Subscribing for Cisco Voice CORBA Gateway Events

Cisco Voice CORBA Gateway sends asynchronous events to subscribed Network Management (NM) layer applications as CORBA Notifications via CORBA Notification Service. Orbix Notification Service provided by Iona Technologies is used as CORBA Notification Service.

Orbix Notification Service provides the consumer applications (notification receivers, in this case, NM layer applications) an interface to filter the notifications based on the notification content.

Cisco Voice CORBA Gateway sends the following types of events:

1. Object events (Inventory events)
2. Alarm events

Two channels are created in Orbix Notification service, Object Event Channel for inventory events and Alarm Event Channel for alarm events. NM layer applications will subscribe for notifications on these channels with Orbix Notification Service.

Cisco Voice CORBA Gateway uses Containment path (Full Distinguished Name – FDN) as the object identifier, so NM applications do not need to have knowledge of EM specific Object Identifiers to identify the managed objects.

Object Events

Three types of Object events are sent by Cisco Voice CORBA Gateway.

- Object-Create Event: Generated when an EMS object is created in the EMS object tree
- Object-Delete Event: Generated when an EMS object is deleted from EMS object tree
- AttributeValue-Change Event: Generated when there is a change in the attribute(s) of an EMS object.

Cisco Voice CORBA Gateway creates a channel in the Orbix Notification Service with the name **“cisco.mcg.eventChannel.ObjectEventChannel-Channe**l” for sending Object Events.

Format of CORBA notifications for object events is shown in the following figure. Remaining body field is present only in case of AttributeValue-Change events.

Table 5-1 CORBA Notifications for Object Events

domain_type (=MCG)	Fixed Header
event_type (=GenericObjectEvent)	
event_name (=no value)	

CLASS	Value	Filterable body fields
PATH	Value	
PARENT	Value	
OID	Value	
EVENTTYPE	Value	
TIMESTAMP	Value	

Attribute 1	Value	Remaining body
Attribute 2	Value	
Attribute n	Value	

Fixed Header:

- Domain_type – Indicates the domain type of the event and is always set to MCG
- Event_type – Specifies the type of event and for Object Events, it is always set to GenericObjectEvent
- Event_name – No value set

Filterable body fields:

- CLASS – EMS Class name of the object responsible for the event
- PATH – The containment path of the object (Fully Distinguished Name) responsible for the event
- PARENT - The containment path of the parent object
- OID – Object Identifier in hexa-decimal format (CEMF specific format)
- EVENTTYPE – Type of Object Event (Object-Create or Object-Delete or AttributeValue-Change)
- TIMESTAMP – Time stamp when the event is generated

Remaining body:

- Attribute (1…n) – List of attributes and their values that are changed.

Alarm Events

Three types of Alarm events are sent by Cisco Voice CORBA Gateway.

- Alarm-Create Event: Generated when an Alarm is created
- Alarm-Delete Event: Generated when an Alarm is deleted
- Alarm-Change Event: Generated when there is a change in the Alarm state.

Cisco Voice CORBA Gateway creates a channel in the Orbix Notification Service with the name “cisco.mcg.eventChannel.AlarmEventChannel-Channel” for sending Alarm Events.

Format of CORBA notifications for alarm events is shown in the following figure.

Table 5-2 CORBA Notifications for Alarm Events

domain_type (=MCG)	Fixed Header
event_type (=AlarmEvent)	
event_name (=no value)	

CLASS	Value	Filterable body fields
PATH	Value	
OID	Value	
TIMESTAMP	Value	
EVENTTYPE	Value	
STATUSINFO	Value	
ACKSTATUSINFO	Value	
SEQUENCNUBER	Value	
SEVERITY	Value	
PROBABLECAUSE	Value	
ADDITIONALINFO	Value	
SPECIFICPROBLEM	Value	

Fixed Header:

- Domain_type – Indicates the domain type of the event and is always set to MCG
- Event_type – Specifies the type of event and for Object Events, it is always set to AlarmEvent
- Event_name – No value set

Filterable body fields:

- CLASS – EMS Class name of the object responsible for the event
- PATH – The containment path of the object (Fully Distinguished Name) responsible for the event
- OID – Object Identifier in hexa-decimal format (CEMF specific format)
- TIMESTAMP – Time stamp when the event is generated
- EVENTTYPE – Type of Object Event (Alarm-Creation or Alarm- Deletion or Alarm-Change)
- STATUSINFO – Status of the alarm

