CTX-Logging
User Guide

Contents

CTX-Logging User Guide 1
Contents 2
Versions 3
Document Revisions 3
Module Versions 3
Preface 4
About this Manual 4
Audience 4
Related Material 4
Abbreviations used in this Document 4
Requirements 5
1 Overview 6
1.1 Using the module 6
1.2 Database Data Model 9
2 Logging-CL-Cortex-Logging 10
2.1 Inputs 10
2.2 Outputs 11
2.3 Using the subtask examples 11
2.3.1 Basic end to end example 11
2.3.2 Create Process, stage and event with parameters 12
2.4 Database Queries 13
2.4.1 Pivot tables 13

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

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

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 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

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.

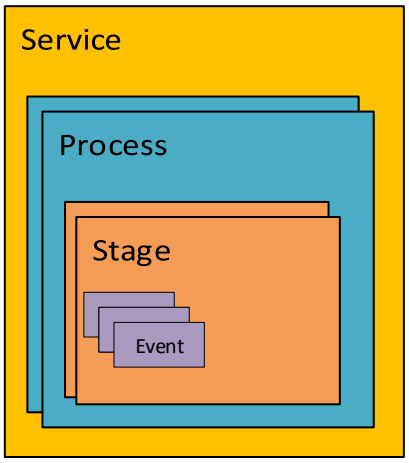


Figure 1 - Logging Architecture

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

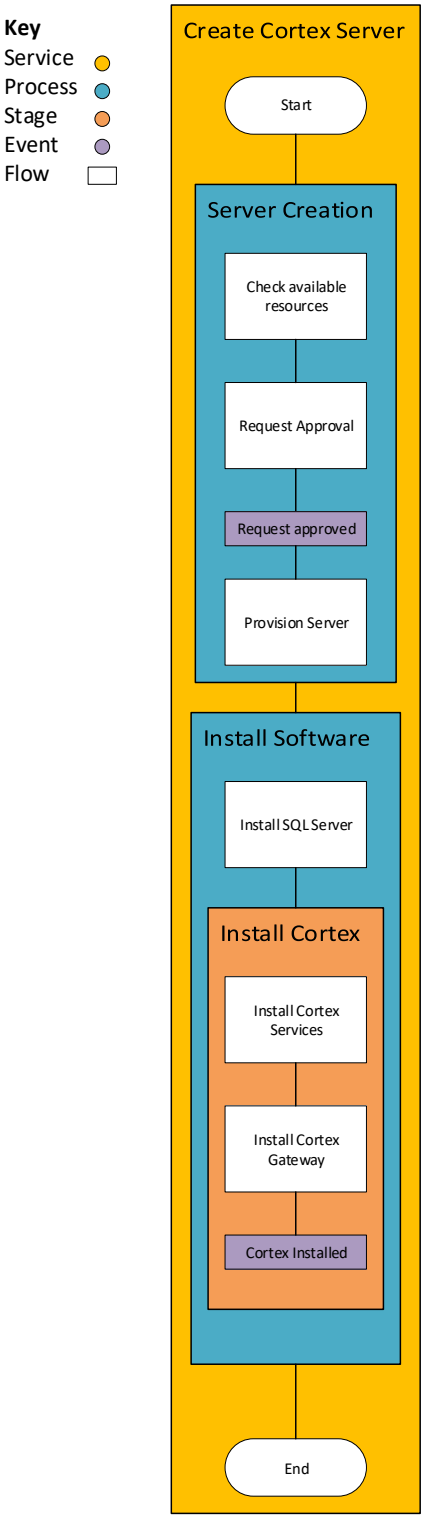
        ⚑ This gives the ability to create a report on approval request: time, approved by and reason (parameters stored with the event)
    - Install Software
        - this process is composed of 3 Cortex flows
        - considered critical to log separately the stage of Installing Cortex

        ⚑ This means the start and end time of the stage would be logged within the process

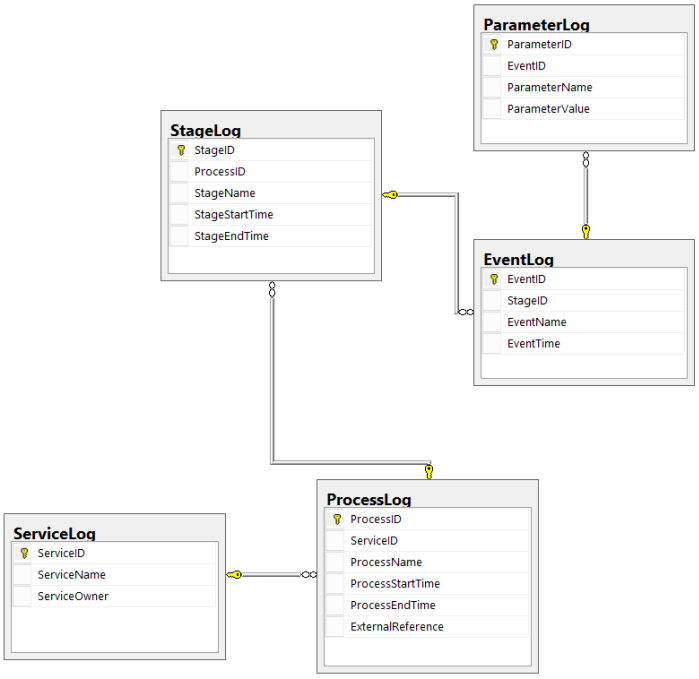
        - considered critical to log the event when Cortex installation is finished (event) and some key parameters

        ⚑ 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.



