CTX-Logging User Guide

Contents

Versions

Document Revisions

The following revisions have been made to this document

Date	Revision	Notes
04/12/2018	1.0	First release
04/01/2019	1.1	Changed section 1. Overview: • Added section 1.1 Using the Module
11/03/2019	2.0	Updated document to reflect the new partitioned solution

Module Versions

The following revisions have been made to this document

Date	Revision	Notes
04/12/2018	1.0	Creation of: • Logging-CL-Cortex-Log
11/03/2018	2.0	Modifications to support the change in DB Schema as part of partitioned DB Solution

Preface

About this Manual

This document is a user guide for the CTX-Logging module.

Audience

The audience for this document is those wanting to understand how to use the CTX-Logging module.

Related Material

None

Abbreviations used in this Document

SQL	Structured Query Language

Requirements

The CTX-Logging subtasks require the following:

- Minimum Cortex v6.4 installed on the Cortex Application Server
- SQL Cortex-Logging database installed

Integration

Integration with Third-Party Systems

For the CTX-Logging subtask to work, the Cortex-Logging DB must be deployed in the Cortex Environment. This is covered in the Deployment Guide.

The subtask calls 2 stored procedures which are deployed as part of the SQL Script:

- usp_CommitLog
    - This Stored Procedure handles any standard logging actions such as starting or ending a Process, Stage, or Event.
- usp_AddParameters
    - This Stored Procedure adds the parameters to a relevant Event if required, based on the optional structure input to the subtask

The script also takes care of the partitioning, based on the SQLCMD variables set at the top of the .sql file.

Integration with Existing Infrastructure

None

1 Overview

The Cortex logging module allows flow authors to easily log process information in a structured manner. The logging architecture was designed to allow for complex reporting and audit logging. Before implementing this module, considerations should be made on how the data will be used. If data is being logged for reporting, the reports should be designed beforehand.

The designed logging architecture defines services as the top-level components. Services are a set of activities delivered to an outside party, such as an end-user, customer or partner. A service is defined by the business and is further defined by processes that enable the service.

A process will span over time, has a start and end time, and can have an external reference to link it to other systems. Processes can further be decomposed into stages.

Stages also span over time and can contain multiple events, which don't span over time periods. Events can have parameters associated with them such as error messages or any other relevant information. Each parameter will have a name and a value. Events should be used to log milestones in the stage/process both on successful and exception scenarios.

Note that the Database solution uses SQL Table Partitioning – more details are available in **Appendix B: Database Data Model**.

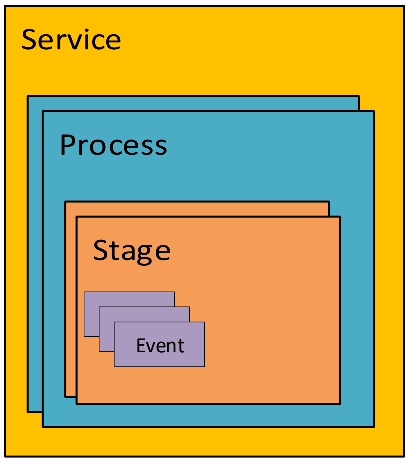


Figure 1 - Logging Architecture

1.1 Using the module

The module offers a single subtask and a database implementation which can be used to log:

- a process start and end
- a stage start and end
- an event occurrence and related parameters

The service definitions should be pre-defined in the database manually so that processes can be associated with them.

There are also some default behaviours that have been included with the module so that there is full flexibility when using it. These behaviours are described below:

- If a process is ended its child stage will be ended automatically
- If a stage is started without creating a process, the process will be created automatically with the same name as the stage
- If a stage is started for a process that already has an open stage, the currently open stage will be ended automatically.
- A stage can be linked to a process using either it's ID or external reference ID
- If an event is created without a stage or process, these items will be created automatically using the same name as the event

Logging design example

Below a practical example of how the logging can be structured. This is just an illustrative example and one possible way of setting it up. Different configurations can be used depending on the data requirements.