- ACKSTATUSINFO – Acknowledgement status
- SEQUENCENUMBER – Sequence number of the alarm
- SEVERITY – Severity of the alarm
- PROBABLECAUSE – Probable cause of the alarm
- ADDITIONAL INFO – Additional information regarding the alarm

Writing an Event Consumer Application

Writing a consumer application requires the Notification Service libraries and IDL files. Cisco Voice CORBA Gateway supports push consumers. The consumer for Cisco Voice CORBA Gateway's event channels is same as writing any other consumer applications.

The following steps are involved in writing a consumer application:

- Initialize the ORB

```
orb = CORBA::ORB_init(argc, argv);
CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
root_poa = PortableServer::POA::_narrow(obj);
poa_manager = root_poa->the_POAManager();
poa_manager->activate();
```

- Get the reference to the required channel in the notification service

  Reference to the Alarm Event Channel is created in this example and is based on reading the IOR of the channel from a file. Instead, naming service can be used to achieve the same purpose.

```
CosNotifyChannelAdmin::EventChannel_var
get_event_channel(CORBA::ORB_ptr orb, CORBA::ULong channel_num)
{
    CosNotifyChannelAdmin::EventChannel_var ec;
    CosNotifyChannelAdmin::ChannelID id;
    CosNotification::AdminProperties init_admin(0);

    CosNotification::QoSProperties prop(3);
    prop.length(3);

    prop[0].name =
         CORBA::string_dup(CosNotification::EventReliability);
    prop[0].value <<= (CORBA::Short)CosNotification::BestEffort;
    prop[1].name =
         CORBA::string_dup(CosNotification::ConnectionReliability);
    prop[1].value <<= (CORBA::Short)CosNotification::BestEffort;

    prop[2].name = CORBA::string_dup("MaxRetries");
    prop[2].value <<= (CORBA::ULong)3;

    try
    {
         ifstream is("/AlarmEventChannel.ref");
         char ior_ref[5000];

         if(is.good()) {
              is >> ior_ref;
         } else {
              cout << "Error opening AlarmEventChannel IOR!" << endl;
              exit(-1);
         }
         ec = CosNotifyChannelAdmin::EventChannel::_narrow(
                                      orb->string_to_object(ior_ref));.
    } catch (CORBA::SystemException& se) {
         cerr << "System exception occurred while creating channel: "
              << se << endl;
         exit(1);
    }
    return ec._retn();
}
```

- Connect the consumer to the channel

  There is only one channel in the current example. NotifyPushConsumer_i is the class that is implementing the method for processing the events.

```
void connect_event_consumers(
                   CosNotifyChannelAdmin::EventChannel_ptr channel,
                   CORBA::ULong num_channel)
{
    CosNotifyChannelAdmin::ConsumerAdmin_var ca =
         channel->default_consumer_admin();
    assert(!CORBA::is_nil(ca));
    consumers[num_channel] = new NotifyPushConsumer_i[num_consumers];

    for (CORBA::ULong i = 0; i < num_consumers; i++) {
         consumers[num_channel][i].my_POA(root_poa);
         consumers[num_channel][i].myid(
              root_poa->activate_object(&consumers[num_channel][i]));

         CosNotifyChannelAdmin::ProxyID proxy_id;
         CosNotifyChannelAdmin::ClientType ctype =
              CosNotifyChannelAdmin::SEQUENCE_EVENT;

         CosNotifyChannelAdmin::ProxySupplier_var obj =
              ca->obtain_notification_push_supplier(ctype, proxy_id);

         CosNotifyChannelAdmin::SequenceProxyPushSupplier_ptr pps =
         CosNotifyChannelAdmin::SequenceProxyPushSupplier::_narrow(obj);

    // Set PacingInterval and MaxBatchSize QoS on the proxy
    CosNotification::QoSProperties prop(2);
    prop.length(2);

    IT_Duration pi_tmp(1000000, 0); // wait a long time
    TimeBase::TimeT pacing_interval = pi_tmp.m_ticks;
    prop[0].name = CORBA::string_dup(
    CosNotification::PacingInterval );
    prop[0].value <<= pacing_interval;

    prop[1].name = CORBA::string_dup(
    CosNotification::MaximumBatchSize );
    prop[1].value <<= CORBA::Long( num_events );

    pps->set_qos( prop );

    consumers[num_channel][i].connect(pps);

    CosNotification::EventTypeSeq *eventType1=0;
         eventType1 = pps->obtain_offered_types(
              (CosNotifyChannelAdmin::ObtainInfoMode) 1);
         cout <<"\nto get the event types offered by "
              << "ObjectEventChannel \n" <<endl;

         CORBA::ULong leng=eventType1->length();
         cout << "the length of event types is " <<leng <<endl;

         for (int k=0; k <leng; k++ ) {
              cout << "Domain name = "<<
                   (*eventType1)[k].domain_name<<endl;
              cout << "Type name = "<< (*eventType1)[k].type_name<< endl;
         }
         CORBA::release(pps);
    }
}
```

