



# RANOREX

STUDIO FUNDAMENTALS  
USERGUIDE

## TABLE OF CONTENTS

<b>RANOREX STUDIO FUNDAMENTALS.....</b>	<b>4</b>
RANORIZE YOURSELF IN 20 MINUTES.....	5
<i>Download and install Ranorex Studio</i> .....	5
<i>Plan your first test</i> .....	8
<i>Create a new solution</i> .....	10
<i>Record your first test</i> .....	13
<i>Analyze your recording</i> .....	18
<i>Run a test and check the report</i> .....	20
RANOREX STUDIO .....	23
<i>Ranorex Studio start page</i> .....	29
<i>Sample solutions</i> .....	33
<i>Create a new solution</i> .....	34
<i>Create a new project</i> .....	41
<i>Working environment and views</i> .....	46
RANOREX RECORDER .....	51
<i>Stand-alone Recorder</i> .....	51
<i>Before you start recording</i> .....	52
<i>Record a test</i> .....	58
<i>Analyze a recording</i> .....	63
<i>Run and debug recordings</i> .....	69
<i>Manage recording modules</i> .....	79
<i>Recorder control center &amp; hotkeys</i> .....	85
TEST SUITE .....	92
<i>The test suite structure</i> .....	96
<i>Build a test</i> .....	104
<i>Execute a test suite</i> .....	113
<i>Manage multiple test suites</i> .....	124
ACTIONS.....	133
<i>Actions and repository items</i> .....	135
<i>Manage actions</i> .....	138
<i>All actions and their properties</i> .....	146
<i>Invoke actions</i> .....	170
<i>User code actions</i> .....	173
REPOSITORY .....	181
<i>Repository items and actions</i> .....	186
<i>Create repository items</i> .....	187
<i>Manage repository items</i> .....	189
<i>Structure repository items</i> .....	198
<i>Clean up the repository</i> .....	205

<i>Represent multiple elements with a single repository item .....</i>	<i>206</i>
<i>Manage multiple repositories .....</i>	<i>211</i>
<i>Embed a repository .....</i>	<i>215</i>
TEST VALIDATION .....	217
<i>Understanding validation .....</i>	<i>217</i>
<i>Text-based validation example .....</i>	<i>225</i>
<i>Attribute-based validation example .....</i>	<i>231</i>
<i>Image-based validation example .....</i>	<i>239</i>
<i>Validation of tooltips .....</i>	<i>246</i>
WHITELISTING .....	251
<i>Editing your whitelist .....</i>	<i>251</i>
RANOREX COACH .....	254
REPORTING .....	260
<i>Actions and the report .....</i>	<i>261</i>
<i>Report levels .....</i>	<i>267</i>
<i>Ranorex standard reporting .....</i>	<i>273</i>
<i>Customization basics .....</i>	<i>290</i>
<i>Complex report customization .....</i>	<i>314</i>
<i>Convert existing reports to PDF .....</i>	<i>325</i>

# Ranorex Studio fundamentals

This section describes the tools and concepts you need to **start automating basic tests** in Ranorex Studio. Each topic includes easy-to-follow step-by-step guides and self-contained examples so that you can gain hands-on experience immediately. If you enjoy learning from a **written tutorial**, we recommend starting with → [Ranorexize yourself in 20 minutes](#) to get a feel for Ranorex Studio while producing your first test. If you prefer learning from a **video tutorial**, we recommend watching the Ranorex Studio Quick Start [screencast series](#).



Ranorexize yourself in 20 minutes



Ranorex Studio



Actions



Ranorex Recorder



Repository



Test Suite



Test Validation



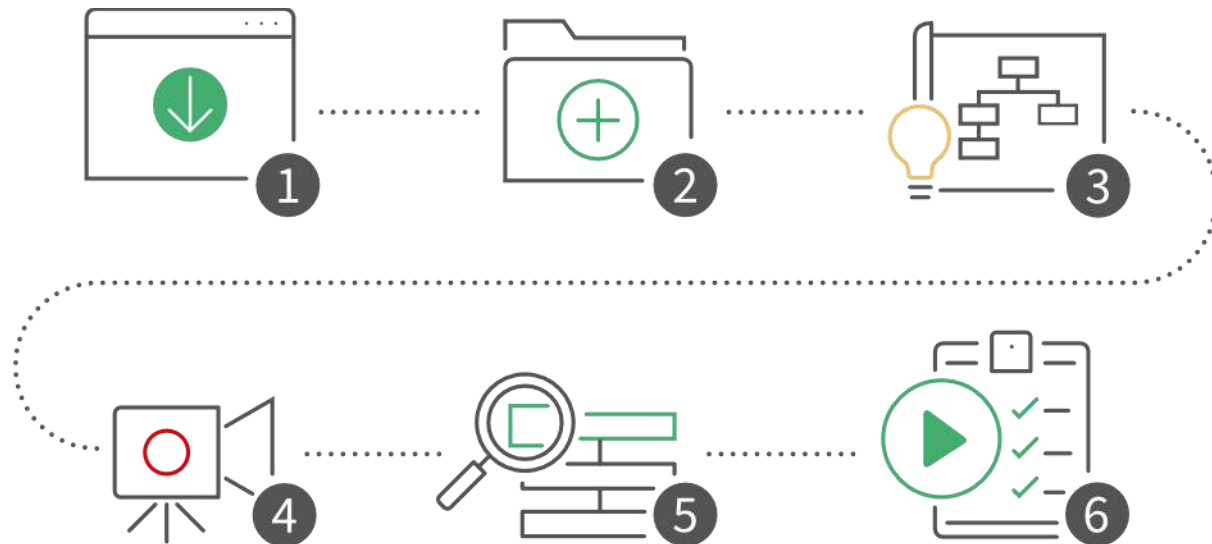
Whitelisting



Reporting

# Ranorize yourself in 20 minutes

Welcome to **Ranorize yourself in 20 minutes!** In this guide, you'll take your first steps with Ranorex Studio and will become familiar with some of its basic functions. You'll accomplish this by **creating your first automated software test in 6 simple steps.**



**20 minutes and a working computer is all you'll need.** We'll cover everything else as we go along.

## Download and install Ranorex Studio



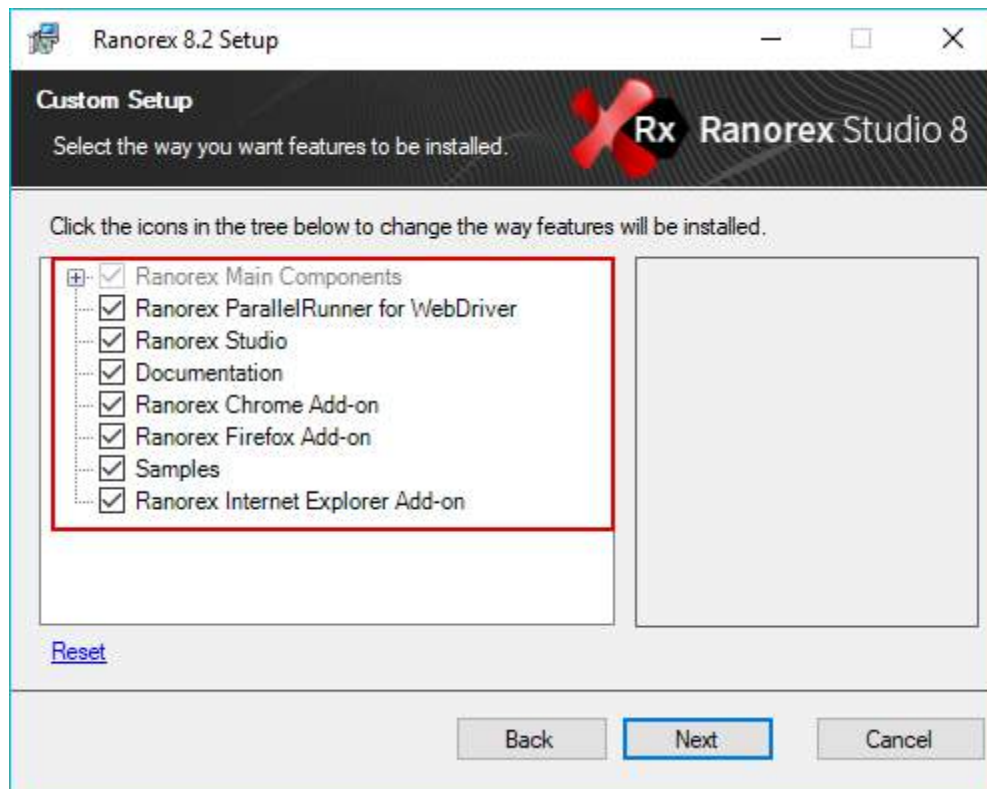
### Screencast

The screencast “Download and Install Ranorex Studio” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Instructions

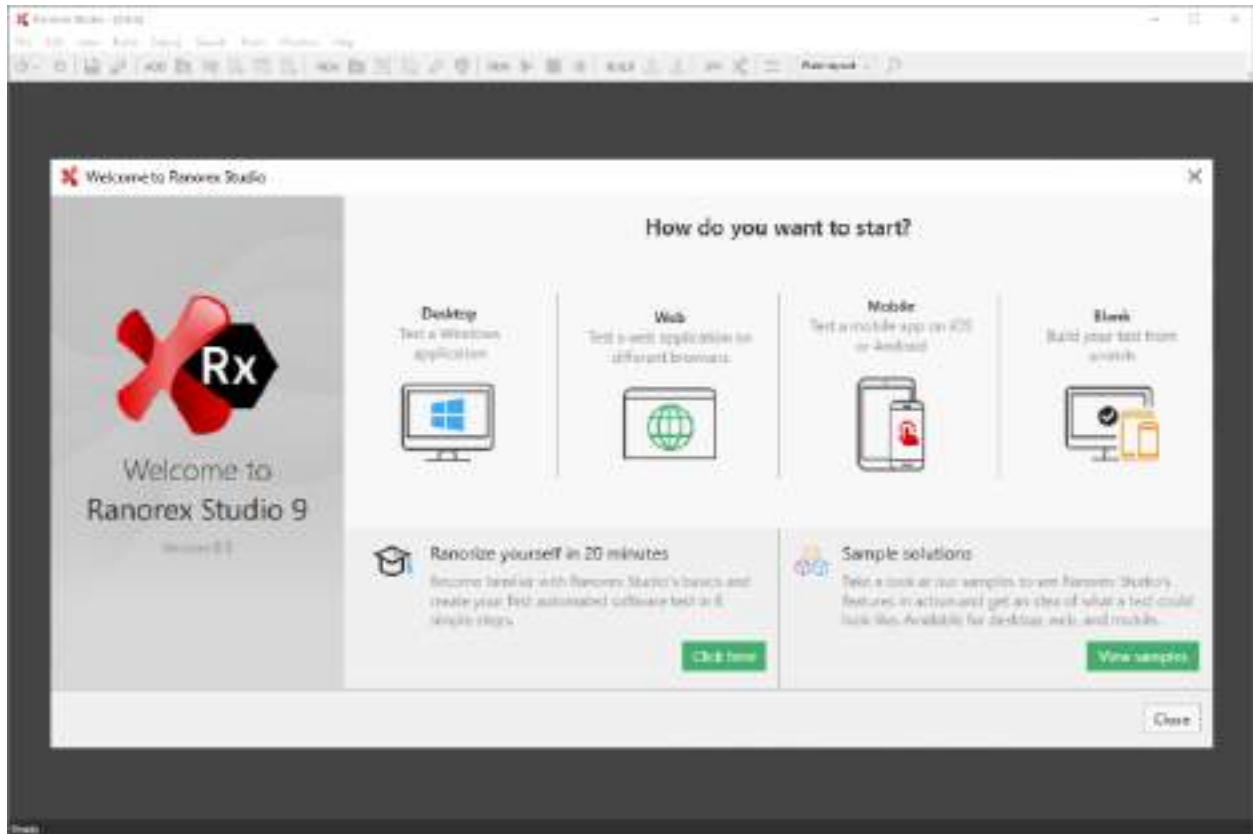
1. Go to [/free-trial/](#) and follow the instructions to download a free trial of Ranorex Studio.
2. Once downloaded, **run** the setup file and follow the instructions in the Installation Wizard. To ensure complete installation, **do not deselect** any features in this dialog:



- a **Select** and **insert** a valid license key if you already purchased one.
- b **Select** if you want to use Ranorex Studio up to 30 days in trial mode.



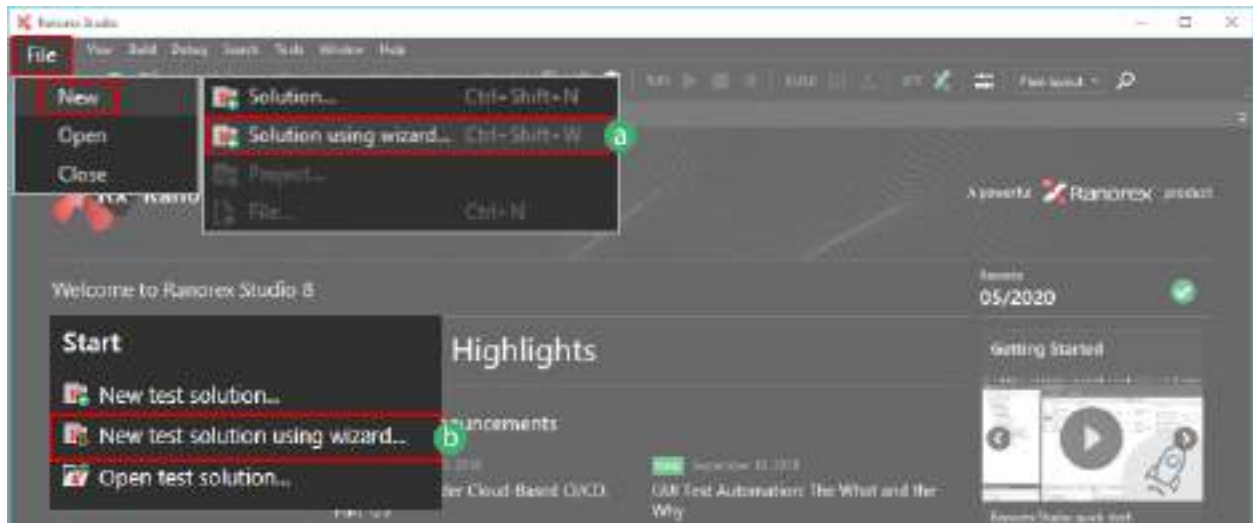
- Once finished, Ranorex Studio launches to the start page and displays the RocketStart solution wizard.



#### Note

If the RocketStart solution wizard does not appear, you can bring it up by clicking one of the options below:

- File > New > Solution using wizard...**
- New test solution using wizard...



## Plan your first test

Before we start building our test, we need to think about what we're actually going to do. This helps **prevent errors** and tedious restructuring of tests.

To plan the test, we are going to create in this guide, we ask ourselves the following questions. These also apply when you create more complex tests:

- **What do we want to test?**
- **How can we test it?**

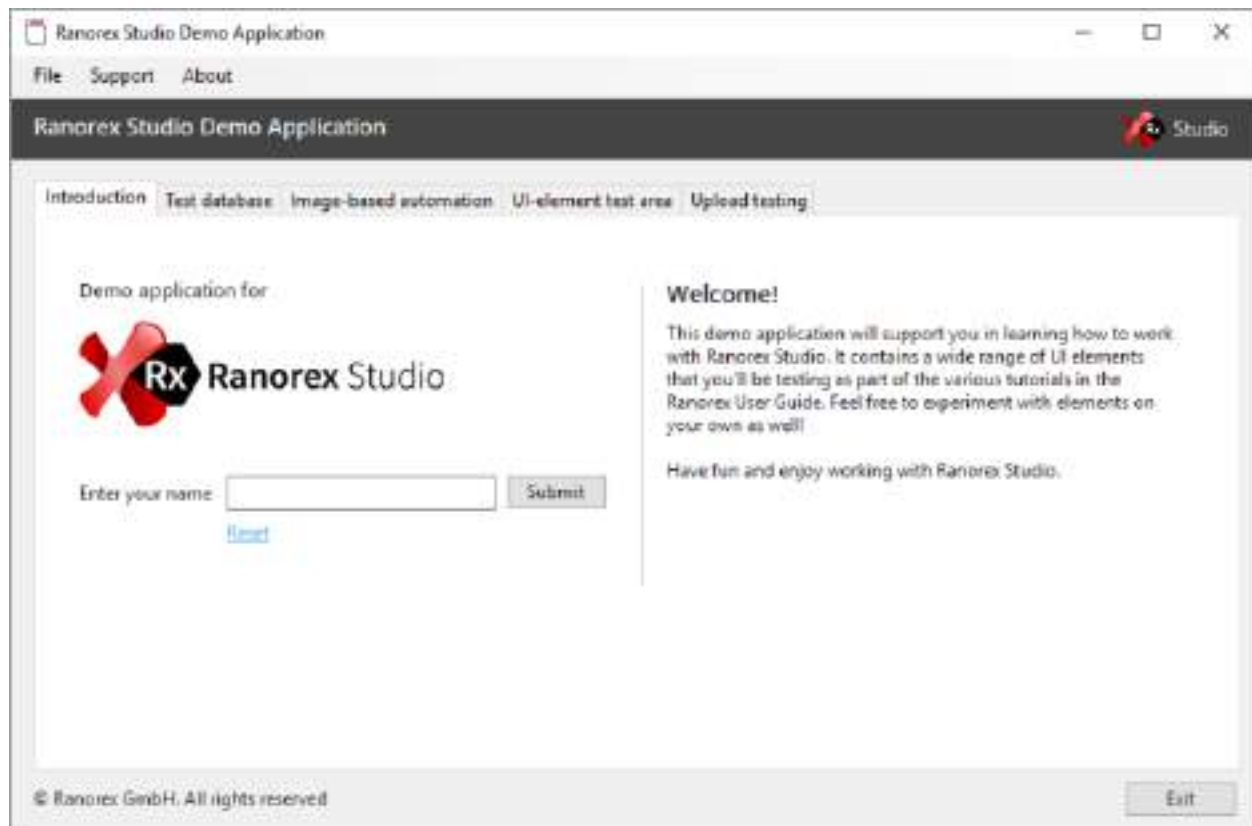
### What do we want to test?

In this guide, **we test the Ranorex Studio Demo Application**, a simple program to support you in learning how to work with Ranorex Studio.

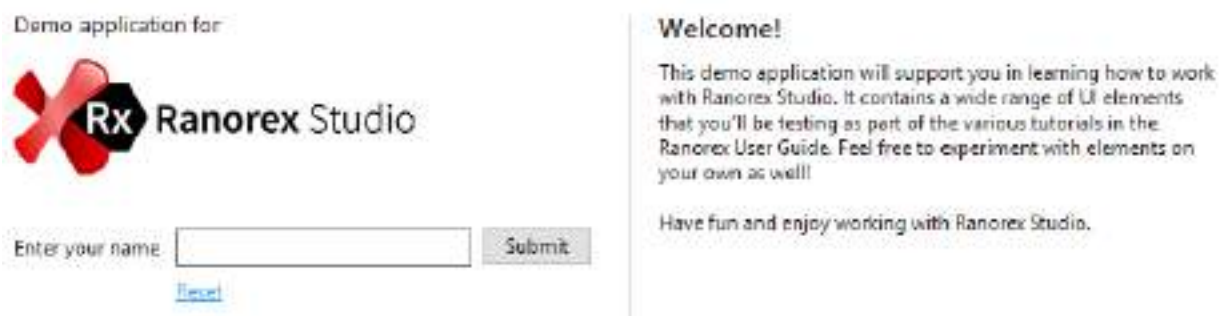
#### Demo Application download

The demo application can be downloaded here → [Ranorex Demo Application](#). Extract the application into any folder of your choice. For the purpose of this guide, we'll assume that it's saved to the `/Downloads/` folder of your system.





We will create a test to confirm that text entered in the **Enter your name** field appears in the welcome message on the right, after clicking **Submit**.



## How can we test it?

To test the behavior described above, **our test will need to include the following actions:**

1. **Start** the application
2. **Click** the text box
3. Enter “Harry” in the text box
4. **Click Submit**
5. Verify that the text is displayed in the welcome message

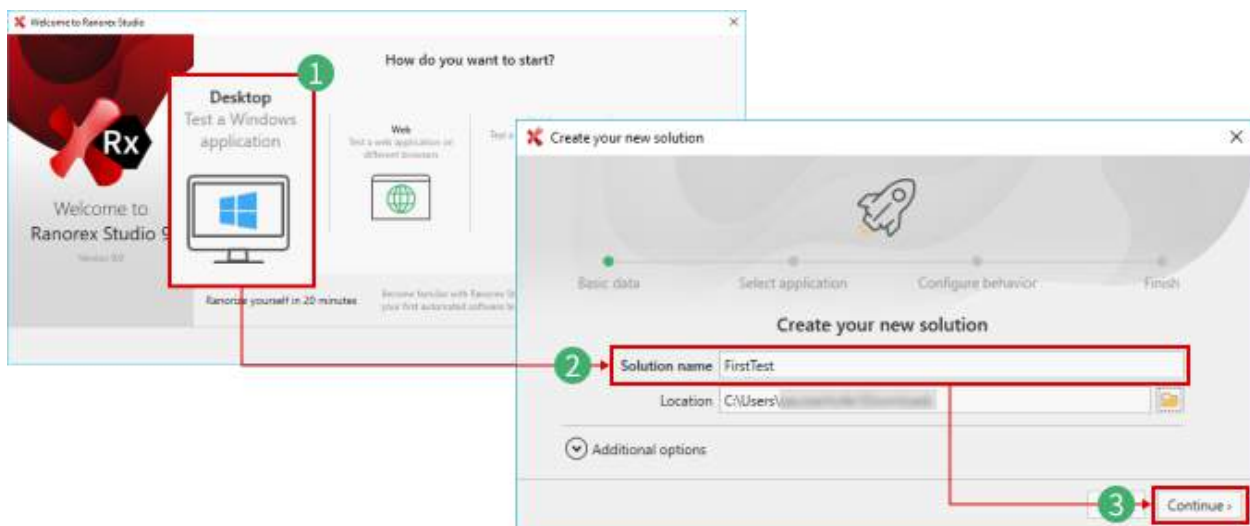
6. Close the application

The test is successful if the submitted name appears in the welcome message.

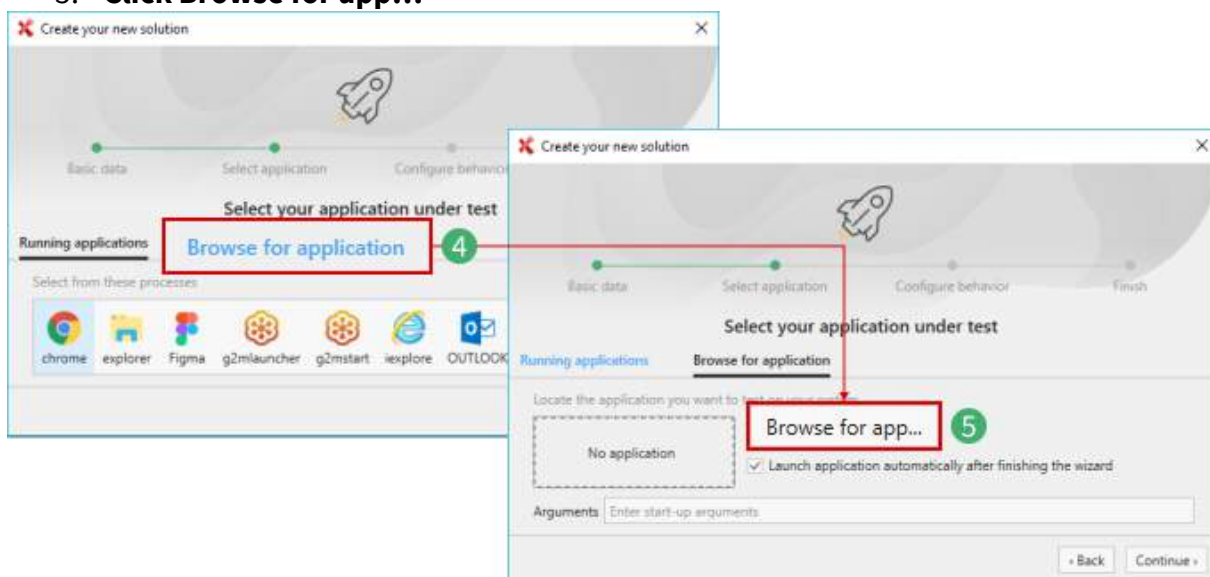
## Create a new solution

In this step, you will create a new solution and take a first look at the Ranorex Studio user interface.