- The service offered is "Create Cortex Server" which is split into two automation processes:
    - Server creation:
        - this process is composed of 3 Cortex flows
        - considered critical to log the event of approving a request

          This gives the ability to create a report on approval request: time, approved by and reason (parameters stored with the event)
    - Install Software
        - this process is composed of 3 Cortex flows
        - considered critical to log separately the stage of Installing Cortex

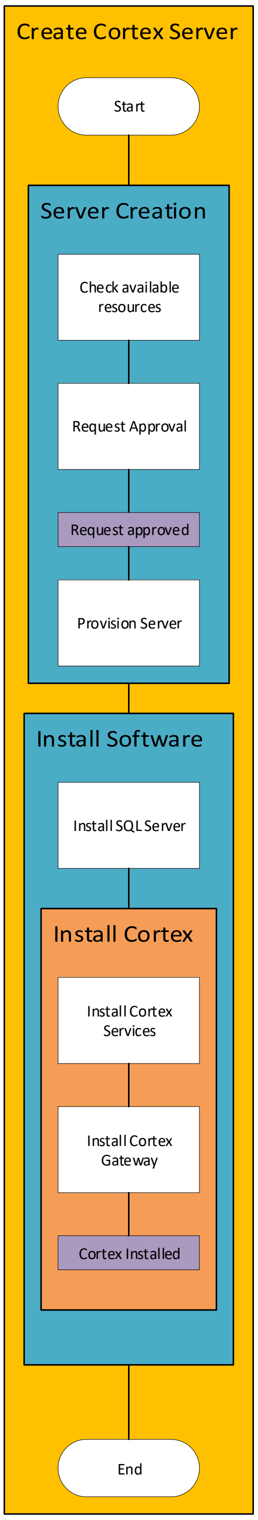
          This means the start and end time of the stage would be logged within the process
        - considered critical to log the event when Cortex installation is finished (event) and some key parameters

          This gives the ability to create a report on Cortex installations for licensing purposes

In the diagram below it is possible to see a graphical representation of how the logging module would have been applied to this scenario.

Key

Service	●
Process	●
Stage	●
Event	●
Flow	▭



2 Module usage

2.1 Logging-CL-Cortex-Logging

2.1.1 Inputs

Variable Name	Description
cl_i_Event-Name-To-Create	Name of the event that will be created
cl_i_Stage-Name-To-Create	Name of the stage that will be created
cl_i_Process-Name-To-Create	Name of the Process that will be created
cl_i_End-Process	Takes values 'yes' or 'no'. Yes will end the process. 'No' will not end the process, this is the default behaviour if a value is not provided
cl_i_End-Stage	Takes values 'yes' or 'no'. Yes will end the stage. No will not end the stage, this is the default behaviour if a value is not provided
cl_i_Log-Handler	Contains logging information, this variable should not be manually modified and should be passed in and out of all subtasks through the process. If the structure is not passed in a new one will be created automatically.
cl_i_Commit-Logs	Takes values 'yes' or 'no'. 'Yes' will commit all the logs recorded by the 'Log-Handler' and the 'Log-Handler' structure will be cleared to prevent double commits. 'No' will continue to append the 'Log-Handler' with more logs, this is the default behaviour if a value is not provided.
cl_i_Connection-String	If 'i_Commit-Logs' is set to 'yes', a connection string for the database needs to be provided. Example: Server=localhost;Database=Cortex-Logging;Trusted_Connection=True;
cl_i_Parameters	Structure of name/value pairs of parameters to be added to an event. Note: the creation of an event is mandatory
cl_i_External-Reference	A reference to the process that offers another option to link a stage to a process
cl_i_Service-ID	Optional parameters that can be provided on the create of a process to create a link to the service

cl_i_Process-ID	Optional parameter that can be provided on the creation of a stage to create a link to the process

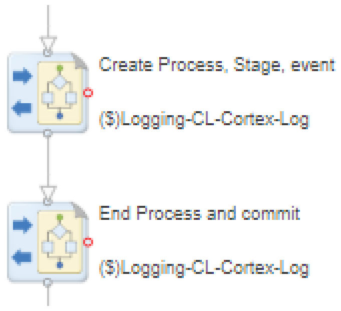
2.1.2 Outputs

Variable Name	Description
cl_o_Flow-Reference	UUID of the flow, may be useful for process linking
cl_o_Log-Handler	Contains logging information, this variable should be passed out of every subtask

