



# CTX-State-Engine User Guide

## Contents

---

CTX-State-Engine User Guide .....	1
Contents .....	2
Versions .....	5
Document Revisions .....	5
Module Versions .....	5
Preface .....	6
About this Manual .....	6
Audience .....	6
Related Material .....	6
Abbreviations used in this Document .....	6
Requirements .....	7
Integration .....	8
Integration with Third-Party Systems .....	8
Database .....	8
Flow API .....	9
Integration with Existing Infrastructure .....	9
Configuration Store Module .....	9
Logging Module .....	10
1 State Engine Overview .....	12
1.1 Process Design Example .....	13
1.2 Using the Module .....	17
1.2.1 Configuring Processes and Stages .....	19
1.2.2 Authoring flows for the State Engine .....	25
1.3 Flow/Database Architecture .....	26
1.4 User Experience .....	27
1.4.1 State Engine UI .....	27
1.4.2 Configuration Definition UI .....	38
1.4.3 Definition Update UI .....	39
1.4.4 Log Browser UI .....	41
2 State Engine Subtasks .....	42
2.1 EDBQ-State-Engine-Execute-DB-Query .....	42
2.1.1 Overview .....	42
2.1.2 Inputs .....	42
2.1.3 Outputs .....	42
2.2 ES-State-Engine-End-Stage .....	43
2.2.1 Overview .....	43

2.2.2 Inputs.....	43
2.2.3 Outputs.....	44
2.3 State-Engine-Params-to-HTML .....	45
2.3.1 Overview .....	45
2.3.2 Inputs.....	45
2.3.3 Outputs.....	45
2.4 ES-State-Engine-Escalate-Stage .....	46
2.4.1 Overview .....	46
2.4.2 Inputs.....	46
2.5 ES-State-Engine-Stage-Completed .....	47
2.5.1 Overview .....	47
2.5.2 Inputs.....	47
3 State Engine Flows .....	48
3.1 State-Engine Manage-Executions-UI.....	48
3.1.1 Overview .....	48
3.1.2 States.....	48
3.2 State-Engine-Configure-Definition-UI .....	50
3.2.1 Overview .....	50
3.2.2 States.....	50
3.3 State-Engine-Definition-Update-UI .....	51
3.3.1 Overview .....	51
3.3.2 States.....	52
3.4 State-Engine-Log-Browser-UI .....	53
3.4.1 Overview .....	53
3.4.2 States.....	53
3.5 State-Engine-Monitor-Stage-Executions .....	54
3.5.1 Overview .....	54
3.5.2 States.....	54
3.5.3 Inputs.....	55
3.6 State-Engine-Start-Process-Execution .....	56
3.6.1 Overview .....	56
3.6.2 States.....	56
3.6.3 Inputs.....	56
3.7 State-Engine-Convert-List-To-HTML .....	57
3.7.1 Overview .....	57
3.7.2 States.....	57
3.7.3 Inputs.....	57
3.7.4 Outputs.....	57

3.8 State-Engine-Convert-Structure-To-HTML .....	58
3.8.1 Overview .....	58
3.8.2 States.....	58
3.8.3 Inputs.....	58
3.8.4 Outputs.....	58
3.9 State-Engine-Disaster-Recovery.....	59
3.9.1 Overview .....	59
3.9.2 States.....	59

## Versions

---

### Document Revisions

The following revisions have been made to this document:

Date	Revision	Notes
13/05/2021	1.0	First release
02/09/2021	2.0	Second Release

### Module Versions

This version of the CTX-State-Engine deployment plan is relevant up to version 2.0 of the CTX-State-Engine module.

## Preface

---

### About this Manual

This document provides a guide on how to user the CTX-State-Engine module

### Audience

This document is intended for those wanting to understand how to use CTX-State-Engine module.

### Related Material

Document
CTX- State-Engine – Deployment Plan
State-Engine-Configuration-Bulk-Insert : Bulk-Insert-Templates for Process, Stage and Stage Start Condition csv files
Cortex-State-Engine-Export-Instructions.sql

### Abbreviations used in this Document

**SQL**      Structured Query Language

**DB**        Database

## Requirements

---

The CTX-State-Engine module requires the following:

- Minimum Cortex v6.5 installed on the Cortex Application Server
- CTX-Logging Module, including the SQL Cortex-Logging database
- CTX-Configuration-Store Module, including the SQL Cortex-ConfigStore database
- CTX-Gateway Module
- CTX-Email Module

## Integration

### Integration with Third-Party Systems

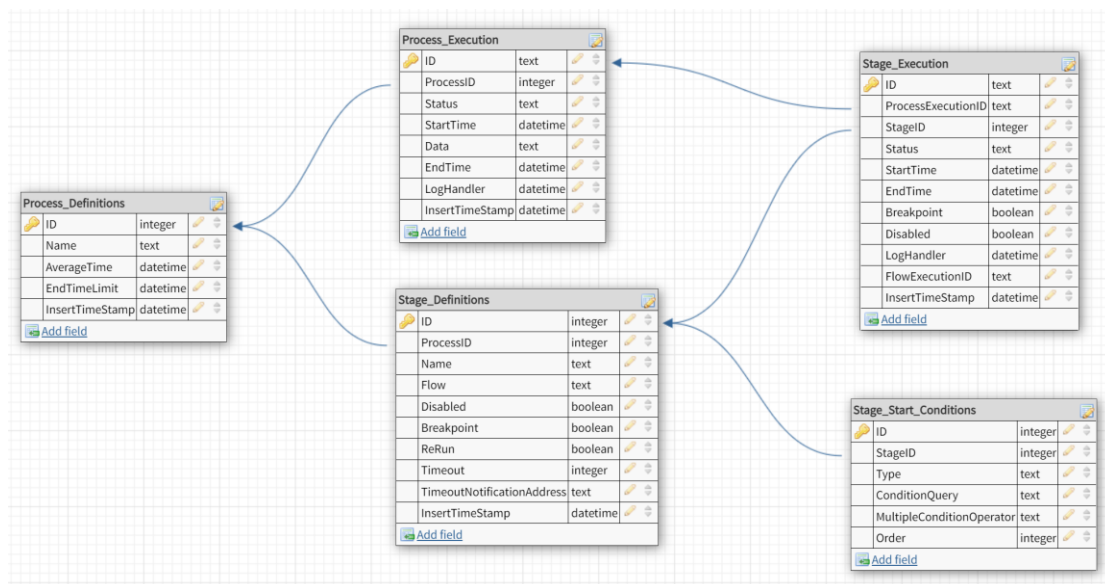
#### Database

For the flows and subtasks to function in the CTX-State-Engine module, the Cortex-State-Engine database needs to exist on the server containing the Cortex databases. Instructions on how to set up this database, including for replicated instances, can be found in the 'CTX-State-Engine - Deployment Plan' document.

The tables included in this database are:

- **Process\_Definitions** - Stores the details of processes. Should be configured by the user.
- **Stage\_Definitions** - Stores the details of stages, including what process each are a part of and what flow to call upon execution. Should be configured by the user.
- **Stage\_Start\_Conditions** - Stores the conditions which must be fulfilled for a stage execution to be started. Should be configured by the user.
- **Process\_Executions** - Stores the details of each instance of a process being executed while it is being executed. Should not be configured by the user.
- **Stage\_Executions** - Stores the details of each instance of a stage being executed while it is being executed. Should not be configured by the user.

For details on how to configure the relevant tables see section 1.2.1.





## Flow API

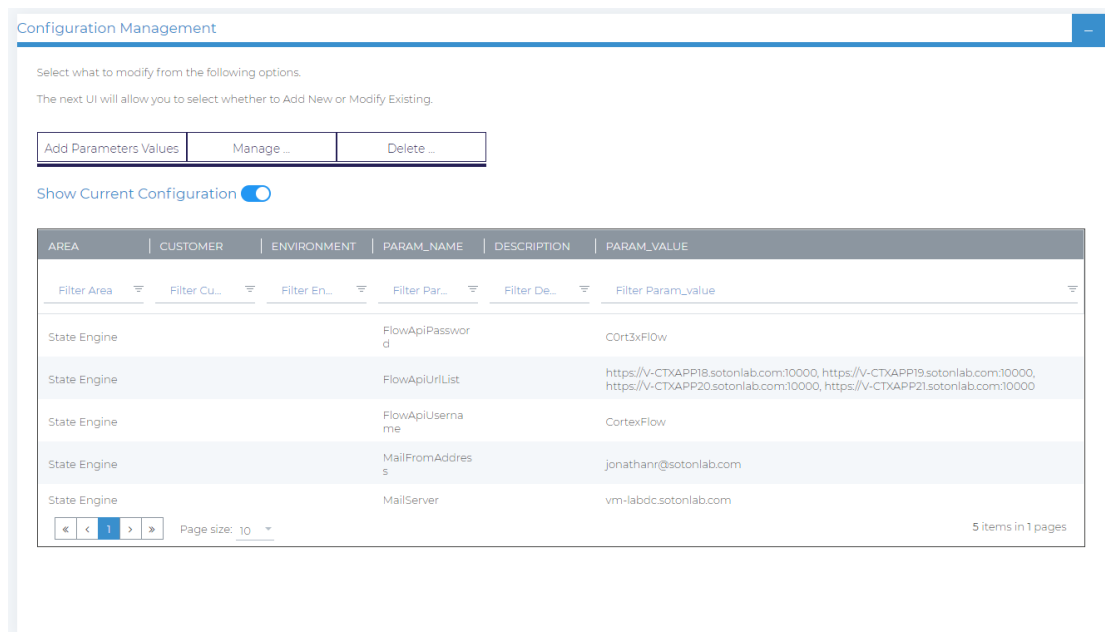
The State Engine flows call the Cortex Flow API via REST to dynamically trigger other Cortex flows on whichever application server they are run on.

Instructions on how to set up the relevant Flow API URLs, usernames and passwords can be found in the 'CTX-State-Engine - Deployment Plan' document, but an example of what they should look like in the configuration store can be found below in the relevant part of the 'Integration with Existing Infrastructure'.

## Integration with Existing Infrastructure

### Configuration Store Module

The State Engine flows use an Area in the Cortex-ConfigStore database to store the relevant configuration parameters, an example of which is shown below.



Configuration Management

Select what to modify from the following options.  
The next UI will allow you to select whether to Add New or Modify Existing.

Show Current Configuration ☒

AREA	CUSTOMER	ENVIRONMENT	PARAM_NAME	DESCRIPTION	PARAM_VALUE
State Engine			FlowApiPassword		Cort3xFI0w
State Engine			FlowApiUrlList		https://v-CTXAPP18.sotonlab.com:10000, https://v-CTXAPP19.sotonlab.com:10000, https://v-CTXAPP20.sotonlab.com:10000, https://v-CTXAPP21.sotonlab.com:10000
State Engine			FlowApiUsername		CortexFlow
State Engine			MailFromAddresses		jonathanr@sotonlab.com
State Engine			MailServer		vm-labdc.sotonlab.com

5 items in 1 pages

These parameters are required for the State Engine flows to function correctly.

**FlowApiUsername** - The default username for the flow API.

**FlowApiPassword** - The default password for the flow API.(Password can be encrypted using Cortex Encryptor utility)

**FlowApiUrlList** - A comma separated list of possible flow API urls, the State Engine flows will use the one from the list which contains the hostname of the application server they are running on.

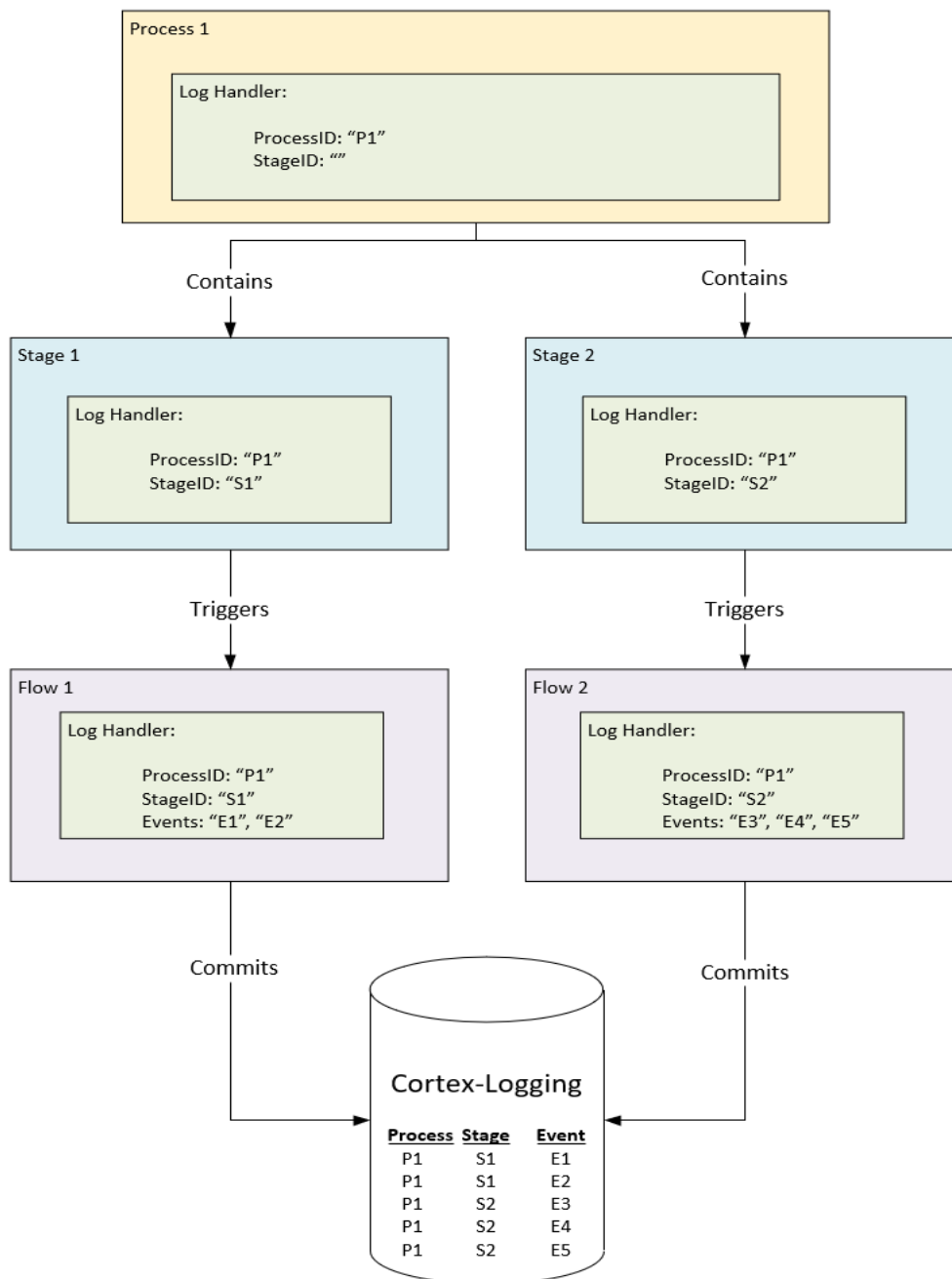
**MailFromAddress** - The email address from which to send email notifications.


