CTX-Continuous-Integration User Guide

Contents

Versions

Document Revisions

The following revisions have been made to this document

Date	Revision	Notes
21/02/2019	1.0	First Release

Module Versions

The following revisions have been made to this document

Date	Revision	Notes
21/02/2019	1.0	Creation of the flows: • CCI-Package-Deployment-UI • CCI-Package-Deployment • CCI-Deployment-Process-UI • CCI-Test-Management-UI • CCI-Test-Cases-Execution Creation of the subtasks: • Gateway-EF-Export-Flows • Gateway-GAT-Get-Authentication-Token • Gateway-GIFS-Get-Ids-From-StudioPackage • Gateway-ISP-Import-Studio-Package • Gateway-PSP-Publish-Studio-Package • AutomatedTest-GRTC-Get-Relevant-Test-Cases • AutomatedTest-ITC-Import-Test-Case • AutomatedTest-IVID-Insert-Variables-Into-Database • AutomatedTest-RTOF-Run-Test-On-Flow • AutomatedTest-VTCF-Validate-Test-Cases-Files • CCI-PPR-Process-PowerShell-Response

Preface

About this Manual

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

Audience

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

Related Material

Document
CTX-Continuous-Integration – Deployment Plan
CTX-Continuous-Integration.studiopkg

Abbreviations used in this Document

SQL   Structured Query Language

DB   Database

CCI   Cortex Continuous Integration

JSON   JavaScript Object Notation

Requirements

The CTX-Continuous-Integration module requires the following:

- Minimum Cortex v6.4 installed on the Cortex Application Server
- CTX-Logging Module
- SQL Cortex-Logging database installed
- SQL Cortex-ContinuousIntegration database extension installed
- Cortex PowerShell OCI

Integration

Integration with Third-Party Systems

For the flows and subtasks to work in the CTX-Continuous-Integration module, the expanded Cortex-ContinuousIntegration database and schema needs to exist on the server containing the Cortex databases. The Cortex-ContinuousIntegration database is built on top of the Cortex-Logging database base schema. Instructions how to set up the basic Cortex-Logging database are provided in the 'CTX-Logging – Deployment Plan', and how to add the extra tables required here in 'CTX-Continuous-Integration – Deployment Plan'.

The tables added by the CTX-Continuous-Integration module in the Cortex Continuous Integration schema are:

- TestPlan – Stores details of test plans.

- TestPlan_Case – Links one test plan to one or more test cases.

- TestCase – Stores details of test cases.

- TestCase_Flows – Links one test case to one or more flows, as well as storing the inputs and expected outputs for each test on each flow.

- Flows – Stores details of flows.

- FlowType – Stores different types of flows, typically one of:

  - PreTest – A flow to set up the system ready for a test.

  - Test – A flow to be tested.

  - PostTest – A flow to reset the system once testing has concluded.
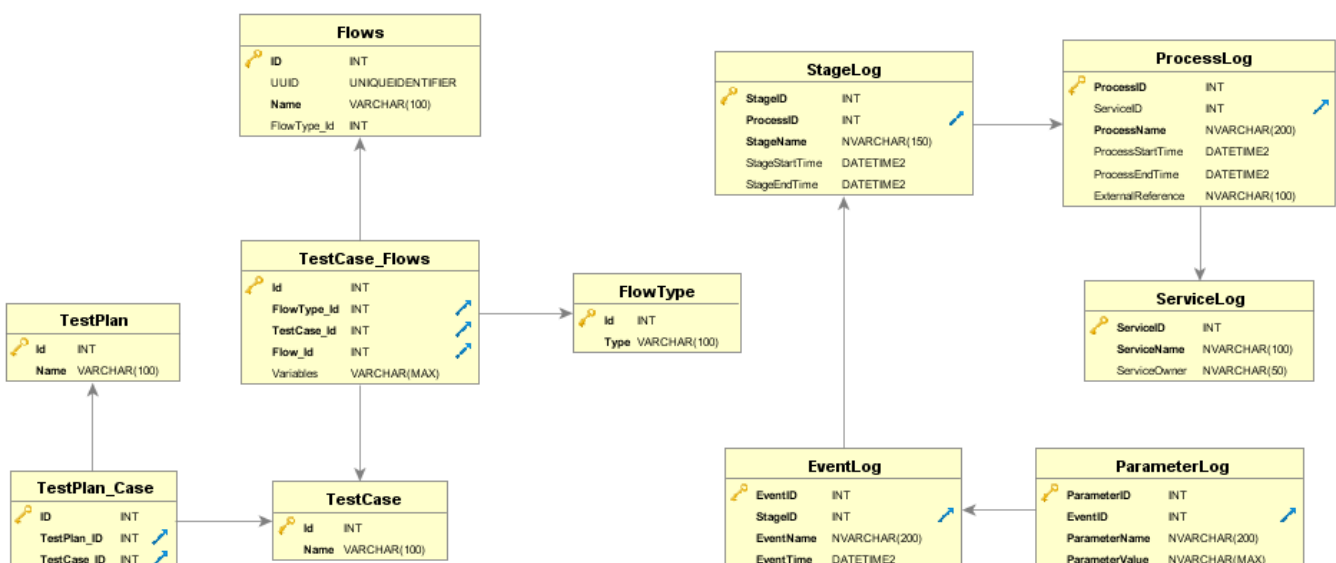
Integration with Existing Infrastructure

None Required.

1 Continuous Integration Overview

'Continuous automation integration' is the practise of repeatedly progressing automation software (Cortex flows) through different environments (Cortex instances). In order to achieve it, Cortex flows and any associated functionality part of the solution need to be promoted between the environments, and testing needs to be performed at each stage to ensure the solution is still functioning as specified.