1.2 Database Data Model



2 Logging-CL-Cortex-Logging

2.1 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=CortexLogging;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

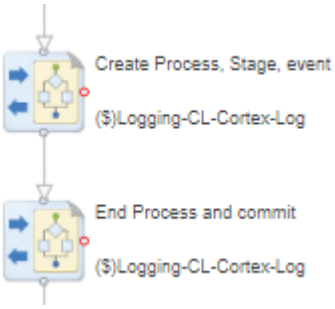
2.2 Outputs

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

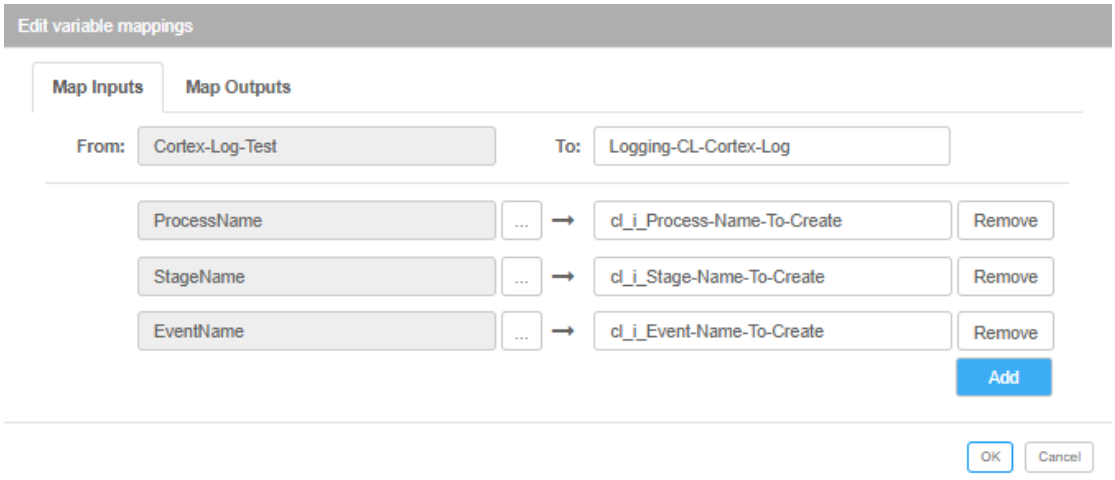
2.3 Using the subtask examples

2.3.1 Basic end to end example

Create Process, Stage and Event then End Process and commit

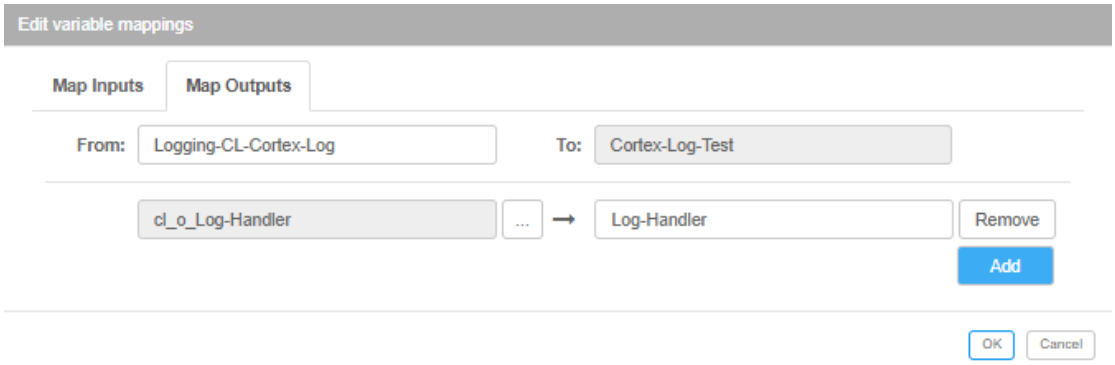


Subtask 1 inputs



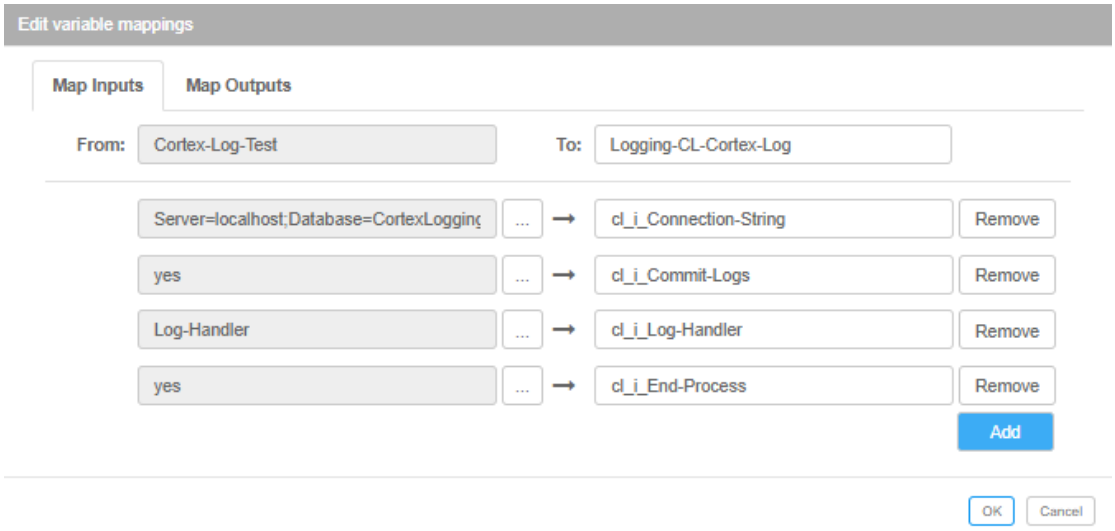
'i_Log-Handler' is not required in the first instance of the subtask

Subtask 1 outputs