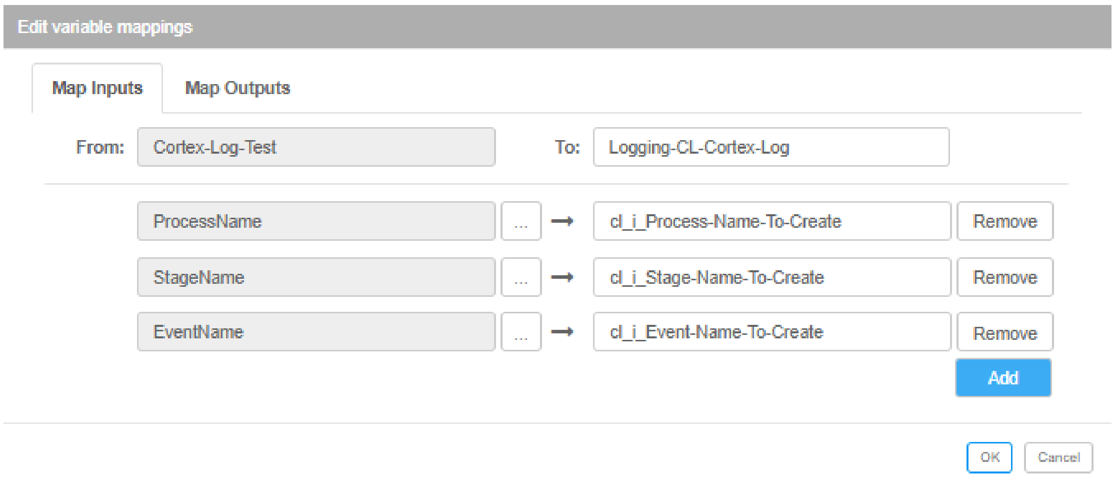
2.2 Using the subtask examples

2.2.1 Basic end to end example

Create Process, Stage and Event then End Process and commit



Subtask 1 inputs



Edit variable mappings

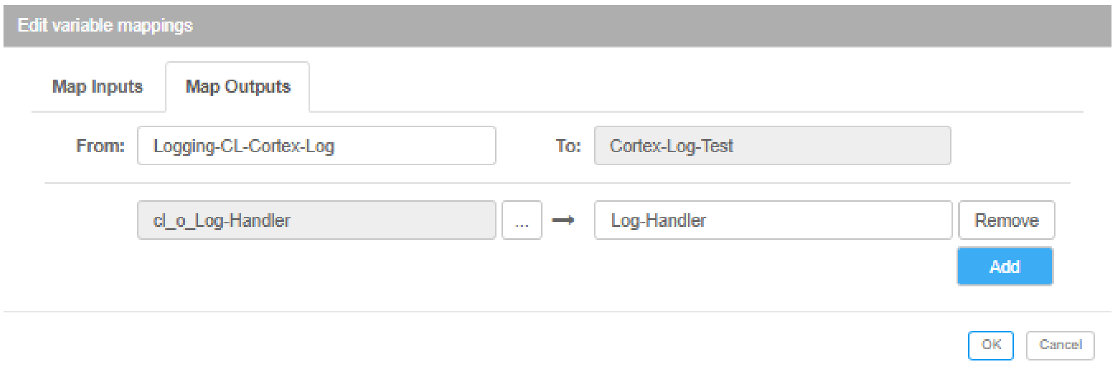
Map Inputs | Map Outputs

From: Cortex-Log-Test   To: Logging-CL-Cortex-Log

ProcessName	→	cl_i_Process-Name-To-Create
StageName	→	cl_i_Stage-Name-To-Create
EventName	→	cl_i_Event-Name-To-Create

