Automation Specialist Level 1

SURVEY

Tricentis Automation Specialist | Level 1

Course Transcript

- Version 2022
- Designed to be used with Tricentis Tosca version 14.x

Course 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 transcript

- This transcript is specifically designed to supplement training of the Tricentis Automation Specialist | Level 1
- The transcript is divided into the 39 Lesson sections and supplies the script used for the lesson videos.
- This transcript is not aiming to be a complete manual.

Introduction

INTRODUCTION

Introduction to Automation Specialist Level 1

Hello and welcome to the Automation Specialist Level 1 course. My name is Jordan Grantham and I am the Global Education Strategist at Tricentis. Today marks the start of your journey through Test Automation using Tricentis Tosca.

By the end of this course, you will gather the knowledge and skills required to automate tests for a simple HTML application using Tosca. This is an ambitious yet achievable goal. Let's take some time in this video to get to know the topics covered in the course, and also the delivery method. The course is broken down into five sections. Each section covers one specific Tosca element.

We'll start by looking at what we're testing. We refer to this as the **System Under Test,** also called the SUT. In this particular training, the SUT that we're testing is a HTML Web Shop, especially designed by Tricentis.
You can't really test an application if you don't know how it works. Therefore, the goal is to get an idea of how the SUT works, and which use cases need testing.

In section one of this course, you'll also get a brief overview of the capabilities of the test automation tool Tricentis Tosca.

Once you're acquainted with Tosca, and the testing environment, you'll learn how to capture the technical information from the SUT by simply scanning it. The scanned information will be stored in elements known as Modules. The best part is that the process is code-free.

In the next section, you'll learn how to create TestCases using these Modules, and then populate these TestCases with the necessary Business Information. You'll learn how to configure your TestCases to achieve high testing coverage, increased reusability, and resilience within your testing portfolio.

By this point, you'll have created your TestCases. The next step is to execute these TestCases. The substantial benefit of executing your TestCases is that Tosca captures the results of your Execution runs. Now, you'll have automated your complete tests.

However, there's even more that Tosca has to offer. Before closing the testing cycle, you'll need to link your tests back to the Test Requirement that it's expected to cover. We'll also look at working with Requirements too. In a nutshell, Tricentis Tosca enables you to build automated tests quickly and efficiently. As a tester, you'll probably face various challenges while automating tests in your work. However, Tricentis will help you to tackle them. We accompany you throughout your entire testing lifecycle. We don't only help you test your SUT to see if it is working defect-free, but also help you decide what to test, and if what you're testing will scale properly.

Last but not least, Tricentis enables you to centrally manage your testing projects. To get the most out of this course, we've put together a series of lesson videos, quizzes, and a final exam. You'll also be able to practice along the way with exercises and the base subset that was created for the purposes of this training. And because you're here to learn, we've also prepared solution videos for all exercises in this course. Once you've completed all lessons, you can test your knowledge of Tricentis Tosca by taking the final exam.

Once finished, you'll receive a **Certificate of Achievement.** After that, you can take further courses to deepen your Tosca knowledge, or to learn about other tools in the Tricentis offering. If you want to expand on your Tricentis product skills even more, remember to register for our webinars, follow us on Facebook and LinkedIn, and subscribe to our Tricentis Academy YouTube channel. There, you can find hundreds of videos in the form of User tutorials, Guides, and Tips for using Tricentis tools. Now let's get started with Automation Specialist Level 1.

Workspace Set Up

WORKSPACE SET UP
Lesson 01 - Introduction to Tosca

In this lesson, you will learn how to **set up your workspace in Tosca Commander**, how to select the correct type of repository, where your workspace, and how to import and export artifacts using Tosca Subsets.

To help you meet these learning objectives, we need to answer four basic questions: What types of workspaces can you create in Tosca? Where within Tosca can you create a new workspace? How should you go about it? And last but not least, why should you choose a specific type of workspace over the other. Let's begin with where within Tosca you can create a new workspace and how you should go about it.

Once you've installed Tricentis Tosca, and you have a valid license, you're ready to start. Open **Tosca Commander** and click on "Create a new workspace". Before continuing with the workspace creation, let's see what types of workspaces you can create. And this is based on the types of repositories that you can select from, and that's single-user and multi-user repositories.In the "Select type of repository" dropdown select "None" for a single-user workspace. For training purposes, we're choosing **single-user workspace** because this repository type is intended for only one workstation. However, in real-life scenarios, you will be most likely using a multi-user environment. Once this is done, enter the name for the new workspace. Click on "Use workspace template", and then locate the **base subset** that you downloaded. Click "Ok". If you don't use this option, you will have a blank workspace.

When you click the check box, the default option will be "Standard.tsu" which is the standard subset provided to you for free with your Tricentis Tosca installation. The "Standard.tsu" file contains some **useful Modules** for automating your workflow, predefined virtual folders, and standard reports.

Once a workspace has been set up, Tosca Commander is connected to a repository containing the Tosca artifacts.

But why differentiate between different types of workspaces? What's the benefit? In Tosca, a workspace represents a **defined working area** that is located either locally or on a network drive in relevance to the single-user workspace. Workspaces are the fundamental unit within you will work to automate tests in Tricentis Tosca. Workspaces are made up of **various sections** that help you plan, create, and execute your tests.

In a single-user workspace, **only one person** has access to the workspace. That's why in this case, the administration of the data and sources is not necessary. Multi-user workspace makes the data administration simpler since all the data of a project is safe to one central location, that is the common repository. Individual objects can be modified **without interfering with other user's work**. And to ensure this, the user marks the required data as checked out for editing. This data is locked for other users until the user releases it again by using the functionality "Check-in". Tricentis recommends assigning **only one user** to each workspace.This is however, only relevant to single-user workspaces.

While working in Tosca, it will be necessary to import and export subsets. The subset is a file containing **Tosca artifacts** that can be transferred from one project to another. If you want or need to move elements from one common repository to another, or between single-user workspaces, you will **export a subset** from the first workspace and import it into the second one.

However, as previously mentioned, you will only work in a single-user workspace for this training. To export a subset, you simply select the objects or folders that you want to export and select "Export Subset". When you want to import a Tosca subset, or a .tsu file, simply select the "Import" option. How the objects are imported into your workspace will be determined by how they were exported. But please note that it might be helpful to **create a component folder** while importing subsets, so that it is easier to locate the new objects in your workspace.

You could also use a subset as a **template** for a workspace, like we've done earlier. Remember, while creating a new workspace, select "Use workspace template", and then simply browse your machine, for a file that you want to use. The elements will be imported to the new workspace just as they were originally exported.

And this already marks the end of this lesson. You're now able to set up your workspace in Tosca Commander, select the correct type of repository while doing so, and import and export artifacts using Tosca Subsets.

Lesson 02 - Navigate your Tosca

In this lesson, you'll learn how to **navigate Tosca**. More specifically, by the end of this video, you will be able to steer the Navigation and Working panes in Tosca. You will be able to recognize and manage key Tosca sections and you will know how to save, undo, and redo your work.

To help you meet these learning objectives, we need to answer four basic questions: What are the key sections within Tosca? Where can you find those? Why are there multiple key sections and how can you navigate between all of those sections? Alright, let's get right into it!

We'll start by locating the **key sections within Tosca**, so that you have a visual representation of what we are talking about. This is the default view of Tosca in a new workspace. The menu on top has a **ribbon** design that allows you to perform corresponding functions. Save, undo, and redo are also possible from this menu. What is important to mention here is that after saving your work, you will not have the option to undo or redo.

On the left-hand side, you have the **Navigation pane**, which enables you 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, we have the **column header**. There are various columns available, and they can be shown or hidden as needed. Simply right-click on the column header and open the "Column Chooser" to view your options.

Double-click on any column name to open that specific column. To hide columns, click on the column name and drag downward until an X appears and then you can just release it. On top, just below the ribbon, you have the window tabs, which are used to switch between different sections.

Speaking of sections, what sections are there in Tosca? There are **nine different sections** in total, but in this course, we will only cover **four** of them. Since you don't always need all sections for every project, we're going to focus on the basic ones for now and these are: **Modules, TestCases, Execution and Requirements**. Let's go through them one by one!

We'll start with the **Module section**. Think of Modules as folders that contain controls in your System Under Test, so, for example, the log in page with its fields. In other words, Modules comprise the **technical information** required to steer your System Under Test.

To run a test, you also need **business information**, such as login credentials and in the **TestCases section**, you're going to enter this business information necessary to instruct Tosca to automate your System Under Test.

Then, after creating your TestCases, you can execute these Tests in the **Execution section**. This is also where you'll log the results of different Execution Runs over time and you can identify and report bugs step by step.

We don't only want to run TestCases, we also want to design a test structure that mirrors our business cases and risks, and this is done in the **Requirements section**, where you can set up the requirements of your System Under Test. This section will also provide you with an overview of the automation status of your entire project. As you can tell, each section has its own purpose, and this is also why we keep them separated. That doesn't mean though that they can't communicate with each other. They can, and we're going to talk about this in another lesson!

For now, let's see how you can manage these sections in Tosca. You can view multiple sections 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" and "Move to the Next Tab Group".

Another way to open different sections is to open them via "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, Execution, and so on, and you can also create objects here. This already brings us to the end of this lesson!

You are now able to steer the Navigation and Working panes in Tosca. You know how to recognize and manage key sections in Tosca, and you know how to save, undo, and redo your work.

Lesson 03 - Tosca Workspace Best Pratice

In the previous lessons, you got an in-depth overview of how to create and navigate a Workspace in Tosca. In this lesson, we're going to take a look at **four fundamental Best Pratices** to follow when creating your Workspace so that by the end of this lesson, you will be able to apply them.

To help you meet these learning objectives, we need to answer four basic questions. First of all, what are the four fundamental Tosca Best Pratices to adhere to when setting up your Workspace? Secondly, where within Tosca can you actually implement them? Then we are going to look at how to implement them and why should you follow those Best Pratices. Let's start with the what, where, and how.

It's most advisable to **use naming conventions** throughout your entire project. It is important to agree on a common naming convention within your team and apply to all artifacts within your project. Choose a name first and add a more granular description later. Following naming conventions will keep your project readable and make it easier for everyone to understand when working in a distributed team environment. Also, it creates better business readability. Another Best Pratice we recommend is related to how you set up your **Repositories**. So let's take a look at that.

Depending on whether you follow an Agile or Waterfall approach, you would set up your Repository accordingly. We either way recommend creating Component folders. And within those, you will place your section folders. Here are some tips for structuring your Repository.

Have a component folder that includes **templates and references**. Template folders contain sample structures with your naming conventions that you can reuse as you add new components to your Tosca project. Have a Component folder that includes content that is relevant for all projects. That would be **Libraries and Modules** for example. Have one component folder for each project. Within the folder, use the project name and the Requirements, TestCaseDesign, TestCases, Execution and Module folders.

As mentioned earlier, it is also important to define an appropriate naming convention throughout these folders. Another Best Pratice is to create artifacts based on the **four eyes principle**. This means using the three approval stages for artifact creation, which are: "In work", "Ready for Approval", and "Approved". New artifacts should only be created within "In work" folders and then moved to "Ready for approval" folders. As soon as they're approved by someone else, they can be moved into the "Approved" folders where they can be used.

Additionally, "In work" and "Ready for approval" should include one folder for each Tosca user. But why should you follow these Best Pratices while setting up your Workspace? What is the benefit? Applying those Best Pratices when working with Tosca will ensure **easier maintenance, greater efficiency, and higher readability**. However, it is important to identify which Best Pratices to use depending on your project environment. Because as with any software, Tosca can be used in different ways to achieve the same outcome.

Make sure you use Best Pratices that most suit the needs of your project. By following these Best Pratices, you will ensure a **consistency throughout your project, better readability and communication**. And that's not only within your team, but also with other business stakeholders. These Best Pratices will make your life a lot easier when working with Tosca.

We've now reached the end of this lesson, which means you're now able to apply Best Pratices when creating your Workspace. Go ahead and put these into practice while completing the following exercise.

Lesson 04 - Introduction to the SUT

In this lesson, we want to take a look at the System Under Test used in this course for training purposes. We will see how to navigate the website and log-in with an account. Additionally, we will cover how to install the correct plug-in to prepare your browser for testing with Tricentis Tosca.

To help you meet these learning objectives, we need to answer four basic questions: What is the System Under Test? Why should you know the System Under Test inside out? Where can you access the browser plug-ins and how do you install them? The System Under Test or SUT that we will be using for the Automation Specialist Level 1 Course is the so-called **Demo Web Shop**. We will refer to it as the Web Shop through this training. It is used exclusively for Tricentis training purposes and it is hosted on a public website. As the name suggests, the Web Shop is meant to resemble an online shop.

Knowing your System Under Test inside out is going to be crucial when approaching testing. You will need to know what the **expected and the desired outcomes** are, as well as the different possible paths to reach them. This knowledge is going to be central in understanding possible bugs and defects found while testing. Alright! Let's see how it looks like.
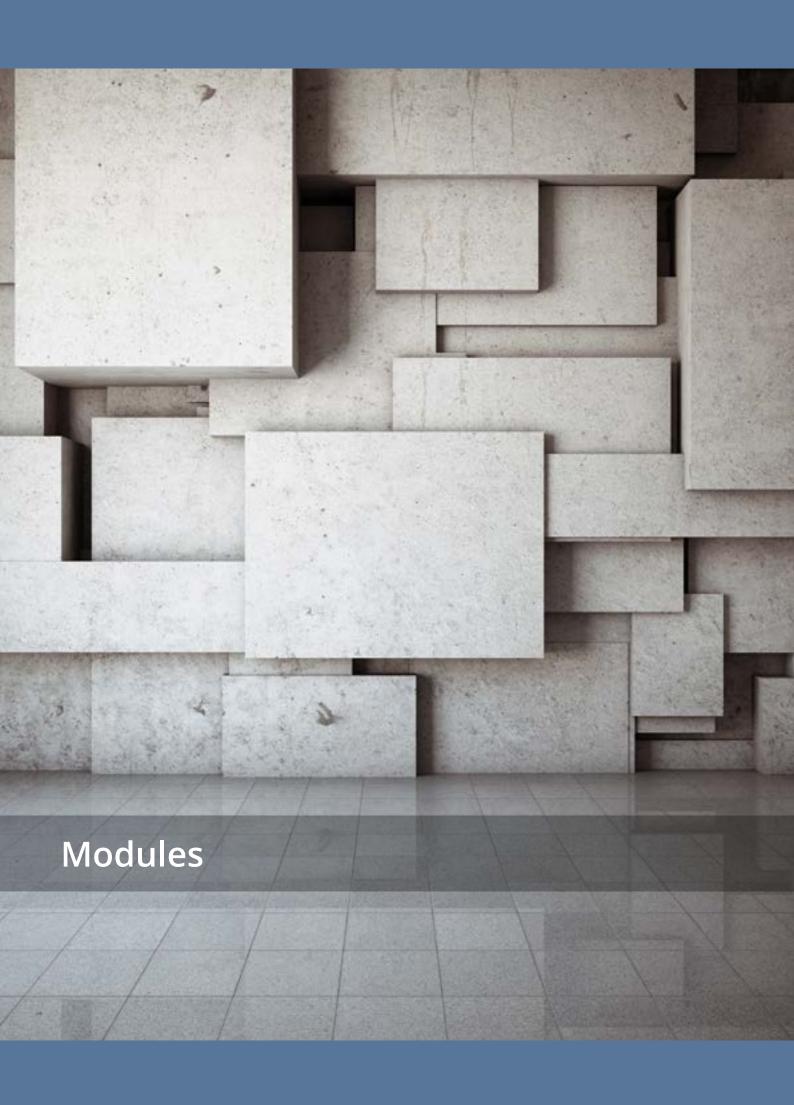
Within the Web Shop, users are able to create an account, browse items for sale, and complete the checkout process. There are numerous paths that you can take when using the Web Shop and depending on the path chosen, the final price or outcome will vary. For example, the items purchased may vary, the Payment or Shipping Method, or whether a discount code is being applied or not.