- Automation of Deployment

  The below solution artefacts need to be taken into consideration:

  o Cortex flows

  o Application configuration

  o Other custom applications

     ⚑ E.g.: External user interfaces

  o Other custom databases

     The promotion of these will depend on the implementation and deployment process, but if it is understood and documented, it can be automated together with the Cortex flows promotion process achieving an end-to-end automated promotion process

- Test Automation

  As mentioned earlier test automation is a critical step of the Continuous Integration practice, which supports the delivery of test results and shortens test execution cycles. It is the method by which software is used to control test executions and the comparison of the outcomes of the tests. This is invaluable when working with large repositories of flows that are used in multiple projects, as for example, if a subtask is used in three different projects and requires a change, steps must be carried out to ensure that none of these changes are breaking its use. In a small repository for a single project, testing manually would be fine, however, in a large repository this can become time consuming. It is also difficult to keep track of the many use cases that a subtask may be required for. Therefore, it is important to write unit tests for flows and subtasks that are going to be reused, store them and easily execute them regularly to ensure their normal operation.

  The process of test automation life cycle includes the below stages:

  o Test design and planning

     Tests should be designed and planned to ensure the requirements of the functionality being tested are met. If the functionality changes overtime, it will be required to modify previous tests or add additional tests to ensure the new requirements are tested.

  o Test management and execution

     Test management and execution helps to track the relationship of tests to requirements, organize the test cases, save the test configuration, execute or plan the execution of the test cases and report on the test execution.

1.1 Using the Module

The current Cortex Continuous Integration module focuses on the promotion of a Cortex .studiopkg package into a single environment and the automated test execution of all the tests associated with the package flows. To achieve this the module deliveries the following functionality:
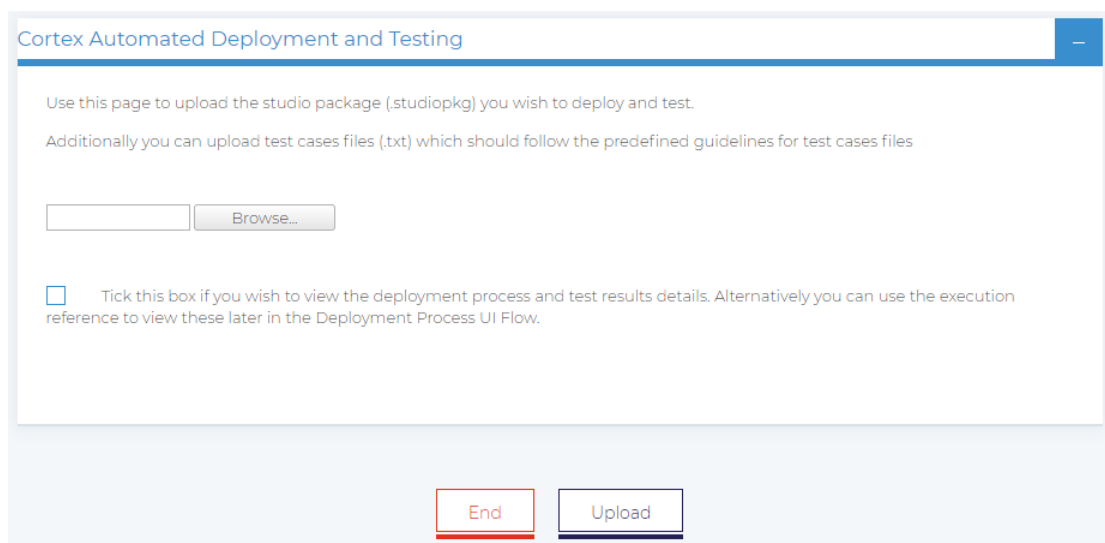
1. Package deployment trigger via LivePortal

    a. User uploads the package files, which comprises of:

        i. Cortex .studiopkg with:

            1. Functional flows and subtasks

            2. Test flows if required

            ⌦ If flows are not built to return variables, there will be the need to build specific test flows that not only execute the required flow but also check if the flow has correctly executed (by for example checking external systems the flow might connect to)

            ⌦ Subtasks to be tested required test flows to be built which wrap up the subtask and expose global variables in order for it to be called by the automated test execution process

        ii. A set of test cases configuration files if required (as specified in Appendix A – Test Case Configuration Best Practices)

        ⌦ This allows for faster upload of test cases into the Continuous Integration database. Alternatively, this can be done using the Test Management LivePortal

2. Automated test execution after package deployment

    a. The Cortex Continuous Integration module will automatically run all tests associated with the flows deployed and log the execution details

3. Test Management LivePortal user interface that allows:

    a. Create, edit, delete and manually execute test cases

    b. Group multiple test cases into test plans that can be then saved for later use

    c. Manually execute test plans

4. Deployment Process and Tests Results LivePortal user interface that allows to view:

    a. The logs of a deployment process

    b. The results of the test executions done as part of the deployment or manually triggered from the CCI-Test-Management-UI flow

---

1.2   User Experience
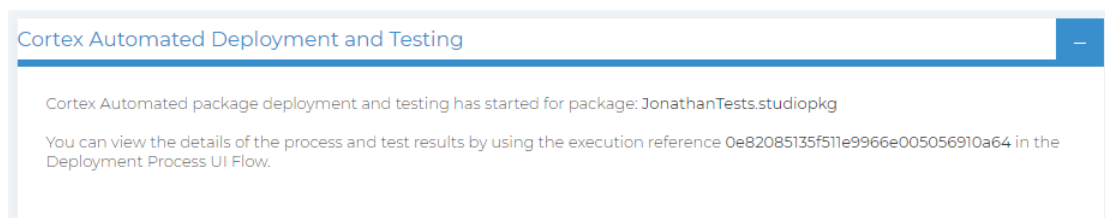
1.2.1   CCI-Package-Deployment-UI

This flow is used to import a test package, consisting of a single .studiopkg file and zero or more test case files.

1. To use this part of the module, the user should run the CCI-Package-Deployment-UI service request through Cortex LivePortal.

2. The user is presented with a page allowing them to upload a single .studiopkg file containing the flows to be deployed and tested, as well as any test case files to be used. The user should also select, by checking the box, if he wishes to wait and see the results or get the execution reference and see it later.

3. Once 'Upload' is selected and the test cases have passed validation, the automated saving, publishing and testing of the flows will commence.

   a. If the user chose not to wait for the results, the next page will inform that the process has started and what the execution reference is.

   b. If the user chose to wait for the results, the next page will show the package upload and tests results details

1.2.2   CCI-Test-Management-UI

The 'CCI-Test-Management-UI' is used by the user to

- Create, edit, delete and execute test cases provided that a flow has been imported and tested at least once using 'CCI-Package-Deployment-UI' outlined in Section 1.2.1.