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

Subtask 2 Inputs



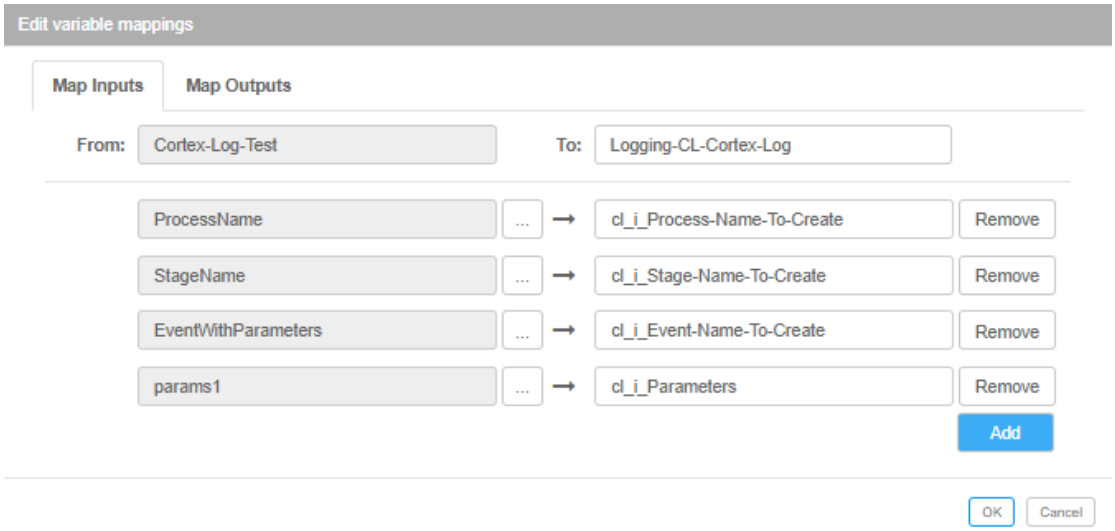
In this case the stage is not being ended explicitly, therefore it will be closed automatically when the process is ended

Subtask 2 Outputs

Because the logs have been committed no outputs are required

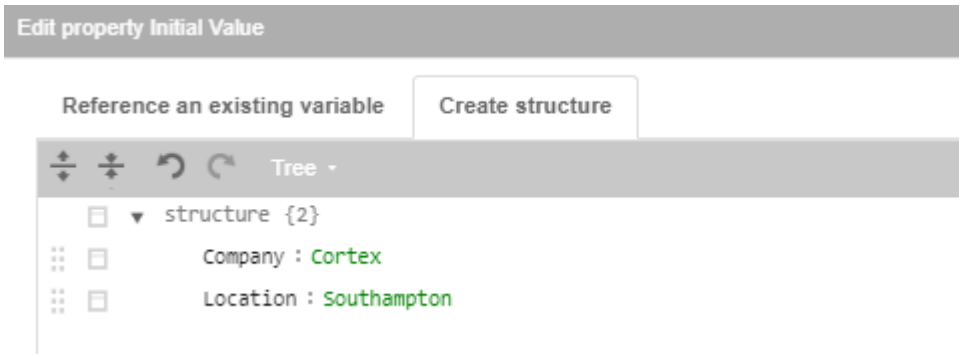
2.3.2 Create Process, stage and event with parameters

Subtask Inputs



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

Params1 variable example:



The amount of name value pairs isn't limited

2.4 Database Queries

2.4.1 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
```

