Managing the Cisco Voice CORBA Gateway Object Model

This chapter describes how to configure and view EM objects through the CORBA/IDL interface, and contains the following sections:

- Accessing EM Objects, page 3-2
- Working with XML Arguments, page 3-8

Accessing EM Objects

Use the **invoke** method of the ObjectAccess interface to perform operations on EM objects. This method takes the following parameters:

- Object identifier for the target object (input)
- Operation name (input)
- Input arguments specified using an XML string (input)
- Results in an XML string, and possibly an iterator (output)

ObjectAccess Interface Definition

The ObjectAccess interface is defined using the Interface Definition Language (IDL), as shown below.

```
interface resultIterator
{
   boolean getNext(in short howMany, out string outargs);
};

interface ObjectAccess {
   enum ExceptionType
   {
      EXCPT_NOT_IMPLEMENTED,
      EXCPT_INTERNAL_ERROR,
      EXCPT_INVALID_INPUT,
      EXCPT_ENTITY_NOT_FOUND,
      EXCPT_FTP_ERROR,
      EXCPT_FILEACCESS_ERROR,
      EXCPT_NOT_IN_VALID_STATE
   };

   exception invokeException
   {
      ExceptionTypetype;
      string errorReason;
   };

   boolean invoke(in string oid, in string func, in string inargs,
        out string outargs, out resultIterator reIt) raises(invokeException);
};
```

The two IDL interfaces, resultIterator and ObjectAccess, are described in subsequent sections.

ObjectAccess invoke Method

The **invoke** method provides access to all Cisco gateway device objects.

```
boolean invoke(in string oid,
    in string func,
    in string inargs,
    out string outargs,
    out resultIterator reIt)
    raises(invokeException);
```

Syntax Description

This section describes the parameters for invoke.

Specifying the OID Argument

Network management layer applications use object identifiers to specify the objects on which to perform operations. Object identifiers are exported as part of the containment hierarchy, and may be obtained by using the operations that traverse the containment hierarchy of the underlying object model.

Identify objects in the model by containment and identifier:

- Containment path—The EM site and Cisco Gateway IP address
- Object identifier—The identifier for a specific object in the containment path

The following table identifies all the containment characteristics and object identifiers for Cisco Gateway objects:

Table 3-1 Cisco Gateway Containment Path

Level 1 Objects	Level 2 Objects	Level 3 Objects
Session_Set_n[1]	Session_Set_Grp_n	Session_Set_Mgr_n
SCC_Slot_nn	Sonet_Line_n	–
	FastEther_Line_n	–
BSC_Slot_nn	DS1_Line_nn	–
	DS3_Line_nn	–
NSC_Slot_nn	DS1_Line_nn	–
DMC_Slot_nn	DS3_Line_nn	–
DlsapProfile_nn	–	–
MacsapProfile_nn	–	–

1. The nn or n suffix indicates the object number.

When specifying the containment path and OID, separate each branch in the path by a forward slash (/). For example:

```
cmgm:/CMGM_Site_default/172.16.38.201/NSC_Slot_4/DS1_Line_3
```

Specifying the func, inargs, and outargs Arguments

The func parameter of the invoke method supports the following operations. The table also summarizes inargs and outargs usage.

Table 3-2 Operation Keywords

Operation	Description
ACTION	Performs a specific action on the specified Cisco Gateway object, such as adding a DS1 line. The type of action depends on the type of object. Client applications specify the action type and attribute names and values as an XML encoded string using the inargs argument.
QUERY	Returns a list of containment paths that satisfy the conditions of the input statement. Clients can use this operation to discover the path for all objects of a specific type that satisfy filter criteria. Clients specify the scope and filter values as an XML encoded string using the inarg argument. The results of the query are returned as an XML string in the outarg argument. An iterator, described in the "ObjectAccess getNext Method" section on page 3-5, may be returned.
UPLOAD	Selects the objects in the EM's containment hierarchy according to a set of criteria, and FTPs their OIDs and attributes to the specified host. Clients specify the selection criteria as an XML encoded string, which includes the following types of information: • Starting point (root path) specified as a path to a portion of the containment hierarchy. A special value, EM:, indicates that the starting point is the absolute root, and to upload the entire tree. • FTP destination file name. • FTP host name and account information. • Selection filter.
GET	Obtains a list of attribute values for an object. The client application selects the desired object using an XML encoded string. Returns the results in an XML string that contains a list of attribute names and values for the objects.
SET	Changes one or more attributes for an object. The client application specifies a list of attribute names and values as an XML string. Cisco Voice CORBA Gateway triggers an attribute change event for objects that have changed.
BULKGET	Obtains object attributes from one or more Cisco Gateways. Clients select the desired objects and attributes using an XML encoded string. Returns the results in an XML string that contains a list of attribute names and values organized by chassis.
BULKSET	Changes object attributes in one or more Cisco Gateways. Clients specify a list of attributes and their new values as an XML string. Cisco Voice CORBA Gateway triggers an attribute change event for the object or objects that have changed.
BULKACTION	Downloads software images to multiple Cisco Gateway nodes.