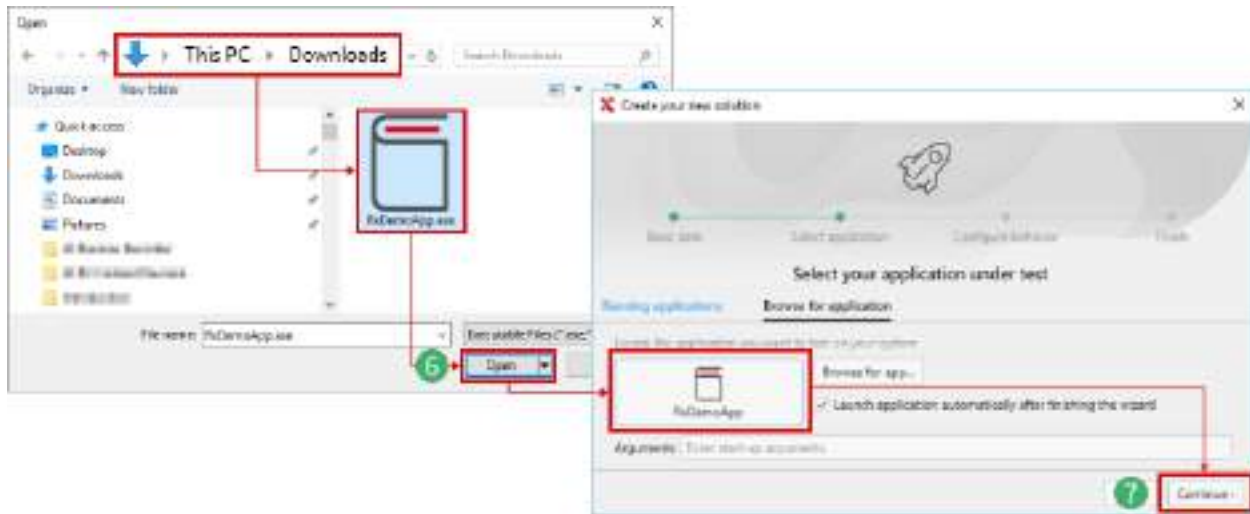
1. In the RocketStart solution wizard, **click Desktop**.
2. In the dialog that opens and enter a **Solution name**.
3. **Click Continue**.



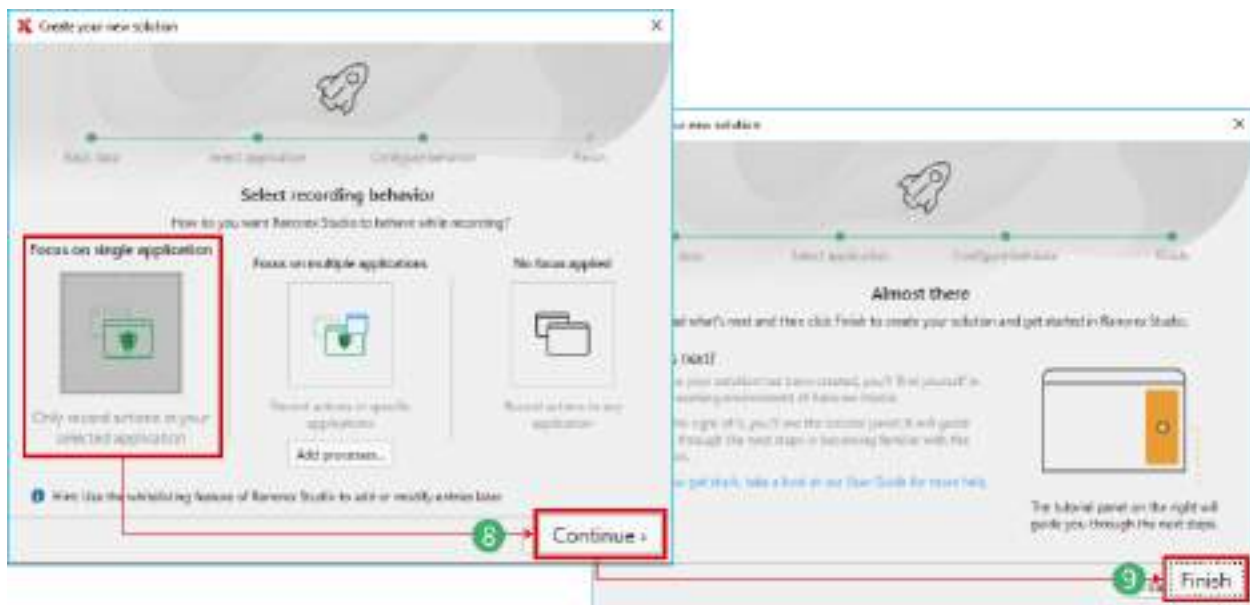
4. **Click Browse for application.**
5. **Click Browse for app...**



6. **Browse** to where you saved the Ranorex Studio Demo Application to and **click Open**.
7. The Demo Application appears in the RocketStart solution wizard. **Click Continue**.



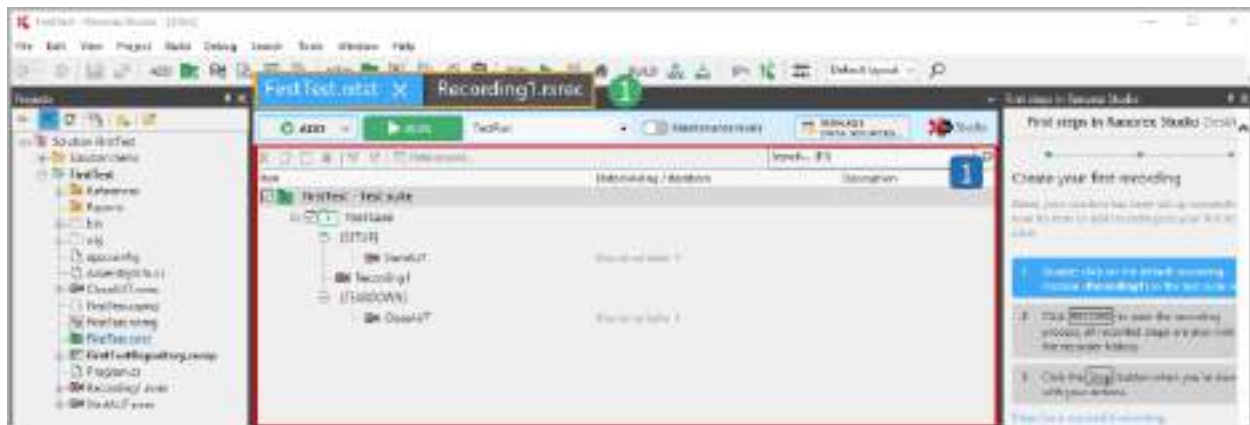
8. **Select Focus on single application** and **click Continue**.
9. **Click Finish** to complete the setup.



## The Ranorex Studio user interface

Let's take a quick look at the different elements of the Ranorex Studio user interface.

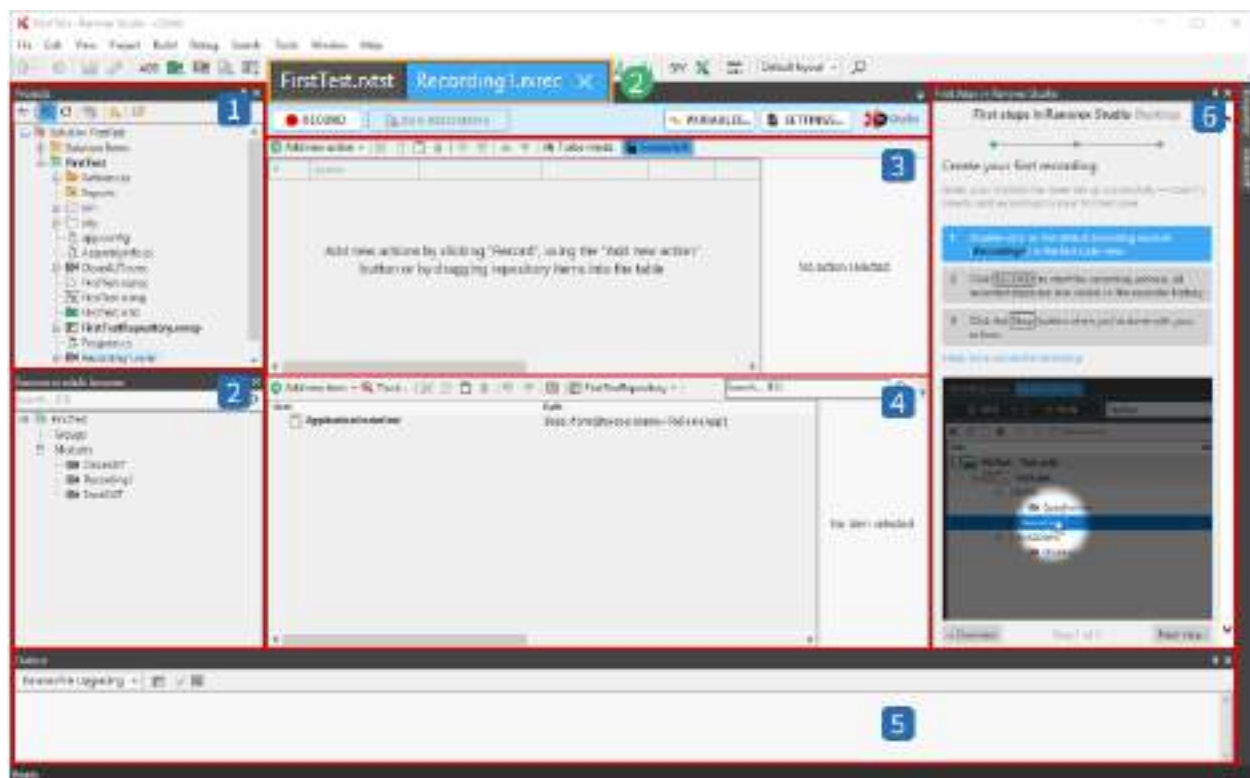
- 1 **Click** the tab **FirstTest.rxtst** to switch to the test suite view.



## 1 Test suite view

The test suite is where you build and control your tests. Your test suite already comes with a simple test case that contains three recording modules: **StartAUT** starts the Demo Application, **Recording1** will contain the test actions we'll record in the next chapter, and **CloseAUT** closes the Demo Application.

## 2 Click the tab **Recording1.rxrec** to switch to the recording module view.



## 1 Projects pad

The projects pad displays all projects, folders, references, and files associated with your test solution.

## 2 Module browser pad.

The module browser provides quick access to all modules and module groups sorted by project.

## 3 Recording module view – actions table

This view is where you fill your recording module with actions and edit them. They are listed chronologically in an “actions table.”

## 4 Recording module view – repository

The repository contains all repository items that are referenced by actions. Repository items represent UI elements.

## 5 Output pad

This panel displays build information, errors, debug information, and search results.

## 6 Tutorial panel

This panel shows a quick tutorial that guides you through the first steps of building a test. While you are going through “Ranorize Yourself in 20 Minutes”, you do not need to worry about it.

## Record your first test

It’s time to record our test! This means we’ll **manually perform** the actions we determined in the previous chapter and **let Ranorex Studio record** them.



### Screencast

The screencast “Record a Test” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Steps for successful recording

Ranorex Studio supports a wide range of environments, test specifications, and technical settings. So that **your first recording is a success** regardless of your environment, **we recommend the following:**



**While recording, use only the mouse for navigation**

Avoid using other input devices such as graphic tablets, touchpads, pens, ...



**Click every step**

Do not use the tab key to navigate through forms



**Close other applications**

Close any other application or tool that you don't need for the test



**Open the user guide on another machine**

If possible, open the user guide on another machine or tablet for reference while recording your test



**Click pause/continue in the recorder**

To check back to the user guide or perform actions that you don't want to be recorded, just click the **pause** button in the Recorder control center

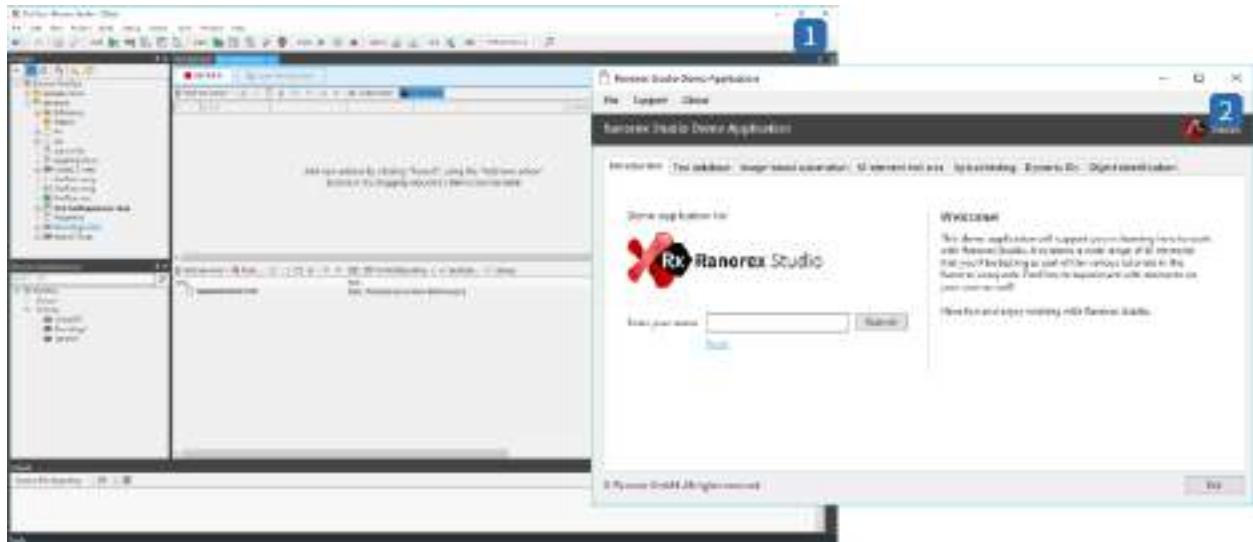


**Set display scaling to 100%**

In the Windows display settings, set display scaling to 100 % for all of your displays.

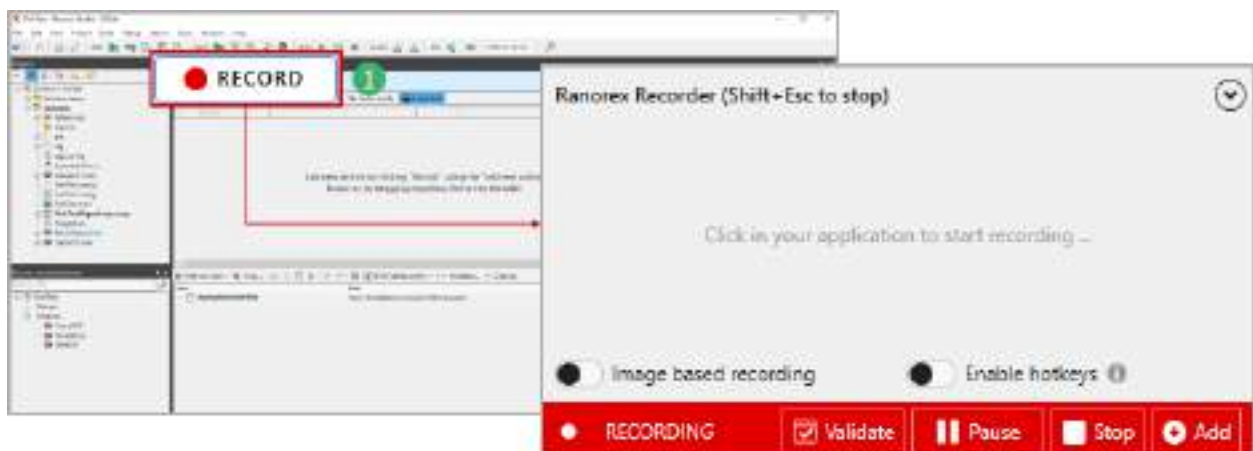
## Record the test

In Ranorex Studio, make sure you're in the recording module view of **Recording1.rxrec**. Also ensure that the Demo Application is running. This should already be the case if you followed our instructions for creating a solution with the RocketStart solution wizard.



- 1 **MyFirstRecording.rxrec** opened in recording module view.
- 2 Running Demo Application showing the welcome screen.

- 1 Click the **RECORD** button to start the recording. Ranorex Studio automatically switches to the demo application and the Recorder controls appear.



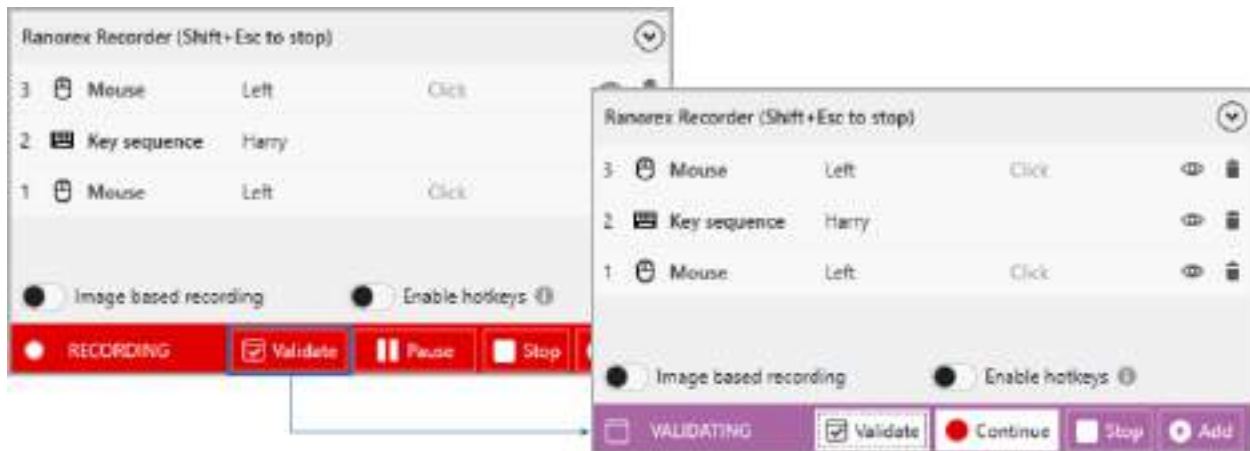
- 2 Click the **Enter your name** text field and click **Submit**.

## Validate a UI element

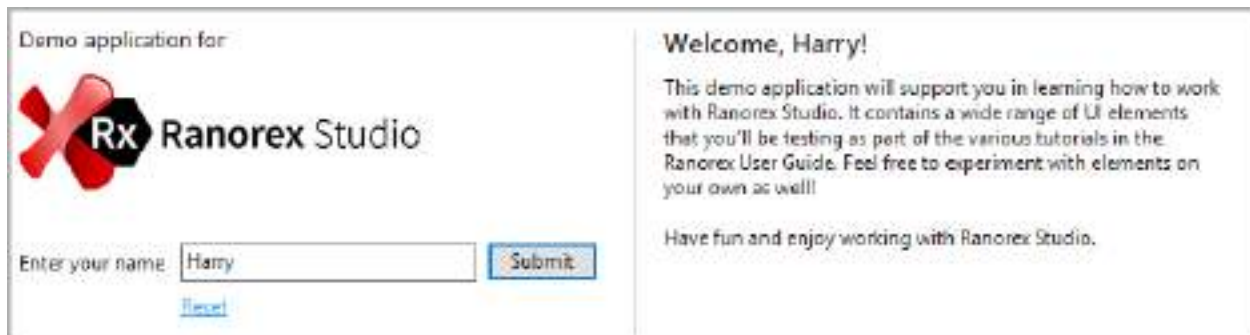
We can see that Harry has appeared in the welcome message. Now we'll add a **validation** so that Ranorex Studio verifies this during testing.



1. **Click Validate** in the recorder controls to pause recording and switch to validation mode.



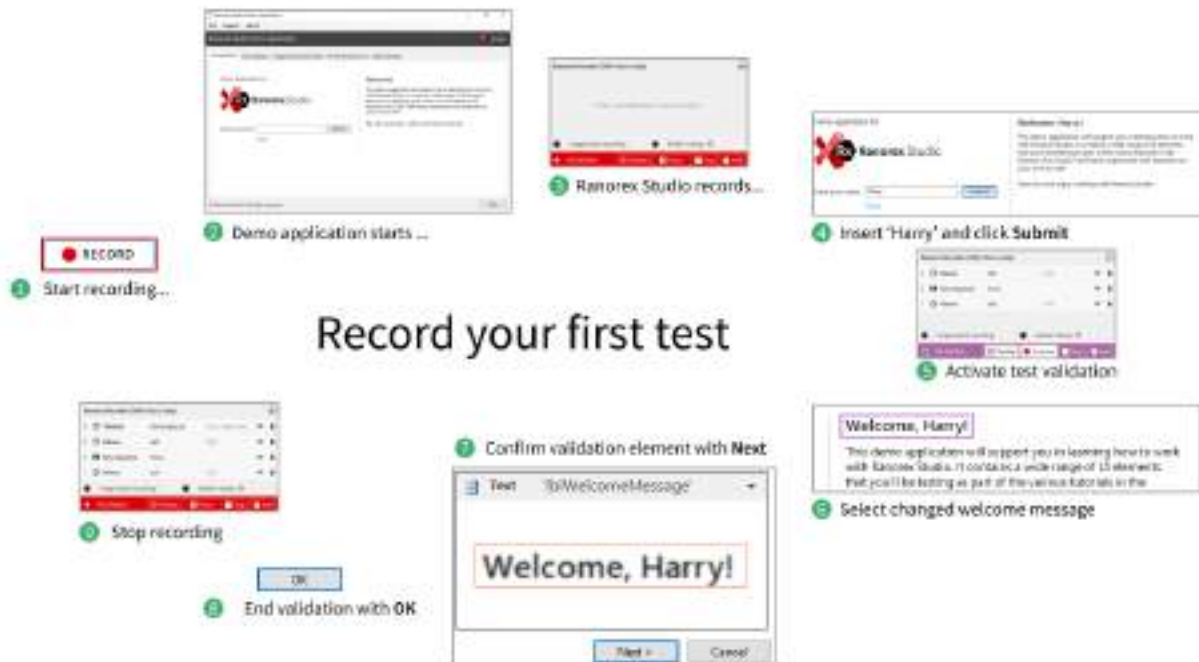
2. **Mouse over** the welcome message. A purple frame appears. This means the UI element has been identified. **Click it**.



3. The **Select element** window opens. **Check** whether the screenshot in the lower right shows the UI element you want to validate. In our case, that's the welcome message. **Click Next** to confirm.
4. The **Validation settings** window opens. Here, you can select which attributes to validate. In our case, **Text** is correctly preselected. **Click OK** to confirm and **switch back** to recording mode.







## Record your first test

### Analyze your recording

After each recording, analyze the **recorded actions** and **repository items** generated, and how these are connected. This helps prevent errors.

### Recorded actions

The **actions table** in the recording view contains 4 individual actions, numbered accordingly. Let's take a closer look at them.

+ Add new action ▾						Turbo mode: Screenshot	
	Icon	Action	Target	Method	Object		
1	🖱️	Mouse	Click	Left	Relative	EnterYourName	1
2	🗑️	Key sequence	Harry			EnterYourName	2
3	🖱️	Mouse	Click	Left	Relative	BtnSubmitUserName	3
4	📄	Validate	AttributeEqual	Text	Welcome, Harry!	LblWelcomeMessage	4

#### 1 Action #1 – Mouse click

This action performs a click in the **EnterYourName** text box.

#### 2 Action #2 – Key sequence

This action enters **Harry** in the **EnterYourName** text box.

### 3 Action #3 – Mouse click

This action clicks **Submit**, updating the welcome message with the name entered in step3

### 4 Action #4 – Validation

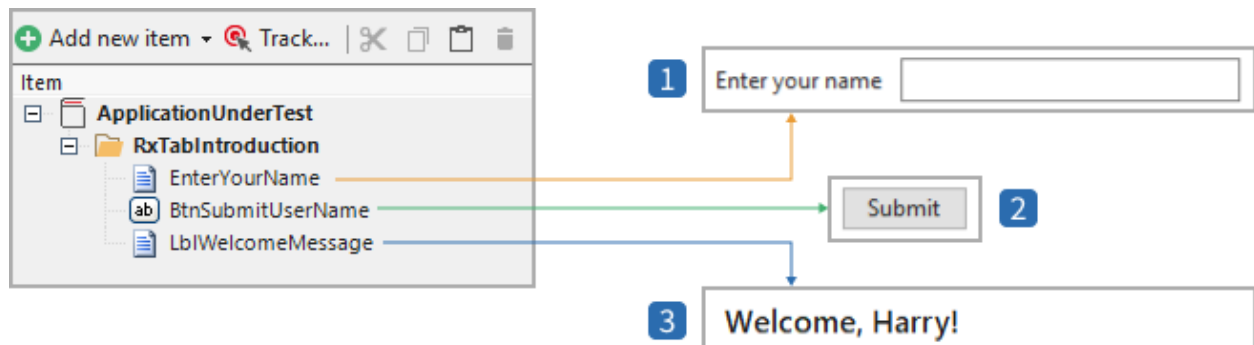
This action validates whether the welcome message has been updated correctly.

#### Hint

If you did not follow the recording instructions precisely, there may be extra recorded actions. If so, identify actions which do not belong to the intended test and delete item.

## Repository items

Several of the actions listed above **manipulate UI elements**. These UI elements are **represented as repository items** in the **repository**, which appears in the lower half of the recording view.



