort exception was raised\n" << endl;
    }
    catch (const CORBA::Exception& e) {
    cerr << "FAILED: Finding EventPort with wrong key" << e._name() <<
            "\nExpected to get NoEventPort exception\n";
        exit(5);
    }

    // Resolve the EventChannel
    name[0].id = CORBA::string_dup("event_channel");


    CosEventChannelAdmin::EventChannel_var channel =
  resolve_name<CosEventChannelAdmin::EventChannel>(root_nc,name);

    CosEventChannelAdmin::SupplierAdmin_var supplier_admin =
    channel->for_suppliers();

    try {
        JIDM::EventPort_var ep = epr->create_event_port(key,
event_criteria,
                                                    supplier_admin);
    }
    catch (const CORBA::Exception& e) {
     cerr << "FAILED: Could not create event port\n" << e._name()<<
endl;
        exit(8);
    }
   // Event port has been successfully created....
    cout << "PASSED: Event port creation\n" << flush;


  //Listen to events by creating the push_consumer object

  // Set policy for the root POA
  policies[(CORBA::ULong)0] =
  root_poa->create_lifespan_policy(
  PortableServer::TRANSIENT);


//ThePushConsumerImplisderivedfromPOA_CosEventComm::PushConsumer
  // and implements method push to receieve events.
  PushConsumerImpl push_consumer;
```

CODE EXAMPLE 2-2 Creating a `JIDM::EventPort` and Registering for Events *(Continued)*

```
   PortableServer::POAManager_var rootManager =
   root_poa->the_POAManager();

   root_poa->activate_object(&push_consumer);

   rootManager->activate();

 //---------------------endofexample2.2--------------------------
 ---------
 Note: This example activates a POA push consumer object.
```

2.4.3 Obtaining References to `JIDM::EventPort`

CORBA manager applications can obtain references to `JIDM::EventPort` objects by using the `EventPortFinder` interface. This interface provides a `find_event_port` method which returns a reference to the `CosEventChannelAdmin::SupplierAdmin` object.

2.4.4 Finding a `JIDM::EventPort` given the `AE-title`

The `EventPortRegistry` interface provides a method for finding a `JIDM::EventPort` given the `AE-title`. The current implementation of SEM CORBA Gateway has a one-to-one relationship between titles and event ports.

2.5 Interacting With Solstice EM Metadata Gateway

The Solstice EM Metadata Gateway (MGW) implements the Metadata Repository interface, which provides methods to access ASN1 metadata in Solstice EM. The MGW is independent of the CORBA Gateway of the JIDM, which means that it can be used by any application (even a non-JIDM application) to obtain information about events or attributes, or to traverse the type tree.

See Chapter 5 and Chapter 6 for examples on how to interact with the Solstice EM MGW.

2.6 Controlling Access and Authorization

SEM CORBA Gateway access control involves two aspects:

- Encrypting and decrypting the user profile
- Authenticating user profiles

2.6.1 Encrypting and Decrypting the User Profile

Encryption and decryption are implemented in the following two libraries:

- `/opt/SUNWconn/em/lib/libauth_server.so`
- `/opt/SUNWconn/em/lib/libauth_client.so`

These libraries are built based on implementation of classes declared in the following two header files respectively:

- `/opt/SUNWconn/em/include/auth_helper/auth_server_handle.hh`
- `/opt/SUNWconn/em/include/auth_helper/auth_client_handle.hh`

You can use these class definitions to implement your own encryption and decryption.

Specific steps to be performed are as follows:

1. **Implement** `*.cc` **files for** `auth_server_handle.hh`
2. **Implement** `*.cc` **files for** `auth_client_handle.hh`
3. **Create a** `Makefile` **to compile the** `*.cc` **files and to build the libraries (create two separate libraries, one for the client and one for server object files).**
4. **Make a backup of the client and server libraries provided with Solstice EM.**
5. **Stop SEM CORBA services.**
6. **Replace (overwrite) the default libraries with those created in Step 3.**
7. **Restart SEM CORBA services.**

2.6.2 Authenticating User Profiles

The authentication of user profiles is implemented as part of `ProxyAgentFinder`. To be able to successfully connect to the Gateway, every application that connects to the SEM CORBA Gateway must provide a correct user profile which is authenticated while accessing the `ProxyAgentFinder`.

2.7 Enabling Access From Non-Unix Environments

The SEM CORBA Gateway can be accessed from any environment, irrespective of the machine, the operating system, or the programming language. The only requirement is that the ORB used to develop the client applications use Object Management Group's (OMG's) Internet Inter-ORB Protocol (IIOP) to communicate with the SEM CORBA Gateway.

The examples included with the Solstice EM package in the `/opt/SUNWconn/em/src/corba_gateway/requests` directory implement sample clients in both Java and C++.

2.8 Enabling Internet Connections to Solstice EM via CORBA Gateways

The SEM CORBA Gateway is a set of objects. The applications that wish to connect to the gateways must obtain references of these objects.

The SEM CORBA implementation publishes the references of objects through the *naming service* and all the gateways are accessed by resolving the naming service.

A CORBA Gateway can be accessed from anywhere on the network, as long as client applications use IIOP. The firewalls and the gateways can be programmed to pass IIOP messages and hence it is possible for a client application to communicate with a CORBA Gateway from anywhere on the network.

2.9 Providing an Extra Layer of Authentication

You can use the SEM CORBA Gateway to add an extra layer of authentication by providing the following exposed classes:

- `AuthenticationClientHandle`
- `AuthenticationClientBody`
- `AuthenticationClient`
- `AuthenticationServerHandle`
- `AuthenticationServerBody`
- `AuthenticationServer`

Client applications that want to connect to the SEM CORBA Gateway use `AuthenticationClientHandle` to initialize the `AuthenticationClient`. The following code example shows how to initialize the `AuthenticationClient`.

CODE EXAMPLE 2-3 Initializing `AuthenticationClient`

```
AuthenticationClient* ac =
        new AuthenticationClientHandle("UNIX_CLEAR");

char user_name[128];

cout << "Enter User Name: " << flush;
cin >> user_name;
const char* password_prompt = "Enter Password: ";

// getpassphrase return a pointer to static data which should not
be
// deleted
char* raw_password = getpassphrase(password_prompt);


//Assuming that a_criteria is already declared and
//first entry in the criteria is JIDM_MANAGER_TITLE
a_criteria[1].name = CGWGlobal::JIDM_USER_PROFILE;
a_criteria[1].value = *(ac->encrypt_user_profile(
        user_name,
        raw_password,
        NULL));
```

The `AuthenticationClientHandle` class implements the following two useful methods:

- `get_user_profile()`, and
- `encrypt_user_profile()`

The current Solstice EM-supplied implementation of the `AuthenticationServerHandle` class is used only for decrypting the user profiles sent by client applications. The method implemented for this purpose by `AuthenticationServerHandle` is `decrypt_user_profile()`.

Application developers can extend these classes to implement additional layers of security.

Note – Implementations of the exposed functions mentioned earlier must retain their signatures and also the names of the shared libraries are to be retained as per the ToolKit requirement.

CHAPTER 3

Managing Networks With SEM CORBA Gateway

This chapter discusses network management through the SEM CORBA Gateway.

This chapter describes the following topics:

3.1 General Concepts

The SEM CORBA Gateway provides internetworking between CORBA-based management stations and Solstice Enterprise Manager (Solstice EM). Since Solstice EM provides Solstice EM south-bound interfaces to various network technologies (such as SNMP, CMIP and RPC), the SEM CORBA Gateway also accommodates these technologies.

The following sections are adapted from *Developing C++ Applications* guide for Solstice Enterprise Manager™ 4.1; refer to the guide for more information.

3.1.1 Modeling Objects

Network management in the Solstice EM environment follows the ISO network management model. This object-oriented model is based around manager and agent applications that exchange network management information. To manage your

resources in the Solstice EM environment, you must define an object model for those resources. The object model defines the characteristics of resources your application will manage.

3.1.2 Managers

A manager performs the following functions:

- Issuing management requests to one or more agents
- Collecting and filtering information from agents
- Presenting information about the managing system to human operators

A manager receives information from agents in the form of notifications and responses, as follows:

- Notifications - A notification is an unsolicited message sent to the manager indicating that a change has occurred in the managed resource.
- Responses - A response is a message sent in response to a management request containing either the information the manager requested or a confirmation that the request was carried out.

A manager resides in the managing system.

3.1.3 Agents

An agent acts as an intermediary between a manager and managed resources. An agent responds to requests and issues notifications. Each agent acts as an interpreter and a filter, sending commands to each of the managed resources it controls to get the data it requires. Each agent in a managed system is responsible for carrying out management directives to control or return information from managed resources.

An agent can reside in a managed resource or be located elsewhere and operate remotely.

3.1.4 Managed Resources

A managed resource is any network resource that can be managed. The resource can be a physical device such as a host, server, router, or subnet, or it can be a conceptual entity such as a line, a queue, or some other aspect of network operation that must be managed.

3.1.5 Managed Objects

The ISO management model on which Solstice EM is based is object oriented. According to this model, a managed resource is represented as a managed object. A managed object is a software abstraction of a managed resource. The managed object presents the information needed to manage the resource. A managed resource may be represented by one or more managed objects. An agent typically contains or provides views of many managed objects.

3.1.6 Management Protocols

A management protocol is a set of rules that specifies how information is exchanged between two entities that are communicating, such as a manager and an agent. A management protocol provides the common language required to enable managers and agents to exchange information.

A management protocol defines:

- The types of management requests and responses that agents and managers are allowed to issue.
- The syntax and encoding of each type of request and response.
- The sequence in which management requests and responses are allowed to be issued.

3.1.7 Concepts Specific to CORBA and TMN

Some concepts are specific to CORBA and Telecommunication Management Network (TMN). The objects (both managed and manager) are grouped into *domains* according to some specific criteria. Domains are identified by *titles* and are individually accessible using titles. Hence, it is necessary for an entity (manager or managed) to gain access to the domain in order to access the object.

Access to a domain is granted through a *port* (not to be confused with TCP/IP ports). Two types of ports are defined in the JIDM standards, the `EventPort` and the `DomainPort`.

When a manager (or agent) gains access to a managed (or manager) domain, a *session* has been established. A session can be released (terminated), after which no further communication can take place. Establishing a session involves checking for the user's authorization and privileges. Any number of sessions can be active at any given time between any two objects.

3.2 Operating on CORBA Clients and Objects

CORBA clients first need to get access to the domain (or get connected to the gateways) so that they can operate on objects. The operations on the objects can be carried out either *synchronously* or *asynchronously*. These operations basically map to the CMIS services on managed objects. Each of these operations is explained with examples in the following sections.

Note – The operation of creating objects requires two different functions for synchronous and asynchronous creation, while other operations on the objects requires only one function, the `end_of_replies_handler`, based on whose value the operation on the object is synchronous or asynchronous.

3.2.1 Operating Asynchronously and Synchronously

The `LinkedReplyHandler` and `EndOfRepliesHandler` interfaces defined in the JIDM standards enable the clients or applications to handle the replies sent back by the gateways.

These two interfaces must be implemented by the client or manager application. CMIS functions such as `cmis_get()` require references to these two interfaces to be passed as arguments. The `LinkedReplyHandler` interface handles replies sent back (both error and no error responses) and is mandatory. The `EndOfRepliesHandler` interface is optional; if implemented and passed as an argument to the CMIS function, it will render the CMIS operation asynchronous; if `NULL` is passed in place of this argument, the operation will be treated as synchronous and client applications will be blocked until a reply or an error message is received.

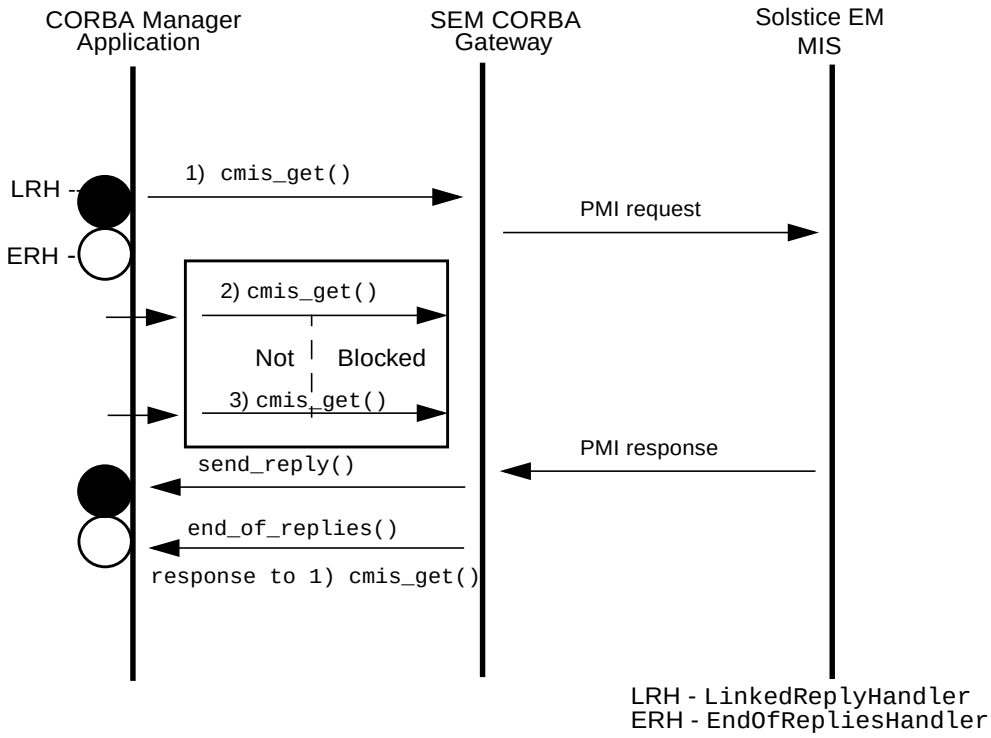


FIGURE 3-1 `get` Asynchronous Operation

In a `get` asynchronous operation, the manager application creates both `LinkedReplyHandler` and `EndOfRepliesHandler`.

When the CORBA application does a `cmis_get()`, it passes the reference for `LinkedReplyHandler` along with the other arguments to the SEM CORBA Gateway. The Gateway passes on the request to Solstice EM MIS. In an asynchronous operation the CORBA manager application is not blocked, and can continue processing.

When a reply from the Solstice EM MIS corresponds to the `get` request, the reference to `EndOfRepliesHandler` that was passed by the CORBA manager application is used to send the reply back to the application, by executing the `end_of_replies()`.

Note – All other operations are similar to the `get` operation except the create operation. The `cmis_create_sync` (synchronous create) command does not have `LinkedReplyHandler` nor `EndOfRepliesHandler`, whereas `cmis_create()` (asynchronous create) has only `LinkedReplyHandler`.

If there is an error in processing the request, the SEM CORBA Gateway executes either `send_subtree_error()` or `send_no_error()`, depending on the error condition.

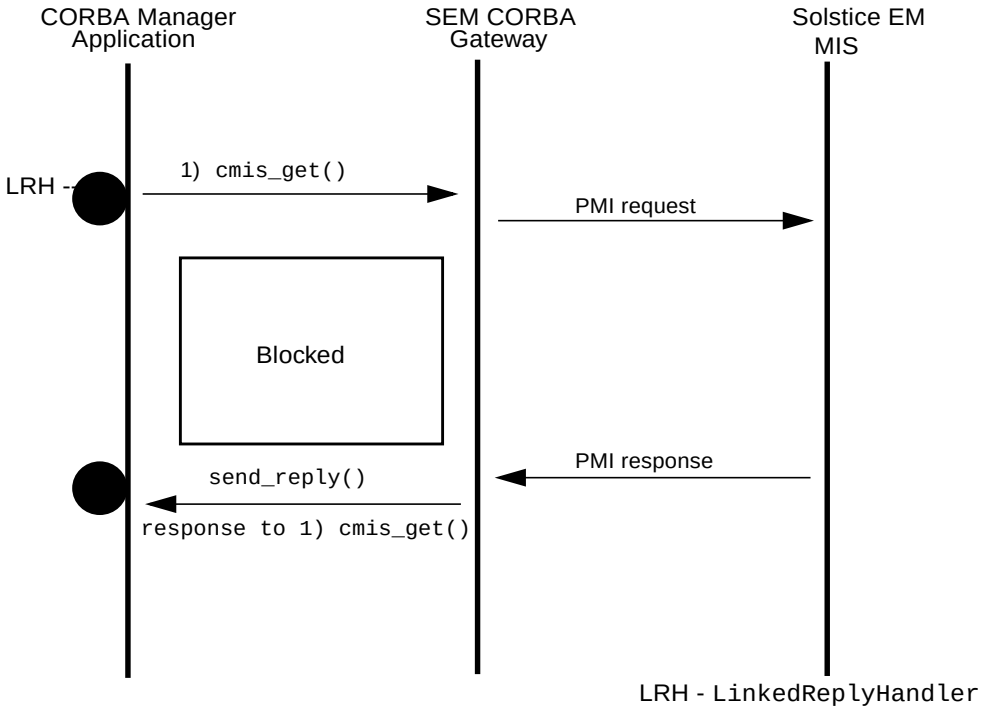


FIGURE 3-2 `get` Synchronous Operation

In a `get` synchronous operation, the manager application creates only the `LinkedReplyHandler` and a NULL is passed in place of `EndOfRepliesHandler`. Once the request is sent to the Solstice EM MIS, communication between the CORBA manager application and the SEM CORBA Gateway is blocked.

When a reply is received, it is forwarded to the CORBA manager application by executing `send_reply()`.

If there is an error in processing the request, the SEM CORBA Gateway executes either `send_subtree_error()` or `send_no_error()`, depending on the error condition.

3.2.2 Handling Replies and Errors in Client Applications

The replies sent back by the SEM CORBA Gateway in response to a request are handled by the `send_reply()` method exposed by the `LinkedReplyHandler` interface. The `send_reply()` method returns the object interface, the object Id, the time at which the command was handled, and the result of the command.

The error condition returned by the SEM CORBA Gateway is handled by the `send_subtree_error()` or the `send_mo_error()` method.

The following code example shows the implementations of `send_reply()`, `send_subtree_error()` and `send_mo_error()` for a `cmis_get()` command.

CODE EXAMPLE 3-1 Implementation of `LinkedReplyHandler`

```
// Copyright 05/11/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)get_linked_reply_handler_impl.cc1.1 99/05/11
Sun Microsystems"

#include <pmi/sys_type.hh>
#include <corba_utils/corba_utils.hh>

#include "get_linked_reply_handler_impl.hh"
#include "corba_gateway_connection.hh"

GetLinkedReplyHandlerImpl::GetLinkedReplyHandlerImpl(
) throw (
    CORBA::SystemException
) : POA_OSIMgmt::LinkedReplyHandler()
{
}

void
GetLinkedReplyHandlerImpl::send_reply(
    const char* object_interface,
    const CosNaming::Name& object_name,
```

CODE EXAMPLE 3-1 Implementation of `LinkedReplyHandler` *(Continued)*

```
    const X711CMI::ASN1_GeneralizedTimeOpt& current_time,
    const CORBA::Any& reply_info
) throw (CORBA::SystemException)
{
    cout << "get_client received a response from CORBA Gateway\n"
<< flush;

    try {
    if ((object_interface !=  NULL) && strcmp(object_interface,
""))
        cout << "Object Interface: " << object_interface << endl;

    CORBA::ULong oi_len = object_name.length();
    for (int j=0; j < oi_len; j++)
        cout << "object name[" << j << "]: " << object_name[j].id
<< endl;

    #if defined ORBACUS || defined ORBIX
    const X711CMI::GetResultAttributeListType* attr_list = new
X711CMI::GetResultAttributeListType;
    #else
    X711CMI::GetResultAttributeListType attr_list;
    #endif
    if (!(reply_info >>= attr_list)) {
        cout << "Can't extract GetResultAttributeListType\n" <<
flush;
        return;
    }

    #if defined ORBACUS || defined ORBIX
    CORBA::Long n = (*attr_list).length();
    #else
    CORBA::Long n = attr_list.length();
    #endif
    for (int i=0; i<n; i++) {
        #if defined ORBACUS || ORBIX
     X711CMI::AttributeType tmp = (*attr_list)[i];
     cout << "Attribute ID[" << i << "]: " <<
                 tmp.attributeId.globalForm() << endl << flush;

                cout << "Attribute Value [" << i << "]: "  << flush;
                   CORBA::Long max_log_size;
                   if (tmp.attributeValue >>= max_log_size)
                      cout << max_log_size << endl << flush;
                   else {
                     cerr << "FAILED to extract result from any\n";
```

CODE EXAMPLE 3-1 Implementation of `LinkedReplyHandler` *(Continued)*

