Platform SDK Developer's Guide

Management Layer

4/10/2025

Management Layer

You can use the Management Platform SDK to write Java or .NET applications that interact with the Genesys Message Server, Solution Control Server and Local Control Agents (LCAs). Most people will want to use this SDK to make their applications visible to the Genesys Management Layer so they can monitor them with Solution Control Server.

This document shows how to implement the basic functions you will need to write a simple voice application. It is organized to show the kind of structure you will probably use to write your own applications.

Java

Making Your Application Visible to the Genesys Management Layer

A Genesys Local Control Agent (LCA) runs on each host in the Genesys environment, enabling the Management Layer to monitor and control the applications running on that host. This section shows how to use the LCA running on your own host to make your application visible to the Genesys Management Layer.

Connecting to the Local Control Agent

The first step is to create a Local Control Agent Protocol instance, specifying the LCA port in an `Endpoint` object. This sample uses the default LCA port of 4999.

```
LocalControlAgentProtocol lcaProtocol = new LocalControlAgentProtocol(new
Endpoint("localhost", 4999));
```

Now you can configure the connection. Set the `applicationName` to the same value as the name of an application that you have set up in the Configuration Layer. Then set the application status to `Initializing`, and the execution mode to `Backup`.

```
lcaProtocol.setClientName(applicationName);
lcaProtocol.setControlStatus(ApplicationStatus.Initializing.asInteger());
lcaProtocol.setExecutionMode(ApplicationExecutionMode.Backup);
```

The next step is to set up a message handler to process events from LCA. See Event Handling articles for a better understanding of how messages and protocols should be managed. The code snippets below show how to handle events from LCA.

```
MessageHandler lcarMessageHandler = new MessageHandler() {
  public void onMessage(Message message) {
    System.out.println("Message received: \n"+message);
    //process message
```

```
  }
};
lcaProtocol.setMessageHandler(lcarMessageHandler);
```

> **Important**
>
> You need to know that your event-handling logic will be executed by using the protocol invoker. Please set the appropriate invoker for your application needs. For more information about the protocol invoker and how to set it, refer to Connecting to a Server.

Once you have finished configuring your protocol, you can open a connection to the LCA.

```
lcaProtocol.Open();
```

Updating the Application Status

When you need to update the status of your application, send a `RequestUpdateStatus`. Here is how to indicate that the application is running:

```
RequestUpdateStatus requestUpdateStatus = RequestUpdateStatus.create();
requestUpdateStatus.setApplicationName(lcaProtocol.getClientName());
requestUpdateStatus.setControlStatus(ApplicationStatus.Running.asInteger());
lcaProtocol.send(requestUpdateStatus);
```

The LCA does not return an event when you change the application status. So for this particular task, you will not need any more code.

Execution Mode and Event Handling

As mentioned, the LCA will not return an event when you change the application status. But when you change the execution mode — for example, from `Primary` to `Backup` — you will receive an `EventChangeExecutionMode`. Unlike most events you receive in the Platform SDK, this event *requires* a response from your application. If the Management Layer does not know that your application is expecting to work in `Primary` mode, for example, it cannot rely on the stability of the Genesys environment.

> **Important**
>
> If you do not respond within the configured timeout period, your application will be terminated by the Management Layer.

After receiving the `EventChangeExecutionMode`, your application must send a `ResponseExecutionModeChanged` to indicate to the Management Layer that you are now ready to run in the new execution mode.

In order to handle these events, you need to setup message handler for a LCA protocol object as shown above.