#### 1 Item #1 – EnterYourName

This repository item represents the **EnterYourName** text box

#### 2 Item #2 – BtnSubmitUserName

This repository item represents the **Submit** button

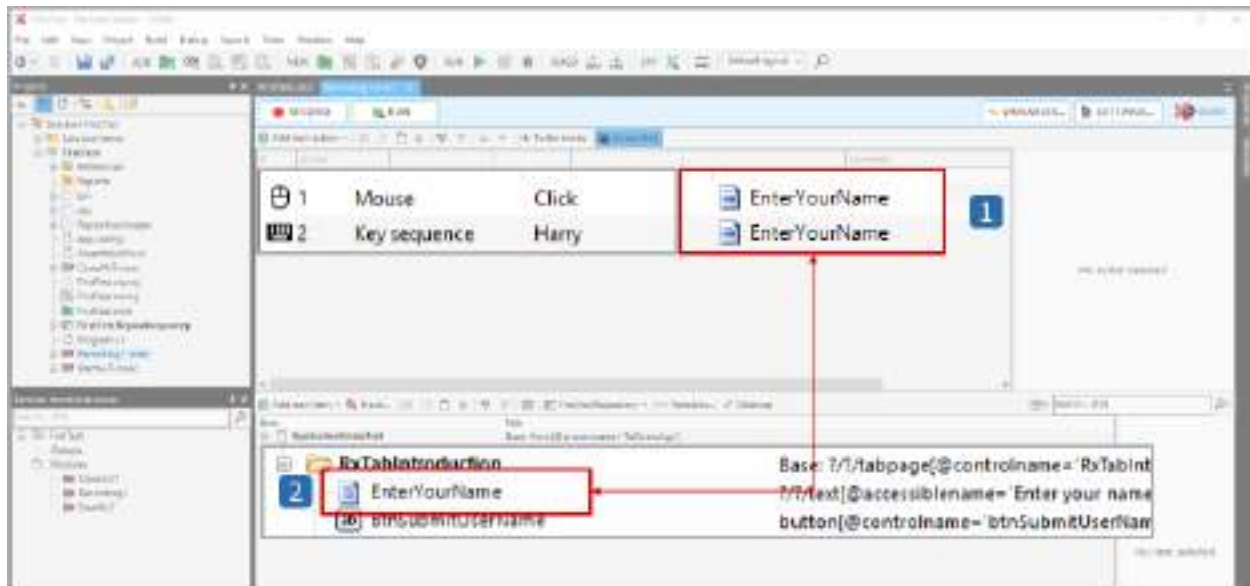
#### 3 Item #3 – LblWelcomeMessage

This repository item represents the **Welcome Message** text label

## The link between actions and repository items

Actions that **manipulate UI elements**, such as clicking a button, are automatically **linked** to the corresponding repository item. The repository item appears next to the action in the actions table. When you click the action, the item is also highlighted in the repository.

Actions that don't manipulate UI elements, such as starting an application, aren't linked to repository items.



1

### Actions that manipulate the EnterYourName text box

The 'Mouse' action performs the mouse click into the text box. The 'Key sequence' action enters **Harry** in it. Both actions are linked to the **EnterYourName** repository item.

2

### Repository item that represents the manipulated UI element.

## Run a test and check the report

It's time to execute the test and review the report.

### Start the test run

1

**Start** the test in one of the ways listed below:

a

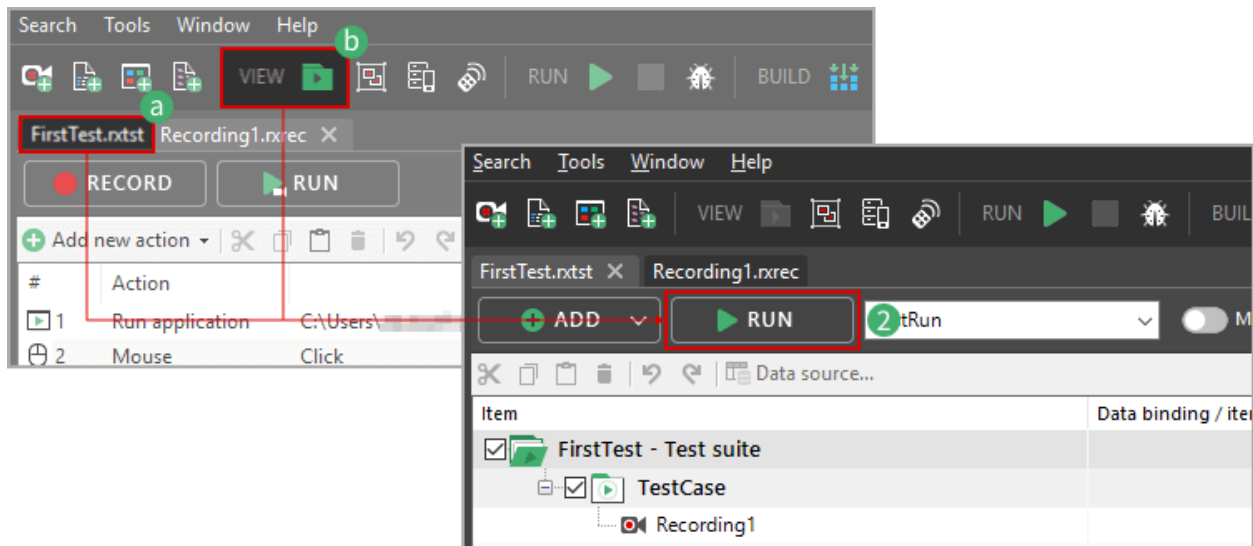
Click the tab for the test suite, **FirstTest.rxtst**, to **switch** from the recording view to the **test suite view**. The test suite is where you manage and run your tests

b

Or, **click** the **View test suite** button.

2

**Click** the **RUN** button in the test suite.

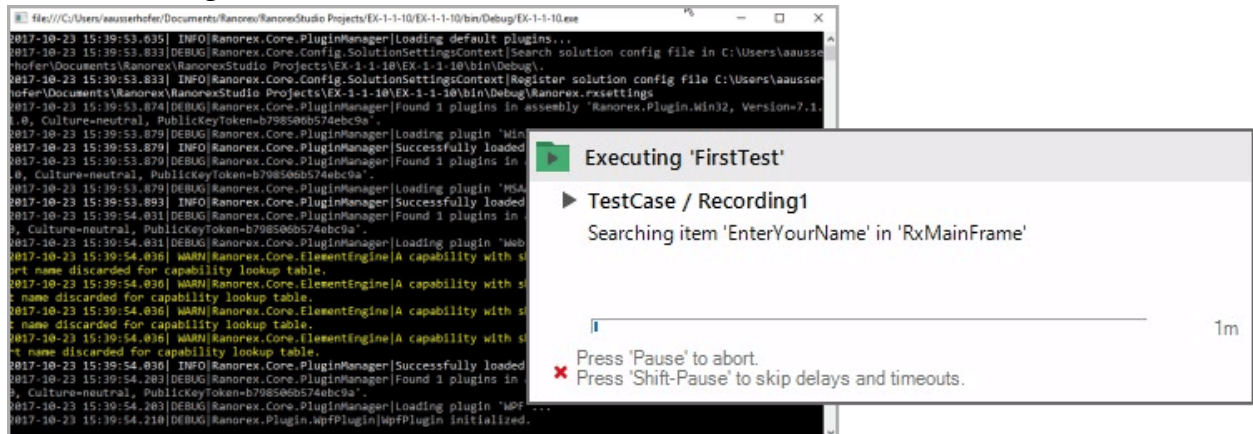


### ⚠ Attention

After you've clicked **RUN**, **do not use the keyboard or mouse**. Doing so would interfere with the test actions and **cause a test failure**.

## Watch the test run

When the test run starts, two status windows appear as Ranorex Studio performs the actions in the recording.



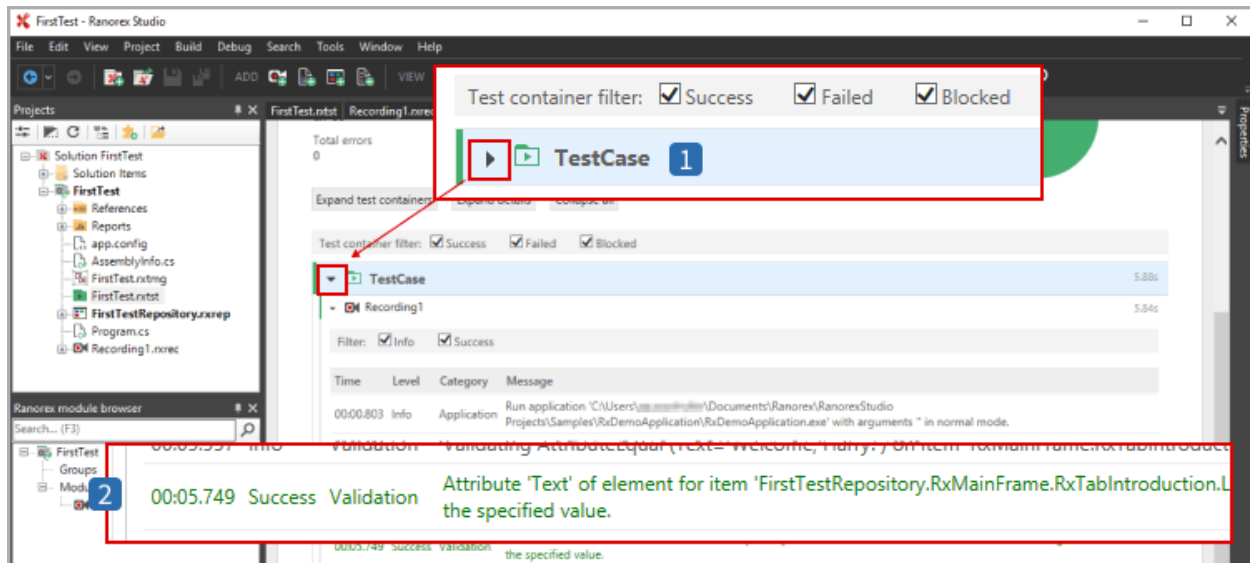
Test run status & progress information

## Review the test report

When the test run finishes, the report opens in a new tab in Ranorex Studio. If everything went as intended, you should see a large green pie chart.

## Report details

Click the **arrows** next to the test case and the recording module to **display more detailed information** about the test run, such as the validation action details.



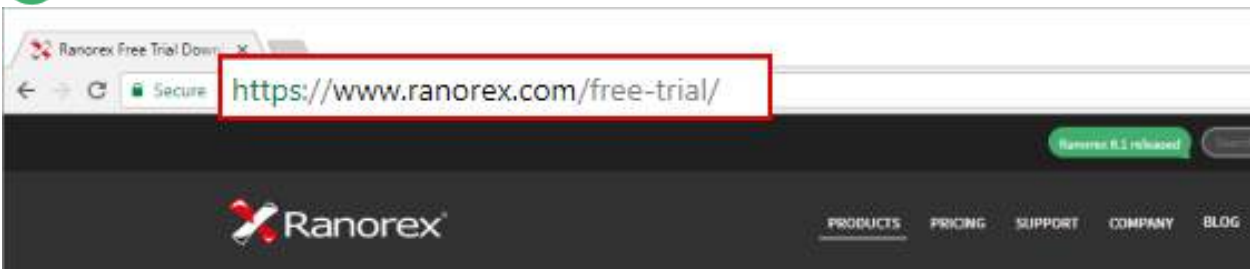
- 1 Button to expand/collapse details
- 2 Detailed information about successful validation

# Ranorex Studio

In this chapter, you'll download and install Ranorex Studio. It's a straightforward procedure that will take about 3 minutes to complete. Just follow the instructions below.

## Download Ranorex Studio

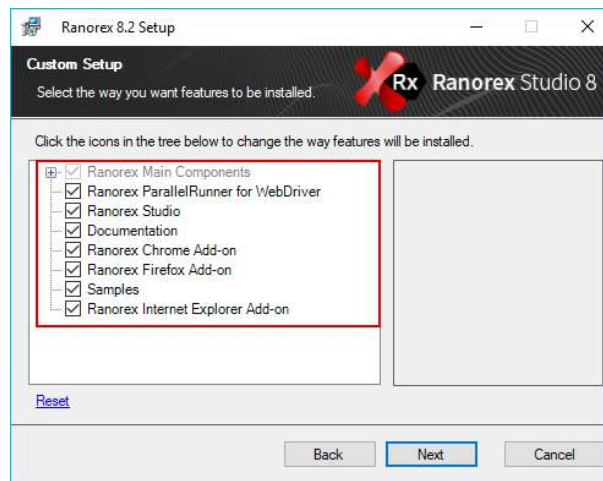
- 1 **Visit** the Ranorex Studio [free trial download](https://www.ranorex.com/free-trial/) web page.
- 2 Enter your **name**, your **company**, and a **valid business email address**.



- 3 **Check** your inbox for an email with the download link.
- 4 **Click** the link to start the download.

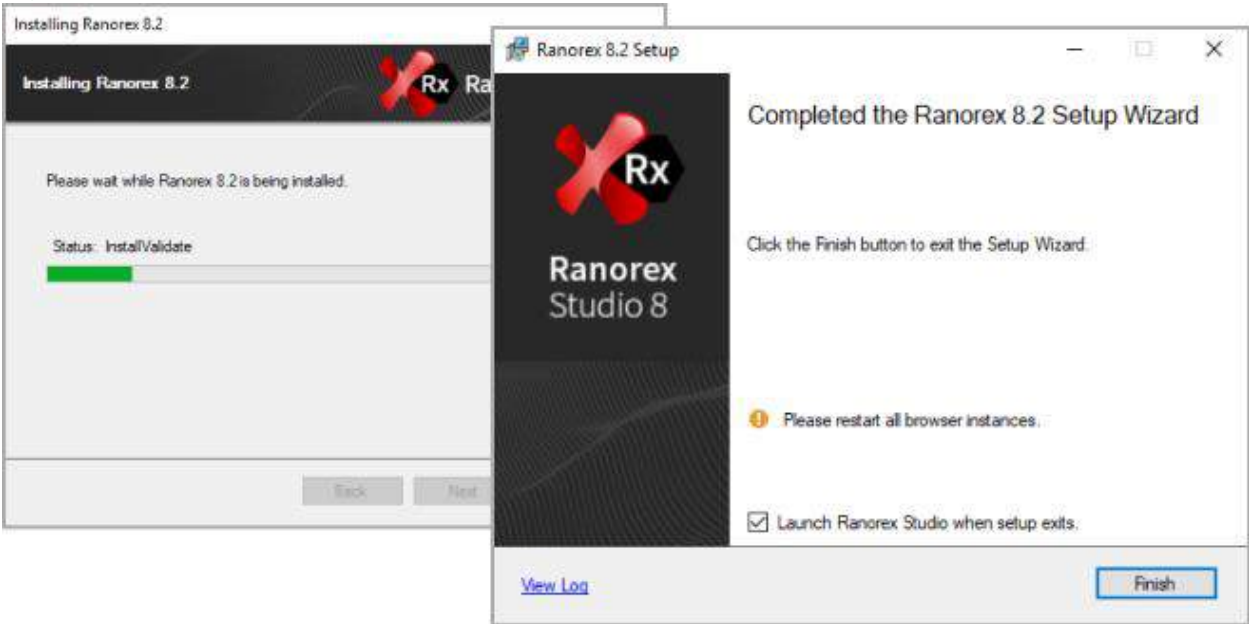
## Ranorex Studio setup wizard

- 1 **Start** setup and follow the instructions.
- 2 **Accept** the license agreement.
- 3 Make sure you **do not deselect any features** when prompted!



## Finish Ranorex Studio installation

- 1 **Watch** the installation progress.
- 2 You should see a message that the installation was successful.
- 3 **Click Finish** to exit the Setup Wizard and start Ranorex Studio.



### 30-day free trial vs. paid license

- a **Select** and **enter** the license information of a purchased Ranorex Studio license
- b You may use this full-featured version of Ranorex Studio for a free trial period of 30 days. Afterward, you must purchase a license to continue using Ranorex Studio.





### Further reading

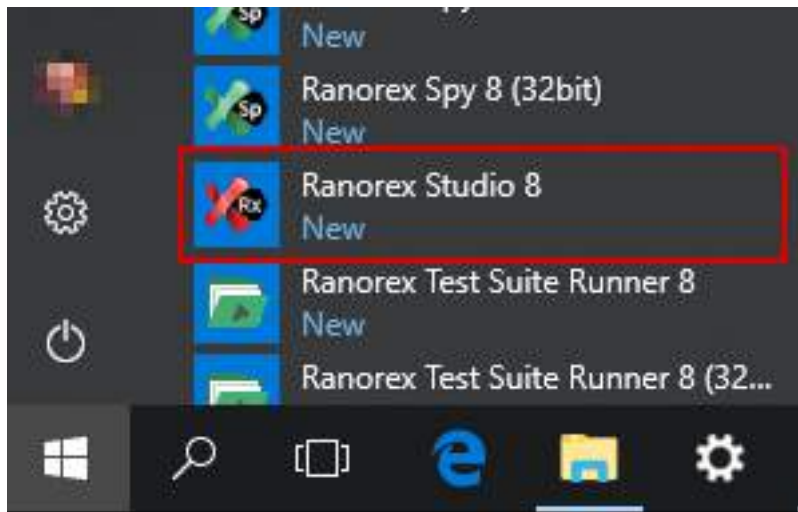
The available license models and the Ranorex License Manager are explained in > Ranorex Studio system details > → [Licensing](#). Pricing information is available on our [website](#).

## Open Ranorex Studio

Ranorex Studio should launch automatically after setup finishes.

To open it manually:

- 1 **Open** the Windows Start menu.
- 2 **Search** for the Ranorex Studio program icon and **click** it.

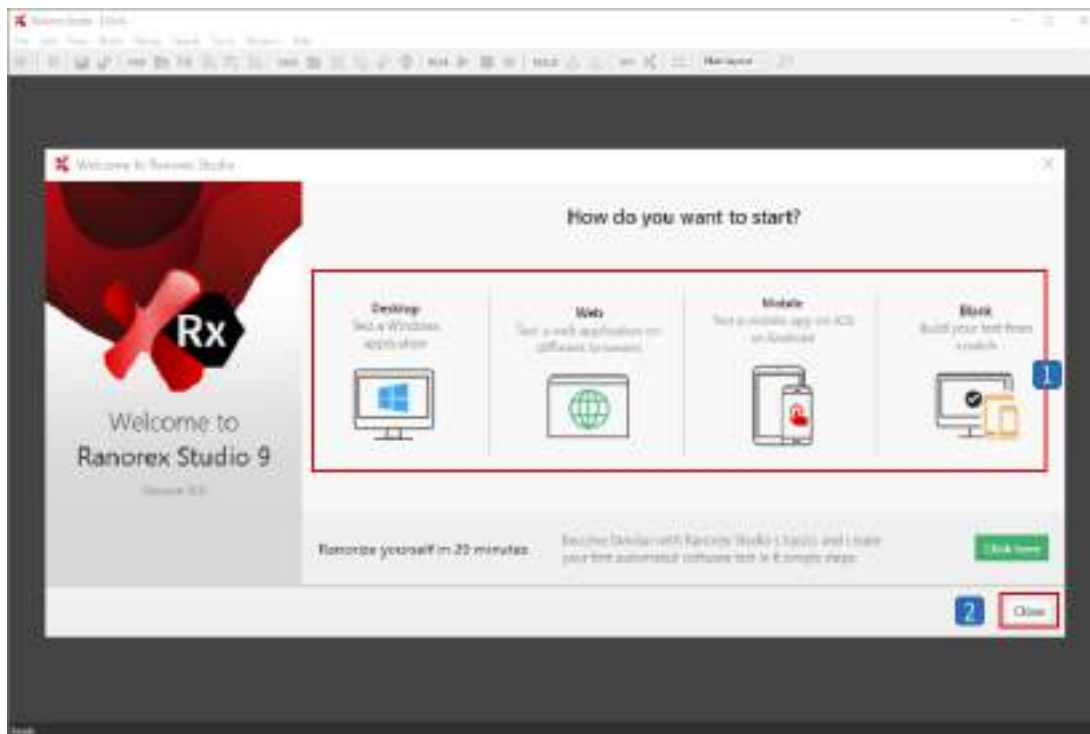


### Note

Ranorex Studio also creates a desktop shortcut automatically.

### Result:

If this is your first time starting Ranorex Studio or if you don't have any projects, the start page opens with the RocketStart solution wizard.

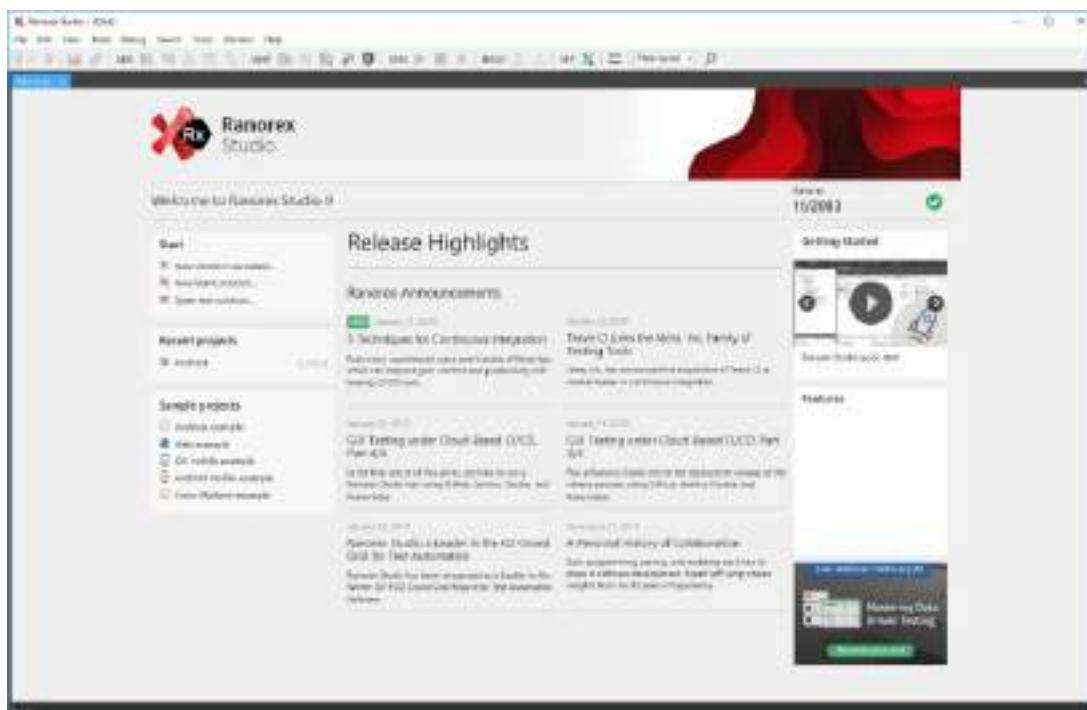


1

Select one of these options, and the RocketStart solution wizard will guide you through the process of creating a solution for the specified testing goal.

2

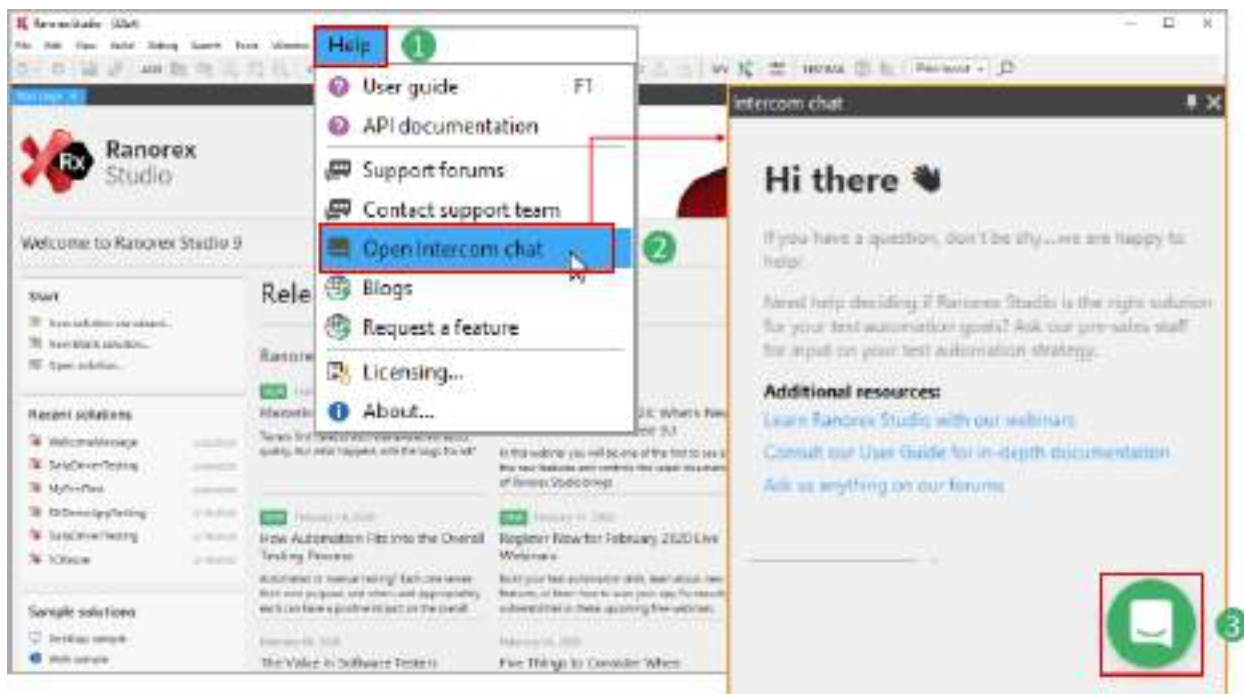
Click to close the RocketStart solution wizard and display the Ranorex Studio start page instead.



## Intercom support chat

In the trial version of Ranorex Studio, you can contact our Pre-Sales department through the integrated Intercom chat. To do so:

- 1 In Ranorex Studio, **click Help**.
- 2 **Click Open Intercom chat**.
- 3 In the window that pops up, **click** the chat symbol.

