Automation Specialist | Level One

LESSON TRANSCRIPT

SURVEY

Tricentis Automation Specialist | Level 1

- Version 2019_02
- Designed to be used with Tricentis **Tosca version 12.x**

Lesson Transcript

This lesson Transcript provides the scripts used during the lesson videos for the Tricentis Automation Specialist | Level 1training

Legal Notice

Tricentis GmbH
Leonard-Bernstein-Straße 10
1220 Vienna
Austria

Tel.: +43 (1) 263 24 09
Fax: +43 (1) 263 24 09-15
Email: academy@tricentis.com

TABLE OF CONTENTS

PREFACE

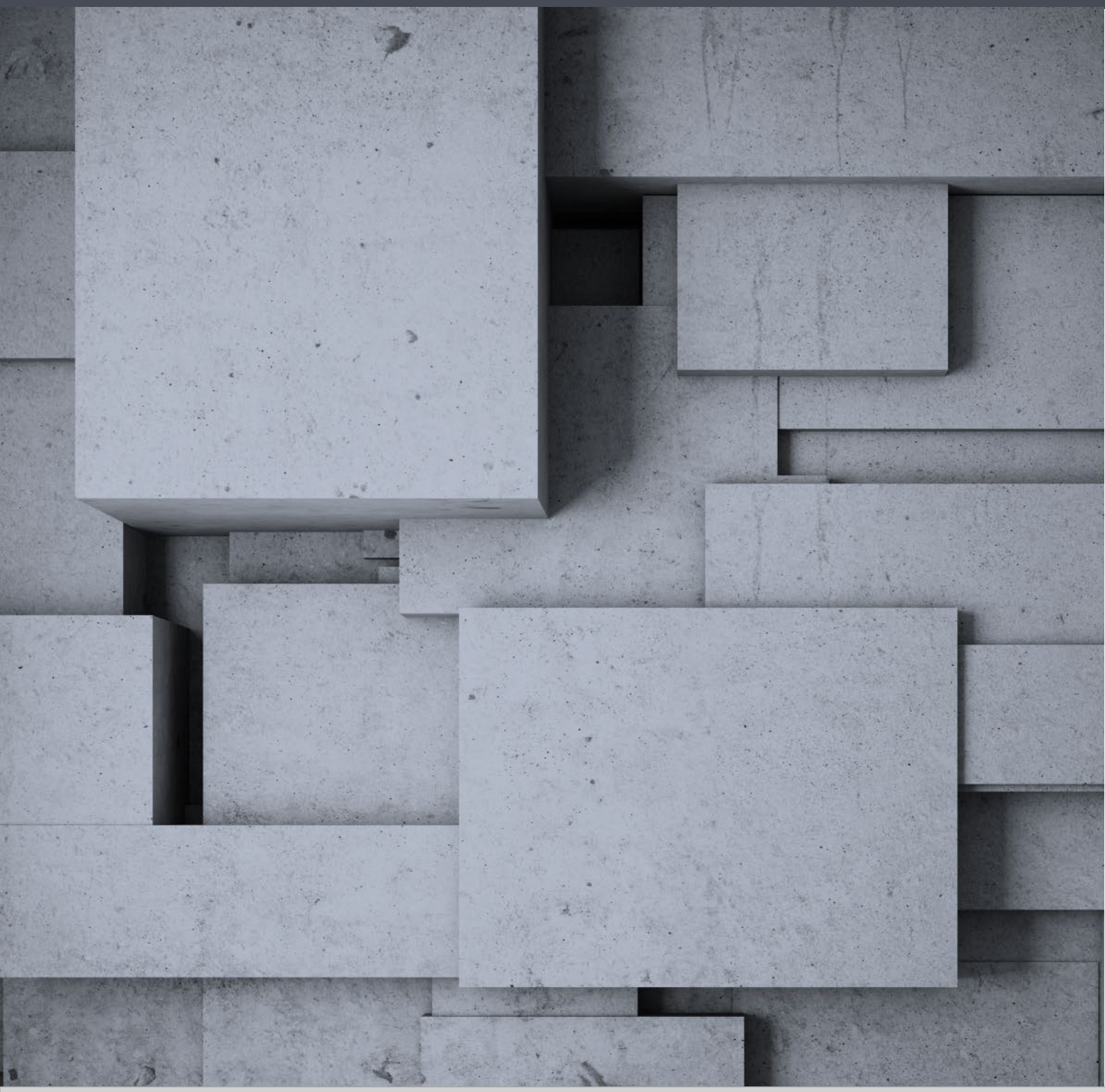
About this workbook

This transcript is specifically designed to supplement training of the Tricentis Automation Specialist | Level 1

The transcript is divided into the 17 Lesson sections and supplies the script used for the lesson videos.

This transcript is not aiming to be a complete manual.

TRICENTIS



GETTING STARTED

INTRODUCTION

Course Structure

Getting Started - The System Under Test

- *Tour the Demo Web Shop and learn the general storyline of the testing project.*

Getting Started - Introduction to Tosca

- *Tour the default view of Tosca and create a new workspace using the Base Subset as a template.*

TestCase 1 Shipping Costs

- *Automate a complete TestCase and become familiar with essential Tosca navigation and automation functionalities.*

TestCase 2 Payment Process
TestCase 3 Discount Code
TestCase 4 Reorder
TestCase 5 Total Price all Orders

- *Learn additional Tosca functionalities including Libraries, Recovery Scenarios, Test Configuration Parameters, Business Parameters, Dynamic Values, Dynamic IDs, as well as Loops and Conditions.*

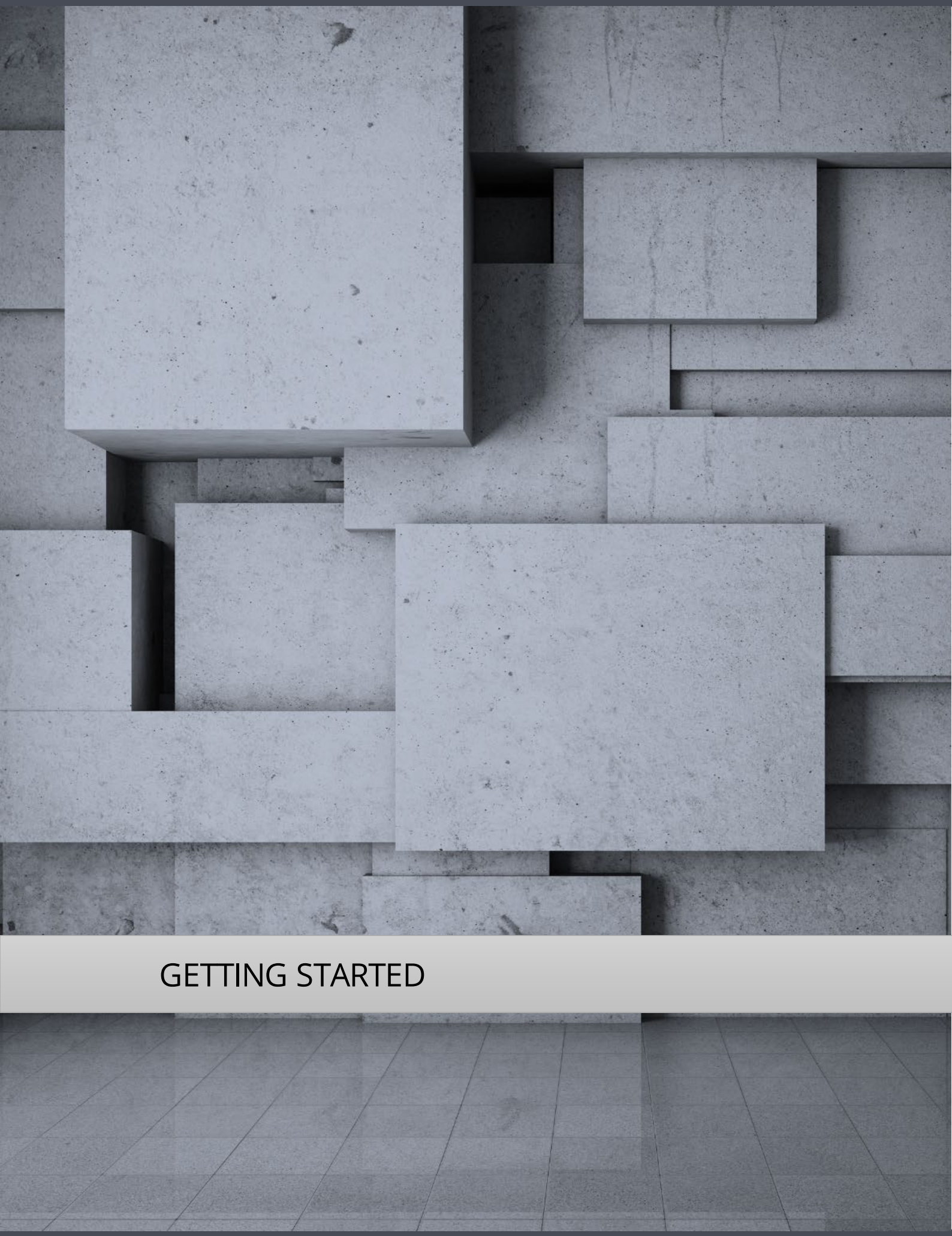
Assessment Exams

- *Test your knowledge after completing each TestCase.*

Final Exam

- *Prove your knowledge of all topics covered in the course in order to earn a certificate.*

TRICENTIS



GETTING STARTED

GETTING STARTED

The System Under Test

The Automation Specialist course will focus on the creation and execution of five test cases for a single system under test. This system under test, or SUT, is called the DemoWebShop, which is referred to as the Web Shop throughout this training. This was created exclusively for Tricentis training purposes and is held on a public website and resembles a true online shop.

Within the Web Shop, users are able to create an account, browse through items for sale, and complete the checkout process, among other things. There are numerous paths a user may take when using the Web Shop. For example, it may vary what items are being purchased, how they are shipped, which method of payment is used, and even if a discount code is being applied. You will soon learn that each of these paths can be represented for testers as **TestCases**. Within this course, five different test cases will be created and executed in Tricentis Tosca to test the SUT using various paths.

The test cases have been predetermined based on the most important requirements of the Web Shop. These TestCases are named shipping costs, payment process, discount code, reorder, and total of all orders. Each of these refer to a requirement of the Web Shop that is required to run successfully. In testing projects, the most important requirements are tested first.

By the end of the course, students will have the skills necessary to create and execute fully automated Html test cases.

In order to test the Web Shop, you, as the tester, will need to become a registered user and be familiar with the Web Shop. Once you complete the following exercise to register, please tour the website, adding items to your cart and observing the purchase process.