```
                CORBAUtils::print_any(cout, tmp.attributeValue,
orb_, true);
                }
        #else
             X711CMI::AttributeType& tmp = attr_list[i];
             cout << "Attribute ID[" << i << "]: " <<
     tmp.attributeId.globalForm() << endl << flush;

             cout << "Attribute Value [" << i << "]: "  << flush;
                 CORBA::Long max_log_size;
             if (tmp.attributeValue >>= max_log_size)
        cout << max_log_size << endl << flush;
                 else {
                   cerr << "FAILED to extract result from any\n";
        CORBAUtils::print_any(cout, tmp.attributeValue, orb_,
true);
             }
        #endif
    }
    #if defined ORBACUS
    free((X711CMI::GetResultAttributeListType*)attr_list);
    #endif
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED: Unexpected CORBA exception:\n" << endl;
    }
    catch (...) {
        cerr << "FAILED: Unexpected exception\n";
    }
}

void
GetLinkedReplyHandlerImpl::send_subtree_error(
    const char* object_interface,
    const CosNaming::Name& object_name,
    const X711CMI::ASN1_GeneralizedTimeOpt& current_time,
    CORBA::Short error_code,
    const CORBA::Any& error_info
) throw (CORBA::SystemException)
{
    cout << "get_client received subtree error from CORBA
Gateway\n" << flush;

    try {
    if ((object_interface != NULL) && strcmp(object_interface,
""))
```

CODE EXAMPLE 3-1 Implementation of `LinkedReplyHandler` *(Continued)*

```
        cout << "Object Interface: " << object_interface << endl;

    CORBA::ULong oi_len = object_name.length();
    for (int i=0; i< oi_len; i++)
        cout << "object name[" << i << "]: " << object_name[i].id
<< endl;

    if (current_time._d() == TRUE)
        cout << "time: "<< current_time.value() << endl;

    cout << "error code: "<< error_code << endl;

    cout << "error info:\n" << flush;
    CORBAUtils::print_any(cout, error_info, orb_, true);
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED: Unexpected CORBA exception:\n" << endl;
    }
    catch (...) {
        cerr << "FAILED: Unexpected exception\n";
    }
}

void
GetLinkedReplyHandlerImpl::send_mo_error(
    const char* object_interface,
    const CosNaming::Name& object_name,
    const X711CMI::ASN1_GeneralizedTimeOpt& current_time,
    CORBA::Short error_code,
    const CORBA::Any& error_info
) throw (CORBA::SystemException)
{
   cout << "get_client received MO error from CORBA Gateway\n" <<
flush;

    try {
    if ((object_interface != NULL) && !strcmp(object_interface,
""))
        cout << "Object Interface: " << object_interface << endl;

    CORBA::ULong oi_len = object_name.length();
    for (int i=0; i< oi_len; i++)
        cout << "object name[" << i << "]: " << object_name[i].id
<< endl;

    if (current_time._d() == TRUE)
```

CODE EXAMPLE 3-1 Implementation of `LinkedReplyHandler` *(Continued)*

```
        cout << "time: "<< current_time.value() << endl;

    cout << "error code: "<< error_code << endl;

    cout << "error info:\n" << flush;
    CORBAUtils::print_any(cout, error_info, orb_, true);
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED: Unexpected CORBA exception:\n" << endl;
    }
    catch (...) {
        cerr << "FAILED: Unexpected exception\n";
    }
}

GetEndOfRepliesHandlerImpl::GetEndOfRepliesHandlerImpl(
) throw (CORBA::SystemException)
    : POA_OSIMgmt::EndOfRepliesHandler()
{
}


void
GetEndOfRepliesHandlerImpl::end_of_replies(
) throw (CORBA::SystemException)
{
    cout << "get_client received end of replies from CORBA
Gateway\n" << flush;

    try {
        CORBAGatewayConnection::disconnect();
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED: Unexpected CORBA exception:\n" << endl;
        exit(1);
    }
    catch (...) {
        cerr << "FAILED: Unexpected exception\n";
        exit(1);
    }
}
```

3.2.3 Creating Objects

The JIDM standard specifies two ways of creating managed objects:

- Using the `ManagedObjectFactory` interface
- Using the `ProxyAgent` interface

The SEM CORBA Gateway supports creating managed objects using only the `ProxyAgent` interface.

Managed objects can be created synchronously or asynchronously.

Synchronous creation of managed objects uses the `cmis_create_sync()` function of the `OSIMgmt::ProxyAgent` interface.

Asynchronous creation of managed objects uses the `cmis_create()` function of the `OSIMgmt::ProxyAgent` interface.

`cmis_create_sync()` and `cmis_create()` return the following conditions to indicate errors:

- `Duplicate Invocation`
- `Mistyped Argument`
- `Resource Limitation`
- `Unrecognized Operation`
- `Access Denied`
- `Class Instance Conflict`
- `Duplicate Managed Object Instance`
- `Invalid Attribute Value`
- `Invalid Object Instance`
- `Missing Attribute Value`
- `No Such Attribute`
- `No Such Object Class`
- `No Such Object Instance`
- `No Such Reference Object`
- `Processing Failure`
- `Processing Failure Empty`

CODE EXAMPLE 3-2 is a complete program that does exception handling for all the ORB and JIDM operations. It creates a managed object with the following instance name:

```
/systemId=name:<MIS-HOST>/emApplicationDatabase="TEST1"
```

CODE EXAMPLE 3-2 Creating Managed Objects

```
// Copyright 05/25/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)create_client.cc1.1 99/05/25 Sun
Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include <CosNaming.h>
    #include <jidm/OSIMgmt.h>
    #include <jidm_ext/ASN1TypesExt_skel.h>
#else
#ifdef ORBIX
        #include <omg/orb.hh>
        #include <CosNaming.hh>
        #include <jidm/OSIMgmt.hh>
        #include <jidm_ext/ASN1TypesExtS.hh>
#else
    #include <cos/CosNaming_c.hh>
    #include <jidm/OSIMgmt_c.hh>
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif

#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "create_linked_reply_handler_impl.hh"

int main(int argc, char **argv)
{
    if (argc < 4) {
       cout << "create_client -SVCnameroot <Root Naming Context>"
        << "  <MIS host name> [-ORBagentaddr <I/P address>/<host
name>]\n"
         << flush;
    exit(1);
    }

    CORBA::ORB_var orb;

    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
```

CODE EXAMPLE 3-2 Creating Managed Objects *(Continued)*

```
    PortableServer::POAManager_var poa_manager;


    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    //*************************************************
    // STEP NO 1: Initialize the ORB,
    //            Get NameService and root naming context
    //*************************************************
    try {
    orb = CORBA::ORB_init(argc, argv);
        cout << "PASSED: Resolving RootPOA reference" << endl;

        root_poa =  PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

        action_poa = root_poa->create_POA("action_poa", NULL,
policies);

    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: Caught CORBA Exception " << endl;
    exit(1);
    }

    CORBAGatewayConnection cgw_connection();
```

CODE EXAMPLE 3-2 Creating Managed Objects *(Continued)*

```
    //*********************************************************
    // STEP NO 2: Get ProxyAgentFinder
    //*********************************************************

    JIDM::ProxyAgentFinder_var proxy_agent_finder =
    cgw_connection.get_proxy_agent_finder();

    JIDM::ProxyAgent_var proxy_agent;
    try {
    proxy_agent = cgw_connection.get_proxy_agent();
    }
    catch (
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unable to obtain proxy agent:\n" << flush;
    }

    try {
    OSIMgmt::ProxyAgent_var osi_agent=
        OSIMgmt::ProxyAgent::_narrow(proxy_agent);


    //*********************************************************
    // STEP NO 3: Build the Managed Object name to be created
    //*********************************************************

    const char* interface_name = (const char
*)"emApplicationDatabase";

    DynamicOutputStringStream tmp_buf;
    tmp_buf << "systemId=name:'" << argv[3] << "'";

    CosNaming::Name object_name(3);
    object_name.length(3);
    object_name[0].id = (const char *)"root";
    object_name[1].id = (const char *)tmp_buf.get_string();
    object_name[2].id = (const char *)"emApplicationType='TEST1'";

    OSIMgmt::CreationKind kind = OSIMgmt::simple;

    // dummy access control
    X711CMI::AccessControlTypeOpt access_control;
    access_control._default();

    OSIMgmt::AttributeValueSeq attribute_id_list(1);
    attribute_id_list.length(1);
```

CODE EXAMPLE 3-2 Creating Managed Objects *(Continued)*

```
    ASN1_Choice oc_choice;
    oc_choice.selector = 0;
    oc_choice.value <<= (const char *)"emApplicationDatabase";

    attribute_id_list[0].attribute_id = (const char
*)"objectClass";
    attribute_id_list[0].value <<= oc_choice;

     //*******************************************************
     // STEP NO 4: Prepare reply handler
     //*******************************************************


    CreateLinkedReplyHandlerImpl reply_handler;

    cout << "\ncreate_client: LinkedReplyHandler is ready\n" <<
flush;

    action_poa->activate_object(&reply_handler);

    CosNaming::Name ref_oi(0);  // empty
    ref_oi.length(0);

    //*******************************************************
    // STEP NO 5: Build the Managed Object name to be created
    //*******************************************************

    osi_agent->cmis_create(
        interface_name,
        kind,
        object_name,
        access_control,
        ref_oi,
        attribute_id_list,
        reply_handler._this()
    );

        poa_manager->activate();
        orb->run();
    }
    catch(
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: caught exception " << endl;
    exit(1);
    }
```

CODE EXAMPLE 3-2 Creating Managed Objects *(Continued)*

```
    catch(...) {
    cerr << "FAILED: Unexpected exception " << endl;
    exit(1);
    }

    //*******************************************************
    // STEP NO 6: "release the Session" or destroy ProxyAgent
//*******************************************************

    try {
    JIDM::Criteria_var return_criteria = proxy_agent->destroy(
JIDM::ProxyAgent::gracefully, a_criteria );
     if ((JIDM::Criteria*)NULL != return_criteria) {
     cout << "PASSED : Deleted Proxy Agent gracefully" << endl;
     }
    }
    catch(...) {
    cerr << "FAILED: Unable to delete Proxy Agent gracefully\n" <<
flush;         exit(1);
    }
    return 0;
}
```

Note – <*MIS-HOST*> is passed as the third parameter to this test program. `MIS-HOST` is the host machine on which Solstice EM MIS is running, `NAME-HOST` is the host machine on which naming services is running and `NAME-PORT` is the TCP/IP port number on which the naming service on host <*NAME-HOST*> is listening for IIOP messages coming from client or manager applications.

The steps in this program are:

1. **Initializing the ORB and object adapters; Resolving the naming services and getting the root naming context.**

   Note that the SEM CORBA package provides a class called `CGWGlobal` which has static declarations for the `ORB` and `POA` attributes.

2. **Getting the reference for** `ProxyAgentFinder`**.**

3. **Building the** `key` **and** `criteria` **for getting access to the** `ProxyAgent` **which is implemented by the RGW (see Section 2.2.1 “Connecting Clients for the First Time” on page 2-5).**

   The `key` and `criteria` access parameters are critical for identifying the domain of the managed object (done by calling `access_domain()`).

   `key` consists of a `key_id` and a `key_kind`. The following key values are adopted by JIDM standards for `key_id` and `key_kind`.

   - `key_id`:
     - `"OSI Management"`
     - `"Internet Management"`
   - `key_kind`:
     - `"XSM environment"`

   `criteria` is the second parameter in the call `access_domain()` and contains the information needed to establish the session between the managed object domain and the manager. `criteria` has many components, but only *domain title* is identified as being required for all the Systems Management Reference Models.

   The following are the components of `criteria`:

   - `criteria_name`:
     - *domain title* of type `CORBA::string`
     - *controller object* of type `JIDM::ProxyAgentController`
     - *user profile* of type `CORBA::Any`

Note – This list of `criteria_name` components is for general domain access and applies to both OSI and the Internet. There are some `criteria_name` parameters that are specific to the OSI management model. A complete list can be found in Appendix C.

Note – In CODE EXAMPLE 3-2, `criteria` has two components — *domain title* and *user profile*. *domain title* is the title associated with managed objects, and is defined for Solstice EM as `"Solstice EM MIS"`.

   The `value` passed for *user profile* is the encrypted user profile. The user profile is encrypted by invoking the `AuthenticationClientHandle::encrypt_user_profile` operation.

   The *controller object* lets manager applications control the destruction of `ProxyAgent`s. The value passed in this case must be the reference of the `JIDM::ProxyAgentController`.

4. **Getting the** `ProxyAgent` **reference for the** `key` **and** `criteria` **parameters.**

5. **Creating a reply handler.**

   The details of what the reply handler class does (`CreateLinkedReplyHandlerImpl`) is not shown in the example.

6. **Deciding the name of the managed object to be created.**

7. **Calling the `OSImgmt::cmis_create` method.**

   The `cmis_create()` function requires the following input parameters:

   - `interface_name`: The interface to be exported by the newly created object.
   - `creation_kind`: The type of creation mechanism to be used; also identifies the use of the next parameter; the possible values are:
     - `simple`: Create the object named in the following parameter.
     - `autonaming`: Ignore the contents of the following parameter, and automatically assign a name for the newly created object.
     - `subordinate`: The name specified in the next parameter is the name of the superior object under which the object is to be created.
   - `object_name`: Specifies the IDL name of the managed object to be created (if `creation_kind` is `simple`) or the name of the superior object (if `creation_kind` is `subordinate`); if `creation_kind` is `autonaming`, the contents of this parameter are ignored.
   - `access_control`: This parameter of type `X711CMI::AccessControlTypeOpt` is optional and contains information to be used as input to access control functions.
   - `reference_object`: This parameter indicates the reference to a managed object needed to create the new object.
   - `req_attribute_values`: Specifies a set of attribute values to be assigned at object creation time.
   - `LinkedReplyHandler`: Interface implemented by the client or the application and a reference is passed. For more information see Section 3.2.1 “Operating Asynchronously and Synchronously” on page 3-4.

   For synchronous operation using `cmis_create_sync()` refer to the IDL file, `OSIMgmt.idl`.

8. **Releasing (terminating) the session by destroying the `ProxyAgent`.**

3.2.4 Deleting Objects

You can delete managed objects using the `cmis_delete()` function of the `OSIMgmt::ProxyAgent` interface. If you attempt to delete a non-existent object, it will result in a sub-tree error. For details see Section 3.2.1 “Operating Asynchronously and Synchronously” on page 3-4.

`cmis_delete()` returns the following conditions to indicate errors:

- `Operation Cancelled`
- `Access Denied`
- `Complexity Limitation`
- `Complexity Limitation Empty`
- `Invalid Scope`
- `No Such Object Class`
- `No Such Object Instance`
- `Processing Failure`
- `Processing Failure Empty`
- `Synchronous Not Supported`

The following code example shows how to delete a managed object asynchronously

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously

```
// Copyright 05/19/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)delete_client.cc1.1 99/05/19 Sun
Microsystems"

#include <unistd.h>
#include <iostream.h>
#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include "CosNaming.h"
        #include "jidm/OSIMgmt.h"
        #include <jidm_ext/ASN1TypesExt_skel.h>
#else
#ifdef ORBIX
        #include <omg/orb.hh>
        #include "CosNaming.hh"
        #include "jidm/OSIMgmt.hh"
        #include <jidm_ext/ASN1TypesExtS.hh>
#else
    #include "cos/CosNaming_c.hh"
    #include "jidm/OSIMgmt_c.hh"
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
```

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously *(Continued)*

```
#endif

#include "em_c++utils/dynamic_output_string_stream.hh"
#include "em_c++utils/ts_shutdown_manager.hh"

#include "corba_gateway_connection.hh"
#include "delete_linked_reply_handler_impl.hh"

int
main(
    int argc,
    char **argv
) {
    if (argc < 4) {
        cout << "delete_client -SVCnameroot <Root Naming Context>"
         << "  <MIS host name> [-ORBagentaddr <I/P address>/<host
name>]\n"
         << flush;
    exit(1);
    }

    CORBA::ORB_var orb;

    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;


    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    orb = CORBA::ORB_init(argc, argv);
        cout << "PASSED: Resolving RootPOA reference" << endl;

        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }
```

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously *(Continued)*

```
        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
        action_poa = root_poa->create_POA("action_poa", NULL,
policies);



    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED:Caught CORBA Exception  " << endl;
    exit(1);
    }

    CORBAGatewayConnection cgw_connection(
    CORBA::ORB::_duplicate(orb)//,

    );

    JIDM::ProxyAgentFinder_var proxy_agent_finder =
    cgw_connection.get_proxy_agent_finder();

    JIDM::ProxyAgent_var proxy_agent;
    try {
    proxy_agent = cgw_connection.get_proxy_agent();
    }
    catch (
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unable to obtain proxy agent:\n" << flush;
    }

    try {
    // beginning of preparing reply handler
    OSIMgmt::ProxyAgent_var osi_agent =
        OSIMgmt::ProxyAgent::_narrow(proxy_agent);

    // Issue cmis_delete()
```

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously *(Continued)*

```
    // construct oc
    const char* interface_name = (const char *)"log";

    // construct oi
    DynamicOutputStringStream tmp_buf;
    tmp_buf << "systemId=name:'" << argv[3] << "'";  // mis host
name

    CosNaming::Name object_name(3);
    object_name.length(3);
    object_name[0].id = (const char *)"root";
    object_name[1].id = (const char *)tmp_buf.get_string();
    object_name[2].id = (const char *)"logId=string:'AlarmLog'";

    // construct scope
    X711CMI::ScopeType scope;
    scope.individualLevels(1);  // NTH_LEVEL 1

    // construct filter:
    // "item : equality : {objectClass, objectCreationRecord}"

    X711CMI::CMISFilterType filter;
    X711CMI::FilterItemType item_type;
    X711CMI::AttributeType val;

    val.attributeId.globalForm((const char *)"objectClass");

    ASN1_Choice ac;
    ac.selector = 0;
    ac.value <<= (const char *)"objectCreationRecord";
    val.attributeValue <<= ac;

    item_type.equality(val);
    filter.item(item_type);

    item_type.equality(val);
    filter.item(item_type);

    // construct sync
    X711CMI::CMISSyncType sync = X711CMI::bestEffort;

    // dummy access control
    X711CMI::AccessControlTypeOpt access_control;
    access_control._default();

    DeleteLinkedReplyHandlerImpl reply_handler;
```

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously *(Continued)*

```
    // If want to test sync version of cmis_delete(), make
    // end_of_replies_handler to NULL.

    DeleteEndOfRepliesHandlerImpl end_of_replies_handler;

    cout << "\ndelete_client: LinkedReplyHandler is ready\n" <<
flush;


    action_poa->activate_object(&reply_handler);


    // If want to test sync version of cmis_delete(),
    // don't call obj_is_ready(end_of_replies_handler).

    action_poa->activate_object(&end_of_replies_handler);


    osi_agent->cmis_delete(
        interface_name,
        object_name,
        scope,
        filter,
        sync,
        access_control,
        reply_handler._this(),
        end_of_replies_handler._this()
    );

    cout << "cmis_delete() returns " << endl;


    poa_manager->activate();
        orb->run();
    }
    catch(
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: impl is interrupted" << endl;
    exit(1);
    }
    catch(...) {
    cerr << "FAILED: Unexpected exception " << endl;
    exit(1);
```

CODE EXAMPLE 3-3 Deleting Managed Objects Asynchronously *(Continued)*

```
    }
    return 0;
}
```

3.2.5 Obtaining Object Attributes

You can obtain the value or the attribute of a managed object by calling the `cmis_get()` function of the `OSIMgmt::ProxyAgent` interface.

`cmis_get()` returns the following conditions to indicate errors:

- `Get List Error`
- `Operation Cancelled`
- `Access Denied`
- `Complexity Limitation`
- `Complexity Limitation Empty`
- `Invalid Scope`
- `No Such Object Class`
- `No Such Object Instance`
- `Processing Failure`
- `Processing Failure Empty`
- `Synchronous Not Supported`
- `Duplication Invocation`
- `Mistyped Argument`
- `Resource Limitation`
- `Unrecognized Operation`

The following code example shows how to get object attributes using `cmis_get()` asynchronously.

CODE EXAMPLE 3-4 Getting Object Attributes using `cmis_get()`

```
// Copyright 05/17/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)get_client.cc1.2 99/05/17 Sun Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include "jidm/OSIMgmt.h"
```

CODE EXAMPLE 3-4 Getting Object Attributes using `cmis_get()` *(Continued)*

```
    #include "jidm_ext/ASN1TypesExt_skel.h"
#else
#ifdef ORBIX
    #include <omg/orb.hh>
    #include "jidm/OSIMgmt.hh"
    #include "jidm_ext/ASN1TypesExtS.hh"
#else
    #include "cos/CosNaming_c.hh"
    #include "jidm/OSIMgmt_c.hh"
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif

#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "get_linked_reply_handler_impl.hh"

int main(int argc, char **argv) {

    if (argc < 4) {
       cout << "get_client -SVCnameroot <Root Naming Context> <MIS
host name>"
        << "          [-ORBagentaddr <I/P address>/<host name>]\n"
         << flush;
    exit(1);
    }

    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;

    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

        cout << "PASSED: Resolving RootPOA reference" << endl;
        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));
```

CODE EXAMPLE 3-4 Getting Object Attributes using `cmis_get()` *(Continued)*

```
        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
        action_poa = root_poa->create_POA("action_poa", NULL,
policies);

    cout << "Connecting to the CORBA Gateway ...\n" << flush;
    CORBAGatewayConnection cgw_connection(
        CORBA::ORB::_duplicate(orb));

    JIDM::ProxyAgentFinder_var proxy_agent_finder =
        cgw_connection.get_proxy_agent_finder();
    JIDM::ProxyAgent_var proxy_agent =
cgw_connection.get_proxy_agent();

    OSIMgmt::ProxyAgent_var osi_agent =
        OSIMgmt::ProxyAgent::_narrow(proxy_agent);

    // Issue cmis_get()
    // construct oc
    DynamicOutputStringStream tmp_buf;
    tmp_buf << "systemId=name:'" << "gemini" << "'";  // mis host
name

    const char* interface_name = (const char *)"log";

    // construct oi
    CosNaming::Name object_name(3);
    object_name.length(3);
    object_name[0].id = (const char *)"root";
    object_name[1].id = (const char *)tmp_buf.get_string();
    object_name[2].id = (const char *)"logId=string:'AlarmLog'";
```

CODE EXAMPLE 3-4 Getting Object Attributes using `cmis_get()` *(Continued)*

```
    // construct scope
    X711CMI::ScopeType scope;
    scope.individualLevels(0);

    //  construct filter: "item : equality : {objectClass, log}"
    X711CMI::CMISFilterType filter;
    X711CMI::FilterItemType item_type;
    X711CMI::AttributeType val;

    val.attributeId.globalForm((const char *)"objectClass");

    ASN1_Choice ach;
    ach.selector = 0;
    ach.value <<= (const char *)"log";
    val.attributeValue <<= ach;

    item_type.equality(val);
    filter.item(item_type);

    // construct sync
    X711CMI::CMISSyncType sync = X711CMI::bestEffort;

    // dummy access control
    X711CMI::AccessControlTypeOpt access_control;
    access_control._default();

    // construct attribute_id_list
        OSIMgmt::ASN1_ObjectIdentifierSeq attribute_id_list(0);
        attribute_id_list.length(0);
    #if 0
    OSIMgmt::ASN1_ObjectIdentifierSeq attribute_id_list(1);
    attribute_id_list.length(1);
    attribute_id_list[0] = (const char *)"maxLogSize";
    #endif

    GetLinkedReplyHandlerImpl reply_handler;
    GetEndOfRepliesHandlerImpl end_of_replies_handler;

    cout << "\nget_client: LinkedReplyHandler is ready\n" <<
flush;

    action_poa->activate_object(&reply_handler);
    action_poa->activate_object(&end_of_replies_handler);

    osi_agent->cmis_get(
        interface_name,
```

CODE EXAMPLE 3-4 Getting Object Attributes using `cmis_get()` *(Continued)*

```
        object_name,
        scope,
        filter,
        sync,
        access_control,
        attribute_id_list,
        reply_handler._this(),
        end_of_replies_handler._this());

    cout << "get_client has sent the M-Get request to CORBA
Gateway\n"
         << flush;

    poa_manager->activate();
        orb->run();
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: \n" << endl;
    exit(1);
    }
    catch (...) {
    cerr << "FAILED: Unexpected exception caught\n";
    exit(1);
    }
    return 0;
}
```

The following code example shows how to get object attributes using `cmis_get_text` asynchronously.

CODE EXAMPLE 3-5 Getting Object Attributes using `cmis_get_text()`

```
// Copyright 06/01/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)get_text_client.cc1.1 99/06/01 Sun
Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include <jidm_ext/ASN1TypesExt_skel.h>
#else
#ifdef ORBIX
```

CODE EXAMPLE 3-5 Getting Object Attributes using `cmis_get_text()` *(Continued)*

```
    #include <omg/orb.hh>
    #include <jidm_ext/ASN1TypesExtS.hh>
#else
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif

#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "get_text_linked_reply_handler_impl.hh"

//
*****************************************************************
*******
// Usage:
// get_text_client -SVCnameroot <Root Naming Context> <MIS host
name>
//                [-ORBagentaddr <I/P address>/<host name>]
//
// get_text_client sample program starts a connection with the
CORBA Gateway,
// obtains a proxy_agent
// It sends a cmis get command to get the maxLogSize attribute
value of the
// alarmLog.
//
*****************************************************************
*******

int
main(
    int argc,
    char **argv
) {
    if (argc < 4) {
      cout << "get_text_client -SVCnameroot <Root Naming Context>"
        << "  <MIS host name> [-ORBagentaddr <I/P address>/<host
name>]\n"
         << flush;
    exit(1);
    }

    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
```

CODE EXAMPLE 3-5 Getting Object Attributes using `cmis_get_text()` *(Continued)*

```
    PortableServer::POAManager_var poa_manager;


    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

        cout << "PASSED: Resolving RootPOA reference" << endl;
        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
        action_poa = root_poa->create_POA("action_poa", NULL,
policies);


    cout << "Connecting to the CORBA Gateway ...\n" << flush;
    CORBAGatewayConnection cgw_connection(
        CORBA::ORB::_duplicate(orb)
    );

    JIDM::ProxyAgentFinder_var proxy_agent_finder =
        cgw_connection.get_proxy_agent_finder();

    JIDM::ProxyAgent_var proxy_agent =
cgw_connection.get_proxy_agent();
```

CODE EXAMPLE 3-5 Getting Object Attributes using `cmis_get_text()` *(Continued)*

```
    OSIMgmtExt::ProxyAgent_var osi_agent =
        OSIMgmtExt::ProxyAgent::_narrow(proxy_agent);

    // Issue cmis_get_text()
    // construct oc
    const char* interface_name = (const char *)"log";

    // construct oi
    DynamicOutputStringStream object_name;
    object_name << "/systemId=name:'" << argv[3]
        << "'/logId=string:'AlarmLog'";

    // construct scope
    X711CMI::ScopeType scope;
    scope.individualLevels(0);

    // construct filter: "item : equality : {objectClass, log}"
    CMIExt::CMISFilterType filter = CORBA::string_dup("and : {
}");

    X711CMI::CMISSyncType sync = X711CMI::bestEffort;

    // construct attribute_id_list
        OSIMgmt::ASN1_ObjectIdentifierSeq attribute_id_list(0);
        attribute_id_list.length(0);

    GetTextLinkedReplyHandlerImpl reply_handler;
    GetEndOfRepliesHandlerImpl end_of_replies_handler;

    cout << "get_text_client: LinkedReplyHandler is ready\n" <<
flush;

    action_poa->activate_object(&reply_handler);
    action_poa->activate_object(&end_of_replies_handler);

    cout << "BEFORE CMIS_GET_TEXT " << endl;
    osi_agent->cmis_get_text(
        interface_name,
        object_name.get_string(),
        scope,
        filter,
        sync,
        attribute_id_list,
        reply_handler._this(),
        end_of_replies_handler._this()
```

CODE EXAMPLE 3-5 Getting Object Attributes using `cmis_get_text()` *(Continued)*

```
    );

    cout << " AFTER CMIS_GET_TEXT " << endl;

    cout << "get_text_client has sent the M-Get(text) request to
CORBA"
         << "Gateway\n" << flush;

    poa_manager->activate();
        orb->run();
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: \n" << endl;
    exit(1);
    }
    catch (...) {
    cerr << "FAILED: Unexpected exception caught\n";
    exit(1);
    }
    return 0;
}
```

3.2.6 Obtaining Multiple Object Attributes

You can obtain multiple attributes from the RGW by specifying the scope and the filtering to be applied when calling the CMIS functions. You can also get multiple attributes by using `attribute_id_list` in a special way, as discussed later in this section. You can also define the interaction characteristics of selected objects (through scoping and filtering) by specifying synchronization and access control parameters.

3.2.6.1 Selecting Objects Through Scoping and Filtering

To specify the set of managed objects on which the operation is to be applied, select the appropriate integer value for the scope option:

- A value of 0 specifies the base object alone.
- A value of 1 specifies the first level only.
- A value of 2 specifies the whole subtree.

To specify the set of filtering tests to be performed on the set of managed objects (selected by applying scope) use the filter option. Group multiple tests together using combinations of AND, OR and NOT.

You can test for the following filter conditions:

- equality
- greater than or equal to
- less than or equal to
- presence
- subset of
- superset of
- non-Null intersection
- substring

If the managed object is not selected by the scope specified, the filter assertion test for that managed object evaluates to FALSE, and the managed object is not chosen for that operation; if no filter is specified, however, all of the managed objects included by the defined scope are selected for the operation.

3.2.6.2 Synchronization and Access Control

The synchronization parameter has two possible values, `Best Effort` and `Atomic`. `Best Effort` synchronization only requires that all the managed objects chosen (through filtering and scoping), be requested to perform the operation without any guarantee of a successful response. `Atomic` synchronization ensures that all the managed objects chosen (through scoping and filtering) are able to successfully perform the operation together—if one or more objects cannot perform the operation, then none of them are requested to perform it.

If only the base object is selected by the scope for the operation, the synchronization parameter is ignored.

The access control option specifies the functions to be used for access control. If this optional parameter (as specified in the JIDM standards) is not specified, the access control parameters specified at the time of `ProxyAgent` creation are used instead. At present the SEM CORBA Gateway implementation only supports access control through `ProxyAgent`.

3.2.6.3 `attribute_id_list` parameters

You can use the `attribute_id_list` parameters available for some CMIS operations (such as `get`) to enable clients to implement scope operations. This list is a sequence of `ASN1_ObjectIdentifier`s of the managed objects on which the operation is to be carried out. If this list is empty, all the managed objects under the current base object will be fetched.

The following code segment shows how to use a `get` operation to read all the attributes under "`systemId=name:faith/logId=string:AlarmLog`".

CODE EXAMPLE 3-6 Obtaining Multiple Object Attributes

```
DynamicOutputStringStream tmp_buf;
    CosNaming::Name object_name(3);
    object_name.length(3);
    object_name[0].id = (const char *)"root";
    tmp_buf << "systemId=name:'" << "faith" << "'";
    object_name[1].id = (const char *)tmp_buf.get_string();
    object_name[2].id = (const char *)"logId=string:'AlarmLog'";

    // construct scope
    X711CMI::ScopeType scope;
    // Whole substree
    scope.level(2);

    X711CMI::CMISFilterType filter;
    X711CMI::FilterItemType item_type;
    X711CMI::AttributeType val;

    // Construct a null filter
    _and_seq filter_seq(0);
    filter_seq.length(0);
    filter._cxx_and(filter_seq);

    // construct sync
    X711CMI::CMISSyncType sync = X711CMI::bestEffort;

    // dummy access control
    X711CMI::AccessControlTypeOpt access_control;
    access_control._default();

    // construct attribute_id_list
    // We want to get all the attributes under the subtree, so set att
list = 0
    OSIMgmt::ASN1_ObjectIdentifierSeq attribute_id_list(0);
    attribute_id_list.length(0);

   // Rest of the code is not shown in this code segment
```

3.2.7 Modifying Object Attributes

You can modify or set the managed object attributes using the `cmis_set()` function exposed by `OSIMgmt::ProxyAgent` interface. CODE EXAMPLE 3-7 shows how to set or modify attributes of an object.

`cmis_set()` returns the following conditions to indicate errors:

- `Operation Cancelled`
- `Access Denied`
- `Complexity Limitation`
- `Complexity Limitation Empty`
- `Invalid Scope`
- `No Such Object Class`
- `No Such Object Instance`
- `Processing Failure`
- `Processing Failure Empty`
- `Synchronous Not Supported`
- `Set List Error`

CODE EXAMPLE 3-7 Modifying Object Attributes

```
// Copyright 05/17/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)set_client.cc1.1 99/05/17 Sun Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include "CosNaming.h"
    #include "jidm/OSIMgmt.h"
    #include "jidm_ext/ASN1TypesExt_skel.h"
#else
#ifdef ORBIX
    #include <omg/orb.hh>
    #include "CosNaming.hh"
    #include "jidm/OSIMgmt.hh"
    #include "jidm_ext/ASN1TypesExtS.hh"
#else
    #include "cos/CosNaming_c.hh"
    #include "jidm/OSIMgmt_c.hh"
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif
```

CODE EXAMPLE 3-7 Modifying Object Attributes *(Continued)*

```
#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "set_linked_reply_handler_impl.hh"

int
main(
    int argc,
    char **argv
) {
    if (argc < 4) {
       cout << "get_client -SVCnameroot <Root Naming Context> <MIS
host name>"
        << "          [-ORBagentaddr <I/P address>/<host name>]\n"
         << flush;
        exit(1);
    }

    CORBA::ORB_var orb;


    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;


    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    orb = CORBA::ORB_init(argc, argv);
        cout << "PASSED: Resolving RootPOA reference" << endl;

        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();
```

CODE EXAMPLE 3-7 Modifying Object Attributes *(Continued)*

```
        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
        action_poa = root_poa->create_POA("action_poa", NULL,
policies);

    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: " << endl;
    exit(1);
    }

    CORBAGatewayConnection cgw_connection(
    CORBA::ORB::_duplicate(orb)//,

    );

    JIDM::ProxyAgentFinder_var proxy_agent_finder;
    JIDM::ProxyAgent_var proxy_agent;
    try {
    proxy_agent_finder = cgw_connection.get_proxy_agent_finder();
    proxy_agent = cgw_connection.get_proxy_agent();
    }
    catch (
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unable to obtain proxy agent:\n" << flush;
    }

    try {
    // beginning of preparing reply handler
    OSIMgmt::ProxyAgent_var osi_agent =
        OSIMgmt::ProxyAgent::_narrow(proxy_agent);

    // construct oc
    DynamicOutputStringStream tmp_buf;
    tmp_buf << "systemId=name:'" << argv[3] << "'";  // mis host
name
```

CODE EXAMPLE 3-7 Modifying Object Attributes *(Continued)*

```
const char* interface_name = (const char *)"log";

// construct oi
CosNaming::Name object_name(3);
object_name.length(3);
object_name[0].id = (const char *)"root";
object_name[1].id = (const char *)tmp_buf.get_string();
object_name[2].id = (const char *)"logId=string:'AlarmLog'";

// construct scope
X711CMI::ScopeType scope;
scope.individualLevels(0);

//
//  construct filter: "item : equality : {objectClass, log}"
//
X711CMI::CMISFilterType filter;
X711CMI::FilterItemType item_type;
X711CMI::AttributeType val;

val.attributeId.globalForm((const char *)"objectClass");

ASN1_Choice ach;
ach.selector = 0;
ach.value <<= (const char *)"log";
val.attributeValue <<= ach;

item_type.equality(val);
filter.item(item_type);

// construct sync
X711CMI::CMISSyncType sync = X711CMI::bestEffort;

// dummy access control
X711CMI::AccessControlTypeOpt access_control;
access_control._default();

// construct modify_list
OSIMgmt::SetOperationArgument modify_list(1);
modify_list.length(1);
modify_list[0].modify_operator = OSIMgmt::replace;
modify_list[0].attribute_id = (const char*)"maxLogSize";
modify_list[0].attribute_value <<= (CORBA::Long)900000;

SetLinkedReplyHandlerImpl reply_handler;
SetEndOfRepliesHandlerImpl end_of_replies_handler;
```

CODE EXAMPLE 3-7 Modifying Object Attributes *(Continued)*

```
    action_poa->activate_object(&reply_handler);
    action_poa->activate_object(&end_of_replies_handler);

    cout << "\nset_client: LinkedReplyHandler is ready\n" <<
flush;

    osi_agent->cmis_set(
        interface_name,
        object_name,
        scope,
        filter,
        sync,
        access_control,
        modify_list,
        reply_handler._this(),
        end_of_replies_handler._this()
    );

    cout << "set_client has sent the M-Set request to CORBA
Gateway\n"
         << flush;

    poa_manager->activate();
        orb->run();
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: Unexpected CORBA Exception" << endl;
    exit(1);
    }
}
```

3.2.8 Performing an Operation on a Managed Object

You can use the `cmis_action()` function of the `OSIMgmt::ProxyAgent` interface to perform a certain action on a managed object and indicate the result of the action back to the client.

`cmis_action()` returns the following conditions to indicate errors:

- `Operation Cancelled`
- `Access Denied`
- `Complexity Limitation`
- `Complexity Limitation Empty`
- `Invalid Scope`
- `No Such Object Class`
- `No Such Object Instance`
- `Processing Failure`
- `Processing Failure Empty`
- `Synchronous Not Supported`
- `Invalid Argument Value`
- `No Such Action`
- `No Such Argument`

The following code example shows how to perform an operation on a managed object.

CODE EXAMPLE 3-8 Performing an Operation on a Managed Object

```
// Copyright 05/25/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)action_client.cc1.1 99/05/25 Sun
Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include <CosNaming.h>
        #include <jidm/OSIMgmt.h>
        #include "jidm/X711CMI.h"
        #include <jidm_ext/ASN1TypesExt_skel.h>
#else
#ifdef ORBIX
        #include <omg/orb.hh>
        #include <CosNaming.hh>
        #include <jidm/OSIMgmt.hh>
        #include "jidm/X711CMI.hh"
```

CODE EXAMPLE 3-8 Performing an Operation on a Managed Object *(Continued)*

```
        #include <jidm_ext/ASN1TypesExtS.hh>
#else
    #include <cos/CosNaming_c.hh>
    #include <jidm/OSIMgmt_c.hh>
    #include "jidm/X711CMI_c.hh"
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif
#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "action_linked_reply_handler_impl.hh"

int main(
    int argc,
    char **argv
) {
    if (argc < 5) {
        cout << "action_client -SVCnameroot <Root Naming Context>"
         << "  <MIS host name> <Device Name>"
         << "  [-ORBagentaddr <I/P address>/<host name>]\n"
         << flush;
        exit(1);
    }

    CORBA::ORB_var orb;

    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;


    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    orb = CORBA::ORB_init(argc, argv);
        cout << "PASSED: Resolving RootPOA reference" << endl;

        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));
```

CODE EXAMPLE 3-8 Performing an Operation on a Managed Object *(Continued)*

```
        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
    CORBA::PolicyList policies;
    policies.length(1);
    policies[(CORBA::ULong)0] =
    root_poa->create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
    action_poa = root_poa->create_POA("action_poa", NULL,
policies);

    }
    catch(
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Caught CORBA Exception " << endl;
    exit(1);
    }

    CORBAGatewayConnection cgw_connection(
    CORBA::ORB::_duplicate(orb)
    );

    JIDM::ProxyAgentFinder_var proxy_agent_finder;
    JIDM::ProxyAgent_var proxy_agent;
    try {
    proxy_agent_finder = cgw_connection.get_proxy_agent_finder();
    proxy_agent = cgw_connection.get_proxy_agent();
    }
    catch (
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unable to obtain proxy agent:\n" << flush;
    }

    try {
    // beginning of preparing reply handler
    OSIMgmt::ProxyAgent_var osi_agent =
OSIMgmt::ProxyAgent::_narrow(proxy_agent);
```

CODE EXAMPLE 3-8 Performing an Operation on a Managed Object *(Continued)*

```
// Issue cmis_action()
// construct oc
const char* interface_name = (const char *)"topoNodeDB";

// construct oi
DynamicOutputStringStream mis_host_name;
mis_host_name << "systemId=name:'" << argv[3] << "'";

CosNaming::Name object_name(3);
object_name.length(3);
object_name[0].id = (const char *)"root";
object_name[1].id = (const char *)mis_host_name.get_string();
object_name[2].id = (const char *)"topoNodeDBId=NULL";

// construct scope
X711CMI::ScopeType scope;
scope.level(0);  // BASE_OBJECT

// construct filter: default
X711CMI::CMISFilterType filter;
//_and_seq filter_seq(0);
//filter_seq.length(0);
//filter._cxx_and(filter_seq);

// construct sync
X711CMI::CMISSyncType sync = X711CMI::bestEffort;

// dummy access control
X711CMI::AccessControlTypeOpt access_control;
access_control._default();

// construct action name - topoNodeGetByName
ASN1_ObjectIdentifier action_name =
    CORBA::string_dup("topoNodeGetByName");

cout << "Device Type: " << argv[4] << endl;

// construct action info
ASN1_DefinedAny action_info;
action_info <<= (const char*)argv[4];

ActionLinkedReplyHandlerImpl reply_handler;
ActionEndOfRepliesHandlerImpl end_of_replies_handler;

action_poa->activate_object(&reply_handler);
```

CODE EXAMPLE 3-8 Performing an Operation on a Managed Object *(Continued)*

```
    action_poa->activate_object(&end_of_replies_handler);

    cout << "\n action_client: LinkedReplyHandler is ready\n" <<
flush;

    osi_agent->cmis_action(
        interface_name,
        object_name,
        scope,
        filter,
        sync,
        access_control,
        action_name,
        action_info,
        reply_handler._this(),
        end_of_replies_handler._this()
    );

    poa_manager->activate();
    orb->run();
    }
    catch(
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unexpected CORBA Exception" << endl;
    exit(1);
    }
}
```

3.2.9 Cancelling a Request

To cancel a request that has already been sent, delete or deactivate the `LinkedReplyHandler` interface before the reply is received. The SEM CORBA Gateway that is processing the request will only send a reply if the `LinkedReplyHandler` interface still exists. If the `LinkedReplyHandler` interface does not exist, the gateway assumes that the request has been cancelled.

The following code example shows how to cancel a request that has already been sent.

CODE EXAMPLE 3-9 Cancelling a Request

```
// Copyright 06/01/99 Sun Microsystems, Inc. All Rights Reserved.