'i_Log-Handler' is not required in the first instance of the subtask

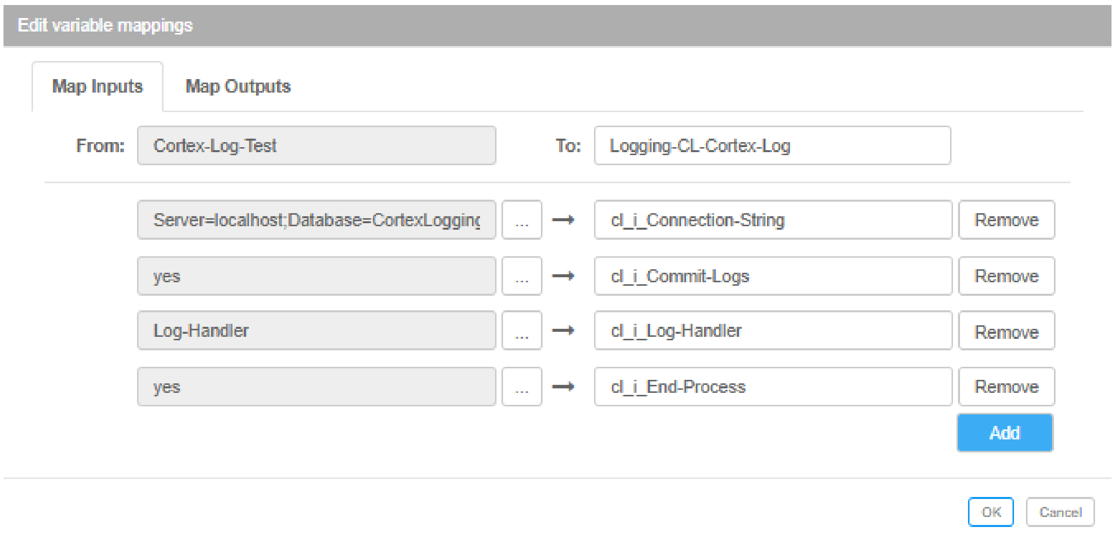
Subtask 1 outputs



From:	Logging-CL-Cortex-Log	To:	Cortex-Log-Test
	cl_o_Log-Handler	→	Log-Handler

The log-handler needs to be passed out of the subtask as it contains uncommitted logging information

Subtask 2 Inputs



From:	Cortex-Log-Test	To:	Logging-CL-Cortex-Log
	Server=localhost;Database=CortexLogginç	→	cl_i_Connection-String
	yes	→	cl_i_Commit-Logs
	Log-Handler	→	cl_i_Log-Handler
	yes	→	cl_i_End-Process

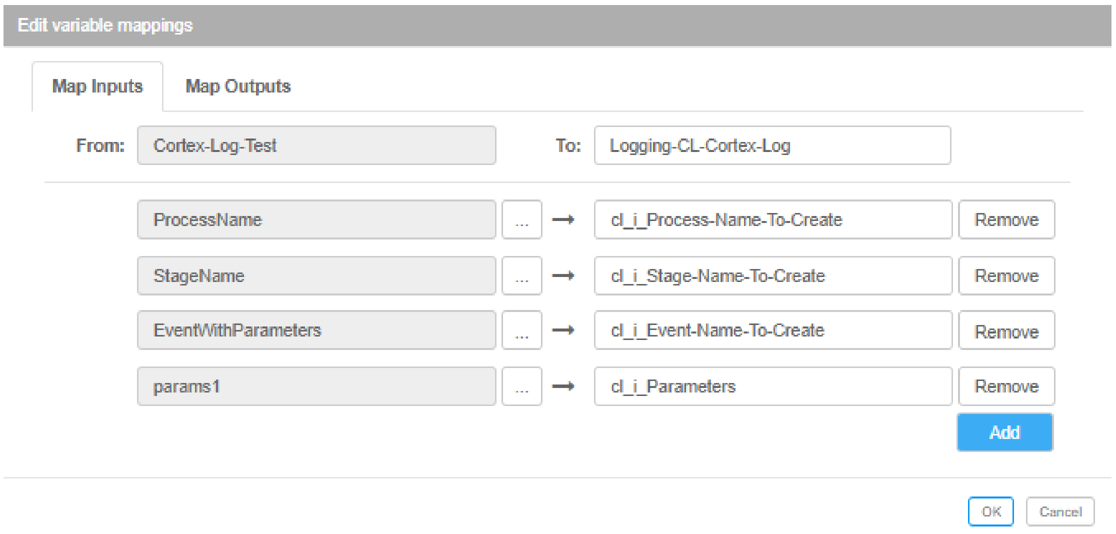
In this case the stage is not being ended explicitly, therefore it will be closed automatically when the process is ended