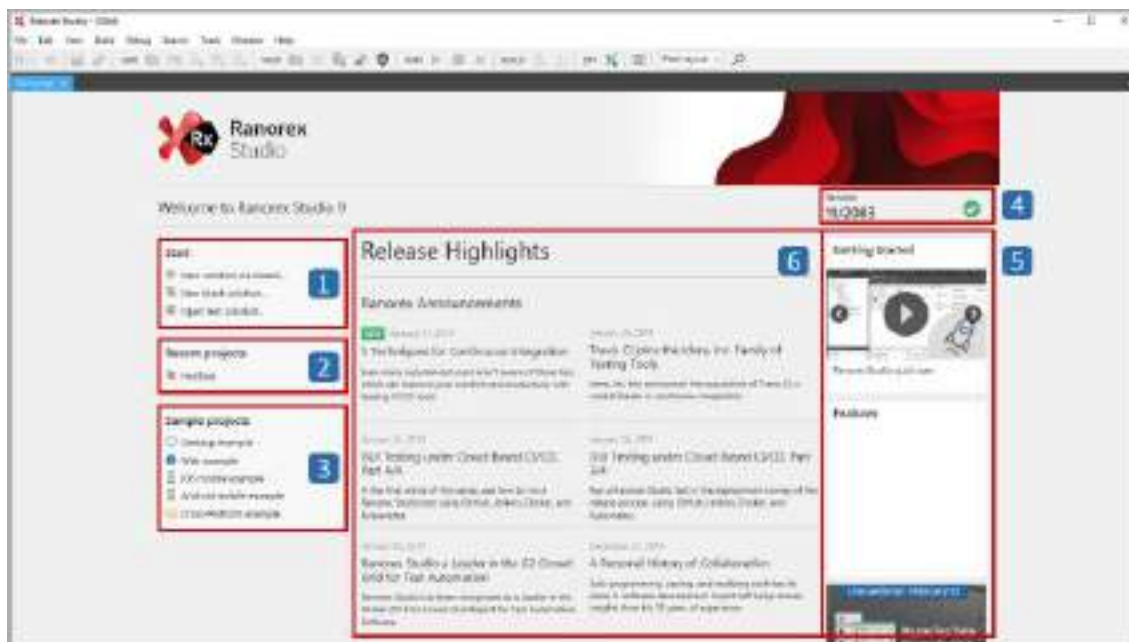


## Ranorex Studio start page

The Ranorex Studio start page is the base from which you start a new solution or open an existing one. It's also a valuable resource for the latest information from Ranorex.

### Ranorex Studio start page overview

The Ranorex Studio start page is the initial working environment after opening the program. Many of the page components described below are continually updated and so require an active internet connection.



1

#### Start section

Control center for creating a new solution or opening an existing solution

2

#### Recent solutions

List of recently opened solutions

3

#### Sample solutions

Available sample solutions

4

#### License information

Current license type with expiration date. Click this area to learn more about your current license. If you're using the 30-day free trial, the remaining time to use is displayed.

12 trial days left...



5

### Information region

Link to resources such as the User Guide and featured information on automated testing

6

### Release highlights and announcements

- Release highlights describing the current release of Ranorex Studio with a link to the release notes
- Announcements section including information such as new Ranorex Studio releases, upcoming webinars, and the latest informative articles in our automated testing blog

## Switch between light and dark theme

Ranorex Studio is available in a light and a dark theme.

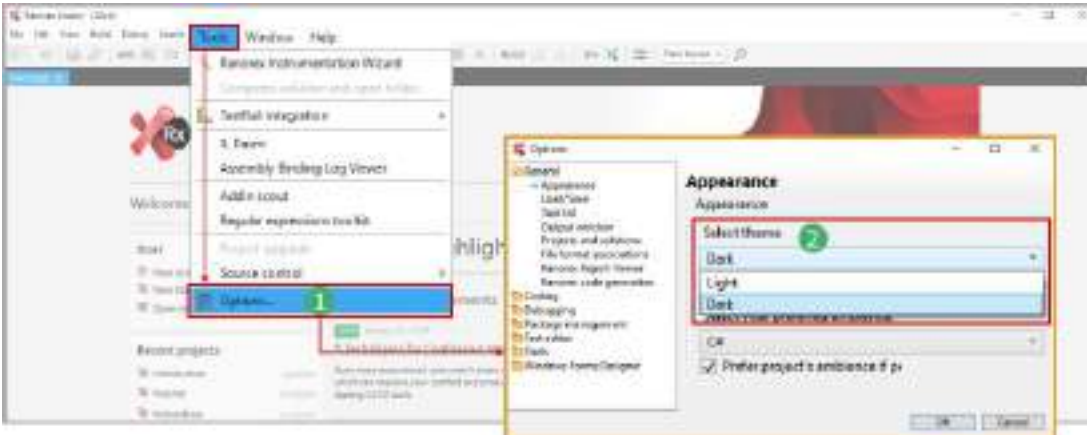
To switch between them:

1

Click **Tools > Options... > General > Appearance**.

2

Under **Select theme**, select your desired theme.



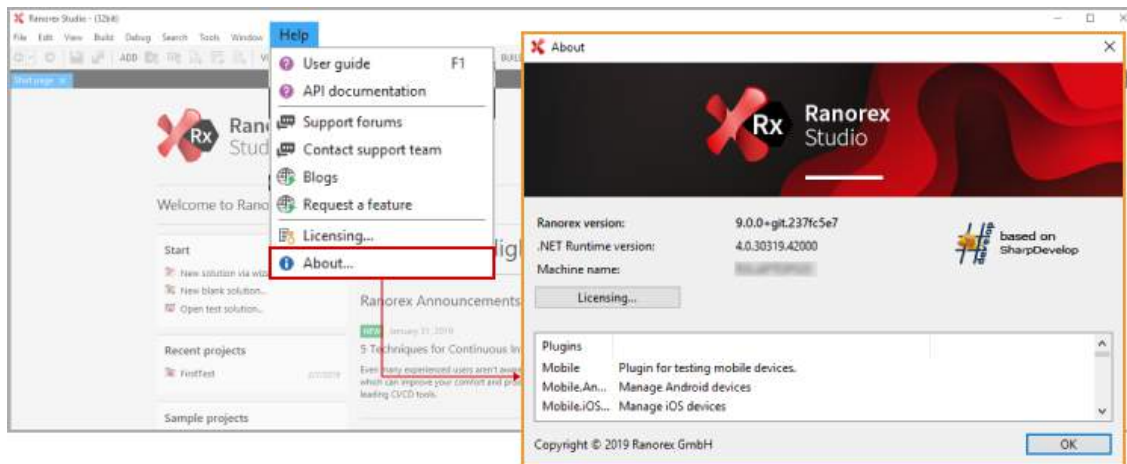
### **i** Note

You need to restart Ranorex Studio for theme changes to take effect.

## Version information

To view detailed information about your version of Ranorex Studio, follow the step below.

- 1 Go to **Help > About** in the menu bar.




## Available update


When you are connected to the internet, Ranorex Studio automatically verifies whether the current release is installed. If a newer version is available, a green information bar appears on the start page, similar to the example below.



## License information

Click the license information area for details about your current license model.

 Configure Ranorex License ✕

**Licensing**  Studio

☐ Node-locked ☒ Floating

License

Server

▼

↻

Type

Auto

▼

Status

Using License Manager

✓

Contact Support

Remove configuration

[Privacy statement](#) [EULA](#)

OK



## Further reading

Information about the available license models and the License Manager can be found in Ranorex Studio System Details > [→ Licensing](#)



## Sample solutions

The sample solutions on the Ranorex Studio Start page provide an introduction to key concepts of automated testing. Study these examples to learn how to create automated tests for desktop, web, and mobile applications. There is also a cross-platform sample solution that demonstrates how to perform tests on desktop, web, and mobile in a single test scenario.

### Desktop sample



This sample performs automated tests on a demonstration desktop application.

**Learn something about:** Variable binding, data connectors, user code actions, arguments, optional actions, smart folders, module groups, and global setup/teardown regions

### Web sample



This sample performs automated tests on an online blog tool.

**Learn something about:** Module library, popup watcher, project organization, report screenshot, validation, web testing, handling flyout menu, setup/teardown regions, and data binding.

## Mobile apps on Android and iOS



These samples for Android and iOS perform automated tests on a mobile application.

**Learn something about:** Mobile testing, Android, iOS, deploy mobile app, module groups, user code actions, setup/teardown regions, data connectors, and arguments

## Cross-platform sample



This sample performs automated tests on desktop, web, and mobile applications.

**Learn something about:** Cross-platform test, end2end test, iOS, Android, deploy mobile app, module library, project organization, popup watcher, web testing, mobile web testing, setup/teardown regions, and data binding.

## Create a new solution

In Ranorex Studio, a **solution is the top-level container** that contains all other test files. Solutions are organized into one or more **projects**. There are different project types in Ranorex Studio, but the **test suite project** is typically used to build a test. A test suite project contains one or more test suites, which in turn represent the structure of your test with its test cases and modules.

In this chapter, you'll learn how to create a new solution to test a desktop application. New solutions you create **always** come with a test suite project.



## Reference

If you want to add a project to an existing solution, refer to

Ranorex Studio fundamentals > Ranorex Studio > [→ Create a new project](#)

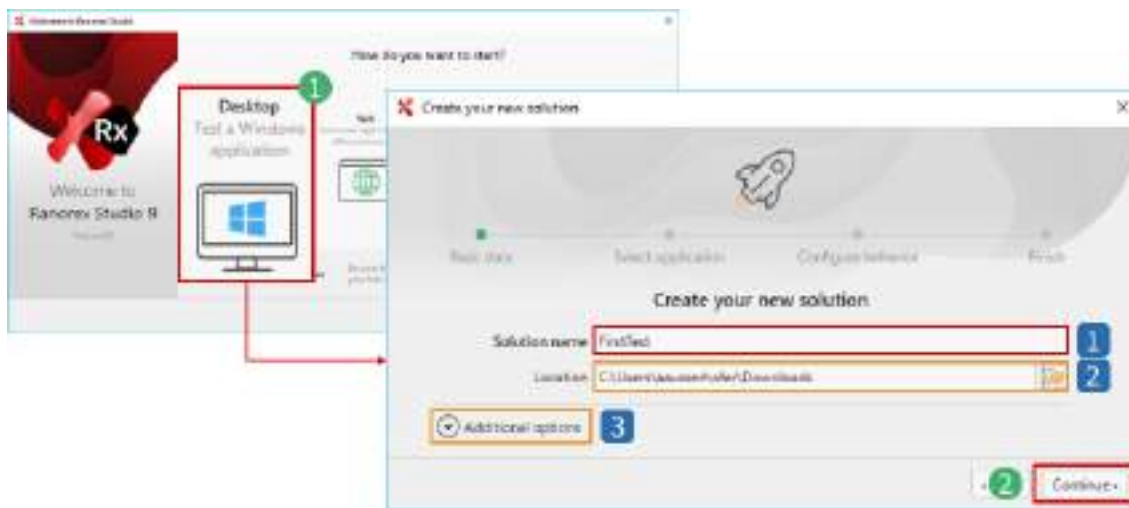
If you want to create a solution for web or mobile testing, refer to

Web and mobile testing > [→ Web testing](#) and

Web and mobile testing > [→ Android testing](#) / [→ iOS testing](#)

## RocketStart solution wizard

- 1 **Click File > New > Solution using wizard...** to open the RocketStart solution wizard. In the dialog that appears, **click Desktop**.
- 2 **Configure** your solution (see explanations below); then **click Continue**.



### 1 Solution name

- Give the solution a meaningful name or use the default name.
- By default, the solution name is used as the name for the test suite project in this solution, too.

## 2 Location

- Unless changed, the new project directory is created in `\Ranorex\RanorexStudio Projects\`

## 3 Click **Additional options** to access more options:

Additional options

Project name  4

Language  5

☒ Create directory for solution 6

☐ Add solution to source control 7

< Back 2 Continue >

- 4 As mentioned above, the test suite project name is identical to the solution name by default. Here you can specify a different name for the project.
- 5 Programming language of the project. C# is the default, VBNet is also available.
- 6 When deselected, the solution files are saved in the project's folder instead of in their own parent folder of the project's folder.
- 7 If desired, check the box to add your solution and the project to a [source control](#) system.

### Screencast

Watch the screencast “Improved start and close AUT actions” to see how release 9.1 increases the stability of your tests and the solution wizard.

[Watch the screencast now](#)

## Select the application under test

### 1 Choose one:

- a Click **Running applications** and **select** one of the listed applications.
  - b Click **Browse for application > Browse for app** and **browse** to an application using Windows Explorer.
- 2 Click **Continue**.

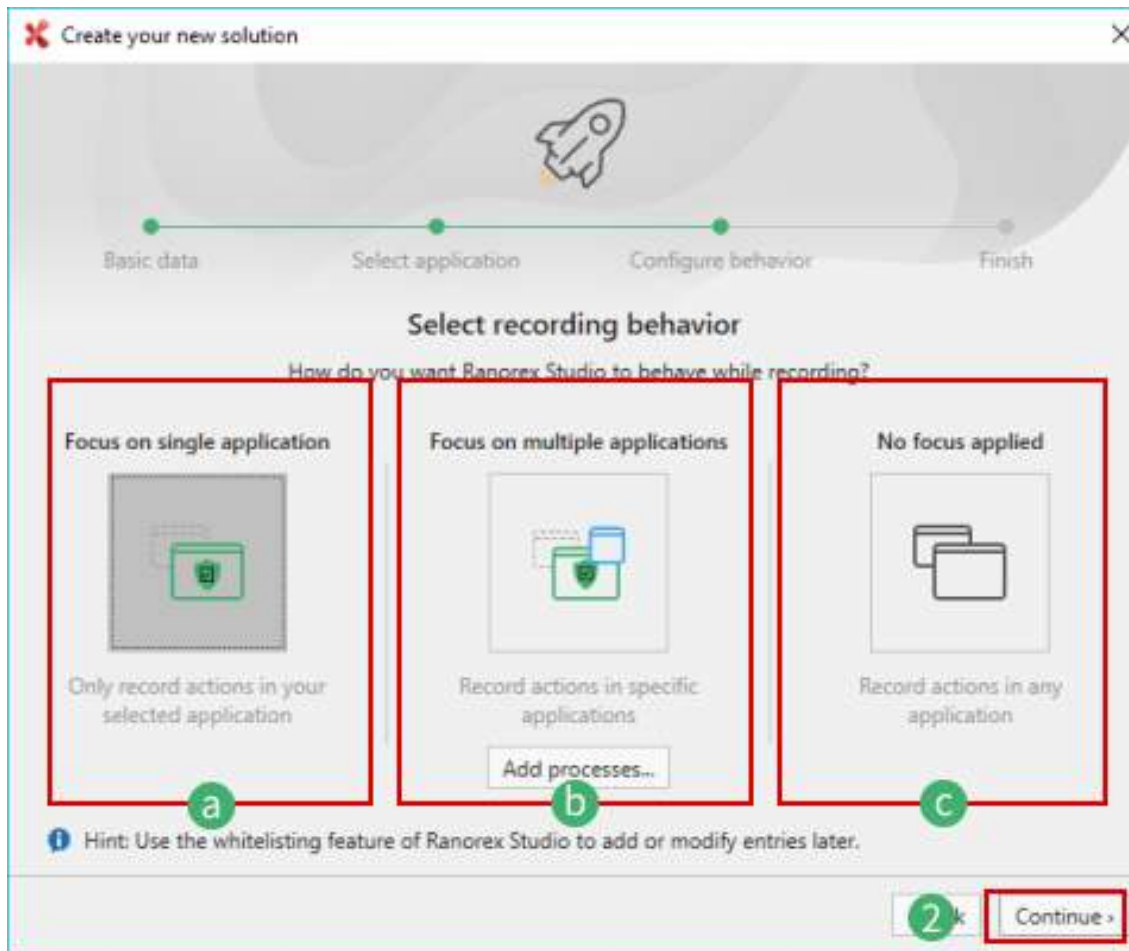


- 1 List of running desktop applications.
- 2 Section to browse for an application and specify optional command line arguments.

## Select recording behavior

Use this screen to configure whether Ranorex Studio evaluates just one application, several applications, or all running applications when searching for UI elements. This is part of Ranorex Studio's [whitelisting](#) capabilities.

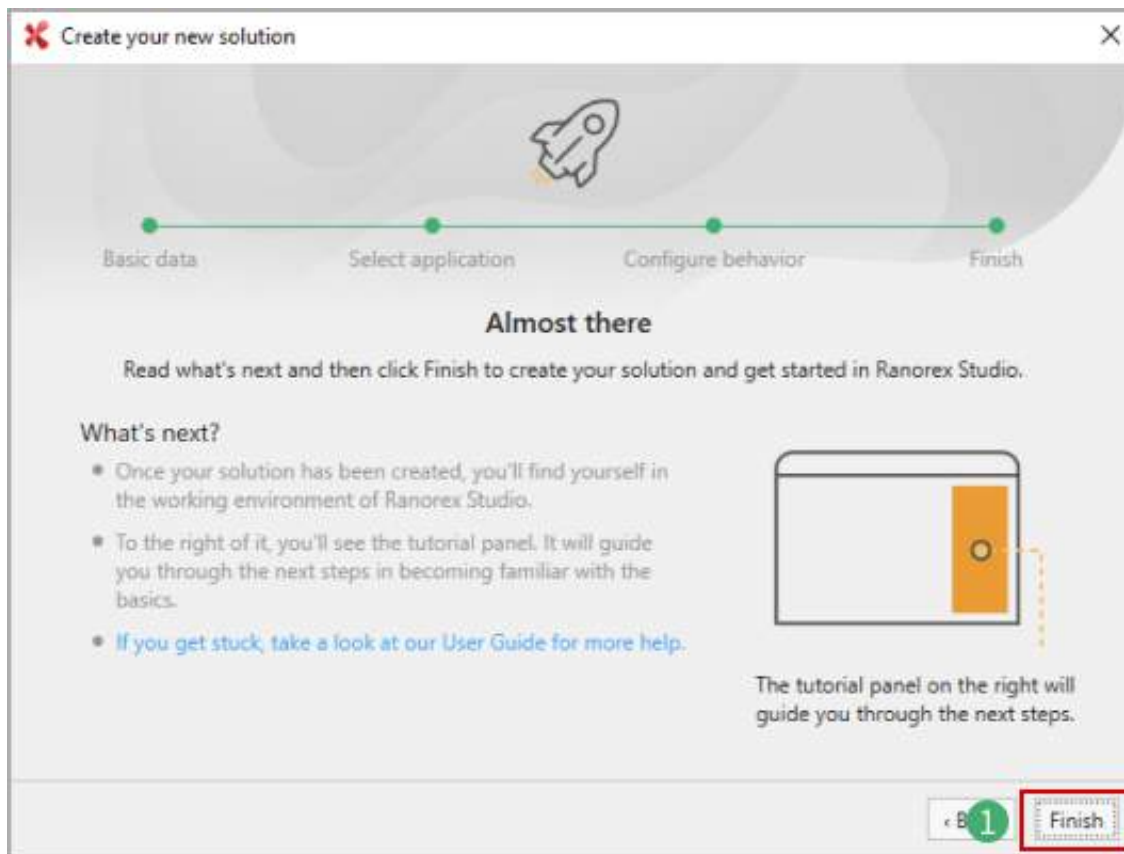
- 1 **Choose one:**
  - a **Focus on single application**  
Ranorex Studio records user interactions only in your previously selected application under test. Everything else is ignored. The whitelist contains only your AUT.
  - b **Focus on multiple applications**  
In addition to your AUT, you can add other processes to the whitelist to be recorded by Ranorex Studio.
  - c **No focus applied**  
All user interactions are recorded. The whitelist is empty.
- 2 Click **Continue**.



## Final screen and tutorial panel

The final screen explains what happens once setup completes, i.e. that the tutorial panel will appear and provide you with a quick tutorial to guide you through the first steps with your test.

- 1 **Read** through the instructions and then **click Finish** to complete setup.



## Structure of the desktop solution and its project

Once you've finished setup, Ranorex Studio opens in the test suite view with a simple prebuilt desktop test suite. This test suite is part of the test suite project that was created automatically with this desktop solution.



**1****Test suite view**

This is where you build and control your tests.

**2**

Your solution comes with a project that contains a simple prebuilt test suite. It contains a test case with three recording modules: **StartAUT** which starts the Demo Application, **Recording1** which is empty and ready for you to record test actions, and **CloseAUT** which closes the Demo Application.

**3****Recording module view**

In the recording module view of **Recording1**, you can record and manage test actions.

**4****Empty actions table**

This is where your recorded actions appear.

**Reference**

For more information on recording tests, refer to:

Ranorex Studio fundamentals > → [Ranorex Recorder](#)

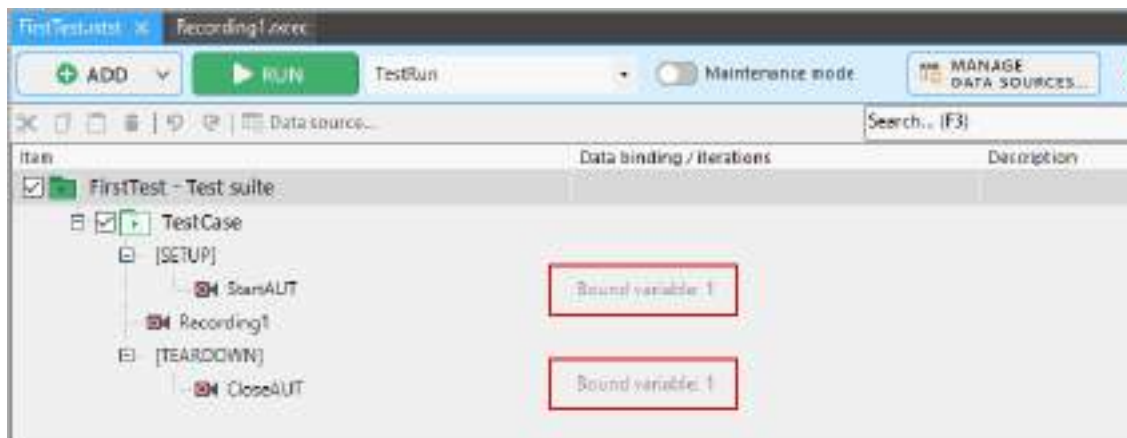
For more information on test suites, refer to:

Ranorex Studio fundamentals > → [Test suite](#)

**StartAUT and CloseAUT modules**

The StartAUT and CloseAUT modules in the desktop solution template use a special mechanism to ensure the correct instance of the started AUT is closed. This improves stability for tests based on the template. The mechanism is based on the advanced concept of variables and parameters, which is why the solution comes with a variable bound to both modules.





### Note

You don't need to be familiar with variables and parameters to use the desktop solution template.



### Reference

If you want to find out more about variables and parameters, refer to

Ranorex Studio advanced > Data-driven testing > → [Define variables](#)

Ranorex Studio advanced > Data-driven testing > → [Parameters](#)

## Create a new project

Solutions are organized into one or more **projects**. There are different project types in Ranorex Studio, but the **test suite project** is typically used to build a test. A test suite project contains one or more test suites, which in turn represent the structure of your test with its test cases and modules.

Solutions you create always come with a test suite project. You can also add more test suite projects or other types of projects to a solution. We recommend you create **test suite projects** with the **project wizard**. All **other project types** must be created with the **blank project dialog**.

## Project wizard

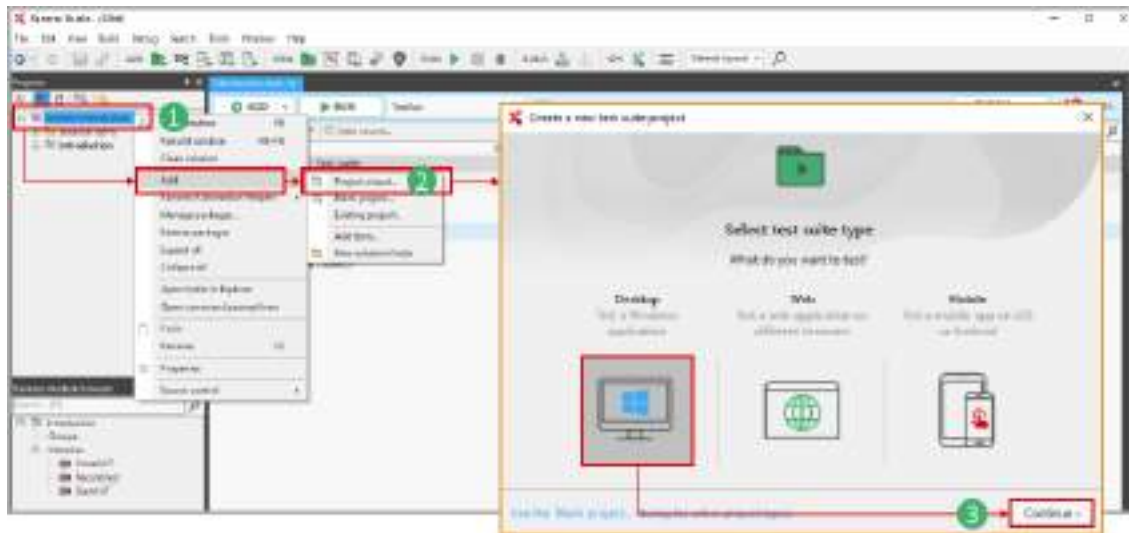
Use the project wizard to quickly create a test suite project for desktop, web, or mobile tests.

### Note

All other project types must be created with the blank project dialog.