#pragma ident  "@(#)cancel_get_client.cc1.2 99/06/01 Sun
Microsystems"

#include <unistd.h>
#include <iostream.h>

#ifdef ORBACUS
    #include <OB/CORBA.h>
    #include "jidm/OSIMgmt.h"
    #include <jidm_ext/ASN1TypesExt_skel.h>
#else
#ifdef ORBIX
    #include <omg/orb.hh>
    #include "jidm/OSIMgmt.hh"
    #include <jidm_ext/ASN1TypesExtS.hh>
#else
    #include "jidm/OSIMgmt_c.hh"
    #include <jidm_ext/ASN1TypesExt_s.hh>
#endif
#endif

#include <pthread.h>

#include <em_c++utils/dynamic_output_string_stream.hh>
#include <em_c++utils/ts_shutdown_manager.hh>

#include "corba_gateway_connection.hh"
#include "cancel_get_linked_reply_handler_impl.hh"

CORBA::ORB_var orb;

    PortableServer::POA_var root_poa;
```

CODE EXAMPLE 3-9 Cancelling a Request *(Continued)*

```
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;
    PortableServer::ObjectId_var oid;
    PortableServer::ObjectId_var oid1;


void*
deactivate_reply_handler(
    void*
) {
    cout << "Cancelling get request...\n" << flush;

    action_poa->deactivate_object(oid);
    //action_poa->deactivate_object(oid1);

    return NULL;
}


int
main(
    int argc,
    char **argv
) {
    if (argc < 4) {
        cout << "cancel_get_client -SVCnameroot <Root Naming
Context>"
         << "  <MIS host name> [-ORBagentaddr <I/P address>/<host
name>]\n"
          << flush;
    exit(1);
    }
/*
    PortableServer::POA_var root_poa;
    PortableServer::POA_var action_poa;
    PortableServer::POAManager_var poa_manager;
*/
    TSShutdownManager& shutdown_mgr =
TSShutdownManager::instance();
    SampleShutdownCallback* sample_shutdown_cb = new
SampleShutdownCallback();
    shutdown_mgr.add_callback(sample_shutdown_cb);

    try {
    orb = CORBA::ORB_init(argc, argv);
        cout << "PASSED: Resolving RootPOA reference" << endl;
```

CODE EXAMPLE 3-9 Cancelling a Request *(Continued)*

```
        root_poa = PortableServer::POA::_narrow(orb-
>resolve_initial_references("RootPOA"));

        if (CORBA::is_nil(root_poa)) {
          cout << "Unable to get RootPOA context!!" << endl;
          exit(1);
        }

        poa_manager = root_poa->the_POAManager();

        // Create policies for our action POA
        CORBA::PolicyList policies;
        policies.length(1);
        policies[(CORBA::ULong)0] =
        root_poa-
>create_lifespan_policy(PortableServer::TRANSIENT);

       // Create the action servant and activate it on action_poa
        // Create action poa with our own policies
        action_poa = root_poa->create_POA("action_poa", NULL,
policies);

    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: Caught CORBA Exception " << endl;
    exit(1);
    }

    CORBAGatewayConnection cgw_connection(
    CORBA::ORB::_duplicate(orb)//,

    );

    JIDM::ProxyAgentFinder_var proxy_agent_finder;
    JIDM::ProxyAgent_var proxy_agent;
    try {
    proxy_agent_finder = cgw_connection.get_proxy_agent_finder();
    proxy_agent = cgw_connection.get_proxy_agent();
    }
    catch (
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unable to obtain proxy agent:\n" << flush;
    }
```

CODE EXAMPLE 3-9 Cancelling a Request *(Continued)*

```
try {
// beginning of preparing reply handler
OSIMgmt::ProxyAgent_var osi_agent =
    OSIMgmt::ProxyAgent::_narrow(proxy_agent);

// construct oc
const char* interface_name = (const char *)"log";

// construct oi
DynamicOutputStringStream tmp_buf;

CosNaming::Name object_name(3);
object_name.length(3);
object_name[0].id = (const char *)"root";
tmp_buf << "systemId=name:'" << argv[3] << "'";
object_name[1].id = (const char *)tmp_buf.get_string();
object_name[2].id = (const char *)"logId=string:'AlarmLog'";

// construct scope
X711CMI::ScopeType scope;
scope.individualLevels(1);

//
// construct filter:
// "item : equality : {objectClass, nerveCenterAlarmRecord}"
//
X711CMI::CMISFilterType filter;
X711CMI::FilterItemType item_type;
X711CMI::AttributeType val;

val.attributeId.globalForm((const char *)"objectClass");

ASN1_Choice ach;
ach.selector = 0;
ach.value <<= (const char *)"emAlarmRecord";
val.attributeValue <<= ach;

item_type.equality(val);
filter.item(item_type);

// construct sync
X711CMI::CMISSyncType sync = X711CMI::bestEffort;

// dummy access control
X711CMI::AccessControlTypeOpt access_control;
```

CODE EXAMPLE 3-9 Cancelling a Request *(Continued)*

```
    access_control._default();

    // construct attribute_id_list
    OSIMgmt::ASN1_ObjectIdentifierSeq attribute_id_list(1);
    attribute_id_list.length(1);
    attribute_id_list[0] = (const char *)"logRecordId";

        GetLinkedReplyHandlerImpl reply_handler;
        GetEndOfRepliesHandlerImpl end_of_replies_handler;


    oid = action_poa->activate_object(&reply_handler);
    oid1 = action_poa->activate_object(&end_of_replies_handler);

    cout << "\ncancel_get_client: LinkedReplyHandler is ready\n"
<< flush;

    osi_agent->cmis_get(
        interface_name,
        object_name,
        scope,
        filter,
        sync,
        access_control,
        attribute_id_list,
        reply_handler._this(),
        end_of_replies_handler._this()
    );

    cout << "cancel_get_client has sent the M-Get request to CORBA
Gateway"
         << endl << flush;

    pthread_t tid;
    if(pthread_create(&tid, NULL, deactivate_reply_handler,
NULL)) {
        cerr << "FAILED: unable to start
deactivate_reply_handler\n"
     << flush;
    }

        poa_manager->activate();
        orb->run();

    }
    catch(const CORBA::Exception& e) {
```

CODE EXAMPLE 3-9 Cancelling a Request *(Continued)*

```
    cerr << "FAILED: Unexpected CORBA Exception" << endl;
    exit(1);
    }
}
```

3.2.10 Subscribing to an Event

Applications that collect events need to subscribe to events as shown in CODE EXAMPLE 3-10. The SEM CORBA Gateway provides an OSIMgmt Extended `ProxyAgent` (`OSIMgmtExt::ProxyAgent`) that includes the functions `subscribe_events()` and `unsubscribe_events()`.

`subscribe_events()` creates Event Forwarding Discriminators (EFDs) using `cmis_create_text`, and returns the subscription Id.

`subscribe_events()` returns the following conditions to indicate errors:

- `Access Denied`
- `Processing Failure`

`unsubscribe_events()` deletes the EFDs (using `cmis_delete_text`) and returns the following conditions to indicate errors:

- `Access Denied`
- `Processing Failure`
- `Invalid Subscription`

A pictorial representation of the steps involved in subscribing to an event is given in the following figure.

]

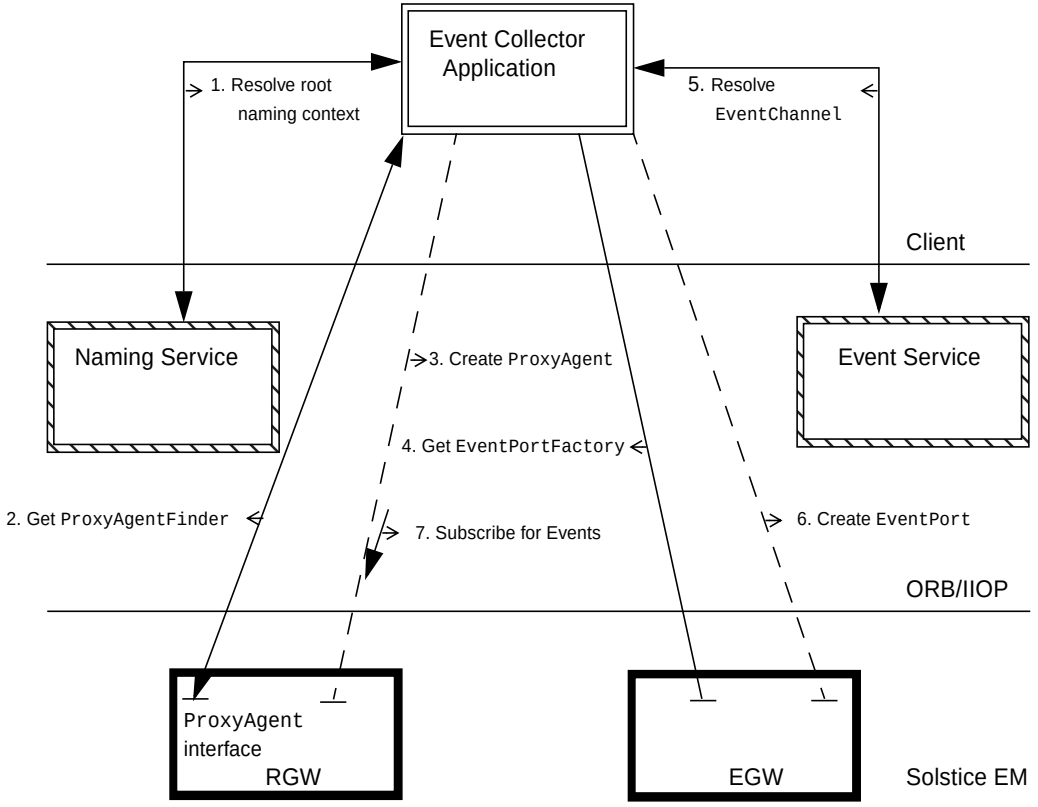


FIGURE 3-3 Subscribing to an Event

CODE EXAMPLE 3-10 Subscribing to an Event

```
// Copyright 06/01/99 Sun Microsystems, Inc. All Rights Reserved.
#pragma ident  "@(#)subscribe_events.cc1.299/06/01SunMicrosystems"

#include <unistd.h>
#include <iostream.h>

#include <cos/CosNaming_c.hh>
#include <jidm_ext/OSIMgmtExt_c.hh>

#include <auth_helper/auth_client_handle.hh>
#include <corba_utils/corba_utils.hh>
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
#include <jidm_ext/ASN1TypesExt_c.hh>
#include <em_c++utils/dynamic_output_string_stream.hh>

/*******************************************************
How to run this test program:
----------------------------

subscribe_events  -a <ae_title> -SVCnameroot Quake

*******************************************************/

int
main(
    int argc,
    char **argv
) {
    CORBA::ORB_var orb;

    if (argc < 5) {
        cout << argv[0] << " -SVCnameroot Quake -a <ae_title>\n";
        exit(1);
    }

    int c;
    extern int optind;
    extern char* optarg;
    const char* ae_title;

    while ((c = getopt(argc, argv, "a:S:")) != EOF) {
    switch (c) {
        case 'a':
    ae_title = CORBA::string_dup(optarg);
    break;

        case 'S':
    break;

        case '?':
        default:
        cerr << "Usage: " << argv[0] <<
    " -a <OID-AE-title> -SVCnameroot <default root naming context>"
    << endl;
    return 1;
    }
    }
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
    try {
    orb = CORBA::ORB_init(argc, argv);
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: to initialize the ORB" << e;
    exit(1);
    }

    cout << "PASSED: ORB initialization\n" << flush;

    CosNaming::NamingContext_var root_nc;
    try{
    CORBA::Object_var object =
        orb->resolve_initial_references("NameService");
    root_nc = CosNaming::NamingContext::_narrow(object);

    if(CORBA::is_nil(root_nc)) {
       cerr << "FAILED: Unable to obtain root naming context\n" <<
flush;
        exit(2);
    }
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: Unable to obtain root naming context:\n"<<e;
    exit(3);
    }

    cout << "PASSED: Obtained root naming context\n" << flush;

    CosNaming::Name name;
    name.length(1);
    JIDM::ProxyAgentFinder_var proxy_agent_finder;
    try{
    name[0].id = CORBA::string_dup("JIDM::ProxyAgentFinder");
    name[0].kind = CORBA::string_dup("");

    CORBA::Object_var object = root_nc->resolve(name);

    proxy_agent_finder = JIDM::ProxyAgentFinder::_narrow(object);

    if (CORBA::is_nil(proxy_agent_finder)) {
      cerr << "FAILED: Unable to obtain correct Proxy Agent Finder\n"
     << flush;
        exit(4);
    }
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
    if(proxy_agent_finder->_non_existent()) {
      cerr << "FAILED: Proxy Agent Finder does not exist\n" << flush;
       exit(5);
    }
    }
    catch(const CORBA::Exception& e) {
    cerr << "FAILED: Unable to resolve JIDM::ProxyAgentFinder:\n" <<
e;
    exit(6);
    }
   cout << "PASSED: Obtained JIDM::ProxyAgentFinder reference\n" <<
flush;

    JIDM::Key a_key;
    a_key.length(1);
    a_key[0].id = CORBA::string_dup("OSI Management");
    a_key[0].kind = CORBA::string_dup("XSM environment");

    JIDM::Criteria a_criteria;
    a_criteria.length(2);
    a_criteria[0].name = CORBA::string_dup("domain title");
    a_criteria[0].value <<= (const char*)ae_title;

    cout << "Enter User Name: " << flush;
    char user_name[128];

    cin >> user_name;

    const char* password_prompt = "Enter Password: ";
  // getpassphrase return a pointer to static data which should not be
    // deleted
    char* raw_password = getpassphrase(password_prompt);

    AuthenticationClient* ac = new
AuthenticationClientHandle("NBS_DES");

    a_criteria[1].name = CORBA::string_dup("user_profile");
    a_criteria[1].value = *(ac->encrypt_user_profile(
           user_name,
           raw_password,
           NULL));

  // This will ensure that pass won't retain the raw password if process
    // dumps core
    memset(raw_password, 0x0c, sizeof(raw_password));
    delete ac;
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
    if(a_criteria[1].value.type()->kind() == CORBA::tk_null) {
    cerr << "FAILED: Unable to obtain encrypted user profile\n"
         << flush;
    exit(1);
    }

   cout << "PASSED: Obtained criteria from encrypting user profile\n"
     << flush;

    JIDM::ProxyAgent_var proxy_agent;
    try {
    cout << flush;
    proxy_agent = proxy_agent_finder->access_domain(a_key,
a_criteria);

    if (CORBA::is_nil(proxy_agent)) {
        cerr << "FAILED: Unable to obtain correct Proxy Agent"
     << endl << flush;
        throw 0;
    }

    if(proxy_agent->_non_existent()) {
       cerr << "FAILED: Proxy Agent does not exist" << endl << flush;
        throw 0;
    }
    }
    catch(const CORBA::UserException& e) {
    cerr << "FAILED: to get proxy_agent\n" << e << endl;
    exit(1);
    }
    catch(
    const CORBA::Exception& e
    ) {
    cerr << "FAILED: Unexcepted exception when trying to get
proxy_agent\n";
    exit(1);
    }

    cout << "PASSED: Created a new Proxy Agent\n" << flush;

    OSIMgmtExt::ProxyAgent_var osi_agent=
        OSIMgmtExt::ProxyAgent::_narrow(proxy_agent);

    StringSeq event_list;
#if 0
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
    // For all events: equivalent to and :{}
    event_list.length(0);
#endif

    event_list.length(3);

    event_list[0] = CORBA::string_dup(
    "'Rec. X.721 | ISO/IEC 10165-2 : 1992': objectCreation");
    event_list[1] = CORBA::string_dup(
    "'Rec. X.721 | ISO/IEC 10165-2 : 1992': objectDeletion");
    event_list[2] = CORBA::string_dup(
    "'Rec. X.721 | ISO/IEC 10165-2 : 1992': attributeValueChange");

    StringSeq object_class_list;
    object_class_list.length(0);

    StringSeq object_name_list;
    object_name_list.length(0);

    OSIMgmtExt::ProxyAgent::SubscriptionId_var sid;
    try {
    sid = osi_agent->subscribe_events(
        event_list,
        object_class_list,
        object_name_list
    );
    cout << "Subscription ID = " << sid << endl;
    cout << endl << " ++++++++++++++++++++++++++++++" << endl;
    }
    catch (const CORBA::Exception& e) {
    cout << "subscribe_events() failed: " << e << endl;
    return 1;
    }

    cout << "Hit return to continue\n";
    char reply[3];
    cin >> reply;
    cout << "Now trying to unsubsribe for events...\n" << flush;

    osi_agent->unsubscribe_events(sid);
    cout << "Unsubsribed the previously subscribed event\n";

    try {
    JIDM::Criteria_var return_criteria = proxy_agent->destroy(
        JIDM::ProxyAgent::gracefully,
        a_criteria
```

CODE EXAMPLE 3-10 Subscribing to an Event *(Continued)*

```
    );
    if ((JIDM::Criteria*)NULL != return_criteria) {
        cout << "PASSED : Deleted Proxy Agent gracefully" << endl;
    }
    }
    catch(...) {
    cerr << "FAILED: Unable to delete Proxy Agent gracefully\n" <<
flush;
    exit(1);
    }

    return 0;
}
```

CHAPTER 4

Handling Events With SEM CORBA Gateway

This chapter discusses how to handle events with the SEM CORBA Gateway.

This chapter describes the following topics:

The SEM CORBA Event Gateway (EGW) is a conceptual entity comprising more than one UNIX process. The SEM CORBA Gateway consists of interfaces for registering a CORBA `EventChannel`, and an event distribution mechanism or CORBA-enabled Event Distribution Server (EDS) sink. The mechanism for subscribing to and unsubscribing from events is provided by the RGW.

The main component of the EGW subsystem is the CORBA-enabled EDS sink. This is similar to other EDS sinks except that the events distributed by the sink are CORBA events. It is possible for multiple instances of CORBA-enabled EDS sinks to be running on a single system, thereby allowing different CORBA clients to receive their events from different CORBA-enabled EDS sinks. However, CORBA clients are not aware of this multitude of CORBA-enabled EDS sinks nor do they need to know about it. Transparency is the outcome of this approach.

Note – Henceforth, the term EGW and CORBA-enabled EDS sinks will be used interchangeably, unless there is a need to differentiate them.

Three JIDM IDL interfaces and one SEM CORBA Gateway-specific interface are implemented by the EGW:

JIDM IDL interfaces:

- `EventPort`
- `EventPortFactory`
- `EventPortFinder`

SEM CORBA Gateway interface:

- `EventPortRegistry`

`EventPortFactory` and `EventPortFinder` CORBA objects are singleton objects for the entire system (in this case, one instance of Solstice EM). CORBA client applications receive their event notifications through `EventPort` objects, one of which is typically created for each CORBA client application.

`EventPort` objects are dynamically created based on the `AE_Title` in the `criteria`. The Event Port Registry (EPR) sets up a connection with the `CosEventChannelAdmin::SupplierAdmin` object (a part of the Event Services that is supplied by the Object Request Broker (ORB) vendor, or any other vendor) associated with the `EventPort` object that has the appropriate title.

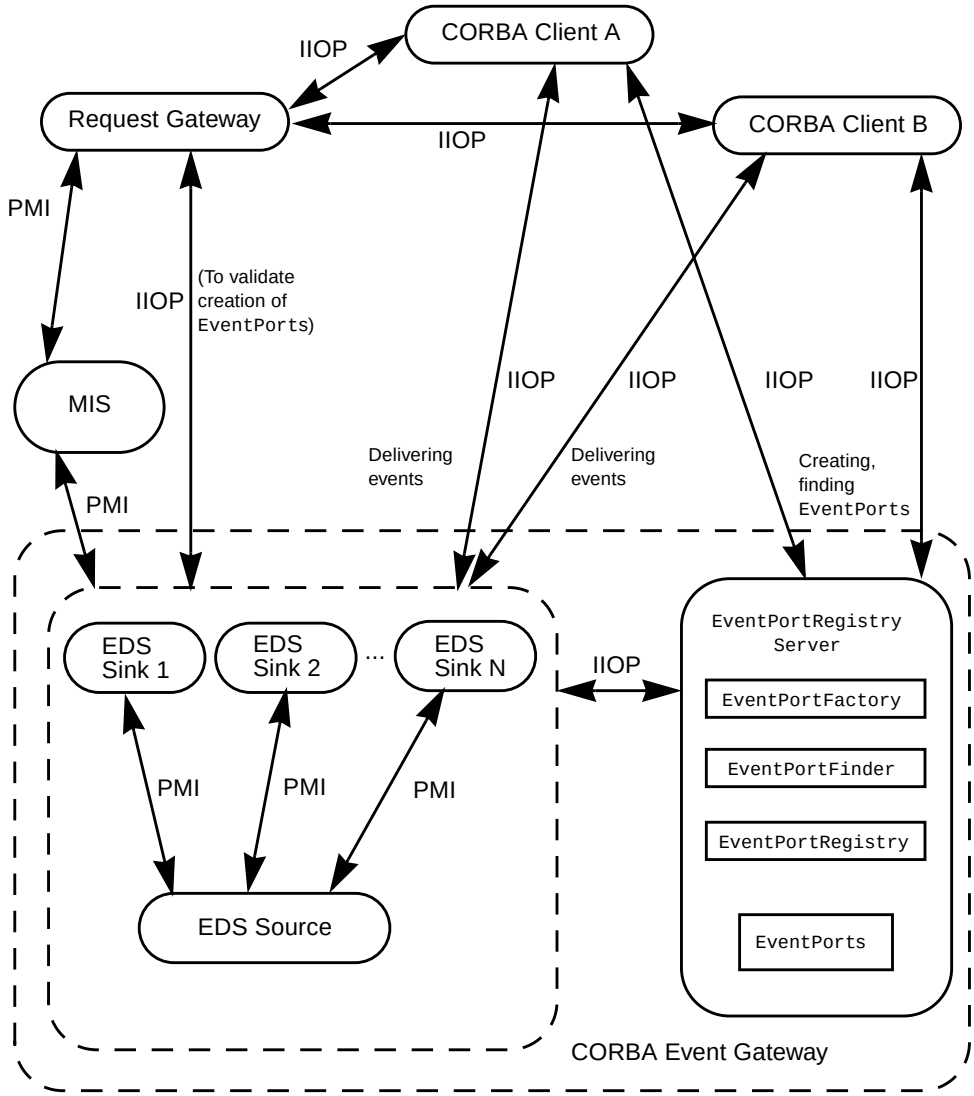


FIGURE 4-1 The CORBA Event Gateway and Its Interfaces

4.1 Enabling Inter-Process Communication Between EDS Sinks and CORBA Clients

The `EventPortFactory` and `EventPortFinder` server objects are implemented in a separate stand-alone server. This server is a dæmon UNIX process and is called an `EventPortRegistry` object. This process also implements `EventPortRegistry` objects that provide inter-process communication between EDS sinks and CORBA clients.

On start up, the `em_vb_corba_epr` server process ((for VisiBroker) creates the `EventPortRegistry` CORBA object which is registered with the CORBA Naming Service. The server then proceeds to create an `EventPortFinder`, and finally, creates an `EventPortFactory`.

The following code example gives the IDL definition for `EventPortRegistry`.

CODE EXAMPLE 4-1 IDL Definition for `EventPortRegistry`