Implement your event-handling logic:

```
MessageHandler lcarMessageHandler = new MessageHandler() {
  public void onMessage(Message message) {
    switch(message.messageId()){
      case EventChangeExecutionMode.ID:
          OnEventChangeExecutionMode(message);
          break;
      // other messages
    }
  }
};
```

Here is a sample of the handler you might set up for the EventChangeExecutionMode. This handler includes your ResponseExecutionModeChanged:

```
private static void OnEventChangeExecutionMode(Message message)
{
  if(message instanceof EventChangeExecutionMode)
  {
     EventChangeExecutionMode eventChangeExecutionMode = (EventChangeExecutionMode)message;
    System.out.println("eventChangeExecutionMode received: \n"+eventChangeExecutionMode);

    ApplicationExecutionMode mode = eventChangeExecutionMode.getExecutionMode();

    ResponseExecutionModeChanged response =
ResponseExecutionModeChanged.create(mode);
    System.out.println("Sending response: " + response);
    try {
      lcaProtocol.send(response);
    } catch (ProtocolException e) {
      e.printStackTrace();
    }
  }
}
```

> ### Tip
> However, if your real application didn't successfully do the switchover, you can skip sending ResponseExecutionModeChanged. That way SCS will revert the switchover and put the application in Unknown status. That's convenient for admins as they can have alarms on that.

Closing the Connection

When you are finished, close the connection to the LCA:

```
lcaProtocol.Close();
```

Monitoring Your Application with Solution Control Server

Solution Control Server can be used to monitor applications running in the Genesys environment.

Here is how to obtain information about hosts and applications.

Connecting to Solution Control Server

Create a protocol instance and supply the necessary parameters. The `ClientName` is the name of a Solution Control application that has been set up in the Configuration Layer, while the `ClientId` is the DBID of that application.

```
SolutionControlServerProtocol  scsProtocol = new SolutionControlServerProtocol(new
Endpoint("host", port));

scsProtocol.setClientName(scsApplicationName);
scsProtocol.setClientId(scsApplicationDBid);
scsProtocol.setUserName(userName);
```

Setting Up Event Handling

You will need to set up some event handling code, since Solution Control Server will return `EventInfo` or `EventError` messages in response to your requests for information. The code for this is similar to the LCA-related code shown above.

```
MessageHandler scsrMessageHandler = new MessageHandler() {
  public void onMessage(Message message) {
    switch(message.messageId()){
      case EventInfo.ID:
          OnEventInfo(message);
          break;
      //Other events.
    }
  }
};
scsProtocol.setMessageHandler(scsrMessageHandler);

...

private static void OnEventInfo(Message message)
{
  System.out.println("Event info: \n"+message);
  //Handling logic here info.
}
```

Open Connection

Once you have configured your protocol, you can open your connection to the SCS:

```
scsProtocol.open();
```

Requesting Application Information

Here is how to request the status of an application, using its DBID:

```
RequestGetApplicationInfo requestGetApplicationInfo =
  RequestGetApplicationInfo.create(applicationDbid);

scsProtocol.send(requestGetApplicationInfo);
```

When you send this request, you will receive an EventInfo that includes the status of the application:

```
'EventInfo' ('1')
message attributes:
  attr_app_work_mode [int] = 0 [Primary]
  attr_client [int] = 660
  attr_ref_id [int] = 4
  attr_message [str] = "APP_STATUS_RUNNING"
  attr_obj_live_status [int] = 6 [Running]
  attr_cfg_obj_id [int] = 109
  attr_cfg_obj_type [int] = 9 [Application]
```

If you want to be notified when the status of an application changes, send a RequestSubscribe.

```
RequestSubscribe requestSubscribeApp = RequestSubscribe.create();
requestSubscribeApp.setControlObjectType(ControlObjectType.Application);
requestSubscribeApp.setControlObjectId(applicationDbid);
scsProtocol.send(requestSubscribeApp);
```

Whenever the application's status changes, you will receive an EventInfo that informs you of the new status.