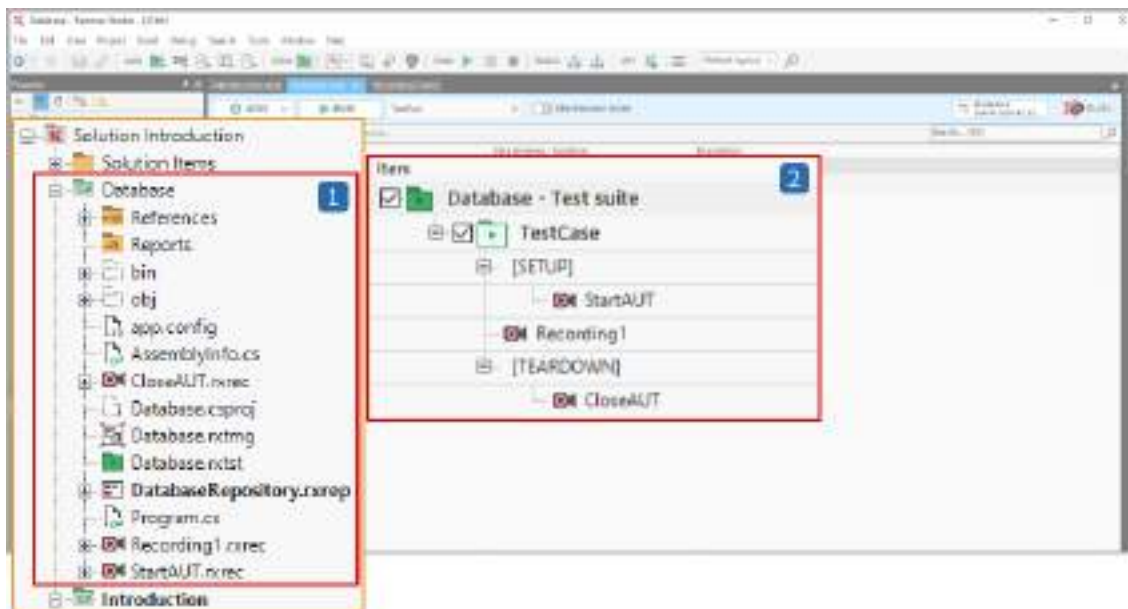
To add a test suite project with the project wizard:

- 1 In the projects view of an open solution, **right-click** the solution.
- 2 **Click Add > Project wizard...**
- 3 **Select** the test suite type (e.g. Desktop) and **click Continue**.



The following steps are the same as those for [creating a new solution](#) with the RocketStart wizard.

When you've finished the project wizard, Ranorex Studio will display the test suite in the test suite view.



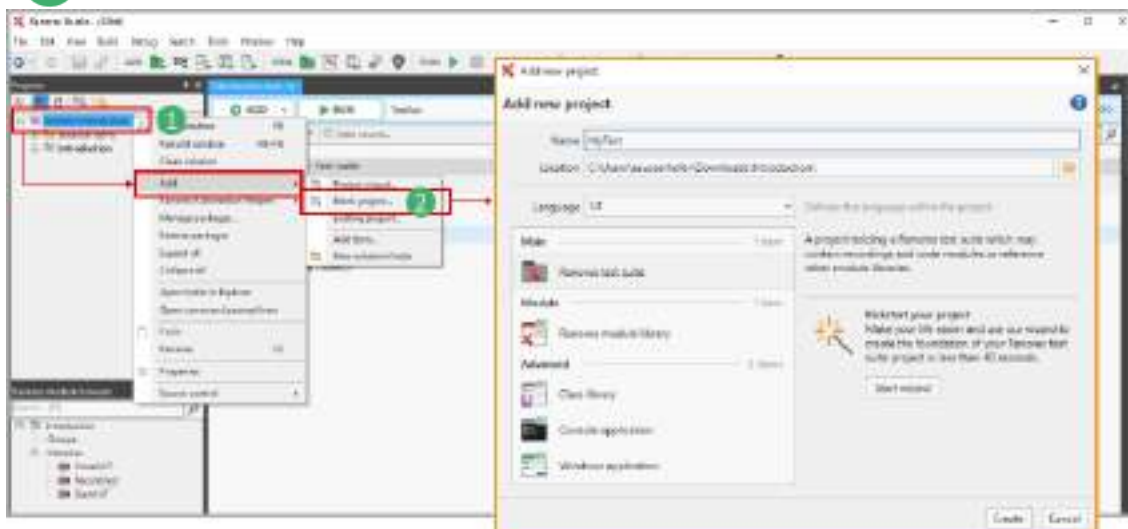
- 1 The new test suite project in the projects view.
- 2 The project's test suite with the prepared test structure.

## Blank project dialog

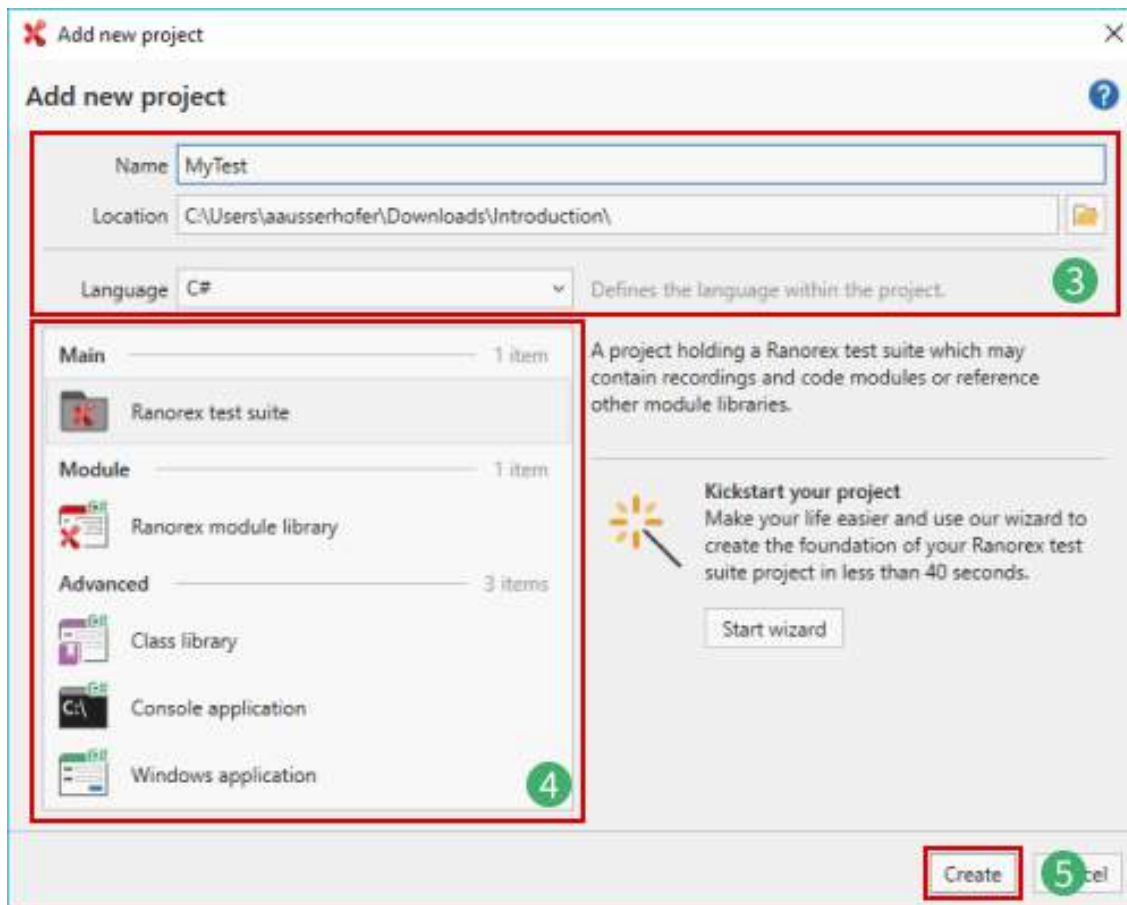
You can also create new projects with the blank project dialog. Project types other than test suite projects must be created with this dialog.

To do so:

- 1 In the projects view of an open solution, **right-click** the solution.
- 2 Click **Add > Blank project...**



- 3 **Name** the project and, if you want to, **change** the location the project is saved to and the programming language.
- 4 **Select** a project type (explained further below).
- 5 **Click Create.**



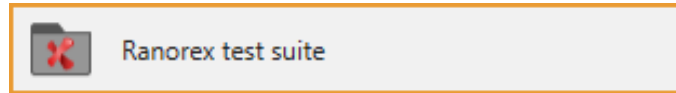
## Project types

There are 5 different project types you can create in Ranorex Studio.

### Note

For testing, usually only the test suite project and the module library project are relevant. The other project types are for when you want to program and application with the Ranorex Studio IDE.

## Test suite project



The test suite project is the default project in Ranorex Studio. Every new solution comes with this type of project. This project can also be created with the project wizard (recommended). Use this project to create tests.

### Properties

- Used to contain:
  - one or more test suites
  - one or more repositories
  - recording modules, code modules, or module groups
- This project can be executed because Ranorex Studio creates an executable when building this project.

## Module library project

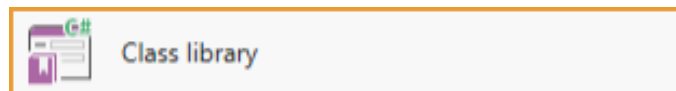


Use this project type to store, organize, and manage module libraries and repositories.

### Properties

- Used to contain:
  - one or more repositories
  - recording modules, code modules, or module groups
- This project can't be executed because Ranorex Studio creates a DLL when building this project.

## Class library project



Use class library projects to contain classes and methods for use in programming applications.

### Properties

- Used to contain classes and methods for programming.

- This project can't be executed because Ranorex Studio creates a DLL when building this project.

### **Console application project**

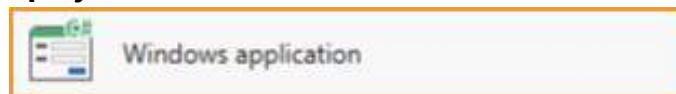


Use this project to program and compile a command-line application from classes and methods.

### **Properties**

- Used to contain classes and methods for compiling a command-line application.
- This project can be executed because Ranorex Studio creates an executable when building this project.

### **Windows application project**



Use this project to program and compile a Windows application from classes and methods.

### **Properties**

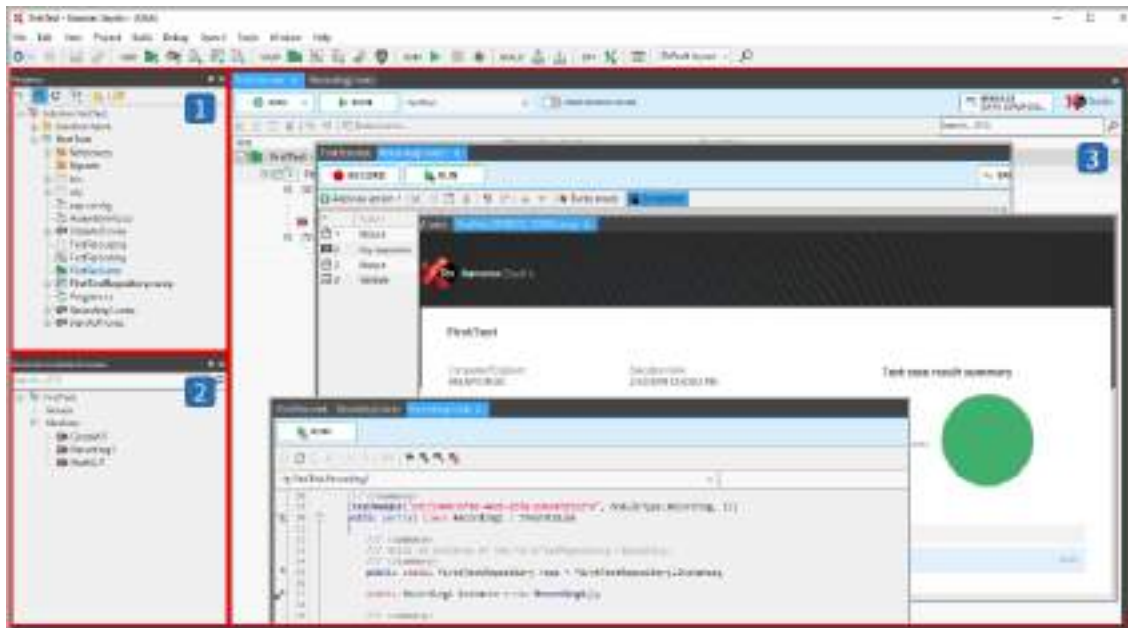
- Used to contain classes and methods for compiling a Windows application.
- This project can be executed because Ranorex Studio creates an executable when building this project.

## **Working environment and views**

The Ranorex Studio working environment provides a central place to manage your test solution and all of its components. This chapter describes the basic working environment and default views.

### **Basic working environment structure**

Ranorex Studio provides a working environment that is simple and organized according to common principles, such as a hierarchical organization. Refer to the overview below for an introduction to the structure of the working environment.



1

## Projects pad

- A Ranorex Studio project is based on files and uses the same project file format as Microsoft Visual Studio 2008/2010.
- The projects view shows all files and references currently associated with the project.

2

## Module browser pad

- The module browser lists all available code and recording modules in the project's code files. It also lists all module groups in the project's module group file.
- In addition, it shows all the variables defined by a module or module group.
- This view is mainly used to drag and drop automation modules and module groups, and to reuse modules and module groups within the test suite view.

3

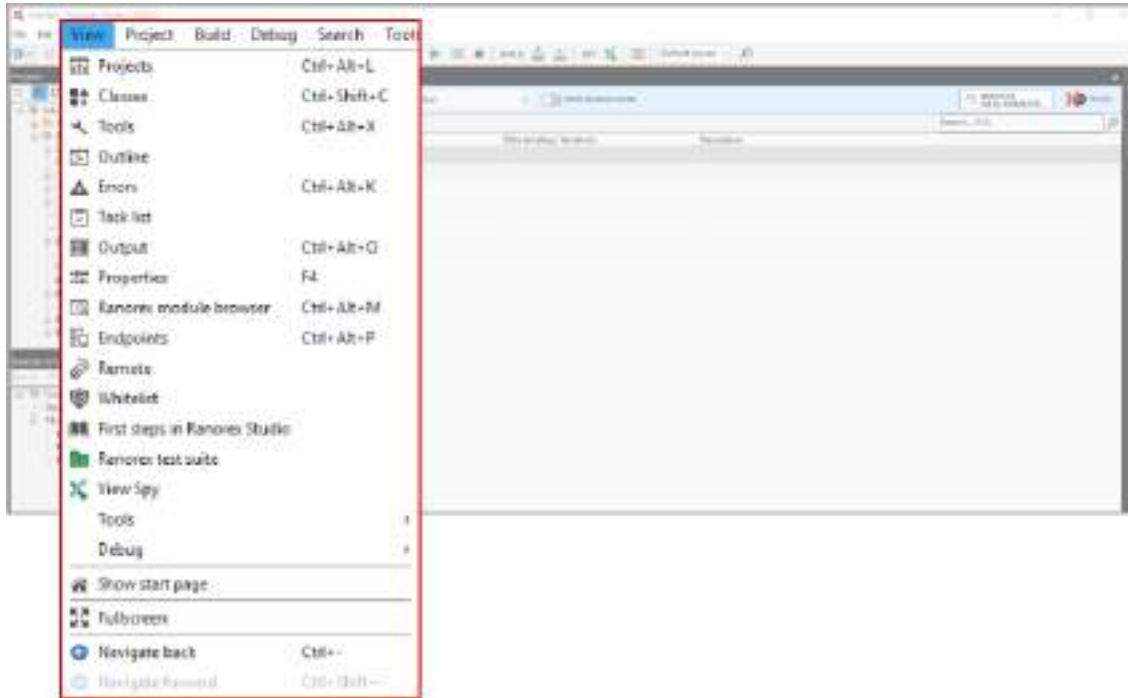
## File view

- Double-click a file in the projects view or a module in the module browser to open the associated file in Ranorex Studio's file view.

- This view displays all available files, such as the test suite view, Recorder view with action table and repository, reports, code modules, and many more.

## Adding views

For a list of all available views, click **View** in the Ranorex Studio menu bar.



### Note

The different views are described in the corresponding chapters of the user guide.

## Changing and selecting views

Add views to the working environment using the **View** menu of the Ranorex Studio menu bar. To switch between active views, click **the corresponding register tab** in the file view.





1

### Test suite view (\*.rxtst)

Test suite files have the file extension **rxtst** = **R**anorex **t**est **s**uite

2

### Recorder view (\*.rxrec)

Recorder files have the file extension **rxrec** = **R**anorex **R**ecorder

3

### Report file (\*.rxlog)

Report files have the file extension **rxlog** = **R**anorex **L**ogfile

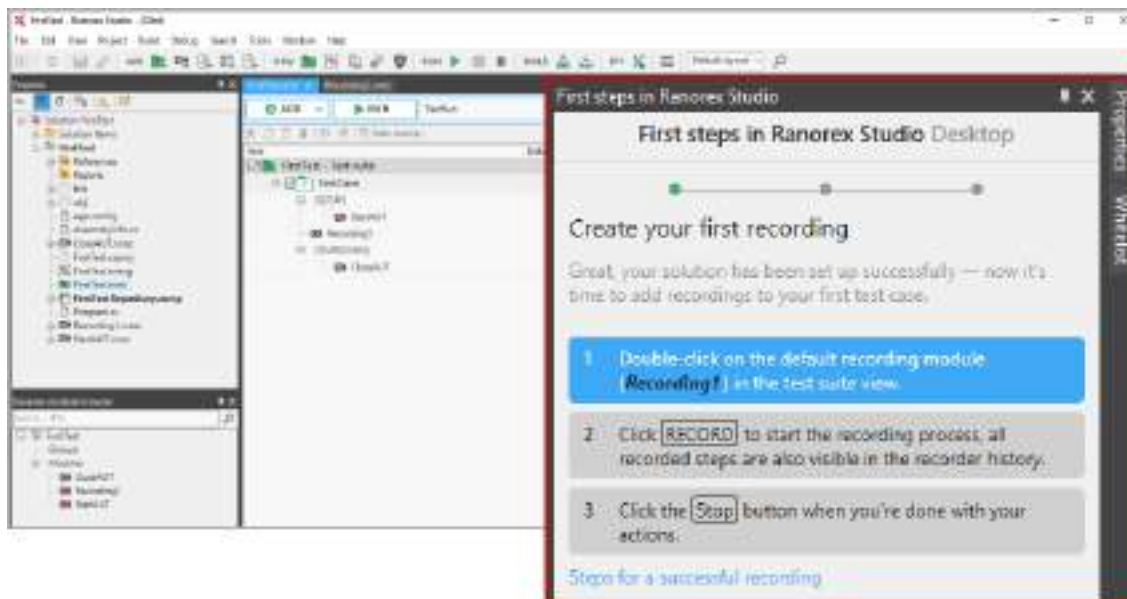


### Hint

Remove an active view from the working environment by clicking the **[x]** next to the file name in the register tab.

## Tutorial panel

When you create a solution using the RocketStart solution wizard, the tutorial panel will be displayed by default. It displays a quick tutorial to guide you through the first steps of working with Ranorex Studio.



You can close the tutorial by clicking the **X** and bring it back up again by clicking **View > First steps**.

# Ranorex Recorder

The Ranorex Recorder allows you to **record** the keyboard and mouse **actions** necessary **for a user interface test**. These actions **appear** in the Recorder's **action table**, where you can **edit** them or **add more**. This way, you can create recordings to suit your testing needs. The Ranorex Recorder is available as an **integrated tool in Ranorex Studio** and as a **stand-alone version**.

## Integrated Recorder

When you start a **new** test project, the working environment appears with an empty recording module opened.

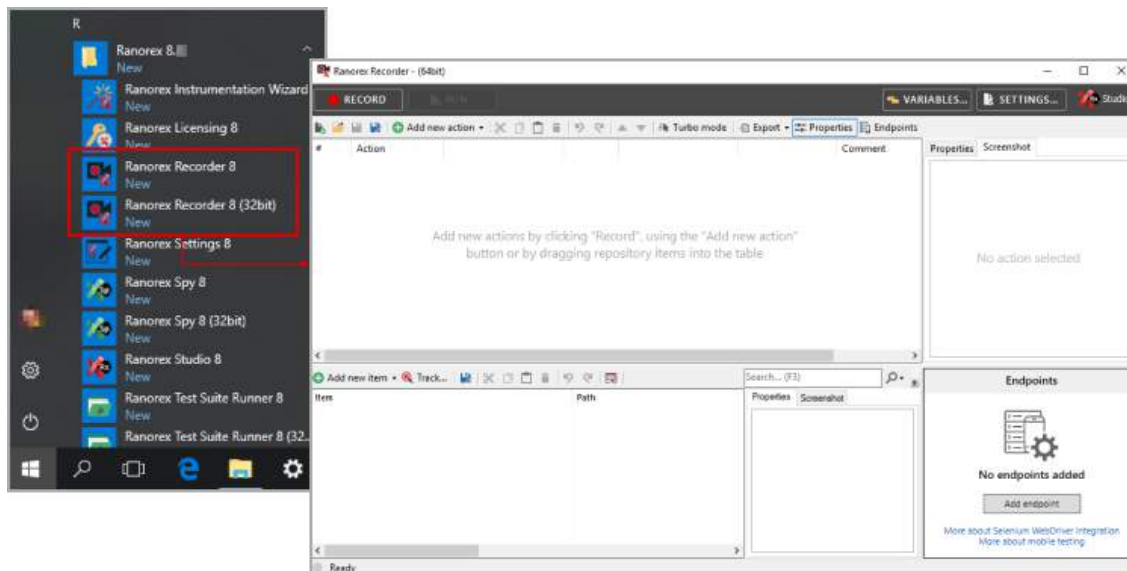


1 Active recording module **Recording1.rxrec**

2 Control panel of the integrated Recorder

## Stand-alone Recorder

Ranorex also provides a stand-alone Recorder. It can be started independently from Ranorex Studio.



- 1 **Open** the Windows Start menu.
- 2 **Click Ranorex > Ranorex Recorder 8.**
- 3 **Start** the stand-alone Recorder in the desired version (32 or 64 bit).

## Integrated vs. stand-alone Recorder

Both Recorder versions are identical in their basic functionality. However, the stand-alone Recorder has fewer features. **That's why we recommend that you use the integrated Recorder.**

One major difference is that the stand-alone Recorder **does not allow user code actions**, which are popular in automated testing. The stand-alone Recorder also uses an embedded repository by default. This can be limiting when dealing with more complex testing tasks.

For these reasons, the **User Guide focuses on the integrated Recorder.**

## Before you start recording

Whether your test is simple or complex, **preparation is key** in test automation. Preparation **prevents mistakes** and **saves time** in the long run. This chapter describes the steps to take before you start recording your test.



## Screencast

To learn how to configure recorder settings and enable process whitelisting to ensure optimum performance when recording an automated test, watch our video

[Watch the screencast now](#)

## Recording recommendations

Take the time to consider the following recommendations. They can make your testing life easier and your work more efficient.

<b>Application under test (AUT)</b>	¿Do you want to start the AUT yourself each time, or do you want Ranorex Studio to start it automatically at the beginning of the recording? In either case, you need to know the installation path and any required starting parameters.
<b>Test data &amp; parameters</b>	Make sure you have all test data and parameters, such as login names and passwords, readily available for the test.
<b>AUT instances</b>	Only run one instance of your AUT during recording. If you've created your solution using the RocketStart solution wizard, Ranorex Studio will start (and quit) your AUT automatically by default.
<b>Recording interaction</b>	Unless you use → <a href="#">whitelisting</a> , all user actions are captured during recording, even accidental ones that are unrelated to the AUT. Make sure to disable anything that could interfere with recording, such as pop-ups.

## Our AUT: The Ranorex Demo Application

To better demonstrate the concepts and methods in this User Guide, we've created a special app, the **Ranorex Demo Application**. It will serve as our AUT for the explanations and samples in the fundamentals and advanced chapters of this User Guide.

## Download the Demo Application



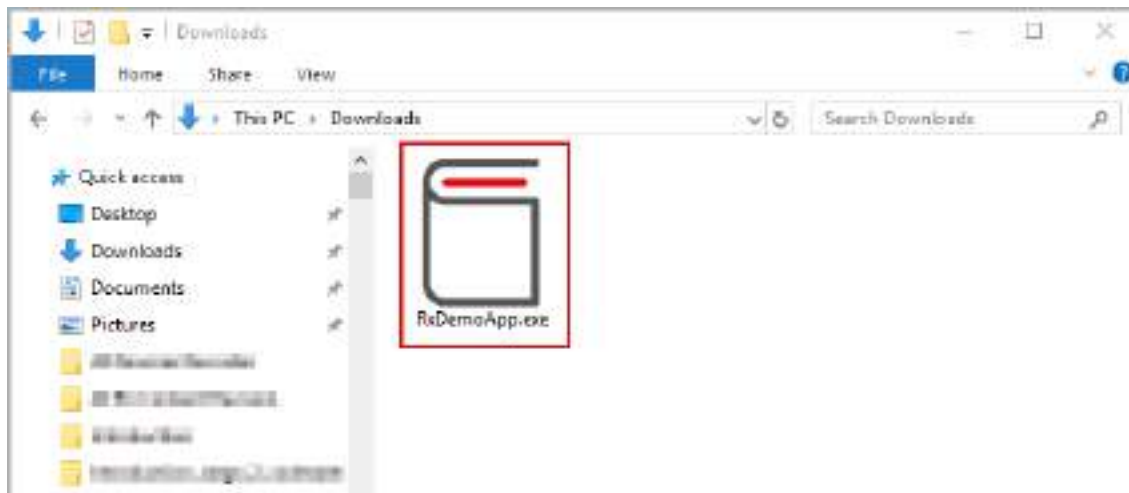
### Download

Download the latest version here: → [Ranorex Demo Application](#). Unzip it to any folder on your system. For the purpose of our tutorials, we assume that it's located in the `/Downloads/` folder.