```
interface EventPortRegistry {

EventPort create_event_port (
                                in Key k,
                                in Criteria creation_criteria,
                           in CosEventChannelAdmin::SupplierAdmin
the_supplier_admin
                      )      raises (InvalidKey, InvalidCriteria,
CannotMeetCriteria, AlreadyExists);
CosEventChannelAdmin::SupplierAdmin find_event_port (
                                in Key k,
                                in Criteria the_criteria
                      )      raises (InvalidKey, InvalidCriteria,
CannotMeetCriteria, NoEventPort);
EventPort find_event_port_by_ae_title(
                                in string ae_title
                         )      raises(NoEventPort);

};
```

4.1.1 Finding an EventPort

The find_event_port_by_ae_title() registry method is used only by EDS sinks that need to know the mapping between AE_Titles and EventPorts. The find_event_port_by_ae_title() method should *not* be used by a CORBA client. Instead, the find_event_port() method—which returns the SupplierAdmin registered previously—should be used.

4.1.2 Creating an EventPort

A CORBA client invokes the EventPortFactory::create_event_port() method to create an EventPort. The client must supply the following creation_criteria parameters:

- The AE_Title of the client (dot notation in an IDL string)
- A reference to the OSIMgmtExt::ProxyAgent CORBA object
- An EventInfoFormat criteria value of either CORBA::Any or CORBA::string (the default). This allows users to choose the encoding or format of event information fields in events.

The following table gives reasons for the typical exceptions raised.

TABLE 4-1 Reasons for Typical Exceptions Being Raised

Exception Raised	Reason
CannotMeetCriteria	Validation of the ProxyAgent object reference failed
InvalidKey	Key is not valid
AlreadyExists	Key and Criteria already exist in the registry or there is a one-to-one mapping between EventPorts and AE_Titles in the given domain

Note – To delete an EventPort field use the destroy method of the EventPort interface.

4.2 Subscribing to Events

CORBA clients listen for events by issuing a `subscribe()` request to the RGW to create an EFD (the definition of `subscribe()` is given in CODE EXAMPLE 4-2). However, before they create any EFDs, they must create and register `EventPorts`. When a subscribe request is processed by the CORBA RGW and sent to the MIS, the EFD attribute `Secty` in MIS will process this create request and send the `AE_title` as a listener to one of the CORBA EDS sinks.

The RGW ensures that it passes the user information in the create request. The `AE_Title` along with the user information and CMIS filter are passed to the EDS sink to make it an event-listener. The user information is required to enforce access control on outgoing events.

CODE EXAMPLE 4-2 IDL Definition of `subscribe()` (from `OSIMgmtExt.idl`)

```
typedef string SubscriptionId;
typedef sequence<string> StringSeq;

SubscriptionId subscribe_events(
            in StringSeq event_list,
            in StringSeq object_class_list,
            in StringSeq object_name_list
        ) raises (
            OSIMgmt::AccessDenied,
            ProcessingFailure
        );
```

If both `object_class_list` and `object_name_list` are empty, this will cause subscription to the `event_types` specified in the `event_list` from the objects. If `object_name_list` is empty and `object_class_list` is specified, this will cause subscription to event types from objects of all classes from the `object_class_list`.

If the `object_name_list` and `object_class_list` are specified, this will cause subscription to event types form the objects and classes (union of `object_class_list` and `object_name_list`).

The subscription is cancelled when it is explicitly unsubscribed or when the `ProxyAgent` is destroyed.

4.3 Unsubscribing From Event Notifications

To explicitly unsubscribe from event notifications previously registered for, use the `unsubscribe_events()` method from the `ProxyAgent` interface:

CODE EXAMPLE 4-3 Method for Unsubscribing From Event Notifications

```
void unsubscribe_events(
            in SubscriptionId subscription_id
        ) raises (
            OSIMgmt::AccessDenied,
            ProcessingFailure,
            InvalidSubscriptionId
        );
```

4.4 Formatting Event Reports

Since JIDM does not explicitly mandate the IDL format for events, the IDL script shown in the following code example (taken from `CMIExt.idl`) is used for formatting event reports.

CODE EXAMPLE 4-4 IDL Event Report Format

```
EventReport// Format of events received from the event_gateway
    struct EventReport {
        string event_type;
        string object_class;
        string object_name;
        string event_time;
        any    event_info;
    };

// Format of text events received from the event_gateway
struct TextEventReport {
        string event_type;
        string object_class;
        string object_name;
        string event_time;
```

CODE EXAMPLE 4-4 IDL Event Report Format *(Continued)*

```
        string event_info;
    };
```

4.5 Sharing Events Between Multiple Clients

Since a CORBA `EventChannel` can serve multiple consumers, Solstice EM event notifications can be distributed to multiple clients through the same `EventChannel`. In this case, the first of these clients gets the `EventPort` and the other clients bind to the same `EventPort`.

4.6 Listening to Events—Client Applications

To listen to event notifications, perform the following steps:

1. **Initialize the ORB, resolve the** `NameServer` **interface and connect to the RGW**
2. **Get the** `ProxyAgent` **interface,** `osi_agent`
3. **Initialize, or get a reference to, an** `EventChannel`
4. **Resolve the** `EventPortFactory` **interface**
5. **Assign a client to the** `EventChannel`
6. **Create an** `EventPort`
7. **Subscribe to the events**

The code fragments in the subsequent sections illustrate the steps listed above.

Note – Steps 1 and 3 vary from ORB to ORB and are not detailed here. Step 2 is described in Section 4.2 “Subscribing to Events” on page 4-6. The rest of the code fragments follow.

4.6.1 Resolving the EventPortFactory Interface

The following code example shows how to resolve the EventPortFactory interface.

CODE EXAMPLE 4-5 Resolving the EventPortFactory Interface

```
/*Resolving the EventPort Factory interface */
CosNaming::Name name;
   name.length(1);
   name[0].id = CORBA::string_dup("EventPortFactory");
    try {
        CORBA::Object_var object = root_nc->resolve(name);
        JIDM::EventPortFactory_var ep_factory =
            JIDM::EventPortFactory::_narrow(object);
        if (CORBA::is_nil(ep_factory)) {
            cerr << "FAILED: Could not get EventPortFactory\n";
            TSShutdownManager::shutdown();
        }
      cout << "PASSED: Accessing the EventPortFactory\n" << flush;
    }
    catch (const CORBA::Exception& e) {
cerr << "FAILED: Could not resolve EventPortFactory name\n";
        throw;
    }
```

4.6.2 Assigning a Client to an EventChannel

The following code example assumes that you have already initialized an EventChannel interface pointer and that communication with the EventChannel (i.e. PushConsumer) has already been established. See Section 4.6.5 “Sample PushConsumer” on page 4-12 for sample code of PushConsumer.

CODE EXAMPLE 4-6 Assigning a Client to an EventChannel

```
PortableServer::POAManager_var rootManager =
    root_poa->the_POAManager();

    root_poa->activate_object(&push_consumer);

    rootManager->activate();

    CosEventChannelAdmin::ConsumerAdmin_var consumer_admin =
    channel->for_consumers();
```

CODE EXAMPLE 4-6 Assigning a Client to an `EventChannel` *(Continued)*

```
    CosEventChannelAdmin::ProxyPushSupplier_var pushSupplier =
    channel->for_consumers()->obtain_push_supplier();
    pushSupplier->connect_push_consumer(&push_consumer);
    supplier_admin = channel->for_suppliers();
```

4.6.3 Creating an `EventPort`

The following code example shows how to create an `EventPort`.

CODE EXAMPLE 4-7 Creating an `EventPort`

```
JIDM::Key key;
        key.length(1);
        key[0].id = CORBA::string_dup("OSI Management");
        key[0].kind = CORBA::string_dup("XSM environment");
        JIDM::Criteria event_criteria;
        event_criteria.length(3);
       event_criteria[0].name = CORBA::string_dup("domain title");
        event_criteria[0].value <<= (const char* )ae_title;
      // You need to pass ProxyAgent's access criteria for creating
event
     // ports. This is required for enforcing connection level access
        // control.
        event_criteria[1].name = CORBA::string_dup(
            "ProxyAgent Access Criteria");
        event_criteria[1].value <<= criteria;
   event_criteria[2].name = CORBA::string_dup("EventInfo Format");
        if (format != NULL)
            event_criteria[2].value <<= (const char* )format;
        else
        event_criteria[2].value <<= (const char* )"CORBA::string";
        ep = ep_factory->create_event_port(key, event_criteria,
supplier_admin);
```

4.6.4 Subscribing to Events

The following code example shows how to subscribe to events.

CODE EXAMPLE 4-8 Subscribing to Events

```
OSIMgmtExt::ProxyAgent::StringSeq event_list;

// For all events: equivalent to CMIS Filter = and :{}
// If you want to subscribe to specific events, then uncomment the
// following line and comment out the previous line of code that
// initializes event_list to zero length
// event_list[0]=CORBA::string_dup("\"Rec.X.721|ISO/IEC10165-2:
1992\": objectCreation");


    event_list.length(0);

// Object Classes should be "document":oc format

 OSIMgmtExt::ProxyAgent::StringSeq object_class_list;
 object_class_list.length(0);

// OIs should be either in the slash format or in brace format
// Example: slash format: /systemId=name:"sesha"/
logId=string:"AlarmLog"
// brace format:
//
distinguishedName:{{{systemId,name:"sesha"}},{{logId,string:"AlarmLog
"}}}

    OSIMgmtExt::ProxyAgent::StringSeq object_name_list;
    object_name_list.length(0);

    sid = osi_agent->subscribe_events(
                event_list, object_class_list, object_name_list);
```

4.6.5 Sample PushConsumer

The following code example shows how to implement PushConsumer.

CODE EXAMPLE 4-9 Sample PushConsumer

```
class PushConsumerImpl : public _sk_CosEventComm::_sk_PushConsumer
{
  public:
    PushConsumerImpl()
    {
    }

    void push(const CORBA::Any& data)
    {
      /* Put Event Processing Code Here */

     }

    void disconnect_push_consumer() {
        cout << "disconnect_push_consumer() invoked" << endl;
        _boa()->shutdown();
    }

  };
```

CHAPTER 5

Translating Data

The translation of data in SEM CORBA Gateway is in compliance with the *JIDM Specification Translation* document.

The data translation (IDL to ASN1 and vice-versa) is based on the IDL files `ASN1Types.idl` and `ASN1Limits.idl`, which contain the base definition and classes.

CORBA client applications can use Metadata Gateway (MGW) to access the ASN1 type information of the attributes or events, and to traverse the Management Information Tree (MIT) in Solstice Enterprise Manager (Solstice EM). The basic design of MGW is independent of the JIDM standards and can therefore be accessed by any client, even a non-JIDM client.

This chapter describes the following topics:

5.1 Metadata Gateway Interface

The MGW interface that is exposed to clients is a *singleton* CORBA object. The UNIX dæmon process, MGW, implements this object and provides the following functionality:

- Get ASN1 type given module name and label
- Look up `Node` by name, given the object type, GDMO document name, and attribute name
- Get `OID` by *name*
- Get `name` by *OID*
- Get GDMO document list
- Get managed object class (MOC) list

- Get MOC attributes by *OID*
- Get MOC attributes by *name*
- Get MOC notifications by *OID*
- Get MOC notifications by *name*

Some of the above functionality is supported in both textual representation and binary representation.

5.2 Encoding and Decoding Attribute Values

Management requests sent by CORBA client applications to the SEM CORBA Gateway are converted from IDL format to the PMI's ASN1 values and conversely responses received by the SEM CORBA Gateway are converted from ASN1 values back to IDL format.

When a CORBA application sends a complex data type (for example, the `action_info` attribute in `cmis_action`) the application must encode the value represented by the complex data type in IDL format.

When a CORBA application receives an attribute value represented by a complex data type (for example, `CORBA:Any`) it must decode this value to extract the information the value contains.

The SEM CORBA Gateway facilitates this by using a set of conversion libraries and the Solstice EM Metadata interface.

The following table gives some sample mappings between ASN1 types and IDL types. For more information on IDL mappings of ASN1 types refer Appendix A.

TABLE 5-1 Sample Primitive Mappings Between ASN1 Types and IDL Types

ASN1 Type	IDL Type
`INTEGER`	`long`
`REAL`	`double`
`BOOLEAN`	`boolean`
`NumericString`	`String`
`PrintableString`	`String`
`VisibleString`	`String`

TABLE 5-1 Sample Primitive Mappings Between ASN1 Types and IDL Types *(Continued)*

ASN1 Type	IDL Type
ObjectDescriptor	String
GraphicString	String
TeletexString	String
ObjectIdentifier	String
OCTET STRING	sequence<octet>
GENERAL STRING	sequence<octet>
VIDEOTEXTSTRING	sequence<octet>

The following figure shows the encoding/decoding done by the CORBA Request Gateway (RGW).

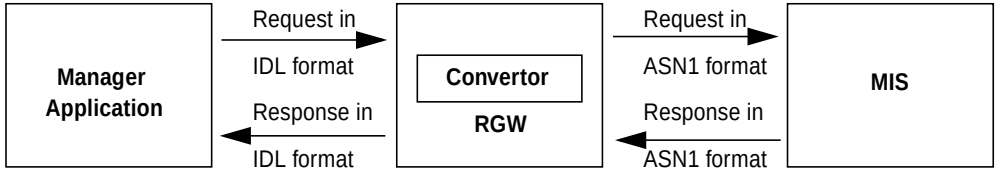


FIGURE 5-1 Encoding/Decoding Done by RGW

Some of the complex IDL to ASN1 mappings are described in the following paragraphs:

- The ASN1 SEQUENCE is mapped to sequence<any>
- CHOICE is mapped to typedef sequence<any> ASN1_Seq

The following code segment gives the ASN1Type syntax of FileAttribute and CurrentAttributes.

```
FileAttribute ::= CHOICE {
                     date-last-used   INTEGER,
                     file-name   VisibleString
                     }


CurrentAttributes ::= SEQUENCE {
                        date-last-used  FileAttribute,
                        file-name  FileAttribute
                        }
```

CODE EXAMPLE 5-1 shows how to encode CORBA::Any corresponding to CurrentAttributes. In the following code example, the CORBA application is attempting to set an attribute fooFile of type CurrentAttributes, with values (100, "filename String").

CODE EXAMPLE 5-1 Encoding of CORBA::Any Corresponding to CurrentAttributes

```
ASN1_Seq aseq(2);

aseq[0] <<= CORBA::Long(100);
aseq[1] <<= "filename String" ;

CORBA::Any any_sel;

any_sel <<= aseq ;
```

In the cmis_set request the encoded value any_sel will be passed to the attribute value.

CODE EXAMPLE 5-2 shows that the user is receiving a CORBA::any corresponding to CurrentAttribute as a reply from a get request for an attribute that needs to be decoded.

CODE EXAMPLE 5-2 Decoding an Attribute Value

```
// assumed that attribute_value ( of type CORBA:Any ) is received
// as response.

      ASN1_Seq seq(2);

      CORBA::Long j;
      char *visstr2;

      attribute_value >> seq ;

      if ( seq[0] >>= j ) {
                cout << "date_last_used" << j ;
      }

      if ( seq[1] >>= visstr2 ) {
          cout << "file Attribute " << visstr2 ;
    }
```

CODE EXAMPLE 5-3 gives the mapping of data from CORBA IDL format to GDMO format. This following code represents CORBA data in a variable `foo` of type `ASN1_Seq`.

CODE EXAMPLE 5-3 Mapping data from CORBA IDL Format to GDMO Format

```
typedef sequence<any> ASN1_Seq;
ASN1_Seq foo(2);
  long counter=2;
  string message="";
  CORBA::Any corba_any1, corba_any2 ;

  corba_any1 <<= counter ;
  corba_any2 <<= message ;

  foo[0] = corba_any1 ;
  foo[1] = corba_any2 ;
```

The following table gives the steps for mapping CORBA IDL data to GDMO format.

TABLE 5-2 Steps for Mapping CORBA IDL Data to GDMO Format

Step No.	Conversion Step	Resulting GDMO expression
1	`ASN1_Seq`	`SEQUENCE`

2	`ASN1_Seq foo(2)`	`foo ::=SEQUENCE {`
		`--`
		`--`
		`}`

3	`foo[0] = corba_any1`	`foo ::=SEQUENCE {`
		`any`
		`--`
		`}`

TABLE 5-2 Steps for Mapping CORBA IDL Data to GDMO Format *(Continued)*

Step No.	Conversion Step	Resulting GDMO expression

4	`foo[0] = corba_any2`	`foo::= SEQUENCE {`
		`any`
		`any`
		
		`}`

5	`corba_any1 <<= counter`	`foo ::= SEQUENCE {`
		
		`INTEGER counter,`
		`any`
		
		`}`

6	`corba_any2 <<= message`	`foo::= SEQUENCE {`
		
		`INTEGER counter,`
		`VisibleString message`
		
		`}`

5.3 Decoding Events and Responses

The following two code segments denote the two formats in which events will be received from a CORBA Gateway:

```
struct EventReport {
        string event_type;
        string object_class;
        string object_name;
        string event_time;
        any    event_info;
    };
```

```
struct TextEventReport {
        string event_type;
        string object_class;
        string object_name;
        string event_time;
        string event_info;
    };
```

The `event_info` contains attributes of type `Notification`. In the `EventReport` structure, `event_info` contains a sequence of attributes that is defined for the `event_type`, and corresponds to the ASN1 type defined in the GDMO document.

The following code example gives the `attributeValueChange Notification` as defined in *dmi.gdmo* document.

CODE EXAMPLE 5-4 Definition of `attributeValueChange Notification`

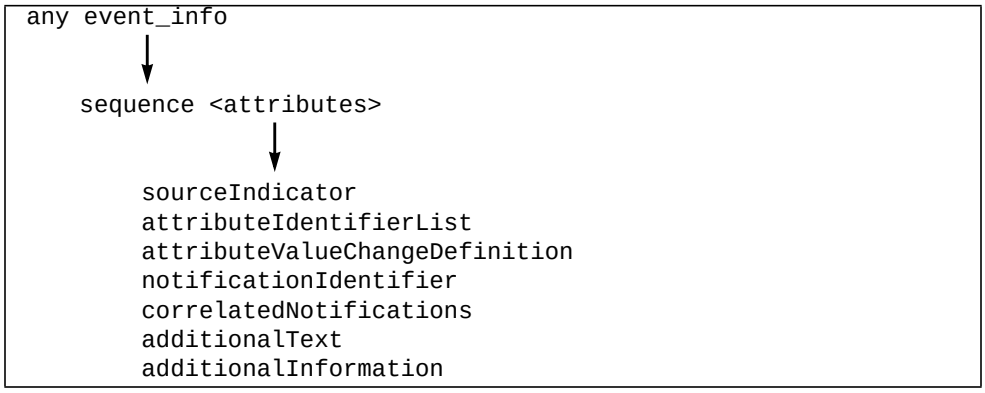
```
attributeValueChange    NOTIFICATION
    BEHAVIOUR    attributeValueChangeBehaviour;
    WITH INFORMATION SYNTAX Notification-
ASN1Module.AttributeValueChangeInfo
        AND ATTRIBUTE IDS
         sourceIndicator          sourceIndicator,
        attributeIdentifierList        attributeIdentifierList,
  attributeValueChangeDefinition attributeValueChangeDefinition,
        notificationIdentifier        notificationIdentifier,
        correlatedNotifications        correlatedNotifications,
        additionalText            additionalText,
        additionalInformation        additionalInformation;

REGISTEREDAS {joint-iso-ccittms(9)smi(3)part2(2)notification(10)
1};
```

When an `attributeChange` event occurs, the event information corresponds to the `AttributeValueChangeInfo`, which is defined as follows:

```
AttributeValueChangeInfo ::= SEQUENCE {
    sourceIndicator                    SourceIndicator OPTIONAL,
  attributeIdentifierList    [1] AttributeIdentifierList OPTIONAL,
  attributeValueChangeDefinition  AttributeValueChangeDefinition,
   notificationIdentifier         NotificationIdentifier OPTIONAL,
  correlatedNotifications    [2] CorrelatedNotifications OPTIONAL,
    additionalText                     AdditionalText OPTIONAL,
   additionalInformation       [3] AdditionalInformation OPTIONAL}
```

The event report for the `attributeValueChange` event is as follows:



Here each attribute is expanded based on the types defined in the *dmi.gdmo* document.

The ASN1 type definitions for the attributes as defined in *dmi.asn1* document is as follows:

```
SourceIndicator ::= ENUMERATED { resourceOperation(0),
managementOperation(1), unknown(2)}


AttributeIdentifierList ::= SET OF AttributeId

AttributeId ::= CHOICE {
        globalForm       OBJECT IDENTIFIER,
        localForm        INTEGER
}
```

```
SourceIndicator ::= ENUMERATED { resourceOperation(0),
managementOperation(1), unknown(2)}


AdditionalText ::= GraphicString

AdditionalInformation ::= SET OF ManagementExtension


AttributeValueChangeDefinition::= SET OF SEQUENCE {
                attributeID    AttributeId,
              oldAttributeValue    [1] ANY DEFINED BY attributeID
OPTIONAL,
              newAttributeValue    [2] ANY DEFINED BY attributeID}

 NotificationIdentifier ::= INTEGER

 CorrelatedNotifications ::= SET OF SEQUENCE {
          correlatedNotifications   SET OF NotificationIdentifier,
                sourceObjectInst        ObjectInstance OPTIONAL }
```

In the following code segment, only the `AttributeIdList` is expanded; other attributes also are expanded in the same manner.