Subtask 2 Outputs

Because the logs have been committed no outputs are required

2.2.2 Create Process, stage and event with parameters

Subtask Inputs


Edit variable mappings

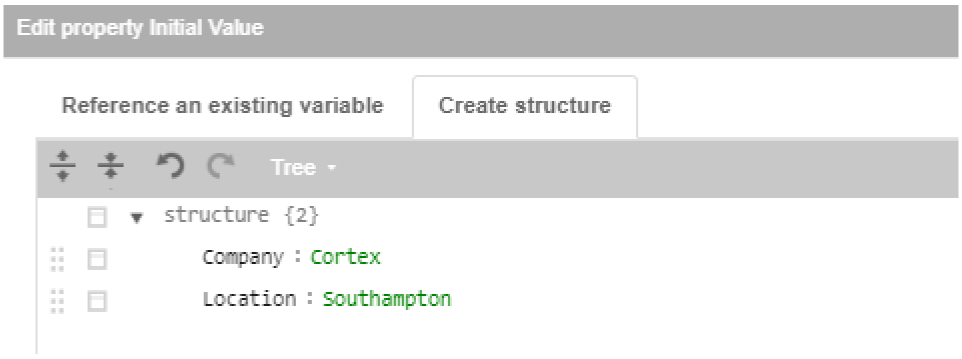
Map Inputs | Map Outputs

From: Cortex-Log-Test To: Logging-CL-Cortex-Log

ProcessName	cl_i_Process-Name-To-Create
StageName	cl_i_Stage-Name-To-Create
EventWithParameters	cl_i_Event-Name-To-Create
params1	cl_i_Parameters

In this case no log handler was passed in as it was the first instance of the subtask

Params1 variable example:


Edit property Initial Value

Reference an existing variable | Create structure

```
structure {2}
    Company : Cortex
    Location : Southampton
```

The amount of name value pairs isn't limited

2.2.3 Simple Example – User Provisioning

For this example, we have a small Service Request which will provision a new user. As this is a simple example, there will be 2 main actions – Provisioning the user in Active Directory and the creation of the Email Account.

This Service Request is contained in 1 flow which takes the inputs 'gi_FirstName' and 'gi_LastName'. This example flow can be found in the CTX-Logging Studiopkg file.

2.2.3.1 Details to Log

The below table is a representation of what we want to log in the Database:

Service	Process	Stage	Event	Parameter
N/A	Provision User	Begin Process	N/A	N/A
		Setup AD Accounts	Setup in Corporate AD	N/A
			Setup in Internal AD	N/A
		Setup Email Account	Configure Outlook Account	<email address>
			Create Default Signature	<signature>

2.2.3.2 Logging Process

Initialise Process State

The first subtask creates the Process Log and passes back the output Structure to a Global Variable, which is added to throughout the logging process. This then logs the stage 'Begin Process' - this is not a requirement but makes it clear that this is the first action.

Setup AD Accounts State

First the 'Setup AD Accounts' state is created, and then 2 events are tied to this state. By passing the Global Structure in, the logging subtask will automatically tie Events to the Stage, and the Stage to the Process
At the end, the structure is passed back into the subtask to Commit the Logs along with the value 'Yes' for Commit Logs.
Note that the relevant IDs (Process, Stage, Event) are passed back in the structure.

Setup Email Account State

Because the last logs were already committed, we need to pass in the ProcessID when creating another stage so that it is added to the same process.
We then create the Stage 'Setup Email Account' as part of the process, with 2 events.
Each event has a structure which contains the Parameter, which is also added to the log.
This is then committed, finishing our process.

2.2.4 Complex Example – Ticket Processing

A more complex example for the Logging Solution would use multiple flows, triggered from one main flow - for example, the Service would loop through open tickets and perform actions on each one (each one being a separate flow).
Each different action (flow) would be a Process tied to the same Service, and this would be split into Stages and Events as usual. The Service would need to be defined under ServiceLog.