Requesting Host Information

You can also request information about the status of a host. But in this case, you must issue a RequestSubscribe before you will receive any information about the host. Here is how:

```
RequestSubscribe requestSubscribeHost = RequestSubscribe.create();
requestSubscribeHost.setControlObjectType(ControlObjectType.Host);
requestSubscribeHost.setControlObjectId(hostDbid);
scsProtocol.send(requestSubscribeHost);

RequestGetHostInfo requestGetHostInfo = RequestGetHostInfo.create();
requestGetHostInfo.setControlObjectId(hostDbid);
scsProtocol.send(requestGetHostInfo);
```

If you just send the RequestSubscribe, you will be notified any time the host status changes. If you also send the RequestGetHostInfo, you will also receive an immediate notification of the host's status, whether it has changed or not. Here is a sample of the information you will receive.

```
'EventInfo' ('1')
message attributes:
  attr_client [int] = 660
  attr_ref_id [int] = 3
  attr_message [str] = "HOST_STATUS_RUNNING"
  attr_obj_live_status [int] = 2 [StopTransition]
  attr_cfg_obj_id [int] = 111
  attr_cfg_obj_type [int] = 10 [Host]
```

Once you have subscribed to a host, you can send a RequestGetHostInfo at any time to receive information about its status.

Closing the Connection

When you are finished, close the connection to Solution Control Server:

```
scsProtocol.close();
```

Sending a Log Message to Message Server

You can easily send log messages to Message Server using the Management Platform SDK. This sample shows how to log application events, raise alarms and view them in Solution Control Interface.

First you need to create the Protocol object:

```
MessageServerProtocol messageServerProtocol = new MessageServerProtocol( new Endpoint(new
URI(serverURI)));
```

Now you can configure the Protocol object and open the connection to Message Server. Specify the application type, application name, host on which the application is running, and application DBID. Note that the `CfgAppType` enum is defined in the Configuration Protocol package: `com.genesyslab.platform.configuration.protocol.types`

```
messageServerProtocol.setClientType(CfgAppType.CFGGenericServer.ordinal());
messageServerProtocol.setClientName ("Primary_Server_App");
messageServerProtocol.setClientHost (applicationHostName);
messageServerProtocol.setClientId(applicationDBID);
```

Now you can configure the Protocol object and open the connection to Message Server:

```
messageServerProtocol.open();
```

Create `RequestLogMessage` to log an application event. To raise an Alarm with this event, specify `requestLogMessage.EntryId` equal to the alarm detect ID. (There is more information about configuring Alarm conditions in Configuration Manager at the end of the article.)

```
RequestLogMessage requestLogMessage = RequestLogMessage.create();
requestLogMessage.setEntryId(9600);
requestLogMessage.setEntryCategory(LogCategory.Application);
requestLogMessage.setEntryText("Primary_Server_App out of service...");
requestLogMessage.setLevel(LogLevel.Alarm);
```

Once you have created the request, you can send the request to Message Server.

```
messageServerProtocol.send(requestLogMessage);
Thread.sleep(15000); //stop execution to view raised alarm in SCI
```

You can cancel an alarm after your application is restored. Specify the cancel alarm event ID and send that message to Message Server.

```
requestLogMessage = RequestLogMessage.create();
requestLogMessage.setEntryId(9601);
requestLogMessage.setEntryCategory(LogCategory.Application);
requestLogMessage.setEntryText("Primary_Server_App back in service...");
requestLogMessage.setLevel(LogLevel.Alarm);

messageServerProtocol.send(requestLogMessage);
```

When you are finished, you should close the connection:

```
messageServerProtocol.close();
```

Configuring Genesys Management Framework

You can view event logs and active alarms created by this code snippet in Solution Control Interface. However, Genesys Management Framework should be configured according to the list of required settings, below:

- Solution Control Server must be connected to Message Server. See the *Connections* tab in the Solution Control Server properties dialog.
- Solution Control Interface must be connected to Solution Control Server. See the *Connections* tab in the Solution Control Interface properties dialog.
- Solution Control Interface must be connected to the DataBase Access Point. See the *Connections* tab in the Solution Control Interface properties dialog.
- Message Server must be connected to the DataBase Access Point. See the *Connections* tab in the Message Server properties dialog.
- Message Server must have the `db_storage=true` property. See the *messages* section under *Options* of the Solution Control Interface properties dialog. This option is required to store messages in the database.
- The DataBase Access Point must be associated with DBServer. See the *General* tab of the DataBase Access Point. Check that the *DB Info* tab has the proper connection options to SQL Server.