```
AttributeIdentifierList
                ↓
     sequence <AttributeId>
                ↓
      struct ASN1_Choice {
            unsigned short selector  → 1
            any            value
            };
                              ↓
                          localForm
                              ↓
                            Integer
```

CHAPTER 6

Accessing Information Through Metadata Gateway

The SEM CORBA Metadata Gateway (MGW) implements the Metadata Repository Interface (MRI), which provides methods to access the ASN1 metadata in Solstice Enterprise Manager (Solstice EM).

The purpose of the MGW is to make available IDLs with the following functionality:

- Functionality to return the ASN1 type of a given attribute name or `oid`. This functionality is used to encode or decode an IDL value for an attribute and decode an event received as an IDL value.
- Functionality to extract information from the metadata. This functionality is used to verify what is already loaded into the Metadata Repository (MDR), before loading an updated object model.
- Utility functions to convert `oid` to `name` and vice versa.

This chapter discusses the MGW and how to access information through the MGW.

This chapter describes the following topics:

6.1 Browsing Metadata

The CORBA MGW object acts as a tree-structured repository for ASN1 types encapsulated in the `Node` IDL interface. The root of the tree is either a known ASN1 type or a type of class (`attribute`, `actioninformation`, `eventtype` or `actionreply`). Structured ASN1 types such as `SET` and `SEQ` contain `Node` structures as sub-components for ASN1 tree navigation.

The Node structure wraps the actual ASN1 type definitions in IDL format. The following code segment gives the definition of a Node structure.

```
// ASN1 Type node in the metadata tree/graph
    struct Node {
        Asn1Kind kind ;
        Asn1Kind base_kind ;
        any type_info ;
```

Each Node structure contains the underlying ASN1 type node in type_info. The kind is identified by the built-in type member.

The client program must connect to the MGW to use the IDL functions.

To connect to the MGW, the client application must initialize the ORB and resolve the root naming context before the MGW can be resolved, as shown in the following code example.

CODE EXAMPLE 6-1 Connecting to the MGW

```
// Example to show how to get connected to Metadata Gateway.

#include <iostream.h>
#include <corba.h>
#include <cos/CosNaming_c.hh>
#include <metadata_gw/SEMMetaData_c.hh>
#include <pmi/hi.hh>

int
main(
    int argc,
    char **argv
) {
    CORBA::ORB_var orb;

    // CORBA Orb intialization
    try {
        orb = CORBA::ORB_init(argc, argv);
    }
    catch(const CORBA::Exception& e) {
      cerr << "ORB Init FAILED: " ;
      exit(1);
    }

    // Get rootnaming context
    CosNaming::NamingContext_var root_nc;
    try{
```

CODE EXAMPLE 6-1 Connecting to the MGW *(Continued)*

```
        CORBA::Object_var object =
            orb->resolve_initial_references("NameService");
        root_nc = CosNaming::NamingContext::_narrow(object);
        if(CORBA::is_nil(root_nc)) {
        cerr << "FAILED: Unable to obtain root naming context\n" <<
flush;
            exit(2);
        }
    }
    catch(const CORBA::Exception& e) {
       cerr << "FAILED: Unable to obtain root naming context:\n" ;
        exit(3);
    }
    SEMMetaData::MetaDataRepository_var meta_data_repository;
    try{
        CosNaming::Name name;
        name.length(1);
        name[0].id =
CORBA::string_dup("SEMMetaData::MetaDataRepository");
        name[0].kind = CORBA::string_dup("");

        CORBA::Object_var object = root_nc->resolve(name);
        meta_data_repository =
            SEMMetaData::MetaDataRepository::_narrow(object);
        if (CORBA::is_nil(meta_data_repository)) {
            cerr << "FAILED: Unable to obtain correct MetaData
Repository\n"
                 << flush;
            exit(4);
        }
        if(meta_data_repository->_non_existent()) {
         cerr << "FAILED: MetaData Repository does not exist\n" <<
flush;
            exit(5);
        }
    }
    catch(const CORBA::Exception& e) {
        cerr << "FAILED: Unable to resolve"
SEMMetaData::MetaDataRepository:\n
        exit(6);
    }

 }
```

The ASN1 types follow an inheritance hierarchy. Individual ASN1 classes are modeled as structures when translated to IDL, and instances of a particular class are represented in IDL as a sequence of structures derived from the base class.

The following figure shows the class hierarchy followed in the IDL representation.

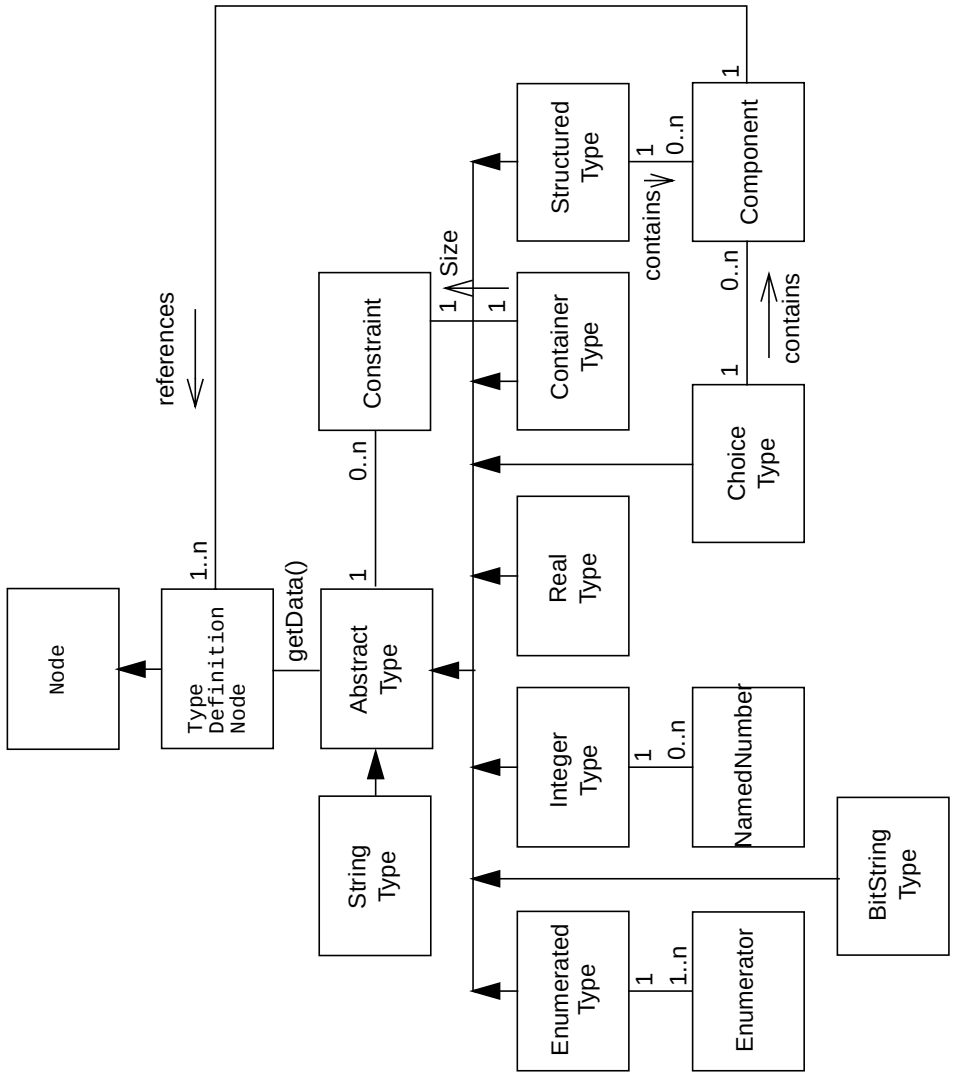


FIGURE 6-1 Class Hierarchy Followed in the IDL Representation

The following figure shows the decomposition of IDL data structures defined in `metadatagw.idl`

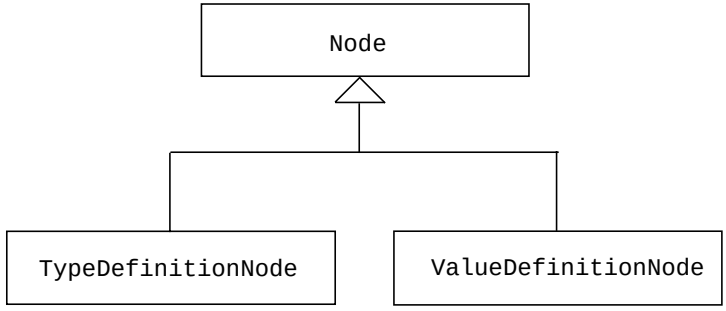


FIGURE 6-2 Decomposition of IDL Data Structures Defined in `metadatagw.idl`[1]

The following figure shows the mapping of ASN1 Defined Type into the IDL structure `DefinedType`.

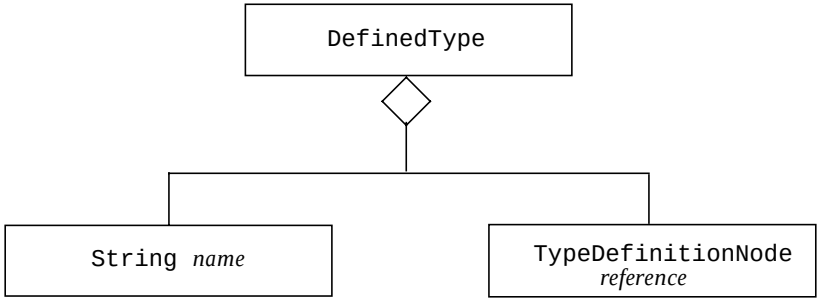


FIGURE 6-3 The ASN1 Defined Type Mapped Into the IDL Structure `DefinedType`

1. In the arrow notation used in these figures, a diamond-shaped arrowhead indicates an “Aggregation” relationship; a triangular arrowhead indicates a “Generalization” relationship; a blank arrow between boxes indicates an association between them.

The ASN1 *Structured Types*[1] are used for building complex data types, and are mapped to IDL as follows. The four structured types SEQUENCE, SET, SEQUENCE OF, and SET OF are represented by ASN1 as ComponentType.

The following figure shows the decomposition of the IDL Component Type.

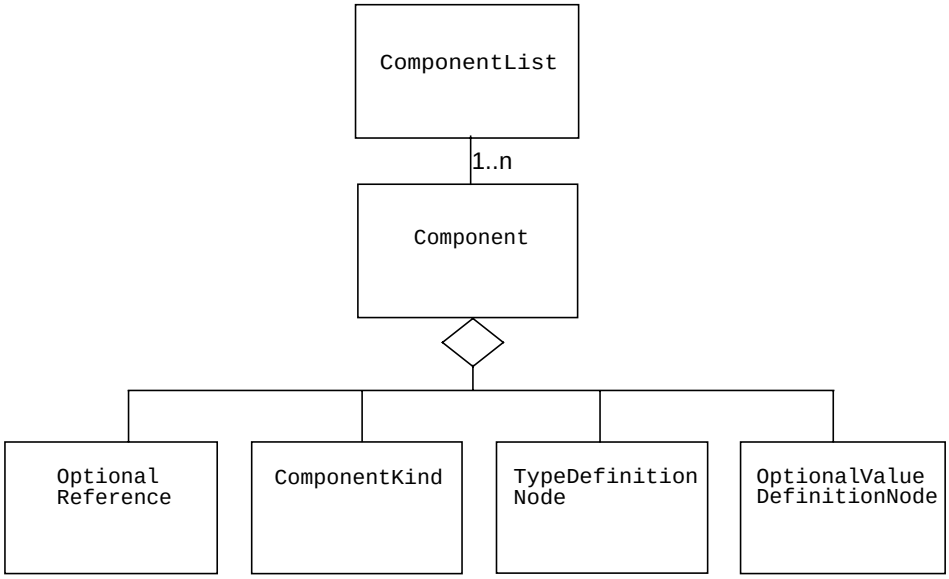


FIGURE 6-4 Decomposition of Component IDL Types

The Asn1SubType is derived from ParentType by restricting the set of values defined for ParentType. Six different forms of SubTypes are present in Asn1Type. The decomposition of these Asn1SubType mappings is shown in FIGURE 6-5.

1. Structured types are those consisting of components.

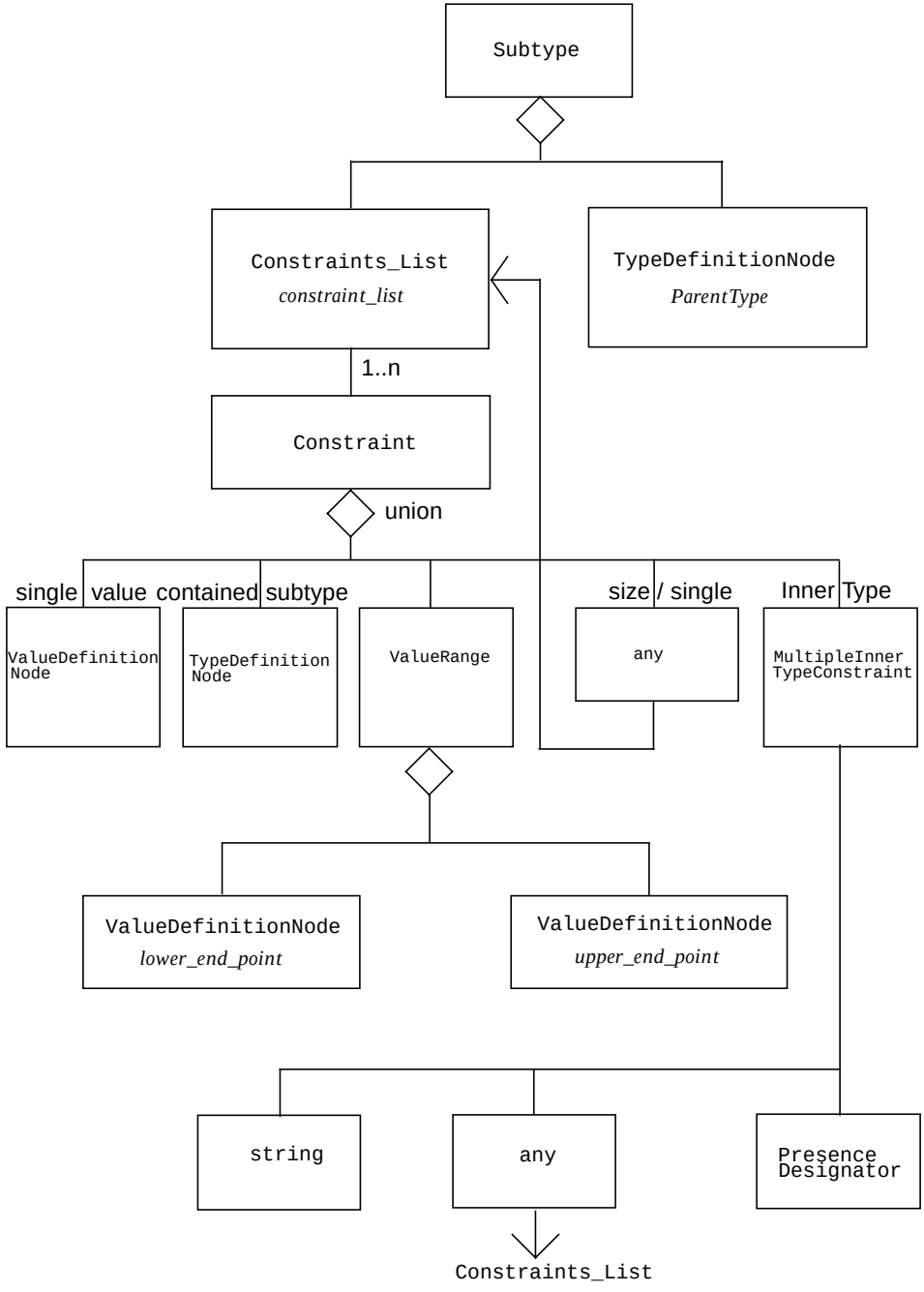


FIGURE 6-5 Decomposition of IDL Subtype

The NamedNumber format of ASN1 follows the IDL mapping decomposition shown in the following figure:

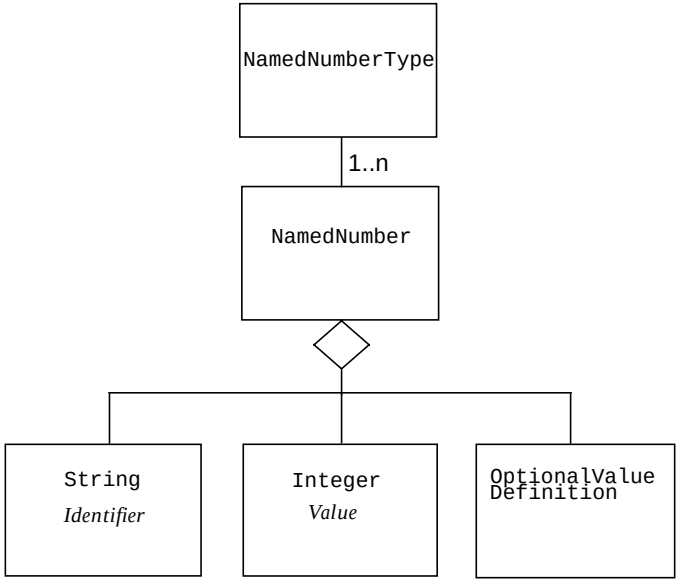


FIGURE 6-6 Decomposition of NamedNumber Format

6.2 Walking Through Metadata

When you wish to encode an IDL value for an attribute, you should get the type information for the attribute, *walk*[1] the type, and encode IDL values based on the subtypes using the following methods:

```
1.Node get_asn1_type (in string modname, in string label)
```

This method returns Node(ASN1Type) representing modname:label PMI: Asn1Type type(modname, label)

```
2. Node lookup_node_by_name (in ObjectType object_type, in
  string gdmo_doc_name, in string name) raises (NotFound);
```

Node get_asn1_type (in string modname, in string label) gets the IDL type Node structure, which is a wrapper of the Asn1Type.

1. "Walking" through the Metadata means traversing the subtypes based on ASN1 type.

The following code segment is an example for Asn1SubType ValueRange.

```
 TopoNodeId ::= INTEGER (0..4294967295)
```

The BNF syntax for the ASN1Tree structure for the type Asn1SubType ValueRange is as follows:

```
Type::= DefinedType
DefinedType::= ExternalvalueReference  | valueReference
Subtype ::=  SingleSubType | ValueRange| SizeConstrainst|
SingleInnerSubType| Multiple.....
```

The Asn1 SubType ValueRange applies only to INTEGER/REAL. It is specified by giving numerical values of the end points of the range.

The following code segment gives the corresponding IDL type mapping wrapped into the Node structure.