**MailServer** - Fully qualified machine name of the SMTP server from which to send email notifications.

## Logging Module

The State Engine is fully integrated with the Cortex Logging module.

Whenever a process execution is started, a log handler structure is generated for it. For each stage of that process execution a Stage ID is generated, stored in a copy of this log handler, and passed to the relevant flow that is triggered by the stage execution. The triggered flow can then log whatever events it needs to against this log handler, and once committed the data in the Cortex-Logging database will match the process-stage architecture of the Cortex-State-Engine database.



 Logs for a currently executing process may be viewed using the Management UI flow, and past logs with the Log Browser UI flow

## Email Module

If a stage execution times out, an email notification will be sent to the address or addresses configured in the “TimeoutNotificationAddress” field on that stage’s record in the Stage\_Definitions table, found in the State Engine Database.

To do this, the Email-SECF-Send-Email-Cortex-Format subtask included in the CTX-Email module is used.

## Gateway Module

To trigger a dynamically chosen flow from the flow State-Engine-Monitor-Stage-Executions, the subtask SFA-Start-Flow-Async from the CTX-Gateway module is used.

## 1 State Engine Overview

---


The Cortex State Engine allows flow authors to easily execute their flows in parallel ensuring that dependencies are adhered to, as well as view the progress of the executions and manually interact with them, handling any errors that occur.


The architecture is similar to the Cortex Logging Module but simplified to only include processes, which contain one or more stages. See 'CTX-Logging User Guide' for more information on this architecture.

Each stage may be thought of as a container for a flow to be executed (each stage of a process is a container of a single solution flow) , with extra data associated with it such as a display name and a timeout period, which should be configured in the Stage\_Definitions table of the Cortex-State-Engine database. One or more start conditions may also be configured for each stage in the Stage\_Start\_Conditions table, which may be simply the completion of another stage, or any SQL query that returns a true or false value.

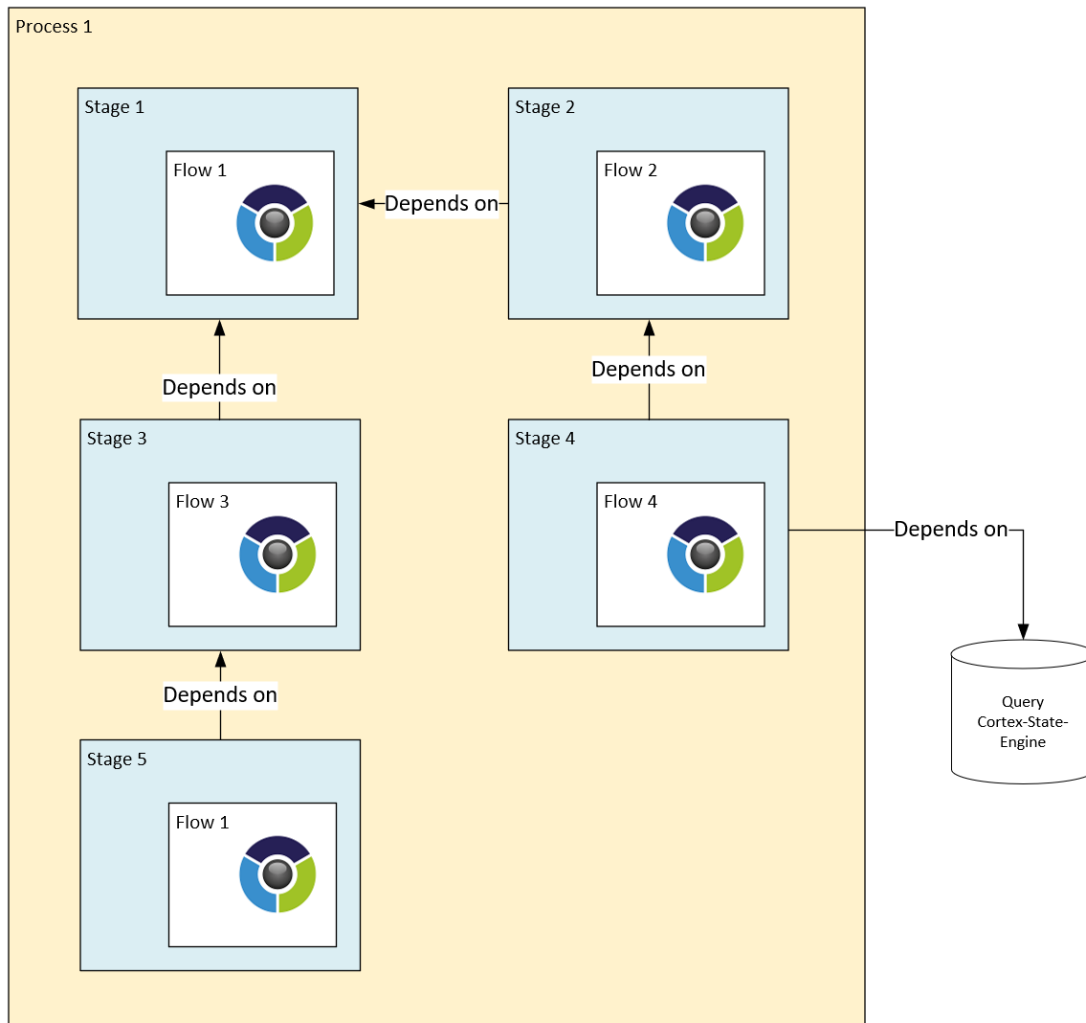
To use the module, perform the following high-level steps:

1. Ensure the prerequisite components detailed in the Requirements section are present.
2. Install everything as detailed in the CTX-State-Engine Deployment Plan, including the scheduling of the monitoring flow.
3. Configure process definitions, stage definitions and stage start conditions as in Section 1.2.1
4. For any flows to be called by the state engine, i.e. associated with a stage, ensure that they have the following:
  - a. Correct input variables, as in Section 1.2.2.1
  - b. State Engine Subtasks used correctly, as in Section 1.2.2.2
5. Using the State Engine Manage Executions UI flow: trigger, monitor and interact with process executions.


 If the monitoring flow is running, triggering a process execution will start each stage in the order configured (i.e. depending on the start conditions of each stage). When each stage is triggered, the associated flow will be called asynchronously.

 The goal of the State Engine module is to allow users to build automation flows for self-contained stages of a given manual process. Then, the order in which they are executed is handled purely by configuring the state engine database. This can then be updated at any time by a user without changing any flows.

## 1.1 Process Design Example



In the above example, Process 1 contains five stages, labelled Stage 1 to 5. Each of these has one of four flows associated with them, labelled Flow 1 to 4. Stage 1 is configured to begin immediately (as soon the process is started on demand by user or through scheduler), for information on how to do this configuration see section 1.2.1. Both Stage 2 and Stage 3 depend on the completion of Stage 1, Stage 5 only depends on the completion of stage 3, and Stage 4 depends on both the completion of Stage 2 and a query to the Cortex -State-Engine-Database returning the value 'True'.


 When a stage has multiple start conditions, for each one a logical operator may also be provided, an AND or an OR. In the above example, Stage 4 will execute if Stage 2 is completed AND the database query returns 'True'. This could easily be configured so that Stage 4 will execute if either one condition OR the other is fulfilled.



So, when executing the process, the following will occur in Cortex:

1. Flow 1 will execute.
2. On completion of Flow 1: Flow 2 and Flow 3 will execute in parallel.
3. On completion of Flow 3, irrespective of the result of Flow 2: Flow 1 will execute again since it is associated with Stage 5 as well as Stage 1.
4. On completion of this second execution of Flow 1: Flow 2 and Flow 3 will not trigger again.
5. On completion of Flow 2: Flow 4 still will not execute until the result of a user defined query to the Cortex-State-Engine Database returns 'True'.

From this, we can see that very complex parallelisation of automation and networks of dependencies can be achieved by abstracting from flows triggering each other directly to this system of processes and stages, orchestrated by the state engine flows.

 All the user has to do is create the flows to perform whatever independent automation tasks are required, then configure the process and its stages in the database to define the order of their executions. If the monitoring flow is running, triggering the process will cause each stage to run in the sequence configured, and each time a stage is executed its associated flow is triggered.

For reference, the database tables for the above configuration would look like the below:

### Process Definitions

ID	Name	AverageTime	EndTimeLimit
F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Process1	120	180

### Stage Definitions


ID	ProcessID	Name	Flow	Disabled	ReRun	Timeout	TimeoutNotificationAddress
E09DEB3A-A568-40A9-B274-091887301764	F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Stage1	Flow1	FALSE	TRUE	30	example@test.com
64CA69BA-7E9A-4BA7-A0CA-0BF366E1883D	F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Stage2	Flow2	FALSE	TRUE	30	example@test.com
6CD6414F-44AC-4522-9703-0C02652EBFB8	F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Stage3	Flow3	FALSE	TRUE	30	example@test.com
8162EB52-38B2-44EF-B442-0C13EF996A7F	F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Stage4	Flow4	FALSE	TRUE	30	example@test.com
EF37FDEF-A44F-41B6-8287-0C410B833EBA	F3711AEC-9D3A-4BAB-B53F-24E723F44E20	Stage5	Flow1	FALSE	TRUE	30	example@test.com

## Stage Start Condition

ID	StageID	Type	ConditionQuery	MultitpleConditionOperator	Order
7DC8FCB7-4CBD-4A9F-955E-0EA7E141B977	E09DEB3A-A568-40A9-B274-091887301764	Sql-query	1	NULL	1
DF67AD2C-7A09-4775-867C-0EF02C3C81E5	64CA69BA-7E9A-4BA7-A0CA-0BF366E1883D	Stages-chain	Stage1	NULL	1
4F3A85F9-B02F-486A-B852-0F07D516303E	6CD6414F-44AC-4522-9703-0C02652EBFB8	Stages-chain	Stage1	NULL	1
615F2F7E-7D82-4905-9BE5-0F8E20BF0663	EF37FDEF-A44F-41B6-8287-0C410B833EBA	Stages-chain	Stage3	NULL	1
4F385AAE-BB8B-42AC-879B-101F89D4FF37	8162EB52-38B2-44EF-B442-0C13EF996A7F	Stages-chain	Stage2	NULL	1
F6F58DD6-CEF6-4173-9D91-11AF28081406	8162EB52-38B2-44EF-B442-0C13EF996A7F	Sql-query	<a SQL query that can return 'true' or 'false' only>	AND	2

 For details on how to configure these tables and explanations of each field, see section 1.2.1

 The ID column of each table is automatically generated upon insertion of a record and is used by other tables to refer to it.

 The combination of the Type being 'Sql-query' and the ConditionQuery being '1' leads to the stage in question being executed as soon as the process execution starts. This should be used for the first stage in a sequence, but as many stages as necessary may be configured to start immediately.