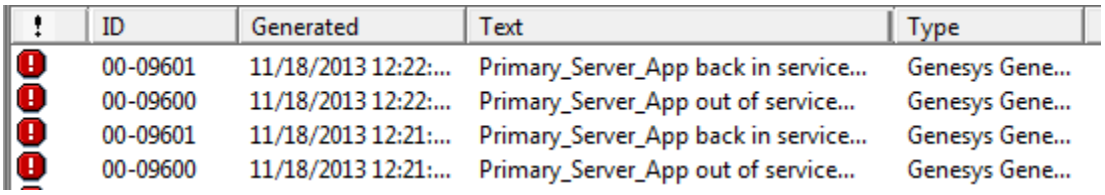
Configuring Alarm Conditions

You need to create the Alarm Condition that will trigger an Alarm for log events sent by the code snippet. To do this, open Configuration Manager, find the Alarm Conditions section and create a new Condition.

- On the *General* tab specify a condition name and description, then select `Major` for the category.
- On the *Detect Event* tab specify a Log event ID that will raise the alarm. This refers to the `RequestLogMessage.EntryId` value.
- On the *Detect Event* tab choose Select By Application for the Selection Mode and choose the application for which an event will be triggered.
- On the *Detect Event* tab specify a Log event ID that will cancel the alarm. This refers to the `RequestLogMessage.EntryId` value.

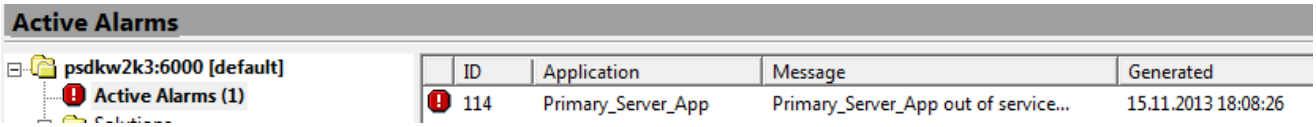
You can observe the results of running the application from Solution Control Interface.

Here is what the log messages look like in SCI:



!	ID	Generated	Text	Type
	00-09601	11/18/2013 12:22:...	Primary_Server_App back in service...	Genesys Gene...
	00-09600	11/18/2013 12:22:...	Primary_Server_App out of service...	Genesys Gene...
	00-09601	11/18/2013 12:21:...	Primary_Server_App back in service...	Genesys Gene...
	00-09600	11/18/2013 12:21:...	Primary_Server_App out of service...	Genesys Gene...

And here is what the alarm entry looks like while the alarm is active:



.NET

Making Your Application Visible to the Genesys Management Layer

A Genesys Local Control Agent (LCA) runs on each host in the Genesys environment, enabling the Management Layer to monitor and control the applications running on that host. This section shows how to use the LCA running on your own host to make your application visible to the Genesys Management Layer.

Connecting to the Local Control Agent

The first step is to create a Local Control Agent Protocol instance, specifying the LCA port in an `Endpoint` object. This sample uses the default LCA port of 4999.

```
LocalControlAgentProtocol lcaProtocol = new LocalControlAgentProtocol(new
Endpoint("localhost", 4999));
```

Now you can configure the connection. Set the `applicationName` to the same value as the name of an application that you have set up in the Configuration Layer. Then set the application status to `Initializing`, and the execution mode to `Backup`.

```
lcaProtocol.ClientName = applicationName;
lcaProtocol.ControlStatus = (int)ApplicationStatus.Initializing;
lcaProtocol.ExecutionMode = ApplicationExecutionMode.Backup;
```

The next step is to set up a message handler to process events from LCA. See Event Handling articles for a better understanding of how messages and protocols should be managed. The code snippets below show how to handle events from LCA.

```
private void OnLcaMessageReceived(object sender, EventArgs args)
{
  Console.WriteLine("New message: {0}",((MessageEventArgs)args).Message);
  // Message handling logic here.
}
. . .

lcaProtocol.Received += OnLcaMessageReceived;
```

> **Important**

> You need to know that your event-handling logic will be executed by using the protocol invoker. Please set the appropriate invoker for your application needs. For more information about the protocol invoker and how to set it, refer to Connecting to a Server.