```
Struct Node
{
      AsnKind kind,
      Asn1Kind base_kind,
      Any type-info
}
```

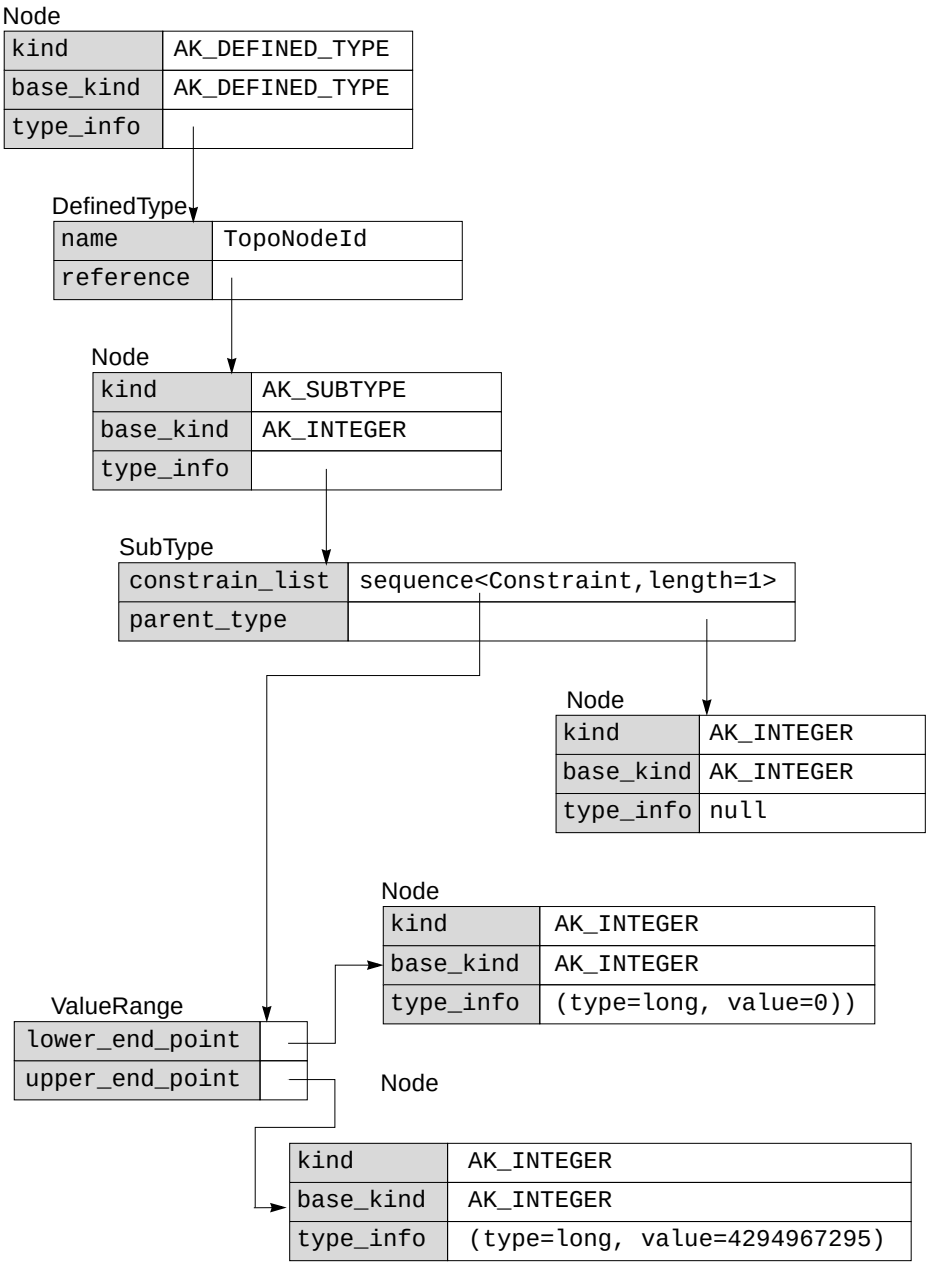


FIGURE 6-7 IDL Mapping Wrapped Into Node Structure

The following code example shows how to get the ASN1 type of an attribute from the metadata.

CODE EXAMPLE 6-2 Obtaining the ASN1 Type of an Attribute From the Metadata

```
try {

        const char* asn1_module_name = "ASN-1-TEST";
       const char* label = "PatientIdentifier";
       SEMMetaData::ASN1ElementName input;
    input.asn1_module_name = CORBA::string_dup(asn1_module_name);
       input.label  =  CORBA::string_dup(label);

       SEMMetaData::Node_var node =
           meta_data_repository->get_asn1_type(
               input
           );

       CORBA::Any any ;
       any <<= node;
       cout << "Output : Contents of Node: " << endl;

       CORBAUtils::print_any(cout, any, orb, true);
      }
      catch( const SEMMetaData::NotFound& nfe)
      {
            .......
      }
      catch(const SEMMetaData::ProcessingException& pe)
      {
   ........
      }
```

6.3 Obtaining Metadata Information

The MGW provides the following utility functions for obtaining information from the MDR:

- `get_doc_list()`
- `get_moc_list()`
- `get_moc_attributes_by_name()`
- `get_moc_attributes_by_oid()`
- `get_moc_notifications_by_name()`
- `get_moc_notifications_by_oid()`

- `get_textual_rep_by_name()`
- `get_textual_rep_by_oid()`

You can use these interfaces (utility functions) to verify what is already loaded on the MDR before loading an updated object model.

6.3.1 Listing Documents in the MDR Using the `get_doc_list()` Method

The `get_doc_list` method lists all the documents loaded into the MDR.

The following code example shows how to invoke the MDR interface to list all documents loaded on MDR (it is assumed that the client has connected to the `metadata_gw` and has the object reference, `meta_data_repository` ready).

CODE EXAMPLE 6-3 Invoking the MDR Interface to List All Documents Loaded on MDR

```
try {

      SEMMetaData::StringSeq_var attr_list =
           meta_data_repository->get_doc_list();

       cout << "OUTPUT Printing doc_list :" << endl;
       for (int j = 0; j < attr_list->length(); ++j) {
           cout << "OUTPUT: " << "attr_list" << "["
                << j << "]  :" << attr_list[j] << endl << flush;
       }
   }
   catch(
       const SEMMetaData::NotFound& nfe
   ) {
    cerr << "FAILED: Request failed [Not Found]: " <<  endl;
       exit(7);
   }
   catch(
       const SEMMetaData::ProcessingException& pe
   ) {

    cerr << "FAILED: Request failed [Processing Failure]: " << endl;
       exit(8);
   }
:
:
```

6.3.2 Listing Managed Object Classes in the GDMO Document Name

`get_moc_list()` lists all the managed object classes defined in the GDMO document name.

6.3.3 Getting the Managed Object Class Attributes

You can use the `get_moc_attributes_by_oid` and `get_moc_attributes_by_name` IDL interfaces to obtain the attributes of the specified managed object class from the MDR.

The syntax for these interfaces is:

```
GDMOElementNameSeq
  get_moc_attributes_by_oid (in string moc_oid)
    raises( NotFound, ProcessingException)


  GDMOElementNameSeq
 get_moc_attributes_by_name( in GDMOElementName gdmo_element_name)
     raises ( NotFound, ProcessingException)
```

`GDMOElementName` is a `struct` type defined in the `metadatagw.idl`.

```
 struct GDMOElementName {
        string gdmo_doc_name;
        string label;
    };
```

The following code example shows how to obtain the managed object class attributes based on GDMO document name and object class.

CODE EXAMPLE 6-4 Obtaining Managed Object Class Attributes Based on GDMO Document Name and Object Class

```
  try {
      const char* gdmo_doc_name = "'Rec. X.721 | ISO/IEC 10165-2 :
1992'" ;
       const char* name = "alarmRecord";
```

CODE EXAMPLE 6-4 Obtaining Managed Object Class Attributes Based on GDMO Document Name and Object Class *(Continued)*

```
        SEMMetaData::GDMOElementName input;
        input.gdmo_doc_name = CORBA::string_dup(gdmo_doc_name);
        input.label  =  CORBA::string_dup(name);

        SEMMetaData::GDMOElementNameSeq_var attr_list =
       meta_data_repository->get_moc_attributes_by_name( input );

        for (int j = 0; j < attr_list->length(); ++j) {
            cout << "OUTPUT: " << "attr_list" << "["
                 << j << "]  :" << attr_list[j].gdmo_doc_name
                 << " : " << attr_list[j].label << endl << flush;
        }

       }
       catch( const SEMMetaData::NotFound& nfe)
       {
             .......
       }
       catch(const SEMMetaData::ProcessingException& pe)
       {
    ........
       }
```

6.3.4 Getting Managed Object Class Notifications

You can use the IDL interfaces `get_moc_notifications_by_oid` and `get_moc_notifications_by_name` to get notifications defined in a managed object class from the MDR.

The following code example shows how to obtain notifications defined in a managed object class from the MDR.

CODE EXAMPLE 6-5 Obtaining Notifications Defined in a Managed Object Class From the MDR

```
GDMOElementNameSeq
get_moc_notifications_by_oid(in string moc_oid)
  raises ( NotFound, ProcessingException)
```

CODE EXAMPLE 6-5 Obtaining Notifications Defined in a Managed Object Class From the MDR *(Continued)*

```
  GDMOElementNameSeq
get_moc_notifications_by_name(inGDMOElementNamegdmo_element_name)
  raises ( NotFound, ProcessingException)
```

The following code example shows how to get the notifications of a managed object class based on its `oid`.

CODE EXAMPLE 6-6 Obtaining Notifications of a Managed Class Object Based on Its `oid`

```
try {
        SEMMetaData::GDMOElementNameSeq_var notif_list =
            meta_data_repository->get_moc_notifications_by_oid(
              CORBA::string_dup("1.3.6.1.4.1.42.2.2.2.19.3.1") );


       }
       catch( const SEMMetaData::NotFound& nfe)
       {
             .......
       }
       catch(const SEMMetaData::ProcessingException& pe)
       {
    ........
       }
```

6.3.5 Obtaining the Textual Representation of an Attribute

The ASN1 textual representation of class `Attribute`/`EventType`/`ActionReply`/`ActionInfo` can be obtained by providing the `oid` or `name`.

The following IDL interfaces are used in the operation:

```
TextualRepresentation
   get_textual_rep_by_name(in ObjectType object_type,
                           in GDMOElementName gdmo_element_name)

    TextualRepresentation
    get_textual_rep_by_name(in ObjectType object_type,
                in string oid)
```

`TextualRepresentation` is defined in `metadatagw.idl` as a sequence of strings.

The following code example shows how to obtain the ASN1 textual representation of an attribute.

CODE EXAMPLE 6-7 Obtaining the ASN1 Textual Representation of an Attribute

```
     try {

      const char* gdmo_doc_name = "'Rec. X.721 | ISO/IEC 10165-2 :
1992'" ;
        const char* name = "probableCause";

        SEMMetaData::MetaDataRepository::ObjectType object_type =
          SEMMetaData::MetaDataRepository::OT_ATTRIBUTE;

        SEMMetaData::GDMOElementName input;
      input.gdmo_doc_name = CORBA::string_dup(gdmo_doc_name);
        input.label  =  CORBA::string_dup(name);


        SEMMetaData::TextualRepresentation_var text_var =
          meta_data_repository->get_textual_rep_by_name(
                object_type,
                input
                );

        cout << text_var[0] << endl;
}
        catch( const SEMMetaData::NotFound& nfe)
        {
             .......
        }
        catch(const SEMMetaData::ProcessingException& pe)
        {
    ........
        }
```

CHAPTER 7

Managing Agents

The “JIDM Interaction Translation” standards specify three levels of interfaces to support interworking with different management environments:

- Generic interfaces which are management model independent
- Generic interfaces which are management model dependent
- Specific interfaces which are both information and management model dependent

The management model dependent generic interfaces define two management reference models, OSI and SNMP. These sets of interfaces extend the generic JIDM interfaces to support OSI- and SNMP-specific concepts.

This chapter discusses management model dependent generic interfaces that Solstice EM implements and how they can be used for managing both SNMP and CMIP agents.

The topics described in this chapter are:

7.1 Solstice EM-specific Generic Interfaces

The SEM CORBA Gateway implements the generic interfaces that are specific to supporting the OSI management model. Hence, the SEM CORBA Gateway implements the interfaces that support the following functionality:

- Creating managed objects
- Deleting managed objects
- Getting managed object attributes and values
- Setting managed object values

- Carrying out actions on managed objects

The CORBA Gateway converts all the requests from the client applications to CMIP / LPP (Lightweight Presentation Protocol) before sending them to SEM MIS for processing.

7.2 Managing OSI / CMIP Objects

The following figure depicts how CMIP objects are managed from CORBA manager applications.

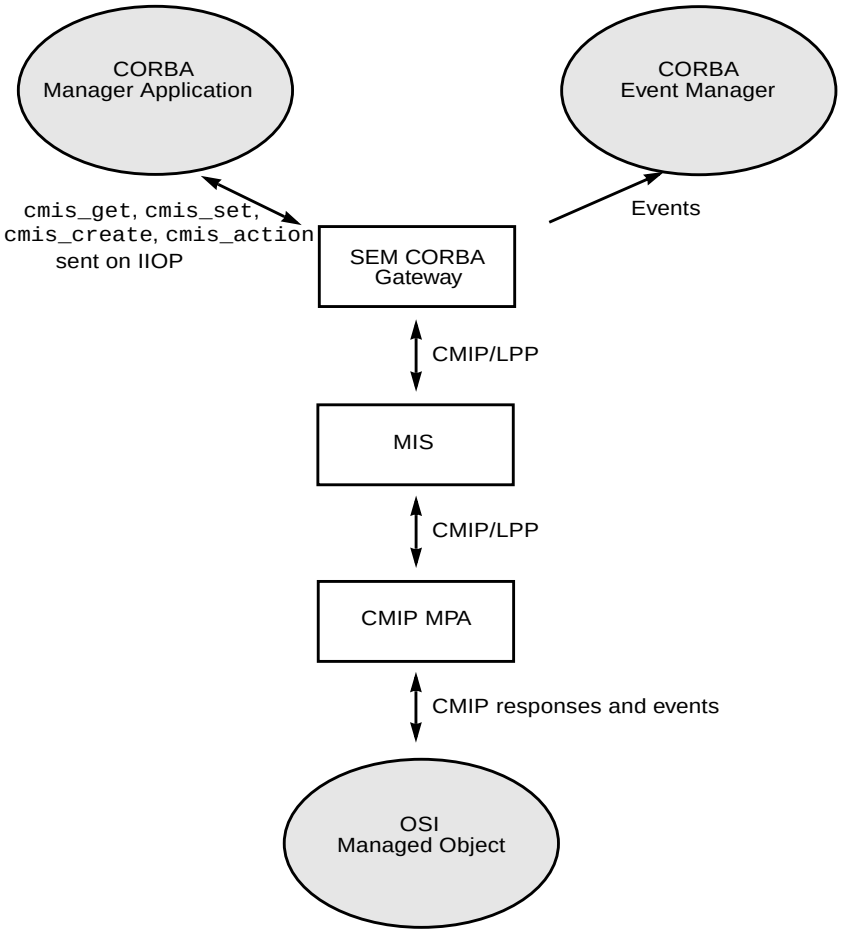


FIGURE 7-1 Managing CMIP Objects From CORBA Manager Applications

7.3 Managing SNMP Objects

The SEM CORBA Gateway uses the SNMP MPA to manage SNMP objects. The following figure depicts how an SNMP object is managed by the CORBA application manager.

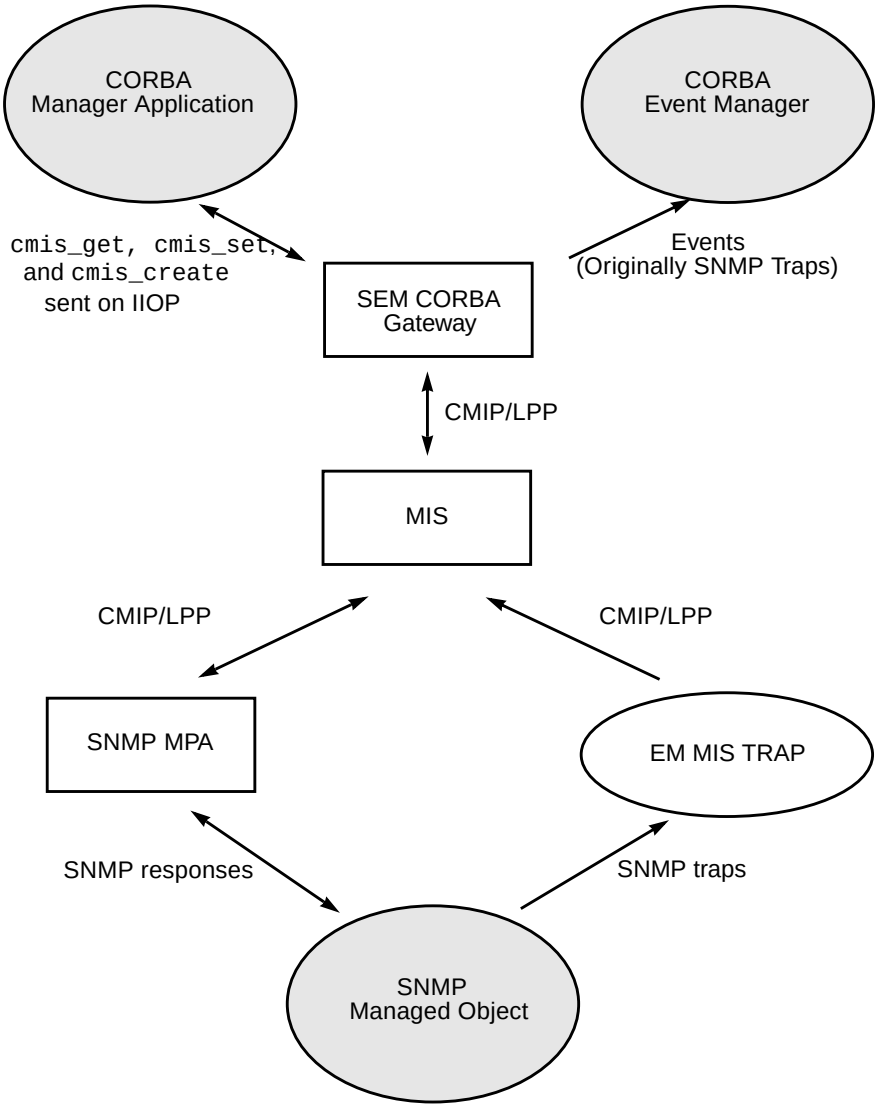


FIGURE 7-2 Managing SNMP Objects From CORBA Manager Applications

7.4 Management of CORBA Objects

The SEM CORBA Gateway implementation does not as yet provide a CORBA southbound interface, and hence, cannot manage network elements that are pure CORBA objects.

CHAPTER 8

Interoperating SEM CORBA Gateway

The SEM CORBA Gateway consists of a set of CORBA objects built as per the JIDM standards. These objects can be built and deployed to meet your requirements depending on the ORB that you have installed. It is also possible to implement these objects on one ORB and develop the client applications on another ORB and communicate with Solstice Enterprise Manager (Solstice EM) in a seamless way. This chapter describes the issues of interoperability.

This chapter describes the following topics:

8.1 Background on Interoperability

Interoperability in terms of CORBA standards is defined as “the ability of a client on ORB A to invoke an OMG IDL-defined operation on an object on ORB B, where ORB A and ORB B have been independently developed by the same vendor or by different vendors.”

Interoperability as applicable to SEM CORBA Gateway is defined as “the ability of a manager application developed on any SEM CORBA ToolKit supported ORB to be able to interact with SEM CORBA Gateway interfaces developed on any SEM CORBA ToolKit supported ORB and carry out all the Solstice EM supported functions on any of the managed objects transparently.”

Interoperability to a large extent is ensured by IIOP, which is a mandatory OMG requirement for all ORB vendors. All products that are compliant with CORBA 2.0 standards must implement IIOP as a standard communication protocol.

Some of the ORBs that are commercially available for developing client applications are listed in the following sections. Steps to follow when developing manager applications are also discussed.

8.2 ORBs for Developing Client / Manager Applications

The SEM CORBA ToolKit currently supports the following ORBs:

- VisiBroker 4.5
- Orbacus 4.0.5
- Orbix 2000 1.2.1

Any of these ORBs can be used for developing client / manager applications.

8.3 Implementing Client Applications on Other ORBs

If you wish to develop and implement client applications using other ORBs, which are not supported by the SEM CORBA ToolKit, perform the following steps[1]:

1. **Create a template file for the ORB.**

   For example, if you are building applications for HP Orb Plus, you can create a template file `hp.tmpl`. The template file defines the various parameters and flags required for IDL to CPP / Java compilation.

   The `.tmpl` file already exists for the ORB chosen at the time of SEM CORBA ToolKit installation. You can use this `.tmpl` file as reference while creating the `.tmpl` file for the current ORB.

1. It is assumed here that the client applications are to be written using C++.

2. **Create Makefiles for the ORB in the following IDL directories:**

   - `jidm`
   - `auth_proxy`
   - `jidm_ext`
   - `event_gw`
   - `metadata_gw`
   - `cos`

   The `Makefile` already exists for the ORB chosen at the time of SEM CORBA ToolKit installation. You can use this `Makefile` as reference while creating the `Makefile` file for the current ORB.

   While creating Makefiles, make sure that the header files, source files and the libraries are installed in the required directories.

Note – The template file created in Step 1 is included in the `Makefile` created in Step 2.

---

3. **Execute the Makefiles individually in the directories mentioned in Step 2 by using the following command:**