2.2.4.1 Details to Log

Service	Process	Stage	Event	Parameter
Ticket Processing	Provision User	Setup AD Accounts	Setup in Corporate AD	N/A
			Setup in Internal AD	N/A
		Setup Email Account	Configure Outlook Account	<email address>
			Create Default Signature	<signature>
	Request Hardware	Request Approval	Get Line Manager Details	N/A
			Request Line Manager Approval	N/A
		Approval	Manager Approval	<boolean>
		Order Hardware	Request Hardware	N/A
			Inform User	N/A
	Reset Password	Get User Details	Get Username of Employee	<username>
		Reset Password	Generate Random Password	N/A
			Reset Account Password	N/A
		Send Password Details	Email Details to Employee	N/A

2.2.4.2 Logging Process

The logging process for this would be similar to the Simple Example, with each main flow (i.e. Process) having separate Process / Stage / Event / Parameter logs.

The subtasks would be setup in the same way as the simple example, using the above table for reference.

2.3 Database Queries

2.3.1 CTX-Logging View

As part of the deployment script, a view will be created. While individual implementations of this logging module may require custom queries built for reporting, this should offer an insight into the status of the platform and processes. This is named 'View-Process'.

It is suggested that this view has a clause added to it to order the results by newest first:

```
ORDER BY ProcessStartTime desc, StageStartTime desc, EventTime desc
```

2.3.2 Pivot tables

To access the parameters when reporting on the logging data pivot tables may be required, below is an example:

```
SELECT *
INTO #Temp
FROM
(
        SELECT Pa.ParameterName, Pa.ParameterValue, P.ProcessID
        FROM ProcessLog P
        INNER JOIN StageLog S
        ON P.ProcessID = S.ProcessID
        INNER JOIN EventLog E
        ON E.StageID = S.StageID
        LEFT JOIN ParameterLog Pa
        ON Pa.EventID = E.EventID
) SRC
PIVOT
(
        MAX(ParameterValue)
        FOR ParameterName IN ([Customer], [CRM_System]) --Input parameters here
) PIV

SELECT ProcessName, ProcessStartTime, ProcessEndTime, ExternalReference, T.*
FROM #Temp T
INNER JOIN ProcessLog P
ON P.ProcessID = T.ProcessID
```

3 Appendix A: Logging Subtask Usage Reference

3.1 Process

3.1.1 Create Process

To create a log Process, the only input required is the Process Name. The Output Structure should be returned to the flow.

Inputs:

- cl_i_Process-Name-To-Create – Pass in the Process Name

Outputs:

- cl_o_Log-Handler – Return this to a Global Structure

3.1.2 End Process

A log process can be ended by passing in the Logging Structure and the text ‘yes’ to the End Process input.

Inputs:

- cl_i_End-Process – Pass in the value ‘Yes’
- cl_i_Log-Handler – the Global Logging Structure

Outputs:

- cl_o_Log-Handler

3.2 Stage

3.2.1 Create Stage

Creating a Stage tied to a process can be done 2 ways:

- Passing in the Log Structure (if the logs have not yet been committed)

  This will automatically add the stage as a child item.

- Passing in the ProcessID UUID (if the Process Logs have already been committed)

  The subtask will locate this and create the child item

The Name of the Stage also needs to be provided

Inputs:

- cl_i_Stage-Name-To-Create – Pass in the Stage Name
- cl_i_Log-Handler – the Global Logging Structure
- cl_i_Process-ID (optional) – the UUID which is returned from the ‘Commit’ operation

  - This is only required if the logs have already been committed, and will ensure that the Stage is logged against the appropriate Process

Outputs:

- cl_o_Log-Handler

3.2.2 End Stage

A stage can be ended automatically if either of the following conditions are met:

1. A new Stage is created (this will end the previous one)
2. The Process is ended (this will end all)

Otherwise, the stage can be ended by the subtask. To do this, the value 'yes' must be passed in to the End Stage variable

Inputs:

- cl_i_End-Stage – Pass in the value 'Yes'
- cl_i_Log-Handler – the Global Logging Structure

Outputs:

- cl_o_Log-Handler

3.3 Event

3.3.1 Create Event Log

The Event will be automatically tied to the previous Stage. To log an Event on its own, the Event Name needs to be passed in.