You will soon learn that each of these paths can be represented for testers as TestCases. And in order to test the Web Shop, you, as the tester, will need to become a registered user. Once you've registered, please tour the website. That means, add items to your cart, complete the Checkout Process, and overall get familiar with the website layout.

Before starting your testing journey, one final step is needed. You will need to **install a browser plug-in** to allow a smooth communication between Tosca and the browser of your choice. You can find the link to the browser plug-in in the Tosca manual, accessible from our **Support Portal**. Let's see how to install it.

You will first need to select and download the right plug-in for the browser that you plan to use. For the purpose of this training, please use **Google Chrome**. Once you have installed it, the Tricentis icon will appear on the top right corner. You are now ready to test.

You have reached the end of this lesson, which means that you're now able to access and navigate the Demo Web Shop. You're also able to install the available browser plug-ins, depending on which browser you want to use for testing. This lesson also marks the end of our introduction section to the Automation Specialist Level 1 Course. Go ahead and test your knowledge with this exercise.

Modules

MODULES

Lesson 05 - Introduction to the MBTA

By the end of this lesson, you will be able to explain and demonstrate how Tosca uses Model-Based Test Automation to increase testing efficiency and testing effectiveness.

This lesson will help you meet this learning objective by answering four basic questions. First of all, what is Model-Based Test Automation? Secondly, where does it fit within the testing cycle? How does Model-Based Test Automation differ from Script-Based testing? And last but not least, why should you choose Model-Based Test Automation over Script-Based where possible?

As we have highlighted in the Platform Introduction Course, Tricentis Tosca uses the Model-Based Test Automation approach instead of the Script-Based Automation one. Let's see what this model has to offer.

Model-Based Test Automation is a **software testing technique.** During testing, Models are created. They contain the **technical information and the expected behavior** of the System Under Test. From there, the model is fed business data that is needed for steering the system and conducting the testing.

But where in the process does this happen? At run time, the model automatically combines the technical and the business data together and checks them against the System Under Test. Therefore testers don't have to do the testing manually.

How does Model-based Testing differ from the Script-Based one? With Model-Based Test Automation, **anyone** from developers, the business experts can contribute to test automation. Developers can provide information to the System Under Test and identify which part of it is problematic and needs testing.

On the other hand, business experts can easily view, control, and update the Test Data as needed. This approach also **eliminates the maintainers** burden that you erodes most tests automation initiatives. For example, if your application changes, something is added, or deleted, you can update the model and the change will automatically be made to all the impacted tests. Tricentis Tosca's Model-Based approach supports easy test automation at the UI and API layers. It also supports Service Virtualization and Exploratory Testing. Instead of programming a test automation framework, we can rapidly scan the application's UI or APIs with Tricentis Tosca to create a business-readable automation model.

The modules are Lego-like "building blocks" that can be **combined and reuse** to create your tests. They can also be reused as many times as needed. Thanks to these models, testers no longer have to script, rather, only enter the correct test data. This also means that you'll no longer need to maintain a code repository.

And this is why Model-Based Test Automation has the edge against the Script-Based one. The amount of overall maintenance needed is also reduced. The models only need to be changed once and the changes are applied everywhere. The user interface in Tosca will also stay consistent for all technologies, making it easier to test the system end-to-end.

And this, brings us to the end of this lesson. You are now able to explain and demonstrate how Tosca uses Model-Based Test Automation to increase testing efficiency and testing effectiveness.

Lesson 06 - Standard Modules

Although there are thousands of different digital solutions and software applications, many of them share the same basic elements and functions. When talking about HTML based apps for example, just think about opening a URL. The process is the same. Creating new Modules for this type of repetitive functions, whenever starting a new project, results in a **misuse of your time and resources** that you could otherwise use better. Is there a solution?, I hear you ask. There is indeed, and this is using the Tricentis Tosca Standard Modules.

By the end of this lesson, you will be able to use these Modules successfully. For this, we will take a look at what Standard Modules are, where you can find them, and how you can import them. Of course, we'll also highlight the benefits of using Standard Modules when working with Tosca.

Tricentis Standard Modules are a **series of modules useful for performing common execution tests** within or outside your SUT. Here is a list of the Standard Modules Tricentis offers at the moment. As you can see, you have a wide range of Modules you can choose from. From basic Wndows operations, to process operations, SAP related Modules, and other specialized Modules. Where exactly can you find them and how can we import them? The Standard Modules are part of the **Tricentis Standard Subset**, a subset that Tricentis provides you with.

You can recall this subset when creating your workspace, or you can import this subset at a later point as well. Once imported, you can find the Standard Modules within the Module section, under "Modules>>Standard Modules".

As some functions are almost standardized, or very much alike, it would be redundant to have to scan them every time you need to use them. For this, Tricentis provides you with the **pre-created objects and their technical information**, you don't have to worry about scanning and identifying the right technical information on your own.

This simplifies and speeds up your test automation. Overall, standard Modules work very much like any other Module, as they are technically just like any other Module. That's why they can be steered immediately without having to scan any additional information.You simply create **TestCases** and then **TestSteps** by adding the Standard Modules to the TestCase section and then filling in the details.

Why would you use the Standard Modules? The benefit is that these Modules allow you to **quickly and easily automate tasks or objects** that would otherwise be complex, or time consuming to automate every time. The use of Standard Modules is advised and is Best Pratice as this are tested modules known for their resilience and reliability.

Additionally, Tricentis updates and expands the repository of Standard Modules with each subsequent Tosca release. This means that you will be able to always use them to their full potential. If you need information on the Standard Module scope and their directions of use, you can find this information in the description column of each Module. This information is also available for all Standard Module attributes. If the description column is not visible to you, remember that you can use the Column Chooser to add any column you like.

You've now reached the end of this lesson and are now able to use the Tricentis Standard Modules successfully.

Lesson 07 - Introduction to the XScan

You've already learned about Standard Modules. To create Modules for the pages and controls that are not standardized, you have to scan those using **Tosca's XScan functionality**. By the end of this lesson, you will be able to navigate the XScan window and use the different XScan views and buttons. This lesson will help you meet these learning objectives by answering four basic questions: What is XScan and what can you scan with it? Where you can initiate XScan? How do you scan your application with it, and why is XScan's on screen highlight method useful when creating your testing Modules?

Before creating your first Module, let's familiarize ourselves with XScan and see what it is. Tricentis Tosca XScan can **scan various applications** and it also supports almost all major technologies. The Automation Specialist Level 1 Course focuses on scanning and testing an HTML-based SUT.

To be able to scan the SUT, you need to have the browser open on the relevant page before starting Tosca's XScan. It's also important to make sure that your system display **resolution is set to 100%** and that the browser **zoom is set to 100%** before starting the scan.

We have already covered how to install the browser extension for Tosca and how to set up your workstation and browser resolution. Now, where can you start the XScan? In the Module section, right-click on the Module folder where you wish to create the Module and select "Scan >> Application". The XScan window will now open and show you all applications that you currently have open on the desktop. Choose the browser with your SUT open and click "Scan". The XScan reports back with the technical information found. Now, the XScan window will by default open in its **Basic view**. Click on the objects on the SUT page that you want to automate. All these fields will be automatically selected in XScan.

Now that you know how to start Tricentis Tosca's XScan, let's see how you can use it. We will cover how you can manage the different XScan views and also how to scan your controls. Let's start with the **Advanced view**, which you can access by clicking on "Advanced". In this view, you will have direct access to all the **technical information and controls** found in the SUT.

The "Select on screen" view allows you to click on controls directly in the browser. This is very similar to what the "Basic view" allows you to do. However, here you additionally see the available technical information of the entire page. You can turn off the "Select on screen" view by clicking again on "Select on screen". Remember, "Select on screen" does not deselect automatically.

To switch back to the "Basic view", just click on the button "Basic"on top of XScan. Please note that once the Basic view is chosen, the "Select on screen" mode is enabled by default.

Lastly, let's see the Condensed view of XScan, which you can access by clicking on "Condensed". In this view, the XScan window minimizes and pins itself to the right side of the screen, giving you full access to the SUT page, where you can select 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. On the other hand, the "Finish Screen" button 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 SUT page again. This is useful when you want to create different Modules for different parts of the same page, or the next one, without having to restart the XScan every time. The button "Save" will not clear any information from the XScan, giving you the opportunity to keep working on the controls you have already selected. Press the button "Close" to close the XScan.

Finally, let's see why you would need the method "Highlight on screen" in XScan. Let's suppose you are not sure which objects to select in your SUT. In this case, you can use the option "Highlight on screen". The "Highlight on screen" option highlights the controls selected in XScan in your SUT Page. This is the reverse of the "Select on screen" method that we used before. If you know the technical properties of the control, you will be able to find it in the tree and it will show you its location on the SUT page.

This is particularly useful if there are **several controls with similar or unclear names**. You can select the technical properties in the tree to see the controls to which they relate to in your SUT.

Now you have now completed the introduction to XScan and are able to navigate the XScan window and use the different XScan views and buttons.

Lesson 08 - Scan the SUT with XScan

In this lesson, you will learn to **scan controls on an HTML webpage using XScan**. We will help you achieve this learning objective by covering these four questions: What are the different ways to create a Module? How can you do this? In which Tosca section can you create a Module? Why should you identify your controls? Let's take a look at the different ways to create a Module containing the fields you need to steer the application or transaction you are working on.

There are two ways to do this: you can select the needed controls by check-marking them or you can use the button "Select on screen". Either way, you can create a Module within the Module section, as the name suggests.

Let's go to the Module section. We'll start with the first method and show you how it works. With the XScan window open, you can see all available fields listed on the left pane. Using the **Filter option**, you can also access controls that are not necessarily visible on screen.

Let's say you wish to add the "Log out" link to your Module. This control is missing as we scanned the Module before while being logged out from the Demo Web Shop. The only thing that you need to do is to add a check mark next to this control in your object tree to select it. Another method of adding this control to the Module is to use "Select on screen". You can go directly to the page and click on the control you need.

If the control you are selecting is not aligned with the actual control position on the screen, please check your system resolution and browser zoom, and make sure they are set to 100%. After adding each field, a green message appears, indicating that the "The selected item is unique". This means that the **selected control is uniquely identifiable** at the time of the scan. On the righthand side, you can see which Technical Properties have been used by Tosca to identify this object. The check marks indicate which **Properties** will be used to identify the object during automation.

In addition to identifying a control by its own Technical Properties, there are **four other identification methods:** identify by anchor, identify by parent, identify by image, and identify by index. In this course, we will focus only on the first two methods: identification by properties and by anchor. There are cases when you will need to use one or more identification methods. This really depends on your SUT and its properties. That is why in Tosca you can use multiple identification methods. Once you've added all your controls and they are all uniquely identifiable, you can name your Module, but just make sure to choose a name that describes the Module's purpose, or the page scanned.

You are now ready to save your Module. This works in the same way regardless of whether you are scanning a normal desktop application or an HTML application. The Module created is now visible in Tosca. If you need to make changes to controls that have already been added to a Module, update a Module after changes to the application, or add new controls that were not added to the Module originally, you can do so by using the "Rescan" function.

To **rescan a Module**, you need to have the SUT open at the desired page. Then you need to select the relevant Module and right-click on it. Select "Rescan". XScan will automatically detect the open window of your application, so you don't have to select the application window again.

By default, the XScan window will show you the list of the previously selected controls. These controls can be deselected, added, and their identification method can be changed. Once you are done with the modifications, you can save the scan. This will update your existing Module.

You have now learned to create Modules, and these will be the building blocks for your automated TestCases. This means, you are now able to scan controls on an HTML webpage using XScan.

Lesson 09 - Identify Controls

As mentioned already, you can identify controls in your test application by using multiple identification methods. In this lesson, you will learn to **identify controls by properties**. We'll start by taking a look at what identification by properties is, how and where you can define it and why should you use this method of control identification.

Identification by properties is the **default identification method** in Tosca XScan. By identifying controls using specific properties, you make it easier for Tosca to find that specific control when executing your TestCases. You can use an unlimited number of properties for identification. However, we recommend only using those that are **necessary and stable**. This is because when you run your test, these properties must all be found in the same state in order to identify your object properly.

The more properties you select for identification, the greater the chance that one of the property values is dynamic or has changed and the control will not be found. This may make maintenance more difficult. You can define the properties by which the control can be identified in the "Identify by Properties" window, which you can find in xScan. This window contains all the **technical information and properties of the selected control**. You can find this window on the right side in XScan. Let's look at an example.

If you want to add the control "Books" to a Module, you will get the message that this control is not uniquely identifiable. That's because in the Demo Web Shop, there are two controls with the InnerText "Books". We can make it uniquely identifiable by changing the properties Tosca uses to find this control in the SUT.

To have an overview of all the properties you can use for identification, click on "Load all Properties". The tab will expand and show you all the properties you can use for this scope. To add a new identification property to your control, simply select the one you think is necessary by ticking the box and if needed, add the corresponding value to the column on the right.
Why should you use this method of control identification? Identification by properties 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 the InnerText and select InnerHtml** to ensure that these properties can be read on different browsers.

Since we are talking about Module properties, let's take a look at the Property "Title", because this one is particularly helpful. For this, let's switch to the Demo Web Shop and take a look at the area we scanned earlier. As you can see, these controls are present on several pages. For this reason, we can reuse the Module we created to steer these controls from different pages. To do this, we will need to change the technical property "Title", just 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. Let's switch now to the XScan.To tell Tosca that this Module can be found on any page beginning with those words, regardless of what text comes after it, you will use a wildcard and you will put this wildcard after the words "Demo Web Shop". Before saving, change the name of the Module to something clear for anyone who will use it in the future.