Introduction to Tricentis Tosca

This course is a tool training for Tricentis Tosca. Tricentis Tosca will be used to complete each of the exercises, to ensure students gain hands-on experience while testing real world project examples.
Once you have installed Tricentis Tosca and have a valid license, it's ready to use. When you first start Tosca Commander, a window will appear for creating a new workspace. Once a **Workspace** has been set up, Tosca Commander is connected to a repository containing the Tosca artifacts. For the exercises in this course, a single-user workspace will be used. The following exercise will walk you through creating this workspace, step-by-step.

This is the default view of Tosca in a new workspace. The **menu** is seen at the top, which is made up of ribbons. Its many tabs allow you access to the corresponding functions. On the left-hand side is the **navigation pane**, which is a great place to keep an overview of your work. In the middle is the **working pane**, which is where you will focus on the individual items as you carry out your work. At the top of the working pane is a **column header**. Various columns are available and can be shown or hidden as needed. *Right click in the column header and open the column chooser to view your options*. Double click on a column name to open it. To hide columns from view, click on the column name and drag downward until an X appears and release it.

At the top, below the ribbon are the window tabs, which are used to switch between different sections. Multiple sections can be viewed in the same window, by dragging tabs toward the middle of the panes and dropping them in a **docking** destination, as seen here. This is also possible by right clicking on the section tab, where you have the options to close, close all but this, float, new vertical group, move to the next tab group.

Sections can also be opened from "*Go to"* in the menu above. Click on section to see the project root folders you can open. If you see a black arrow, this indicates more options for root folders. For example, root folders contained within component folders.

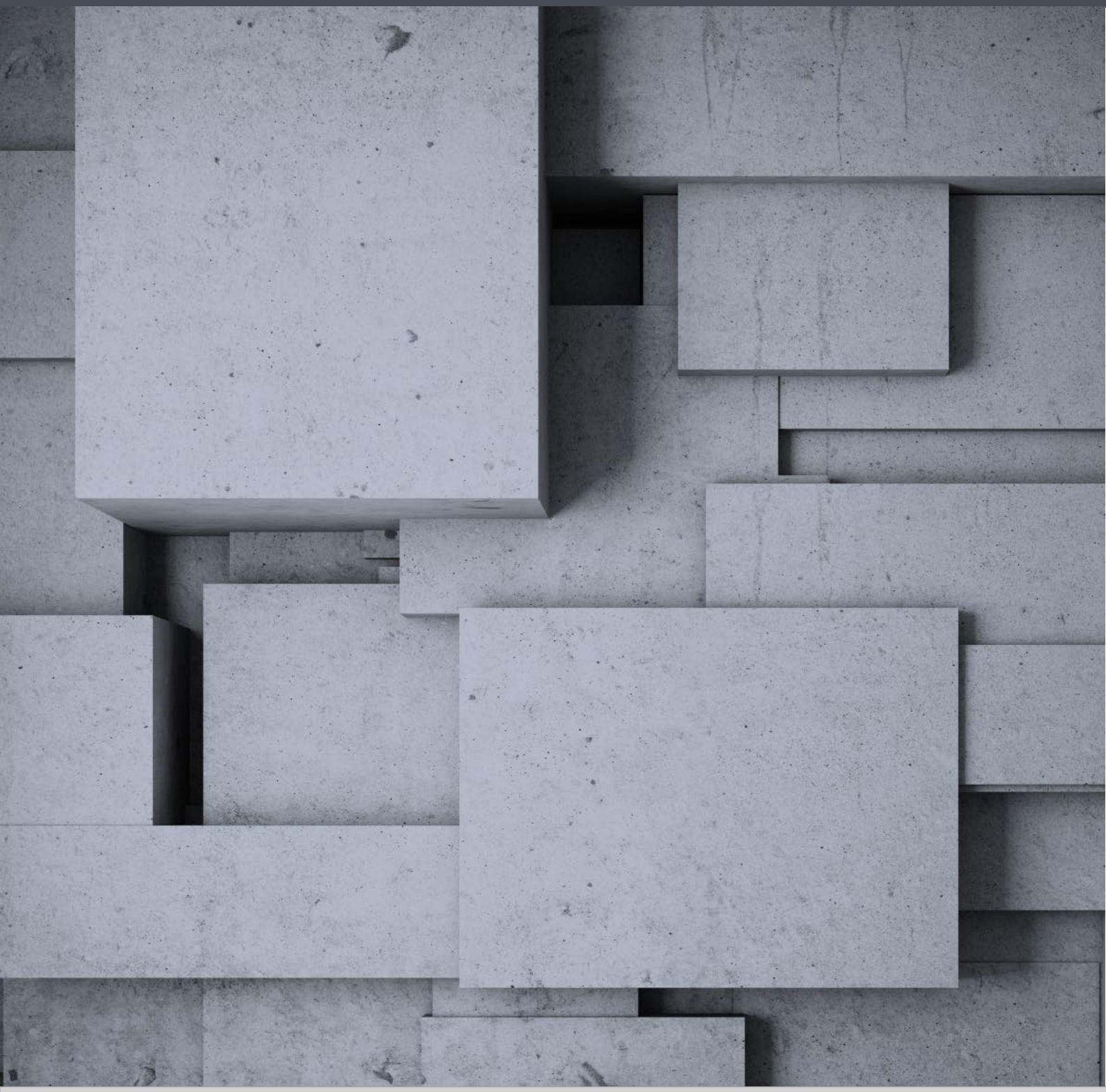
Component folders are additional organizational elements for your project. You can create these folders by right clicking on the Project root and selecting "Create component folder". Within these folders, you can add additional Tosca sections, such as TestCases, Requirements, ExecutionLists, and so on, and create objects here.

Save, **undo**, and **redo** are also possible from the menu.

While working in Tosca, it will be necessary to import and export Subsets. A **Subset** is a file containing Tosca artifacts that can be shared with other projects. When elements need to be moved from one common repository to another or between single user workspaces, a Subset must be exported from the first workspace, and imported into the second. For this training, we will only work in single user workspaces. In a later lesson, you will learn the basics of working in a multiuser environment, including checking in and out objects from a common repository.

To export a subset, you simply select the objects or folders you want to export, and select "Export subset". When you want to import a Tosca subset or .tce file, simply select the import option. How the objects are imported into your workspace will be determined by how they were exported. It may be helpful to create a component folder when importing subsets, so that it is easier to locate the new objects in your workspace.

You can also use a Subset as a template for a workspace. When creating a new workspace, select *Use Workspace Template*. Then simply browse your machine for the file you want to use. The elements will be imported to the new workspace just as they had been originally exported.



TESTCASE 1

TESTCASE ONE

Lesson 1 · Creation of a Module

Overview

In automation, an application is being steered. In order for Tosca to know where to steer the application automatically, testers must first tell Tosca where to go. This requires the use of both technical and business information. Technical information being, the programming details within the System Under Test (SUT), and business information being relevant processes and workflows as well as corresponding business data. For example, in our SUT, the Demo Web Shop, the business information refers to credit card information, items being purchased etc.

Practical Usage

As we run our 5 test cases, Tosca will be steering the Web Shop through its pages, including the "Login" screen, "Shopping Cart", "Shipping" page, etc.

Using Tosca XScan, the pages will be scanned, which collects the technical information. This information is saved in elements called Modules. Modules can be reused again and again when steering a screen or part of a screen. For example, one user may use the Web Shop to purchase blue jeans, while another purchases a computer. Modules for the log in, Shopping cart, payment, and shipping pages could be reused in both of these cases since the technical information stays the same for those pages.

In addition to Modules being used in multiple TestCases, they can also be used multiple times within the same TestCase. For example, if a menu at the top of a website stays the same for all pages of the site, one module containing all the controls in that menu can be used every time the menu items are used, regardless of the website page.

Demonstration

Let's get acquainted with the XScan before you create your first Module. Tosca XScan can scan various application types. For this training, we will only scan an Html Browser. To scan a web page, a browser must be opened to the relevant page.

In Tosca, right click on a folder in the Modules section and choose *Scan Application* >> *Desktop.* The XScan will now show you all applications it can recognize that you currently have open on the desktop. You can see that for our browser, it shows us only the most recent tab we have had open. Choose the page you want to scan and click *"Scan"*. The XScan reports back the technical information found.

The XScan opens, by default, in "Basic View". Click on the controls directly in the browser, and they will be automatically selected in XScan. By Clicking on the "Advanced" button you can access the Advanced view of the XScan. This Advanced view gives you direct access to all the technical information and controls found on the page. The "Select on Screen" button allows you to click on controls on the screen. This is very similar to what the "Basic View" allowed you to do. Additionally, the technical information of the whole page will be available.

By clicking again on "Select on Screen" you will disable the option to add controls directly from the webpage. Remember, "Select on Screen" does not de-select automatically.

You can switch back to the "Basic View" by clicking on the "Basic" button on the top of XScan: Note that once the Basic view is chosen, the "Select on Screen" mode is enabled by default. By clicking on "Condensed" you can access the "Condensed View" of XScan.

The XScan window minimizes and pins itself to the right side of the screen, allowing you full access to the webpage or application where you are selecting the controls you need. The "Basic" button will send you back to the Basic view of XScan. The "Finish Screen" button will save your work as a Module in the Module section of Tosca

On the top left side of both Basic and Advanced views you can find the "Save" button. This button will save the current set of selected controls as a new Module in the Module section of Tosca. The difference Between the "Save" and "Finish Screen" buttons is that by clicking on "Finish Screen" the scan will save and create a new Module. The XScan will be reset to its starting position with no scanned controls. Once this is done, it will give you the option of scanning the page again. This is useful when you want to create different modules for different parts of the same webpage, without having to re-start the XScan each time.

The "Save" button will not clear any information from the XScan, giving you the opportunity to keep working on the controls you have already selected. Press the "Close" button to close the XScan. "Highlight on Screen", highlights the Controls selected in XScan in your browser.

This is the reverse of the select on screen that we used before. If you know the technical properties of the control, you are able to find it in the tree and it will show the location on the page. This is particularly useful if there are a number of controls with similar or unclear names. You can select them in the tree and see which of the controls it relates to in your SUT.

Additionally:

- Filter is used to filter the types of controls that are visible
- Search can search through the controls

The following buttons assist in identifying controls in different ways. On the left are controls that have been found, which can be identified using the information on the right side. By default, the properties of the controls are visible. Examples of properties include Visible, Enabled, URL, and ClassName.

Let's say I need the Log in link for my Module. I would add a checkmark next to this control. Another method to add this control to the Module is to use Select on Screen. I can go directly to the page and click on the control I need. After clicking on the link "Log in" a green message appears, indicating that the "Selected item is unique".