For more information, see the "Creating Input Arguments" section on page 3-8.

Using the resultIterator Argument

The resultIterator parameter facilitates iteration through a sequence of operations. If a non-null iterator object is returned with the results, use the GetNext function of the iterator to obtain the next set of results.

ObjectAccess getNext Method

This method provides an iterator for return values of the **invoke** method.

Syntax

```
boolean getNext(in short howMany, out string outargs);
```

Syntax Description

The following table describes the valid arguments.

Table 3-3 Arguments for getNext

howMany	The number of additional results to return, expressed as a number
outargs	The results, encoded as an XML string

Usage Guidelines

This function takes a single parameter, an integer, that specifies how many more ResultUnit sections of the XML encoded results need to be returned. The function returns the next set of ResultUnit sections of the XML.

ObjectAccess Exceptions

ObjectAccess defines the following exception types:

Table 3-4 Exceptions

Exception	Cause
EXCPT_NOT_IMPLEMENTED	The client invokes an invalid operation.
EXCPT_INTERNAL_ERROR	A ObjectAccess internal error.
EXCPT_INVALID_INPUT	The XML string is incorrect or incomplete.
EXCPT_ENTITY_NOT_FOUND	The object does not exist.
EXCPT_FTP_ERROR	The upload or software download operation failed due to incorrect host or user information.
EXCPT_FILEACCESS_ERROR	A file access problem exists.
EXCPT_NOT_IN_VALID_STATE	A chassis is not in a valid state.

Catch these exceptions using standard procedures for the specific programming language in use.

Example Program

The following example shows how to connect to VCG Server (ObjectAccess) using naming service and also shows how to perform an operation using the invoke IDL method:

```
#define MAXCOUNT 300
#define MAXATTRCOUNT 10000

CosNaming::NamingContext_var rootContext;
CosNaming::Name_var name;
CORBA::Object_var objVar;
ObjectAccess_var objectAccessVar = NULL;

char oid[MAXCOUNT];
char inXML[MAXATTRCOUNT];
char operation[MAXCOUNT];
int intResult;
char *outargs = NULL;
resultIterator_ptr& reIt=0;

try {
    objVar = orb_ptr->resolve_initial_references("NameService");
    rootContext = CosNaming::NamingContext::_narrow(objVar);

    if (CORBA::is_nil(rootContext.in())) {
        cerr << "CosNaming::NamingContext::_narrow ";
        cerr << "returned a nil object reference" << endl;
        return obAcVar;
    } else {
        name = new CosNaming::Name(2);
        name->length(2);
        name[0].id = CORBA::string_dup("Cisco");
        name[0].kind = CORBA::string_dup("");
        name[1].id = CORBA::string_dup("ObjectAccess");
        name[1].kind = CORBA::string_dup("");
        objVar = rootContext->resolve(name);
        objectAccessVar = ObjectAccess::_narrow(objVar);

        if (!CORBA::is_nil(objectAccessVar)) {
           cout <<"Narrow to ObjectAccess works" <<endl;
        } else {
           cout <<"Failed to resolve ObjectAccess via name server" <<endl;
    }
  }
} catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
    exit(1);
} catch(...) {
    cerr << "The Use of Naming Service Failed" << endl;
    cerr << "Unexpected exception " << endl;
    exit(1);
}

if(!strcmp(operation,"QUERY")){
    intResult = objectVar->invoke(oid,"QUERY",inXML,outargs, reIt);
} else if (!strcmp(operation,"GET")) {
    intResult = objectVar->invoke(oid,"GET",inXML,outargs, reIt);
} else if (!strcmp(operation,"SET")){
    intResult = objectVar->invoke(oid,"SET",inXML,outargs, reIt);
} else if (!strcmp(operation,"ACTION")){
    intResult = objectVar->invoke(oid,"ACTION",inXML,outargs, reIt);
} else if (!strcmp(operation,"BULKACTION")){
    intResult = objectVar->invoke(oid,"BULKACTION",inXML,outargs, reIt);
} else if (!strcmp(operation,"UPLOAD")){
    intResult = objectVar->invoke(oid,"UPLOAD",inXML,outargs, reIt);
} else if (!strcmp(operation,"BULKGET")){
    intResult = objectVar->invoke(oid,"BULKGET",inXML,outargs, reIt);
} else if (!strcmp(operation,"BULKSET")){
    intResult = objectVar->invoke(oid,"BULKSET",inXML,outargs, reIt);
} else {
    cout <<"Invalid Operation" <<endl;
}
cout << "result:" << intResult << endl;
cout << "outargs:" << outargs <<endl;
```