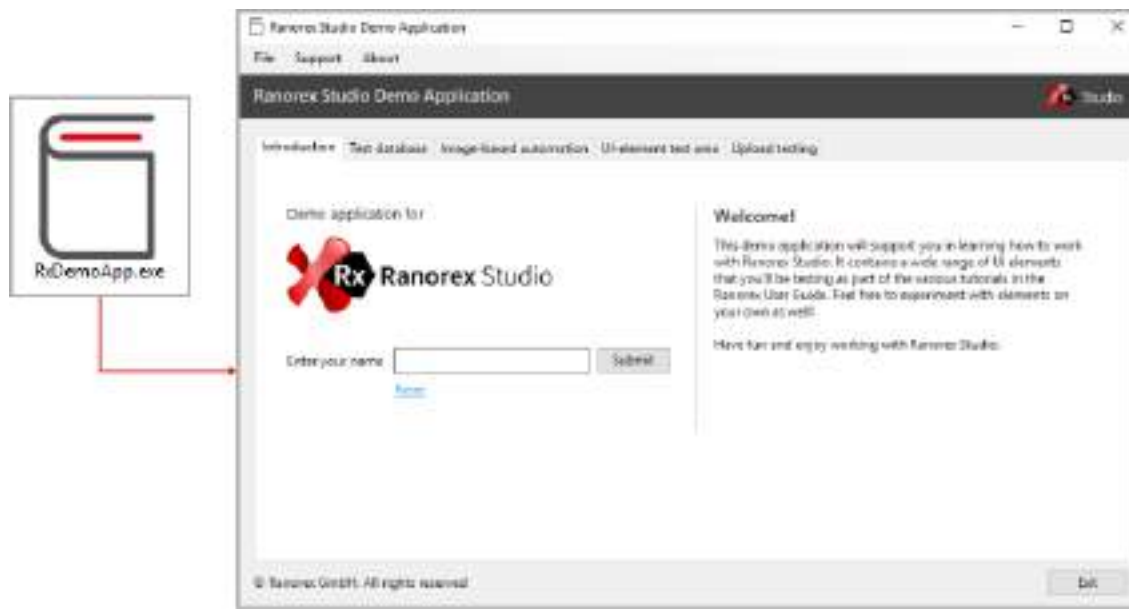
## Start the AUT

Before planning a test in detail, **start the AUT to ensure that it is installed properly and is working.**

- 1 **Browse** to the location where you unzipped the Demo App.
- 2 **Double-click** it.



- 3 The Demo App's Welcome screen appears.
- 4 After you've checked the Demo App opens and works correctly, close it again.



### Note

A link to the demo application is also included with all downloadable sample solutions throughout the fundamentals and advanced chapters of this user guide.

## Plan the recording

Plan your recording well. It helps prevent tedious restructuring and editing later.

### Use a recording script

- Think about the steps you need to record.
- Prepare the necessary input data (usernames, passwords, etc.).
- Perform the test at least once without recording.

### Consider the recording size

- Keep recordings as small as possible.
- Larger, modular test cases can be built out of smaller, reusable recordings.

### Plan how mouse movements will be recorded

- By default, **mouse movements are not recorded** without mouse clicks.

- When navigation through menus is important for the test, use mouse clicks for detection, or use the option for recording mouse movements within the Recorder control center.



## Reference

Learn about recording mouse movements in Ranorex Studio fundamentals > Ranorex Recorder > → [Recorder control center & hotkeys](#).

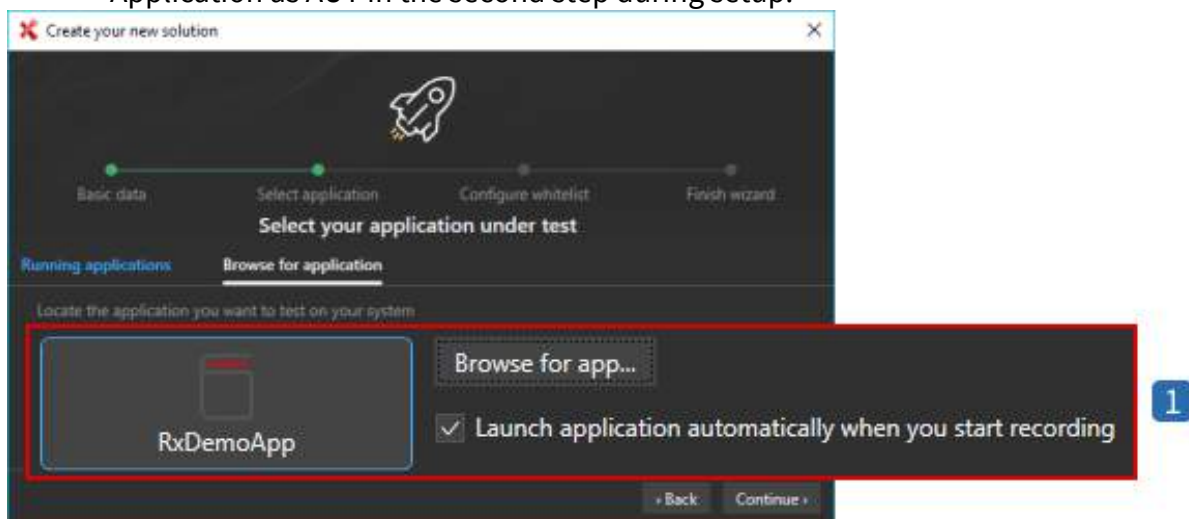
## Test definition

Following the recommendations above, we have defined a simple test with five steps. This same test serves as an example in the coming Ranorex Recorder chapters.

1. **Open** the Ranorex Demo Application.
2. In the “Enter your name” field and click **Submit**.
3. **Verify** that the welcome message changes accordingly.
4. **Reset** the welcome message.
5. **End** the demo application and **stop** the recording.

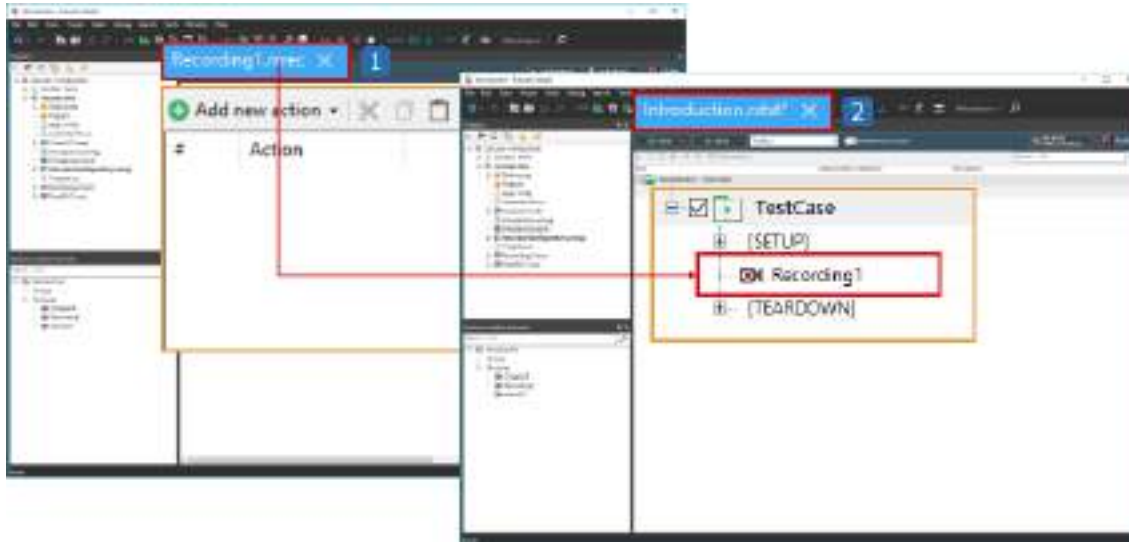
## Create a desktop test solution

- 1 In the Solution Wizard, → [create a desktop test solution](#) and **select** the Demo Application as AUT in the second step during setup.





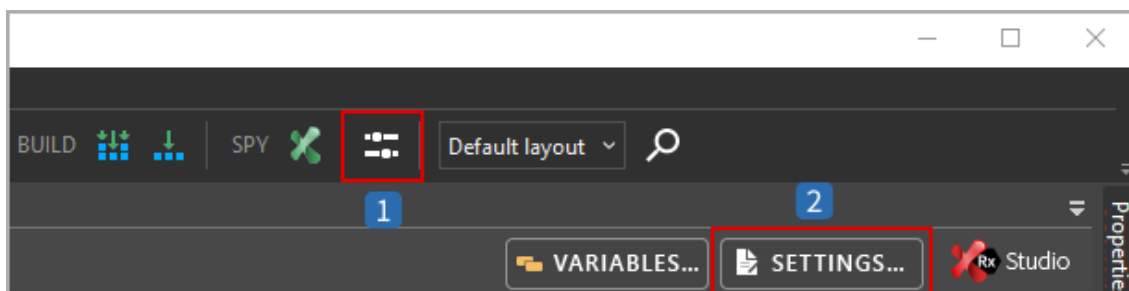
- 1 Demo Application selected as AUT and set to launch automatically when you start recording.
- 2 Once you're done, the solution will open in Ranorex Studio as shown below:



- 1 The empty recording module **Recording1** in the recording module view. In the test suite view, **Recording1** is located in a test case that has setup and teardown regions.
- 2 The former contains a recording module that starts the Demo Application, the latter contains one that closes it.

## Recorder default settings

There are many settings you can configure to customize the behavior of the Recorder. The **default settings are usually suitable** for most recording tasks, including the ones in this chapter. However, for some recording tasks, you may need to **change the Recorder settings**. To access them, **click Settings** in the toolbar of the Recorder working environment or in the Ranorex Studio toolbar.



- 1 Global Recorder settings
- 2 Local Recorder settings valid for the current recording module



### Further reading

Learn about Recorder settings and their impact on recordings in Ranorex Studio system details > Settings and configuration > → [Ranorex Recorder settings](#).

## Record a test

In this chapter, you'll learn how to record a test and become familiar with the underlying automation concepts.



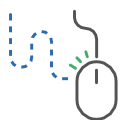
### Screencast

To learn how to create a basic recording with a validation step, watch our video:

[Watch the screencast now](#)

## Steps for successful recording

Ranorex Studio supports a wide range of environments, specifications, and settings. To make **your first recording a success** regardless of your environment, **we recommend the following:**



### While recording, use only the mouse for navigation

Avoid using other input devices such as graphic tablets, touchpads, pens, etc.



### Click every step

Do not use the tab key to navigate through forms.



### Close other applications

Close any other applications or tools that you don't need for the test.



### Open the user guide on another machine

If possible, open the user guide on another machine or tablet for reference while recording your test.



### Click pause/continue in the Recorder

To check back to the user guide or perform actions that you don't want to be recorded, just click the **Pause** button in the Recorder control center.



### Set display scaling to 100 %

In the Windows display settings, set display scaling to 100 % for all of your displays.

## Record the test



### Hint

Remember, if you're not using whitelisting, all user interactions are captured once recording has started, even if they are not performed on the AUT.

- Click **Pause** to pause recording. Click **Continue** to resume recording.
- Click **Stop** to end recording.

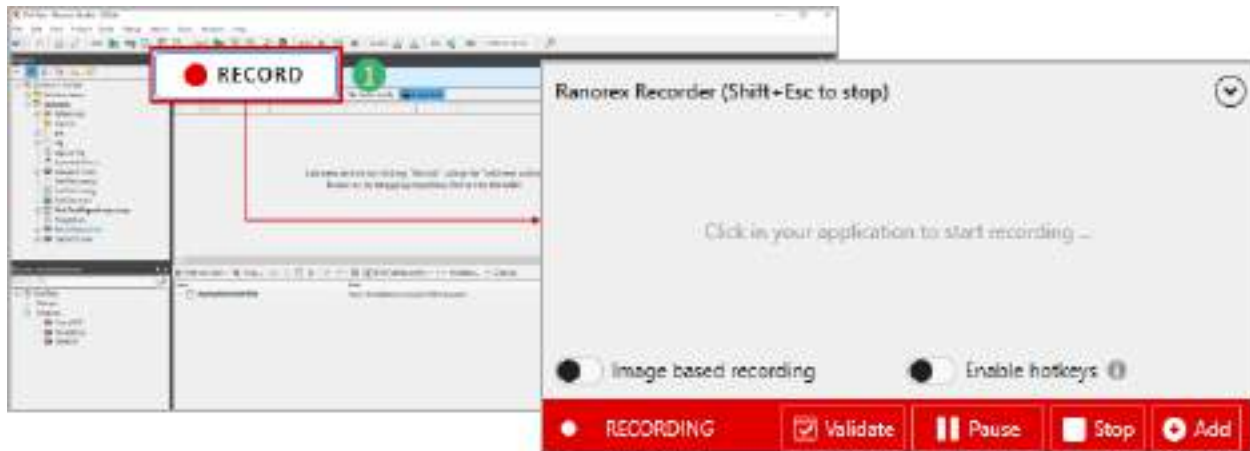
Learn more about the Recorder control center in Ranorex Studio fundamentals > Ranorex Recorder > → [Recorder control center & hotkeys](#).

Read about whitelisting in Ranorex Studio fundamentals > → [Whitelisting](#).

1

To start recording, **click Record**. Ranorex Studio is minimized to the taskbar automatically.

The Recorder control center shows that recording is active.



2 The application under test comes into focus.

3 In the text field and **click Submit**.

## Validate the test

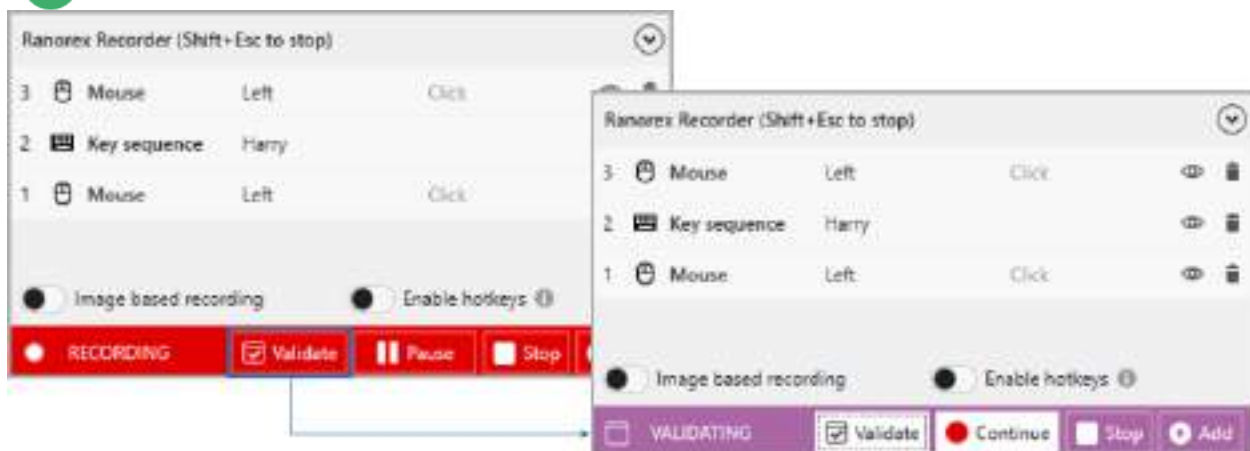
We've recorded the UI interaction. Now it's time to **validate** whether the interaction has led to the desired result, i.e. if the welcome message has changed accordingly.



### Further reading

The concept of test validation is introduced in Ranorex Studio fundamentals > Test validation > → [Introduction](#).

1 Click **Validate**. Recording pauses and the Recorder switches to validation mode.



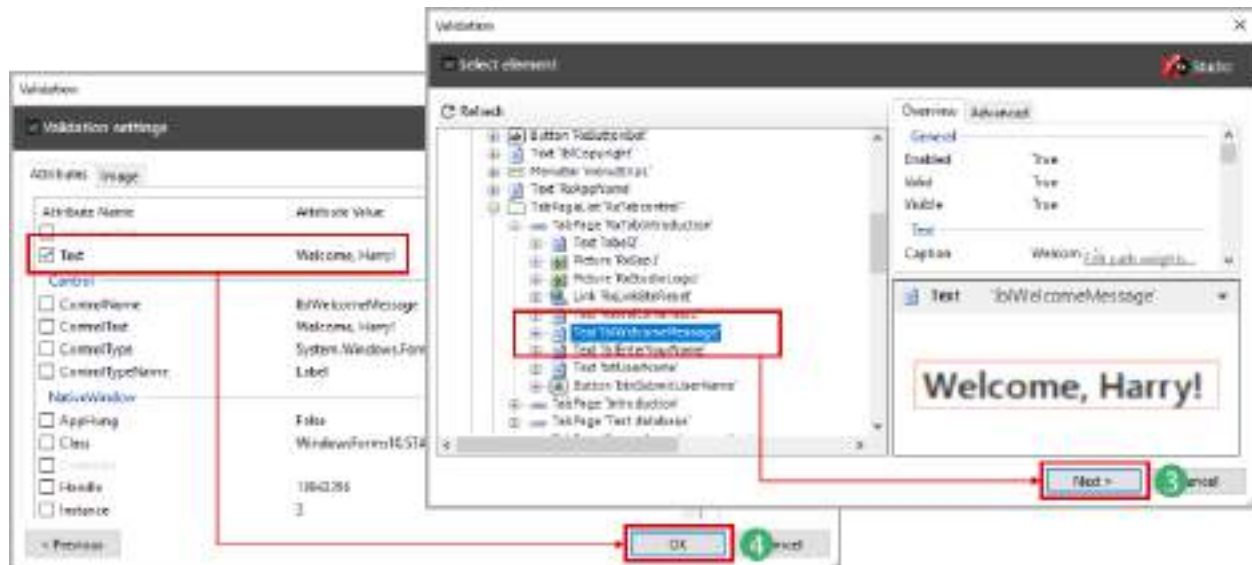
## 2 Select the UI element to validate:

- **Mouse over** the changed welcome message. A **purple frame** follows your mouse movement.
- The purple frame indicates which element is currently selected for validation.
- Once your selection matches the welcome message, **click it**.

## 3 To confirm the UI element, **click Next**.

## 4 Select the validation attributes:

- **Text** and **Visible** are the default validation attributes for text-based validations. No other attributes are required.
- To confirm the selection, **click OK**.

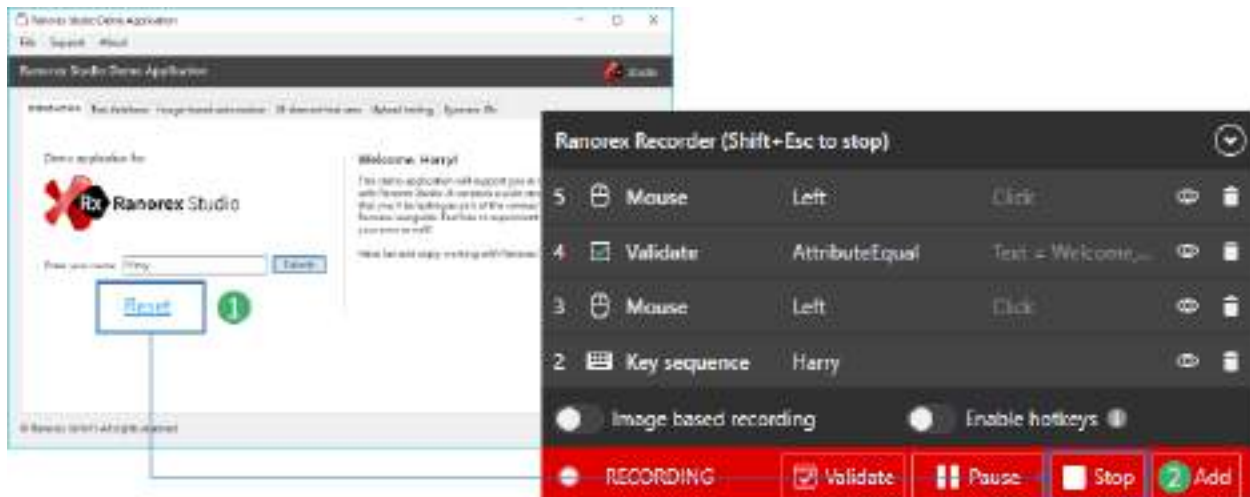


## Finalize and end recording

Once you've finished the validation action, Ranorex Studio automatically continues recording. The next step is to finalize and end the test recording.

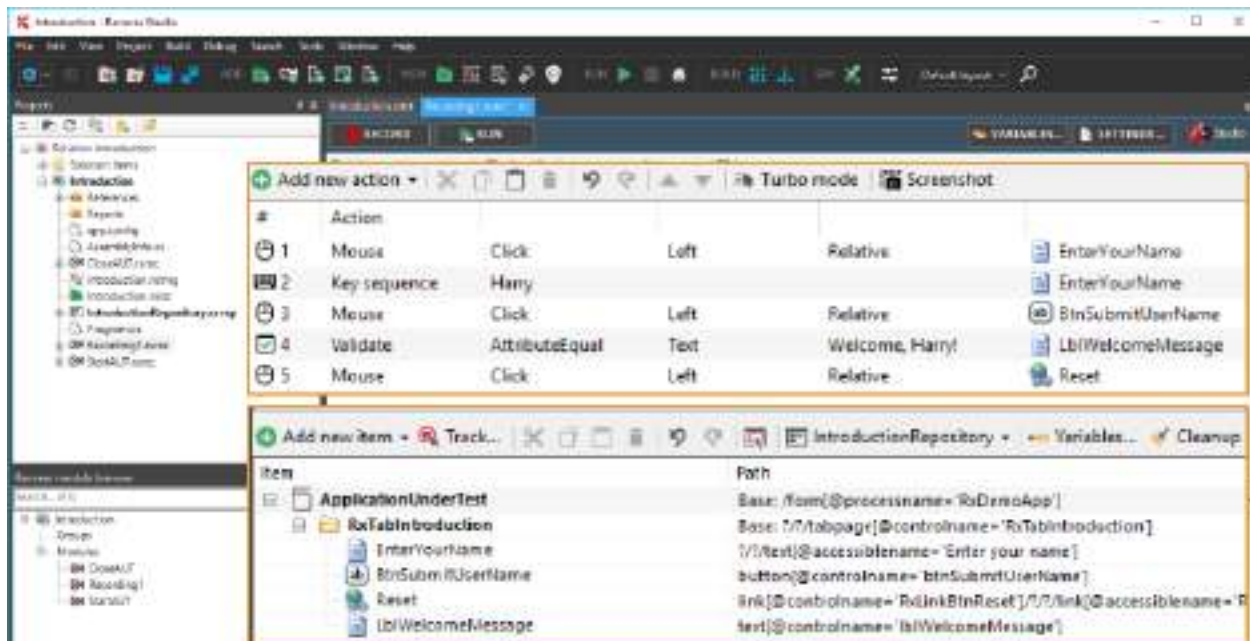
## 1 Click **Reset** to reset the welcome message to its initial state.

## 2 In the Recorder control center, click **Stop** to end the recording.



## Result(s):

The recording ends, and you return to the Recorder view. If you followed the instructions exactly, you should see **5 recorded actions** and **4 identified UI elements**, organized in two folders as shown below.



## Download the sample solution

You can build your own test solution according to the instructions in this chapter. Alternatively, you can download a prepared sample test solution.

## Installation:

- 1 Unzip to any folder on your computer
- 2 Start Ranorex Studio and open the solution file `Introduction.rxsln`



### Hint

The sample solution is available for Ranorex Studio versions 8.0 or higher. You must accept the automatic solution upgrade for versions 8.2 and higher.

## Analyze a recording

In this chapter, you'll learn more about the Recorder working environment and analyze the recording of the example test.

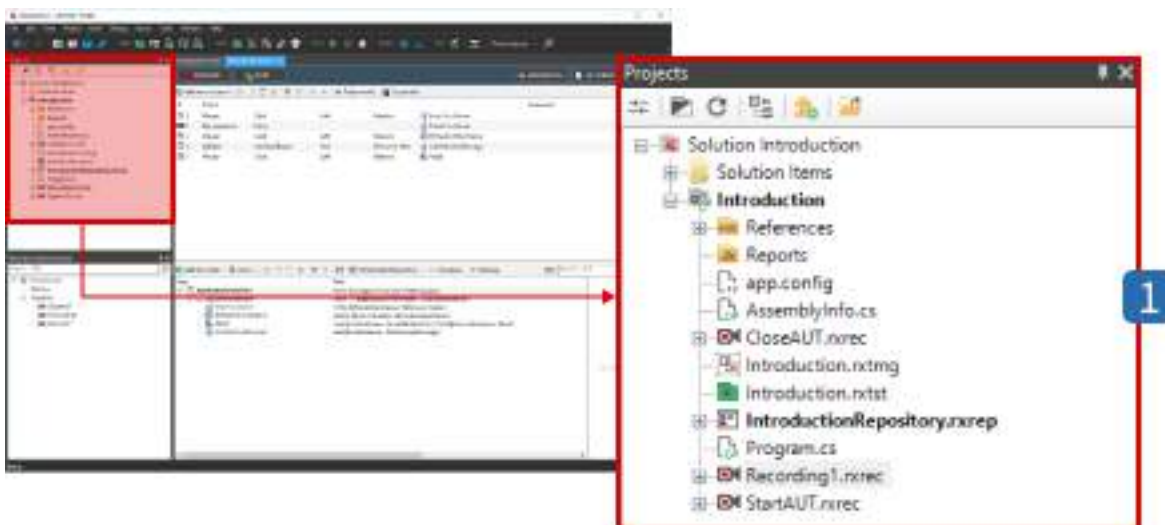
### Recorder working environment

The Recorder working environment has four regions.