Inputs:

- cl_i_Event-Name-To-Create – Pass in the Event Name
- cl_i_Log-Handler – the Global Logging Structure

Outputs:

- cl_o_Log-Handler

3.3.2 Create Event & Parameters Log

If Parameters are required too, the Structure containing these must be passed in to the Parameters input. Multiple can be provided – the Structure Attribute will be the Parameter Name and the Structure Value will be the Parameter Value.

Inputs:

- cl_i_Event-Name-To-Create – Pass in the Event Name

- cl_i_Log-Handler – the Global Logging Structure
- cl_i_Parameters – structure containing the parameter(s) to log

Outputs:

- cl_o_Log-Handler

3.4 Commit

3.4.1 Commit Current Logs

To Commit the logs, you must pass in the Logging Structure, the value 'Yes' to the Commit Logs input and a valid Connection String pointing to the Logging Database.

The Logging Structure will be cleared to prevent duplicating logs, but the IDs will be passed back (enabling more Stages to be added to the same Process).

Inputs:

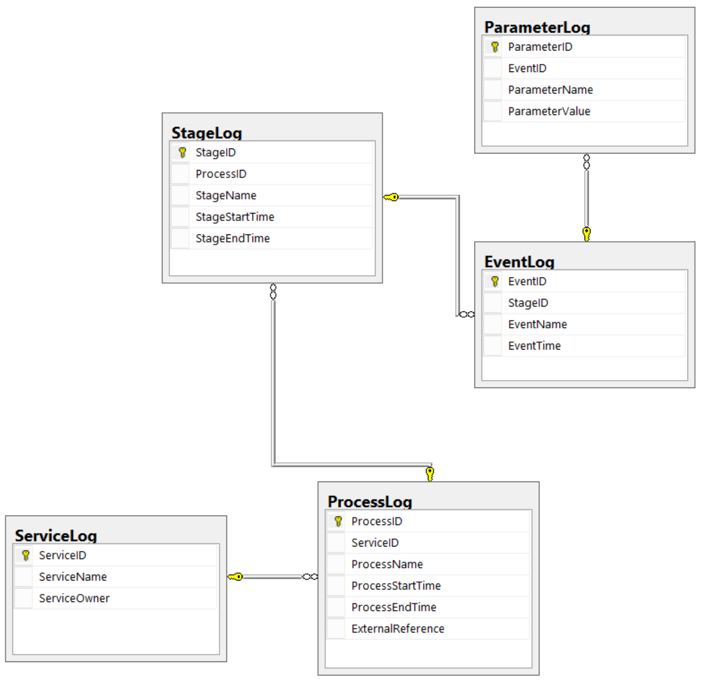
- cl_i_Commit-Logs – Pass in the value 'Yes'
- cl_i_Log-Handler – the Global Logging Structure
- cl_i_Connection-String – The Connection String for the Cortex Logging Database

Outputs:

- cl_o_Log-Handler – this will be cleared and the IDs will be added.

4 Appendix B: Database Data Model

Due to DB partitioning, all of the below 'Tables' are actually Views (except for ServiceLog). Each view will be made up of a series of tables based on the customer requirements, and each table will be split on the date.



As a result of this partitioning schema, there most of the Foreign Keys shown above are not actually in place and the database integrity is maintained by the subtasks and stored procedures. The ServiceLog field 'ServiceID' maps to the partitioned ProcessLog tables 'ServiceID' – this is the only true FK Constraint.

One example for an implementation would be to have 7 partitioned tables for each view, and each table could accept values for 24 hours. See below for an example of ProcessLog assuming that the current date is 15-03-2019.

Table	Date (ProcessStartTime)
ProcessLog_001	10-03-2019
ProcessLog_002	11-03-2019
ProcessLog_003	12-03-2019
ProcessLog_004	13-03-2019
ProcessLog_005	14-03-2019
ProcessLog_006	15-03-2019 (Current Date)
ProcessLog_007	16-03-2019 (Tomorrows Date)

This is just an example - in this scenario data would only be retained for 7 days and then would be lost. This could be improved by having more partitions per-table or by having each partition accept data for more than 24 hours. It is also suggested to have 3 extra partitions.