Working with XML Arguments

The input and output for the invoke method are XML strings. This section provides an overview of XML string usage. For details, see Chapter 4, "Using Cisco Voice CORBA Gateway Operations.".

Creating Input Arguments

The ACTION, GET, SET, BULKGET, BULKSET, and QUERY operations require operation details that are encoded in an XML string. The input argument varies by operation type.

For example, an ACTION request specifies the action type and provides a list of attribute names and values:

```
<ACTIONREQUEST>
  <ACTIONNAME>actionType</ACTIONNAME>
  <OBJECTATTRIBUTE name="attributeName">attributeValue</OBJECTATTRIBUTE>
  ...
</ACTIONREQUEST>
```

A GET request specifies a list of attribute names, as follows:

```
<GETREQUEST>
  <OBJECTATTRIBUTE name="attributeName1"></OBJECTATTRIBUTE>
    <OBJECTATTRIBUTE name="attributeName2"></OBJECTATTRIBUTE>
  ...
</GETREQUEST>
```

A BULKGET request specifies a list of attribute names, organized by managed object, as follows:

```
<BULKGETREQUEST>
  <MO identity="cmgm:CMGM_Site_default/172.16.38.201">
    <OA name=attributeName1></OA>
    <OA name=attributeName2></OA>
    ...
  </MO>
  <MO identity="cmgm:CMGM_Site_default/172.16.38.111">
    <OA name=attributeName3></OA>
    ...
  </MO>
  ...
</BULKGETREQUEST>
```

These examples represent a few operations. For more information, see Chapter 4, "Using Cisco Voice CORBA Gateway Operations.".

Table 3-5 Input XML Tags

XML Tag	Description
<ACTIONREQUEST></ACTIONREQUEST>	Encloses the attributes for an ACTION operation
<ACTIONNAME></ACTIONNAME>	Encloses the type of ACTION
<GETREQUEST></GETREQUEST>	Encloses the attributes for a GET operation
<SETREQUEST></SETREQUEST>	Encloses the attributes for a SET operation
<BULKACTION></BULKACTION>	Encloses the attributes for a BULKACTION operation
<BULKGETREQUEST></BULKGETREQUEST>	Encloses the attributes for a BULKGETREQUEST operation
<BULKSETREQUEST></BULKSETREQUEST>	Encloses the attributes for a BULKSETREQUEST operation
<MO identity="value"></MO>	Specifies the identity of a Managed Object and encloses a list of attributes
<QUERY></QUERY>	Encloses the criteria for a QUERY operation
<CHILDCLASS></CHILDCLASS>	Specifies class name of the Managed Objects on which QUERY operation has to be performed
<OBJECTATTRIBUTE name="value"> </OBJECTATTRIBUTE>	Specifies the attribute name of an object for GET, SET, and ACTION operations
<OA name="value"></OA>	Specifies the attribute name of an object for BULKGETREQUEST, BULKSETREQUEST, and BULKACTION operations

Specifying Attribute Names

When creating an input argument, identify the target parameter with an attribute name. These names are based on the storage location and the relevant RFC standard. The storage location is one of the following:

- SNMP—Specifies an attribute stored in a Cisco Gateway
- LocalDB—Specifies an attribute stored in the local EM database
- EMCtlr—Specifies a controller attribute

To construct an attribute name, follow these steps:

Step 1 Look up the attribute to view or change and note its Cisco EMF attribute name and source location. Refer to Appendix A, "Cisco Media Gateway Manager 3.0/3.0.1 Attribute Summary".

Step 2 Prefix the attribute name with the desired source keyword and a colon. The source keyword is SNMP, LocalDB, or EMCtlr. For example:

```
SNMP:RFC1213-MIB.sysContact
```

Interpreting Result Strings

The ACTION, SET, GET, BULKGET, BULKSET, and QUERY operations return results as an XML string. All results are wrapped in the <OPERATIONRESULTS> and <RESULTUNIT> tags. The other XML tags vary by operation type.