1

#### Projects view

Located in the upper left, the projects view displays all files that are created and managed as part of a test project in a folder hierarchy.





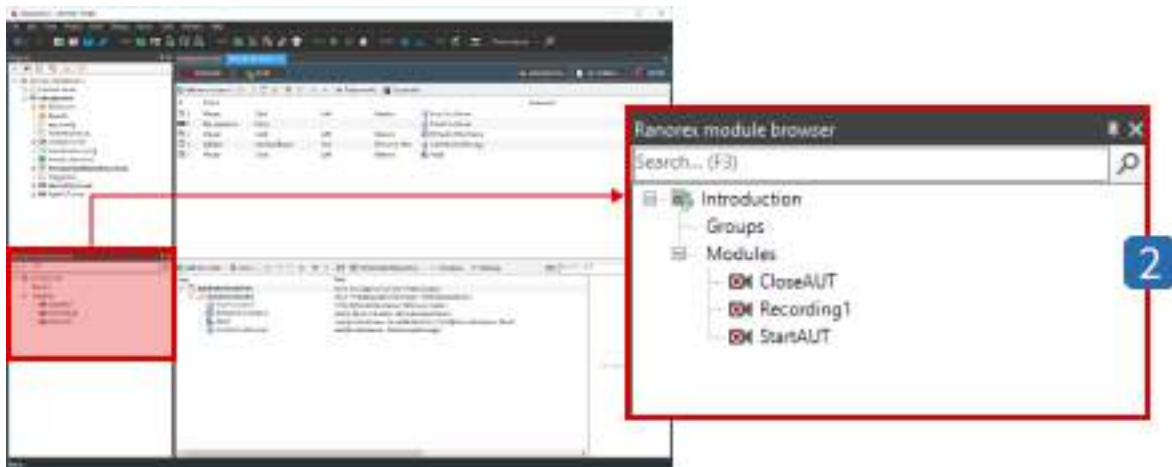
## Further reading

The projects view is discussed in Ranorex Studio fundamentals > Ranorex Studio > [Ranorex Studio Start page](#).

2

### Module browser

Located in the lower left, the module browser shows all modules and module groups that can be used to build tests.



## Further reading

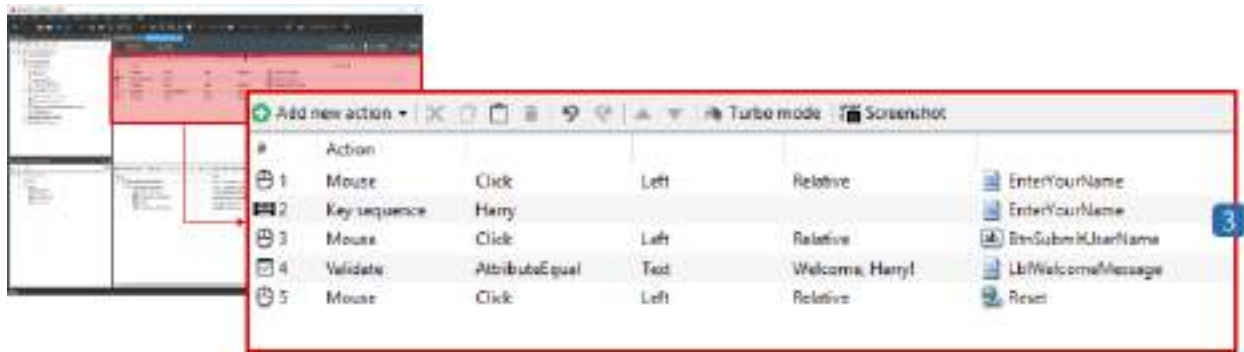
The module browser and its applications are described in Ranorex Studio fundamentals > [Test suite](#).

3

### Action table

- The steps you recorded appear as actions in the action table.
- They are listed in the order they were executed.
- In our example, the list contains five actions.





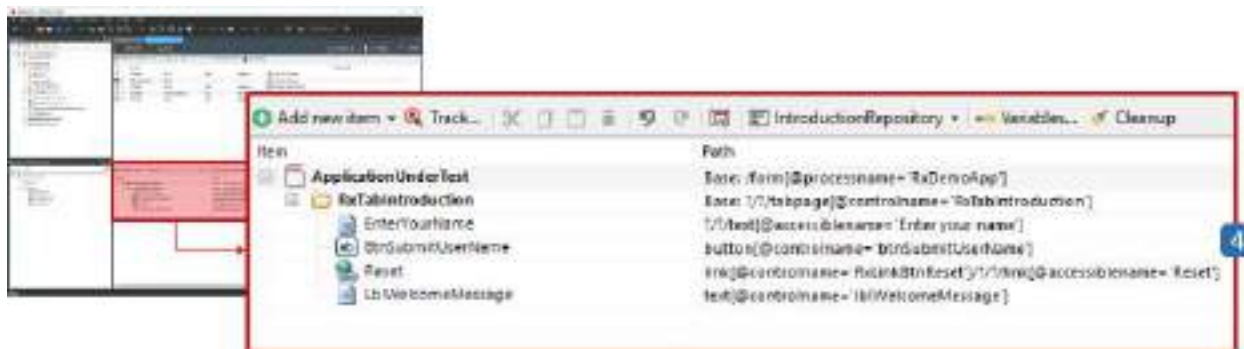
## Further reading

The concept of actions is explained in > Ranorex Studio fundamentals > [Actions](#).

## 4

## Repository

- Repository items are representations of UI elements.
- Each UI element affected by an action during recording is referenced by a corresponding repository item.
- In our example, there are four UI elements organized in two folders.



## Further reading

Repositories and their applications are described in Ranorex Studio fundamentals > [Repository](#).

## Action table analysis

Let's have a closer look at the five actions in the action table.

Add new action • ✂ 📄 🗑 ↺ 🔍 Turbo mode Screenshot						
#	Action					
1	Mouse	Click	Left	Relative	EnterYourName	1
2	Key sequence	Harry			EnterYourName	2
3	Mouse	Click	Left	Relative	BtnSubmitUserName	3
4	Validate	AttributeEqual	Text	Welcome, Harry!	LblWelcomeMessage	4
5	Mouse	Click	Left	Relative	Reset	5

### 1 Mouse click action

- This action represents the click into the name text field.
- The last column of this action contains a reference to the repository item that represents the UI element 'EnterYourName' (i.e. the name text field).

### 2 Key sequence action

- This action represents the text input into the name text field.
- In our example, **Harry** is the input, and the UI element 'EnterYourName' is the target.

### 3 Mouse click action

- Action #4 represents the click on the **Submit** button in the Demo App.
- The **Submit** button UI element is referenced in the last column by a corresponding repository item.

### 4 Validation action

- The test validation is also represented as an action.
- Ranorex matches the text in the changed welcome message against a reference text.

### 5 Mouse click action

- This action represents clicking on **Reset** to reset the welcome message.



## Further reading

The concept of actions is explored in Ranorex Studio fundamentals > [Actions](#).

## Repository analysis

Our example contains four repository items organized in two folders. Each repository item corresponds to a UI element in the application under test. Let's have a closer look at them.



1

### Item #1 – EnterYourName

This repository item represents the name text field of the Demo App.

2

### Item #2 – BtnSubmitUserName

This repository item represents the **Submit** button of the Demo App.

3

### Item #3 – Reset

This repository item represents the **Reset** link of the Demo App.

4

### Item #4 – LblWelcomeMessage

This repository item represents the text label that contains the welcome message of the Demo App.

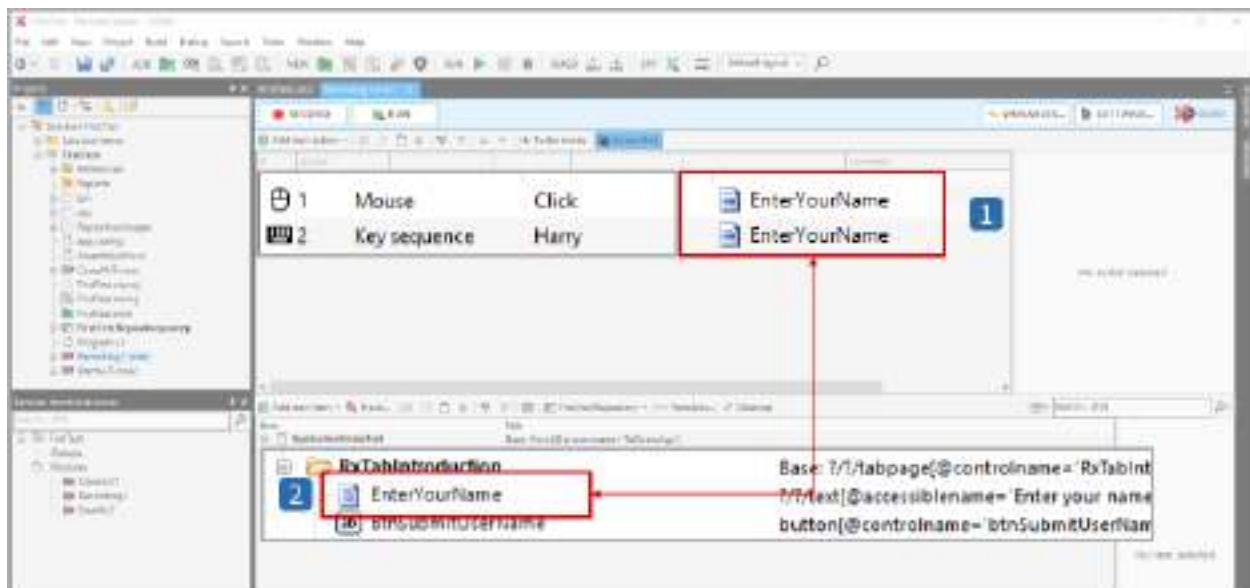


## Further reading

The concept of repositories is explained in Ranorex Studio fundamentals > [Repository](#).

## The link between actions and repository items

Recorded actions and repository items are managed and stored separately, but they are linked to each other.



### 1 Actions referencing UI element(s)

Both the mouse click into the name text field (action #1) and the **Harry** text input into the name text field (action #2) affect the same UI element. Therefore, they are both linked to the '**EnterYourName**' (item #1) repository item that represents this UI element.

### 2 Repository item representing a UI element

- UI elements are represented by repository items.
- Repository items have a name (e.g. EnterYourName) and a location in the GUI (=path).



### Screencast

To learn how to add UI elements to the repository manually, and add actions to a recording that aren't connected to repository items, watch our video → [Ranorex Studio Recorder basics 5: Add elements and actions manually](#)

## Download the sample solution

You can download the sample solution used for this chapter below.

### Installation:



1 Unzip to any folder on your computer.

2 Start Ranorex Studio and open the solution file `Introduction.rxsln`.



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must accept the automatic solution upgrade for versions 8.2 and higher.

## Run and debug recordings

In this chapter, you'll learn how to run and debug recordings.

For now, we'll focus on running a test recording in its simplest form: executing recorded actions one after another. In the larger context of test automation, running a test means executing iterations based on different sets of user data, parameters, and test scenarios. We'll deal with this in later chapters.

## Screencast

The screencast “Run and Debug a Test” walks you through the information found in this chapter.

[Watch the screencast now](#)



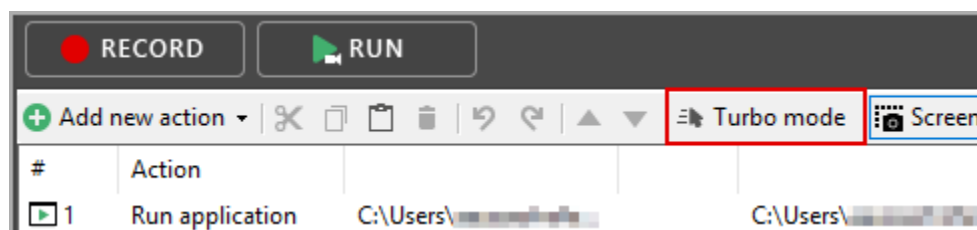
## Screencast

To see how to run a single recording, run a test case or test suite, execute individual steps in a recording, and enable Turbo mode or Debug mode, watch our video [→](#)  
[Ranorex Studio Recorder basics 4: Run and debug a test](#)

## Enable turbo mode

By default, Ranorex Studio uses a predefined delay between actions to control how fast they’re replayed. You can enable **Turbo mode** to speed up replaying by reducing this delay to almost zero.

- 1 In the recording module toolbar, **click Turbo mode** to turn turbo mode on or off.



### Note

- **Turbo mode** is applied to all actions in the active recording module.

## Run a recording module

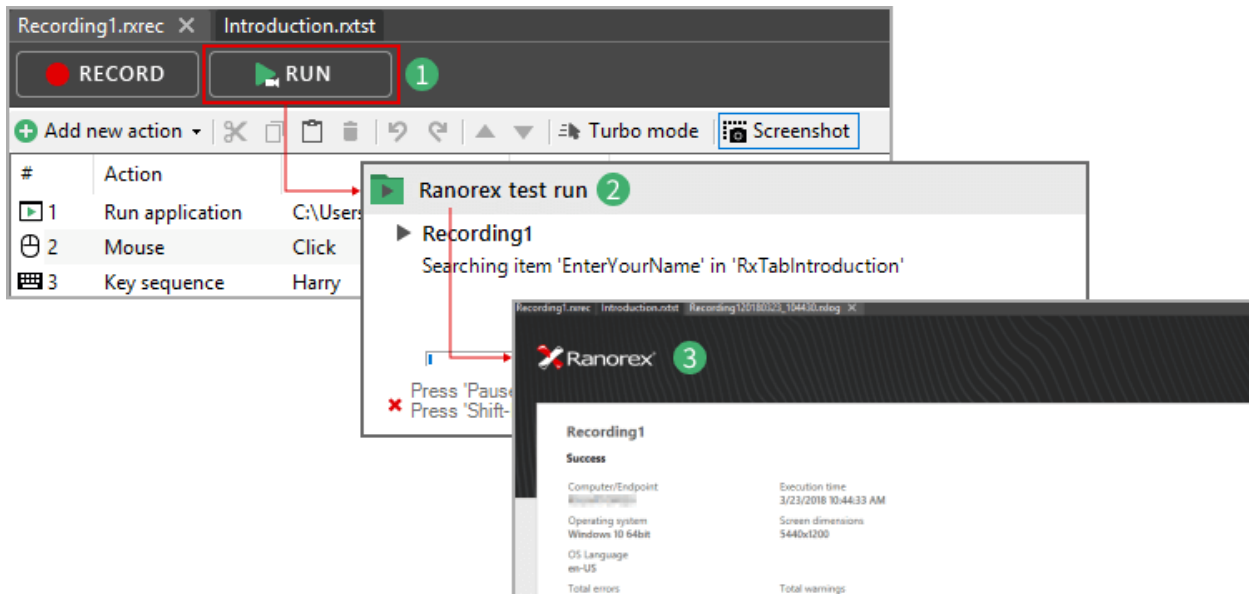
This section demonstrates how to run a single recording module.

- 1 In the Recorder view, **click RUN**.
- 2 Watch as the run starts and progresses.
- 3 When the run finishes, the test report appears.



### Hint

Remember: **Don't touch your mouse or keyboard** during the test run, or you might cause a test failure.



The screenshot illustrates the process of running a recording module in the Ranorex Recorder. It shows three sequential steps:

- Step 1:** The Recorder view is shown with the **RUN** button highlighted by a red box and a green circle with the number 1. The toolbar also shows the **RECORD** button and the **Turbo mode** toggle.
- Step 2:** A red arrow points from the **RUN** button to the **Ranorex test run** button, which is also highlighted with a green circle with the number 2. Below this, a list of actions is visible: 

#	Action	
1	Run application	C:\Users\...
2	Mouse	Click
3	Key sequence	Harry
- Step 3:** A red arrow points from the **Ranorex test run** button to the **Recording1** test run window, which is highlighted with a green circle with the number 3. The window shows the test progress: **Recording1** Searching item 'EnterYourName' in 'RxTabIntroduction'.

The final window shows the test results for **Recording1**, indicating **Success**. The results include:

Computer/Endpoint	Execution time
...	3/23/2018 10:44:33 AM
Operating system	Screen dimensions
Windows 10 64bit	5440x1200
OS Language	Total errors
en-US	
Total warnings	



## Further reading

Test reports, their structure, content, and interpretation are described in Ranorex Studio fundamentals > Reporting > [→ Introduction](#).

## Run the test suite

Running individual recording modules is usually only done when editing or troubleshooting them. Normally, tests are run based on **a test suite that contains test cases**. This is usually done from the **test suite view**. The test suite view is the **central point of control** for modularization and test case management.



## Further reading

The test suite and its uses are explained in greater detail in Ranorex Studio fundamentals > Test suite > [→ Introduction](#).

## Switch to the test suite view

There are two ways to switch to the test suite view.

- a In the Studio toolbar, **click** the **View test suite** button.
- b Alternatively, **click** the tab **Introduction.rxtst** (**rxtst** = **R**anorex **x**test suite **t**est file).



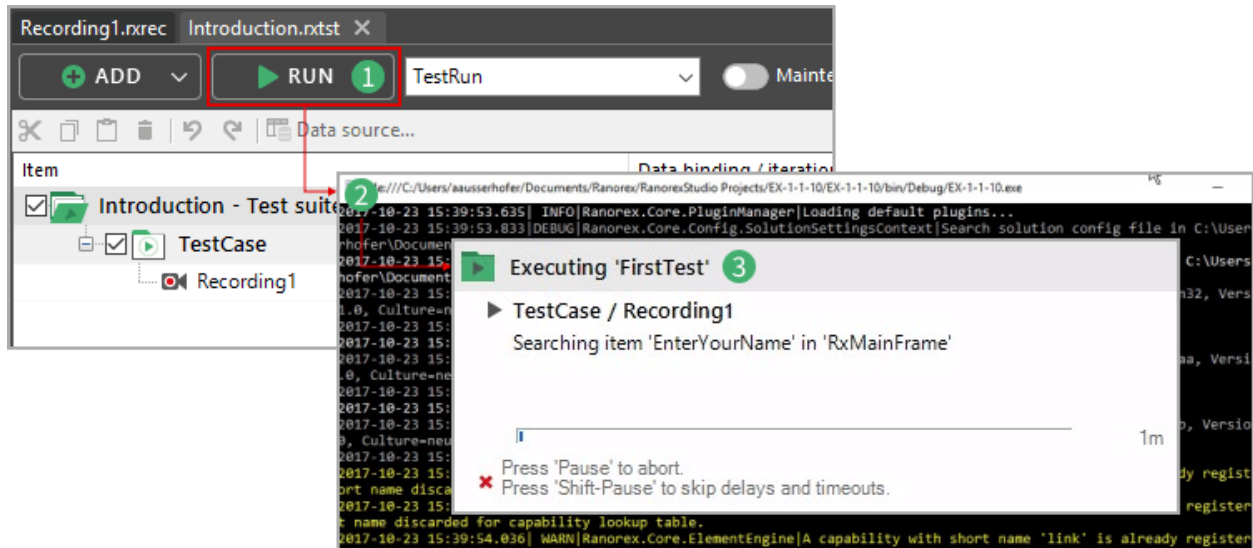
### Hint

- The example test suite has the same name as the test project: **Introduction**.
- The current test suite contains one test case with the default name **TestCase**.
- The test case contains a setup and a teardown region.
- The test case contains a recording module with the default name **Recording1**.
- The ticked checkbox means the test case will be executed during the test run.

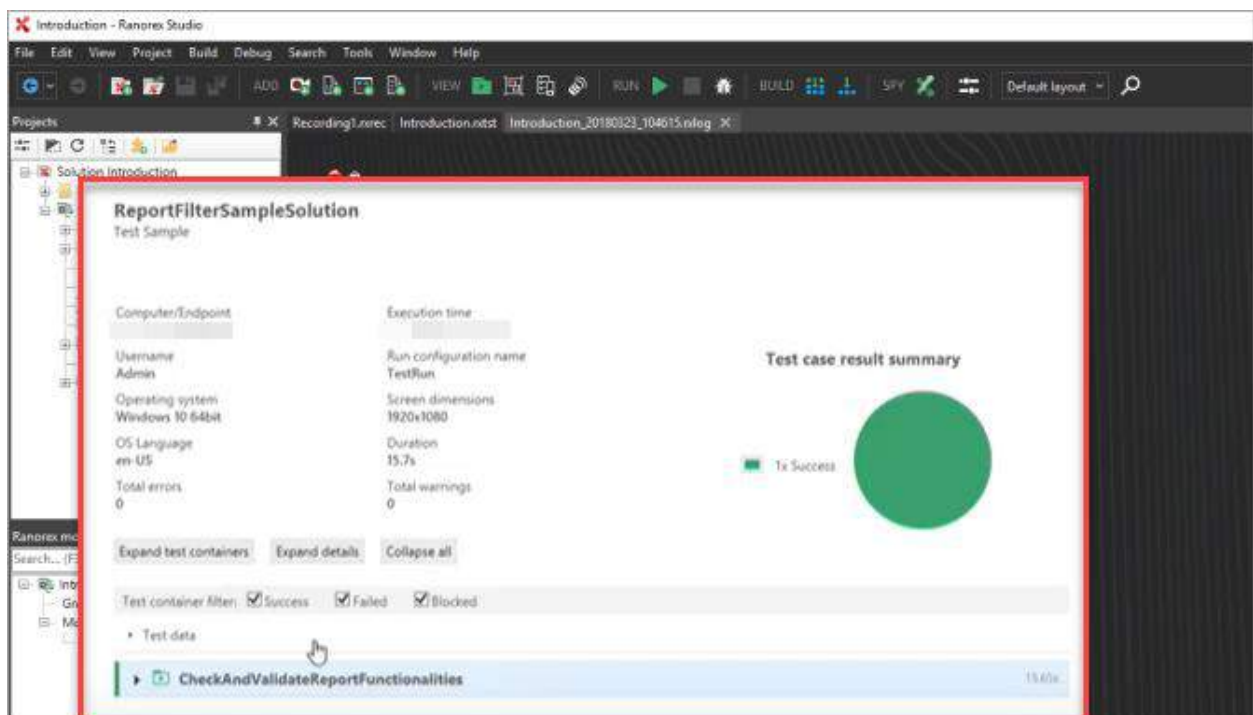


## Execute the test case

- 1 In the test suite view, **click RUN**. This runs the entire test suite. In our example, the test suite contains one test case, so only this test case executes.
- 2 **Watch** as the run starts and progresses.



- 3 When the run finishes, the test report appears.





## Further reading

Reports, their structure, content, and interpretation are explained in greater detail in Ranorex Studio fundamentals > Reporting > [Introduction](#).



## Screencast

The screencast “Progress dialog” introduces you to the real-time information that the progress dialog offers:

[Watch the screencast now](#)

## Pause/resume a test run

You can pause a test run while it is executing and resume it later. This is useful e.g. if you notice your test environment is not set up correctly and want to correct this so your test doesn't fail for a reason that has nothing to do with the application itself.



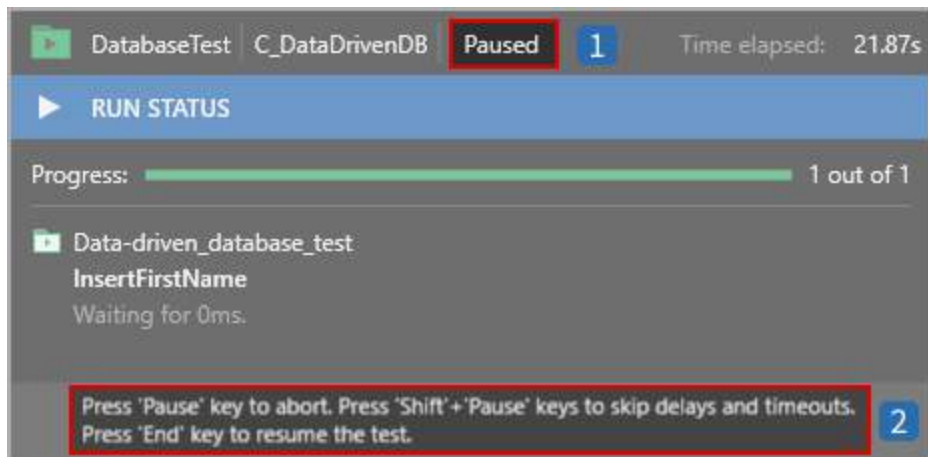
## Note

- If you have activated video reporting, pausing a test run **does not** pause video recording.
- If you pause a test run and make changes to the test in Ranorex Studio, these changes **will not** affect the test run.
- If you pause a test run and make changes to the application/system under test, these will of course affect the test run after resuming.

To pause or resume a test run:



1 While a test is running/paused, **press End**.



- 1 Label in the progress dialog indicating the test run is paused.
- 2 Info section showing the test run hotkeys. **End** pauses/resumes.