This means that the selected control is uniquely identifiable. On the right side, I can see it has been **identified by its technical properties** such as its ID, Name, InnerText or OuterHtml. You can use an unlimited number of properties for identification. Identification by property is the most stable method and is completely independent of resolution or zoom settings. It may, however, not be browser independent.

In some cases one browser defines the InnerText with all capital letters, and one with normal notation. Because of this, you can deselect InnerText and select InnerHtml to ensure these properties can be read on different browsers.

However, if I need the link called Books, you'll see in the Web Shop there are two links with the InnerText Books. After adding a checkmark next to the first control called "Books", an orange message appears. Indicating that "the selected item is not unique". That means that the "Books" control is not uniquely identifiable. This control cannot be identified by InnerText alone.

We can use **4** other methods:

- Identify by Parent
- Identify by Anchor
- Identify by Image
- And Identify by Index

Parent will be demonstrated later in the course, but essentially it is using a properly defined parent control to identify the subordinate control. This is also independent of zoom and resolution. To select a parent control, you simply tick the box next to the parent control and, as long as it is uniquely identified, the control underneath will be identified as well. Note that both controls will be saved in the final Module – both the parent and the target control.

The 3rd method of identification, the **Identify by Anchor** method, sets an anchor on the page which the control relates to, and needs to be uniquely identifiable by its technical properties. There are a couple of differences between parent control and anchor control method. First, the target control can be, but need not be, directly related to the anchor control. Any other identifiable control in the tree can be selected. Second, when selecting the anchor control, you will not be selecting it as a control which appears in your final saved Module. To set an anchor, I first focus on my target control. Then I click on Identify by Anchor.

You must ensure that the desired anchor control is uniquely identified. If you want the control to be used as an anchor but not to be saved when you create the Module, make sure to deselect it! Then just focus once again on the target control, and drag and drop the anchor control in the anchor box on the right side of the XScan window. During automation, Tosca will look first for the anchor control, and then search for the target control in relation to the anchor control. It does this using two possible methods: shorted path or coordinate.

- **With Shortest Path**: the system searches each tree level for the control to be identified starting from the anchor control. The search starts directly beneath the control and goes from bottom to top up to the root element
- **With Coordinate**: the exact position of the control to be identified is searched via coordinates. This option is highly dependent on the used screen resolution. If the resolution is changed, the control might not be found during test execution.
-

When the setting "Auto" is used, Tosca will automatically try to use the method Shortest Path. If the Shortest Path algorithm is unsuccessful, the system automatically changes to the Coordinate algorithm. This method is independent of zoom, but not of screen resolution.

Identify by image would identify the control by taking a screenshot. You can use the image itself as the identifying image which Tosca will search for. You can also select an image elsewhere on the page as an identifying image, and Tosca will search for your control in relation to that image. You can set a level of accuracy for which it must search. The more accurate it must be, the more difficult it may be to find on the screen but the less likely it will be that an incorrect control is chosen. This is highly dependent on resolution,

zoom, or any color/style/visual changes of the control on the screen, and of course can only be used for controls that are visible.

The final method of identification is by **index**. This uses a constraint index to number all controls with the same chosen properties on a page. For example, if I select the InnerText "Books", then Tosca will search for all links with the inner text books. Identify by index will then assign a number to each instance of the controls that match this description in order of their appearance in the code (i.e. first, second, third…) and this is it's index. So a control could be 1 of 2 on a page. If the order of these controls in the code changes or number of this type of control changes, the index will automatically be different. So if the order of the books links were switched, then the link which was 1 of 2 on the page would now be 2 of 2.

For this example, let's use anchor control to identify the books link on the page. Expanding the controls with the filter, we can see that the Books link is contained within a container named UL. I will use this as an anchor control. To ensure this control is uniquely identifiable, I'll select it and check the box ClassName. Now, I am able to click on Books and drag the UL into the Anchor Control box to set it as the anchor. Books is now uniquely identified. By default, this method chooses the shortest path in the tree between the controls to identify it. I will deselect the UL so that I do not see this control in my saved Module. It will still be used as the link's anchor.

Because this Module will be used on multiple pages, we will need to change the technical property "Title" to make sure that it can be found, even when the title of the page changes. In this case, when you click on different pages of the Web Shop, you will see that the title changes but the words "Demo Web Shop" are always at the beginning of the title. We will use a wildcard (*) after the words "Demo Web Shop" to tell Tosca that this Module can be found on any page beginning with those words, regardless of what text comes after it.

Before saving, I change the name of the Module to something clear for anyone who will use it in the future.

Within Modules, links, buttons and radio button controls can be grouped. This helps to keep things organized when they are used in TestSteps. To convert controls to a ControlGroup, select all necessary ModuleAttributes, right click, and select *Convert to Control Group.* Rename the ControlGroup. To see the contents, expand it using the downward arrow. The contents can be ungrouped by using *Right Click >> Convert to separate XModuleAttributes.*

Lesson 2 · TestCase Structure

Overview

It is very important to maintain a well-organized workspace in Tosca with clearly named elements. A structure and naming convention that works for the whole team must be determined at the beginning. When it comes to TestCases, this is especially important.

Usually members of a test team will be sharing the same project, or Tosca artifacts, while working in their own workspaces. A workspace is a local copy of certain parts of the repository for the whole project. Options for working in a multiuser environment, including how to check in and out shared elements, will be taught toward the end of this course.

Practical Usage

TestCase Folders are used to organize **TestCases**. As we've briefly mentioned before, TestCases are the basic elements containing the dialog sequence information, or in other words, containing the information of a path taken through an application. During this training, we will create and execute five TestCases.

Each TestCase should be assigned a **WorkState**. The WorkState tells us which stage of development the TestCase is in. This is important for both reporting and for multiuser environments.

The 3 options for **TestCase WorkState** are:

1. PLANNED
2. IN_WORK
3. COMPLETED

PLANNED is the default WorkState that is set when a TestCase is created. This means that a TestCase is expected to be built, but work has not yet begun.

IN_WORK should be set once you have begun working on, modifying, and editing a TestCase.

Finally, a TestCase should be set to COMPLETED when it is ready for review or for execution.

To set the WorkState of a TestCase, focus on the TestCase and ensure that the WorkState column is visible in the working pane. Open the drop down box under this column at the TestCase level and select the desired option.

The purpose of the first TestCase in this course is to verify the calculation of ground shipping costs for purchasing blue jeans in the Web Shop using a credit card. Let's determine which path is taken through the Web Shop.

Demonstration

- First, as a precondition, a user opens the Web Shop Url in a browser and logs in
- Next, they add a product to the cart
- After that, the checkout process is completed
- Then the prices are verified
- And the order can be confirmed
- If the order is successful, the correct message appears after confirmation
- Finally, the user logs out and closes the browser

To keep this TestCase organized, we must use a TestCase Folder and a clearly named TestCase which will contain folders for the TestSteps.

When a TestCase should be run on a browser, a value must be set to specify which browser to use. This can be done using a predefined **TestConfigurationParameter (TCP)** with the name **Browser**, set at the level of the TestCase. To create a TCP, open the Test Configuration tab, right click on the TestCase and select create TestConfigurationParameter. Then enter the data. The browser must be specified as **Chrome, Firefox, or InternetExplorer** (written exactly as seen here). For this training we will always use InternetExplorer. This field is case sensitive.

Lesson 3 · TestSteps

Overview

Now that we have an organizational structure for the first TestCase, we must define the sequence in steps.

Practical Usage

Consider the path. What is the first step to take? Before anything, the Web Shop needs to be opened in Internet Explorer.

Demonstration

For training purposes, you have been provided with a number of Modules which were imported with the Base Subset. These include the Tricentis Standard Modules, which are used for common steering operations across many applications. This Module set, named Standard.tce, is a standard Subset that comes with your Tosca installation. You have also been provided with Web Shop Modules which have been created for various pages within the Web Shop. These Modules would normally have to be created for each project.

To open a webpage in a browser, you can use the Module named Open Url, found within the Tricentis Standard Modules.

A **TestStep** can be created by dragging and dropping a Module onto a TestCase or a TestStep Folder or by using the **Add TestStep** function. To use the Add TestStep function, you simply right click on the object onto which or after which you want the TestStep to be added and select "Add TestStep". You can also use the shortcut "**Ctrl+T**". This can be done on the following objects:

- TestCases
- TestStep Folders
- TestSteps
- TestStepValues
- Reusable TestStepBlocks

The Add TestStep window then appears. Tosca will use the characters you type into the search field to perform a so-called "Fuzzy Search" to search for matching Module names or Reusable TestStepBlock (which you will learn about in Lesson 7 of this course) names. Tosca will display the first 25 results that most closely match your search term. Characters that match the search term will be highlighted in the result.

The fuzzy search will look for the characters in your search term in the order you type them, but they may not be right next to each other in the result. The closer the characters appear in the name of an item, the higher it will appear on the result list. For example, we will search in our workspace for "Top Menu". When I start to type the word "Top", a list of all results which contain the characters "t", "o", and "p" in that order appear. Top menu appears first, followed by all results where the characters are farther apart. I could also, for example, search for the Module "TBox Window Operation" using the search term "two". You can then simply click on the Module you want to add as a TestStep or use the arrow keys to select and press enter.

Make sure that your TestSteps must be structured chronologically. You can simply drag and drop them to reorder them.

Every TestStep should be renamed to reflect exactly what is happening within it. To steer the Web Shop using this step, instructions must be entered into Tosca. In this case, it is entering the specific URL to open.

TestSteps, TestCases, and TestCase Folders can be run in the **ScratchBook**, and the results are only temporarily saved. This is useful for testing before final execution. To do this, I can right click on single or multiple Folders, TestSteps or TestCases, and click "*Run in ScratchBook*". The folders and TestSteps will be executed in the order they were selected.

Watch here how Tosca automatically opens the URL using Internet Explorer, as this is the browser we specified in our TestConfigurationParameter. The results appear in Tosca. If anything went wrong, a red X would appear next to the step with Loginfo explaining what happened. When everything runs correctly, the steps are marked with a green checkmark.

Lesson 4 • TestStepsValues

Overview

Now that you have created all of the necessary TestSteps, we must enter the instructions to steer the SUT.

Practical Usage

TestSteps are organized in the TestCase in the order we want Tosca to proceed to steer the application. Values need to be entered for the TestSteps to tell Tosca exactly how to do this. In this case, we need to instruct Tosca to open the application, order a product, check out, verify that the correct shipping costs have been added to our total, and then to confirm success and logout.

Demonstration

When I expand the TestSteps, I can see the **TestStepValues**. TestStepValues represent controls on the screen of the SUT.

In this view, many columns can be seen. To hide currently irrelevant columns, hold the mouse over the column name and drag downward until an X can be seen, then release it. Columns can be opened again by right clicking the column row and opening the Column Chooser. Various columns can be opened depending on the section of Tosca you're working in.

When filling in TestStepValues, the columns **Value**, **ActionMode**, **DataType** and **Name** should be visible.

Let's have a look at the options for TestStep Values:

To instruct Tosca to activate a button or link, any single character, like X, can be entered as the value next to the specific control. This automatically sets the ActionMode to Input and the DataType to String. It is also possible to activate a button or link by manually setting the ActionMode to input while leaving the value blank. However, it is best practice to enter a value so anyone working on the TestCase can easily see that the value has been used. You will learn more about ActionModes and DataTypes in upcoming lessons.

{CLICK} is used to display the curser during automation on the screen as it clicks on the control, imitating the UI action of a mouse click.

ENTER is used to simulate the keyboard button „Enter".

Specific text and numbers can be entered directly as **values**. For example, as text box input or for verifying certain values.

When steering a TestStep containing a ControlGroup, the value options are available within a drop down list, as seen here. Drop down lists will also be displayed this way.

Dynamic values often need to be used which are not generated until the test is executed.

These may include item-dependent values; for example, transaction number, customer number, etc.; or time-dependent values, for example, today's date, credit card expiration date, delivery date, etc. Dynamic dates will be covered thoroughly in the next lesson.

Special characters and dynamic values are displayed in the following format in Tosca Commander:

- **Example, {<Syntax>} and/or {<Command>[<Parameter>]}**

The expression **RND** is used to generate random numbers in places where value inputs do not have to be exact. To structure this expression, use {RND[length of random number]} where the length specifies the number of digits this generated number must consist of.

You can also use the RND expression for a random number within a specific range:

{RND[lower limit][upper limit]}

The expression **RANDOMTEXT** allows you to specify a string length. The generated text may contain both randomized numbers and letters. **RNDDECIMAL** generates random decimals where a specified number of digits and specified number of decimal places.

When using dynamic expressions, keep in mind,

- Precise values, special characters and dynamic values can be combined with one another. For example, the input of a value and the confirmation with the return button
- The availability of the special character depends on the individual engine
- To have a clear view of your work, **F9** can be pressed to hide all unused TestStepValues. Press F9 again to see the TestStepValues again
- In the context menu for a TestStepValue, **Translate Value** can be used to test your expressions before execution
- A complete list of special characters and expressions can be found in the online Tosca Commander manual

Tosca also allows calculations to be performed using MATH and CALC. In this training, we will only cover MATH expressions. CALC can be used with Excel functions, but Microsoft Excel must be installed locally for this expression to work.

The syntax for the MATH expression is:

- **{MATH[<Operand 1><Operator><Operand2>..<Operator><Operand n>]}**

Any numerical value can be used as an operand, including any dynamic expression supported by Tosca.

Basic arithmetic operations are supported by Tosca (+, -, *, /) as well as equal to (==), not equal to (!=), less than, (<), greater than (>), and so on. Additionally, mathematical functions such as max, min, round, square root can be used. Please see the Tosca manual for a full list of operators.

A simple example of the MATH function:

{MATH[10*4.5]}

Lesson 5 · Dynamic Dates

Overview

Date expressions can be used in Tosca to allow dates and times to be dynamically generated. In this lesson we will go over the dynamic date expressions and possible formats.

Let's look at some dynamic date expressions:

- The syntax for date expressions is: {EXPRESSION[Basedate][Offset][Format]}
- **In these expressions, each syntax - including the offset - is case sensitive**
- Expression is used to express a date or time value and is based on your current system settings. If no format is used or defined, it will revert back to your system settings.

Practical Usage

- Possible EXPRESSION values are:
    - DATE: which will return current date {Date[][][]}
    - TIME: returns current time {TIME[][][]}
    - DATETIME: generates current timestamp {DATETIME[][][]}
    - MONTHLAST: gives you the last day of the current month {MONTHLAST[][][]}
    - MONTHFIRST: returns the first day of the current month {MONTHFIRST[][][]}

Base date: date value in Tosca date format

Offset: values that are added or subtracted from the base date

- Possible Offset examples:
    - +2y: 2 years in the future {Date[03.06.2017][+2y][]}
    - -4M: 4 months in the past {Date[03.06.2017][-4M][]}
    - w can be used for workdays {Date[03.06.2017][+2w][]}

Format: date expression format of the system under test

- Possible Formats
    - dd: day, always expressed as two digits {Date[][][dd]}
    - %d: day, expressed as one digit where possible {Date[][][%d]}
    - yy: is expressed as a two digit year {Date[][][yy]}
    - yyyy: is expressed as a four digit year {Date[][][yyyy]}
    - %M: is expressed as a 2 digit month where possible {Date[][][%M]}
    - MM: is a 2 digit month (01, 11) {Date[][][MM]}
    - MMM: is the three character abbreviation for the month {Date[][][MMM]}
    - MMMM:d is the full name of the month {Date[][][MMMM]}

Demonstration

Use the command Translate value from a controls context menu to force Tosca to calculate a dynamic value without running a TestCase in ScratchBook or in an ExecutionList. This is often done to verify that the syntax has been correct entered.

The Tosca Date format can be found in Project >> Settings >> TBox >> Dynamic temporal expressions.

For more information about dynamic date expressions, please refer to the Tosca Manual.

Here we see 4 examples of the use of dynamic date expressions:

- Example 1: {DATE[][][]} represents the current date in the format that is currently used in your computers settings
- Example 2: {DATE[][][yyyy]} will only show the current year as a four-digit-number.
- Example 3: {DATE[][+2M][MM]} will add 2 months to the current date and only show the months, with 2 digits
- Example 4: {DATE[03.25.2016][+3y][dd.MM.yy]} represents March 25th 2016 as the base date, adds 3 years and shows it in the format dd.mm.yy

Practical Usage

Keep in mind, everything in these dynamic date expressions is case sensitive (e.g. *mm* represents *minutes*)

Lesson 6 · ActionModes & Table Steering

Overview

If we want to steer an object, we have to tell Tosca which action to perform to steer it. ActionModes do just this, by telling Tosca how the value field should be applied to the control.

Practical Usage

So far, we have seen basic ActionModes Input which transfers values to the test object. We can also use Select, Buffer, Constraint, Insert, Verify and WaitOn. In this lesson, we will focus on Select, Buffer, and Verify. WaitOn and Constraint will be covered later in this training. We will not cover Insert because it is only for nonUI automation.

ActionMode Select selects a control to be used without any additional actions.

The ActionMode Buffer can be used to save any type of value during the execution of the TestCase to be used later on.

The purpose of the TestCase is always to verify or check certain values and controls. Whenever I come to the point in my TestCase where I need to perform these checks, I would use the ActionMode Verify.

Demonstration

Let's look at an example:
Let's say I want to store the price of the Blue Jeans I have purchased for verification later on. I would find the control which has the property I want to save. In this case, it is the container PriceBlueJeans's InnerText. Most objects have default properties and behavior, for example Tosca knows that Textboxes have Text, and it treats it accordingly. Not all objects will have default properties, such as DIVS or any other type of container. Because this is a container, we have to explicitly tell Tosca to use the InnerText of the control for buffering. I then choose a name for the Buffer, which is the name behind which I store my value to recall later on. The name is not important per se, but it is ESSENTIAL that when we reuse it later on we type exactly the name, with the exact syntax and cases, as we used for the Buffer. Here I will use the name PriceBlueJeans. Capital P B J and no spaces. I then set the ActionMode to Buffer.

If I run the TestStep in the ScratchBook, I get a message saying that the Buffer was stored. In the menu item *Project* >> *Settings* >> *Engine* >> *Buffer,* you can see all of the Buffers you have stored.

Important to remember – every time I run that TestStep, the value will be overwritten by the new value. So if it is changing in the application, my buffer will also change. To recall a Buffer value, I use the syntax {B[]}

ActionMode Verify is used for verification in every TestCase. We can verify properties of a control, for example whether a control is Visible, Exists, has InnerText with a certain value. It can also be used to verify a numeric value or a string. Keep in mind – verify is case sensitive and uses exact match.

In this case, we could say we would like to verify that if I order a physical product with shipping method ground, that the shipping costs should be 10.00. Here I would type in 10.00 for the shipping cost value, and set the ActionMode to verify.

Keep in mind, the datatype defines how Tosca views the data. You will have to think about the way the object is displayed in the SUT as well as the value you are using for verification.

For example, if you have the datatype set to String, then Tosca will verify an exact match of the text. Typing in 10.00 will then mean that if your Web Shop total is 10 without the „.“ or the „00“, verification will fail. In this case, you could try setting the datatype to numeric, so that Tosca will recognize that 10.00 and 10 are the

same number. To change Tosca numeric formats, like the decimal character separator, go to *Settings* >> *TBox* >> *Number formats* and type „.“ in the value column for Target decimal character. The default matches your machine settings.

What if I want to verify dynamic values?

I can use dynamic expressions in the value column. I can also use any buffered values to verify.

Earlier we buffered the price of the blue jeans and called it exactly PriceBlueJeans. Now, I would verify that the subtotal of my order is 25 times the price of my blue jeans, since I ordered 25 units.

Remember that we already mentioned that MATH functions can be combined with any dynamic values in Tosca. Now, rather than simply multiplying 25*1.00, I will create a Math function, and inside recall the Buffer value for the price of blue jeans in place of the 1.00 in the formula. I will do this using the syntax {MATH [{B[Buffername]}]} or {MATH {B[PriceBlueJeans]}]}. Once I enter {B[, the list of Buffers in the system will be visible in a drop down menu which I can choose from. I then multiply this price by the quantity and select verify. Now, when I run my TestCase, I will see that the price is stored and then later multiplied by 25 and the subtotal is compared and verified.

The sub-total page has the order details displayed in the form of an HTML table. HTML tables allow web designers to display data like text, images, links, etc. into rows and columns of cells.

To steer a table you will first select the respective table control. Then either select the row or column which contains the cell you want to steer. A dropdown on both rows and columns shows you the values that were available when the table was scanned. These values will be either the header of the table for column and/or default values for row that Tosca stores therein e.g. $1, $<n>, $last and so on. While ‘#<n>’ is an absolute value that selects the n-th entry in the table, ‘$<n>’ is always the n-th row or column *after* the header entry defined when the table was scanned. For example, if I define “$1” in the table, and the header is set to the first row, Tosca will actually steer the first row after the header: row 2. You can also use dynamic values as identifiers for rows or columns.

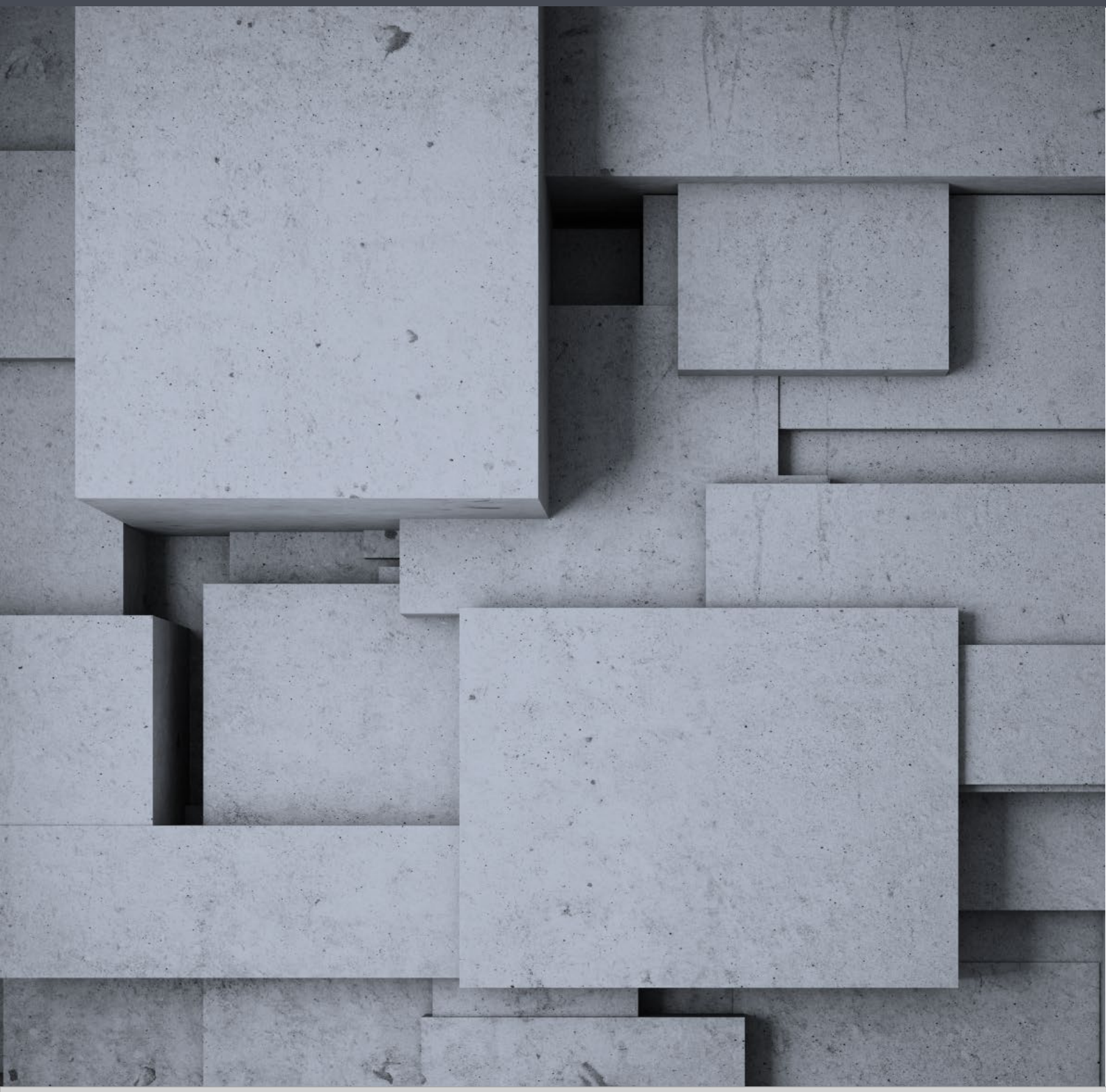
The third step would be to select the cell itself and enter the required values in the Value column. You also need to select the appropriate ActionMode for each cell, depending on whether you want to store a value from that table in a Buffer, to verify something or to wait on a certain state of a property.

In case there are more than 2 rows or columns with the same header and/or similar content, you can work with ActionMode constraint to put a limitation on the differences in order to identify the cell in question. You will use this ActionMode in a later lesson.

If you click on a selected table, row or column in the navigation pane Tosca shows you a detailed view of the object below in the ‘table steering editor’. You also have a “Table Tools” tab in the toolbar for table steering options.

To reset a row/column or a cell simply right click and select ‘Delete’ from the mini toolbar or hit the delete key from your keyboard.

To add a row, column, or cell between 2 already existing objects, right click on the first cell and, in the mini toolbar, click on the ellipsis (…) and select “Create XTestStepValue after this”.



TEST CASE TWO

TESTCASE TWO

Lesson 7 · Libraries

The purpose of this TestCase is to verify the calculation of an additional payment fee for purchasing blue jeans in the Web Shop using a check/money order.

You can imagine that as we create more TestCases, there will be certain steps or sequences of steps which we may need to use more than once. Rather than wasting time by creating these steps from scratch every time, we can create a Library in which we can store these steps, and we will simply have to drag and drop them later on when we want to use reuse them.

Practical Usage

In Tosca, this Library is called a **TestStepLibrary**. The steps inside the Library which we plan to reuse are called **Reusable TestStepBlocks**. Tosca TestStepLibraries can contain any number of Reusable TestStepBlocks which can be reused any number of times. The steps added to the Library should be those which will remain unchanged in any TestCase in which it is used.

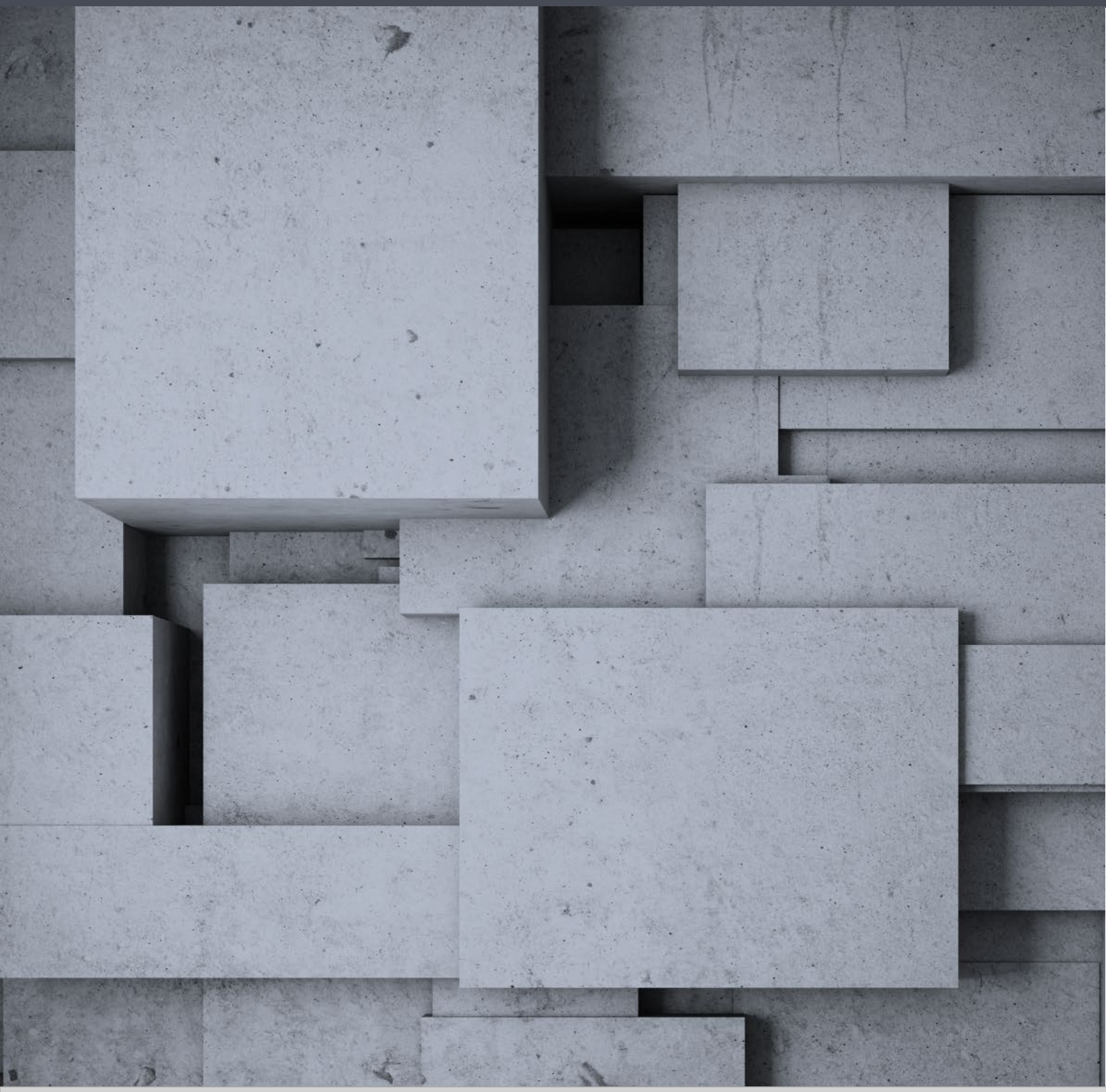
In automation of TestCases, it is best practice to start and end the TestCase at the same place. In this project, we are always beginning by opening the browser, navigating to the Web Shop Url, and logging in. We always end the TestCase by logging out of the Web Shop and closing the browser. To make the creation of multiple TestCases faster and more efficient, we can simply add these TestSteps to our TestStepLibrary.

Demonstration

In the rest of our TestCases, not only will we reuse the precondition and postcondition TestSteps, but we will reuse a few other steps as well. Let's take a look at how the creation of the TestStepLibrary and Reusable TestStepBlocks works.

First, we need to create the TestStepLibrary. A Library can be added to any TestCase folder which does not already contain a Library. This is very simple, you just right click on the folder and select Create TestStepLibrary. To create reusable TestStepBlocks from TestSteps we have already created, simply drag and drop them from their original location. You will see that these TestSteps immediately become **References** to Reusable TestStepBlocks. Where the original TestStep was located, a white arrow now appears to indicate that it is now a Reference to the Library. You may also notice that the Reusable TestStepBlocks in the Library are automatically put into alphabetical order. To save a sequence of TestSteps, you can add them to a Reusable TestStepBlock Folder. TestSteps in folders will not be rearranged into alphabetical order.

If you want to use a TestStep which is in the Library, but modify it without modifying the Library, just for this one TestCase, you can remove the link to the Library by right clicking on the Reference and selecting Resolve Reference. Once a Reference is resolved, it cannot be relinked.



TEST CASE THREE

TESTCASE THREE

Lesson 8 · Rescan, ValueRange, ActionMode WaitOn, Module Merge

In TestCase 3, we will be verifying the application of a discount code and the calculation of the discount.

Overview

In this lesson, we will look at an additional ActionMode called WaitOn. This action would be selected when we want to perform a dynamic *wait on* a control property to reach a specified state, or an entered value, before continuing. For example, to wait on a message being visible or for a control to exists.

We will also look at the **Rescan** function, which allows you to amend Modules instead of creating a new one every time you need an additional Attribute. The major benefit of Rescan is that you can reuse Modules or re-map controls that can no longer be identified.

On the ModuleAttribute level we will add values to the **ValueRange** column of a ModuleAttribute. This column can be used to give users a list of possible values for this Attribute, which will then provide the Tester with a drop down list of values to choose from in the TestCase section.

Finally, we will look at the Module Merge function, which allows you to combine 2 Modules together. In the event that ModuleAttributes are duplicated in the Modules, you can also link these Attributes and decide which method of identification you want to keep.

Demonstration

For us to apply a discount code to our order, we have to enter the code on the Shopping cart page. We don't have the discount code text box yet in our Shopping cart Module. I will now go through the checkout process in the Web Shop and enter a discount code „AutomationDiscount2". Text appears letting me know that I have applied a discount code. I now go back to the Shopping cart Module, right click, and select **Rescan**. This will now take me into the XScan, where I can simply amend my Module by adding new Attributes. Now I will select the controls I want to add and click save.

In the ValueRange column for the **Discount Code**, I want to add a few different discount code options. In this case, the possible discount codes to enter.

At this point, I now have two Shopping Cart Modules. To avoid duplication I will use Module Merge, which brings the Attributes I select and their methods of identification together into one Module, and deletes the second Module. To do this, select both Modules and click **Merge Modules** on the "Modules" tab in the top ribbon.
Click Show.
Both Modules and all of their Attributes will appear below respectively as the Target Module and the Source Module.

The **Target Module** is the one we will keep, and the **Source Module** is the one that will be combined into the Target Module and subsequently deleted. We need to select the Attributes we want to bring from the **Source Module** into the target Module. You can determine which Module is the target and which is the source by selecting the option to **Switch Modules**.

When you have identical Attributes with identical properties, Tosca should automatically recognize this and link them. When you have similar attributes with different properties, they will appear separate, but should show up as **link candidates**. You can then select whether you want to link them or not from a dropdown box. The different properties will appear as conflicts which need to be resolved before the merge is finalized. You

can then select whether you want to use the property from the target module, the source module, or get rid of them both.

When you are ready, press “Merge”. Tosca will let you know how many Attributes have been updated, which usages have been relinked to the new Module, and deletes the Source Module. Press **Close** and save your work.

Once I have done this, I can now drag and drop my new Module into the TestCase DiscountCode. Here I will tell Tosca to input one of my discount codes and to WaitOn the Discount Code Applied message to appear using the value **.Visible == True** before accepting the terms of service and continuing to the next test step. If the message does not appear, the TestCase will not continue.

Remember that adding a discount code will affect our final price verification. Instead of just adding shipping costs to the subtotal, we will also be applying a discount of 20% to the total value of the order.

Lesson 9 · Self-Defined Test Configuration Parameters

To run a TestCase in the ScratchBook or in an Execution List, we may want or need to specify values such as the desired testing environment, test object versions, business chain identifiers, and so on. Earlier, we used a predefined Test configuration parameter (TCP) named Browser, to tell Tosca which browser to run the TestCase in.

Practical Usage

In addition to predefined TCPs, we can set our own values to **self-defined Test Configuration parameter** on the TestCase level.

Demonstration

All objects on which a TCP can be defined have a Test Configuration tab.

As mentioned before, you simply have to open this tab, right click on the object, and choose the option for adding a TCP.

While TCPs and Buffers are used to generate values, they are different. TCP values are generated by a tester and are stored within the repository even before the TestCase is run. On the other hand, a Buffer is generated during run time and is stored locally.

As we know the purpose of this TestCase is to apply a discount code and to verify the calculation of this discount, we can set the value for this code (AutomationDiscount2) and the value of the discount (20%) as a TCP.

We would then just add a Reference to these TCPs wherever we need to recall these values in the TestSteps. The syntax for this is {CP[Parameter]}.

The benefit of using a TCP rather than typing the value directly into the test step values, is that if the value needs to be changed, rather than changing each occurrence, the value can be amended just once in the TCP.

Lesson 10 · Business Parameters

We already learned how to reuse TestSteps by using a TestStepLibrary. Until now, any modification we would have made in our Reusable TestStepBlock would have applied to any References to these blocks in our TestCases. Since user data varies for different TestCases, it would become a problem if each tester were attempting to use the same data. Luckily, there is a solution which allow each tester to use their own user data.

BusinessParameters allows the tester to modify values in Reusable TestStepBlock References without changing the Reusable TestStepBlock in the Library.

Practical Usage

In our example, we want to reuse our precondition of opening the Url and logging into the website, but let's assume we want the input data for Url, Username, and Password to change depending on which TestCase we run. We could add **BusinessParameters** which would allow the test to be run in the future with different user data. When using BusinessParameters, remember that unused values will be hidden. "CTRL +" can be used to show or hide values.

Demonstration

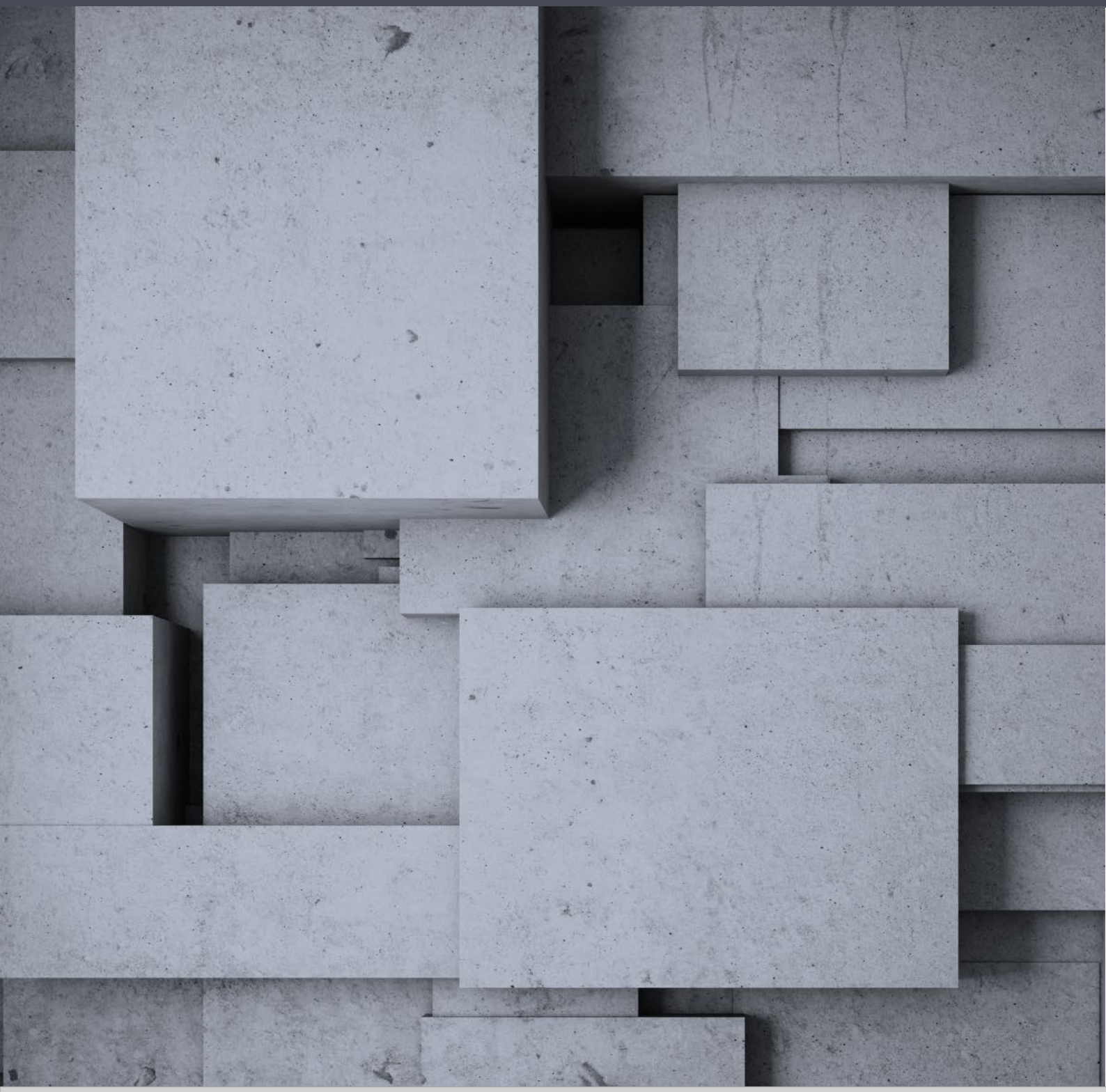
There are two ways to create BusinessParameters. The first is to create them from scratch.

In the Library, on the Reusable TestStepBlock that you want to add BusinessParameters to, you would right click and select Create BusinessParameters. Then you would simply Create Parameters as needed inside this used business parameters container. Once the parameter is created, you have to drag and drop it into the Value column for the relevant TestStepValue so that Tosca knows where to use this information. It will create a parameter link - {PL[Parameter]}.

The second way to add Business Parameters is when you already have the parameters defined in the TestStep. You can simply drag and drop them into the Business Parameters container, and they will create the Reference in the value column as well as the BusinessParameter itself.

If you now look at the Reference to the Reusable TestStepBlocks in our TestCase, you will see that the other steps are hidden, and we now can enter specific values for these parameters.

If you press "CTRL +" you can see all other TestSteps and values, not just the business parameters. To toggle between any {PL[]} links and the actual value to be entered, use the shortcut Shift+F11.



TEST CASE FOUR

TESTCASE FOUR

Lesson 11 · Parent Control, Dynamic ID and Dynamic Comparison

TestCase 4
To reorder a product in the Web Shop, one must first place an order, navigate to „my orders" page, view the details of the most recent order, and then reorder the products. The objective of this TestCase is to test that this works.

Overview
In our first 3 TestCases, we were only able to verify static strings. In this TestCase, we introduce dynamic comparisons, also called **{XB},** which allow you to verify the static portion of a dynamic string or value by excluding only the dynamic portion from the verification. The syntax for {XB} is {XB[]}. We also have the option to store the dynamic portion as a Buffer. If you would like to Buffer the excluded dynamic portion, the syntax for this is {XB[Buffername]}. With {XB} you must always use the ActionMode Verify, NOT Buffer.

Practical Usage
On the order successful page, we receive a message with the string „Order Number:" and then the number of the order we just placed in the system. The text „Order number:" is static, but the number itself changes every time. In order to verify that this message appears, we can't use the order number itself because it will be different each time we run the TestCase. We can use the XB to exclude the order number and only verify the text „Order number:".

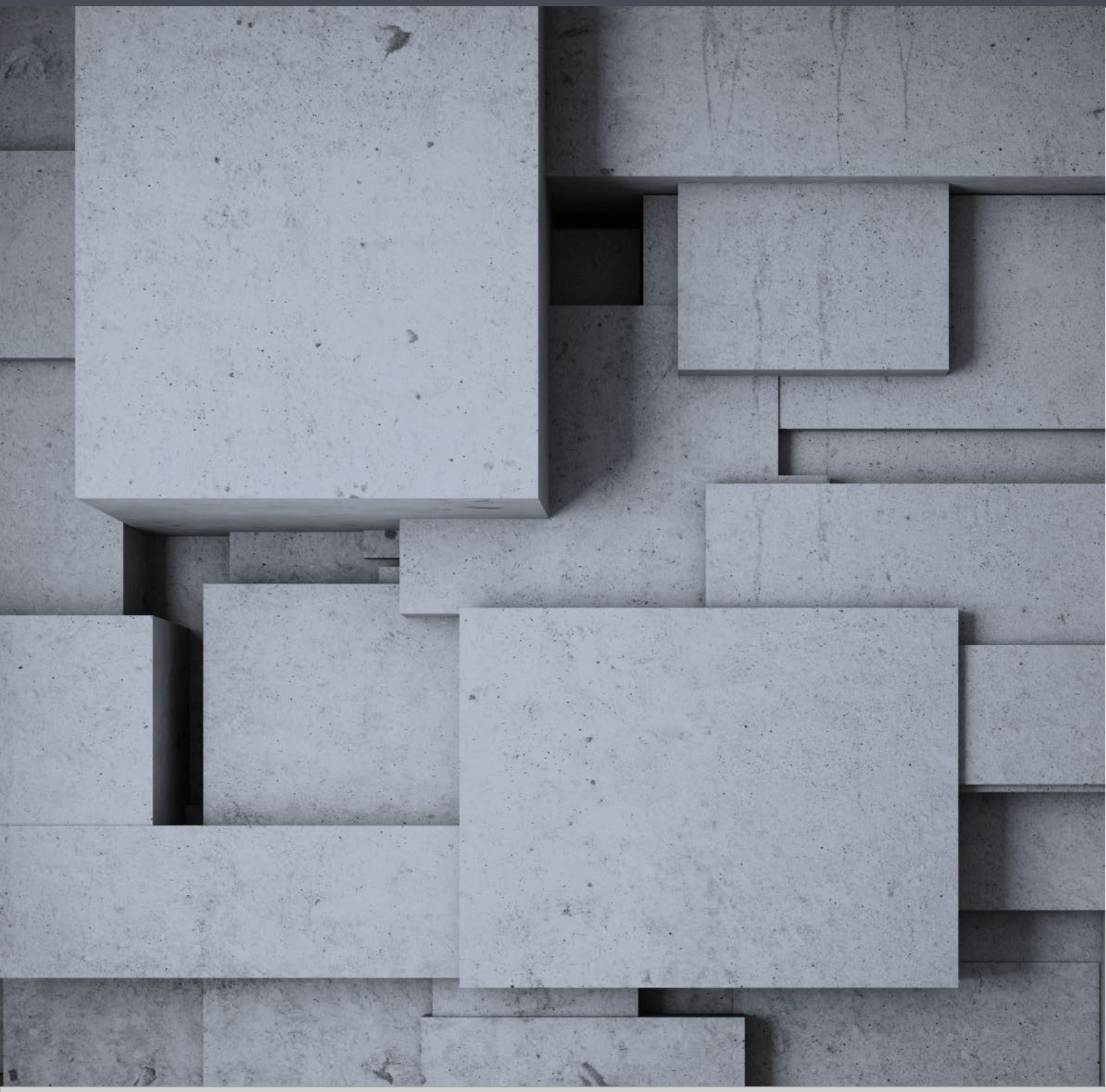
Demonstration
We will also want to buffer this number so that when we want to reorder this exact order, we can type a buffer name into the square brackets of the XB. Our syntax for this TestStepValue would then look like this: {XB[OrderNumber]}, and we set the ActionMode to Verify.

Overview
So far, we have learned how to identify a control by its own properties and by an anchor control. It is also possible to identify a control using its **parent control**. This means that we can use a control that is higher up the object tree and within which the control we want to use is located. Tables and Containers can be used as parents.

Practical Usage
In our Web Shop, there is a page to view our previous orders. If we want to view the details of our most recent order, it will be difficult to identify exactly which details button to click on, as there are multiple identical details buttons. What we can do, is identify the details button of our order by the container in which it is located. We tell Tosca to identify the container using the exact order number at the top of the list.
However, we know that after we place more orders, this exact order will no longer be our most recent order at the top of the list. If controls have dynamic properties, as in this case, we should reflect this by creating **Dynamic Ids** in the properties of that control.

Demonstration
Let's go into the XScan. If we look at the properties of one of these containers, we see that they are identified by the exact order number. If we want to make sure that this number is dynamic, we will have identify the order using the Buffer we created earlier using XB in the parent properties. So, when we run our TestCase and receive the success message, we will Buffer the placed order. We then use this order to identify the container on the orders page to find the correct details button that will show us the most recent order.

Remember, these controls defined by dynamic properties are only uniquely identifiable during run-time.



TEST CASE FIVE

TESTCASE FIVE

Lesson 12 · ResultCount and Repetition

TestCase 5

In this TestCase, we want to go to our orders page and calculate the total sum spent on all orders so far. This TestCase is shorter than the other 4, but we will have a few new terms to learn before we complete it.

Overview

ExplicitName is a configuration parameter which can be added to a Module which allows a change in the TestStepValue name which reflects a technical identification. **Index** numbers objects that have the same properties, or do not have a unique identifier, in the order they appear on the page, using #1, #2, #3 and so on. ExplicitName allows Tosca to steer these objects using the index, taking into account any dynamic changes you have in fact already used ExplicitName whilst working with tables. The ability to use #2 to navigate to the second column is possible because of the ExplicitName Parameter. Additionally using a <Row> or <Col> dropdown is using a value range from ExplicitName.

ResultCount is a property that is the result of a counting process which can be used for all controls. RowCount and ColumnCount can also be used for tables.
TBox Set Buffer is a standard module which allows you to set a value to a Buffer manually.
Repetition is a parameter which instructs Tosca to repeat the steps within the specified level a specified number of times. We can specify repetition on the TestStep Folder level only, which tells Tosca to repeat the steps within the folder.

Practical Usage

On our orders page, we have a list of all orders placed so far. We will have to locate each order, regardless of the amount of orders on the page, find the order total for that order, and add it to the overall total one by one.
To do this we will use all the elements listed previously;

- Explicit name to identify each different order in our order history by adding a dynamic identifier in the module control properties.
- Result count to count how many orders are in our order history
- TBox Set buffer allows us to set a buffer at 0 and one that is blank and then used to add together all of the totals within each occurrence of your orders.
- Repetition will repeat the TestSteps within the TestStep Folder as many times as there are orders.

Demonstration

Let's start with the module Orders Overview one more time to modify the Attribute properties. We will add an ExplicitName configuration parameter with the value „True" which allows us to rename the DIV later in the TestCase section. We also want to make the Order Total container dynamic.
We will change the **INNERTEXT** by adding a wildcard.
The next step would be to use the standard Module „Set Buffer" and set the Buffer named „Sum" to 0. This will be the basis for our overall total.

We will also now set the Buffer named OrderNumber and leave the Value blank. We do this because in the last TestCase we changed the outer text of the module to reflect the buffered order number from the order confirmation. This meant that we could uniquely identify the last order on the page. We need to leave this element in so that the previous test case will execute, however in this TestCase it will have the knock on effect that our count up will only ever find this last ordered item (as the buffer still has the value from the previous

TestCase). To get around this we set the buffer to a blank value. Therefore Tosca will identify all the modules on the page with outer text "Order Number:"

We know that we will need to find the order total for X number of orders (the number of orders on the page). Let's have Tosca use the ResultCount property to find the total number of orders on the page and Buffer this number. This is now the number of orders Tosca will have to look for.

We then tell Tosca to Buffer the order total for each DIV and add it to the Buffer Sum. We want this sequence to repeat the amount of times as determined by the Buffer.

We want to repeat this sequence of steps as many times as the „NumberofOrders" Buffer we created earlier.

Since the DIV we are referring to will change, we will now rename that TestStepValue to signify to Tosca that it should invoke the ExplicitName property, allowing us to use the same Attribute more than once. Since we do not know exactly how many times we will use it, we will tell Tosca to refer to the number of repetitions previously specified on the Folder level. Whichever repetition number we are on, Tosca will refer to the corresponding DIV.

When I finish, the Buffer „Sum" should be equal to the total amount of money I spent in the Web Shop so far.

Lesson 13 · Requirements

Overview

The Requirements section of Tosca provides a dashboard for simple and effective test management. It is used for planning test processes, monitoring progress, and risk coverage optimization.

Requirements are criteria, both functional and nonfunctional, that are relevant to a testing project. These refer to the business requirements that we need to check to ensure that our application is running properly. By weighting our Requirements, in other words specifying the level of risk and importance of all aspects of our project, we can make sure that we are taking a risk based approach to testing and lowering the risk that a release of our application will result in losses or damages.

Requirements are organized in RequirementSets which represent different aspects of the application (for example, frontend, backend or nonfunctional

Practical Usage

The Pareto Principle says that 80 percent of our risk comes from 20 percent of our application. If we take this risk weighted approach, we can prioritize our testing to cover most of the risk. Risk levels are calculated using "Requirements". Linking the TestCases and ExecutionLists to the Requirements shows the progress and the state of the entire project. No TestCase should be created that does not cover a requirement.

Demonstration

You have been provided with the Requirements for the Web Shop.

The frequency column shows us how frequently each Requirement is expected to be used in our shop. The more frequently it would be used, the higher the weighting. The Damage class column is related to the degree of financial risk associated with each requirement. These weightings typically would come from the business analyst. The weight is then calculated using an exponential function to show the relative importance of each requirement to all others at the same level.

You can see that, for example, the Shopping cart and Order process have the highest weight compared to all other requirements. Within the Shopping cart, Add products has the highest weight compared to all other elements in the Shopping cart, while in the Order process requirement, Calculate Shipping Costs and Payment Methods have the highest weight. The contribution (%) shows the relative percentage of risk coverage for each Requirement in a RequirementSet according to its and its parent's weights.

To link together Requirements and TestCases, drag and drop your TestCases (or Template instances) to the relevant Requirement. As soon as a TestCase is complete, it should be linked immediately to the corresponding Requirement.

Lesson 14 · ExecutionLists

Overview
Once our TestCases are complete, we no longer want to run them in the ScratchBook, but in the actual ExecutionLists.

Practical Usage
ExecutionLists run and store the results for the executed TestCases. Unlike the ScratchBook, the ExecutionLists store all historic results. They can be run unattended and overnight.

Demonstration
In the Execution section we can organize our elements with ExecutionList Folders and ExecutionEntry Folders. Within them we will have our ExecutionLists, which contain a series of Execution Entries that need to be run. Each TestCase is represented as an Execution Entry and can only exist once within each ExecutionList. To run the TestCases, we must first link them to the ExecutionLists by dragging and dropping the TestCases from the blue section onto the corresponding ExecutionList in the green section.

If you drag and drop only individual TestCases into the ExecutionList, you will have to add or delete any new TestCases manually into the ExecutionList. If you drag and drop a TestCaseFolder, you can simply use right click „Synchronize“ on the ExecutionList to update the ExecutionList to match the current state of the folder in the TestCases section.

Once they are linked, you can run ExecutionLists and let Tosca do its work. To run a TestCase more than once, you can modify the property Repetitions of an ExecutionEntry Folder or an ExecutionEntry itself.

There are three options for executing the ExecutionLists:
- Run: which automates the TestCases in the ExecutionLists
- Run from here: which will start running from the selected ExecutionEntry and continues until the end of the ExecutionList
- Run as a Manual TestCase: which, as the name suggests, would only be used for manual tests

If there are individual TestCases in the ExecutionList you do not want to run, you can right click on them and choose Disable from the context menu to exclude them from execution.

If TestConfigurationParameters are set on the TestCases, and also an additional TCP on the ExecutionLists, the TCP set on the ExecutionList will override the one set on the TestCase.

If a TestCase fails, unlike in the ScratchBook, your Recovery Scenarios and CleanUp Scenarios will kick in when necessary to continue running the executions. When an execution passes, it will appear with a green box with a white arrow. When an execution entry fails, it will appear with a red box with a white X inside. You can see the duration of executions as well as a summary of the Loginfo. If an execution is rerun, it will appear with the most recent information. You can expand the execution entry to see all previous runs.

You can manually right click on an ExecutionEntry and select Set result to: passed, failed, or no result.

Link the ExecutionLists to their respective RequirementSet. By doing this, we will be able to have an overall view of the state of our project. We can see how much of our requirements have been tested in the Execution State column, and how much of our Requirements have been specified in TestCases in the Coverage specified column.

Lesson 15 · Loops and Conditions

Overview

Some steps should only be done given a certain condition. In this case it is possible to use special expressions which contain conditions and sometimes loops. In Tosca, a WHILE statement tells Tosca to repeat a step as long as a particular Condition is met. When it is no longer met, the loop stops and Tosca will move on to the next step.

Practical Usage

In our Web Shop, we will tell Tosca to empty the shopping cart as long as there are items inside it to remove since each item has a separate Remove checkbox and the number of items in the cart is dynamic, we will create a WHILE statement with a loop. So, as long as there is an item in the cart, remove it. When there are no more, it will move on to the next step.

Demonstration

Inside of a TestStepFolder, we can right click and select "Create a WHILE statement" from the context menu. We then have a condition, where we must tell Tosca which condition should be met for the step to be completed, and the loop, where we tell Tosca what to do in the event that the condition holds true. Just like in other TestSteps, we drag and drop the modules we need for each, and fill in the necessary TestStepValues.

In this case, we want to ensure that the shopping cart table exists in order for the loop to kick in. We then instruct Tosca to remove the last product in the cart. And then it continues performing this loop until there are no more products.

If there are no products in the cart, it means that it will no longer exist on the page.
On the properties Tab of the loop, it is by default set to repeat a maximum of 30 times. This WHILE statement is actually a TestStep, which can be added to any TestCase or Recovery Scenario, which we will talk about in the next lesson.

Aside from WHILE, we also have IF and DO statements. As we said before, WHILE will continue to repeat a function until the condition is no longer met.

- DO is exactly like WHILE, but it will run the loop at least once, before checking whether the condition is met.
- IF will perform a function only once given a certain condition, but we recommend Never ever using IF statements.

Lesson 16 · Recovery Scenarios

Overview

You may have already realized, as you ran the ScratchBook in previous exercises, problems can occur while running TestCases. For example a random popup may appear, or a TestCase did not execute properly. Imagine if you had hundreds of TestCases to run, you would not want to react manually every time this happened. As an alternative, we will create instructions for Tosca, so in case of an error occurring, it will know what to do to try to recover the TestCase; or in the event that it is not recoverable, to clean up the space for further TestCases. RecoveryScenarios will ONLY kick in during final execution, not in the Scratchbook.

Practical Usage

To attempt to solve the problem to prevent TestCase failure, we can add:

- Recovery Scenarios on the TestCase, TestStep, or TestStepValue level. This is called the Retry Level and can be specified on the property tab of the Recovery Scenario. This will tell Tosca, in the event of a problem, how far to go back to reattempt running the TestCase.
- To use Recovery Scenarios we have to make sure that we have recovery enabled in our Settings. In Settings, go to Settings->TBox->Recovery. The first 3 define if dialog failure, exception failures and verification failures should be recovered. Usually you would want only to recover dialog and exception failure, since verification failures may point to actual errors in your System Under Test.

The last three define how often specific recoveries should be tried, according to the set retry level

All those Settings may be set as TestConfigurationParameters on TestCase or TestCase Folder level – those will then override these Settings

- Once we have enabled recovery, we then create a Recovery Scenario Collection in our TestCase. The Recovery Scenario Collection will then be populated with a series of TestSteps which will instruct Tosca with what to do. Successful recovery would mean that all TestSteps are successfully completed.
- If a TestCase cannot be recovered, a CleanUp scenario added to our Recovery Scenario Collection instructs Tosca to stop trying to recover this TestCase, clean up, reset our testing environment, and to move on to the next TestCase.

Demonstration

We will create a Recovery Scenario Collection by right clicking on the level where we want the Recovery Scenarios or CleanUps to be. We then right click again and create a RecoveryScenario. Make sure to set the Retry Level on the properties tab. We can now add any steps we want into our recovery as well as drag and drop the Reusable TestStepBlock we created in the last section (our WHILE statement) into it.

If recovery at this level doesn't work, we want Tosca to try to reset and restart the Web Shop then move on to the next TestCase. Since we know our TestCase begins by opening the browser and logging in, we want to tell Tosca to log us out and to reset using a CleanUp scenario which closes the browser. To end a task, we can use the standard module StartProgram and just tell Tosca that we want to kill a task using the argument "taskkill", and then tell it which particular program or window we want to kill. In this case, it will be our Internet Explorer browser denoted by iexplore.exe (chrome would be chrome.exe).

If you wish to see a graphical representation of the flow or path of your TestSteps, click on the Control Flow Diagram tab. Here you can see how Tosca checks whether a condition is met (Yes or No) and then determines which path to take.

Keep in mind that loops like WHILE do not have to go inside Recovery scenarios, and should only be used when absolutely necessary. In fact Recovery Scenarios can often be used INSTEAD of loops.

Remember: Recovery scenarios kick in only during actual execution in the ExecutionLists, NOT in the ScratchBook.

You can see the recovery scenario in action: when there is an additional item in the shopping cart, the Verification of prices step will fail, the Recovery scenarios kick in, and the test case is run again with an empty shopping basket. Looking at the LogInfo in the execution entry shows that overall the testcase passed, although when you look in detail, the first run through failed. You can also see the CleanUp Scenario in action when a TestCase does not log in – you will then be unable to proceed through the checkout process. Our recovery scenario will kick in on dialogue failure, for example when it cannot find the continue control for shipping address, and will close the browser and move on to the next test case.

Lesson 17 · Constraint and FireEvent

Overview

In this lesson, we will cover a couple of topics which are unrelated to our five TestCases: the ActionMode Constraint, and FireEvent. The FireEvent will be used in solving an obstacle on the Tricentis Obstacle Course. This online obstacle course is made up of many challenges which can be solved using Tosca Automation. This particular obstacle is helpful in testing what you've learned through this course and is rated at a medium difficulty level.

Let's first begin with learning about the ActionMode Constraint. It may occur that you want to perform an action on an object or value in a table, but you need to limit the search for the object by using more than one identifier.

Practical Usage

This can be used in our Shopping cart. We are able to steer a table by looking for a particular product, and then delete the product by looking for its name. For example, let's say we have a Music 2 product worth $3.00 in our cart, a Music 2 product worth $10.00, and a book worth $10.00. If we want to delete the Music 2 product worth $10.00, it is not enough to search for just the product called Music 2, as there are more than 1, and neither is it enough to search for a product with a price of $10.00. Instead, we have to search for both product name Music 2 and price $10.00.

Demonstration

If we create a TestStep using the Shopping cart Module, we will be able to steer this products table to search for the product Music 2 and the Price 10.00. In contrast to our earlier table-steering, where we specified the row or column which contained the cell we wanted to steer, the ActionMode Constraint will allow us to find or identify a row by searching for specific values within it. We will then set the ActionMode Constraint to tell Tosca that both of these values must be met in order to select the appropriate row in the products table. This will also then work no matter what position in the table this product falls. Constraint can be used anywhere that you have a hierarchy within your Modules, not only for tables, but the most common use in Html is in tables.

We can tell Tosca to then select remove, and update Shopping cart.

Overview

Another issue you may face is that you want to trigger a FireEvent. FireEvent notifies the SUT that a change has been made to the properties of an object.

Practical Usage

For example, let's say a text box allows you to type in a number, but then there is no option to click enter or submit to alert the system that a number has manually been entered. Instead, we want to trigger a FireEvent which alerts the system that a number has been entered.

Demonstration

This steering parameter will be added to the ModuleAttribute on which the FireEvent should be executed. Now the property will be set to the input value specified in the TestCase. You can also change other properties with FireEvent, a full list of options can be found in the manual.

We will now create a TestCase using this Module. We will tell Tosca to Buffer both numbers, and then add them together. When we run the TestCase, the TestStep will not only insert the number into the text box but will also notify the system that a change has been made to the control according to the FireEvent parameter.