For example, an ACTION request returns the results of the operation:

```
<OPERATIONRESULTS>
  <RESULTUNIT>
    <ACTIONRESULTS> Text message and status </ACTIONRESULTS>
  </RESULTUNIT>
</OPERATIONRESULTS>.
```

A GET request returns a string of attribute names and values:

```
<OPERATIONRESULTS>
       <RESULTUNIT>
    <GETRESULTS>
      <OBJECTATTRIBUTE name="attributeName">attributeValue</OBJECTATTRIBUTE>
      ...
    </GETRESULTS>
  </RESULTUNIT>
</OPERATIONRESULTS>
```

A BULKACTION request returns a string of attribute names and values grouped by path:

```
<OPERATIONRESULTS>
  <RESULTUNIT>
    <BULKACTIONRESULTS>
      <PATH name="cmgm:CMGM_Site_default/10.1.1.1">
        <OBJECTATTRIBUTE attributeName> attributeValue</OBJECTATTRIBUTE >
        ...
      </PATH>
      <PATH name="cmgm:CMGM_Site_default/10.1.1.2">
        <OBJECTATTRIBUTE attributeName> attributeValue</OBJECTATTRIBUTE >
      </PATH>
      ...
    </BULKACTIONRESULTS>
  </RESULTUNIT>
<OPERATIONRESULTS>
```

These examples represent a few operations. For more information, see Chapter 4, “Using Cisco Voice CORBA Gateway Operations.”.

The following table summarizes the results tags.

Table 3-6 Results XML Tags

XML Tag	Description
<OPERATIONRESULTS> </OPERATIONRESULTS>	Encloses the overall operation results
<RESULTUNIT></RESULTUNIT>	Encloses results units
<ACTIONRESULTS> </ACTIONRESULTS>	Encloses the data of a RESULTUNIT, in response to an ACTION operation
<GETRESULTS> </GETRESULTS>	Encloses the data of a RESULTUNIT, in response to a GET operation
<SETRESULTS> </SETRESULTS>	Encloses the data of a RESULTUNIT, in response to a SET operation
<BULKGETRESULTS> </BULKGETRESULTS>	Encloses the data of a RESULTUNIT, in response to a BULKGET operation
<BULKSETRESULTS> </BULKSETRESULTS>	Encloses the data of a RESULTUNIT, in response to a BULKSET operation
<BULKACTIONRESULTS> </BULKACTIONRESULTS>	Encloses the data of a RESULTUNIT, in response to a BULKACTION operation
<QUERYRESULTS> </QUERYRESULTS>	Encloses the QUERY results
<OBJECTATTRIBUTE name="value"> </OBJECTATTRIBUTE>	Specifies the attribute name of an object
<PATH> </PATH>	Specifies the containment path to an object

Working with Upload Strings

Upload operations are wrapped with the following XML tags. For more information, see the "Cisco Voice CORBA Gateway Upload" section on page 4-11.

Table 3-7 Upload XML Tags

XML Tag	Description
<UPLOADREQUEST></UPLOADREQUEST>	Encloses an UPLOAD request.
<ROOTPATH></ROOTPATH>	Encloses the containment PATH of the Managed Object where UPLOAD starts.
<USERNAME></USERNAME>	Encloses a username for the host used during FTP upload.
<PASSWORD></PASSWORD>	Encloses a password for the host used during FTP upload.
<FTPFILENAME></FTPFILENAME>	Encloses the destination file name on the client host for the uploaded information.
<FTPHOSTNAME> </FTPHOSTNAME>	Encloses the host name of the destination machine to be used.
<CLASSFILTER></CLASSFILTER>	Encloses filter strings that apply during UPLOAD. Only the information related to the classes and attributes specified in the class filter is uploaded.
<MO identity ="value"></ MO>	Specifies the identity of a Managed Object and encloses a list of attributes.
<TB name="value"></ TB>	Specifies the name of a table.
<ROW></ ROW>	Specifies the row in the table.
<TA name="value"></ TA>	Specifies the multiple columns for each row
<OA name="value"></OA>	Specifies name of the Object Attribute for the UPLOAD operation.
<INVENTORYUPLOAD> </INVENTORYUPLOAD>	Encloses the UPLOAD results.
<PATH></PATH>	Encloses the containment path of the Managed Object. The PATH and OID uniquely identify an object.
<OID></OID>	Encloses the object identifier of the Managed Object.