Once you have configured your protocol, you can open your connection to the LCA:

```
lcaProtocol.Open();
```

Updating the Application Status

When you need to update the status of your application, send a `RequestUpdateStatus`. Here is how to indicate that the application is running:

```
RequestUpdateStatus requestUpdateStatus = RequestUpdateStatus.Create();
requestUpdateStatus.ApplicationName = lcaProtocol.ClientName;
requestUpdateStatus.ControlStatus = (int)ApplicationStatus.Running;
lcaProtocol.Send(requestUpdateStatus);
```

The LCA will not return an event when you change the application status. So for this particular task, you will not need any more code.

Execution Mode and Event Handling

As mentioned, the LCA will not return an event when you change the application status. But when Solution Control Server going to change your execution mode — for example, from `Primary` to `Backup` — you will receive an `EventChangeExecutionMode`. Unlike most events you receive in the Platform SDK, this event requires a response from your application. If the Management Layer does not know that your application is expecting to work in `Primary` mode, for example, it cannot rely on the stability of the Genesys environment.

> **Important**
>
> If you do not respond within the configured timeout period, your application will be terminated by the Management Layer.

After receiving the `EventChangeExecutionMode`, your application must send a `ResponseExecutionModeChanged` to indicate to the Management Layer that you are now ready to run in the new execution mode.

In order to handle these events, you need to setup message handler for a LCA protocol object as shown in the article above.

Implement your event-handling logic:

```
private void OnLcaMessageReceived(object sender, EventArgs args)
{
  IMessage message = ((MessageEventArgs)args).Message;
```

```
  switch (message.Id)
  {
    case EventChangeExecutionMode.MessageId:
        OnEventChangeExecutionMode(message);
        break;
        //  . . .
  }
}
```

Here is a sample of the handler you might set up for the EventChangeExecutionMode. This handler includes your ResponseExecutionModeChanged:

```
private void OnEventChangeExecutionMode(IMessage theMessage)
{
  EventChangeExecutionMode eventChangeExecutionMode  = theMessage as EventChangeExecutionMode;
  if (eventChangeExecutionMode != null)
  {
    ApplicationExecutionMode mode = eventChangeExecutionMode.ExecutionMode;
    ResponseExecutionModeChanged response = ResponseExecutionModeChanged.Create(mode);
    Console.WriteLine("Sending response: " + response);
    lcaProtocol.Send(response);
  }
}
```

> **Tip**
>
> However, if your real application did not successfully do the switchover, you can skip sending ResponseExecutionModeChanged. That way SCS will revert the switchover and put the application in Unknown status - a convenient event for administrators as they can have alarms trigger.

Closing the Connection

When you are finished, close the connection to the LCA:

```
lcaProtocol.Close();
```

Monitoring Your Application with Solution Control Server

Solution Control Server can be used to monitor applications running in the Genesys environment. Here is how to obtain information about hosts and applications.

Connecting to Solution Control Server

Create protocol instance and supply the necessary parameters. The ClientName is the name of a Solution Control application that has been set up in the Configuration Layer, while the ClientId is the DBID of that application:

```
var scsProtocol = new SolutionControlServerProtocol(new Endpoint("host", port));
scsProtocol.ClientName = applicationName;
scsProtocol.ClientId = applicationDBid;
scsProtocol.UserName = userName;
```

Once you have configured your protocol, you can open your connection to the SCS:

```
scsProtocol.Open();
```