- Group multiple test cases into test plans that can be then saved for later use.

- Execute test plans.

1. To use this part of the module, the user should run the CCI-Package-Deployment-UI service request through Cortex LivePortal.

2. The user is presented with a menu where may select whether to manage test cases or test plans.

⚑   Hovering the cursor over buttons can reveal more information about the results of pressing it
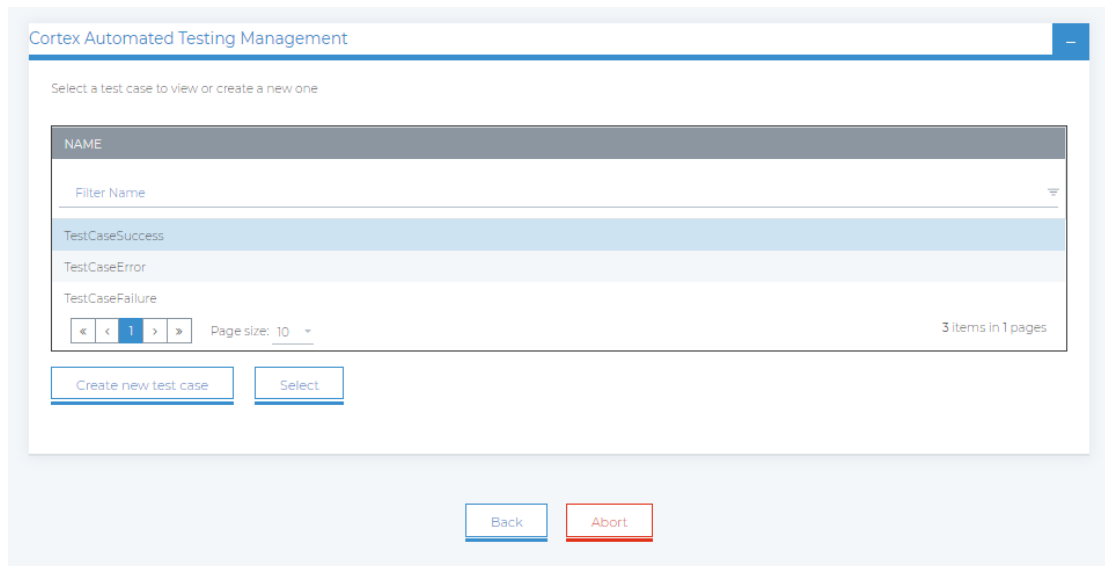
1.2.2.1 Manage Test Cases

1. If the user selects the option to managed test cases, they will be presented with a list of test cases, from which they may select one to view and/or edit, or select the option to create a new one.

2. To create a test case, the user should select 'Create new test case' from the above screen.

   a. The user is presented with a text field in which the test case JSON must be entered.

   b. It is advised to use a JSON editor to do this, then the text copied and pasted here.

   c. The text field may be resized by clicking and dragging the bottom right corner to the desired height.

   d. The test case JSON takes the form a list of structures, with each structure containing the 'FlowName' text, 'Inputs' structure and 'ExpectedOutputs' structure. For more guidance in creating and formatting this, see Appendix A.

Cortex Automated Testing Management ___

Create A New Test Case

Test Case Name *
Calculator-Test

Test Case JSON
```
[
  {
    "FLOW": "CCJ-Calculator",
    "EXPECTEDOUTPUTS": {
      "o_result": 2,
      "o_error": ""
    },
    "INPUTS": {
      "I_A": 1,
      "I_B": 1,
      "I_C": "Add"
    }
  }
]
```

Save

Back   Abort

3. To edit a test case, select it from the 'Select Test Case' screen.

    a. The user will be presented with an identical page to when creating test cases, but with the name and JSON pre-populated.

    b. Selecting 'Delete' will remove the test case.

    c. Selecting 'Execute' will run the test case on the identified flow with the inputs provided and store the results in the Cortex-Logging database.

Cortex Automated Testing Management ___

Edit Calculator-Test's details

Test Case Name *
Calculator-Test

Test Case JSON
```
[
  {
    "FLOWNAME": "Calculator",
    "EXPECTEDOUTPUTS": {
      "o_Result": 2,
      "o_Error": ""
    },
    "INPUTS": {
      "i_A": 1,
      "i_B": 1,
      "i_C": "Add"
    }
  }
]
```

!
Save   Delete   Execute
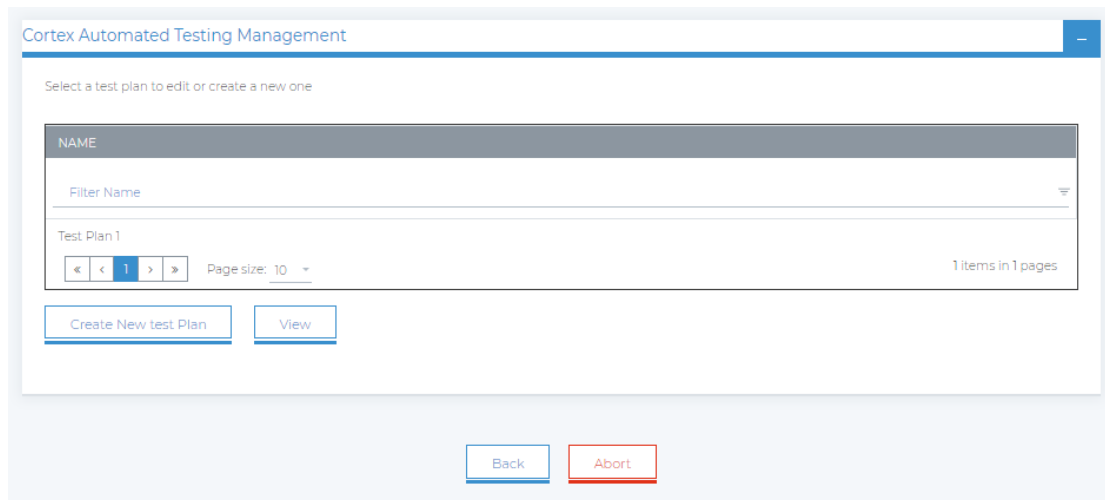
☐ Tick this box if you wish to view the deployment process and test results details. Alternatively you can use the execution reference to view these later in the Deployment Process UI Flow.

Back   Abort

1.2.2.2   Manage Test Plans

1. If the user selects the option to manage test plans, they will be presented with a list of test plans from which they may select one to view and/or edit, or to create a new one.
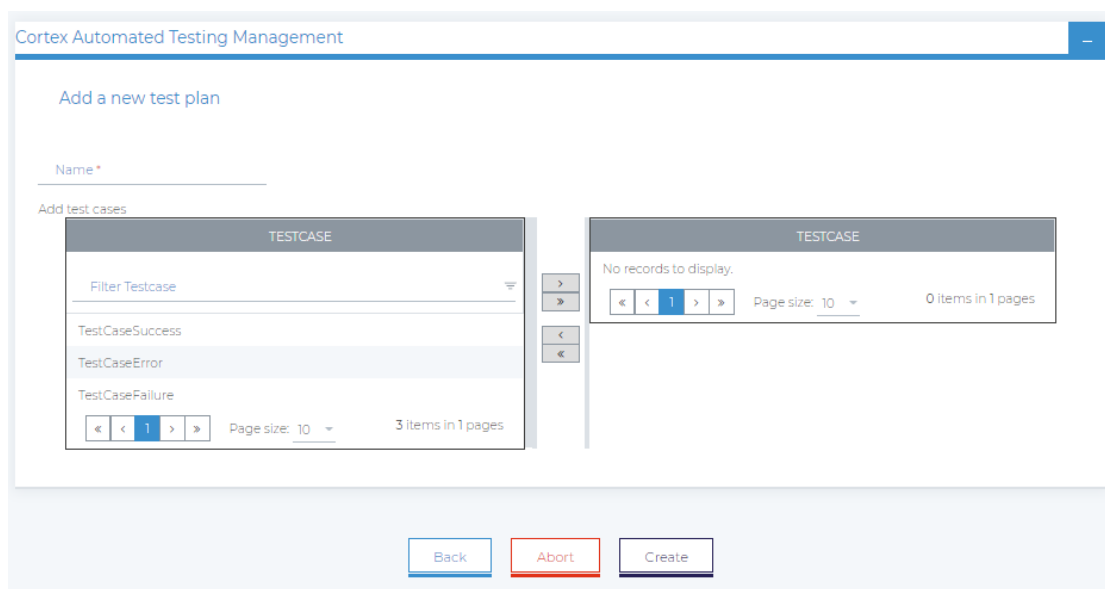
2. To create a new test plan the user must select 'Create a New Test Plan from the above screen.

   a. They are then shown a screen where they may define the name of their new test plan and add test cases to it.
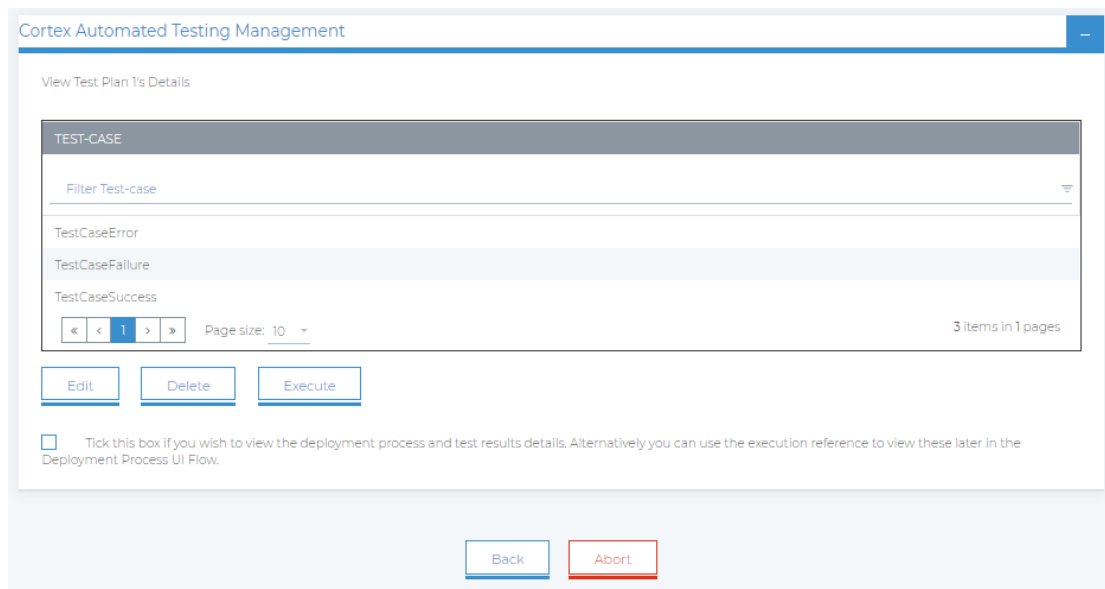
3. To view the test cases associated with a test plan, select a test plan from the 'Select Test Plan' screen and press 'View'.

   a. The user may choose to edit, delete or execute the test plan from here

b. Selecting 'Edit' will present the user with a screen identical to when creating the test plan, but with the name and associated test cases pre-populated.

c. Selecting 'Delete' will remove the test plan, but the individual test cases will remain, so they may be executed or added to another test plan in the future.

d. Selecting 'Execute' will run the associated test cases sequentially on their associated flows and store the results in the Cortex-Logging database.

1.2.3 CCI-Deployment-Process-UI

The 'CCI-Deployment-Process-UI' is used to view the logs of a deployment process as well as the results of the test executions done as part of the deployment or manually triggered from the CCI-Test-Management-UI flow.

1. To use this part of the module, the user should run the CCI-Deployment-Process-UI service request through Cortex LivePortal.

2. The user is presented with a page allowing them to specify an execution reference for which the process details and tests results will be fetched.

Cortex Automated Deployment and Testing Log Viewing   —

Please provide the Automated Deployment and/or testing execution reference to retrieve the details.

Example: 2968fae12f8611e9966d005056910a64

Execution Reference *

[End]   [Next]

3. The user is presented with page which will show two tables:

   a. The process details

   b. The performed tests results details

Cortex Automated Deployment and Testing   —

Process View:

PROCESSNAME	STAGENAME	STARTTIME	ENDTIME
Filter Processname	Filter Stagename	Filter Starttime	Filter Endtime
Package Deployment	Import Package	2019-02-21 16:28:16	2019-02-21 16:32:42
Package Deployment	Perform Automated Tests	2019-02-21 16:32:42	2019-02-21 16:32:53

« ‹ 1 › »   Page size: 10   2 items in 1 pages

Tests Results:

TEST	FLOW	RESULT	RESULT_DETAIL	INPUTS	EXPECTEDOUTPUTS	ACTUALOUTPUTS
Filter Test	Filter Flow	Filter Re...	Filter Re...	Filter Inp...	Filter Expe...	Filter Actua...
Test Plan 1 - TestCaseError	CCI-Jonathan-Test	FAIL	Error: Invalid operator	structure (I_A: 1, I_B: 1, I_C: "Exp")	structure (O_RESULT: 1, O_ERROR: "")	structure (O_ERROR: "Error: Invalid operator")
Test Plan 1 - TestCaseFailure	CCI-Jonathan-Test-Two	FAIL	None	structure (I_A: 1, I_B: 1, I_C: "Add")	structure (O_RESULT: 3, O_ERROR: "")	structure (O_RESULT: 2)
Test Plan 1 - TestCaseSuccess	CCI-Jonathan-Test-Two	PASS	None	structure (I_A: 1, I_B: 1, I_C: "Add")	structure (O_RESULT: 23, O_ERROR: "")	structure (O_RESULT: 2)
Test Plan 2 - TestCaseSuccess	CCI-Jonathan-Test-Two	PASS	None	structure (I_A: 1, I_B: 1, I_C: "Add")	structure (O_RESULT: 23, O_ERROR: "")	structure (O_RESULT: 2)

« ‹ 1 › »   Page size: 10   4 items in 1 pages

[Back]   [End]

2   Continuous Integration Subtasks

2.1   Gateway-EF-Export-Flows

2.1.1   Overview

This subtask exports a set of Flows and Subtasks from Gateway and stores the export package file in a location.

2.1.2   Input variables

Name	Type	Comments
EF_i_Bearer-Auth-Token	Text	REQUIRED, the Gateway access token. The format is Bearer <tokenid>.
EF_i_GatewayURL	Text	REQUIRED, the URI for the Cortex Gateway. Example: https://<servername>/gateway
EF_i_Export-Location	Text	REQUIRED, the location where the export file will be saved.
EF_i_File-Name	Text	REQUIRED, the export file name
EF_i_Flows-Ids	List	REQUIRED, a list of Flows and Subtasks Ids that will be exported.

2.1.3   Output variables

None

2.2   Gateway-GAT-Get-Authentication-Token

2.2.1   Overview

This subtask logs in to Cortex Gateway using a REST request.

This returns a Bearer Token which can be used to authenticate REST requests such as Importing, Exporting and Publishing flows.

2.2.2   Input variables

Name	Type	Comments
GAT_i_GatewayURL	Text	Required, the URI for the Cortex Gateway. Example: https://<servername>/gateway
GAT_i_GatewayUser	Text	Required, the Gateway user with admin permissions. This value can be Cortex encrypted.
GAT_i_GatewayPassword	Text	Required, the Gateway user password. This value can be Cortex encrypted.

2.2.3   Output variables

Name	Type	Comments
GAT_o_Bearer-Auth-Token	Text	The user authentication token. The format is "Bearer <tokenid>"

2.3   Gateway-GIFS-Get-Ids-From-StudioPackage

2.3.1   Overview

This subtask gets the names, ids and types of flows from a specific Cortex studio package.

2.3.2   Input variables

Name	Type	Comments
GIFS_i_File-Location	Text	REQUIRED, the location where the export file will be saved.
GIFS_i_File-Name	Text	REQUIRED, the export file name.

2.3.3   Output variables

Name	Type	Comments
GIFS_o_Flows-Ids	List	A list of the Flows and Subtasks Ids from the Cortex studio package.

2.4   Gateway-ISP-Import-Studio-Package

2.4.1   Overview

This subtask uploads and imports a Cortex Gateway studio package file.

2.4.2   Input variables

Name	Type	Comments
ISP_i_Bearer-Auth-Token	Text	REQUIRED, the Gateway access token. The format is Bearer <tokenid>.
ISP_i_GatewayURL	Text	REQUIRED, the URI for the Cortex Gateway. Example: https://<servername>/gateway
ISP_i_Export-Location	Text	REQUIRED, the location where the export file will be saved.
ISP_i_File-Name	Text	REQUIRED, the export file name.

2.4.3   Output variables

Name	Type	Comments

ISP_o_Package-Id	Text	The uploaded package id.

2.5 Gateway-PSP-Publish-Studio-Package

2.5.1 Overview

This subtask publishes a set of Flows and Subtasks.

2.5.2 Input variables

Name	Type	Comments
PSP_i_Bearer-Auth-Token	Text	REQUIRED, the Gateway access token. The format is Bearer <tokenid>.
PSP_i_GatewayURL	Text	REQUIRED, the URI for the Cortex Gateway. Example: https://<servername>/gateway
PSP_i_Flows-Ids	List	REQUIRED, a list of Flows and Subtasks Ids that will be published.
PSP_i_Server-Name	Text	REQUIRED, the Server name where the Flows and Subtasks Ids will be published.
PSP_i_Gateway-Database-Server	Text	The Cortex Gateway database server name. Default value is localhost
PSP_i_Gateway-Database-Name	Text	The Cortex Gateway database name. Default value is CortexWeb..