```
dmake <makefile-name-for-the-orb> install
```

   This step will create the C++ and header files; build the libraries and install them in the required directories.

   After creation and installation of the header files (.hh) and shared libraries; include the header files in your client applications; compile and link the header files to the shared libraries to build applications.

APPENDIX **A**

IDLs Used by SEM CORBA Gateway

The first section of this appendix describes some of the IDLs that are part of either the JIDM standards or the OMG standards. The IDL files specific to Solstice Enterprise Manager (Solstice EM) (which are not part of any standards) are documented in the second section of this appendix.

This appendix describes the following topics:

The following is a list of IDLs that the SEM CORBA Gateway implementation includes:

1. From JIDM standards:

- `ASN1Limits.idl`
- `JIDM.idl`
- `X501Inf.idl`
- `ASN1Types.idl`
- `OSIMgmt.idl`
- `X711CMI.idl`

2. From OMG standards:

- `CosLifeCycle.idl` (Life Cycle Service)
- `CosNaming.idl` (Naming Service)
- `CosEventChannelAdmin.idl` (Event Channel Admin)
- `CosEventComm.idl` (Event Service)

3. SEM CORBA Gateway specific:

- `SEMAuthenticationProxy.idl`
- `EventPortRegistry.idl`
- `ASN1TypesExt.idl`
- `CMIExt.idl`

- OSIMgmtExt.idl
- SEMMetaData.idl

A.1 IDLs Based on Standards

1.ASN1Limits.idl

This IDL file defines the maximum and minimum (biggest negative) double values that your machine can hold, for IDL interfaces. Use conditional compilation to support multiple architectures.

2. JIDM.idl

This IDL file comprises a collection of interfaces that together define a basic set of services for developing systems management applications based on CORBA. Following the JIDM reference model, these interfaces may be used between manager applications and JIDM Frameworks, or between JIDM Frameworks and agent applications.

From the Manager application perspective, the following interfaces are used:

- The ProxyAgent interface
- The ProxyAgentController interface
- The ProxyAgentFinder interface
- The EventPort interface
- The EventPortFactory interface

From the Agent application perspective, the following additional interfaces are used:

- The DomainPort interface
- The DomainPortFactory interface
- The EventPortFinder interface

3. X501Inf.idl

This IDL file contains mappings for the pre-mapped or pre-defined types as part of the JIDM standard.

4. ASN1Types.idl

This IDL file provides generic IDL mapping for those ASN1 types that do not have a generic IDL mapping in the CORBA/TMN Interworking standard (also known as JIDM). This generic mapping is also known as *Generic Translation* (GT). The mappings apply *only* to those ASN1 types that are not already mapped by the JIDM standard in X711CMI.idl and X501Inf.idl; types mapped in these IDL files are known as pre-mapped or pre-defined types.

5. `OSIMgmt.idl`

This IDL file defines the following interfaces:

- The `ProxyAgent` interface
- The `NamingContext` interface
- The `ManagedObject` interface
- The `ManagedObjectFactory` interface
- The `LocalRoot` interface
- The `LinkedReplyHandler`, `EndOfRepliesHandler` and `MultipleRepliesHandler` interfaces
- The `RepliesIterator` and `BufferedRepliesHandler` interfaces
- The `LName` interface

6. `X711CMI.idl`

This IDL file contains mappings for the pre-mapped or pre-defined types also, as part of the JIDM standard.

7. `CosLifeCycle.idl` (Life Cycle Service)

The Life Cycle Service defines the conventions used when creating, copying, deleting, and moving CORBA objects.

The IDL file defines the following interfaces:

- The `FactoryFinder` interface
- The `LifeCycleObject` interface
- The `GenericFactory` interface

8. `CosNaming.idl` (Naming Service)

The Naming Service obtains remote references to application-specific objects.

The IDL file defines the following interfaces:

- The `NamingContext` interface
- The `BindingIterator` interface
- The `NamingContextFactory` interface
- The `ExtendedNamingContextFactory` interface
- The `LNameComponent` interface
- The `LName Interface`
- The `LNameFactory` interfaces
- The `Log` interface

9. `CosEventChannelAdmin.idl` (Event Channel Administration Services)

This IDL file defines the following interfaces:

- The `ProxyPushConsumer` interface
- The `ProxyPullSupplier` interface
- The `ProxyPullConsumer` interface
- The `ProxyPushSupplier` interface

- The `ConsumerAdmin` interface
- The `SupplierAdmin` interface
- The `EventChannel` interface
- The `EventChannelFactory` interface
- The `EventFactory` interface

10. `CosEventComm.idl` (Event Service)

The Event Service provides a decoupled communication channel between CORBA objects.

The IDL file defines the following interfaces:

- The `PushConsumer` interface
- The `PushSupplier` interface
- The `PullSupplier` interface
- The `PullConsumer` interface

A.2 IDLs Specific to SEM CORBA Gateway

This section documents each of the IDLs listed above in the SEM CORBA Gateway-specific list.

`1.SEMAuthenticationProxy.idl`

This IDL file implements an interface called `AuthenticationProxy`. The methods provided by this interface serve the following functionality:

- To check if access control is turned in the MIS
- To get the User Id and password from the client
- To authenticate the user on the MIS host

The client application developer can use the `AuthenticationClientBody` class to interact with `AuthenticationProxy`.

`2. EventPortRegistry.idl`

This IDL file contains one interface specified by name: `EventPortRegistry`. This is implemented by the Event Port Registry gateway (EPR). The methods provided by the interface serve the following functionality:

- Creation of `EventPort`s
- Find `EventPort` for the `key` and the `criteria`
- Find `EventPort` given the `AE-title`

3. ASN1TypesExt.idl, CMIExt.idl, and OSIMgmtExt.idl

These IDL files provide support for specifying the managed objects in text format and the functions supported are basically the same as those defined in the JIDM IDL files. For example, the `OSIMgmtExt.idl` extends the functions supported in `OSIMgmt.idl` by defining functions as listed below:

TABLE A-1 `OSIMgmt.idl` Functions Extended in `OSIMgtExt.idl`

OSIMgmtExt.idl	OSIMgmt.idl
cmis_get_text	cmis_get
cmis_set_text	cmis_set
cmis_create_text	cmis_create
cmis_create_sync_text	cmis_create_sync
cmis_delete_text	cmis_delete
cmis_action_text	cmis_action

The JIDM document (`OSIMgmt.idl`) defines names of managed objects specified as `CosNaming::Name` format (which complies with the OMG CORBA standards). For more details refer to the OMG Naming Service standards.

6) `SEMMetaData.idl`

This is the IDL file which contains the interface `MetaDataRepository`, along with the definition of all the ASN1 kinds that the Metadata Repository will support.

The interface `MetaDataRepository` has methods which support the following functionality.

- Get ASN1 type for the given ASN1 element
- Get textual name for an element given the ASN1 name
- Look up `Node` by *name*
- Look up `Node` by *id*
- Get textual representation of an object specified in `oid`
- Get `oid` by *name*
- Get `doc` list
- Get managed object class attributes given *oid*
- Get managed object class attributes given *name*
- Get managed object class notifications by *oid*
- Get managed object class notifications by *name*

APPENDIX **B**

Programming Techniques

The SEM CORBA Gateway ToolKit provides you with the tools you need to build and deploy the SEM CORBA Gateway and also provides you with an environment for the development of client applications that interact with Solstice Enterprise Manager (Solstice EM) Management Information Server (MIS) via the SEM CORBA Gateway.

This appendix briefly describes with examples the various steps involved in developing a client application. It also describes steps to be followed for compiling, linking and troubleshooting.

This appendix describes the following topics:

B.1 Compiling and Linking Applications

To compile and link a sample CORBA program, write a UNIX script; example follows:

CODE EXAMPLE B-1 UNIX Script for Compiling and Linking a Sample CORBA Program

```
cd $EM_HOME/src/corba_gateway
source $EM_HOME/bin/emenv.[c]sh
source $EM_HOME/bin/em_corba_env.[c]sh
make clean
make firstmake
make install/all
```

The individual lines of this script are explained as follows:

1. `cd $EM_HOME/src/corba_gateway`

Change your current directory to where your sample program resides

   a. The environment variable `$EM_HOME` is the EM install directory (e.g. `/opt/SUNWconn/em`)

   b. All CORBA sample programs reside in the `$EM_HOME/src/corba_gateway` directory

   c. Metadata Gateway sample programs reside in the `$EM_HOME/src/corba_gateway/metadata` directory

   d. Request Gateway sample programs reside in the `$EM_HOME/src/corba_gateway/requests` directory

   e. Event Gateway sample programs resides in the `$EM_HOME/src/corba_gateway/events` directory

2. `source $EM_HOME/bin/emenv.[c]sh`

Set EM environment variables, for example:

```
setenv EM_HOME          /opt/SUNWconn/em
setenv EM_MIS_HOME      /opt/SUNWconn/em
setenv EM_RUNTIME       /var/opt/SUNWconn/em
```

3. `source` $EM_HOME/bin/em_corba_env.[c]sh

Set EM CORBA environment variables, for example:

```
setenv VB_INSTALL_DIR /net/mars/export/tools/inprise/vbroker
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${VB_INSTALL_DIR}/lib
setenv OSAGENT_PORT 14567
```

4. `make clean`

Remove object files, executable files and libraries in `metadata`, `requests` and `events` directories.

The `Makefile` used is:

```
RM              = /bin/rm -f

                $(EXEC):: $(LIBS) $(OBJS)
                          $(CPLUS) $(CCFLAGS) $(OBJS) $(LIBS)
$(LDIR) $(LDLB) -o
$(EXEC)

                clean::
                        $(RM) $(EXEC)
```

5. make firstmake

   a. Generate C++ stubs in the `idl_generated/cpp` directory.

   b. Delete[1] the generated `*_c.cc` and `*_s.cc` files in the `idl_generated` directory

The `Makefile` used is:

```
# define sub-directories
        IDL_DIRS = cos jidm jidm_ext metadata_gw
        FDIRS= $(IDL_DIRS)

        firstmake include clobber Makefile:
                @for i in $(FDIRS); do \
                        (echo "----- Creating cpp stubs in $$i";\
                         cd idl_generated/cpp/$$i; pwd ;\
                         /bin/rm -rf *.idl;\
                         cp $(ROOT)/idl/$$i/*.idl ./;\
                         make .idl.module); \
                done
                @echo DONE
```

6. make install / all

   a. Compile all the CORBA sample programs in the `metadata`, `requests` and `events` directories.

   b. The generated object files, library files, and executable files will reside in the corresponding directories.

1. Only `*_c.hh` and `*_s.hh` files are generated in the `idl_generated` directory.

The `Makefile` used is:

```
CPLUS           = CC
        CCFLAGS =  -features=no%localfor,no%conststrings -
library=iostream
$($(MODE)FLAG) $($(INCLUDE_TNF_PROBES)TNF)
                   $(INCL) $(DEFN) -DCPP_5_0_OR_HIGHER  -o $@
        SRCS = \
                action_client.cc \
                action_linked_reply_handler_impl.cc

        OBJS = $(SRCS:%.cc=%.o) $(SRCS.c:%.c=%.o)

        LIBS = \
                $(LIBRARY)

        LDIR = \
                -L$(ROOT)/lib \
                $(LD_PATH)

        LDLB = \
                $(LD_LIBRARY) \
                $(MORE_LIBS)

        all:: $(EXEC)
        $(EXEC):: $(LIBS) $(OBJS)
                  $(CPLUS) $(CCFLAGS) $(OBJS) $(LIBS) $(LDIR)
$(LDLB) -o $(EXEC)
:
```

CODE EXAMPLE B-2 Getting Root Naming Context

```
CosNaming::NamingContext_ptr
CORBAGatewayConnection::get_root_naming_context() throw() {
    if (CosNaming::NamingContext::_nil() != root_nc_) {
        return CosNaming::NamingContext::_duplicate(root_nc_);
    }

    try{
        CORBA::Object_var object =
            orb_->resolve_initial_references("NameService");
        root_nc_ = CosNaming::NamingContext::_narrow(object);

        if(CORBA::is_nil(root_nc_)) {
```

CODE EXAMPLE B-2 Getting Root Naming Context *(Continued)*

```
         cerr << "FAILED: Unable to obtain root naming context\n"
<< flush;
            exit(2);
        }
    }
    catch(const CORBA::Exception& e) {
      cerr << "FAILED: Unable to obtain root naming context:\n\t"
<< endl;
        exit(3);
    }
    cout << "PASSED: Obtained root naming context\n" << flush;

    return CosNaming::NamingContext::_duplicate(root_nc_);
}
```

CODE EXAMPLE B-3 Getting `ProxyAgentFinder`

```
JIDM::ProxyAgentFinder_ptr
CORBAGatewayConnection::get_proxy_agent_finder() throw() {
    try{
        CosNaming::Name name;
        name.length(1);
       name[0].id = CORBA::string_dup("JIDM::ProxyAgentFinder");
        name[0].kind = CORBA::string_dup("");

        CORBA::Object_var object;
        object = root_nc_->resolve(name);
        proxy_agent_finder_ =
JIDM::ProxyAgentFinder::_narrow(object);

        if (CORBA::is_nil(proxy_agent_finder_)) {
           cerr << "FAILED: Unable to obtain correct Proxy Agent
Finder\n"
                << flush;
            exit(1);
        }

        if(proxy_agent_finder_->_non_existent()) {
           cerr << "FAILED: Proxy Agent Finder does not exist\n"
<< flush;
            exit(2);
        }
    }
    catch(const CORBA::Exception& e) {
```

CODE EXAMPLE B-3 Getting ProxyAgentFinder *(Continued)*

```
        cerr << "FAILED: Unable to resolve
JIDM::ProxyAgentFinder:\n\t" << endl;
        exit(3);
    }
   cout << "PASSED: Obtained JIDM::ProxyAgentFinder reference\n"
<< flush;

    return
JIDM::ProxyAgentFinder::_duplicate(proxy_agent_finder_);
}
```

CODE EXAMPLE B-4 Getting ProxyAgent

```
JIDM::ProxyAgent_ptr
CORBAGatewayConnection::get_proxy_agent() throw() {
    JIDM::Key a_key;
    a_key.length(1);
    a_key[0].id = CORBA::string_dup(JIDM_OSI_KEY_ID);
    a_key[0].kind = CORBA::string_dup(JIDM_OSI_KEY_KIND);

    JIDM::Criteria a_criteria;
    a_criteria.length(2);
    a_criteria[0].name = CORBA::string_dup(JIDM_MANAGER_TITLE);
    a_criteria[0].value <<= (const char*)"SEM MIS";

    a_criteria[1].name = CORBA::string_dup(JIDM_USER_PROFILE);

    AuthenticationClient* ac =
        new AuthenticationClientHandle("UNIX_CLEAR");

    a_criteria[1].value =
       *(ac->get_user_profile(get_root_naming_context(), NULL));

    if(CORBA::tk_null == a_criteria[1].value.type()->kind()) {
       cerr << "FAILED: Unable to obtain encrypted user profile\n"
            << flush;
        exit(1);
    }

    cout << "PASSED: Obtained criteria from encrypting user
profile\n"
        << flush;

    try {
```

CODE EXAMPLE B-4 Getting `ProxyAgent` *(Continued)*

```
       proxy_agent_ = proxy_agent_finder_->access_domain(a_key,
a_criteria);

        if (CORBA::is_nil(proxy_agent_)) {
           cerr << "FAILED: Unable to obtain correct Proxy Agent"
                 << endl << flush;
            exit(1);
        }
        if(proxy_agent_->_non_existent()) {
            cerr << "FAILED: Proxy Agent does not exist" << endl
<< flush;
            exit(1);
        }
        cout << "PASSED: Created a new Proxy Agent\n" << flush;
    }
    catch(const JIDM::InvalidKey& ue) {
        cerr << "FAILED: Key is not recognized" << endl;
        exit(1);
    }
    catch (const JIDM::InvalidCriteria& ue) {
        cerr << "FAILED: Criteria is not recognized" << endl;
        exit(1);
    }
    catch (const JIDM::CannotMeetCriteria& ue) {
       cerr << "FAILED: Proxy Agent creation criteria is not met"
<< endl;
        exit(1);
    }
    catch (const JIDM::CannotAccess& e) {
       cerr << "FAILED: Access Denied to create proxy agent" <<
endl;
        exit(1);
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED: Uncaught CORBA Exception - " <<  endl;
        exit(1);
    }

    try {
        criteria_ = proxy_agent_->access_criteria();
        cout << "PASSED: Obtained access criteria\n" << flush;
    }
    catch (const CORBA::Exception& e) {
        cerr << "FAILED:Caught CORBA Exception  " << endl;
        exit(1);
    }
```

CODE EXAMPLE B-4 Getting `ProxyAgent` *(Continued)*

```
    SampleShutdownCallback::set_proxy_agent(proxy_agent_);

    return JIDM::ProxyAgent::_duplicate(proxy_agent_);
```

B.2 Troubleshooting Gateway Processes

You can troubleshoot SEM CORBA Gateway processes by:

- Checking the log files associated with SEM CORBA Gateway processes.
- Using `em_debug` to turn on dynamic debugging in a CORBA gateway.

B.2.1 Checking the Log Files

The first thing to do when troubleshooting SEM CORBA Gateway is to check the log files associated with the SEM CORBA Gateway processes. These files contain the error messages that are logged by the Gateway. TABLE B-1 describes the log files that are generated by default by the SEM CORBA Gateway. These files are located in `/var/opt/SUNWconn/em/debug` directory.

TABLE B-1 CORBA Gateway Log Files

File	Description
`em_corba_epr.log`	Event Port Registry log file
`em_corba_rgw.log`	Request Gateway log file
`em_corba_mgw.log`	Metadata Gateway log file
`em_corba_eds1.log`	CORBA EDS 1 log file
`em_corba_eds2.log`	CORBA EDS 2 log file

You can specify different files to be used as log files by changing the values of the log file configuration variables.

CODE EXAMPLE B-5 Sample Log File Contents for RGW

```
rgw_trace: RequestSAPManager::RequestSAPManagerrgw_debug:
Successfully created User SAP
rgw_debug: Started the thread safe PMI scheduler
rgw_debug: Obtained initial name service reference
rgw_trace: JIDM::ProxyAgentController - created
rgw_debug: JIDM::ProxyAgentController is ready
rgw_debug: JIDM::ProxyAgentFinder is ready
rgw_debug: SEM::AuthenticationProxy is ready
rgw_debug: Ready to accept client requests
rgw_trace: ProxyAgentFinderImpl::access_domain() starts
rgw_trace: JIDM::ProxyAgentFinder - validating key and access
criteria
rgw_trace: JIDM::ProxyAgentFinder - authenticating user profile
in [faith MIS]
rgw_trace: ProxyAgentFinderImpl::find_matching_proxy_agent()
starts
rgw_trace: JIDMProxyAgentImpl::JIDMProxyAgentImpl() start
rgw_trace: JIDMProxyAgentImpl::JIDMProxyAgentImpl() end
rgw_debug: OSIMgmtExt::ProxyAgent is ready
rgw_trace: ProxyAgentFinderImpl::returning this -
access_domain() ends
rgw_trace: JIDMProxyAgentImpl::access_criteria() start
rgw_trace: GetPendingRequest::translate_request() start
rgw_debug: message type = get request
rgw_debug: id = 0
rgw_debug: source =
rgw_debug:   aclass = PRIM, atag = 2
  aval =
"[0xff][0xff][0x2][0xe3][0xc4][0x1][0x4][0x81][0x9e][0xe6][0xa3
]"
rgw_debug: dest =
rgw_debug:   aclass = PRIM, atag = 2
  aval =
"[0xff][0xff][0x2][0x15][0xb3][0x1][0x4][0x81][0x9e][0xe6][0xa3
]"
rgw_debug: remote =
rgw_debug:   aclass = DEF, atag = 0
  aval = Du: no data unit allocated
rgw_debug: mode = CONFIRMED
rgw_debug: app_context = Du: no data unit allocated
rgw_debug: oc =
rgw_debug:  Tag  Len Value
```

B.2.2 Using Dynamic Debugging

Occasionally, you may need to use `em_debug` to turn on dynamic debugging for a CORBA gateway. The output of `em_debug` is particularly useful if you have to file a bug against the SEM CORBA Gateway. TABLE B-2 describes the debug objects that you can enable:

TABLE B-2 CORBA Gateway Debugging Objects

Error Objects	Debug Objects	Corresponding Gateway
`rgw_error`	`rgw_debug`	Request Gateway
`mgw_error`	`mgw_debug`	Metadata Gateway
`epr_error`	`epr_debug`	Event Port Registry
`egw_error`	`egw_debug`	CORBA Event Distribution Server

The debug objects[1] print out extensive debug statements, which will involve printing the contents of a message going back and forth. The debug statements are much more than just trace, but are not the actual error indicators.

For example, to see the error messages from the Request Gateway, you can enter the following command:

```
em_debug -port 6666 -c 'on rgw_error'
```

The `em_debug` command sends messages to the window in which the command is executed. If a large number of messages will be generated, it is recommended that the output of the command be redirected to a file which can be viewed using a text-editor or by executing the `tail -f` command.

1. Enabling debug objects degrades the performance of the system because of the large number of messages that get generated. It is recommended that you disable debug objects when you no longer need the debug information.

APPENDIX **C**

`criteria` for `ProxyAgents`

The following table contains the `criteria` for `ProxyAgents`.

TABLE C-1 `criteria` for `ProxyAgents`

`criteria` **Name**	**Type of Value**	**Semantics**
`domain title`	string	Value must be the name of the MIS that client wants to connect to. This `criteria` is *mandatory.*
`gateway title`	string	Value must be the name of the gateway (typically a host name) that the client wants to connect to. The value defaults to some randomly assigned gateway, typically the gateway that is running on the MIS host. This `criteria` is *optional.* This `criteria` allows gateway instances to be distributed. This `criteria` is for future use.
`controller object`	`JIDM::ProxyAgentController`	Value must be a CORBA object reference for `JIDM::ProxyAgentController` object which is created by the manager (client) application. This `criteria` is *optional.*
`user profile`	`any`	Opaque Value. Value is used to extract the username and password. This `criteria` is *mandatory.*
`manager title`	string	Title used to denote the manager (client) which requested access to the OSI managed object domain. This `criteria` is *optional.*

Index

H

I

J

K

L

M

N

O

P

R

S

T

U

X