- Implement the push consumer class that receives the events

  The class that implements the method for receiving the events has to inherit from POA_CosNotifyComm::SequencePushConsumer class. And should override the push_structured_events() method and implement the business logic of the application that processes the event.

```
void
NotifyPushConsumer_i::push_structured_events(
    const CosNotification::EventBatch& events
)
    IT_THROW_DECL((CORBA::SystemException))
{
    // Accept the incoming structured event and
    // process it.
    //
    cout << endl << "Received Event " << endl;

    bool rembody=FALSE;
    MCG::ObjectAttributes* data;

    char *event_type = 0;
    char *domain_name =0;
    char *event_name =0;
    int variableLength=0;
    int filterableLength=0;

    //cout<< "\n consumer: Received Event Batch # " << m_count++<<
endl;

    for (int i = 0; i < events.length(); i++)
    {
         domain_name =
CORBA::string_dup(events[i].header.fixed_header.event_type.domain_name)
;
    event_type =
CORBA::string_dup(events[i].header.fixed_header.event_type.type_name);
    event_name =
CORBA::string_dup(events[i].header.fixed_header.event_name);

    cout <<"\nFIXED HEADER" <<endl;
    cout << "Domain_Type: " <<domain_name <<endl;
    cout <<"Event_Type: " <<event_type <<endl;
    cout <<"Event_Name: " <<event_name <<endl;

    if (strstr(event_type, "LOSTCONNECTION")) {
         cout <<"The server is down, so the consumer exits.\n"
<<endl;
         exit(1);
    }
    variableLength = events[i].header.variable_header.length();

    cout << "\nVARIABLE HEADER" <<endl;
    cout <<"Variable header length is " <<variableLength <<endl;

    filterableLength = events[i].filterable_data.length();

    cout <<"\nFILTERABLE BODY FIELDS"<<endl;
    cout <<"Filterable body length is " <<filterableLength <<endl;

    for (int k=0; k<filterableLength; k++ )
    {
         char * cvalue;
         events[i].filterable_data[k].value >>= cvalue;
```

```
            cout << "Filterable Fields:" <<k<<": "
                 <<events[i].filterable_data[k].name << " = "<< cvalue
<<endl;

            // Remainder of the body is only applicable to
            // AttributeValue change events only
            if (!strcasecmp(cvalue, "AttributeValue-Change"))
            {
                 events[i].remainder_of_body >>= data;
                 rembody =TRUE;
            }
       }
       if (rembody)
       {
            int len = data->attrs.length();
            cout << "\nREMAINING BODY: with len= " << len << endl;

            for( int i=0; i < len; i++)
            {
                 cout << "Attr-" << i << "-Name: "
                      << data->attrs[i].name
                      << " Value: "
                      << data->attrs[i].value <<endl;
            }
          }
       }
       totalEvents += events.length();
       cout <<" Total events = " << totalEvents << endl;
   }
```

- Wait for the events

The consumer application can start waiting for the events by executing **orb->run() in the main** method. The events are asynchronously pushed to the consumer.

Example programs for the consumer application are shipped with the VCG package and are available under <VCG_INSTALL_DIR>/src directory.

The IORs of the two channels are also available in the files for convenience in the root directory of the server machine.

/AlarmEventChannel.ref – Contains IOR of Alarm Event Channel

/ObjectEventChannel.ref – Contains IOR of Object Event Channel