## 1.2 Using the Module


The module focusses on the parallelisation of Cortex flow execution and allows for a system of complex dependencies to achieve this. It offers the following functionality:

1. A LivePortal dashboard to manage process executions. The user may do the following:
  - a. Trigger any process configured in the Process\_Definitions table
  - b. View the state of all process executions, which may be one of the following:
    - i. **Running**  
Stages belonging to the process will execute based on dependencies configured in the Stage\_Start\_Conditions table.
    - ii. **Paused**  
No new stages of the process will be executed even if their start conditions are fulfilled.
  - c. Perform the following actions on a process execution:
    - i. **Pause**  
Given a process execution is in the **Running** state, it may be manually changed to the **Paused** state. Any flows currently executing will continue to do so, but no new ones will be triggered.
    - ii. **Resume**  
Given a process execution is in the **Paused** state, it may be manually changed to the **Running** state.
    - iii. **Cancel**  
Given a process execution is in the **Paused** state, the execution may be manually terminated. Any flows currently executing will continue to do so.
  - d. View the state of all stage executions under any process executions, for an exhaustive list see section 1.4.1.3.
  - e. Perform actions on a stage execution. For an exhaustive list see section 1.4.1.3.
  - f. View the Cortex-Logging logs for a currently executing process.
2. A LivePortal user interface allowing for logs of previous process executions to be extracted from the Cortex-Logging database and easily viewed.
3. A LivePortal user interface allowing for the user to make changes to existing stage definitions, where they may update the following:
  - a. Whether or not a stage is disabled
  - b. Whether or not a stage has a breakpoint
  - c. The notification email address of a stage

4. A monitoring flow to evaluate the status of the system and update the State Engine Database accordingly.

When this monitoring flow is executed, it does the following in order:

- a. Examines all stage executions with the state "Running", "Running with errors" or "Timeout".
  - i. If a stage's flow has completed successfully, and it was in the state "Running" or "Timeout" the stage execution state is updated to "Completed".
  - ii. If a stage's flow has completed successfully, but it was in the state "Running with errors", the stage execution state is updated to "Completed with errors".
  - iii. If a stage's flow has completed unsuccessfully, the stage execution state is updated to "Completed with errors".
  - iv. If a stage's flow has not completed and the timeout limit has been exceeded, the stage execution state is updated to "Timeout".
- b. Examines each process execution with the state "Running".
  - i. If each of a process execution's stages are in the state "Completed", then the process is logged as completed and removed from the Process\_Executions database.
- c. Examines each of the stage executions with the status "Waiting to Start" and has its start conditions fulfilled.
  - i. If it is not disabled or paused, and does not have a breakpoint, the relevant flow is triggered.

 This flow should be scheduled to run with a given frequency, every minute is recommended. For information on how to do this see [CTX-State-Engine - Deployment Guide](#)

5. A flow for disaster recovery that sets all stages to "Completed with Errors" and notifies a point of contact.

## 1.2.1 Configuring Processes and Stages

### 1.2.1.1 Configuring Process\_Definitions

To add a new process to the State engine, take the following steps:

1. Open SQL Server Management Studio
2. Paste the following SQL script and replace the <values>, as necessary.
3. Execute the script.

```
INSERT INTO [Cortex-State-Engine].[dbo].[Process_Definitions]
(Name, AverageTime, EndTimeLimit)
VALUES
(
    <Name>,
    <AverageTime>,
    <EndTimeLimit>
)
```

Value Name	Value Type	Value Description	Example
Name	Text	Name of the process	'Perform-Checks'
AverageTime	Integer	Average time the process should take, in seconds	30
EndTimeLimit	Integer	Maximum time the process should take, in seconds	60

### 1.2.1.2 Configuring Stage\_Definitions

To add a new stage to the State Engine, take the following steps:

1. Open SQL Server Management Studio
2. Paste the following SQL script and fill in the values, as necessary.
3. Execute the script.

```
INSERT INTO [Cortex-State-Engine].[dbo].[Stage_Definitions]
(Name, ProcessID, Flow, Disabled, Breakpoint, ReRun, Timeout,
TimeoutNotificationAddress)
VALUES
(
    <Name>,
    <ProcessId>,
    <Flow>,
    <Disabled>,
    <Breakpoint>,
    <ReRun>,
    <Timeout>,
    <TimeoutNotificationAddress>
)
```

Value Name	Value Type	Value Description	Example
Name	Text	Name of the stage	'Gather-Information'
ProcessId	Uniqueld	Id of the parent process, from the Process_Definitions table	'9E1552C6-7EEB-45F4-AA1A-01680B74A54A'
Flow	Text	Name of the flow to trigger.  If this is a manual stage, i.e. one that does not have any automation and should be manually completed once the task is done, it should take the value 'Manual'	'Get-Information-Flow'
Disabled	Boolean	1 if disabled, 0 if enabled. See Section 1.4.1.3 for more information	1
Breakpoint	Boolean	1 to add a breakpoint, 0 if no breakpoint. See Section 1.4.1.3 for more information	1
ReRun	Boolean	1 if able to re-run, 0 if not able to re-run. See Section 1.4.1.3 for this and other manual interactions that are possible.	0
Timeout	Integer	Timeout period in seconds	45
Timeout Notification Address	Text	Email address to send exceptions to	'notifications@cortex.uk'

When a stage definition is added, it is given a timestamp by the database. It is based on this that the order in which the stages are displayed in the Manage Executions UI is decided. It is advised to have a short delay between each stage insertion as a result.

### 1.2.1.3 Configuring Stage\_Start\_Condtions

To add a new stage start condition to the State Engine, take the following steps:

1. Open SQL Server Management Studio
2. Paste the following SQL script and fill in the values, as necessary.
3. Execute the script.


```
INSERT INTO [Cortex-State-Engine].[dbo].[Stage_Start_Conditions]
(StageID, Type, ConditionQuery, MultipleConditionOperator, [Order])
VALUES
(
    <StageID>,
    <Type>,
    <ConditionQuery>,
    <MultipleConditionOperator>,
    <Order>
)
```

Value Name	Value Type	Value Description	Example
StageID	Uniqueld	Id of the stage to apply start conditions to	'CF20602E-43D8-4F74-B13C-0CC268818244'
Type	Text	Type of condition, either 'SQL-Query' or 'Stages-Chain'	'Stages-Chain'
ConditionQuery	Text	<p>If Type is 'SQL-Query', then an SQL query that returns TRUE or FALSE only.</p> <p>If Type is 'Stages-Chain', then the name of a stage that will have to finish executing before this one starts</p>	<p>'SELECT case when SE.Status='RUNNING' then 'TRUE' else 'FALSE' end</p> <p>FROM Stage_Executions SE</p> <p>WHERE SE.ID = '2D2CD273-1006-45C3-9ADC-051FAFE6BB2D'</p>
Multiple Condition Operator	Text	<p>If a stage's execution already depends on multiple other stages' completion, should this dependency be satisfied as well as, or instead of, these.</p> <p>Either 'AND' or 'OR' (or 'NULL' for the first condition)</p>	'AND'
Order	Integer	If a stage has multiple condition records, order specifies the sequence in which the Multiple condition operator is evaluated	3

## Example

A stage has the following start conditions table (ignoring the columns inserted by the system):

StageID	Type	ConditionQuery	Multiple Condition Operator	Order
F3711AEC-9D3A-4BAB-B53F-24E723F44E20	'SQL-Query'	1	'NULL'	1
F3711AEC-9D3A-4BAB-B53F-24E723F44E20	'SQL-Query'	SELECT 'FALSE'	'OR'	3
F3711AEC-9D3A-4BAB-B53F-24E723F44E20	'Stages-Chain'	'Stage-1'	'AND'	2
F3711AEC-9D3A-4BAB-B53F-24E723F44E20	'Stages-Chain'	'Stage-2'	'AND'	4

 To ensure a stage is started immediately, i.e. on the first execution of the monitoring flow after the process execution is started, it should have a single record in the Stage\_Start\_Conditions table with a Type of "sql-query" and a StartCondition of "1".

When evaluated, this reads as follows:

*(1 AND Stage-1 is completed) OR ((SELECT 'FALSE') AND Stage-2 is completed)*

AND and OR are logical operators, whereas TRUE and FALSE are the values that logical variables may take. They have the following relationship:

X	Y	X AND Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

X	Y	X OR Y
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

In our case, X and Y would take the value of TRUE on completion of another stage or take the value of the result of a SQL query.

So, in the above example, if Stage-1 is completed and Stage-2 is not completed, then our expression would read as follows:

*(TRUE AND TRUE) OR (FALSE AND FALSE)*

= *TRUE OR FALSE*

= *TRUE*

Therefore, the stage would be triggered on the next execution of the monitoring flow.

Note the following:

1. 1 and SELECT 'TRUE' both evaluate to TRUE, similarly 0 and SELECT 'FALSE' both evaluate to FALSE. Use 1 as the only ConditionQuery to force a stage execution to run immediately.
2. The order of evaluation is done based on the Order column, not the position in the table.
3. For each condition query, the multiple condition operator is placed before it when considering how it is evaluated, hence why 'NULL' is required for the first one.
4. There are implicit parentheses around any statements separated by an 'AND' operator. This is a law of logic implemented by SQL Server, similar to algebraic laws around addition and multiplication, but with OR and AND respectively.

E.g.  $a \times b + c \times d$  is equal to  $(a \times b) + (c \times d)$  rather than  $(a \times b + c) \times d$  or any other permutation.

Similarly,  $W \text{ AND } X \text{ OR } Y \text{ AND } Z$  is equal to  $(W \text{ AND } X) \text{ OR } (Y \text{ AND } Z)$ , where W, X, Y and Z can take values of TRUE or FALSE.

From this we can see how important the MultipleConditionOperator and Order columns are, by simply changing the order above from 1, 3, 2, 4 to 1, 2, 3, 4 the evaluation of the start conditions would read as follows:

*TRUE OR (FALSE AND Stage-1 is completed AND Stage-2 is completed)*



## 1.2.2 Authoring flows for the State Engine

While the state engine has been designed and built to be as independent as possible of the flows that it triggers, when authoring these flows, there are two things to bear in mind to fully utilise them.

### 1.2.2.1 Input Variables

Each flow to be called by the state engine must contain the following variables in the global variable store. These are passed to the flow being triggered by the state engine monitoring flow, so the user does not need to assign any initial values.

Variable Name	Description	Example
g_log-handler	The log handler for this stage execution. Any events that the stage's flow is logging should be logged to this if they are to appear in the corresponding stage in the Logging Database.	<pre>{   "IDS": {     "STAGE":     "30f7e532d4744498bb0b7647467391b1",     "EVENT": null,     "PROCESS":     "a1716c3e638646d88be78789cb01987f"   },   "LOGS": [] }</pre>
g_Process-Name	The name of the process as part of which this stage is being executed.	Process1
g_Stage-Name	The name of the stage under which this flow is being executed.	Stage3

### 1.2.2.2 State Engine Interaction Subtasks

There are two subtasks as part of the Cortex State Engine module that may be implemented as part of a solution flow to be triggered by the state engine.

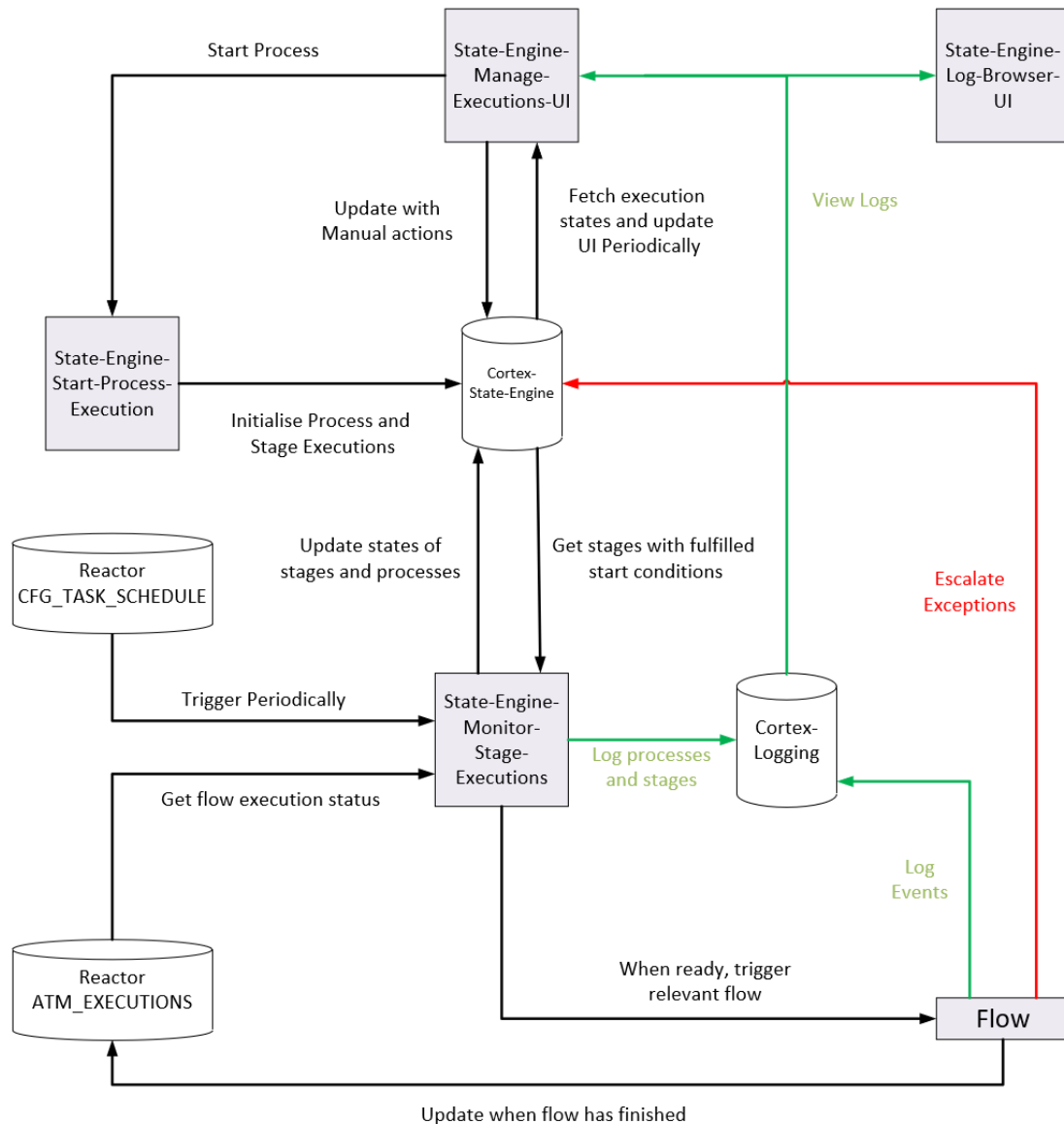
1. State-Engine-Escalate-Stage

Used whenever a flow should report an exception to the state engine, it changes the stage execution to either the "Completed with errors" or "Running with errors" state, depending on the subtask input. See Section 2.4 for a full overview.