2.5.3 Output variables

None

2.6 AutomatedTest-GRTC-Get-Relevant-Test-Cases

2.6.1 Overview

This subtask gets the test cases associated with either a collection of test plans or with a collection of flows. If both are provided will get based on test plans

2.6.2 Input variables

Name	Type	Comments
GRTC_i_ Database-Server	Text	REQUIRED, Server on which the Continuous Integration database is hosted
GRTC_i_ Database-Name	Text	REQUIRED, Name of the Continuous Integration database
One of the below two variables is REQUIRED:		

GRTC_i_Test-Plans	Text	List of test plans with which to find associated test cases
GRTC_i_Flows-IDs	List	List of flow Ids with which to find associated test cases

2.6.3   Output variables

Name	Type	Comments
GRTC_o_Test-Cases	List	List of test cases associated with inputted test plans or flow ids

2.7 AutomatedTest-ITC-Import-Test-Case

2.7.1   Overview

This subtask imports a test case from the Continuous Integration database and converts it to a list of structures.

2.7.2   Input variables

Name	Type	Comments
ITC_i_ Database-Server	Text	REQUIRED, Server on which the Continuous Integration database is hosted
ITC_i_ Database-Name	Text	REQUIRED, Name of the Continuous Integration database
ITC_i_Test-Case-Name	Text	REQUIRED, Name of the test case to be retrieved.

2.7.3   Output variables

Name	Type	Comments
ITC_o_Test-Case	List	List of tests, each containing a flow name and the input and expected output variables.

2.8 AutomatedTest-IVID-Insert-Variables-Into-Database

2.8.1   Overview

This subtask inserts a test case into a database if it does not already exist. If it does exist it will update the input and expected output variables.

2.8.2   Input variables

Name	Type	Comments
IVID _i_ Database-Server	Text	REQUIRED, Server on which the Continuous Integration database is hosted. Example: localhost

IVID _i_ Database-Name	Text	REQUIRED, Name of the Continuous Integration database. Example: Cortex-ContinuousIntegration
IVID_i_Flow-Name	Text	Name of the flow associated with the test case. Example: TestFlowOne
IVID_i_Variables	Text	REQUIRED, Inputs and expected output variables for the test case, for details on formatting this see Section 4.
IVID_i_Test-Case-Name	Text	REQUIRED, Name of the test case to be inserted or updated. Example: TestCase001

2.8.3   Output variables

None

2.9   AutomatedTest-RTOF-Run-Test-On-Flow

2.9.1   Overview

This subtask executes a test on a flow.

2.9.2   Input variables

Name	Type	Comments
RTOF_i_Test	Structure	Contains the flow name, the inputs and expected outputs
RTOF_i_Gateway-URL	Text	Cortex Gateway URL on the appropriate environment

2.9.3   Output variables

Name	Type	Comments
RTOF_o_Report	Structure	Contains the flow name, the result of the test (Pass or Fail), the inputs, the expected outputs, the actual outputs, the error code, the description of any errors.

2.10 AutomatedTest-VTCF-Validate-Test-Cases-Files

2.10.1 Overview

This subtask ensures the test case configuration JSON containing the Inputs and Outputs is of the correct format, if it is converts it to list.

2.10.2 Input variables

Name	Type	Comments
VTCF_i_Test-Case-Configuration-Test	Text	REQUIRED, the test case configuration JSON to be validated
VTCF_i_Test-Case-Name	Text	REQUIRED, the name of the test case to be validated

2.10.3 Output variables

Name	Type	Comments
VTCF_o_Test-Case-Exception	Text	Exception message containing the reason why input is not formatted properly.
VTCF_o_Test-Cases-List	List	List of structures containing the Flow Name, Inputs and Expected Outputs of the test case.

2.11 CCI-PPR-Process-PowerShell-Response

2.11.1 Overview

This subtask processes a PowerShell response. If there were any error records it appends all these and raises an error. If not, it can either return the output as a list or as a text depending on the user's choice.

2.11.2 Input variables

Name	Type	Comments
PPR_i_PS-Response	Structure	REQUIRED, the response from the PowerShell block
PPR_i_Convert-To-Text-Bool	Boolean	Truth value where 'true' indicates to return it as text

2.11.3 Output variables

| Name	Type	Comments
Only one of these outputs will be populated.		
PPR_o_PS-Text	Text	Variable containing the output as text
PPR_o_PS-List	List	Variable containing the output in a list

3   Continuous Integration Flows

3.1   CCI-Package-Deployment-UI

3.1.1   Overview

The 'CCI-Package-Deployment-UI' flow is used to:

- Import a test package, consisting of a single .studiopkg file and zero or more test case files.

- If not already done, save the flows in the .studiopkg file and the tests in the test case files to the database. If they are already in the database, then they will be updated.

- Trigger the publishing and automated testing of the flows based on the relevant test cases.

3.1.2   States
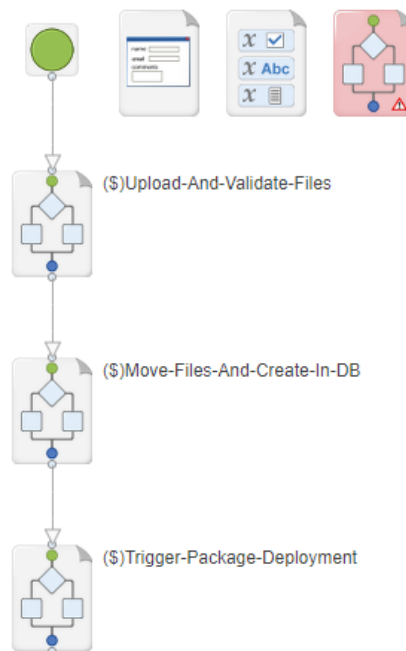
- Upload-And-Validate-Files

Fetches parameters such as the database server, Cortex Gateway URL and credentials from C:\Cortex\ContinuousIntegration.txt. These are stored in the g_config global structure variable.

The user is presented with a screen where they can select a .studiopkg file containing the flows they wish to test, as well as any test case files. Upon selecting 'upload', these files are validated and if any do not pass then the user is returned to the upload screen with a message detailing this.