You have reached the end of this lesson. You are now able to identify controls by properties in XScan. Give it a try by completing this exercise!

Lesson 10 - Identify by Anchor

You've already learned to identify Controls by properties. In this lesson, you will learn to **identify Controls by using an Anchor**. Additionally, we will also cover how to create a ControlGroup in your Module.

We'll take a look at what identification by Anchor is, how and where you can define it, and why you should use this method of control identification. Identification by Anchor is another way of identifying controls. It works by setting an Anchor on the page that the control relates to.

For this to work, the Anchor needs to be **uniquely identifiable by its technical properties**. Just as with any identification method, you can identify a control by an Anchor in the Advanced View of Tosca's XScan.

Let's get more practical and see how you can identify a control by an Anchor. We'll do this with the control "Books". Expanding the controls with the Filter, you can see that the link "Books" is within a container named UL, which is an **Unordered List**. You can use this container as an **Anchor Control**.

To set this as an Anchor, focus first on the Target Control, that is the control you want to identify by an Anchor. Then, click on "Identify by Anchor". Next, focus once again on the Target Control and drag and drop the Anchor Control in the Anchor box, which is on the right side of the XScan window.

To make sure that this control is uniquely identifiable, you can select it and then check the box "ClassName". If you want the control to be used as an Anchor but not to be saved when you create the Module, just make sure to deselect it. 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 by using two methods: the **ShortestPath or Coordinate**. With the ShortestPath, the system searches each tree level for the control to be identified starting from the Anchor Control.
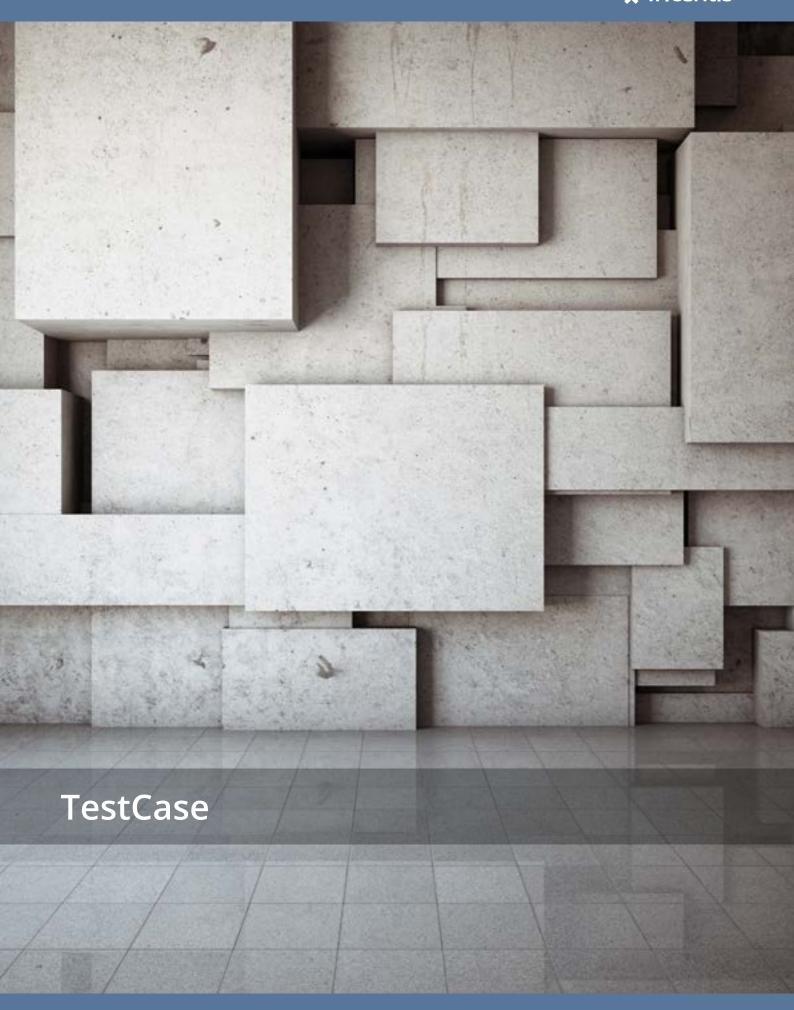
The search starts directly beneath the control and goes from the bottom to the top, up to the root element. When using the Coordinate method, the exact position of the control to be identified is searched via coordinates. However, this option is **highly dependent on the screen resolution**. If the initial resolution changes, the control might not be found during test execution. When the setting "Auto" is used, Tosca will automatically try to use the ShortestPath. If this proves unsuccessful, the system automatically changes to the Coordinate method. One reason why you would want to use this method is because it is independent of zoom. However, it is not independent of screen resolution, so make sure to check your screen resolution and adapt it accordingly. Another reason is that 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.

Also, you can use an Anchor without having to add it to your final saved Module. Now that you have created your Module, you should also take care of **organizing the elements** within it. Within Modules, you can group Links, Buttons, and Radio Button controls. This helps to keep things organized when these **controls are used in TestSteps**. To convert controls to a ControlGroup, select all necessary ModuleAttributes, right-click, and select "Convert to Control Group". Then rename the ControlGroup. To see its contents, expand it using the downward arrow. The contents can then be ungrouped by using "Right Click >> Convert to separate XModuleAttributes".

And this brings us to the end of this lesson. You are now able to identify controls by Anchor in XScan and also create ControlGroups.

This video also marks the end of our Module section. Two out of five sections are already completed. Give it a try by completing this exercise.

TestCase

TESTCASE

Lesson 11 - Introduction to the TestCase Section

By the end of this lesson, you'll be able to **navigate around the TestCase section** in Tosca and **create a folder structure** ready for your TestCase creation. We'll help you achieve these learning objectives by answering these four basic questions: What are TestCases and TestCase Folders in Tosca? Where can you create them within Tosca? Why is it important to use a folder structure for creating your TestCases, and how should you go about it?

TestCases are the basic elements containing the dialogue sequence information, or in other words, containing the information of a path taken through an application. In Tosca, TestCases are built from the **Modules** you created in the Module section. These modules will provide Tosca with a technical information needed to steer your application. To steer the application, you also need business information, which you'll input into the relevant TestCases.

The TestCase section, which is the blue section in Tosca, is where you will provide Tosca with all the business information necessary to steer through your application. The TestCases you will build in this section should be determined by the business requirements of your application. It's always Best Pratice to **define a folder structure** that reflects the business workflow. Another Best Pratice is to use Naming Conventions. Having a clear folder structure and using **Naming Conventions** will keep your projects readable. These Best Pratices also make it easier to work in a distributed team environment.

Moreover, they ensure better **business readability.** You will define your structure by using folders, which we will demonstrate in a moment. It's very important to maintain a well-organized workspace in Tosca with clearly named elements. As you learned in previous lessons, a structure and Naming Convention that works with the whole team must be determined at the beginning. When it comes to TestCases, this is especially important. TestCase Folders are used to organize TestCases. The high level structure you use to set up your TestCase section should closely reflect the structure of your business requirements, which you will learn later in the course. Once you have set up your TestCase Folder structure, you can start creating TestCases.

During this training, you will create and execute three TestCases. In the following section, we'll demonstrate how you can do this. To add a TestCase to a TestCase Folder, you simply select "Create TestCase" or use the shortcut "CTRL N + Control T". Within the TestCase, you will add TestSteps, which make up the flow through the necessary pages of the application.

These TestCases will contain the business information to steer through the objects on those pages. Because some TestCases may include several phases, and then may be different people working on them, it's recommended to set up a base structure of the TestSteps within your TestCase. This helps to make your TestCases more business readable. This means that you should set up a structure for your **TestCase Folders** that reflects review process by making use of the four eyes principle. This allows users to work in their own folder, push for review, and easily view approved TestCases.

We recommend the following TestStep Folder structure: **"Precondition"** – which contains steps that set up your environment, like launching the app or logging in. Then **"Process"** - which contains the steps required to perform a verification and includes the verification or comparison. These can be broken down into smaller folders if necessary. And finally, **"Postcondition"** – which contains all steps necessary to reset your environment to a state where the next TestCases can run. For example, logging out and closing the app again.

These folders can be reused using Tosca libraries, which you can learn more about in later training sessions, or additional Academy material, like our microcontent on YouTube. Finally, make sure to assign a WorkState to each TestCase. A WorkState tells you in which development stage the TestCase is in. This is important both for reporting and for multi-user environments. The three options for TestCase WorkStates are: **PLANNED, IN_WORK, and 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 a WorkState of a TestCase, focused on the TestCase and ensure that the column "Workstate" is visible in the working pane. Open the drop-down box under this column, at the TestCase level, and select the desired option. If you don't have the Workstate column visible, you can add it very easily. To do this, right-click on the header row and select "**Column Chooser"** from the context menu. There, you'll find a list of available columns that you can add to your view. Double-click to choose your desired column.

This marks the end of our first lesson in this section. You are now able to navigate around the TestCase section in Tosca, and create a folder structure ready for TestCase creation. Give it a try by completing this exercise.

Lesson 12 - Create TestSteps

In this lesson, you'll learn how to **populate your TestCase** with TestSteps using Standard Modules. You'll achieve this by looking at: What TestStep are, how and where you can create them within Tosca, and why to follow specific Best Pratices when managing your TestSteps.

As explained in previous lessons, every Tosca installation comes with a **Standard Subset,** called Standard.tsu. This contains a number of Modules, including Standard Modules, which are used to perform common steering operations across many applications.

Looking at an example in the System Under Test, let's say for example, that you need to verify the shipping costs of an order. What is the first step that you need to take? Before anything, you need to open the Demo Web Shop. To open it, you can use the StandardModule named "Open URL", which is found under the path "Standard Modules >> TBox XEngines >> Html".

Once you import a Module in the TestCase section, it becomes a TestStep. A **TestStep** is an action that you want to perform on your System Under Test. In Tosca, you can automate your TestCases and TestSteps in the TestCase section. Let's see exactly what this looks like. You can create a TestStep in multiple ways. One method is by **dragging and dropping a Module** onto a TestCase or a TestStepFolder.

You can also use the function **"Add TestStep".** Simply right-click on the object into which, or after which, you want the Test to be added and select "Search and Add TestStep".

Another common method of creating a TestStep is to use the **shortcut "CTRL+T".** This shortcut can be used on different objects in Tosca, such as TestCases, TestSteps, TestStep Folders, TestStepValues, and 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".** This will search for matching Module names and Reusable TestStepBlock 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 close of the characters appear in the name of the item, the higher it will appear on the results list.

Once the TestStep is added, it will appear at the bottom of the parent TestCase or TestStepFolder. After adding a TestStep to a TestCase, there are a few Best Pratices we suggest following when managing your TestSteps. This way, you'll make sure that there won't be any misunderstandings when working in a multi-user environment.

Therefore, it's Best Pratice to **rename the TestStep to reflect its action**. This ensures business readability when working in a team environment. For instance, in this TestStep, we want to open the Web Shop. We therefore rename the TestStep to reflect the relevant action and this ensures clarity. This is one Best Pratice.

Another Best Pratice is related to the structure of your TestSteps. Make sure your TestSteps are structured in a **chronological order.** To reorder them simply drag and drop them into the appropriate order.

Once you've added your TestSteps, and define your flow through the application, the next step is to add the business information required for Tosca to steer the controls on the screen. This will be covered in the next lessons.

This lesson is complete. You are now able to populate your TestCase with TestSteps, using Standard Modules. Give it a try by completing these exercises.

Lesson 13 - Populate the TestCase

So far, you've learned how to create TestSteps by adding the Modules.

The focus of this lesson is going to be on **populating your TestSteps with the Values** in order to instruct Tosca to steer the SUT accordingly. More specifically, you will learn how to use **basic Values** and **basic DataTypes** for your TestSteps Values. You'll achieve this by looking at: Why do you need TestSteps Values, where within Tosca can you use them, and what are the most commonly used TestStep Values and how would you use them.

Let's begin with a high level overview.

TestSteps are organized in theTestCase in the order you want Tosca to steer the application. To tell Tosca exactly how to do this, you need to enter **TestSteps Values** for the TestSteps.

In our sample SUT, you need to go through the ordering process. Tosca will need to open the SUT, log in, order an item, follow the checkout process before logging out, and closing the SUT.

TestStep Values can be seen, entered or modified from the TestCase section of Tricentis Tosca. To view them, you just need to **expand the TestSteps.** You can do so either by clicking on the arrow, or by right-clicking on the TestStep and selecting "Expand all". You can also use the **shortcut "CTRL +"** or **"CTRL -"** to close them.

TestStep Values represent controls on the screen of the SUT. When filling in TestStep Values, the columns **Name, Value, ActionMode, and DataType** are visible.

In this lesson, we'll talk about the Values you might enter in the columns Value and DataType. The TestCase is now ready to be populated. This is the phase when you enter values that will then instruct Tosca how to steer the SUT.

Let's have a look at what are the different options you can use for adding TestStep Values, and how to utilize them.

To instruct Tosca to activate a button or a link, any single character, like an **X**, can be entered as the value next to the specific control. The **ActionMode column** will automatically be set to "Input" since you have defined an input action. It's also possible to activate the button or link by manually setting the ActionMode to Input while leaving the value blank. However, it is Best Pratice to enter a value so that anybody working on the TestCase can easily see that the value has been used.

Besides from entering an X in your TestStep Value, you can also use the **value "{Click}".** During automation, using "{Click}" will display the cursor on the screen and simulate a user action to move the mouse to the control.

Another TestStep Value you can use is **"{ENTER}".** This simulates the keyboard button "Enter". If you want Tosca to enter text or numbers into any field in an application, you can enter them directly as Values, for example, as a text box input or for verifying certain Values.

Let's now move onto DataTypes.

In Tosca, there are five **DataTypes** that you can select according to the scope and nature of the value you want to enter. **String** is the default DataType for actions such as alpha numeric text, and many other dynamic values. **Date** should be used when entering static or dynamic calendar dates. **Numeric** is used when entering arithmetic formulas. **Boolean** is used to enter one or zero types of data. **Password** is useful when you want to use a password in your TestStep without having it visible in the TestCase. Sometimes you may have a long list of TestSteps Values that haven't been populated.

To have a clear view of your work and hide all **unused TestStep Values**, you can press **"F9"**. To see theTestStep Values again, press "F9" again.

With this, we've concluded this lesson. You're now able to populate your TestSteps using basic TestSteps Values and DataTypes. Give it a try by completing this exercise.

Lesson 14 - Test Configuration Parameters

In this lesson, we'll cover the basics around Test Configuration Parameters.

By the end of this lesson, you'll be able to explain what a **Test Configuration Parameter** is and know how to set it, and populate specific TestCases with Test Configuration Parameters.

You'll achieve this by looking at: What a Test Configuration Parameters is, why we use it, and where and how you can create it in Tosca. A Test Configuration Parameter is a parameter that you can set for Tosca objects. This way, you can configure your tests further. Let's say for example, that a TestCase should be run on a **specific browser.**

By using a **TCP**, you can specify which browser Tosca should use to run a test. For this, you would use the TCP named **"Browser",** and set an appropriate value such as Firefox or Chrome, etc.You can use a TCP to **set specific values,** such as: **Test Object Versions,** for example, Release Numbers. **Identifiers of various Test Environments,** for example, Windows 10. **Connection identifiers,** such as, URLs. And **Business Process Chain Identifiers.** By applying specific values to your Test Configuration Parameters, you can **simplify the maintenance** of your tests and avoid repetitions. This will streamline your test projects substantially.

As briefly mentioned before, you can set a TCP on **different levels**. These include TestCases, TestCase Folders, and ExecutionLists, just to name a few.

Here it's important to mention that if Test Configuration Parameters are set on the TestCases, and also an additional TCP is set on the ExecutionLists, the **TCP set on the ExecutionList will override the one set on the TestCase.**

In the Tricentis Tosca manual, which you can find on our Support Portal, you can access the list of all Tosca objects for which you can set a TCP. All these objects have an additional tab called **Test Configuration in the details view.**

Let's now see what this looks like.

For demo purposes, we'll create a TCP to set the browser for running a TestCase. To create a TCP, open the Test Configuration tab, right-click on the TestCase, and select "Create Test configuration parameter". You can also use the **shortcut "CTRL N + Control .".**

Next, choose the TCP named "Browser" from the dropdown menu. As you can see here, these TCPs are predefined. You can also add self-defined TCPs. This is something that we will cover in another lesson.

The next step is to enter the data. The browser must be specified, for example, Chrome or Firefox. For this training, we'll always use Chrome just as it's written here. This field is **case sensitive**, so make sure to use correct spelling and capitalization. And this is the last technical fact in this lesson, which you've now completed.

Let's revisit the learning objectives.

You are now able to explain what a Test Configuration Parameter is, and populate specific TestCases with Test Configuration Parameters. Give it a try by completing this exercise.

Lesson 15 - Action Modes

In this lesson, the focus will be on ActionModes.

By the end of the lesson, you'll be able to differentiate between the available **ActionModes** in Tosca, and also apply them to steer the controls in your System Under Test. You'll achieve this by looking at what ActionModes are, and why using different ActionModes makes sense. We'll also look at where within Tosca you can apply these ActionModes and how you can use them to **steer TestStepValues.**

Simply put, ActionModes tell Tosca what to do with the TestStepValue to steer a particular control. This means that you can select them within the TestCase section in the **column "ActionMode".**

Let's see what this looks like in Tosca. In the column "ActionMode", you can select from the ActionModes **"Input", "Insert", "Verify", "Buffer", "WaitOn", "Select",** and **"Constraint".** We already saw that the default ActionMode "Input" is used to transfer values to the test objects. In this lesson, we'll focus on the ActionModes "Select", "Verify", and "WaitOn". The ActionModes "Constraint" and "Buffer" will be covered later in the course.

We'll not cover the ActionMode "Insert" as it is only used for non-UI automation, and therefore not within the scope of the training.

Let's begin with the ActionMode "Select". **"Select" is a passive ActionMode** used for navigating tables or dropdown menus but without engaging with the application. For example, you can use this ActionMode to select the table that you want Tosca to focus on, in this case, it's the Cart Total table in the Demo Web Shop. As you can see here, I am not actively engaging with the table. On the contrary, I'm just selecting the table to be able to get to the data inside it.

By comparison, if you were to **actively input** or retrieve data from the table or the application, you must use the ActionMode **"Input"** and not "Select". This is why you'll use different ActionModes for different steering types.

Let's now move on to the ActionMode **"Verify".** The purpose of a TestCase is always to verify or check certain values and controls. After all, a TestCase that does not verify anything is a TestCase without a scope.

For example, it might be important to verify that a particular shipping method you chose for a product you had ordered adds the correct amount to your total. You can see here, choosing the shipping method "Ground" changes my total.
In another example in this TestCase, I want to verify that the message stating that the order was successful is visible and appears in the Status bar. Whenever you come to the point in your TestCase to **perform these checks,** you can use the ActionMode **"Verify".**

Let's now focus on the ActionMode "WaitOn". You select **"WaitOn"** when you want to perform a **dynamic wait** for a control property to reach a specified state. Alternatively, you use "WaitOn" to wait for an entered value before continuing the execution. Once the condition is satisfied, Tosca reacts and continues the automation workflow. For example, sometimes controls take time to load. Once the synchronization time runs out, the TestStep will fail even if there was no error in the application. To avoid this, we use "WaitOn" to wait for a message to become visible or for a control to appear.

This is particularly useful when **expecting mild delays in website** or application responses. Thanks to "WaitOn", a TestCase will wait until a condition is fulfilled and then immediately continues.

This is Best Pratice over inserting a static wait, as this decreases the overall execution time. For example, here you can tell Tosca to input one of the discount codes and to wait on the message "Discount code applied" to appear. For this, you'd use the value **"Visible ==True"** before accepting the Terms of Service and continuing to do the next TestStep.

If you expect the application to load for longer than the standard waiting times, you can also adjust the maximum time Tosca should wait in the settings. To do so, go to the setting **"Maximum Wait Duration"** under Project>>Settings>>Engine>>WaitOn. Adjusting the WaitOn settings gives the TestCase a higher degree of flexibility and resilience.

And this brings us to the end of our lesson. You are now able to differentiate between the available ActionModes "Select", "Verify" and "WaitOn", and also apply them to steer the controls in your System Under Test. Give it a try by completing this exercise.

Lesson 16 - Run your tests

In previous lessons, you have learned to populate your TestCase with TestSteps.

Now it's time to see if your TestSteps, TestCases, and TestCase Folders are working as intended. You can trial run them in the **ScratchBook.**

By the end of this lesson, you'll be able to run your TestCases in Tosca ScratchBook and effectively see and use the run results. We'll first take a look at what the ScratchBook is, and why you would be using it. We'll then switch to practical examples of using the ScratchBook within Tosca, and we'll take a look at where within Tosca you'll use the ScratchBook and how.

The ScratchBook is a Tosca built-in capability that allows you to **perform trial runs of your TestSteps and TestCases.** Such trial runs create only a temporary result. This **temporary result** is not saved and will be overwritten once you run a new TestCase or TestStep in the ScratchBook.

Nonetheless, this is particularly helpful when you build a new TestCase or when you simply want to check if a TestStep that you've just created is working correctly. This means that you'll use the ScratchBook while working within the TestCase section in Tosca. You can run the following objects in the ScratchBook: individual TestSteps, TestCases, or TestCase Folders.

There are a few ways to use the ScratchBook. The first is to **run test objects directly.** To do so, right-click on single or multiple Folders, TestSteps or TestCases, and click "Run in ScratchBook". Alternatively, after selecting the objects needed, simply press the **"F6"** key. The second way is to **run your objects via an already opened ScratchBook.** To access the ScratchBook, use the **shortcut "CTRL + B"** or click on the ScratchBook icon. Here, you can add the objects you want to quickly test in any order. To do so, you can drag and drop only the relevant TestSteps onto the ScratchBook. Then you can right-click and choose "Run" or press "F6" to run. If you only run a section of a TestCase, make sure the SUT is in an appropriate state, for example, already opening the correct web page.

After execution, the results will appear in the ScratchBook. If a TestStep fails, a **red X** appears next to the step with log info, explaining what the error was. When everything runs correctly, the steps are marked with a **green check mark.**

Within the ScratchBook, you can right-click on a TestStep, for example, a failed TestStep, and select **"Jump to TestStep"** or press **"CTRL + J"** to get back to that exact TestStep. You can call up the ScratchBook at any time. It will show you the results of your last run if you haven't closed Tosca Commander in the meantime. However, as the log is not saved, once you close Tosca Commander, the results will be lost.

And that's it for this lesson. You are now able to run your TestCases using Tosca ScratchBook and effectively see and use the run results. Give it a try by completing this exercise.

Lesson 17 - ActionMode Buffer

In previous lessons, we've covered some of the ActionModes you can select in Tricentis Tosca.

In this lesson, we'll continue with the **ActionMode Buffer**. The goal is for you to be able to use this ActionMode properly.

We'll first look at why it's necessary to use the ActionMode Buffer and what it does. We'll then, as usual, show you where to select it and how. An important part of testing is making sure to save and re-use values that the System Under Test has provided you. This makes the testing and verification as flexible and resilient as possible.

For example, ordering a product in the Demo Web Shop will generate an order number. For subsequent TestSteps, it's then important to save both the order number, as well as the price of the product. Instead of using hard-coded data, you can save the generated values by using a Buffer. This would allow you to **use the safe values generated by the system in any subsequent steps.**

Tosca uses the ActionMode Buffer to save any type of value generated during the execution of the TestCase. It can either be **static or a dynamic value**. This value will be temporarily saved in Tosca and can be used later on in the same TestCase. As with all ActionModes, also the ActionMode Buffer can be selected within the TestCase section in the column **"ActionMode".**

Let's look at an example for the ActionMode Buffer in Tosca. Let's say you want to store the price of the blue jeans you've purchased for verification later on. In order to do that, you have to find the control which has the property you want to save.

In this case, it's the container "PriceBlueJeans's' InnerText". Most objects have **default properties** and behavior, for example, Tosca knows that Textboxes have Text, and it treats them accordingly. Not all objects will have default properties. This applies, for example, to DIVs or any other type of container. Because this is a **container**, you have to explicitly tell Tosca to use the **InnerText** of the control for buffering. You can then choose a name for the Buffer. This is a variable name, behind which you can store the value to recall later on.

It's Best Pratice to give **recognizable names** to the Buffers. However, it is essential that when we're using it later on, you type the name exactly as you have defined in the Buffer, with the **exact syntax and capitalization.**

Here, you can use the name "PriceBlueJeans" with capital P, B, and J, and no spaces. You can then set the ActionMode to Buffer. If you run the TestStep in the ScratchBook, you will get a message saying that the Buffer was stored.

It's important to remember that every time you run that TestStep, the value will be **overwritten by the new value**. If it's changing in the application, the Buffer will also change. To recall a Buffer value, you can use the syntax **"{B[BufferName]}".**

In this example, you will have "{B[PriceBlueJeans]}". Thanks to this expression, you can now call up a Buffer value either to verify, or to input it somewhere else in the System Under Test. To see the Buffers directly in the WorkSpace view, navigate to the **Tool section on the Ribbon.**

Here, you can find the **"Buffer Viewer"**. In this window, you can see all the Buffers stored in Tosca. On the left is the **Buffer Name**, and on the right is the **Buffer Value**. Not only that, but you can **create a new Buffer** and edit existing ones by clicking on them as well. You can also **delete a Buffer** you no longer want by pressing the "Delete" button on your keyboard. You can also do the same by clicking on "Project>>Settings>> Engine". Here, you can also see a list of the Buffers created. You can create a new one, and also delete them.

Please note, the Buffers should **only be created and used within the same TestCase** to ensure that the same value is used. Additionally, Buffers are only saved locally, so you cannot expect different uses to pass buffered values to each other across multiple runs.

To share values across tests and testers, Tosca uses the Tricentis Test Data Service. You can find out more information about Test Data Service in the course "Test Design Specialist Level 2".

And this concludes this lesson. You are now able to use the ActionMode Buffer properly. Test your knowledge with this exercise.

Lesson 18 - Dynamic Values

By the end of this lesson, you'll be able to steer the System Under Test using basic **Dynamic Expressions.** More specifically, you'll get to know what Dynamic Expressions are, where to use them in Tosca, and how. But first, let's begin with why you would need to use Dynamic Expressions in Tosca.

In previous lessons, you've learned to populate your TestCases with static data. However, in real life testing scenarios, you don't always have or want to use static data in your TestCases. In many situations, you'll need to use **Dynamic Data** such as **Date, Time, or Random Values.** This is where Dynamic Expressions come into play. You'll use Dynamic Expressions if you need to specify values in your TestCases that are not generated until those particular TestCases are executed. There are multiple Dynamic Expressions that you can use. The Tricentis Tosca Manual covers all of them. Make sure to refer to it whenever you have questions. In this lesson however, we'll start with some basic Dynamic Expressions. We'll cover **MATH Function, Date, and Dynamic text.** You can use Dynamic Expressions at both in Module and TestCase level. Mostly, you'll use them at the TestCases level.

Let's see how to work with Dynamic Expressions in TestCases. They **generate special characters and dynamic value** you'll need to use the following syntax in Tosca. You'll need a command and if needed parameters.

Let's start with an example for MATH Functions. This function allows you to **perform calculations**. The syntax for the MATH Expression is shown on the screen. As an operand, you can use any numeric value, and as an operator, any basic arithmetic operation, plus some additional ones. Please see the Tosca Manual for a full list of operators. Simple examples of the MATH Function as shown here. In the first example, we've multiplied the price of the blue jeans by 25. In the second example, we've added 10 to the price of blue jeans.

Let's now see how to generate Dynamic Dates. The syntax for this is shown on the screen, consisting of an expression, and three parameters. In the syntax, each parameter is case sensitive, make sure you've got your capitalization in place. Let's now look at the syntax closer. The expression is used to indicate the date value you want to get. It is based on your current system settings. **Expression values** are for example, **"DATE" and "DATETIME".**

Each expression would need to be followed by three pairs of square brackets to indicate the base date, the offset, and the format. The Basedate is the date value in Tosca format. If you don't enter anything into this bracket, Tosca will use the default system date. If you enter a value, Tosca will calculate and format the result using this value as the base. **Offset is the value that is added or subtracted** from the base date. For example, "+2y" means two years in the future.

If you don't specify the offset, Tosca will use the base state with no offset. **Format** is the format of the output of the data expression. This would typically be the format needed to input or verify in the System Under Test. Possible formats include: **"d"** - which stands for day, capital **"M"** - which stands for months, and **"y"** - which stands for years. If you don't set the format, Tosca will use the format defined in your system settings. Here are a few examples of how to set up a syntax for Dynamic Dates.

The first example is four months from today's date, with the month being displayed as two digits. The second example is two years from today's date, with the year being displayed as four digits. You can do the same with text and numbers. The expression **"RANDOMTEXT"** allows you to specify a string length. The generated text may contain both randomized numbers and letters. If you want to generate a random number, you will use the expression **"RND"**. In between the brackets, you only need to specify how many digits this number should consist of. For example, here I wanted a 10-digit long number. You can also generate a number within a specific range by adding the upper and lower limits of the random number.

To check if your syntax is correct, simply right-click and choose **"Translate value"**. If it's correct, this command will turn your expression into a value. However, please note that this value won't be the same as the one you will get at runtime. When using Dynamic Expressions, keep in mind that Precise values Special characters, and Dynamic values can be combined with one another. Go to the Support Manual to find out more.

And this brings us to the end of our lesson. You are now able to steer the System Under Test using basic Dynamic Expressions. Take a look at these exercises to test your knowledge.

Lesson 19 - Create Libraries

Up until now, you've managed to create multiple TestCases by duplicating existing ones and adapting those to fit different testing scenarios. This is one method to create TestCases quickly, but it has its limitations. And even more efficient method is using Libraries and Reusable TestStepBlocks.

By the end of this lesson, you'll be able to work with and create **Libraries** and **Reusable TestStepBlocks** in Tosca. We'll first look at what Libraries and Reusable TestStepBlocks are, and why you should use them. Not only that, but we'll also look at where in Tosca, you can create these two types of test objects and how.

You can imagine that as you create more TestCases, there will be certain steps or sequences of steps, which you may need to use more than once. As mentioned earlier, you could **duplicate existing TestCases** and adapt them to fit different testing scenarios, but this is not Best Pratice, as it creates duplicated artifacts.

This makes it a lot harder for you to maintain your TestCases. Whenever you want to change one single value, you have to change that value in every single TestCase. You could also create TestSteps from scratch, but this is also not sustainable. Rather than wasting time on creating these steps from scratch every time, you can create a Library in which you can **store these steps.** The benefit is that whenever you want to use and reuse these steps later on, you can simply drag and drop them where you need them.

In Tosca, this Library is called a **"TestStepLibrary"**. The steps inside the Library that you can reuse are called **Reusable TestStepBlocks**. TestStepLibraries contain any number of reusable TestStepBlocks, which then can be reused throughout numerous TestCases.

This is why you should add steps to the Library that will remain unchanged in any TestCase in which they will be used. As the name TestStepLibrary and Reusable TestStepBlocks already suggests, you'll be working with these in the TestCase section. Let's take a look at an example of how you can create Libraries and Reusable TestStepBlocks. We'll start with a simple example. In the automation of TestCases it's Best Pratice to start and end the TestCase at the same place. In this project, we're always beginning by opening the browser, navigating to the Web Shop URL, and logging in.

We'll 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, you can simply **add these TestSteps to your TestStepLibrary**. But these are not the only TestSteps you can reuse. There are other steps as well. We'll look into those and create a TestStepLibrary and add to it the relevant Reusable TestStepBlocks. 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 or money order. First, you will need to create the TestStepLibrary. You can add a Library to any TestCase folder that does not already contain a Library.

This is very simple; you simply right-click on the folder and select "Create TestStepLibrary". You can also use the **shortcut "CTRL N + CTRL L".** To create Reusable TestStepBlocks from the TestSteps you have already created, simply drag and drop them from their original location to the Library. You will see that these TestSteps immediately become **References** to the 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 in 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.

And this brings us to the end of our lesson. You are now able to create and work with Libraries and Reusable TestStepBlocks in Tosca. Have a go at creating your own with this exercise.

Lesson 20 - Use the Libraries

So far, you've learned to create Libraries and ReusableTestStepBlocks. But, how can you use them? Additionally, what do you need to do if you want to edit a step in a TestCase that is linked to a Library?

By the end of this lesson, you will be able to create a TestCase using Reusable TestStepBlocks, and modify a linked TestCase.

Now, let's look at what you can do with objects that you created in the Library, and where to use them. We'll also look at why you might need to unlink your Test Objects from the Library and how to do so. After creating a Library that contains Reusable TestStepBlocks, you no longer need to copy and paste TestCase Objects. Instead, you only need to add the blocks to the TestCase. Tosca will create a reference to the Library. All these Test Objects, that is, Libraries, Reusable TestStepBlocks and TestCases, can be found in the TestCase section of Tosca. After you have created a Library and Reusable TestStepBlocks, the blocks can be added to a TestCase in the same way as a Module.

You can **drag and drop** them onto the TestCase and then rearrange them. You can also use the **shortcut "CTRL + T"**, to bring up the menu to search and add the items to the TestCase. Once the linked blocks are in the TestCase, they will show up as a Reusable TestStepBlock Reference. The name of the block also has **"_Reference"** at the end.

For example, once you have added the Reusable TestStepBlock **"Precondition",** you will see the Reusable TestStepBlock Reference with **"Precondition_Reference"** in your TestCase. Sometimes, you want to change something in one TestCase that is linked to the Library. However, if you change the Reusable TestStepBlock itself, the change will be applied to every TestCase that uses this Reusable TestStepBlock.

Just for this one TestCase, you can remove the link from a specific artifact to the Library. Simply right-click on the Reusable TestStepBlock Reference and select **"Resolve Reference".** Please note that, once you have resolved a Reference, you cannot relink it back to the Library.

Let's take a look at an example of how to add Reusable TestStepBlocks. Using drag and drop, you can add the Precondition and Postcondition blocks to the TestCase and, create a Process folder for the future steps. Then, you can bring the Reusable TestStepBlock References into their correct location.

Next, in the Process Folder, the Reusable TestStepBlocks can be added using the **"CTRL + T"** key combination. The Reusable TestStepBlocks are at the top of the list, making it easy for you to access them. As the TestStep Shopping Cart Procedures will be modified in a later lesson, you will need to resolve the Reference of the Folder Start Checkout_Reference. To do so, right-click on the folder and choose "Resolve Reference". The folder name is now Start Checkout and you can make edits to the TestSteps inside it without needing to change the Reusable TestStepBlock.

And this brings us to the end of this lesson. You are now able to create TestCases from Reusable TestStepBlocks in a Library. You can also modify a Reusable TestStepBlock Reference using the Resolve Reference function. Try this exercise to test your knowledge.

Advanced Modules

Lesson 21 - Advanced Module Actions

There is still a lot to learn about the TestCases, but before doing so let's make a quick detour to take a look at the **Advanced Module Actions in the Module Section**. By the end of this lesson, you will be able to Rescan existing Modules, merge two existing Modules into one by using the functionality "Module Merge", and you'll also learn how to manage the column "ValueRange" for Modules. To achieve all of this, we will look into what the "Rescan" and the "Module Merge" functionalities are, why they are important, and how to use them. We will also look into where you can find the "ValueRange" column.

Let's start with the "Rescan" function in the Module Section. **"Rescan"** allows you to **change Modules instead of creating new ones** every time you need an additional Attribute. The major benefit of rescanning Modules is that you can reuse Modules, or **re-map controls** that can no longer be identified. For this to work, the System Under Test needs to be open at the page that was scanned initially. The **"Module Merge"** functionality, on the other hand, allows you to **combine two Modules together**. If Module Attributes are duplicated in the Modules, you can also link these Attributes and decide how you want to identify them. The main advantage of these two functionalities is that they provide an **alternative way to creating new Modules every time you need to add new Attributes.** When you already have your Attributes, you can also go to the Module Attribute level and add values to the **"ValueRange" column**. The values added there will then create a list of Attributes that can later be chosen from a dropdown menu in the TestCase section.

Let's see a practical example of how "Rescan" and "Merge Module" work in Tosca. For example, to apply a discount code to your order, you have to enter the code on the "Shopping Cart" page, since you don't have the discount code text box yet in the "Shopping Cart" Module. Because of this, you now have to enter a discount code; for example "AutomationDiscount2". A text appears letting you know that you have applied the discount code. You can now go back to the "Shopping Cart" Module, right-click, and select "Rescan". This will now take you into the XScan, where you can simply change your Module by adding new Attributes. Now, you can select the controls that you want to add, and click "Save". In the "ValueRange" column for the Discount Code, you can add a few different discount code options. In this case, the possible discount codes to enter. When a Module is being rescanned, it can be saved as a duplicate in the Module section.

To avoid duplication, you can use "Module Merge", which brings the Attributes you select and their methods of identification together into one Module and deletes the second Module. To do this, you first need to select both Modules, then navigate to the "Modules" tab in the top ribbon, and click on "Merge Modules". After clicking on "Show", the two Modules will appear as the Target Module and the Source Module. Below them, you will see their **ModuleAttributes**. When working with "Module Merge", please keep in mind that the **Target Module** is the one you will keep, and the **Source Module** is the one that will be combined into the Target Module and subsequently deleted. You need to select the Attributes you 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 as separate Attributes 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 that 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 both of them. Now, when you are ready, press "Merge". Tosca will let you know how many Attributes have been updated and which usages have been relinked to the new Module. Then, it will delete the Source Module. Now, you can press "Close" and save your work.

In addition to merging Modules on your own, Tosca will also try to **automatically merge Modules** that it detects are the same when you import Subsets. These could be things like **duplicate copies** of Standard Modules or manually created Modules for your SUTs. Merging these Modules ensures that you don't have different copies of the same Modules being used throughout your TestCases. Having duplicate Modules can create extra work when it comes to upgrading your Modules. When you **import a subset**, you will be prompted to either "Merge", "Import", or "Cancel". Selecting **"Merge"** will automatically merge the similar Modules it has detected.

On the other hand, **"Import"** will simply bring everything in as is, and not attempt to make any changes to existing Modules. **"Cancel"** will of course, cancel the import all together. Now, if you're importing a very large Subset and are not sure if everything should be merged, it may be better to just import them and then merge any duplicate Standard Modules manually.
More than likely, this will be handled by your Test Architect, but still, you should be aware of the process.

And this brings us to the end of this lesson. You are now able to rescan existing Modules, merge Modules and also manage the column ValueRange for Modules. Give it a try, by completing this exercise

Lesson 22 - Self-defined Test Configuration Parameters

To run a TestCase, you may want or need to specify values such as the desired testing environment, test object versions, business chain identifiers, and so on. To do this, you could use a **pre-defined Test Configuration Parameters**. That is, a **TCP**.

In previous lessons, we used the TCP "Browser" to tell Tosca which browser to use to run the TestCase. Normally, using a TCP is possible when the parameter to recall is included in the list provided by Tosca. But what if you need to use one that is not included?

In this case, you can use self-defined TCPs. How to create and use **self-defined TCPs** is the objective of this lesson. So let's get started!

We'll take a look at what self-defined TCPs are, where and how you can use them in Tosca and, last but not least, why do we recommend using them.

You may need to use a very **specific parameter to run your Testcase**. For example, your own username and password. You can do so by creating a self-defined Test Configuration Parameter.

While both TCPs and Buffers are used to generate values, they're different. A Buffer is generated during run time and is stored locally. On the other hand, **TCP values are generated by a tester and are stored within the repository,** even before the TestCase is run. Just as with any pre-defined TCP, you can also create and use a self-defined TCP on all Tosca objects that have a Test Configuration Tab.

Let's see how to add a TCP and also look at how you can create your own self-defined TCPs.

For this, we'll work with an example in the Demo Web Shop. The purpose of this TestCase is to apply a discount code. To verify the calculation of this discount, you can set the value for this code, AutomationDiscount2, and the value of the discount. To do so, select the Test Configuration Tab, right click on the object, and choose the option for adding a TCP.

Next, create the desired TCPs, in this case, the Discount Name, and the Discount Percentage, and set the relevant values. When you want to recall these exact values in the TestSteps, just add a Reference to these TCPs. The syntax for this is **{CP[Parameter]}.** The benefit of using TCPs and self-defined TCPs rather than typing the values directly in the TestStepValue is that this **lowers maintenance.**

If in the future, you need to change the value, you only have to do so in one place, rather than each and every time it is used in your TestCase.

Now that we've reached the end of this lesson, you are able to create and use self-defined Test Configuration Parameters. Give it a try by completing this exercise!

Lesson 23 - Business Parameters

So far, you've already learned how to reuse TestSteps by using a TestStepLibrary.

Until now, any modification made in the reusable TestStepBlocks were also applied to all References to these Blocks in your 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 that allows each tester to **use their own user data.** The solution lies in using **Business Parameters.** By the end of this lesson, you'll be able to create and use Business Parameters. You'll achieve this objective by looking at what Business Parameters are and why you should use them. In the second part of this lesson, you'll see where to use Business Parameters and how. Business Parameters allow you to modify values in the Reusable TestStepBlocks References without changing the Reusable TestStepBlocks in the Library. They are the business-relevant elements of your TestStep, whose values you want to steer in your TestCase.

The benefit of using Business Parameters lies in **easier maintenance of your TestCases.** They allow you to reuse the structure of your TestCases set in the TestStepLibrary while using your own data. As you can already tell, you'll be using Business Parameters at the TestCase Level.

Let's put this into context and see how you can make the best of using Business Parameters. In this example, you will want to reuse the Precondition of opening the URL and logging into the website.

Let's assume though that you want the **input data for URL, Email, and Password to change,** depending on which TestCase you run. You 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. **"F11"** can be used to show or hide values.

There are two ways to create Business Parameters. The first is to **create them from scratch**. In the Library, on the Reusable TestStepBlock that you want to add Business Parameters to, right-click and select "Create Business Parameters Container". Then simply create Parameters as needed inside this used Business Parameters Container.

Once the Parameter is created, you need 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, which will be **{PL[ParameterName]}.** There are cases when you already have the Parameters defined in the TestStep.

In these scenarios, you can simply drag and drop them into the Business Parameters Container, and they will create the References in the "Value" column as well as in the BusinessParameter itself.

And this brings us to the end of this lesson. You are now able to create and use Business Parameters.

Congratulations! You have now completed the TestCase section! Give Business Parameters a try by completing this exercise!

ExecutionLists

EXECUTIONLISTS

Lesson 24 - ExecutionLists

In this video, you're going to learn about ExecutionList. After watching this video, you will be able to create an ExecutionList, link your test cases to the ExecutionList, and interpret the Execution results.

Once your TestCases are complete, you no longer want to run them in the ScratchBook, but in the actual ExecutionList. This is why in this lesson, we're going to see what ExecutionLists are, why are they relevant?, where can you find them in Tosca?, and how can you use them?

So far, you've run your TestCases in this ScratchBook, but we recommend using ScratchBook for trial runs only as results do not persist. The ScratchBook is a quick way to check if your TestCase will work. For a proper execution of your test cases, we recommend running them in the Execution section by using ExecutionLists. But why is that?

ExecutionLists do the important job of **running the test cases and storing the results** for the executed TestCases. Unlike this ScratchBook, the ExecutionLists store all historic results, which can be consulted if necessary.

Another advantage is that if you have cases that take longer to run, you can use ExecutionLists to **run them unattended** and even overnight. The ExecutionList can be found in the Execution section of Tosca. There, you will be able to organize your elements in ExecutionList folders and ExecutionEntry folders.

Within this, you will have your ExecutionList, which contains a series of **execution entries** that need to be run. Please keep in mind that each TestCase is represented as an ExecutionEntry and can only exist once within each ExecutionList. Let's see, now how you can run the TestCases in your ExecutionList.
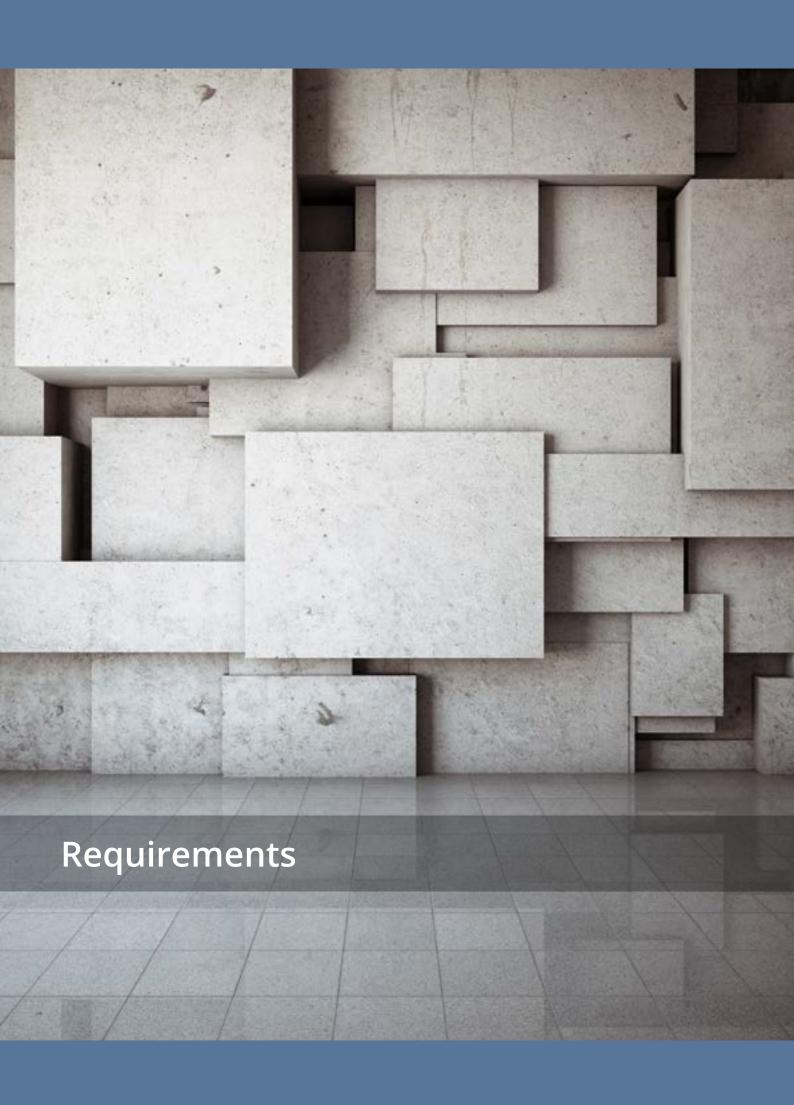
First, you will create the ExecutionList by **dragging and dropping** the TestCase from the blue section onto the corresponding ExecutionList in the green section. If you drag and drop individual TestCases into the ExecutionList, you will have to add or delete any new TestCases manually into the ExecutionList. If you drag and drop a TestCaseFolder, you can simply use right-click **"synchronize",** on the ExecutionList to update it 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 test case more than once you can modify the property repetitions of an ExecutionEntry folder or an ExecutionEntry itself. There are three options for executing the ExecutionList. First we have **"Run"**, which automates the TestCases in the ExecutionList. Then we also have **"Run from here"**, which will start running from the selected ExecutionEntry and continues until the end of the ExecutionList. And the last one, which is **"Run as Manual TestCase",** which as the name says will 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 this and choose **"Disable"** from the Context Menu to exclude them from execution. If Test Configuration Parameters set both on the TestCases and the ExecutionList, the TCP set on the ExecutionList will override the ones set on the TestCase. Unlike in the ScratchBook, if a TestCase fails in the execution, your Recovery Scenarios and CleanUp Scenarios will kick in when necessary to continue running the executions. You can learn more about these scenarios, in a later lesson. If an Execution is rerun, the most recent information will appear. You can expand the ExecutionEntry to see all the previous runs.

When the Execution passes a **green box with a green tick** will appear. When an execution in test fails, a **red box with a white arrow** inside will appear. In this view, you can see the duration of Executions as well as a summary of the logging info. You can also manually right-click on the ExecutionEntry and select set result to **"Passed", "Failed",** or **"no Result".**

And now, you are able to create, link, and interpret your ExecutionLists and your results.

Congratulations! You have almost completed your Automated Testing Journey. Let's take a look at the last section. Give it a try with this exercise.

Requirements

REQUIREMENTS

Lesson 25 - Requirements

We'll start this course section by learning how to grade Requirements, Requirements Sets, and Folders. This will be the objective of our lesson. We'll therefore take a look at what Requirements are, and where and how you can manage them in Tosca. Then, we'll also look into why you should organize your Requirements properly.

Requirements are criteria that the System Under Test is expected to fulfill. This criterion **refers to the business requirements** that you need to check to ensure that your application is running properly.

They can be Functional and Non-Functional. **Functional Requirements** cover those functionalities that the System Under Test must be able to carry out. For example, an online Web Shop must be able to process payments. On the other hand, **Non-Functional Requirements** relate to the operation of the application rather than a specific use case, for example, security or performance testing.

In Tosca, Requirements are organized in **Requirements Sets,** which represent different aspects of the application. For example, you will have a RequirementSet for the front end of your website. This will help to keep your Requirements organized and easy to maintain. You can also go a level up and structure your Requirements Set in Requirement Folders.

Now, let's see how all of this looks in Tosca.

You can manage all of these data and build your structure in the **Requirements Section**. This section provides a dashboard for simple and effective test management. It is used for planning test processes, monitoring progress, and optimizing risk coverage.

Let's see now how you can build your Requirements structure beginning at the top level.

We'll start with **Requirements Folders**. To create a new Requirements Folder, right-click on a folder in the Requirements Section, and select "Create folder" from the mini toolbar. You can also click on "Create Folder" in the Requirements dynamic ribbon menu. Or use the **shortcut "CTRL N"** and then **"CTRL F"** after clicking on the folder where you want to create the new folder.

To create a **RequirementSet**, right-click on a Requirements Folder and select "Create RequirementSet" from the mini toolbar. Last but not least, to create a Requirement right-click on a RequirementSet and select "Create Requirement" from the mini toolbar.

And this is how you can structure your Requirements in Tosca. We've reached the end of this lesson and now you're able to create Requirements, Requirements Sets and Folders.

Lesson 26 - Weight Requirements

You might have the Requirements in place, but this alone does not tell you what to test first.

To get there, you will learn in this lesson how to **weight Requirements** in Tosca. We'll look at what does it mean to weight Requirements, and where and how to do so in Tosca. But first, let's start by looking into why we need to weight Requirements.

Prioritizing your TestCases is crucial for your testing process. Why is that? Some Requirements **carry more business risks** than others. This means, that if they were to fail, the impact for the application and the business overall will be greater than in most other cases.

For example, if the Demo Web Shop would fail to allow you to pay for your ordered items, the risk associated with that error would be much higher than if the shop wouldn't offer the possibility to sign up for a newsletter. To be able to test what's most important first, you need to assess the risks related to each Requirement.

The **Pareto principle** says that **80% of your risk** comes from **20% of your application.** If you take a risk-weighted approach, you can prioritize your testing to cover most of the risks. To make sure that you are taking a risk-based approach, you can specify the level of risk and importance of all aspects of your project.

In Tosca terminology, that means that you can weight your Requirements. This will allow you to **lower the risks** that are released of your application will result in losses or damages. Tosca allows you to do just that! You can weight your Requirements in the **Requirement Section.**

Let's now take a closer look at that, and also see how you can assess the risks.

In Tosca, the weight for each Requirement is calculated based on two values. The **"Frequency class"** and the **"Damage class".** The **"Frequency class"** column shows you how often each Requirement is expected to be used. The more frequently it will be used, the higher the weighting.

The **"Damage class"**, on the other hand, is related to the degree of financial risk associated with each Requirement. These weightings typically will come from the business analyst. Weights are scaled for both frequency and damage classes, you can enter integrates between 1 and 10. In this example, we will, however, use a range between 1 and 5. The weight is then calculated automatically using an exponential function to show the relative importance of each Requirement, to all others at the same level.

In this way, we know where to test first and we make sure to maximize risk coverage. 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", the step "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 to each Requirement in a RequirementSet, according to its and its parent's weight.

And this is how you can take an informed decision on what to test first, which leads us to the end of this lesson That means that you are now able to weight Requirements in Tosca!

Lesson 27 - Link Requirements

Up until now, you have learned to structure and weight your Requirements in Tosca. But what is all this information telling you, and how can you use it to prioritize your testing?

To answer this, you'll need to go a step further and link your TestCases and ExecutionLists to your Requirements. And this is what you will learn today. More specifically, you'll learn what does it mean to link TestCases and ExecutionLists to Requirements, where and how to do so, and what's the purpose of it. Why should you link your Requirements? Let's start from the back.

By **linking your TestCases and ExecutionLists to your Requirements,** you will get an overview of the progress and the state of your entire project. Remember, no TestCase should be created that **does not cover a Requirement.** This will also allow you to see the execution results more vividly, that is, which TestCases that were linked to high-risk Requirements, passed or failed in Execution.

To be able to do so, you need to open the following sections in Tosca: Requirements, TestCases, and Execution. Let's see how it's done!

We'll start with linking TestCases first. To link **TestCases to Requirements,** 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.

As mentioned before, it is also crucial to link your **ExecutionLists to the relevant Requirements**. To do so, just drag and drop the ExecutionLists, which are related to the TestCases you linked earlier, to the RequirementSet. In this way, Tosca will automatically link them and display the Execution Results along with the corresponding Requirements.
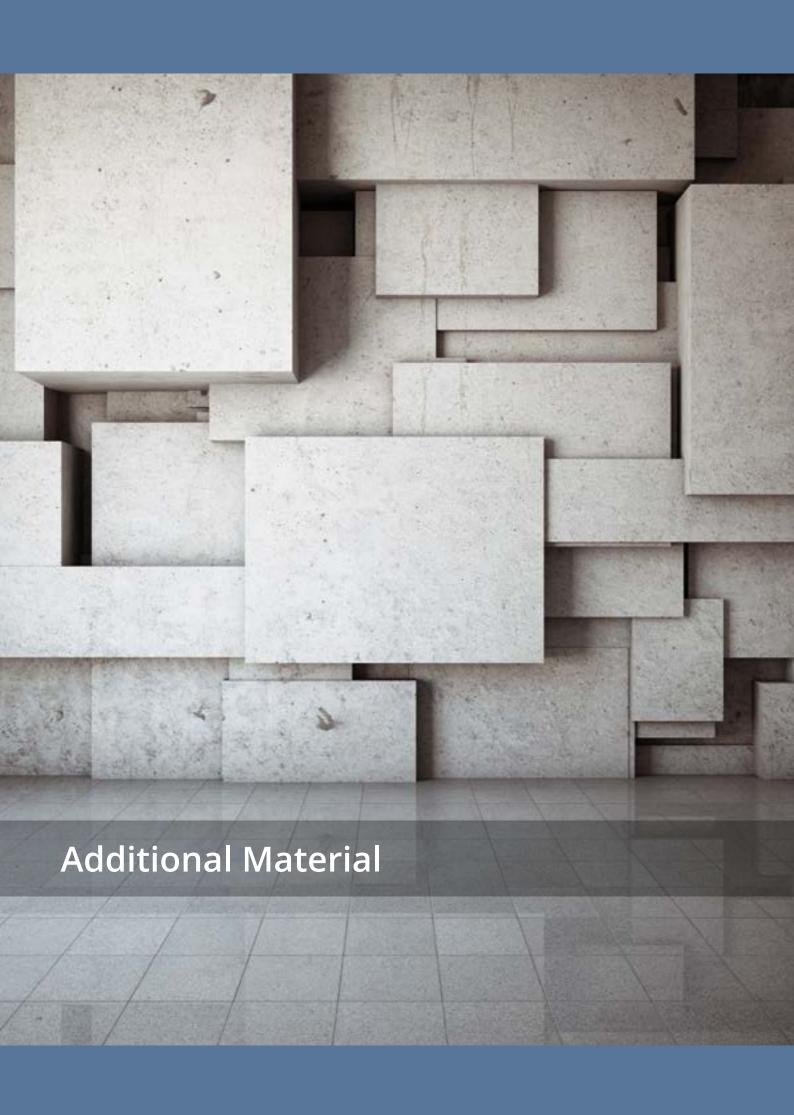
By doing this, we will be able to have an overall view of the state of our project. We can see how much of our Requirements have been tested in the **"Execution State" column**, and how much of our Requirements have been specified in TestCases in the **"Coverage Specified" column.**

And that's how you link your Requirements to the relevant Tosca objects, be it TestCases or ExecutionLists. This marks the end of our section. Let's revise our objectives for this section.

You are now able to create RequirementSets, weight your Requirements and link these to your TestCases and ExecutionLists.

And you are done! Congratulations! You have completed all five sections of the Automated Testing Journey in Tosca. Do not forget to check the lessons of the more advanced features.

Now, put everything you've learned in this section into practice by completing this exercise.

Additional Material

ADDITIONAL MATERIAL

Lesson 28 - Working in the Multiuser Environment

This lesson is all about working in multi-users environment in Tosca. By the end of this lesson, you will be able to login to a mult-iuser environment and use the checkin and checkout functions to lock test artifacts while working on them.

We will first see what a multi-user environment is, why it is relevant for a team of testers, where you can create a connection to an existing multi-user environment, and finally, how it allows multiple users to work in the same repository. So, let's dive right into it.

A **multi-user environment,** also called common repository, is used when several people need access to the same repository. Having a multi-user environment is a great advantage for any testing project. It **allows different testers to access the same workspace at once,** to **share artifacts and requirements** smoothly. Additionally, thanks to the checkin and checkout functions, this can be done free of conflicts. You can create a connection to a common repository through the "**Create New**" repository workflow. Once the connection is made, you can access this repository just like any other repository on your machine. Let's see now how working in the multi-user environment looks like.

From the "Create New" repository screen, select the database that is associated with your multi-user environment from the "Type of Repository" drop-down. You may need to get this information from an Administrator, along with the connection information. Once you've connected to the workspace, you will be prompted to login with your username and password that was created by your Test Architect.

In your workspace, navigate to the Home section, and click on "Update all" on the Ribbon. This will refresh your workspace and will **reflect all the information** entered in the common repository since your last update. You can see if others are working on something when you go into your workspace. For example, here, in the Modules Section, you will see a **red icon**, which means that someone has **checked out** this object and is working on it. We see that also the subsequent objects have been checked out because they also have a red icon. When something has been checked out, the red icon indicates that you will be **not able to do any changes** or add any new Modules to any folders while they have this icon. The object is, in other words, **locked.**

However, even when an object is checked out, **you can still use it.** You can, for example, drag and drop the checked-out Modules to your TestCases, or run TestCases in the ScratchBook. You can also execute the TestCase in the Execution List. If you want to get a better overall view of the project as a whole and what's currently locked out, you should try using the "Project" tab. By default, this tab isn't there when you create a project, but you can simply add it by going to the "Home" section of the toolbar, and clicking on "Project".

This will create a new window that you can drag and drop on the bar with your other Tosca sections. The "Project" view **shows all Tosca Sections** under one folder allowing you to see content in multiple sections at the same time. Let's see now a different example.

Say that you want to create a TestCase. To do so, you will need to right-click on this folder, and then click on the "Checkout" symbol. You can also do this by clicking "Checkout" in the ribbon. Now, **a green icon** will appear, meaning that **you have checked that object out** and you can now make changes to the object. For all checked out objects, you can also click on "Show Checkout Details", within the context menu to access information on **which user has the object checked out and for how long.**

This means that, while you are working on this object, the rest of the users **won't be able to change in this object**. For objects that are not checked out, you won't see this option. You can click again on the folder that you checked out, and from there, you can create a new TestCase. This new TestCase has a white icon next to it. White means that the object **has not yet been updated** to your Tosca project database.

Once you've finished working with the objects, you can **check them in**, which will **free them up** for other users to work on them. You can see here, that although these Modules are checked out by another user, you are still able to use them for your TestCase.

So remember, no matter if an object is being checked out or not, you will still be able to use it for drag and drop operations, such as adding a Module to a TestCase or adding a TestCase to an ExecutionList.

And this brings us to the end of this lesson. Now you are able to login to a multi-user environment and use the check-in and check-out functions to lock test artifacts when working on them.

Lesson 29 - WHILE Statement

In this lesson, you will learn about **conditional statements and loops** within Tosca. These are the WHILE, IF, and DO statements. By the end of this lesson, you will be able to explain what WHILE, IF, and DO statements are, and set a WHILE statement. In order to achieve these learning objectives, you will be going over the following questions: Why are these statements relevant? What WHILE, IF and DO statements are? Where can you use them? And how can they be set up in Tosca?

By now, you know that in Tosca, 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. These expressions allow to repeat a step if a condition is met, but also to go to the next step when the condition is no longer met. Let's begin with the WHILE statement.

A **WHILE statement** is a loop used for running TestSteps many times. It tells Tosca to **repeat a step as long as a particular condition is met**. When the condition is no longer met, the loop stops, and Tosca will move on to the next step. Aside from WHILE, there are also IF and DO statements. As mentioned 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.** However, Tosca will take a couple of seconds to perform the steps in each IF statement. If the IF condition is expected, we recommend using **TestCase Templates** instead, as it is Best Pratice and will save you execution time. You can learn more about this topic in our course Automation Specialist Level 2. Let's take a step back and take a look at the WHILE Statement.

In the Demo Web Shop, you tell Tosca to empty the Shopping Cart and start over with the verification process, in case the verification process cannot continue due to an error. However, since each item has a separate **REMOVE checkbox,** and the number of items in the cart is dynamic, you cannot simply create a TestStep. Instead, you can create a WHILE statement with a loop. Tosca will check if there is an item in the cart, and then remove it. Once there aren't any other items in the cart, Tosca will move to the next step.

But where can you create these statements? You can create a WHILE, IF, or DO statement within a TestCase. Let's see how this works within Tosca and how you can set up the WHILE statement.

First, navigate to a TestCase, right-click and select "Create a WHILE statement" from the context menu. Just make sure to bring your WHILE statement in the **correct position** depending on the structure of your TestCase. The WHILE statement is made up of two parts: **the condition and the loop**. In the condition, you must tell Tosca which condition should be met for the step to be completed. In the loop, you will tell Tosca what to do if the condition is met.

Once you have created the statements, just like in other TestSteps, you can add the Modules you need for each statement and fill in the necessary TestStepValues. Remember, you can do this by selecting the TestStep and using the **shortcut "CTRL+T"** to see the list of Modules.

Following the example given before, you want to ensure that the Shopping Cart Table exists in order for the loop to kick in. You can then instruct Tosca to remove the last product from the cart in the loop. This means that every time Tosca sees the Shopping Cart Table exists, it will continue performing this loop until there are no more products. You can set up how many times the loop should be repeated in the Properties Tab. By default, the loop is set to be **repeated 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. If you wish to see a graphical representation of the flow or path of your TestSteps, click on the **"CTRL 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.

This brings us to the end of this lesson. You are now able to explain what WHILE, IF, and DO statements are, and set up a WHILE statement. Give it a try with the next exercise!

Lesson 30 - Recovery Scenarios

You have already learned about loops and conditions. In this lesson, you will see how to set up Recovery Scenarios.

By the end of this lesson, you will be able to create and use a Recovery Scenario. This lesson will help you meet this learning objective by answering four basic questions: Why are Recovery Scenarios useful and time-saving? What are Recovery Scenarios? Where can you apply these Scenarios in Tosca? And, how can you enable and create Recovery Scenarios for TestCases? Let's begin with why Recovery Scenarios are useful and time-saving.

You may have already realized that, as you ran the ScratchBook in previous exercises, problems can occur while running TestCases. For example, a random pop-up may appear, or a TestCase may not execute properly. Imagine if you had hundreds of TestCases to run! You would not want to react manually every time this happened. Instead, you can use a Recovery Scenario, which will create instructions for Tosca so **it will know what to do, and recover the TestCase.**

What is a Recovery Scenario? **Recovery Scenarios** are instructions that tell Tosca what to do in order to recover a TestCase that might have failed while running a test during the final execution. If a Recovery Scenario is run succesfully, Tosca will retry the TestCase according to the RetryLevel set. If the Recovery Scenario fails, Tosca will go into the next Recovery Scenario available.

Where in Tosca can you apply these Scenarios? Recovery Scenarios can be added into a **Recovery Scenario Collection,** which can be added anywhere within the TestCase Section. The collection will affect any TestCase below it in your Folder Tree. The level in which you have added the Recovery Scenario can be seen in the value called **"Retry Level"** and can be specified on the Property Tab of the Recovery Scenario. This can be either TestCase, TestStep, or TestStepValue level. This will tell Tosca, in the event of a problem, how far to go back to reattempt running the TestCase.

Please note that RecoveryScenarios will only kick in when the TestCase is executed from an ExecutionList, not in the Scratchbook. Now let's move to Tosca to learn how to enable and create Recovery Scenarios for TestCases. Before you get started, you will need to make sure that the **Recovery settings** are enabled. To do this, navigate to Project>> Settings>>TBox>>Recovery. The first three define if "Dialog failures", "Exception failures", and "Verification failures" should be recovered.

Please keep in mind that **"Verification failures"** might point to actual errors in your System Under Test. The last three define how often specific Recoveries should be tried, according to the set RetryLevel. All those settings may be set as Test Configuration Parameters on TestCase or TestCase Folder level. These will then override the Recovery Scenario settings.

Once you have enabled Recovery, you need to create a **Recovery Scenario Collection** in your TestCase. You will do so by right-clicking on the level where you want the Recovery Scenarios, and then selecting "Create Recovery Scenario Collection" from the context menu. This can be found under the ellipsis button on the icon bar of the context menu. You can also use the **shortcut "CRTL+N+ CRTL+R".** Now you can right-click again on the Collection and create a Recovery Scenario. Make sure to set the RetryLevel on the Properties tab.

You can now add any steps you want into your Recovery Scenario. You can either drag and drop a TestCase or use the shortcut **"CRTL+T"** to look for any Module or Reusable TestSteps Folder you want to add. Recovery Scenarios can be filled with any TestSteps you'd like but, ideally you used them to get your SUT back to a state in which it can complete the TestCase. For example, if you tried to run your earlier TestCase examples while still logged in to the Demo Web Shop, you'll know that it fails because the first step is to log in, and Tosca can't find the "Log in" button. You can add steps to a Recovery Scenario that logs you out if the step fails, and then the TestCase will try again and continue. You'll set this up in the exercise that follows this lesson.

Looking at the LogInfo in the **ExecutionEntry** shows that overall, the TestCase passed, although when you look in detail, the first run through failed. Remember, Recovery Scenarios kick in only during actual execution in the ExecutionLists, not in the ScratchBook. And this is how you can work with Recovery Scenarios in Tosca!

You have reached the end of this lesson. You are now able to create and use Recovery Scenarios. Create your own with this exercise!

Lesson 31 - CleanUp Scenarios

Now that you have learned how to set up Recovery Scenarios, you will see how the rest of the test environment reacts if something goes wrong in the Execution by creating a CleanUp Scenario.

By the end of this lesson, you will able to create and use CleanUp Scenarios. This lesson will help you meet this learning objective by answering four basic questions: Why are CleanUp Scenarios useful? What are CleanUp Scenarios? Where you can apply these Scenarios in Tosca? And how can you set up a CleanUp Scenario?

CleanUp Scenarios are very useful when a Recovery Scenario is **unrecoverable.** If the Recovery Scenario fails, you want Tosca to reset and restart the SUT and put it into the correct status for next TestCase. What is it exactly? A CleanUp Scenario is like a Recovery Scenario, but instead of attempting to fix and re-run the TestCase, the CleanUp Scenario attempts to **reset the TestCase.**

In the event that the TestCase is not recoverable, Tosca can be instructed to **clean up the space for further TestCases** by the use of a CleanUp Scenario. After Tosca has executed all available Recovery Scenarios, and the TestCase still fails, it will clean up the testing environment so it is ready for the next test to run.

CleanUp Scenarios **can be added into Recovery Scenarios Collections** and will affect any TestCase that the Collection applies to. If recovery does not work, we want Tosca to try to reset and restart the SUT, then move on to the next TestCase.

Let's see how this works in our case with Demo Web Shop.

Since we know our TestCase begins by opening the browser and logging in, we want to tell Tosca to log out and to reset using a CleanUp Scenario that closes the browser. To do so, first you will need to create a CleanUp Scenario by right-clicking on a Recovery Scenario Collection, or pressing the **shortcut "CTRL N + Crtl C".**

Similar to what we have learned in previous lessons, you will be instructing Tosca to empty the shopping cart and close the browser. For this, you can use the "WHILE" statement you created previously in conjunction with the Module "CloseBrowser". Keep in mind that loops like "WHILE" do not have to go inside CleanUp or Recovery Scenarios, and should only be used when absolutely necessary. Actually, Recovery Scenarios can often be used instead of loops.

A Recovery Scenario will **attempt to recover the TestCase** the number of times we instruct Tosca to do it. If it is unsuccessful, the CleanUp Scenario will kick in making Tosca reset your System Under Test so the next TestCase can run. Remember, Recovery and CleanUp Scenarios will only be triggered when the **TestCase is run from an ExecutionList.**

Great! This is the end of the lesson for CleanUp Scenarios. You are now able to create and use CleanUp Scenarios. Practice what you have just learned with this exercise.

Lesson 32 - Dynamic Comparison

In this lesson, we will look at one of the advanced features of Tosca, called Dynamic Comparison. Once you have completed this lesson, you will be able to set up and use Dynamic Comparison.

We will start by looking at what **Dynamic Comparison** is and why it is used in Tosca. We will also see where it is applied, and how. Dynamic Comparison is a feature within Tosca that allows you to **verify a portion of a string that is static while buffering another portion of the string that may be dynamic.**

The buffered portion is also referred to as an **XBuffer** and it is represented by **{XB}.** XBuffer helps you to verify strings with dynamic elements as well as simultaneously buffering elements within the string, so you can utilize them later on in your Test Case. This is particularly useful when you're trying to **buffer an order number or a confirmation number** from a success message, while at the same time, verifying that the SUT displayed a static success message.

Not only are there examples of this wihtin our Demo Web Shop, but also other common tools such as SAP, Salesforce, or possibly even applications that your company has developed.

Dynamic Comparison is used at the **TestStep level by adding the Dynamic Comparison syntax in the TestStep Value box.** This can be utilized on any control with text that you would like to verify or buffer, such as a label for example.

Now, let's see now how this can be applied.

On the order successful page, we receive a message number with the string "Order number:" and then the number of the order that we just placed. The text "Order number:" is static, but the number itself changes. So, in order to verify that this message appears, we can't simply verify the entire string because the order number will be different each time we run the TestCase. So instead, we can use the XBuffer to **exclude the order number from the verification and buffer the order number** into Tosca at the same time, so that we can use it later on in our TestCase.

You want to buffer the Order number so that when you want to reorder, Tosca can **use that saved Order Number to find it.** So in order to do this, we can use Dynamic Comparison by typing "Order Number:" as well as a buffer name into the square brackets of the XBuffer syntax. So this way Tosca only stores the Order number in the buffer. So our syntax for this TestStepValue would then look like this: **"Order number: {XB[OrderNumber]}"**, and then you set the ActionMode to Verify. We do this because, Tosca will first verify that the static text exists, before saving the Buffer. And do keep this in mind, because that's why the XBuffer notation will not work if you set the ActionMode to Buffer.

If you didn't type in "Order Number: ", Tosca would store the whole string in the buffer. And similarly, if you were to use a normal buffer here, you wouldn't just buffer the order number, but the whole string. That's why both of these options would make it difficult to search for the Order Number later because you would need to figure out how to remove the "Order number" text from the Buffer after you've saved it to Tosca. And this is why you must use a **combination of text and the XBuffer to only buffer the portion of the string that you want.** And this brings us to the end of this lesson!

You're now able to set up and use dynamic comparison. So go ahead and give it a try, by completing this exercise!

Lesson 33 - Parent Control

Let's get into something more advanced.

In this lesson, you will learn about Parent Controls. And by the end of this lesson, you will be able to identify controls by their Parent Control. You will learn what a Parent Control is, why it is relevant in Tosca, where you can find it, and how you can identify control utilizing its Parent Control.

So far, you've learned to identify a control by its own properties and by an Anchor Control, but it is also possible to identify a control using its Parent Control. Simply put, the Parent Control is the control that is higher up in the object tree, and within which the control that we actually want to use is located. You can use tables and containers as Parent Controls, for example.

Parent Controls allow us to **identify objects that were previously not uniquely identifiable,** or unidentifiable entirely based on their technical properties alone. An example may be that we have multiple controls that may have the same properties but are located in different sections of our SUT. By identifying which section of the SUT the control is found in, Tosca can uniquely identify that control.

Let's take a look at the homepage of the Demo Web Shop. We have two menus containing the shopping categories: on the left, and at the top of the page. Let's say you want to select specifically the "Books" link on the left-hand side of the page. It would be really tough to uniquely identify that control because it is nearly identical to the control at the top of the page. But, if we identify the box that the books link is contained within as well, that is the Parent Control, we will be able to **uniquely identify** the desired shopping category. And yes, there are certainly other ways you can make this control unique, and we've covered these techniques throughout this course.

However, in this lesson, we're going to use the example with the control "Books", and show you how to identify this control by a Parent Control, so that you can compare different identification techniques, and decide for yourself when best to use them. But before diving into the example, let's talk about where you can manage Parent Controls.

Parent Controls are normally managed within the **XScan in Tosca.** When looking at the control tree and expanding the filter, you can easily find the parent that the control that you're trying to identify is nested under. Once you've identified which Parent Control you would like to use, you will need to make that control uniquely identifiable as well.

All right. Let's hop back into the Demo Web Shop and we'll show you an example of how to identify by Parent.

Once you've scanned the Web Shop, and you've selected the controls "Books", on the left, you can see that it is not uniquely identifiable, and no matter how many technical properties you select, it still doesn't become unique. But instead, you can drag out the filter bar to display additional controls in the tree. If you go three controls above the "Books" link, there is a "DIV". And this DIV contains all the product links in the menu on the left of the page. If you select it, the "Books" link will now become uniquely identifiable, because Tosca is now looking for a "Books" link that is specifically located within that DIV.

We've mentioned the term DIV here a couple of times, and you might be wondering what that is. **DIV is short for division,** and it's a type of object used by programmers on webpages to **segment the page into different sections.** DIVs can be very handy when steering webpages with Tosca, because they often groups similar controls together. But don't worry, you don't really need to know anything technical about DIVs, but feel free to use them to your advantage when using Tosca with webpages.

Congratulations, we've arrived at the end of the lesson, and now you're able to identify controls by Parent.

Lesson 34 - Dynamic Identification

In this lesson, we will discuss how to use Dynamic IDs within Module properties. And by the end of this lesson, you will be able to identify when a Dynamic ID should be used, and created Dynamic ID in the properties of that control. Specifically, we're going to take a look at what a Dynamic ID is, why you would want to use one, where in Tosca you will apply what you've learned, and how you can use Dynamic IDs to better steer controls in Tosca.

Dynamic ID is a term for a **value of a property on a Module Attribute** that can change during each distinct run, or even step to step. This can be achieved by using various dynamic expressions in Tosca, such as **Date, MATH, or Buffer Operations.**

Sometimes in your TestCases, you may want to steer a control that contains a value that changes every run and is based on a value generated earlier in the TestCase. For example, in the Demo Web Shop on the orders page, you will see the most recent orders placed. And if you want to view the details of your most recent order, it will be difficult to identify exactly which "Details" button to click on, as there are multiple identical "Details" buttons. But what you can do is identify the "Details" button of your order by the container in which it is located.

This container would be its Parent Control. You can tell Tosca to identify the container using the exact order number at the top of the list. However, you know that after you place more orders, this exact order will no longer be the most recent order at the top of the list.

If controls have dynamic properties, as in this case, you should reflect this by creating Dynamic IDs in the properties of that control. And, you can create Dynamic IDs by **editing the properties of your Module Attributes.** And for this, let's jump over the XScan.

If you look at the properties of one of these containers, you can see that they are identified by the exact order number. And, if you want to make sure that the Module is able to handle dynamic changes of the order number, you will have to identify the order using the buffer you created in an earlier step in the properties. When you run your TestCase, and received the success message containing the order number, you will buffer the place order. You can 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.**

Congratulations. You've arrived at the end of this lesson. And now, you're ready to use Dynamic IDs within your Modules to steer Dynamic controls. Go on and give it a try by completing this exercise.

Lesson 35 - ExplicitName

In this lesson, we will discuss how to use the Configuration Parameter ExplicitName.

By the end of the video, you will be able to set the ExplicitName Configuration Parameter on a ModuleAttribute, and use ExplicitName within a TestStep. More specifically, you will learn what ExplicitName does, why you may want to activate it within Tosca, where to activate it within your Modules, and how to use it effectively within your TestCases.

ExplicitName is a **Configuration Parameter** which can be added to a Module. It allows a **change in the TestStepValue name which reflects a technical identification,** such as the index of an object within a group of objects that share the same properties. For example, if you wanted to select the first, second, or third option from a list of items that can change from run to run, you can use ExplicitName to do that, without worrying about whatever properties of the item will be changing.

ExplicitName allows Tosca to **steer these objects using the index, considering any dynamic changes**. As a matter of fact, you've already used ExplicitName while working with tables. Think of the ability to use #2 to navigate to the second column. This is possible because of the ExplicitName Parameter. So, in your TestCases, if you have a list with options that can change from run to run, you can use ExplicitName to always select the first option regardless of what the option is.

You can add the ExplicitName Configuration Parameter to any **ModuleAttribute** in the Properties tab. Ideally, this is a ModuleAttribute that is generic and has multiple instances, such as a list of items. Simply pop out the Properties Tab, right-click on the header, and select the grey box from the Context Menu, to add a new Parameter. In the left column, type ExplicitName, and in the right column, type True.

You can utilize the ExplicitName Configuration Parameter by going to a step that contains the ModuleAttribute and replacing the name with an index number. For example, when selecting from a list of recently viewed items, a list that can change at any time, you can simply tell it to select the first option every time by typing a #1 in the name of the TestStepValue box.

Congratulations! You've arrived at the end of this lesson. And now, you are able to add the ExplicitName Configuration Parameter to a ModuleAttribute and steer within a TestStep, using ExplicitName.

Lesson 36 - TBox Set Buffer

In this video, we are going to focus on the **TBox Set Buffer Module.**

By the end of this lesson, you will be able to find and use the TBox Set Buffer Module. And you will achieve this by looking at what the Tbox Set Buffer Module is, why is it useful, where you can find it, and how you can utilize it in your TestCases.

Let's begin by having a look at what the TBox Set Buffer Module is.

TBox Set Buffer is a **Standard Module that allows you to set a value to a Buffer manually,** or **remove the value of the Buffer.** Tbox Set Buffer is useful in situations when you may want to reset a Buffer's value, set a Buffer to a specific value, store a timestamp, or even perform calculations within a Buffer's value. For example, you can reset the value of a Buffer at the beginning of your TestCase and then continue to reuse that Buffer throughout your TestCase to perform calculations.

By resetting the Buffer at the beginning of your TestCase, you **ensure that your test results from a previous test run won't affect it.** Without TBox Set Buffer, we would only be able to store values that we buffer indirectly from the SUT. The Set Buffer Module can be found in the Standard Modules folder under TBox Automation Tools>>Buffer Operations. You can add this Module to your TestCase whenever you need it. The Tbox Set Buffer Module is very easy to use and is flexible enough to allow you to get creative with how you use it.

Once you've added the Module to one of your TestCases, you will see a **whited-out box with <BufferName>** as a placeholder in the name column. You can type a new or existing Buffer name in the field; this will be the Buffer that the Module changes. Once you do that, a new row will automatically be created underneath the one you just edited.

You can add as many Buffers into one Module as you would like. But if you leave the value field empty, the TestStep will remove any value currently in the Buffer.

Now, if you would like to change the value of the Buffer, simply enter whatever you want to set the Buffer value to in the Value Field. This could be a number, a string, or even a dynamic expression.

Please remember, to always leave the **ActionMode as "Input"** which is what it is by default. This step won't work if you change the ActionMode.

And this brings us already to the end of this lesson. Now you are able to use the TBox Set Buffer Module.

Lesson 37 - ResultCount

In this video, you are going to learn about the ResultCount property, so that by the end of the lesson, you will be able to use this property within your TestSteps. You'll achieve this objective by looking at what ResultCount is, and for that, we'll take a look at some real examples. We will then also look at when and why it can be helpful. And we will finish the lesson by looking at where you can find this functionality in Tosca and how you can use it.

Let's begin by having a look at what ResultCount is.

ResultCount is a property that enables you to **count how many times a certain value or item appears within an HTML page.** For example, here in the Demo Web Shop, you can use ResultCount to find out the number of orders in the order history. Keep in mind that you could also use **"RowCount" and "ColumnCount"** for tables. However ResultCount offers you a lot of value as it counts the occurrences of a particular value, independent of its position.

You may want to use this number to take action elsewhere in your TestCase. Some examples of that could be repeating a step a dynamic number of times, going to the last item in a list, or verifying that there are as many items as we're expected.

You can use the ResultCount property in the **property section of any TestStepValue field**. The properties field is where you have selected InnerText property previously. There are numerous other properties you can manually enter into this field but, for now, we are going to focus on ResultCount. This property is most useful when trying to buffer the number of objects that may appear on screen multiple times, such as a generic DIV or a checkbox.

To understand how to use ResultCount, let's get back to our example.

On the Orders page, you have a list of all orders placed so far. If you want to verify the number of orders that have been placed in the system, you can use ResultCount. Now, using what you've learned previously about Dynamic Identification, you can create the module that treats all of these orders as similar generic items.

When you tell Tosca to find the ResultCount, it will return the total number of items that match that criterion on the page. This was done by setting the **"InnerText" to "Order Number:*".** If this control was uniquely identified, its ResultCount would always be one because, by definition, there can only be one. This is why the control must be generic. And now that you have a generic control to work with, you can type ResultCount into the properties field, along with the name of the Buffer you want to store the count in. Once you have entered the ResultCount and named your Buffer, you'll want to set the ActionMode to "Buffer".

We can also use **ResultCount in conjunction with the ActionMode "Verify"**, if you already know the number that should be present.

And this brings us to the end of this lesson. Now you're able to use the ResultCount property within your TestSteps. Put your knowledge to test with this exercise.

Lesson 38 - Repetition on Folder Level & ExplicitName on TestStepValue

In this video, we will discuss how to use **Repetition at the folder level.** So that by the end of this lesson, you will be able to add a Repetition value to a TestStepFolder and use the Repetition notation within a TestStep. Specifically, we're going to look at: what the Repetition property does, why you would want to use it, where you can configure it, and how you can reference your repetition throughout your TestCases to solve dynamic problems.

Repetition is a parameter that instructs Tosca to **repeat the steps within a particular level, a specified number of times**. You can define the Repetition only on the TestStepFolder level, which tells Tosca to repeat the steps within the folder. You can actually instruct Tosca how many times to repeat that action. You can either decide to repeat the step a **certain number of times,** let Tosca repeat the step a **random number of times,** or you can let Tosca repeat the step a **dynamic number of times** based on some previous action taken in Tosca.

For example, if you need to remove two items from your Shopping cart, you can repeat a TestStep Folder that removes the first item from your cart twice. If you now need to run an action a dynamic number of times, you can use the "ResultCount" function in conjunction with the Repetition property.

Now, let's say for example that you need to tell Tosca to delete all the items in your order list, but you do not know how many items you have in there. In this case, you'd tell Tosca to count all your orders with the "ResultCount" property, and then use that result to define how many items to be deleted from the order list. The Repetition property can be found in the properties of every TestStep Folder. By default, the property is blank.

You can add **any value** that translates to a **whole numerical value** into the property. This includes: **static whole numbers, math Expressions, buffers containing numerical values, ResultsCount values,or RowCount or ColumnCounts from tables.**

But how can you use this repetition to iterate through a table of a group of items? With the Repetition notation, you can reference whichever repetition you are currently on. For example, on the second time, the folder repeats, repetition would be equal to two. And on the third time, it would be equal to three. This means, that if you set the Repetition to the number of items in the Shopping cart in the Demo Web Shop, for example, Tosca will repeat the TestSteps within the TestStep Folder as many times as there are items in the Shopping cart.

Let's see how this would look like in the Workspace.

If what you want is to **repeat an action on every row in a table**, you can place the **"Row"** number in the TestStep with: **"#{Repetition}"** and then the folder will repeat as many times as there are rows that you counted in a previous step.

And this already brings us to the end of this lesson. Now you can start using folder repetition within your TestCases, as well as the Repetition notation. Give it a try by completing this exercise.

Lesson 39 - Advanced Topics - ActionMode Constraint

In this video, we are going to talk about one of the advanced features in Tosca; the **ActionMode Constraint**. Once you complete this lesson, you will be able to use this ActionMode and also identify possible scenarios where you can use it.

To accomplish these objectives, we will cover the following topics: What exactly is the ActionMode constraint? Why can it be effective when creating your TestCases, and where and how can you use it in Tosca.

Constraint is a type of **ActionMode** that allows you to **identify one specific item in a list of similar items.** The item you're looking for needs to be uniquely identifiable. Having this in mind, imagine 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.

Let's see how this looks like in the Demo Web Shop with a real example.

In our shopping cart, you have the following products: "Music 2" with the price of three, "Music 2", with the price of 10, and you also have a book that is also priced 10.

Now you want to delete the "Music 2" product worth of 10 but it is not enough to search just for the product called "Music 2", as there are more than one and neither is enough to search for a product with the price of 10, as also there are more than one product as well. What you can do instead is to search for both product name "Music 2" and the price 10. This means that you need to find this particular product and then delete it by looking for its name. In Tosca you will do this by specifying the data to steer in the product's table from the TestStep.

Now, we have seen that the ActionMode Constraint will **allow you to find or identify a role by searching for specific values** within it. Let's also see how to use it in Tosca.

First, you need to create a TestStep using the Shopping Cart Module to be able to steer the product's table. The next step is to search for the product "Music 2", and the price 10 in the product column of the product table. To do so in the product's table from the TestStep, you will create the price and product rows and set the ActionMode to Constraint.

This will tell Tosca that **both of these values must be met in order to select the appropriate role** in the product's table. Tosca will then be able to find the item in the table, regardless of the row it appears in. We can actually tell Tosca to select the button "Remove" and update the shopping cart. Remember that you can use the ActionMode Constraint wherever you have a hierarchy within your modules, not only for tables, but the most common use in HTML is tables.

And this already brings us to the end of this video. And now you have a better understanding of what the ActionMode Constraint is and how to use it. Put your knowledge to test with this exercise.