2. State-Engine-Stage-Completed

Used whenever a flow has completed its automation and wishes to immediately update the state engine with this fact, rather than waiting for the scheduled monitoring flow to execute. See Section 2.5 for a full overview.

### 1.3 Flow/Database Architecture



The above diagram shows the interaction between the various flows and databases, including a flow named “Flow” that is associated with a stage configured in the Cortex-State-Engine database.

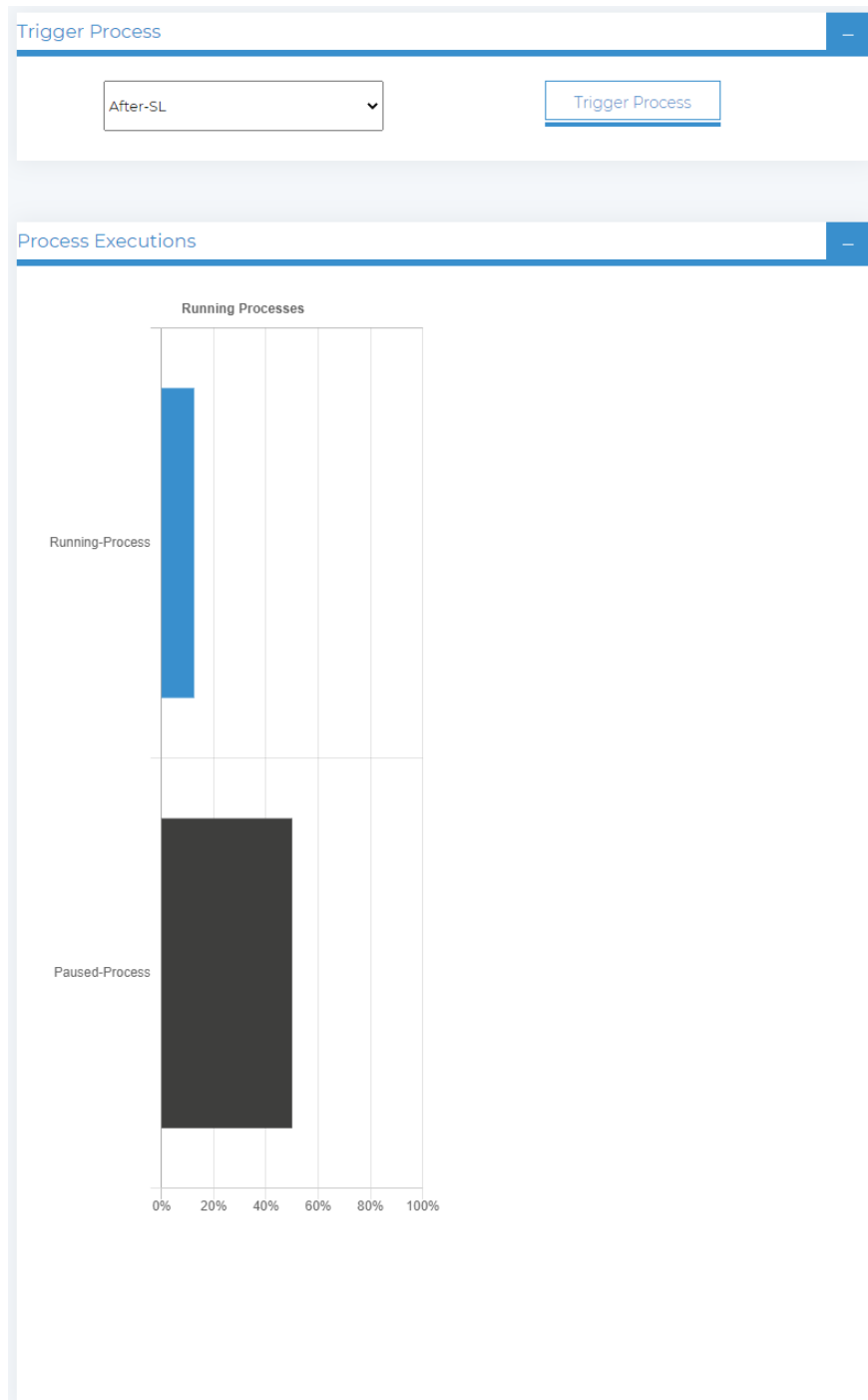
Labelled in **red** are interactions to do with raising exceptions from the stage’s flow, and in **green** are interactions involving logging data.

All State-Engine flows in this diagram also interact with the Cortex-ConfigStore database to fetch relevant configuration data.

## 1.4 User Experience

### 1.4.1 State Engine UI

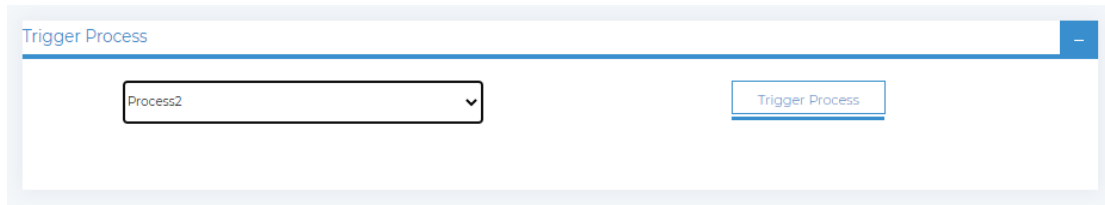
Launch the flow State-Engine-Manage-Executions-UI from Cortex LivePortal. The user will be presented with a screen like the below:



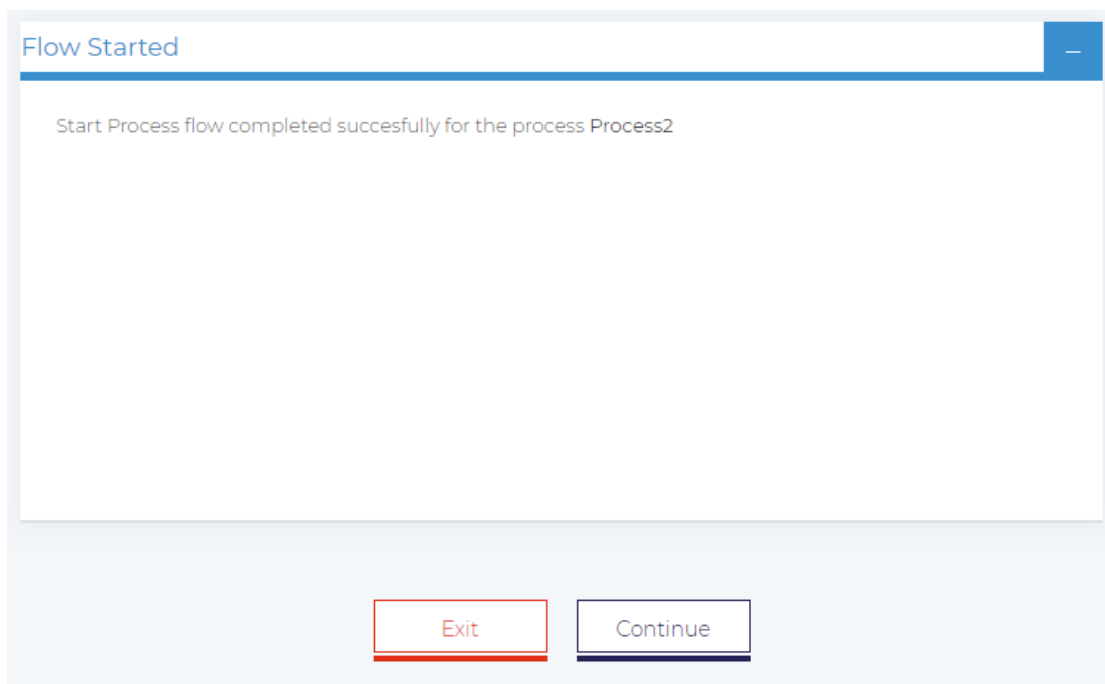
In this case there are two processes, the first currently running and the second paused.

#### 1.4.1.1 Triggering a process execution

1. In the top panel of the UI, under “Trigger Process”, there is a dropdown box where the user can select one of the processes configured in the state engine database:

A screenshot of the "Trigger Process" UI. It features a title bar with the text "Trigger Process" and a close button. Below the title bar is a dropdown menu currently showing "Process2". To the right of the dropdown is a button labeled "Trigger Process".

2. Once the correct process is selected, they can select “Trigger Process” to start the execution.
3. Once the execution is started, they will be presented with a confirmation screen:

A screenshot of the "Flow Started" confirmation UI. It has a title bar with the text "Flow Started" and a close button. The main area contains the message "Start Process flow completed successfully for the process Process2". At the bottom, there are two buttons: "Exit" (highlighted with a red border) and "Continue" (highlighted with a blue border).

4. Selecting “Continue” will take the user back to the main dashboard, where the newly triggered execution will be available to view.
5. Selecting “Exit” will end the State-Engine-Manage-Executions-UI flow.

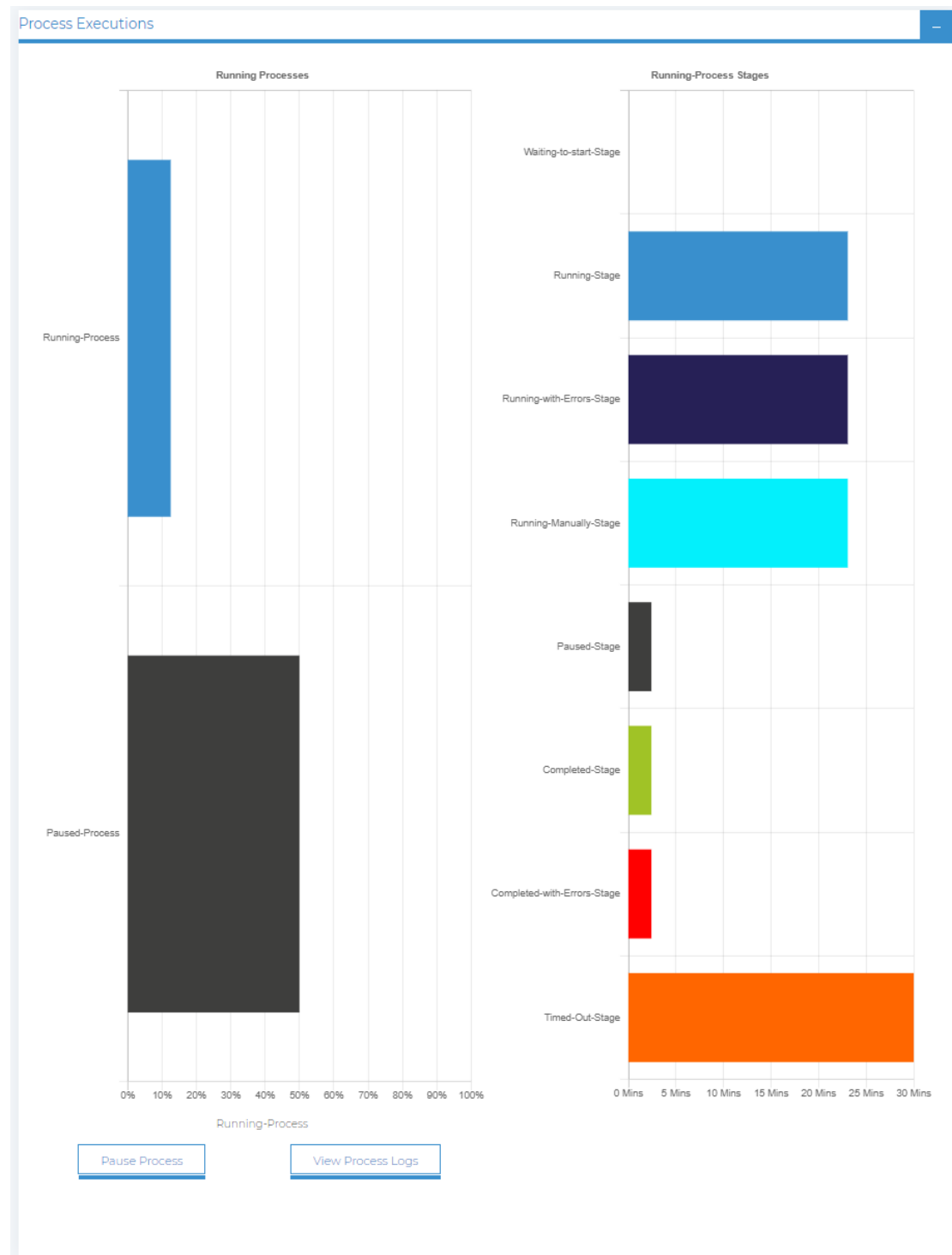
When a process execution is started from the UI, the following will occur:

1. A new process execution is set to “Running” and all its child stage executions are created and set to “Waiting-to-start”.
2. On the next execution of the scheduled monitoring flow, any of these stage executions with their start conditions fulfilled are set to “Running” and their associated flow is triggered.
3. On the next execution of the scheduled monitoring flow, the stages that were set to running are updated if their flows have completed, timed out, or escalated an exception. Then, any stage executions still waiting that now have their start conditions fulfilled are set to “Running” and their associated flow is triggered.
4. This continues until an execution of the scheduled monitoring flow confirms that all of the stage executions for the process have a status of “completed”. At this point the process execution is considered completed and removed from the State Engine database, along with the stage executions.