- Move-Files-And-Create-In-DB

The files are moved to a directory defined in g_config and the flows contained in the uploaded .studiopkg file are added to the Cortex-Logging DB.

As well as this, for each test case file uploaded, if a test case with the same name already exists then it is updated with the details contained within the file, otherwise a new test case is created.

- Trigger-Package-Deployment

The current master versions of any of the uploaded flows already on the Cortex Gateway defined in g_config are backed up, the versions in the uploaded .studiopkg are imported and published, and the automatic testing of them is triggered.

3.1.3 Inputs

None

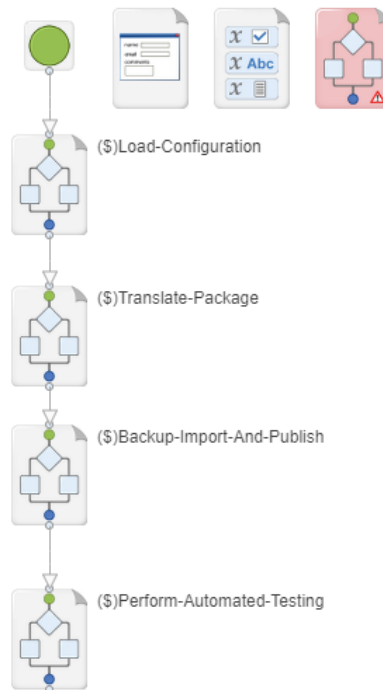### 3.1.4 Outputs

None

3.2 CCI-Package-Deployment

3.2.1 Overview

The 'CCI-Package-Deployment' flow is used to:

- Import a new studio package into the relevant Cortex system

- Trigger automatically the test cases associated with the flows that belong to the package

3.2.2 States

- Load-Configuration

  Loads the Continuous Integration configuration from the configuration file and generates the Gateway Authentication token to be used in the next steps.

- Translate-Package

  Reads the flow package and gets a list of flows. Based on the flows connects to the Continuous Integration database to retrieve all associated test cases.

- Backup-Import-And-Publish

  Backups the flows from the current Cortex system based on the flows to be imported. After the back up is done, the new package will be imported and published into the Cortex System.

- Perform-Automated-Testing

  Synchronously calls the CCI-Test-Cases-Execution flow with the list of test cases retrieved in the translate package state.

3.2.3 Inputs

Name	Type	Comments
i_Studio-Package-Name	Text	REQUIRED, the name of the studio package to be deployed in the system.

3.2.4   Outputs

| Name	Type	Comments
o_Flow-Reference	Text	The flow execution reference identifier.

3.3   CCI-Deployment-Process-UI

3.3.1   Overview

The 'CCI-Deployment-Process-UI' is used to view the logs of a deployment process as well as the results of the test executions done as part of the deployment or manually triggered from the CCI-Test-Management-UI flow.

This flow can be triggered from LivePortal or by another flow. When triggered from another flow the required input needs to be provided.

3.3.2   States
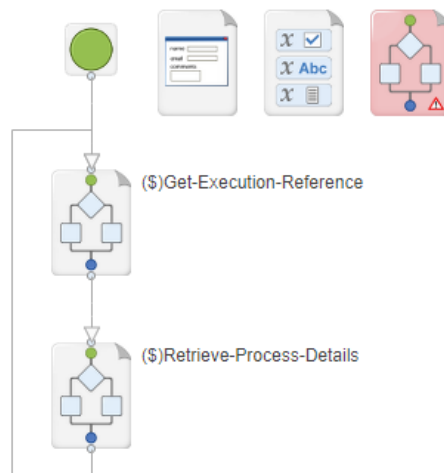
- Get-Execution-Reference

  If the flow is triggered from LivePortal, allows the user to retrieve the process and test executions details using an execution reference execution reference.

- Retrieve-Process-Details

  Retrieves the process and test executions details from the Continuous Integration database based on the provided execution reference.

3.3.3   Inputs

Name	Type	Comments

| i_Execution-Reference | Text | REQUIRED, the deployment process or test execution reference identifier. |

3.3.4   Outputs

Name	Type	Comments
o_Test-Results	Table	A table containing the test results
o_Process-View	Table	REQUIRED, the name of the test case to be validated

3.4   CCI-Test-Management-UI

3.4.1   Overview

The 'CCI-Test-Management-UI' is applied by the user to

- Create, edit, delete and execute test cases provided that a flow has been imported and tested at least once using 'CCI-Package-Deployment-UI' outlined in Section 1.2.1.

- Group multiple test cases into test plans that can be then saved for later use and executed here.

3.4.2   States
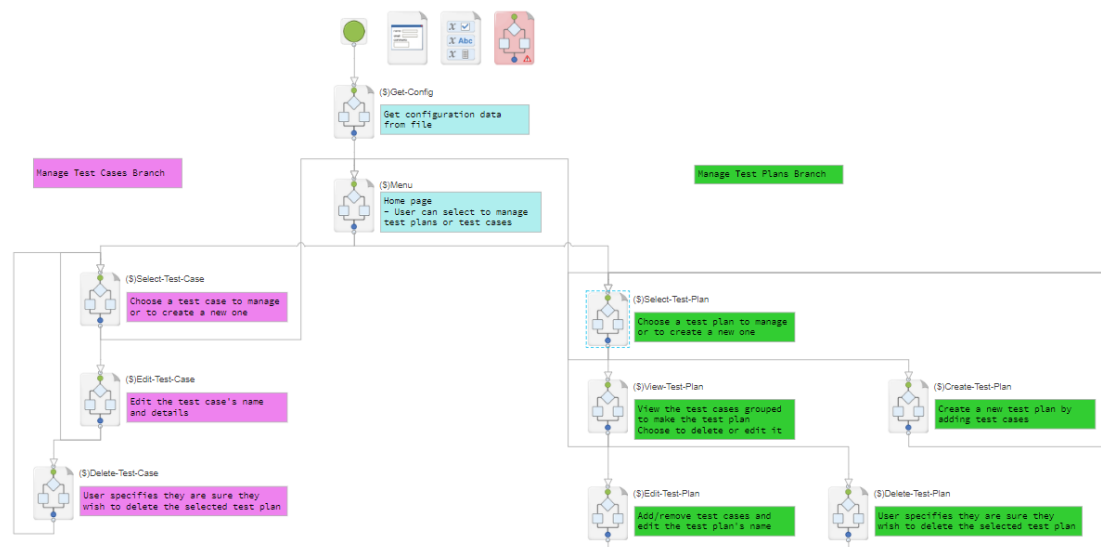
- Generic Branch

  o   Get-Config

  Fetches parameters such as the database server, Cortex Gateway URL and credentials from C:\Cortex\ContinuousIntegration.txt. These are stored in the g_config global structure variable.

  o   Menu

Allows the user to select whether they wish to manage test cases or test plans.

