Setting Up Alerting, Logging, and Error Handling

Introduction

A common problem to be solved is how to alert users about integration issues as well as how to handle errors, and log data to aid in troubleshooting. Since these topics are tightly linked, we'll address them together.

**NOTE:** This design pattern uses Design Studio as an example; you may apply the same concepts in Cloud Studio using similar steps.

Alerting

The problem is that middleware is typically a behind-the-scenes process, and needs to be instrumented such that it self reports errors or critical data. Otherwise a user is in a reactive mode, usually after a problem occurs, and is caught unawares that there is an issue. Far preferable is for the middleware to be configured to alert a user or other system based on rules. The most common rule is to alert on an error, but that is not the only scenario. Alerts can be for thresholds reached, health checks, or business-critical information.

Jitterbit has **Logging and Error Functions** as well as **System Variables** that support the construction of customized alerting, logging and error handling.

A common pattern is to identify expected error scenarios, send a short alert (usually email) and log information in greater detail.

Jitterbit's main alerting method is via email, and is configured using the Email Messages, similar to an Endpoint. Basically, Jitterbit is acting as an SMTP client and therefore must be connected to a host.

With a configured Email Message, there are several script functions that can be used to instrument alerting. Here, we use SendEmailMessage if NetSuite could not insert the record. See Email Functions for more detail.
If(jbroot$jbresponse$addListResponse$writeResponseList$writeResponse.status.isSuccess== true, WriteToOperationLog("Success Id: "+FindByPos(SourceInstanceCount(), jbroot$jbresponse$addListResponse$writeResponseList$writeResponse#.baseRef$1$RecordRef$internalId));
If(jbroot$jbresponse$addListResponse$writeResponseList$writeResponse.status.isSuccess== false, $errormessage="Time Card Entry Error: "+GetEnvironmentName() + "+ "+FindByPos(SourceInstanceCount(), jbroot$jbresponse$addListResponse$writeResponseList$writeResponse#.baseRef$1$RecordRef$internalId) + "+ Message: "+ SumString (jbroot$jbresponse$addListResponse$writeResponseList$writeResponse.status$statusDetail#.message$,".",false); WriteToOperationLog($errormessage); SendEmailMessage("<TAG>Email Messages/Jitterbit NS Data Error Message< /TAG>"
});

If you want to send an attachment, a Send Email with Attachment plug-in is available.

# Error Handling

There are integration scenarios where the number of error cases exceed the success cases, but it is reasonable to focus on the expected error scenarios during development, and add more error handling functionality as modifications after go-live.

Not all error cases require alerts, but generally do require the error to be captured and logged to aid in troubleshooting. Generally speaking, errors come in two varieties: Operation failures, and data-driven errors. The major distinction is based on the source: Operation failures are from Jitterbit itself, while data-driven errors are sourced from the endpoint and Jitterbit is simply passing it along.

## Operation Failure

Operation failure is straightforward and easy to implement, where an email message is attached to an operation and is sent if there is an operation-level failure from the Jitterbit Agent. An example is if an endpoint, like a database, is unreachable by the Agent, and a failure is returned.

## Data Driven Errors

Typical data driven scenarios include:

- Messages or status indications from the endpoint.
- Custom data validation
- Missing Data or schema mis-match

## Logging

Related closely to Alerting is Logging. Jitterbit logs operation-level information automatically. In some cases, like SFDC wizard-built operations, Jitterbit writes failure messages to the log.

A common point of confusion is how to identify the source of the error message as well as the message itself.

For example, this error is from a database:
ODBC Error: (Insert/Update mode): SQL = INSERT INTO [MD_ORDERS] ([SF_ID]) VALUES ('00617000008N2QHAA0')
SqlState = 23000 NativeError = 515

[Microsoft][ODBC SQL Server Driver][SQL Server]Cannot insert the value NULL into column 'src_Order_ID', table 'MDM_HUB.dbo.MD_ORDERS'; column does not allow nulls. INSERT fails.
SQL Exec Direct = INSERT INTO [MD_ORDERS] ([SF_ID]) VALUES ('00617000008N2QHAA0') [CODE:10618] file: DbWriter.cpp, line 339
Failed to perform transformation using local source file: C:/Windows/Temp/jitterbit/OpId_1130166_5e66d5c7-4005-4dd5-9b5e-1c0d593a11cb/1_MDMOpportunitiesSuccess

An example from NetSuite:

Connector Error: org.jitterbit.integration.server.engine.connector.exception.NetSuiteWebServiceRuntimeException: FaultString: The account you are trying to access is currently unavailable while we undergo our regularly scheduled maintenance.
FaultCode: soapenv:Server.userException
detail.code: ACCT_TEMP_UNAVAILABLE

An example from Salesforce:

Call to webservice at https://cs41.salesforce.com/services/Soap/u/33.0/00D5500000065l9 failed. Reported error:
Code: sf:INVALID_FIELD
Message: INVALID_FIELD:
Name, SBQQ__Opportunity2__r.Name, SAP_Quote_Number__c, SBQQ__Account__c
^ERROR at Row:1:Column:42
No such column 'SAP_Quote_Number__c' on entity 'SBQQ__Quote__c'. If you are attempting to use a custom field, be sure to append the '__c' after the custom field name. Please reference your WSDL or the describe call for the appropriate names.

An example from Jitterbit itself:

The operation failed unexpectedly. If this issue persists, contact Jitterbit support.
Include as much information as you can with your bug report, such as 'JITTERBIT_HOME/log/OperationEngine.log' and files in 'JITTERBIT_HOME/DataInterchange/Temp/LOG/' so that the problem can be diagnosed. Also try running the operation with debugging turned on and collect the files generated in DataInterchange/Temp/Debug/

Custom Logging

Additional logging can be added to record key data as an aid to troubleshooting. Key data can be:

- The results of a lookup in another system
- New record IDs
- If using a condition to filter out certain records, you can write out the the input and output calculations

The main function to use is the WriteToOperationLog() function, as in WriteToOperationLog("Error returned is: " + ...);

Example
WriteToOperationLog("----------" + "
" + "Int id: " + searchResponse$searchResult$recordList$record.SalesOrder$internalId + "\r\n" + "Ord id: " + searchResponse$searchResult$recordList$record.SalesOrder$tranId$ + "\r\n" + "LMD: " + searchResponse$searchResult$recordList$record.SalesOrder$lastModifiedDate$ + "\r\n" + "Loc: " + searchResponse$searchResult$recordList$record.SalesOrder$location$internalId + "\r\n" + "LMB: " + searchResponse$searchResult$recordList$record.SalesOrder$customFieldList$Last_Modified_By$value$internalId + "\r\n" + "Cust:" + searchResponse$searchResult$recordList$record.SalesOrder$entity$internalId + "\r\n");

Output Operation Log

----------
Int id: 292774
Ord id: OD003374
LMD: 2016-01-29T07:07:41-08:00
Loc: 10
LMB: 10780
Cust:2857
Skip id
----------
Int id: 297849
Ord id: EU000361
LMD: 2016-01-29T07:04:48-08:00
Loc: 17
LMB: 10780
Cust:2048
Skip id
----------
Int id: 299513
Ord id: DFW001852
LMD: 2016-01-29T07:05:15-08:00
Loc: 35
LMB: 108
Cust:12028
Pass
Customer Id: 12028-16067
--Manager: 773