Setting Up Event Handling

You will need to set up some event handling code, since Solution Control Server will return `EventInfo` or `EventError` messages in response to your requests for information. The code for this is similar to the LCA-related code shown above:

```
scsProtocol.Received += OnScsMessageReceived;
...

private void OnScsMessageReceived(object sender, EventArgs args)
{
  IMessage message = ((MessageEventArgs)args).Message;
  switch (message.Id)
  {
    case EventInfo.MessageId:
        OnEventInfo(message);
         break;
         //case ... other message
  }
}
...

private void OnEventInfo(IMessage theMessage)
{
  var eventInfo = theMessage as EventInfo;
  if (eventInfo != null)
  {
    Console.WriteLine("EventInfo received: \n{0}", eventInfo);
    // Handle this event
  }
}
```

Requesting Application Information

Here is how to request the status of an application, using its DBID:

```
var requestGetApplicationInfo = RequestGetApplicationInfo.Create(applicationDbid);
scsProtocol.Send(requestGetApplicationInfo);
```

When you send this request, you will receive an `EventInfo` that includes the status of the application:

```
'EventInfo' ('1')
message attributes:
  attr_app_work_mode [int] = 0 [Primary]
  attr_client [int] = 660
  attr_ref_id [int] = 4
  attr_message [str] = "APP_STATUS_RUNNING"
  attr_obj_live_status [int] = 6 [Running]
  attr_cfg_obj_id [int] = 109
```

```
  attr_cfg_obj_type [int] = 9 [Application]
```

If you want to be notified when the status of an application changes, send a RequestSubscribe.

```
RequestSubscribe requestSubscribeApp = RequestSubscribe.Create();
requestSubscribeApp.ControlObjectType = ControlObjectType.Application;
requestSubscribeApp.ControlObjectId = applicationDbid;
scsProtocol.Send(requestSubscribeApp);
```

Whenever the application's status changes, you will receive an EventInfo that informs you of the new status.

Requesting Host Information

You can also request information about the status of a host. But in this case, you must issue a RequestSubscribe before you will receive any information about the host. Here is how:

```
RequestSubscribe requestSubscribeHost = RequestSubscribe.Create();
requestSubscribeHost.ControlObjectType = ControlObjectType.Host;
requestSubscribeHost.ControlObjectId = HostDbid;
scsProtocol.Send(requestSubscribeHost);

RequestGetHostInfo requestGetHostInfo = RequestGetHostInfo.Create();
requestGetHostInfo.ControlObjectId = HostDbid;
scsProtocol.Send(requestGetHostInfo);
```

If you just send the RequestSubscribe, you will be notified any time the host status changes. If you also send the RequestGetHostInfo, you will also receive an immediate notification of the host's status, whether it has changed or not. Here is a sample of the information you will receive.

```
'EventInfo' ('1')
message attributes:
  attr_client [int] = 660
  attr_ref_id [int] = 3
  attr_message [str] = "HOST_STATUS_RUNNING"
  attr_obj_live_status [int] = 2 [StopTransition]
  attr_cfg_obj_id [int] = 111
  attr_cfg_obj_type [int] = 10 [Host]
```

Once you have subscribed to a host, you can send a RequestGetHostInfo at any time to receive information about its status.

Closing the Connection

When you are finished, close the connection to Solution Control Server:

```
scsProtocol.Close();
```

Sending a Log Message to Message Server

You can easily send log messages to Message Server using the Management Platform SDK. This sample shows how to log application events, raise alarms and view them in Solution Control Interface.