- Manage test cases branch

  o   Select-Test-Case

Allows the user to select to create a test case or edit a previously created one.

  o   Edit-Test-Case

If the user selected to create a test case, they will be presented with two blank text fields. They should enter the name of their new test case into the first and paste the details of the flows, inputs and expected outputs into the second.

If the user selected to edit a test case, they will be presented the with the same screen, but with the text fields pre-populated with that test case's details. This may be edited, deleted or executed from here.

  o   Delete-Test-Case

Checks with the user that they do indeed wish to delete the selected test case, if they do then it carries out the deletion by removing the test case and its relationship with any flows from the Cortex-Logging DB (but not the flows themselves, these may be reused).

- Manage test plans branch

  o   Select-Test-Plan

Allows the user to select to create a test plan or view a previously created one.

  o   View-Test-Plan

The user is shown a grid containing the test cases belonging to the selected test plan. The user may choose to edit, delete of execute it from here.

  o   Create-Test-Plan

The user is presented with a text field to accept the name of a new test plan, and a double grid from which they may add test cases.

  o   Edit-Test-Plan

The user is presented with a text field to pre-populated with the name of the selected test plan, and a double grid from which they may add or remove test cases.

  o   Delete-Test-Plan

Checks with the user that they do indeed wish to delete the selected test plan, if they do then it carries out the deletion by removing the test plan and its relationship with any test cases from the Cortex-Logging DB (but not the test cases themselves, these may be reused).

3.4.3 Inputs

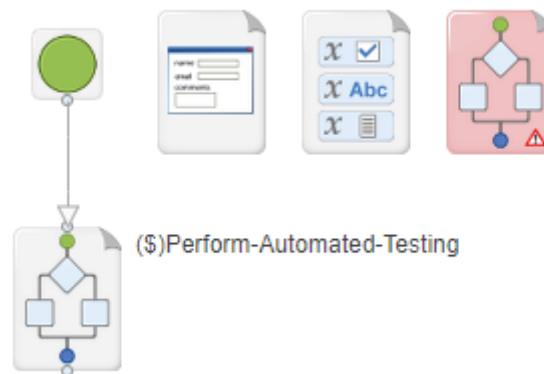None

3.4.4 Outputs

None

3.5 CCI-Test-Cases-Execution

3.5.1 Overview

The 'CCI-Test-Cases-Execution' is used to execute a set of test cases and log the results against the Continuous Integration database.

3.5.2 States

($)Perform-Automated-Testing

- Perform-Automated-Testing

  Loops over the test cases list, retrieves the test case inputs and expected outputs, and executes the test case flows. The test execution details are logged in the Continuous Integration database for audit and reporting purposes.

3.5.3 Inputs

Name	Type	Comments
i_Test-Cases-List	List	REQUIRED, the list of test cases to be run.
i_Commit-Logs	Text	A true or false text to control whether the logs will be committed in this flow or saved in the output variable o_Logs-To-Commit. This is coupled with the use of the CTX-Logging module
i_Stage-Name	Test	Name of the stage that will be created when the flow is executed. This is coupled with the use of the CTX-Logging module
i_Logs-To-Commit	Structure	Contains logging information. This variable should not be manually changed. Only required if logs

		were created outside the scope of this flow. This is coupled with the use of the CTX-Logging module.

3.5.4   Outputs

Name	Type	Comments
o_Logs-To-Commit	Structure	Contains logging information. This is coupled with the use of the CTX-Logging module.
o_Flow-Reference	Text	The flow execution reference identifier.

4   Appendix A – Test Case Configuration Best Practices

4.1 Overview

The test case configuration JSON is a text file visualising a list of structures in a specific format that is defined here. It is used to define the details of a test on a .studiopkg file containing one or more flows. For each of the flows in the package that the user wishes to test, they must include a structure detailing

1. The flow name, as text.

2. The outputs that are expected for the test to pass, as a structure.

3. The variables to be inputted into the flow, as a structure.

The final test case file should be saved as "<test case name>.txt" and be of the following form:

```
[
  {
    "FLOWNAME": "<First flow name>",
    "EXPECTEDOUTPUTS": {
      "<First output name>": "<First output value>,
      "<Second output name>": "<Second output value>,
      ...
    },
    "INPUTS": {
      "<First input name>": "<First input value>,
      "<Second input name>": "<Second input value>,
      ...
    }
  },
  ...
]
```

It is recommended to use a JSON editor to achieve the correct formatting, there are many online options available. It is worth noting that in some editors, structures are referred to as 'objects' and likewise lists as 'arrays', so long as the final text has curly '{ }' brackets and square '[ ]' brackets for structures and lists respectively then it will function as expected.

4.2 Example – Testing a Calculator Flow

Consider a simple flow called `Calculator` that takes the three inputs

1. `i_A`: Integer – The first number to operate on

2. `i_B`: Integer – The second number to operate on

3. `i_C`: Text – The operation to apply to `i_A and i_B`

and has the two outputs

1. `o_Result`: Integer – the result of the operation

2. `o_Error`: Text – any exceptions that may have occurred.

Let us suppose that the user wishes to check that the calculator flow correctly adds one and one together to return two. They would therefore construct a test case file as follows:

```
[
  {
    "FLOWNAME": "Calculator",
    "EXPECTEDOUTPUTS": {
      "o_Result": 2,
      "o_Error": ""
    },
    "INPUTS": {
      "i_A": 1,
      "i_B": 1,
      "i_C": "Add"
    }
  }
]
```