For more information on the monitoring flow, see Section 3.4.

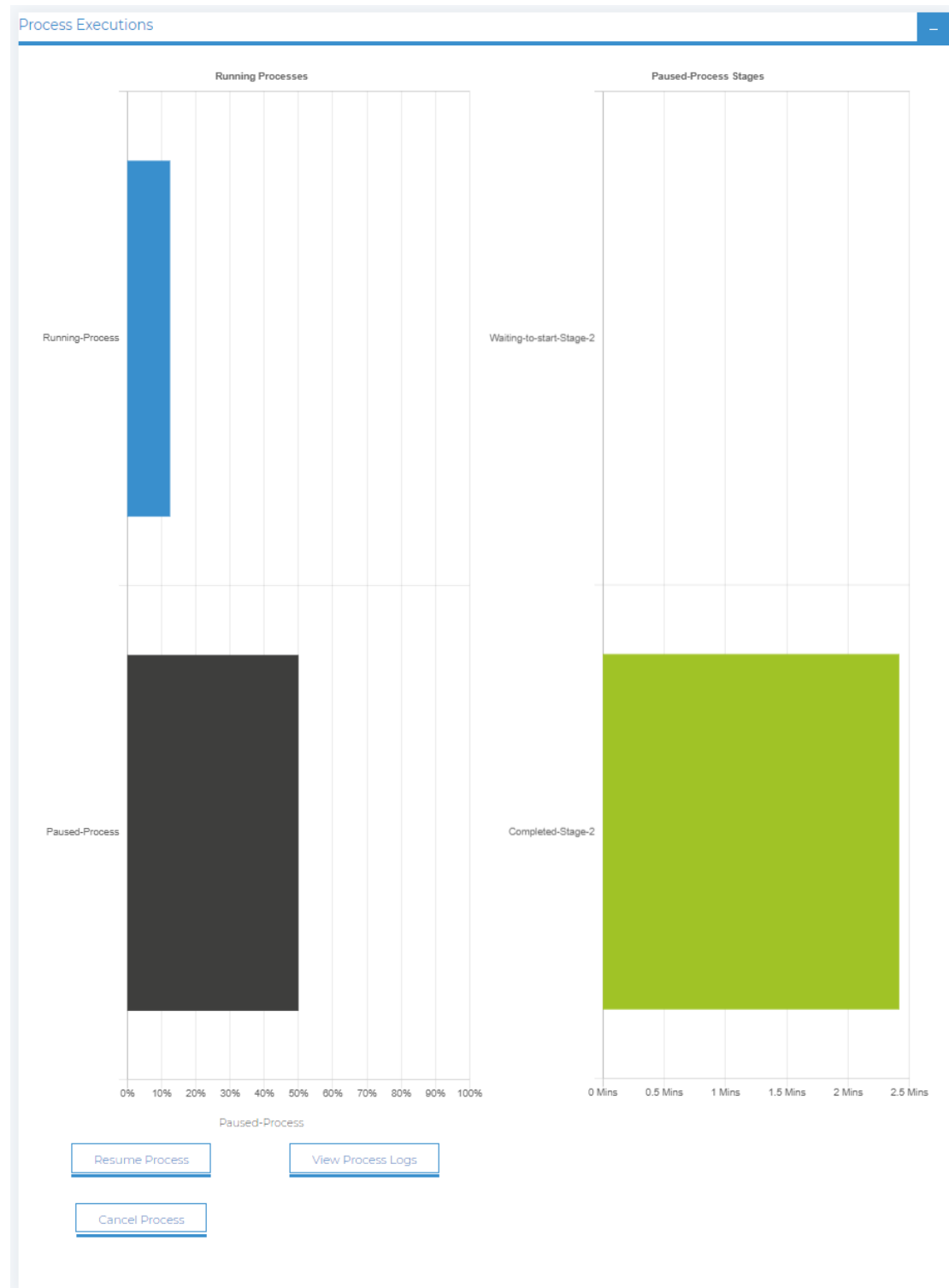
### 1.4.1.2 Interacting with a process execution

Selecting the first process, by clicking on its label or its coloured bar, will display to the user the stages of that processes, and reveal the controls for it, like the below:



The name of the selected process is displayed below the processes chart, along with the controls that are available for a process in that state. In the case of a running process like here, we may pause it or view its logs. The stage executions are colour coded based on the state they are in, with the possible colours being listed in full in section 1.4.1.3.

If we select the paused process, we can see the below:



Now there is the option to either resume or to cancel it the process, and we are presented with the two stages that are part of this process, one waiting to start and one complete.

Cancelling will end the process execution and all associated stage executions.

Selecting “View Process Logs” will present the user with a list of the events of each stage in the process as logged in the logging database, an example of which is shown below:

Process Logs

STAGENAME	EVENTNAME	EVENTTIME	STAGESTARTTIME	STAGEENDTIME
Filter Stagename	Filter Eventname	Filter Eventtime	Filter Stagestartti...	Filter Stageendtime
Dependent-on-Stage-21-and-22			2021-10-26 05:24:24	
Stage-21			2021-10-26 05:24:02	2021-10-26 05:24:20
Stage-22			2021-10-26 05:23:58	2021-10-26 05:24:18
Stage-01			2021-10-26 05:23:08	2021-10-26 05:23:53
PROCESS	Process started by \ravichandran.v	2021-10-26 05:23:01	2021-10-26 05:23:01	2021-10-26 05:23:01
<div><div><div>&lt;&lt;</div><div>&lt;</div><div>1</div><div>&gt;</div><div>&gt;&gt;</div></div><div>Page size: 10</div><div>5 items in 1 pages</div></div>				

Refresh

Back

View

Selecting an event and then “View” will show the user any parameters that were logged against it:

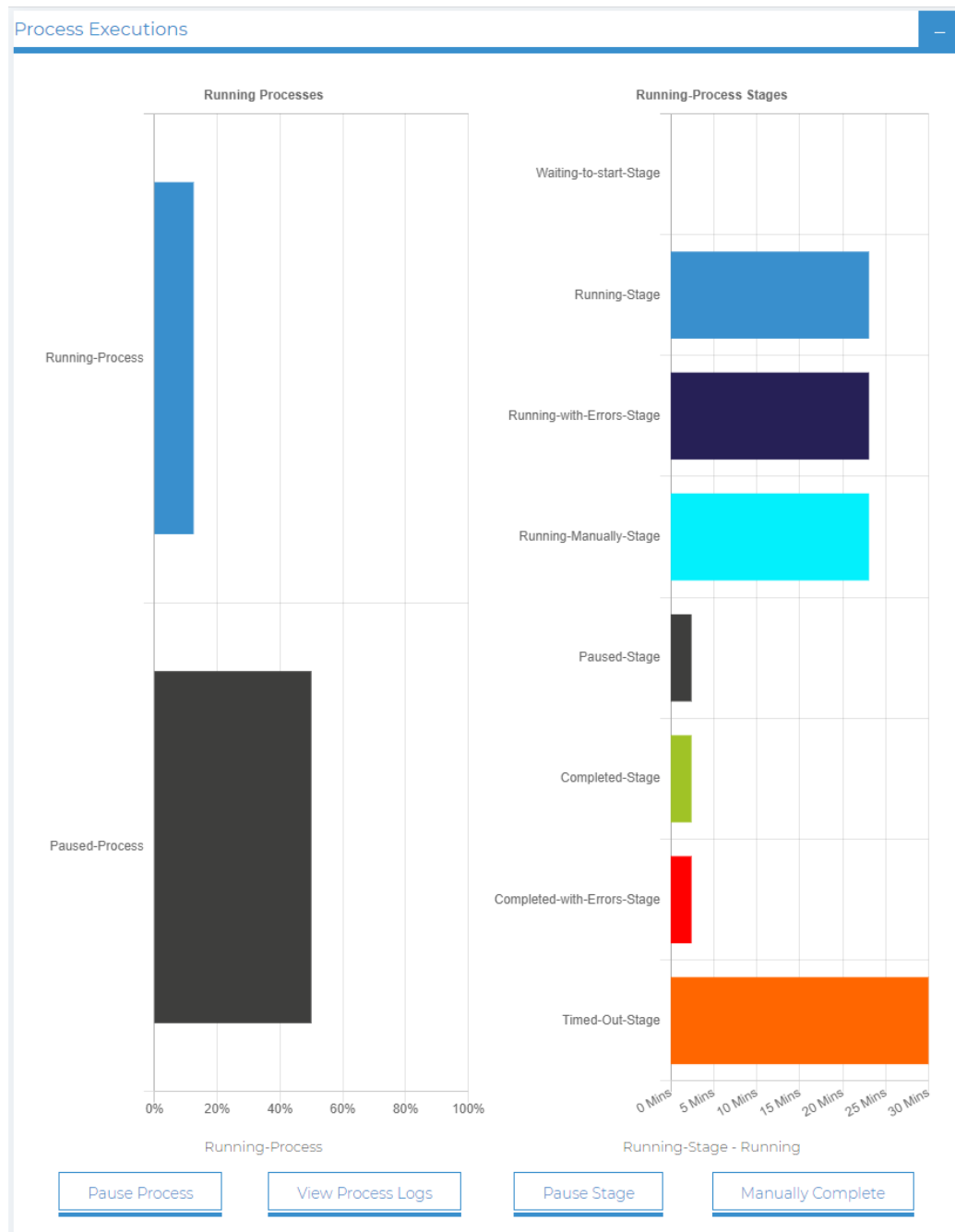
Event Parameters		
EXCEPTION	CODE	J0003
	TYPE	EXCEPTION
	SYSTEM	Cortex Database
	TITLE	Exception submitting the procedure
	SEVERITY	Critical
	DETAILS	Exception Submitting query - Could not execute query "SELECT * FROM STAGE_EXECUTIONS"

For information on how to log information to the logging database, see CTX-Logging - User Guide.










### 1.4.1.3 Interacting with a stage execution

Once a process is selected, in this case the running process, the user can select one of the stages displayed in the same way, in this example the running stage. They will be presented with a view like the below:



The name of the selected stage is displayed under the stages chart, along with the state it is in at the present moment and the manual actions that are available for a stage in that state.

For a full explanation of each state and its colour on the user interface, use the following table:

State	Colour	Description	Available Actions
Waiting to start	None	The initial state of any stage that has not had its start conditions fulfilled yet.	Enable (if disabled) Disable (if enabled) Breakpoint (if no breakpoint) Remove Breakpoint (if breakpoint)
Running		A stage has had its start conditions met, and its flow execution has been triggered and is being monitored.	Pause Manually Complete
Running Manually		A stage is running, but it has no flow configured. This is to represent a manual task. Once the user has completed this task, they may manually complete the stage.	Pause Manually Complete
Running with Errors		A stage flow has encountered an error but continued its execution.	Pause Manually Complete
Paused		A stage's flow is executing but no stages that depend on its will be triggered upon its completion.	Resume Manually Complete
Completed		A stage's flow has completed successfully, fulfilling any start conditions associated with other stages that refer to it.	None
Completed with Errors		A stage's flow has completed but encountered errors, no stages that depend on its completion will be triggered.	Re-Run Manually Complete
Timeout		A stage's flow is still executing despite the timeout period configured in the Stage_Definitions table having elapsed.	Manually Complete

For a full explanation of each action, use the following table:

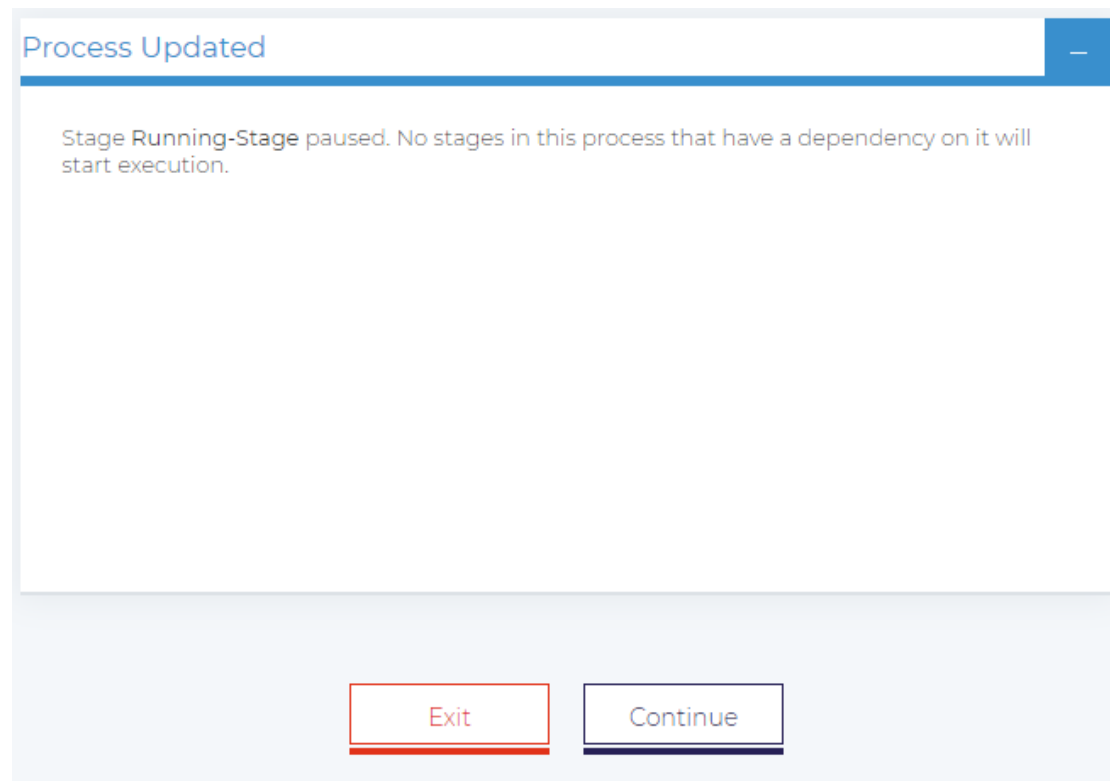
Action	Required state to be available	Description
Pause	Running Running manually Running with Errors	Stage is manually changed to the "Paused" state. Any flows currently executing will continue to do so, but no new ones will be triggered.
Resume	Paused	Stage is manually changed to the "Running" state.
Manually Complete	Running Running manually Running with Errors Completed with Errors Paused Timeout	Stage is manually changed to the "Completed" state.
Re-Run	Completed with errors	Stage is manually changed to the "Waiting-to-start" state.
Disable	Waiting to start (if stage is enabled)	Stage is disabled, i.e. it will be skipped, meaning that if it constitutes another stage's start condition then that start condition will be ignored.
Enable	Waiting to start (if stage is disabled)	Stage is enabled, i.e. it will not be skipped, meaning that if it constitutes another stage's start condition then that start condition will be not be ignored.
Add Breakpoint	Waiting to start (if stage does not have a breakpoint)	Process execution will pause before executing this stage, and the stage's notification email address will be sent a notification.
Remove Breakpoint	Waiting to start (if stage does have a breakpoint)	Process execution will not pause before executing this stage, it will continue as normal.

There are two other parameters associated with a stage, configured in the Stage\_Definitions table as in section 1.2.1.2

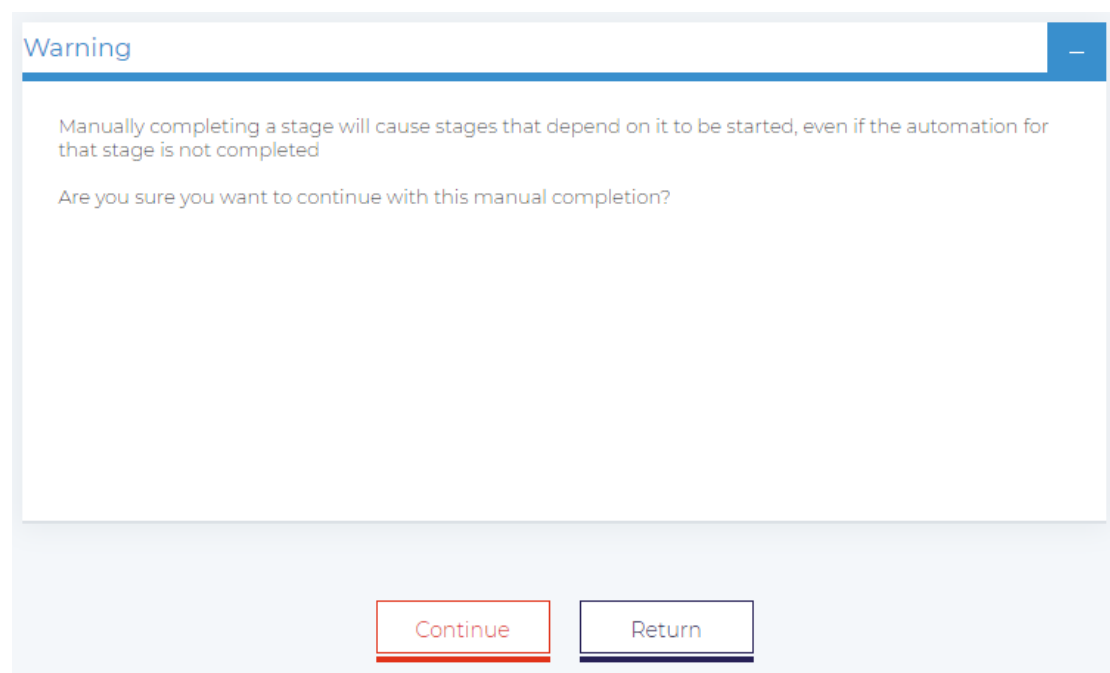
Parameter	Explanation
Disabled	If a stage is disabled, it will not be executed when its start conditions are met. However, any subsequent stages that are configured to depend on it will be treated as if they do not. The effect of this is to skip the stage and proceed as normal.
Breakpoint	If a stage has a breakpoint, it will not be executed when its start conditions are met. The process will pause at this point and an email will be sent to the timeoutNotificationAddress configured for the stage informing them of this.

Once a stage is configured initially, these parameters may be updated for all future executions using the update definitions UI, or for a specific execution using the manage executions UI.

After each action, the user is presented with a confirmation screen outlining what action was taken, for example the below, where a stage called “Running-Stage”, in the “Running” state, was paused.

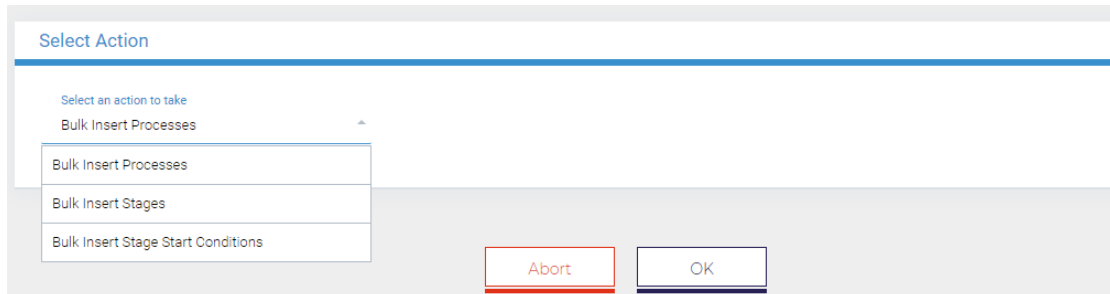


The only exception to this is when manually completing a stage, where due to the possible disruption it may cause the user is presented with a warning first.



## 1.4.2 Configuration Definition UI

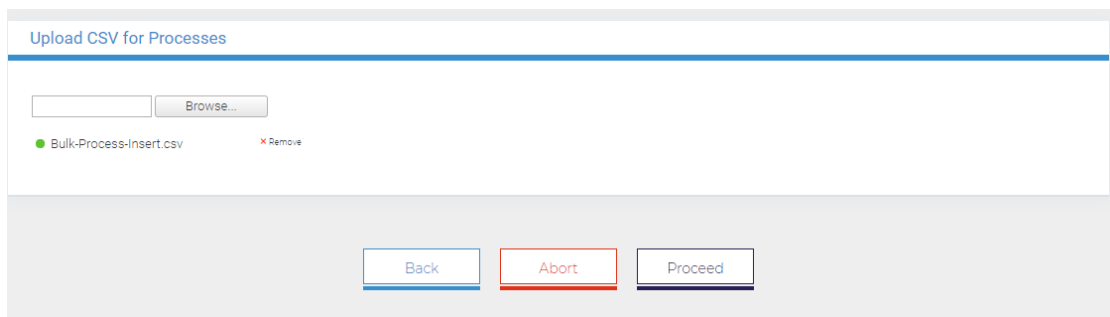
Launch the flow State-Engine-Configure-Definition-UI from Cortex LivePortal. The user will be presented with a screen listing the options for bulk insert into : (1) Process (2) Stage and (3) Stage Start Conditions definition



The 'Select Action' dialog box has a title bar 'Select Action'. Below it, a text prompt says 'Select an action to take'. A dropdown menu is open, showing three options: 'Bulk Insert Processes', 'Bulk Insert Stages', and 'Bulk Insert Stage Start Conditions'. At the bottom right of the dialog are two buttons: 'Abort' (outlined in red) and 'OK' (outlined in blue).

Select one of the options and click ok to upload a csv file. Refer '4. State-Engine-Configuration-Bulk-Insert' templates folder for the templates of the csv in case the definition is done from scratch. To import the definition from another server, refer to 'Cortex-State-Engine-Export-Instructions.sql' to export the content from other server and use the UI for importing.

Use browse option to select the csv file and click proceed button:











The 'Upload CSV for Processes' dialog box has a title bar 'Upload CSV for Processes'. It features a text input field and a 'Browse...' button. Below this, a file named 'Bulk-Process-Insert.csv' is listed with a green dot icon and a red 'X Remove' link. At the bottom are three buttons: 'Back' (outlined in blue), 'Abort' (outlined in red), and 'Proceed' (outlined in blue).

In case of Stage Start Conditions, the existing start conditions for the process will be deleted by the system and then the uploaded start conditions will be inserted.

A screen will show the extracted data from the csv file that can be validated / edited before proceeding with Insert action. Click Proceed button to insert the data.

View / Edit Uploaded Data for Processes









PROCESSNAME	AVERAGETIME	ENDTIMELIMIT	
Filter Processname	Filter Averagetime	Filter Endtimelimit	
Process-with-single-stage	300	300	 
Process-with-sequence-of-stages	300	300	 
Process-with-Parallel-stages	300	300	 
Process-with-mix-of-stages	300	300	 

<< < 1 > >> Page size: 10 4 items in 1 pages

Back Abort Proceed

In case of issues with the data, the user will be presented with issues as below. Correct them as required and continue with the action:

View / Edit Uploaded Data for Processes

PROCESSNAME	AVERAGETIME	ENDTIMELIMIT	
Filter Processname	Filter Averagetime	Filter Endtimelimit	
Process-with-single-stage	300	300	 
Process-with-sequence-of-stages	300	300	 
Process-with-Parallel-stages	300	300	 
Process-with-mix-of-stages	300	300	 

<< < 1 > >> Page size: 10 4 items in 1 pages

Total Number of Records in the table : 4  
 Total Number of Records with Issues : 4  
 The Process "Process-with-single-stage" is already configured. Please remove it from the table.  
 The Process "Process-with-sequence-of-stages" is already configured. Please remove it from the table.  
 The Process "Process-with-Parallel-stages" is already configured. Please remove it from the table.  
 The Process "Process-with-mix-of-stages" is already configured. Please remove it from the table.

Back Abort Proceed

Finally, you will be shown a confirmation screen for the Db updates.

### 1.4.3 Definition Update UI

Launch the flow State-Engine-Definition-Update-UI from Cortex LivePortal. The user will be presented with a screen listing all the configured processes, like the below:

**Process Selection**

Please select one process in the table below and click Next

You can filter the results in the table by entering filter criteria in the field at the top of each column.  
 NOTE: Processes that are not running currently are shown below. Running processes can be managed using State-Engine-Manage-Executions-UI

PROCESSNAME
Filter Processname
Disable-Enable-Stage-Demo
Pause-Resume-Stage-Demo
Process-with-mix-of-stages
Process-with-Parallel-stages
Process-with-sequence-of-stages
Process-with-single-stage
Test

Page size: 10 9 items in 1 pages

End Next

If the user selects a process, then “next”, they are presented with a screen where they may configure certain details about the stages assigned to the process.

**Stage Definition Update**

### Modify Disable-Enable-Stage-Demo Stage Definitions

You have selected to modify the Stage Definitions for the process Disable-Enable-Stage-Demo.

The existing data from the DB is shown in the grid below.

Please edit the required items in the grid:

STAGENAME	FLOWNAME	DISABLED	BREAKPOINT	TIMEOUT	NOTIFICATIONADDRESS
Filter Stagename	Filter Flowname	<input type="checkbox"/>	<input type="checkbox"/>	Filter Timeout	Filter Notificationaddress
Stage-01	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com
Stage-21	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com
Stage-22	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com
Dependent-on-Stage-21	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com
Dependent-on-Stage-22	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com
Dependent-on-Stage-21-and-22	SE-Stage-Flow	<input type="checkbox"/>	<input type="checkbox"/>	180	user1@domain.com,user2@domain.com

Page size: 10 6 items in 1 pages

<< Back Homepage Confirm >>

Select “Confirm” to save changes.



### 1.4.4 Log Browser UI

The State Engine UI allows users to view logs of currently executing processes, but once the process has been completed it is no longer available to view there. Hence, the Log Browser UI should be used to view the logs of previously completed process executions.

Launch the flow State-Engine-Log-Browser-UI from Cortex LivePortal. The user will be presented with a list of previously completed processes, like the below:

Process Logs

PROCESS	START	END
Filter Process	Filter Start	Filter End
Process-with-mix-of-stages	2021-08-31 17:23:38	
Process-with-mix-of-stages	2021-08-31 18:17:48	2021-08-31 21:16:06
Process-with-mix-of-stages	2021-09-01 06:02:54	2021-09-01 07:18:11
State-Engine-Definition-Update-UI	2021-09-02 06:46:40	2021-09-02 06:46:40
Process-with-mix-of-stages	2021-09-07 07:56:09	
Process-with-Parallel-stages	2021-09-07 08:00:47	2021-09-07 08:02:45
Process-with-mix-of-stages	2021-09-07 12:06:03	
Process-with-mix-of-stages	2021-09-07 12:13:56	
Process-with-Parallel-stages	2021-09-07 12:16:20	2021-09-07 12:20:14
Process-with-mix-of-stages	2021-09-08 02:20:33	2021-09-08 02:45:17

« 1 2 3 4 5 6 7 8 » Page size: 10 78 items in 8 pages

Back View

Selecting a process and then “View” will present the user with a list of events associated with each stage of the process. This is identical in functionality to selecting “View Process Logs” on the State Engine UI, the user may then drill down further to view the parameters under each event.

## 2 State Engine Subtasks

---

### 2.1 EDBQ-State-Engine-Execute-DB-Query

#### 2.1.1 Overview

This subtask is used by the state engine flows to interact with a database, usually Cortex-State-Engine but it can be used generically.

#### 2.1.2 Inputs

Name	Type	Description	Example
EDBQ_i_DB-Connection-String	Text	The connection string of the database to connect to	Initial Catalog = DatabaseName; Data Source=Server; pooling=false; Integrated Security=SSPI
EDBQ_i_Query	Text	The query to execute	SELECT Name FROM Process_Definitions
EDBQ_i_List	Text	“true” if the result should be returned as a list variable, “false” if it should be returned as a table variable.  “true” by default	true

#### 2.1.3 Outputs

Name	Type	Description
EDBQ_o_Query-Result	List	If EDBQ_i_List was given the value “true” or not provided, then the result of the database query in the form of a list of structures, each containing a record of the result.
EDBQ_o_Query-Result-Table	Table	If EDBQ_i_List was given the value “false”, then the result of the database query in the form of a table variable

## 2.2 ES-State-Engine-End-Stage

### 2.2.1 Overview

This subtask is used by the state engine monitoring flow to update a stage in the state engine database to one of the completed, completed-with-errors or timeout states, and log this in the logging database.

### 2.2.2 Inputs

Name	Type	Description	Example
ES_i_Engine-Connection-String	Text	The connection string for the state engine database	Initial Catalog = Cortex-State-Engine; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_Logging-Connection-String	Text	The connection string for the logging database	Initial Catalog = Cortex-Logging; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_ExecutionID	Text	The ID of the stage execution to end. This is not necessary if ES_i_Flow-ExecutionID is provided	F7FB1151-A991-11EB-8B7F-00155DAA4D12
ES_i_End-OK	Text	"True" to set the stage to completed. "False" to set the stage to completed-with-errors. "Timeout" to set the stage to timeout.	Timeout

ES_i_Log-Handler	Structure	Log handler of the stage to end	{ "IDS": { "STAGE": "30f7e532d4744498bb0b7647467391b1", "EVENT": null, "PROCESS": "a1716c3e638646d88be78789cb01987f" }, "LOGS": [] }
ES_i_Flow-ExecutionID	Text	Execution ID of the flow execution associated with the stage execution to end. This is not necessary if ES_i_ExecutionID is provided	092CAF55-E06D-4DFD-B548-380B07B3F91D

### 2.2.3 Outputs

Name	Type	Description
ES_o_Log-Handler	Structure	The updated log handler now the stage has been ended.

## 2.3 State-Engine-Params-to-HTML

### 2.3.1 Overview

This subtask takes a set of parameters for an event in the logging database and converts them to an HTML page for viewing in LivePortal.

Parameter values that are lists or structures are also supported, with nested HTML being generated recursively.

### 2.3.2 Inputs

Name	Type	Description	Example
PTH_i_Query-Result-To-Convert	Text	A set parameters for an event in the logging database, in the following form:  <parameter name>: <parameter value>; <parameter name>: <parameter value>; <parameter name>: <parameter value>; ...	CODE: E4009;  TYPE: EXCEPTION;  EXCEPTIONLIST: ["E2342", "E23423", "E645645"];  EXCEPTIONSTRUCTURE: {"EXCEPTION1": "E2342", "EXCEPTION2": "E23423"};

### 2.3.3 Outputs

Name	Type	Description
PTH_o_HTML-Output	Text	The HTML code containing the formatted parameter data, for displaying in LivePortal.

## 2.4 ES-State-Engine-Escalate-Stage

### 2.4.1 Overview

This subtask should be inserted into a flow that is associated with a stage and is therefore called by the state engine. It either ends the stage, setting it to Completed-with-errors, or continues with the execution, setting it to Running-with-errors. The logging of this is also handled.

### 2.4.2 Inputs

Name	Type	Description	Example
ES_i_Engine-Connection-String	Text	The connection string for the state engine database	Initial Catalog = Cortex-State-Engine; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_Logging-Connection-String	Text	The connection string for the logging database	Initial Catalog = Cortex-Logging; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_Log-Handler	Structure	Log handler of the stage to end	<pre>{   "IDS": {     "STAGE": "30f7e532d4744498bb0b7647467391b1",     "EVENT": null,     "PROCESS": "a1716c3e638646d88be78789cb01987f"   },   "LOGS": [] }</pre>
ES_i_is-end-stage	Boolean	TRUE to end the stage, setting it to Completed-With-Errors  FALSE to continue, setting it to Running-With-Errors	TRUE

## 2.5 ES-State-Engine-Stage-Completed

### 2.5.1 Overview

This subtask should be inserted into a flow that is associated with a stage and is therefore called by the state engine. It records the successful ending of the stage in the state engine database and immediately triggers the monitoring flow.

This is only necessary if the stage with which the flow is associated takes a short amount of time compared to the frequency of the monitoring flow's scheduled execution, such that time lost waiting for this scheduled execution adds up to an unacceptable level over many stage executions.

### 2.5.2 Inputs

Name	Type	Description	Example
ES_i_Engine-Connection-String	Text	The connection string for the state engine database	Initial Catalog = Cortex-State-Engine; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_Logging-Connection-String	Text	The connection string for the logging database	Initial Catalog = Cortex-Logging; Data Source=Server; pooling=false; Integrated Security=SSPI
ES_i_Log-Handler	Structure	Log handler of the stage to end	<pre>{   "IDS": {     "STAGE": "30f7e532d4744498bb0b7647467391b1",     "EVENT": null,     "PROCESS": "a1716c3e638646d88be78789cb01987f"   },   "LOGS": [] }</pre>

## 3 State Engine Flows

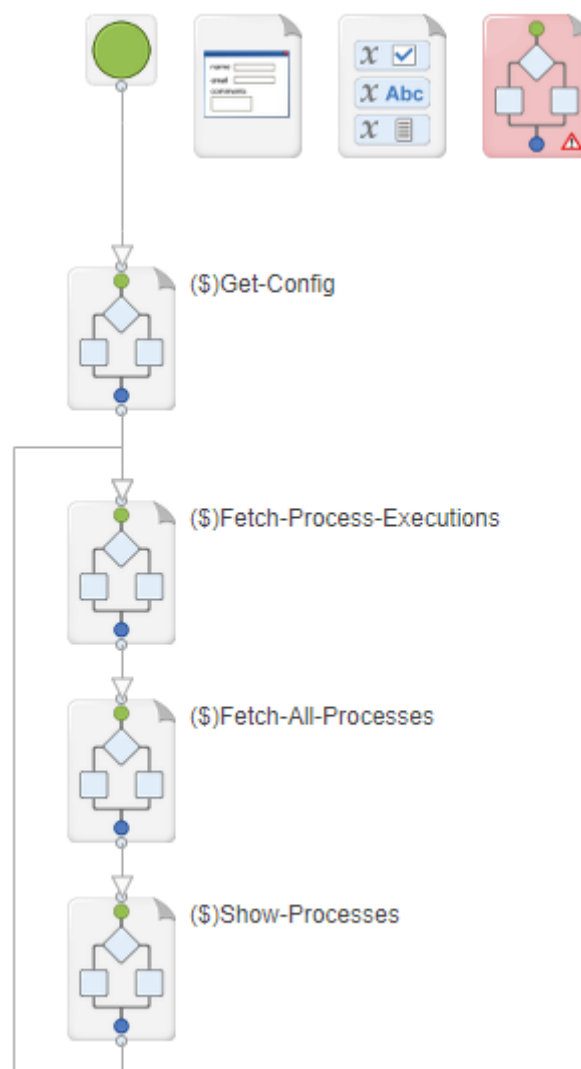
### 3.1 State-Engine Manage-Executions-UI

#### 3.1.1 Overview

This flow allows the user to trigger process executions then monitor them through a dashboard in LivePortal.

The user can also manually interact with process and stage executions by pausing, resuming, cancelling, etc.

#### 3.1.2 States





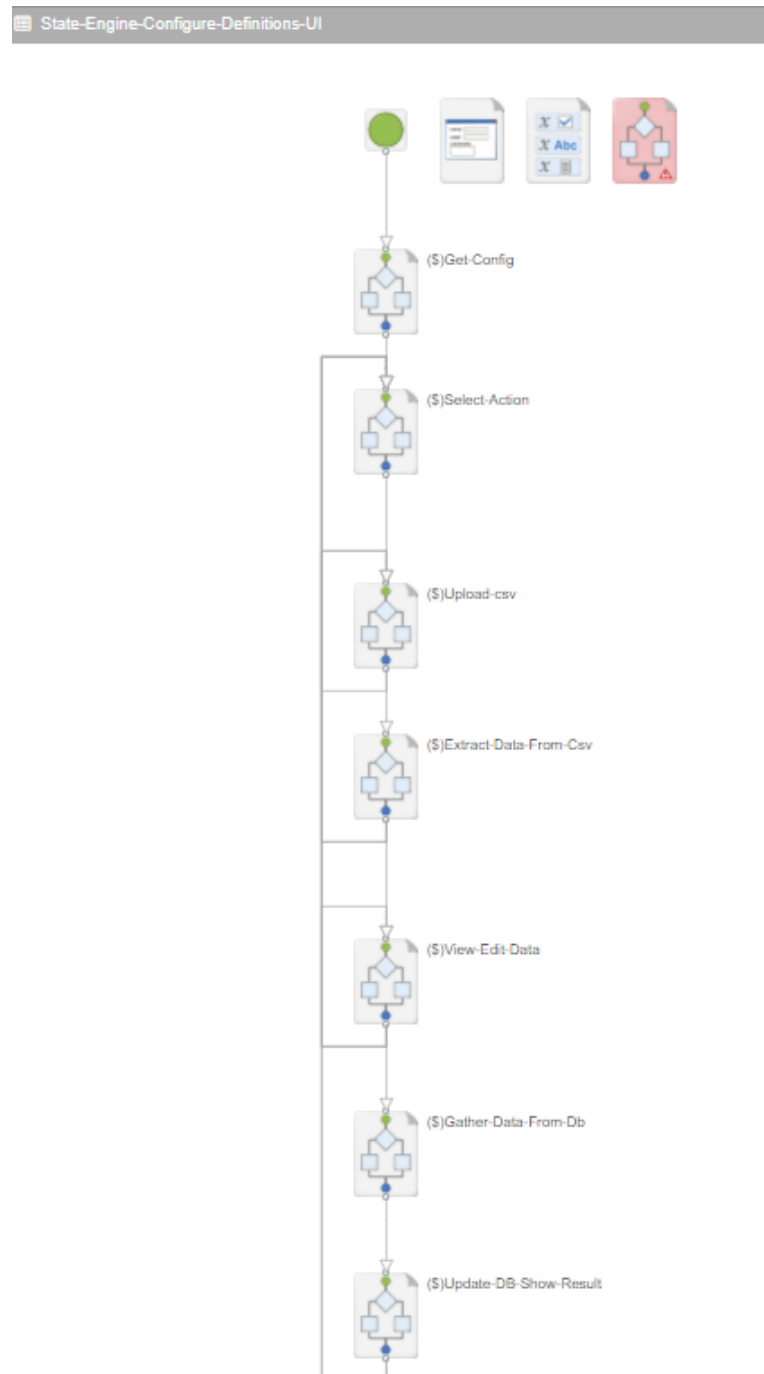
6. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
7. Fetch-Process-Executions
  - a. Fetches details of running process executions and associated stage executions from the state engine database.
  - b. Reformats into a form that the dashboard JavaScript can interpret and render.
8. Fetch-All-Processes
  - a. Fetches the names of all available processes to populate the dropdown box where the user can select a process to execute.
9. Show-Processes
  - a. Displays the UI.
  - b. Contains all the logic for the user to interact with the process and stage executions, for an exhaustive list of these interactions see Section 1.4.1.3.
  - c. Allows the user to view the contents of the logging database for a currently executing process.

## 3.2 State-Engine-Configure-Definition-UI

### 3.2.1 Overview

This flow allows the user to setup solution specific state-engine definitions. (Processes, its Stages and Stage start conditions).

### 3.2.2 States



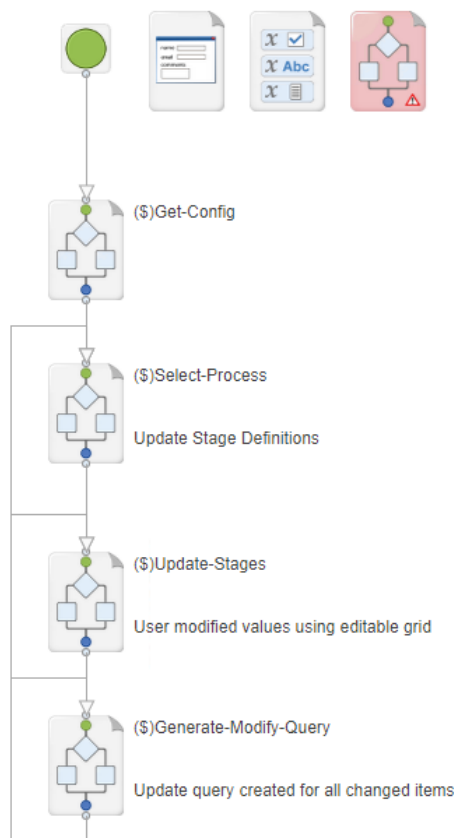
1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
2. Select-Action
  - a. Displays the bulk insert action for user selection. (Process, Stage or Stage Start Conditions)
3. Upload-csv
  - a. Allows the user to upload a csv from his respective windows folder.
4. Extract-Data-From-Csv
  - a. Extracts contents of the uploaded csv file for each of the option.
5. View-Edit-Data
  - a. Displays the extracted data in a table format to the user for viewing / editing.
6. Gather-Data-From-DB
  - a. Extracts the latest definitions for processes and its stages from database.
7. Update-DB-Show-Result
  - a. Validates the uploaded data against the data gathered from the database. Collects all the exceptions and shows them back to the user.
  - b. Generates required insert and select queries. Updates the DB and then selects the data for viewing.
  - c. Shows the results back to the user.

### 3.3 State-Engine-Definition-Update-UI

#### 3.3.1 Overview

This flow allows the user to view and update definitions of stages that are not currently executing. They may change whether a stage is disabled or not and has a breakpoint or not, as well as the notification email address(es).

### 3.3.2 States



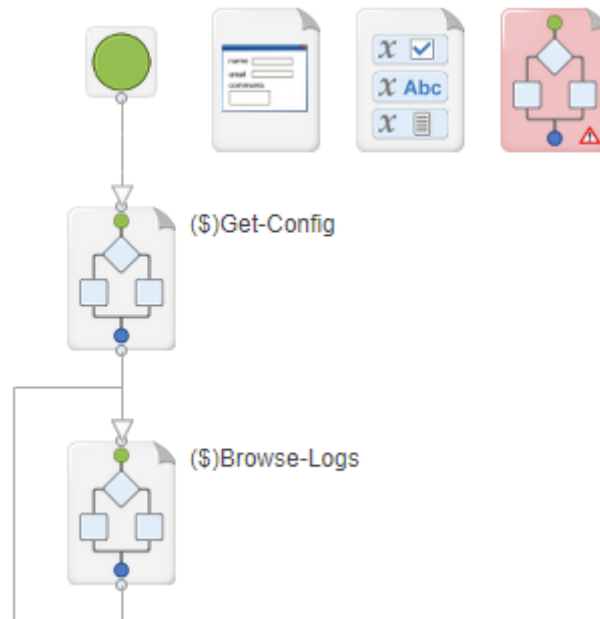
1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
2. Select-Process
  - a. Displays the processes containing stages that are available to configure, allowing the user to select one.
3. Update-Stages
  - a. Allows the user to make any necessary modifications to the stages of the selected process.
4. Generate-Modify-Query
  - a. Builds and executes a SQL query to make the selected changes to the stage definitions in the Cortex-State-Engine database.

## 3.4 State-Engine-Log-Browser-UI

### 3.4.1 Overview

This flow has the same functionality as when viewing logs with State-Engine-Manage-Executions-UI, but allowing the user to select a past process execution to view.

### 3.4.2 States



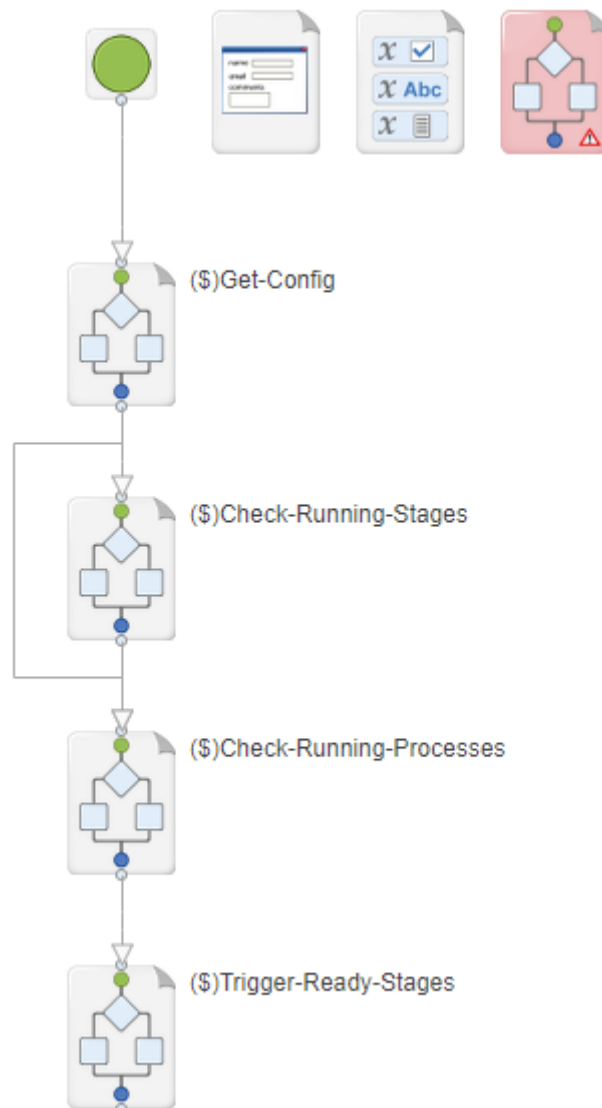
1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
2. Browse-Logs
  - a. Displays a table of past processes.
  - b. When one is selected, displays the stages of that process.
  - c. When one is selected, displays the events in that stage.
  - d. When one is selected, displays the parameters associated with that event.

## 3.5 State-Engine-Monitor-Stage-Executions

### 3.5.1 Overview

This flow checks all flows currently executing that were triggered by the state engine and updates the state engine database with any changes that have occurred since its last execution. It then evaluates the start conditions of any stages that are waiting to start, and if they are fulfilled their associated flows are triggered.

### 3.5.2 States



1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Using them and through evaluating the “hostname” G2 object and LivePortal URL, constructs further parameters including database connection strings.
  - c. Selects the correct Flow API url from the list in the config store.
2. Check-Running-Stages
  - a. Fetches all stage executions from the state engine database that are running, running with errors, or timed out.
  - b. If an associated flow has finished executing since the last update of the database, its stage is set to “completed” or “completed-with-errors” depending on if the flow completed successfully or not.
  - c. If the flow is still executing, if the time since it started is greater than the timeout period then it is set to “timeout”.
3. Check-Running-Processes
  - a. Fetches all process executions from the state engine database where each of their associated stages are in the completed state.
  - b. Logs the ending of the process in the logging database.
  - c. Deletes the process and stage executions from the state engine database.
4. Trigger-Ready-Stages
  - a. Fetches all stages that are ready to start and have their start conditions fulfilled.
  - b. If a stage is not a manual one, and is not disabled or paused:
    - i. Calls its flow via the Flow API.
    - ii. logs the stage starting in the logging database.
    - iii. Sets its state to “Running” in the state engine database.
  - c. If a stage is a manual one, and is not disabled or paused:
    - i. logs the stage starting in the logging database.
    - ii. Sets its state to “Running-Manually” in the state engine database.

### 3.5.3 Inputs

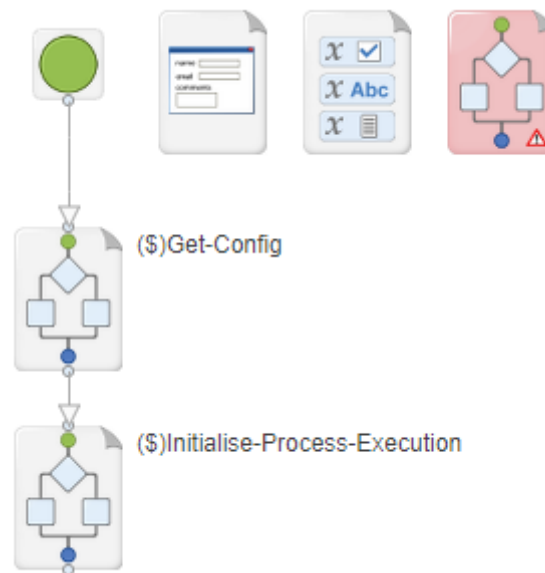
Name	Type	Description	Example
i_only-start-stages	Boolean	TRUE to skip the Check-Running-Stages state. FALSE to include it. Default is FALSE.	TRUE

## 3.6 State-Engine-Start-Process-Execution

### 3.6.1 Overview

This flow starts another execution of a process.

### 3.6.2 States



1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
2. Initialise-Process-Execution
  - a. Inserts a new process execution into the state engine database with the state “running” and logs it.
  - b. Inserts each associated new stage execution into the state engine database with the state “waiting-to-start” and logs them.

### 3.6.3 Inputs

Name	Type	Description	Example
i_Process-To-Start	Text	Name of the process to trigger the execution of	Process1

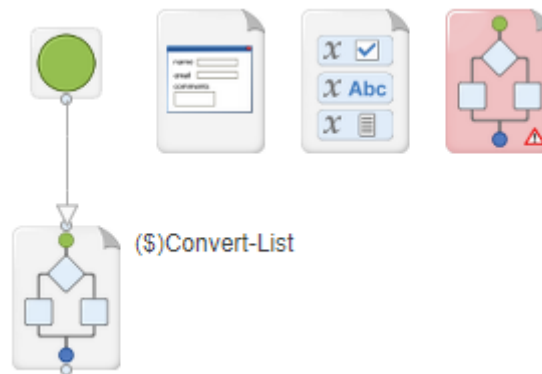


## 3.7 State-Engine-Convert-List-To-HTML

### 3.7.1 Overview

This flow is used by the State-Engine-Params-to-HTML subtask to convert any list parameters to HTML, so they can be displayed in LivePortal.

### 3.7.2 States



1. Convert-List
  - a. Evaluates each element of the list.
  - b. If it is itself a list, calls this flow again synchronously.
  - c. If it is a structure, calls State-Engine-Convert-Structure-To-HTML.
  - d. If is a text/integer/boolean/etc formats into a readable column format.

### 3.7.3 Inputs

Name	Type	Description	Example
LTH_i_List-to-Convert	List	List variable to convert to HTML text that is viewable in LivePortal	[1,2,"hello", TRUE, {colour: red}]

### 3.7.4 Outputs

Name	Type	Description
LTH_o_Output-HTML	Text	HTML text to be displayed in LivePortal

## 3.8 State-Engine-Convert-Structure-To-HTML

### 3.8.1 Overview

This flow is used by the State-Engine-Params-to-HTML subtask to convert any structure parameters to HTML, so they can be displayed in LivePortal.

### 3.8.2 States



1. Convert-Structure
  - a. Evaluates each element of the structure.
  - b. If it is itself a structure, calls this flow again synchronously.
  - c. If it is a list, calls State-Engine-Convert-List-To-HTML.
  - d. If is a text/integer/boolean/etc formats into a readable name: value pair format.

### 3.8.3 Inputs

Name	Type	Description	Example
STH_i_List-to-Convert	Structure	Structure variable to convert to HTML text that is viewable in LivePortal	{name1: value1, name2: [1,2,3], name3:TRUE}

### 3.8.4 Outputs

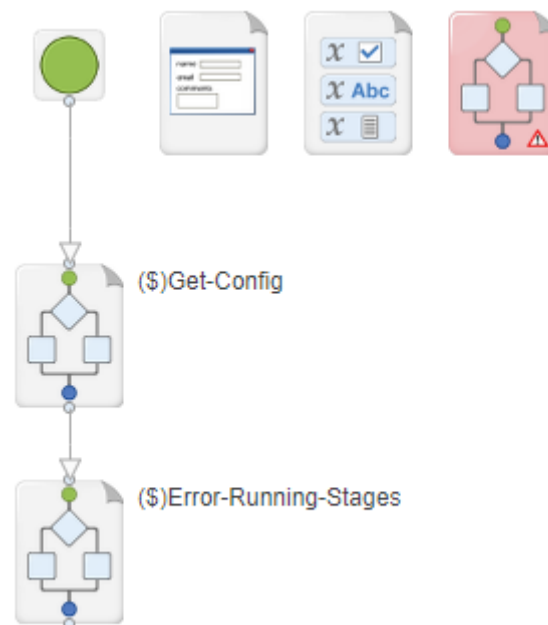
Name	Type	Description
STH_o_Output-HTML	Text	HTML text to be displayed in LivePortal

## 3.9 State-Engine-Disaster-Recovery

### 3.9.1 Overview

This flow is responsible for setting all running stages to the state “Completed with errors”. This should be done in the event of a server shutdown while any are executing, which would result in any flows that were triggered being aborted before they had finished what they were doing.

### 3.9.2 States



1. Get-Config
  - a. Fetches parameters from the config store database.
  - b. Uses them to construct further parameters including database connection strings.
2. Error-Running-Stages
  - a. Retrieve stages that are recorded as “Running” in the Stage\_Executions database table
  - b. Set these to “Completed with errors”
  - c. Notify point of contact as configured in the config store database