First you need to create the Protocol object:

```
MessageServerProtocol messageServerProtocol = new MessageServerProtocol( new Endpoint(new
Uri(serverURI)));
```

Now you can configure the Protocol object and open the connection to Message Server. Specify the application type, application name, host on which the application is running, and application DBID. Note that the `CfgAppType` enum is defined in the Configuration Protocol namespace: `Genesyslab.Platform.Configuration.Protocols.Types`

```
messageServerProtocol.ClientType = (int) CfgAppType.CFGGenericServer;
messageServerProtocol.ClientName = "Primary_Server_App";
messageServerProtocol.ClientHost = applicationHostName;
messageServerProtocol.ClientId = applicationDBID;
```

Now you can configure the Protocol object and open the connection to Message Server:

```
messageServerProtocol.Open();
```

Create `RequestLogMessage` to log an application event. To raise an Alarm with this event, specify `requestLogMessage.EntryId` equal to the alarm detect ID. (There is more information about configuring Alarm conditions in Configuration Manager at the end of the article.)

```
RequestLogMessage requestLogMessage = RequestLogMessage.Create();
requestLogMessage.EntryId = 9600;
requestLogMessage.EntryCategory = LogCategory.Application;
requestLogMessage.EntryText = "Primary_Server_App out of service...";
requestLogMessage.Level = LogLevel.Alarm;
```

Once you have created the request, you can send it to Message Server.

```
messageServerProtocol.Send(requestLogMessage);
Thread.Sleep(15000); //stop execution to view raised alarm in SCI
```

You can cancel an alarm after your application is restored. Specify the cancel alarm event ID and send that message to Message Server.

```
requestLogMessage = RequestLogMessage.Create();
requestLogMessage.EntryId = 9601;
requestLogMessage.EntryCategory = LogCategory.Application;
requestLogMessage.EntryText = "Primary_Server_App back in service...";
requestLogMessage.Level = LogLevel.Alarm;

messageServerProtocol.Send(requestLogMessage);
```

When you are finished, you should close the connection:

```
messageServerProtocol.Close();
```

Configuring Genesys Management Framework

You can view event logs and active alarms created by this code snippet in Solution Control Interface. However, Genesys Management Framework should be configured according to the list of required settings, below:

- Solution Control Server must be connected to Message Server. See the *Connections* tab in the Solution Control Server properties dialog.

- Solution Control Interface must be connected to Solution Control Server. See the *Connections* tab in the Solution Control Interface properties dialog.
- Solution Control Interface must be connected to the DataBase Access Point. See the *Connections* tab in the Solution Control Interface properties dialog.
- Message Server must be connected to the DataBase Access Point. See the *Connections* tab in the Message Server properties dialog.
- Message Server must have the `db_storage=true` property. See the *messages* section under *Options* of the Solution Control Interface properties dialog. This option is required to store messages in the database.
- The DataBase Access Point must be associated with DBServer. See the *General* tab of the DataBase Access Point. Check that the *DB Info* tab has the proper connection options to SQL Server.

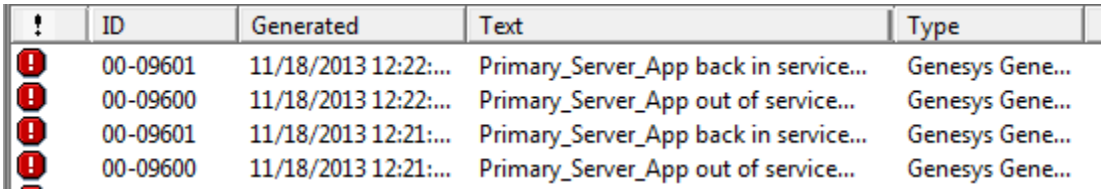
Configuring Alarm Conditions

You need to create the Alarm Condition that will trigger an Alarm for log events sent by the code snippet. To do this, open Configuration Manager, find the Alarm Conditions section and create a new Condition.

- On the *General* tab specify a condition name and description, then select `Major` for the category.
- On the *Detect Event* tab specify a Log event ID that will raise the alarm. This refers to the `RequestLogMessage.EntryId` value.
- On the *Detect Event* tab choose Select By Application for the Selection Mode and choose the application for which an event will be triggered.
- On the *Detect Event* tab specify a Log event ID that will cancel the alarm. This refers to the `RequestLogMessage.EntryId` value.

You can observe the results of running the application from Solution Control Interface.

Here is what the log messages look like in SCI:



And here is what the alarm entry looks like while the alarm is active: