Best Practice

Leveraging Best Practice in Test Modeller

A quick guide to leveraging best practice modelling techniques for creating test assets.

Test Modeller gives users the ability to generate **Test Cases, Test Data and Test Automation** from a **BPMN** visual representation concept (Business Process Model and Notation) of a system under test (SUT). This Best Practice guide of Test Modeller functionality focuses on you generating assured sets of Test Artifacts.

An outline of the Test Modeller tool

Is **set-up with an Automation Frameworks** to push code out according to your environment, inc: **Cypress,** Dynamics365, **Java Selenium,** Postman, C# Selenium, amongst others.

Connects **your Test Cases Management Systems** (ie, JIRA), your **Source Control** (ie, GitHub), **Modelling tools** (ie, Draw.io) amongst others.

UI Scanner **scans a page to auto-create the Automation functions to Overlay on top of your models.** From that the models generate end-to-end Automation Scripts ready to be embedded with a CI/CD pipeline.

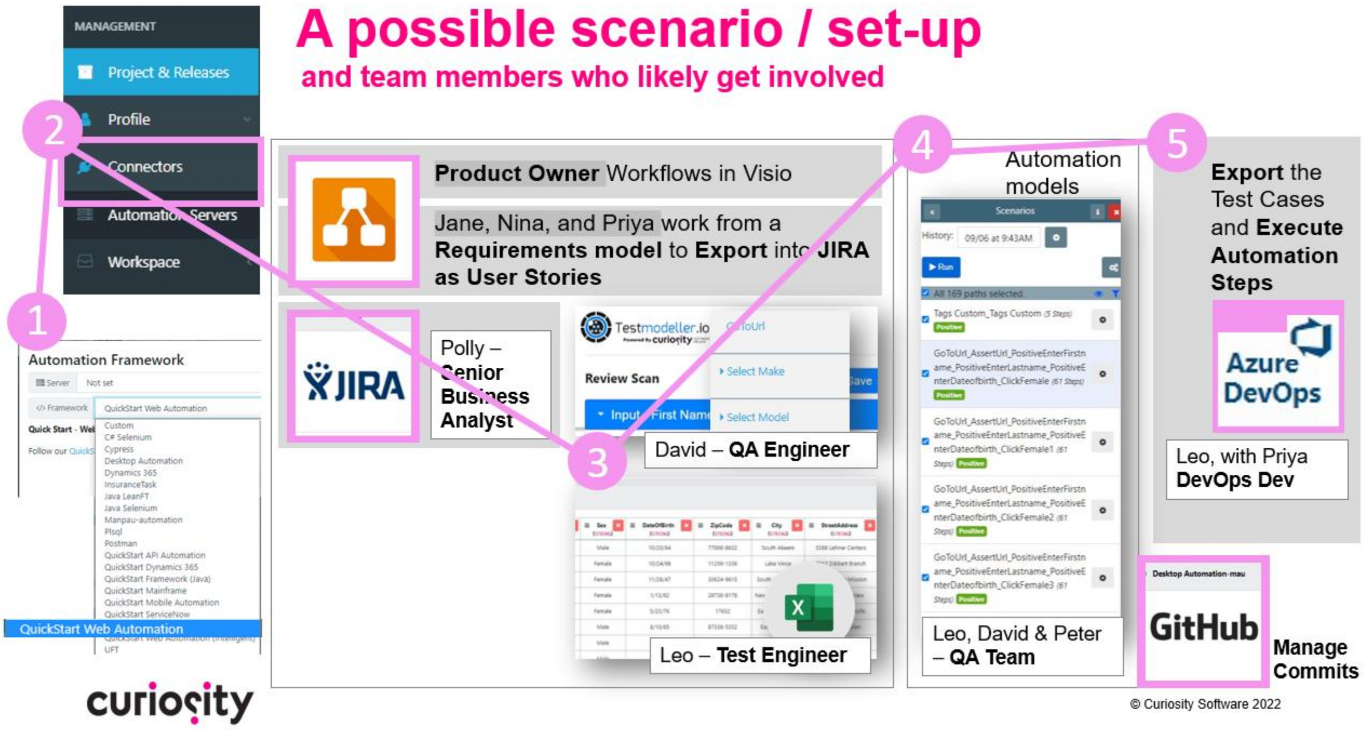
Coverage **is easily staged in a model on a canvas using the BPMN graphical representation concept** (Business Process Model and Notation) so all team members contribute easily.

Key benefits

- Set-up in an Environment you're already familiar with
- Reduces time and ambiguity in setting up Test Cases
- Coverage for Regression Testing can be set differently to exhaustively testing new functionality
- Allows contribution across the to make richer and targeted Test Cases
- Test Cases can be updated quickly as new Requirements get added

A possible scenario / set-up

and team members who likely get involved



Product Owner Workflows in Visio

Jane, Nina, and Priya work from a **Requirements model** to **Export** into **JIRA as User Stories**

Polly – **Senior Business Analyst**

David – **QA Engineer**

Leo – **Test Engineer**

Leo, David & Peter – **QA Team**

Export the Test Cases and **Execute Automation Steps**

Leo, with Priya **DevOps Dev**

Manage Commits

© Curiosity Software 2022

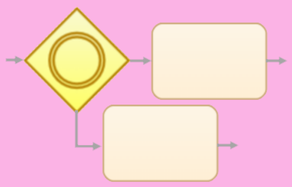
Adopt a mindset of modelling Requirements, Behaviour, Business Logic and Rules rather than individual Test Cases. Once done, Generating Test Cases is automatic.

Start and End blocks (nodes) help define the endpoints/outcomes possible in the System Under Test (SUT) and are the basis for mapping out basic outcomes, be it Requirements and/ or functionality.

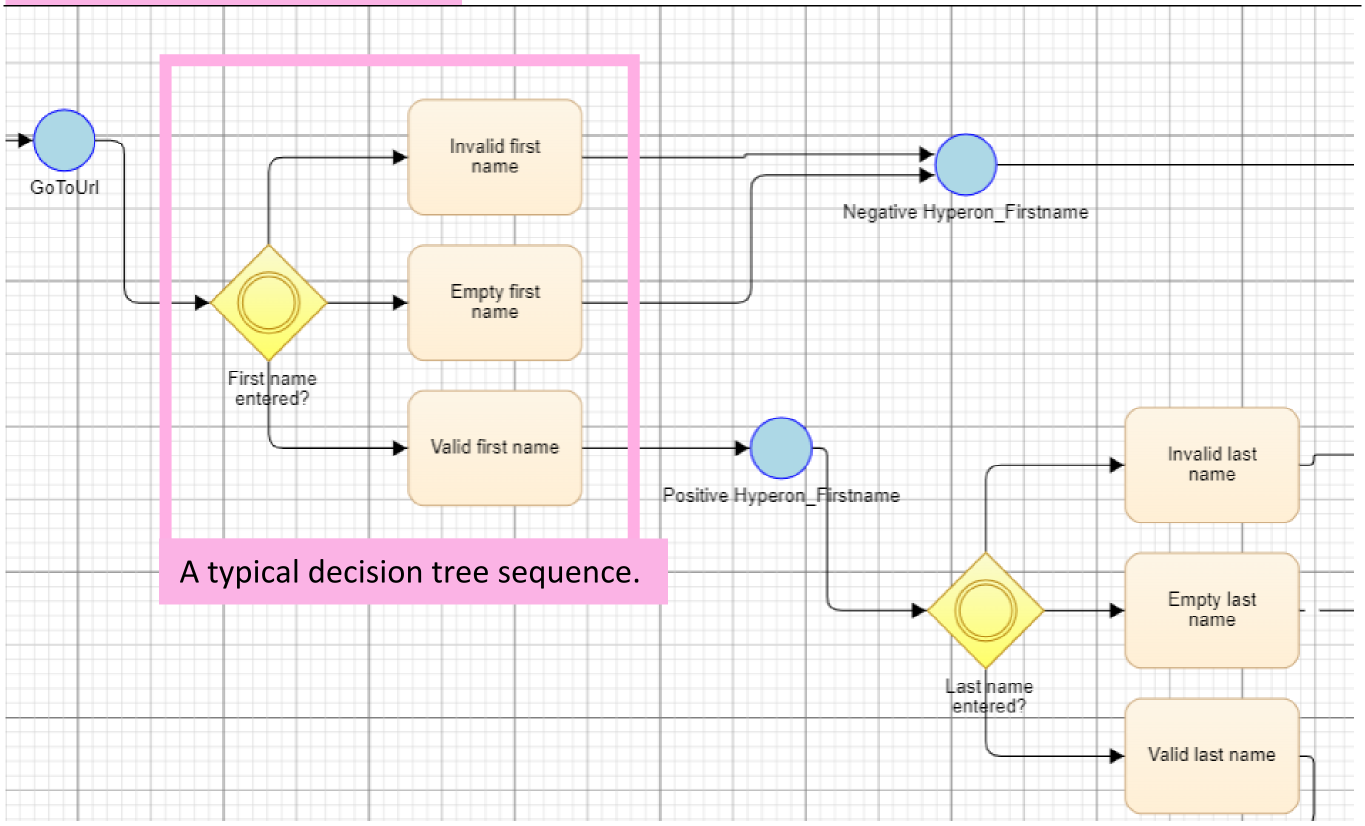
Begin with positive Scenarios (successful User Outcomes eg, user is logged in), and then add edge cases for negative Scenarios.

Map any new functionality to the existing model and Regenerate new Test Suites.

1 Decision Trees

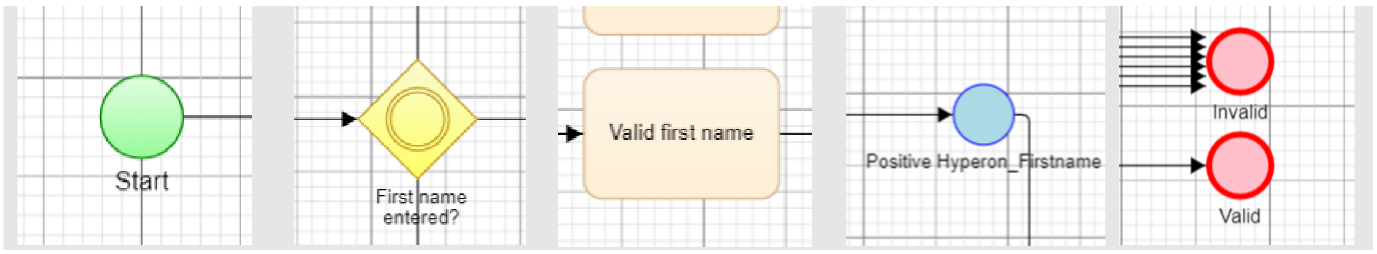


- Model out using the recommended block sequence
- Basic nodes and descriptive labels
- Node usage



The recommended sequence for modelling within the canvas is a Start and End node, then between these follow with Conditions and Task blocks which can be repeated.

Basic nodes Condition Task block Waypoint End



Descriptive Labels

Label the Condition to describe what decision is being taken, like 'User already exists?' Or relate directly to a Variable created **Data Table** within your model's logic, like 'Customer ID'.

Label the **Task node** 'Valid First Name' to reflect such a Test Data Assignment. When hard-coded, match the hard-coded value, like 'George'.

Describe a **Waypoint node** by its functionality, like 'Find Customers in SQL' as in a **Data Job**, or 'Click Next Button' for an **Automation reference.**

Node usage

Task nodes are specific for Assigning Test Data, whereas Waypoints are used for both Test Automation and Test Data.

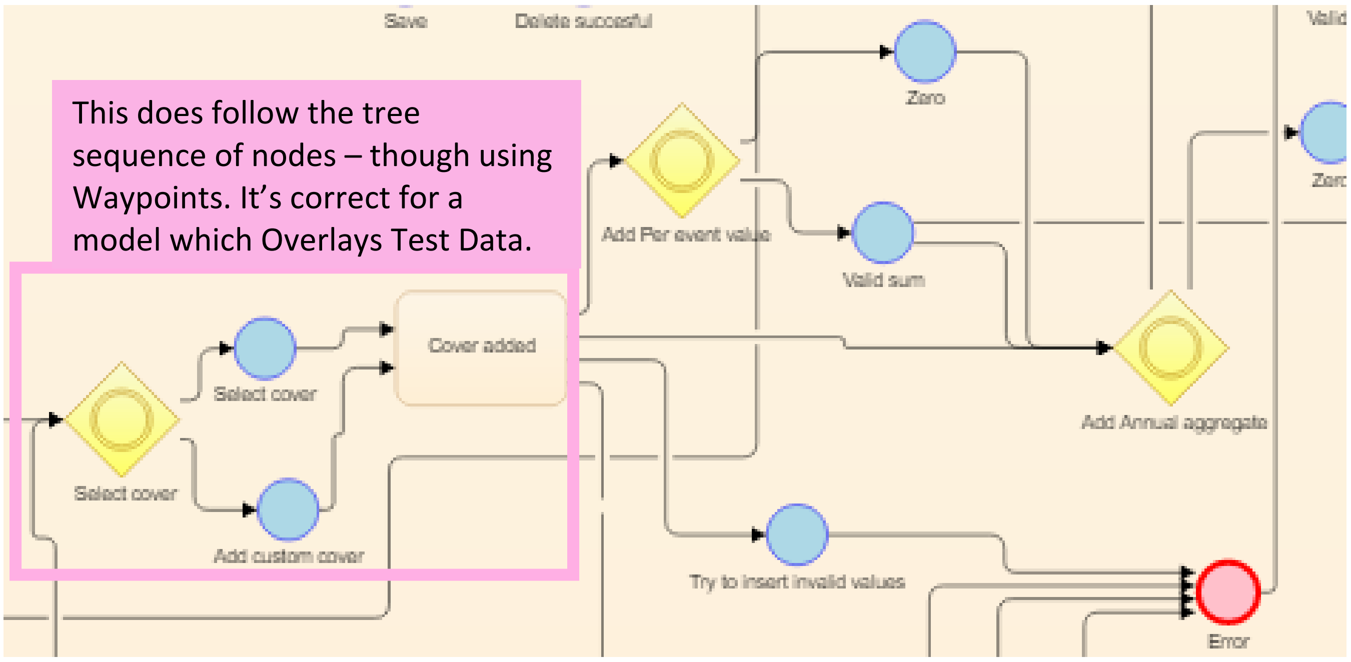
In the examples of a system under test (SUT) below Condition, Waypoint, and Task nodes are used in conjunction with each other to enhance legibility and readability of any of the models being created. This builds a representative and easy to understand model of the SUT.

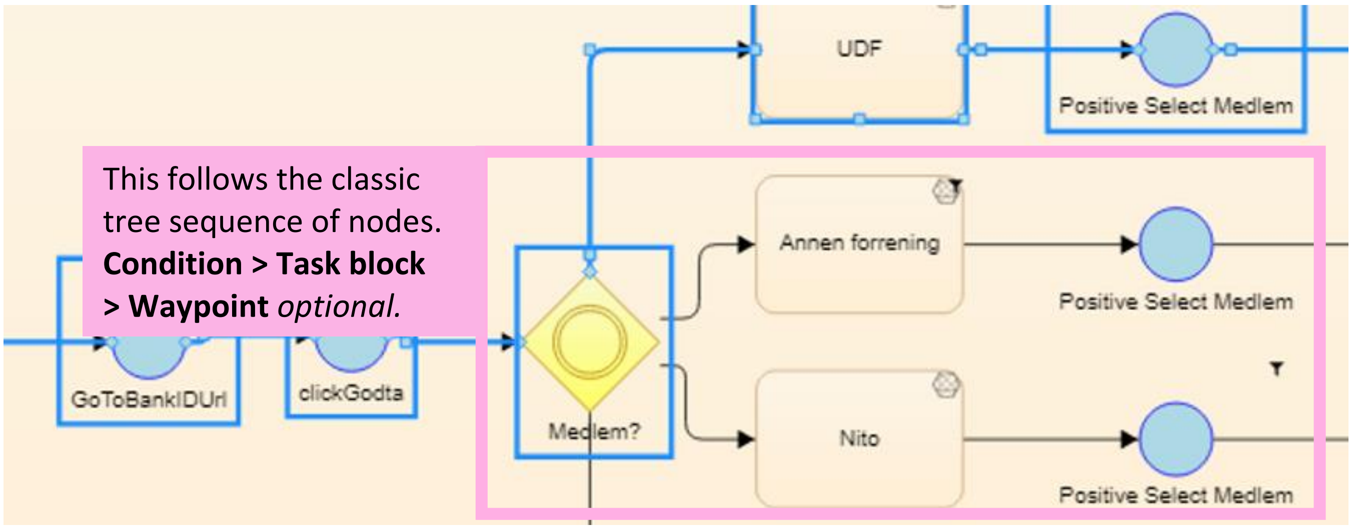
Generally, the blocks (nodes) serve according to the following definitons:

◆ **Condition –** Marks a fork a path through the a SUT's logic or where there is a data variation.

▭ **Task:** An activity performed either by a user or internally by the system, or for overlaying Test Data Assignments.

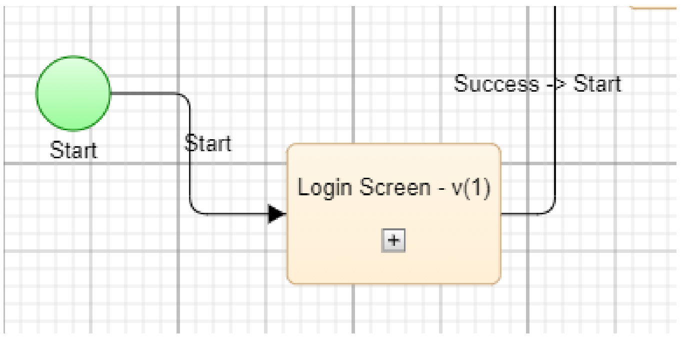
● **Waypoint:** Where Test Automation is overlaid ontop of a model.


This does follow the tree sequence of nodes – though using Waypoints. It's correct for a model which Overlays Test Data.


This follows the classic tree sequence of nodes. **Condition > Task block > Waypoint** *optional.*

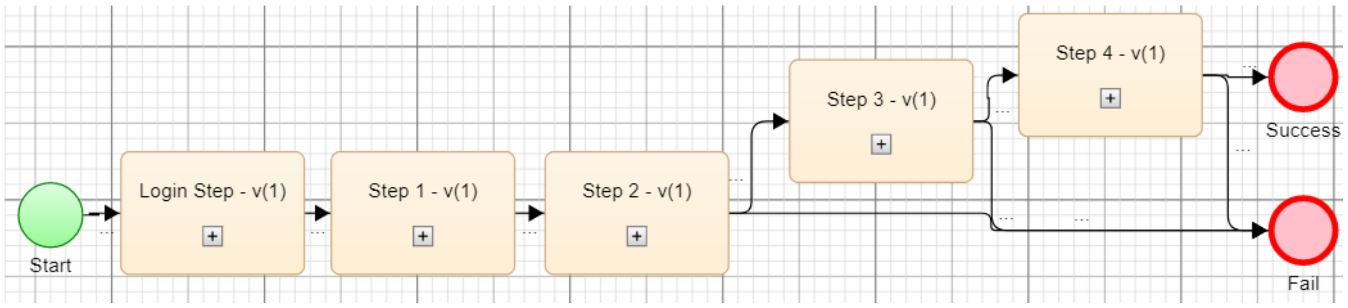
2 Master model Subflows

- Componentize Master models using Subflows and identify generic components to improve visibility/workflow
- Benefits of using Subflows in end-to-end Scenarios
- Subflow Properties pane to Expose Variables In and/or Out and using Parameters effectively



Componentize Master models using Subflows

In the example above, a Login Screen has already been modelled out. Then in a master model it has been Imported as a Subflow. A Login Screen model is ideal to use as a Subflow because it's a process that's likely to be used in many other models, and in almost every Test Case a user needs to login to a SUT.



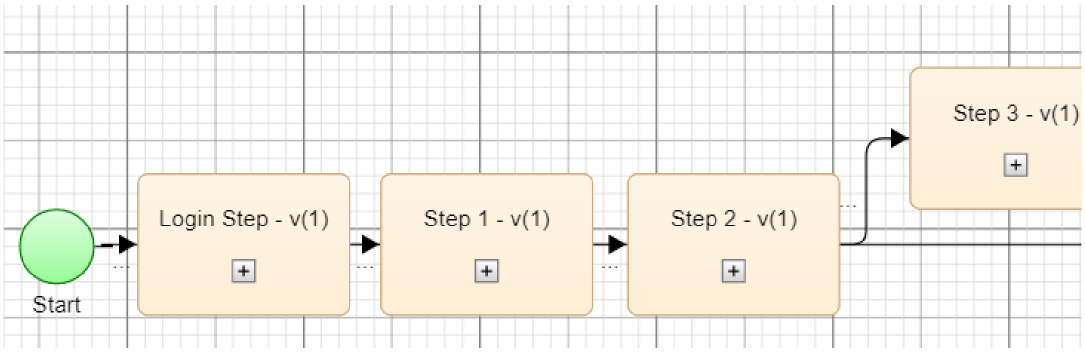
Identify generic components to improve visibility and workflow

Using the example of the Login Screen above, this model starts with a Subflow of Login Step. Then four high-level steps, as Subflow models, have been imported into a Master model. Doing this means all of that Business Logic and Rules have been captured from within each of the Subflows.

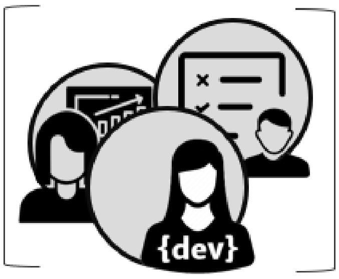
Benefits of using Subflows as components of end-to-end Scenarios

Enhances model readability.

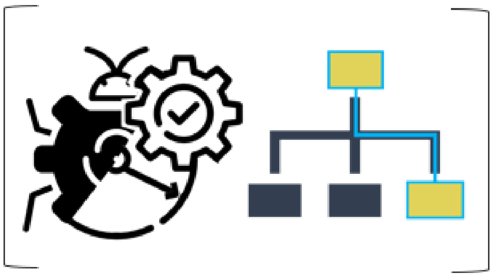
Easily reused as required in larger end-to-end Master models.



Increases the speed at which quality new models are created.

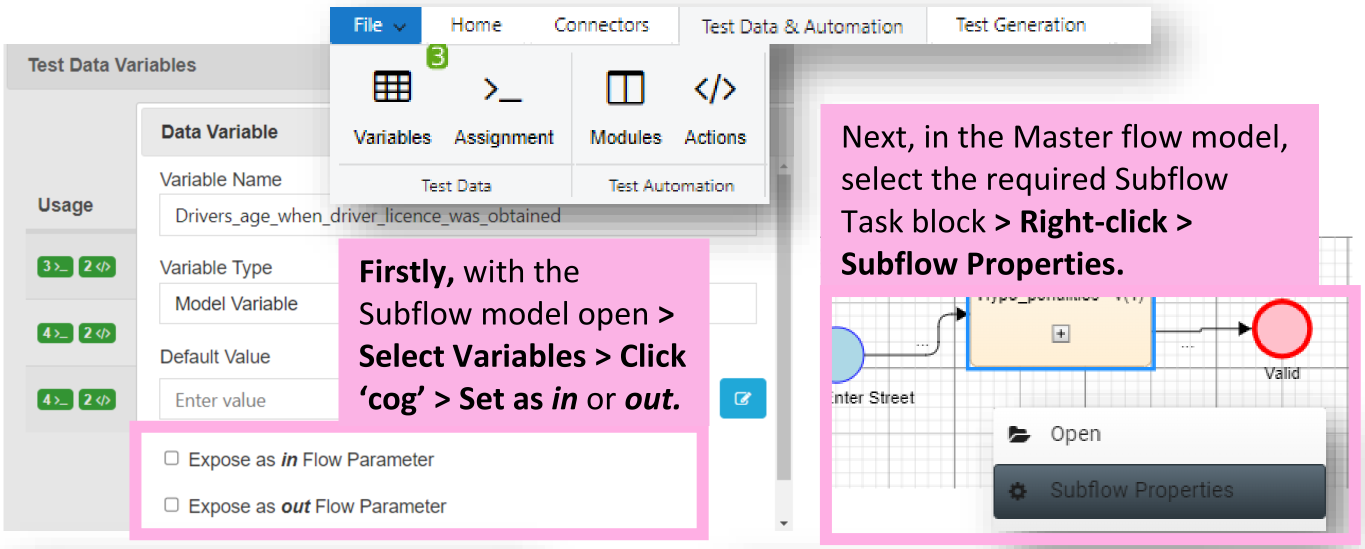


Encourages collaborative effort

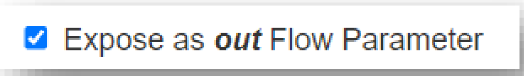


Changes/issues easily updated whilst workflow continues

Subflow Properties pane to Expose Variables In and/or Out and using Flow Parameters effectively


Firstly, with the Subflow model open > **Select Variables > Click 'cog' > Set as *in*** or ***out*.**

Next, in the Master flow model, select the required Subflow Task block **> Right-click > Subflow Properties.**

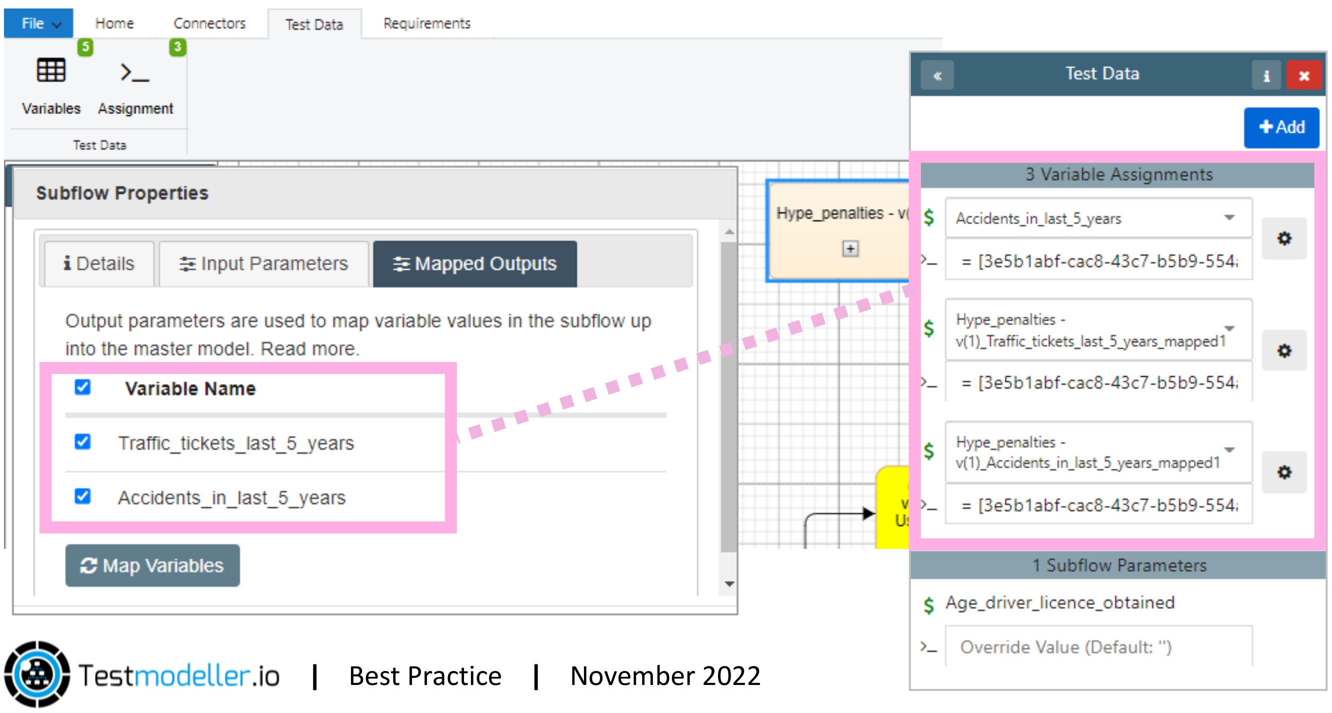

☑ Expose as ***out*** Flow Parameter

☑ Expose as ***in*** Flow Parameter

Expose out This also pushes Variables up into the Master model, from which you can **quickly duplicate any of the Exposed Variables and Assign unique Values**. This helps manipulate the Subflow behaves in the Master model.

Expose in This pushes a Variable up into a Master model to be Assigned a new Value via the Master model's Test Data pane. Smart, as it protects the integrity of the Subflow Parameters for another user to pick up, and **is useful for more involved Rule-Based Generation** with Subflows. **Reference the new Assignment using the = [Variable_Name].**

Test Data pane in the Master model reflects any updated Subflow Parameters


Output parameters are used to map variable values in the subflow up into the master model. Read more.

☑	Variable Name
☑	Traffic_tickets_last_5_years
☑	Accidents_in_last_5_years

3 Variable Assignments

- $ Accidents_in_last_5_years = [3e5b1abf-cac8-43c7-b5b9-554
- $ Hype_penalties - v(1)_Traffic_tickets_last_5_years_mapped1 = [3e5b1abf-cac8-43c7-b5b9-554
- $ Hype_penalties - v(1)_Accidents_in_last_5_years_mapped1 = [3e5b1abf-cac8-43c7-b5b9-554

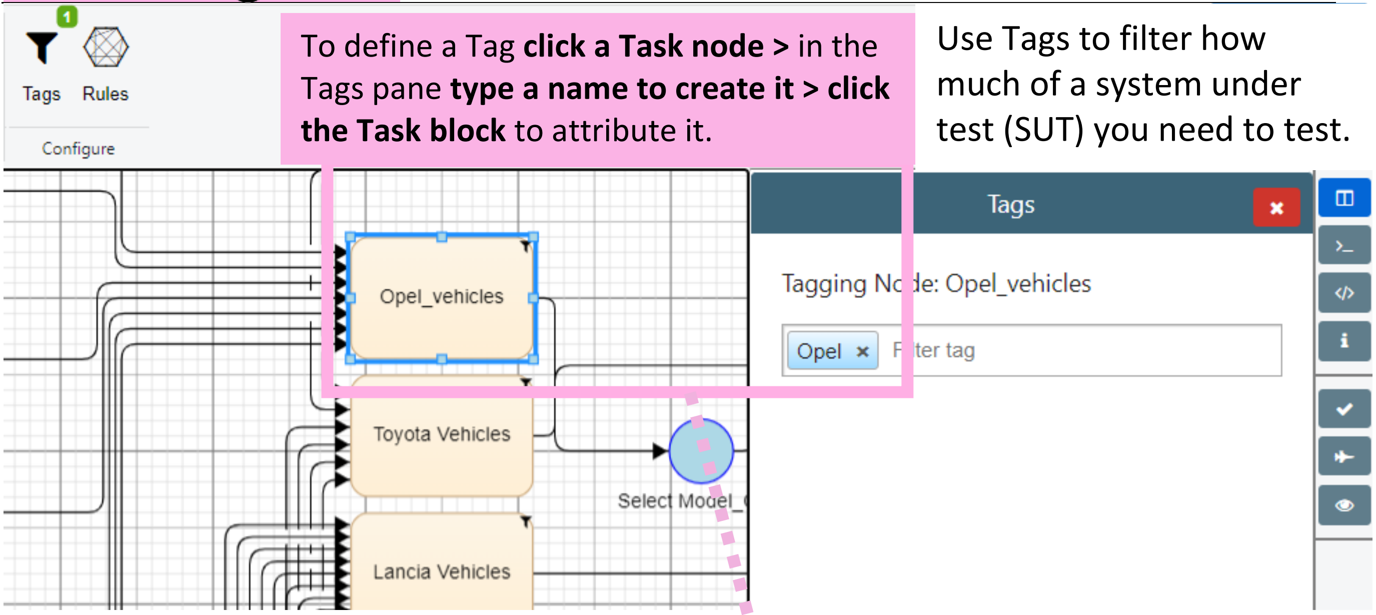
1 Subflow Parameters

- $ Age_driver_licence_obtained — Override Value (Default: '')

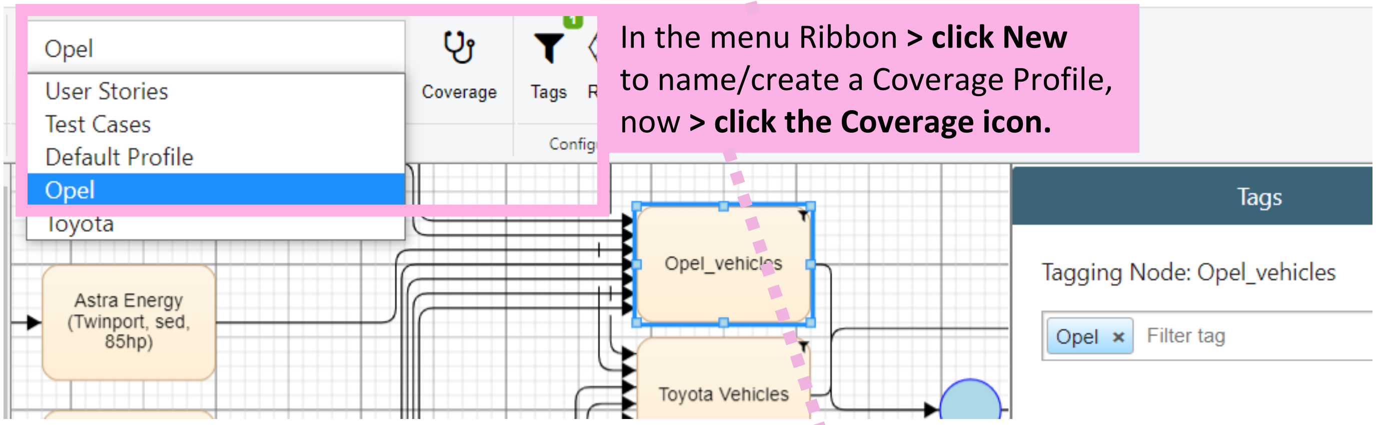
3 Tags & Coverage

- Use Tags to filter Test Cases and align Test Coverage for Smoke, Functionality or Regression Testing to verify new, critical or heavily-reused components

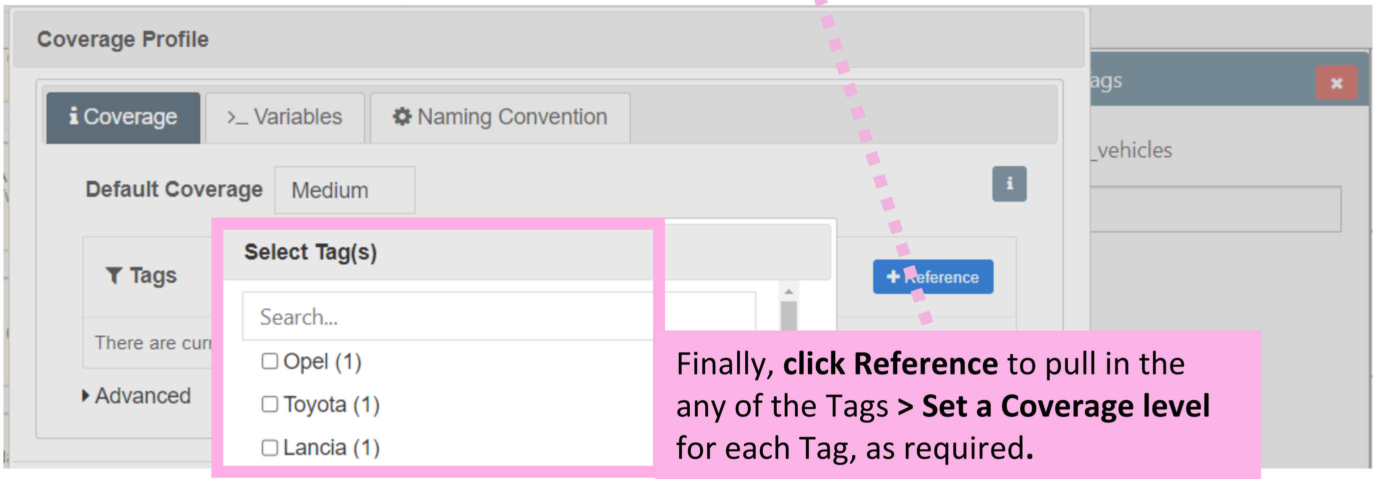
Use Tags to filter how much of a system under test (SUT) you need to test.



To define a Tag **click a Task node >** in the Tags pane **type a name to create it > click the Task block** to attribute it.



In the menu Ribbon **> click New** to name/create a Coverage Profile, now **> click the Coverage icon.**



Finally, **click Reference** to pull in the any of the Tags **> Set a Coverage level** for each Tag, as required.

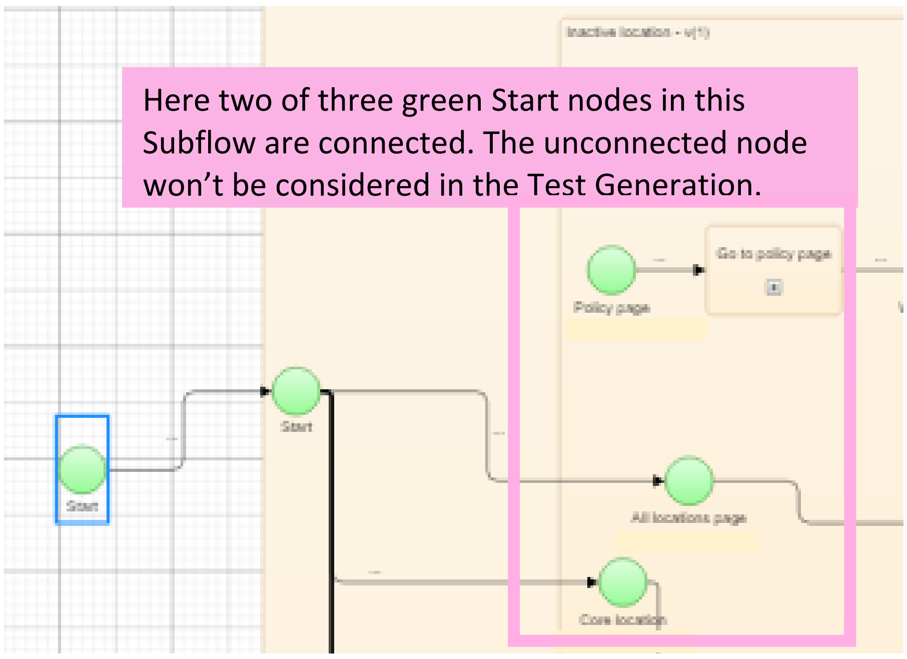
A low Coverage level creates a compact Test Suite of core functionality, which is ideal for **Smoke Testing.**

For New Feature and **Functionality Testing** you can **set Coverage to high or exhaustive;** by contrast to **low Coverage** for Existing functionality, ideal for **Regression Testing.**

4 Test Generation fully visualized models OR Logic Constraint Variables?

- Use Start & End nodes to limit Test Generation
- Setting up
- Drive Rule-based Test Generation | Variables

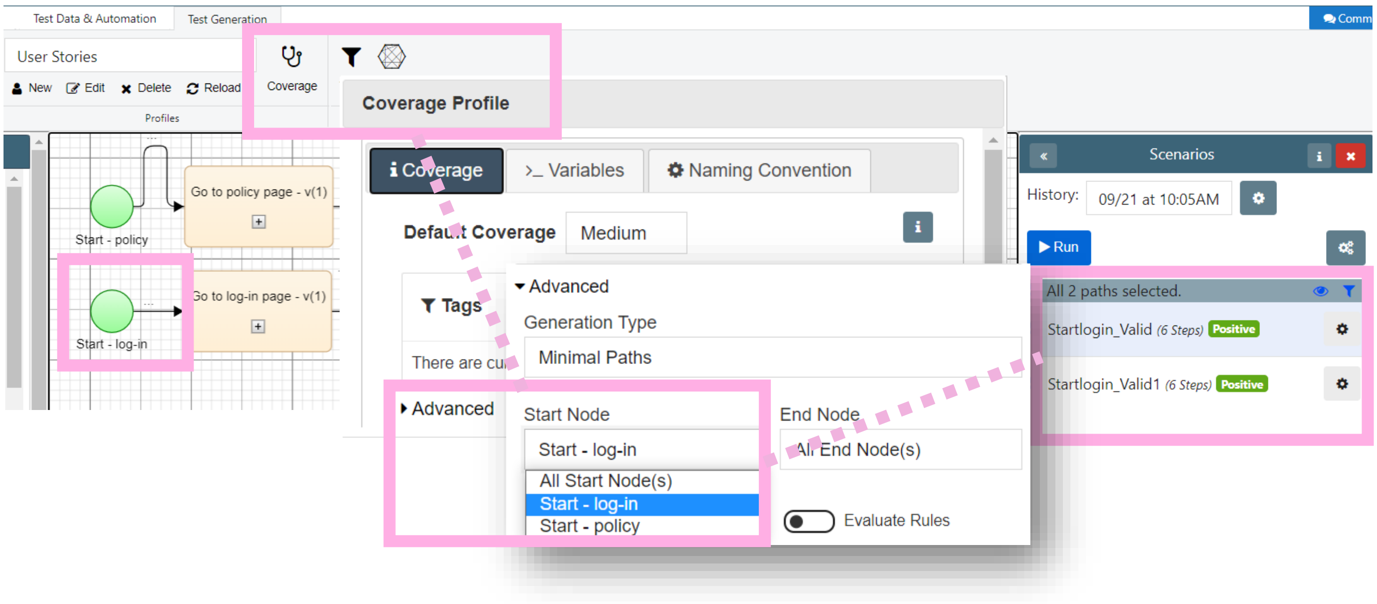
Use Start & End nodes to limit Test Generation


Here two of three green Start nodes in this Subflow are connected. The unconnected node won't be considered in the Test Generation.
Inactive location - v(1)
Go to policy page
Policy page
Start
Start
All locations page
Core location

Using Start & End nodes, particularly to connect a Subflow in a larger end-to-end Scenario/Master Model, is a good way to constrain model Logic avoiding the use of Rules and Tags.

In other situations, Business Rules might be reflected in these Constraints to restrict impossible combinations.

Setting up



Default Coverage: Medium

Advanced

Generation Type: Minimal Paths

Start Node: Start - log-in
- All Start Node(s)
- Start - log-in
- Start - policy

End Node: All End Node(s)

Evaluate Rules

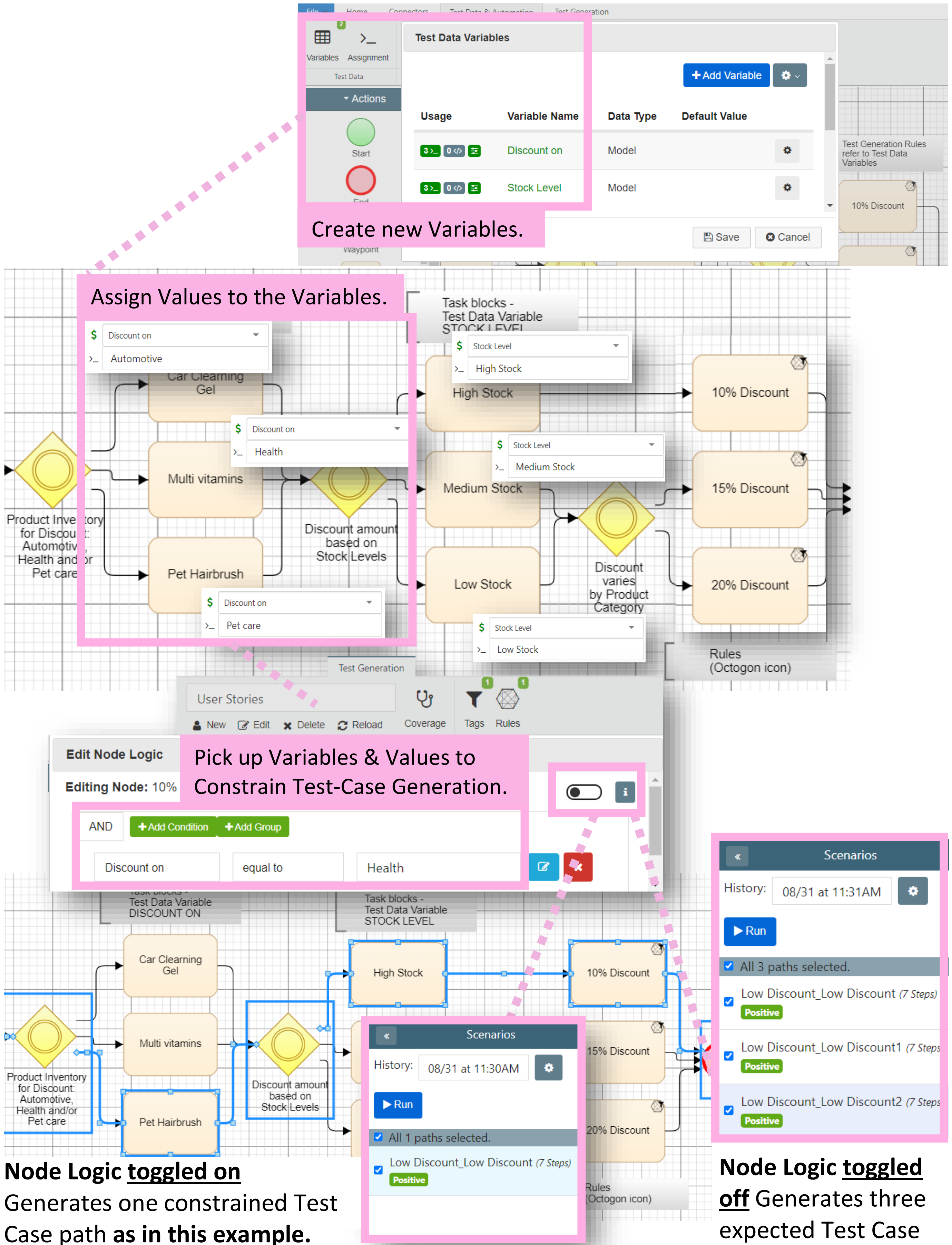
History: 09/21 at 10:05AM

All 2 paths selected.

Startlogin_Valid (6 Steps) Positive

Startlogin_Valid1 (6 Steps) Positive

Drive Rule-based Test Generation | Variables


Create new Variables.
Assign Values to the Variables.
Pick up Variables & Values to Constrain Test-Case Generation.
Variables
Assignment
Test Data
Actions
Start
End
Waypoint
+Add Variable
Save
Cancel
Test Generation Rules refer to Test Data Variables
Task blocks - Test Data Variable STOCK LEVEL
Task blocks - Test Data Variable DISCOUNT ON
Car Clearning Gel
Multi vitamins
Pet Hairbrush
Product Inventory for Discount: Automotive, Health and/or Pet care
Discount amount based on Stock Levels
High Stock
Medium Stock
Low Stock
Discount varies by Product Category
10% Discount
15% Discount
20% Discount
Rules (Octogon icon)
Test Generation
User Stories
New
Edit
Delete
Reload
Coverage
Tags
Rules
Edit Node Logic
Editing Node: 10%
AND
+Add Condition
+Add Group
Scenarios
Run

Test Data Variables

Usage	Variable Name	Data Type	Default Value
3 0	Discount on	Model	
3 0	Stock Level	Model	

Discount on — Automotive
Discount on — Health
Discount on — Pet care
Stock Level — High Stock
Stock Level — Medium Stock
Stock Level — Low Stock

Discount on | equal to | Health

History: 08/31 at 11:30AM
All 1 paths selected.
Low Discount_Low Discount (7 Steps) Positive

History: 08/31 at 11:31AM
All 3 paths selected.
Low Discount_Low Discount (7 Steps) Positive
Low Discount_Low Discount1 (7 Steps) Positive
Low Discount_Low Discount2 (7 Steps) Positive

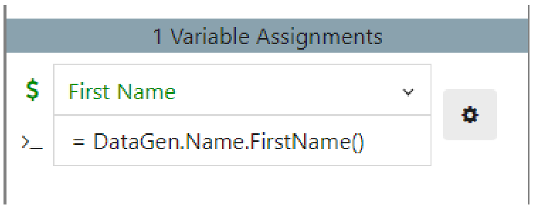
Node Logic <u>toggled on</u>
Generates one constrained Test Case path **as in this example.**

Node Logic <u>toggled off</u> Generates three expected Test Case paths.

5 Test Data

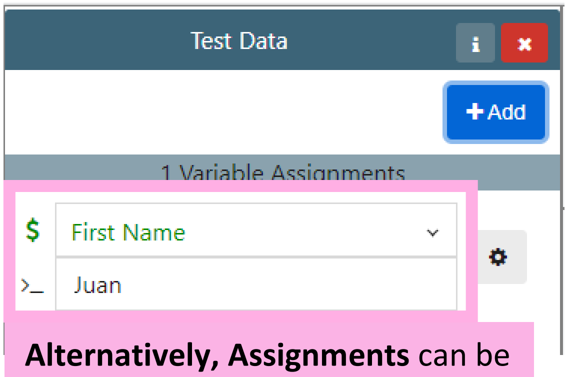
○ Hard-coded and Dynamic Assignments

A Dynamic Value Assignment such as those available in Curiosity's out-of-the-box **Synthetic Data Generation** functions.


1 Variable Assignments

$ First Name

>_ = DataGen.Name.FirstName()


Test Data

+Add

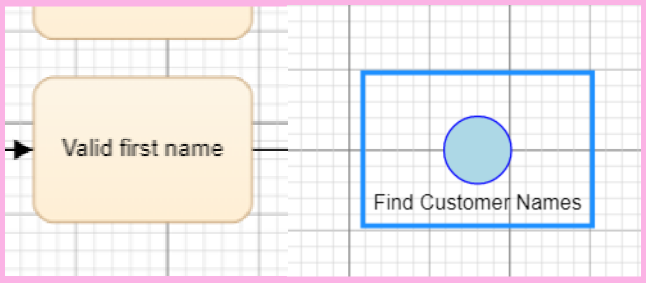
1 Variable Assignments

$ First Name

>_ Juan

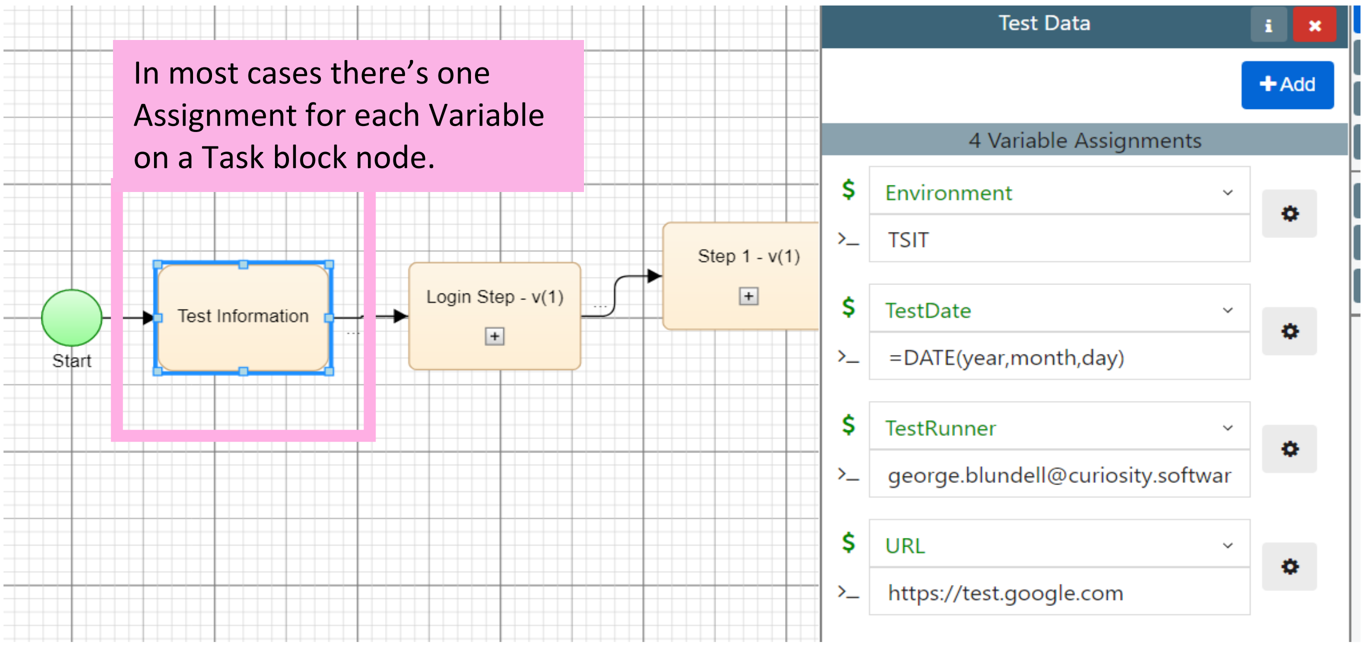
Alternatively, Assignments can be hard-coded as **Static Values.**

Practically, a Test Data Waypoint follows a Task block Assignment.



Dynamic Test Data: **'Finds and Makes'** come from back-end databases and are embedded onto Waypoints and dragged into the model along with the node.

In most cases there's one Assignment for each Variable on a Task block node.



Test Data

+Add

4 Variable Assignments

$ Environment

>_ TSIT

$ TestDate

>_ =DATE(year,month,day)

$ TestRunner

>_ george.blundell@curiosity.softwar

$ URL

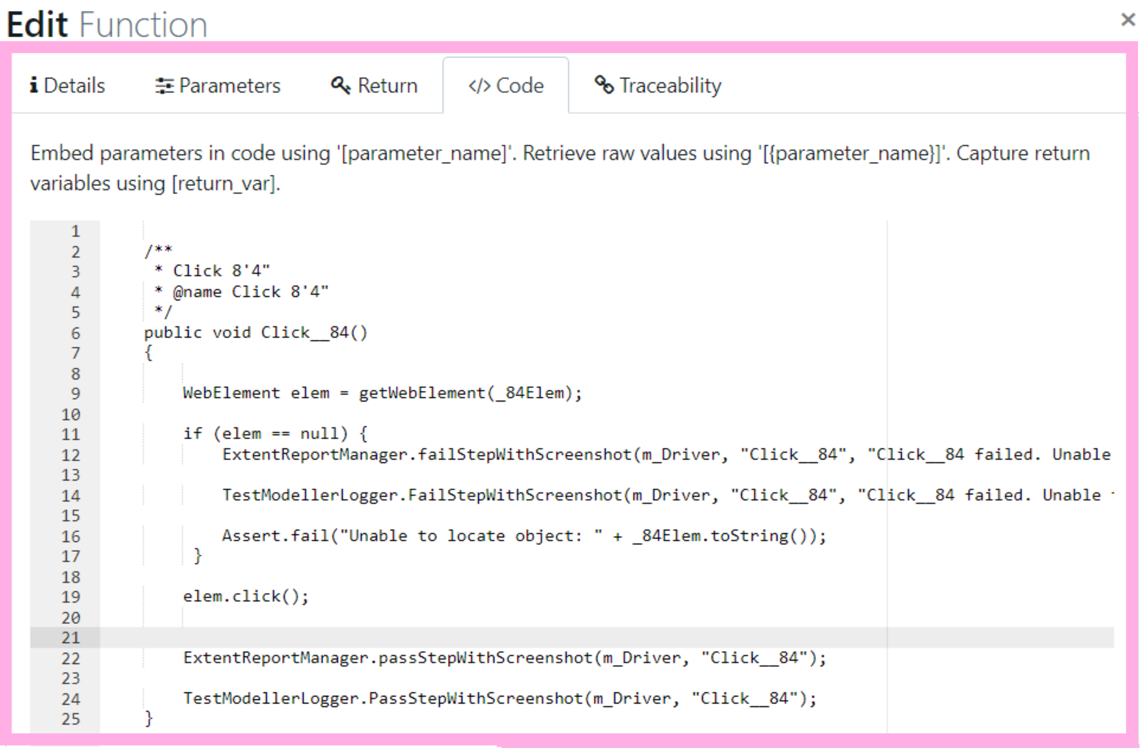
>_ https://test.google.com

Though, you may have a need for multiple Assignments of multiple Variables on one Task block, that's only if you don't need to model out data variations.

In some cases, as for Test Info or Environment Specification, you may want to set multiple Data Assignments to one block.

6 Test Automation

- Custom Functions: embedded and code snippets
- Reusable & Dynamic libraries

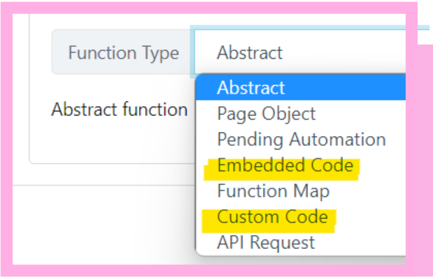


Embed parameters in code using '[parameter_name]'. Retrieve raw values using '[{parameter_name}]'. Capture return variables using [return_var].

```

/**
 * Click 8'4"
 * @name Click 8'4"
 */
public void Click__84()
{

    WebElement elem = getWebElement(_84Elem);

    if (elem == null) {
        ExtentReportManager.failStepWithScreenshot(m_Driver, "Click__84", "Click__84 failed. Unable

        TestModellerLogger.FailStepWithScreenshot(m_Driver, "Click__84", "Click__84 failed. Unable

        Assert.fail("Unable to locate object: " + _84Elem.toString());
     }

    elem.click();



    ExtentReportManager.passStepWithScreenshot(m_Driver, "Click__84");

    TestModellerLogger.PassStepWithScreenshot(m_Driver, "Click__84");
}
```

This is particularly useful **in the case of Cypress**, where you could Embed a **Custom Function** into the Script, without needing to set up a corresponding Function getting called in the Script.

Custom Functions should be either set to **Embedded Code** or **Custom Code**. Embedded code literally embeds the code snippet into the Automated Generated Test Cases.



Alternatively, if you are using a **Java Selenium framework**, it may be more appropriate to use **Custom Code.**

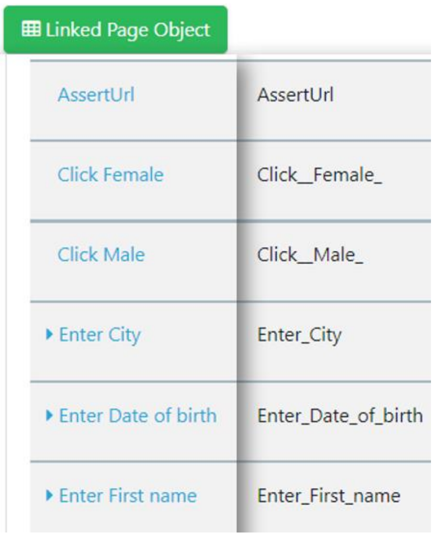
This sets up a **Function within the Page Object File** that can be called by the Test Script.

Dynamic maintenance

Module Collections remove ambiguous manual steps involved in the maintenance of Automation Code.

Meaning, any changes to the Dynamic Automation Function is propagated throughout the Workspace and applied upwards to models using it.

So you need only make a change to the Dynamic library once.



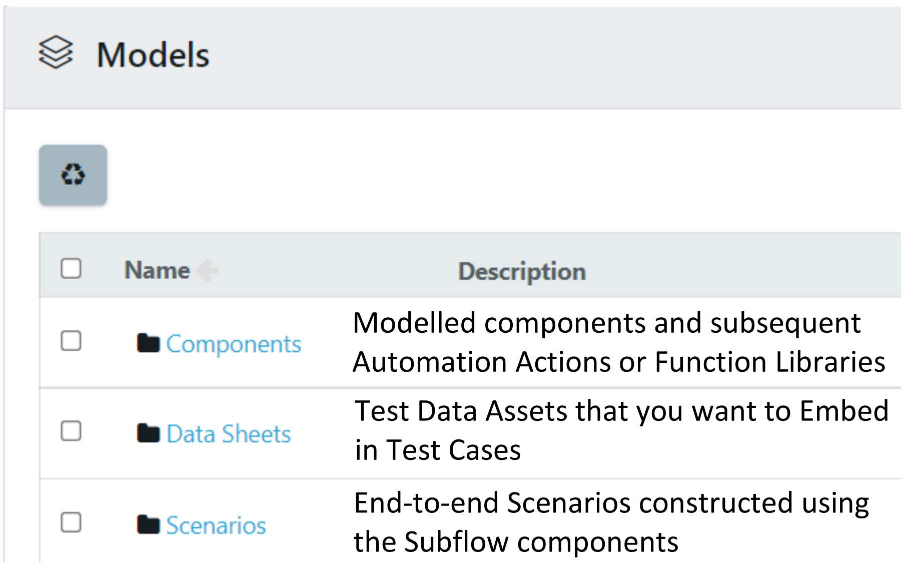
AssertUrl	AssertUrl
Click Female	Click_Female_
Click Male	Click_Male_
Enter City	Enter_City
Enter Date of birth	Enter_Date_of_birth
Enter First name	Enter_First_name

7 File management

- Project file structure by default
- Flexible file structure
- Multiple projects in the Workspace

Project file structure by default

Although there is no enforced project/file structure in Test Modeller, automatically generated folders are listed here. To move assets around between folders click the radio button of the assets, then choose move from the review icon.



Models

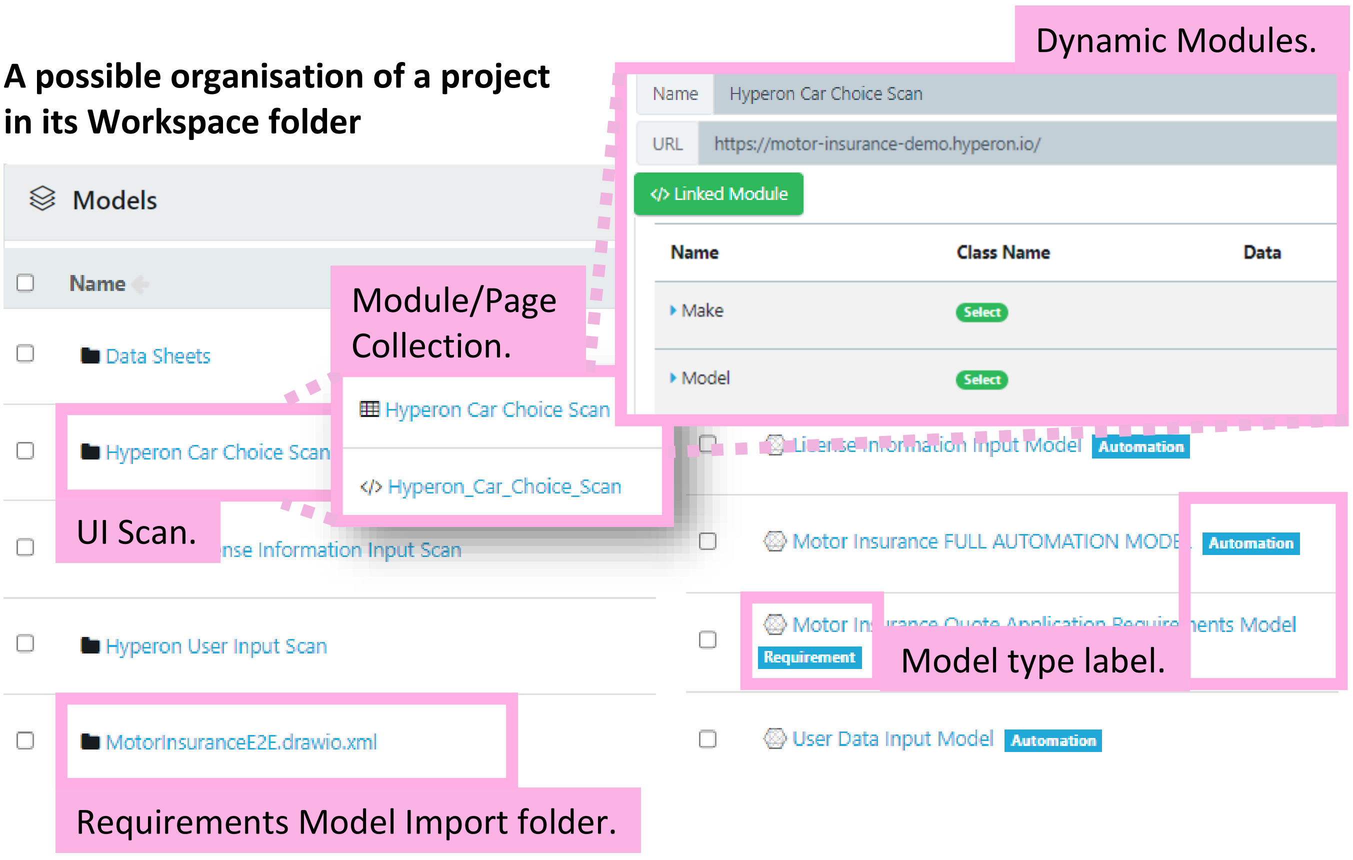
Name	Description
Components	Modelled components and subsequent Automation Actions or Function Libraries
Data Sheets	Test Data Assets that you want to Embed in Test Cases
Scenarios	End-to-end Scenarios constructed using the Subflow components

Flexible file structure

Should you wish to save models & accompanying Modules across two folders, this presents no problem.

Additionally, grouping assets of models and Modules of the system under tested (SUT) together is also fine.

A possible organisation of a project in its Workspace folder



Models

Name
Data Sheets
Hyperon Car Choice Scan
nse Information Input Scan
Hyperon User Input Scan
MotorInsuranceE2E.drawio.xml

Hyperon Car Choice Scan
Hyperon_Car_Choice_Scan

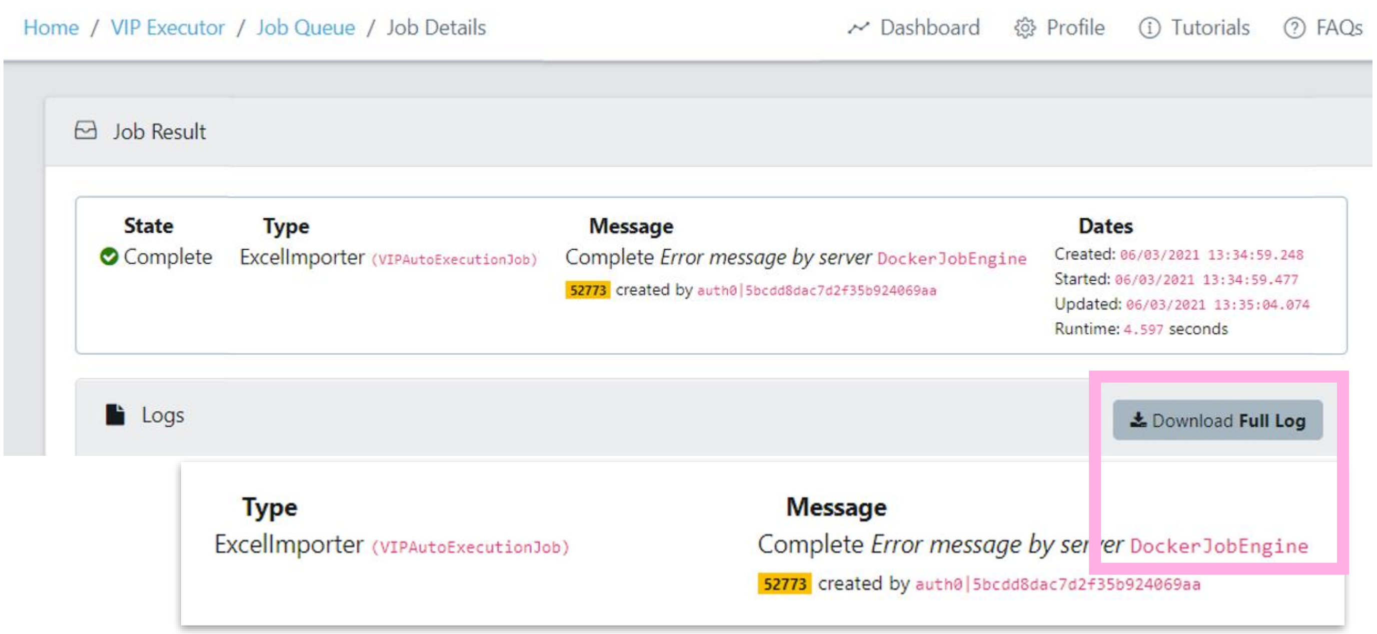
Name: Hyperon Car Choice Scan
URL: https://motor-insurance-demo.hyperon.io/
Linked Module

Name	Class Name	Data
Make	Select	
Model	Select	

License Information Input Model Automation
Motor Insurance FULL AUTOMATION MODE Automation
Motor In rance Quote Application Require ents Model Requirement
User Data Input Model Automation

8 Debugging Logs

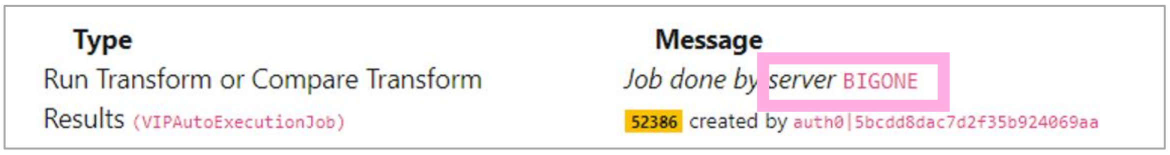
Docker Job Engine
Most standard jobs including Imports, Test Generation and Exports within Test Modeller are executed by the **Docker Job Engine**. Download and view **Logs** by clicking 'Download Full Log' button.



State: Complete
Type: ExcelImporter (VIPAutoExecutionJob)
Message: Complete *Error message by server* DockerJobEngine
52773 created by auth0|5bcdd8dac7d2f35b924069aa
Dates:
Created: 06/03/2021 13:34:59.248
Started: 06/03/2021 13:34:59.477
Updated: 06/03/2021 13:35:04.074
Runtime: 4.597 seconds

Type: ExcelImporter (VIPAutoExecutionJob)
Message: Complete *Error message by server* DockerJobEngine
52773 created by auth0|5bcdd8dac7d2f35b924069aa

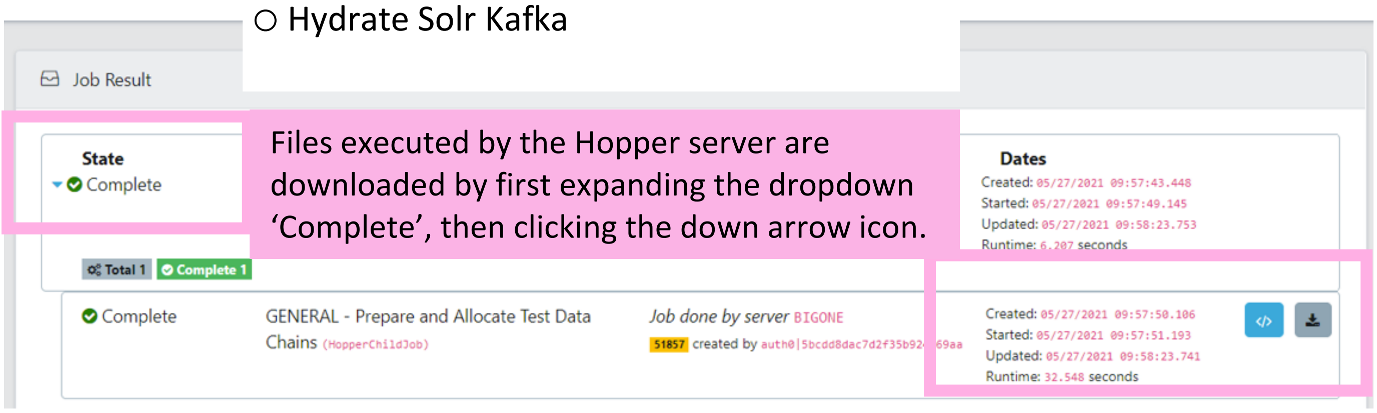
To Download and view Logs on a **standard server** also click 'Download Full Log' button. In this case the standard server is called BIGONE.



Type: Run Transform or Compare Transform Results (VIPAutoExecutionJob)
Message: *Job done by server* BIGONE
52386 created by auth0|5bcdd8dac7d2f35b924069aa

Hopper server | **Jobs run on the Hopper server include:**

○ Prepare and Allocate Test Data Chains
○ Find Test Data Chains
○ Hydrate Solr Kafka



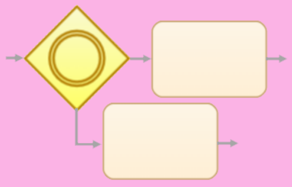
Files executed by the Hopper server are downloaded by first expanding the dropdown 'Complete', then clicking the down arrow icon.

State: Complete
Dates:
Created: 05/27/2021 09:57:43.448
Started: 05/27/2021 09:57:49.145
Updated: 05/27/2021 09:58:23.753
Runtime: 6.207 seconds

Complete | GENERAL - Prepare and Allocate Test Data Chains (HopperChildJob) | *Job done by server* BIGONE
51857 created by auth0|5bcdd8dac7d2f35b92 69aa
Created: 05/27/2021 09:57:50.106
Started: 05/27/2021 09:57:51.193
Updated: 05/27/2021 09:58:23.741
Runtime: 32.548 seconds

9 Appendix Coffee-time clips: Art of Modelling series

1 Decision Trees



Baseline Grammar (2mins 08seconds)
https://www.youtube.com/watch?v=TgqWy790uGw&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=2

Modelling an Existing UI (2mins 25seconds)
https://www.youtube.com/watch?v=NaCClnajb44&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=4

Scope and Articulate Flow (2mins 30Seconds)
https://www.youtube.com/watch?v=bzZkgDlTojs&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=7

2 Master model Subflows

Align Collaborative Effort (3mins 10seconds)
https://www.youtube.com/watch?v=ucleUZidQaE&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=10

3 Tags & Coverage

Specify your Criteria (2mins 08seconds)
https://www.youtube.com/watch?v=KxJJla-V2Ek&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=11

Define the Test Objective (2mins 05seconds)
https://www.youtube.com/watch?v=pBBMhTf4CqY&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=12

4 Test Generation fully visualized models OR Logic Constraint Variables?

Be Mindful of Decision Gates (Trees) (2mins 10seconds)
https://www.youtube.com/watch?v=QkaY70POa5Y&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=8

Overlay Logic to Reduce Repetition (2mins 50seconds)
https://www.youtube.com/watch?v=lSaOp4uiirQ&list=PLd_AqXM4vM-Bihc9Qx3Sl3pUCsC6S8XoV&index=9

Compiled September 2022
Direct contact paul.wright@curiosity.software | Customer Success Engineer
Also github.com/CuriositySoftwareIreland/knowledge_and_support/discussions
and knowledge.curiositysoftware.ie