## Global run buttons

You don't have to be in the test suite view to run or stop a test suite. You can also click the global run and stop buttons in the Studio toolbar. They are always visible, regardless of your current view.



## Run options

Sometimes, you don't want to run an entire recording, or you might want to control the run similar to debugging a computer program, as described below.

- 1 **Change** to the Recorder view.
- 2 **Right-click** an action and select one of the options explained below.

<div> <div>+ Add new action</div> <div> <div>✂</div> <div>📄</div> <div>🗑</div> <div>↶</div> <div>↷</div> <div>⬆</div> <div>⬇</div> <div>⚡ Turbo mode</div> <div>📷 Screenshot</div> </div> </div>						
#	Action		Button	Action spot	Repository item	Comment
1	Run application	C:\Users\...		C:\Users\...\Docu...		
2	Mouse	Click	Left	Relative	EnterYourName	
3	Key sequence	Harry			EnterYourName	
4	Mouse				BtnSubmitUserName	
5	Validate				LblWelcomeMessage	
6	Mouse				Reset	
7	Mouse				RxButtonExit	

▶

▶

▶

●

Run selected item(s)

Run to here (excl. selection)

Run from here

Record from here (after selection)

1

2

3

4

## 1 Run selected item(s)

Runs only the selected action(s).

## 2 Run to here (excl. selection)

Runs the recording starting at action #1 and up to, but excluding, the selected action.

## 3 Run from here

Runs the recording starting at the selected action.

## 4 Record from here (after selection)

Starts a recording and adds the recorded actions between the selected action and the next.

## Debug – enable/disable Continue on fail

When an action causes an exception during a test, the default behavior is that the test aborts. Sometimes, you might want the run to continue instead. This is where the option **Continue on fail** is useful.



### Further reading

The debugging option **Continue on fail** is part of the concept of actions and is explained in detail in Ranorex Studio fundamentals > Actions > → [Managing actions](#).

## Debug – insert a log message

When tests become complex, it's often helpful to add special log messages to the report.



### Further reading

Log messages and other options for controlling the test report are described in detail in Ranorex Studio fundamentals > Reporting > → [Basic report characteristics & data](#).

## Enable/disable debug mode

Ranorex Studio includes a dedicated debug mode that works by setting breakpoints in programming code. This is an expert topic, which is why we only touch on it here briefly.



1

Turning debug mode on/off

2

Debug mode requires Ranorex Studio to be run as an administrator. If this isn't the case, a prompt to restart in administrator mode appears.



### Hint

Debug mode decreases Ranorex Studio performance by around 30%.



### Further reading

Debugging is an expert topic and is explained in Ranorex Studio expert > Ranorex Studio IDE > [Debugging](#).

## Download the sample solution

Download our sample solution to try the steps and options explained in this chapter yourself.



### Sample solution

**Theme:** Run and debug recordings

**Time:** Less than 10min

[Download sample file](#)

### Installation:

- 1 Unzip to any folder on your computer
- 2 Start Ranorex Studio and open the solution file `Introduction.rxsln`



### Hint

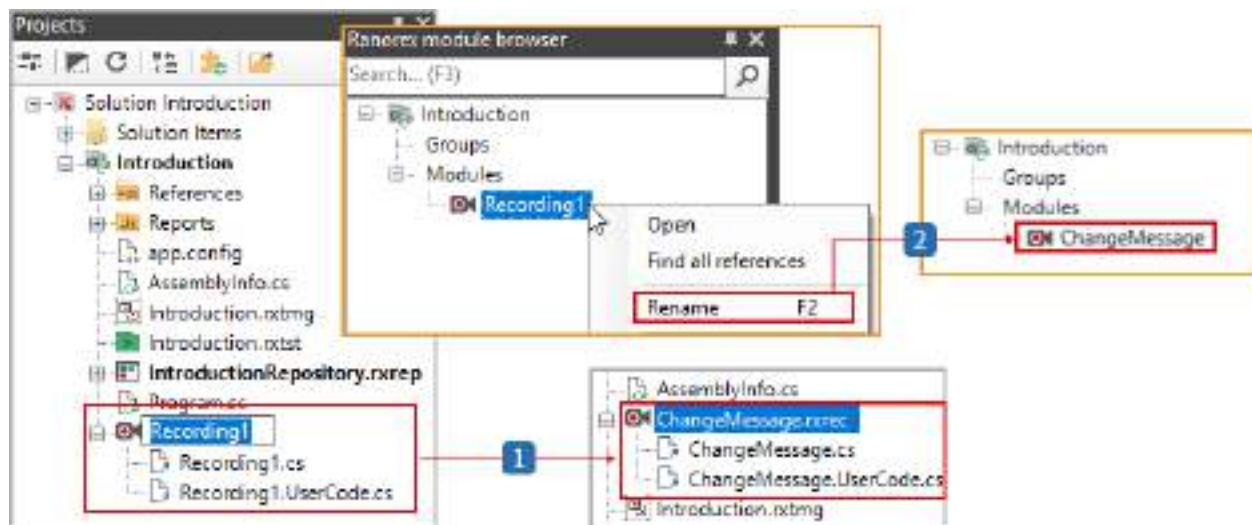
The sample solution is available for Ranorex Studio versions 8.0 or higher. You must accept the automatic solution upgrade for versions 8.2 and higher.

## Manage recording modules

In this chapter, you'll learn about **managing and structuring recordings** in Ranorex Studio.

### Rename a recording module

The default name of a recording module is always **RecordingX.rxrec**, where X is the number of the recording module. You can change the name of a recording module in several places.



#### 1 Renaming in project file view

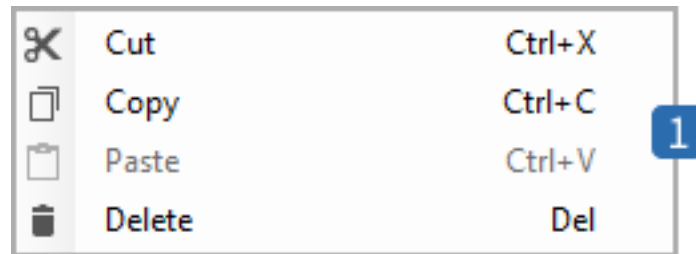
- Use the context menu or press F2.
- The names of all associated files will change automatically.

#### 2 Renaming in module browser view.

- Use the context menu or press F2.

## Cut/copy/paste/delete a module

Cut, copy, paste, and delete recording modules in the projects view using the standard Windows keyboard shortcuts or the context menu.



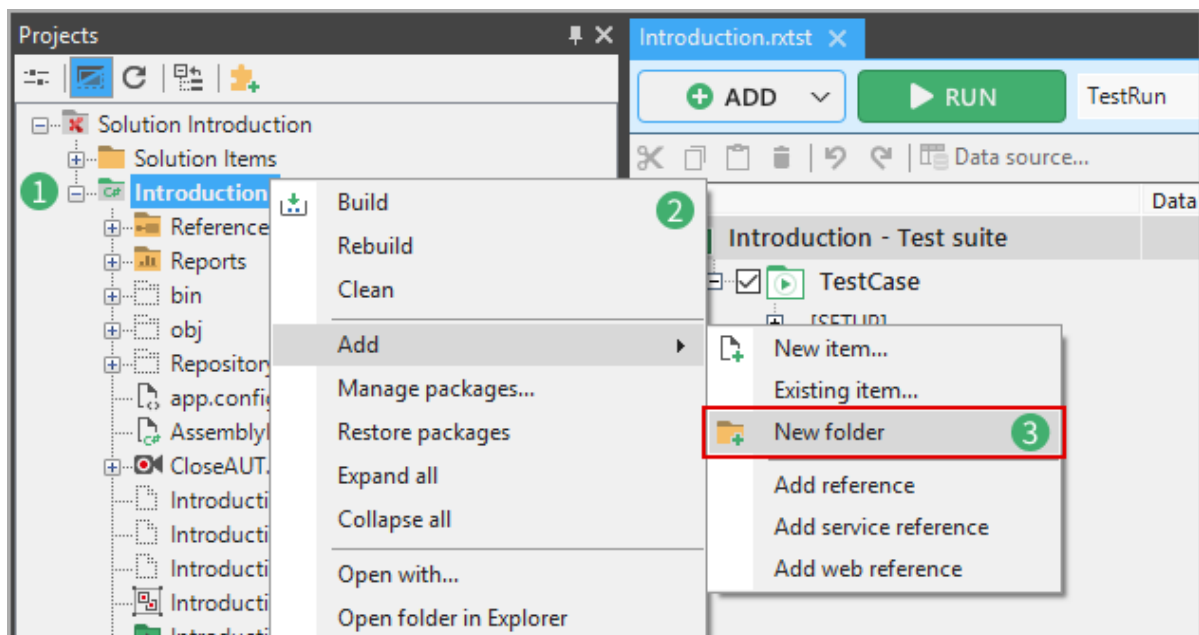
1 Cut/copy/paste/delete in context menu

### Note

You can't cut/copy/paste/delete in the module browser.

## Organize recording modules in folders

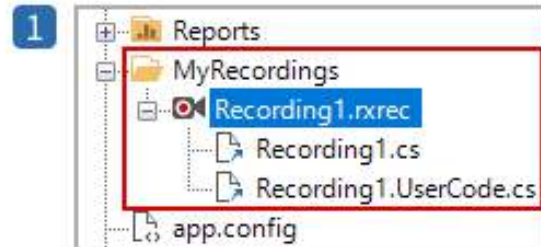
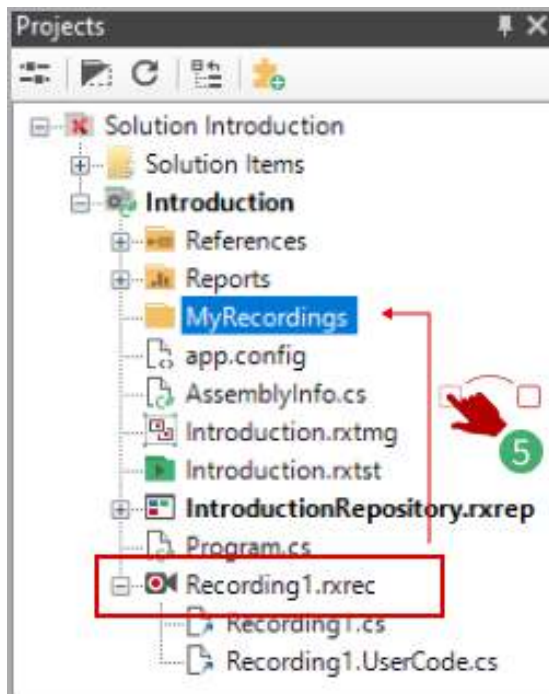
You can organize your recording modules in folders in the projects view.



- 1 In the projects view, **select** the item where you want to create the new folder.
- 2 **Open** the context menu.
- 3 **Click Add > New folder.**



- 4 Give the folder a meaningful name (e.g. MyRecordings).



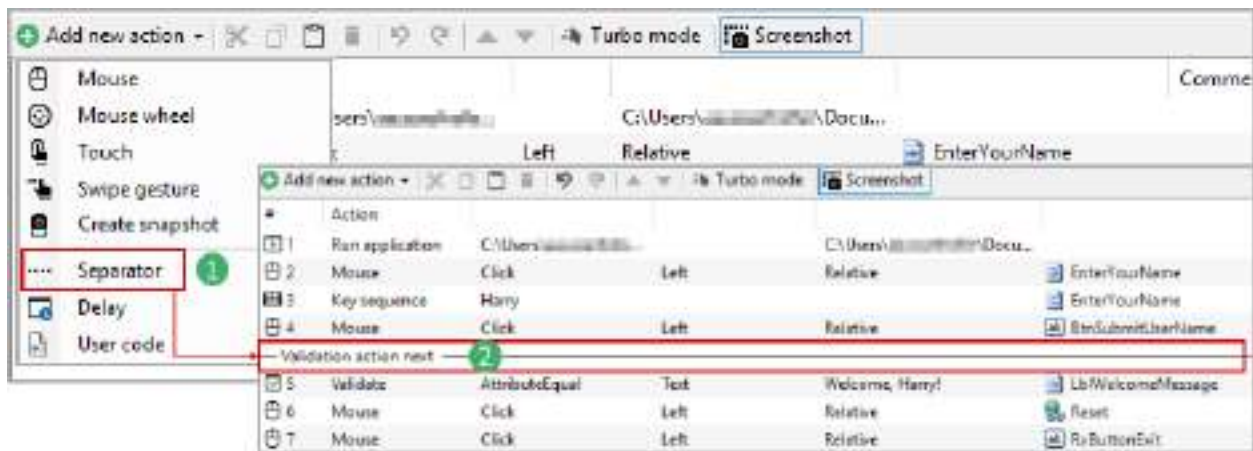
- 5 Drag recording modules to the new folder.

**Result(s):**

- 1 Recording module in new folder in **projects view**
- 2 Recording module in new folder in **module browser**

### Structure actions within recording modules

You can add separators to visually structure actions in the action table. This is purely cosmetic. It has no effect on the actions or their execution.



- 1 Click Add new action > Separator.
- 2 The **separator** is added after the selected action.

### Note

The separator will also appear as a log message in the report.

00:05.385	Info	Mouse	Mouse Left Click item 'RxFMainFrame.RxTabIntroduction.BtnSubmitUserName' at 40;10.
00:06.230	Info	Section	Validation action next
00:06.277	Info	Validation	Validating AttributeEqual (Text='Welcome, Harry!') on item 'RxFMainFrame.RxTabIntroduction.LbWelcomeMessage'.
00:06.437	Success	Validation	Attribute 'Text' of element for item 'IntroductionRepository.RxFMainFrame.RxTabIntroduction.LbWelcomeMessage' match the specified value.

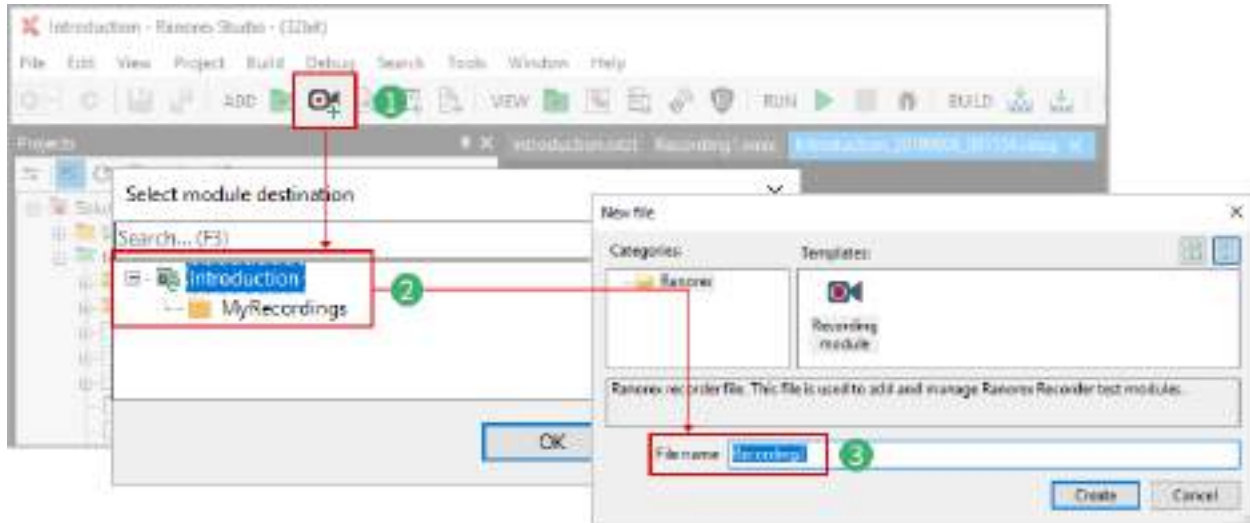


### Screencast

You can edit a recording to make it more maintainable and reusable. See how to combine key sequences, replace a key sequence with an Add Entry function, create a new module by selecting and moving recorded actions, add modules to a test case, delete unnecessary actions, and clean up the repository in our video → [Ranorex Studio Recorder basics 3: Edit a recording](#)

## Add a new recording module

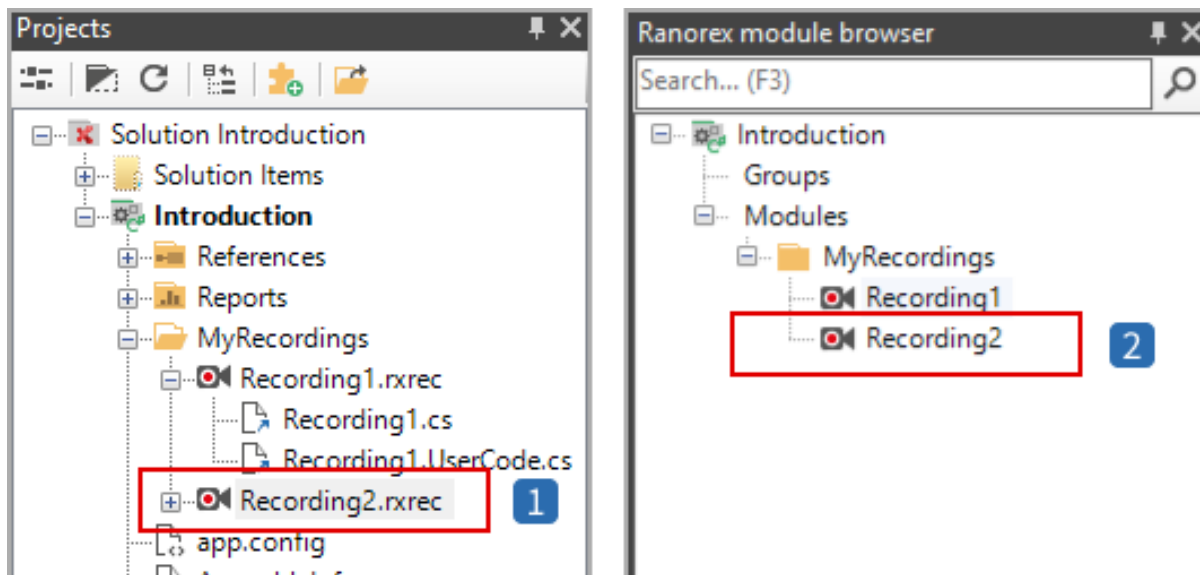
Strive to keep your recordings as small as possible. This is why, sooner or later, you'll need additional recording modules to keep your tests well structured.



- 1 In the Studio toolbar, **click** the **Add recording module** button.
- 2 **Select** the folder you want to save the module to and **click OK**.
- 3 **Give** the module a meaningful name; then **click Create**.

### Result(s):

- The recording module appears in the projects view and the module browser.

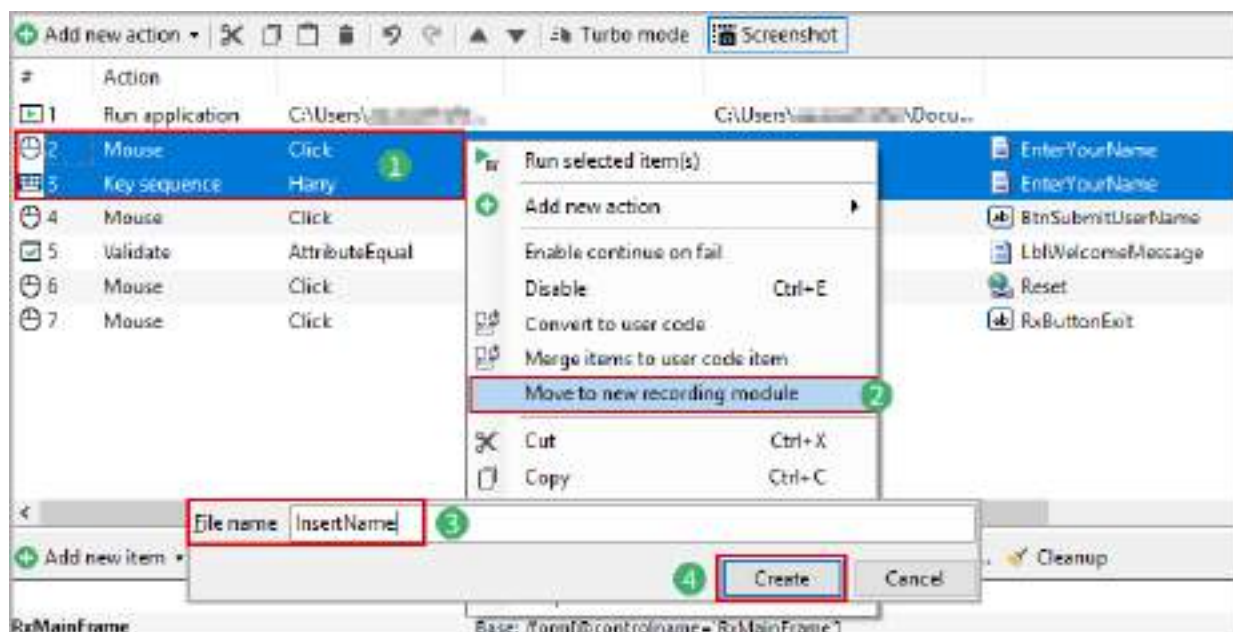


- 1 New recording module in **projects view**
- 2 New recording module in **module browser**

## Create a new recording from existing actions

Sometimes, a recorded test may contain **too many actions**. This makes the recording **hard to reuse**. Ranorex Studio allows you to **split** large recordings into modular, reusable ones, using the **Move to new recording module** function.

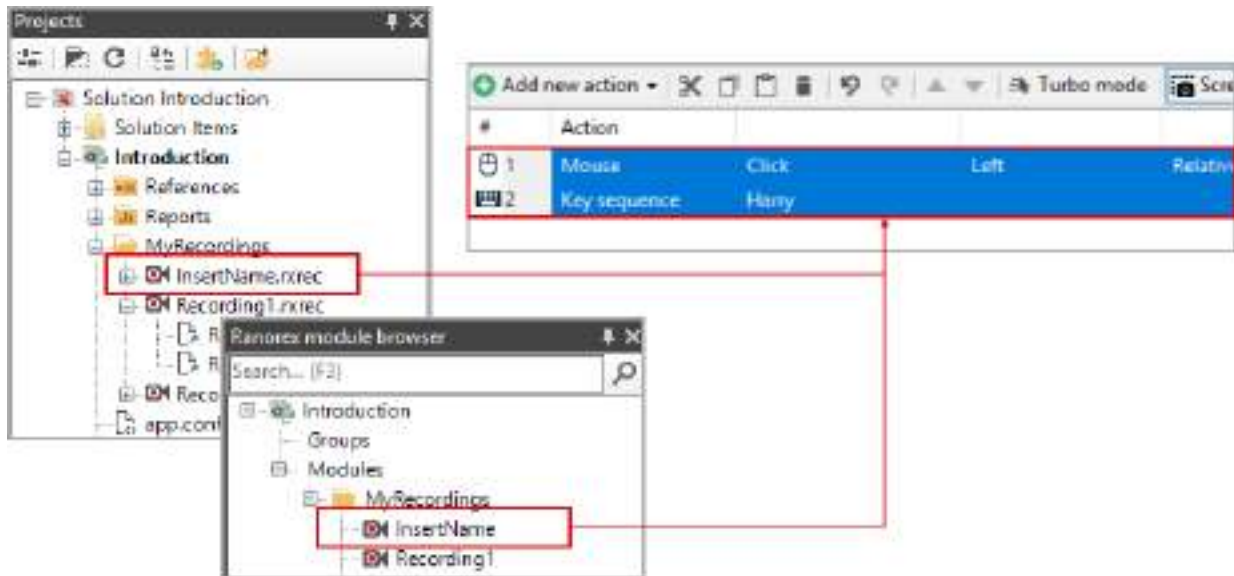
- 1 **Select** the actions you want to move to a new recording and **right-click** to open the context menu.



- 2 **Click Move to new recording module.** If you have custom folders, also **select** a destination folder for the new module.
- 3 **Give** the new recording module a meaningful name.
- 4 **Click Create.**

### Result(s):

The new recording module is created with the selected actions. The new module appears in the projects view and the module browser.

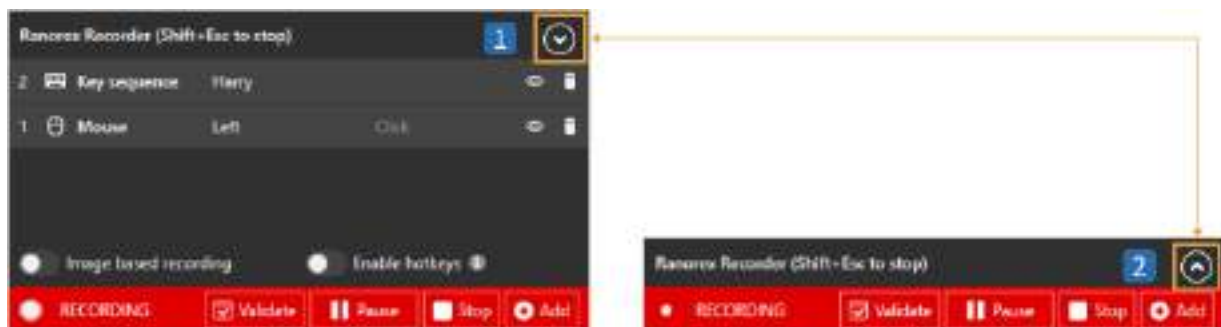


## Recorder control center & hotkeys

In this chapter, you'll learn how to use the Recorder control center and hotkeys effectively.

### Full/minimized view

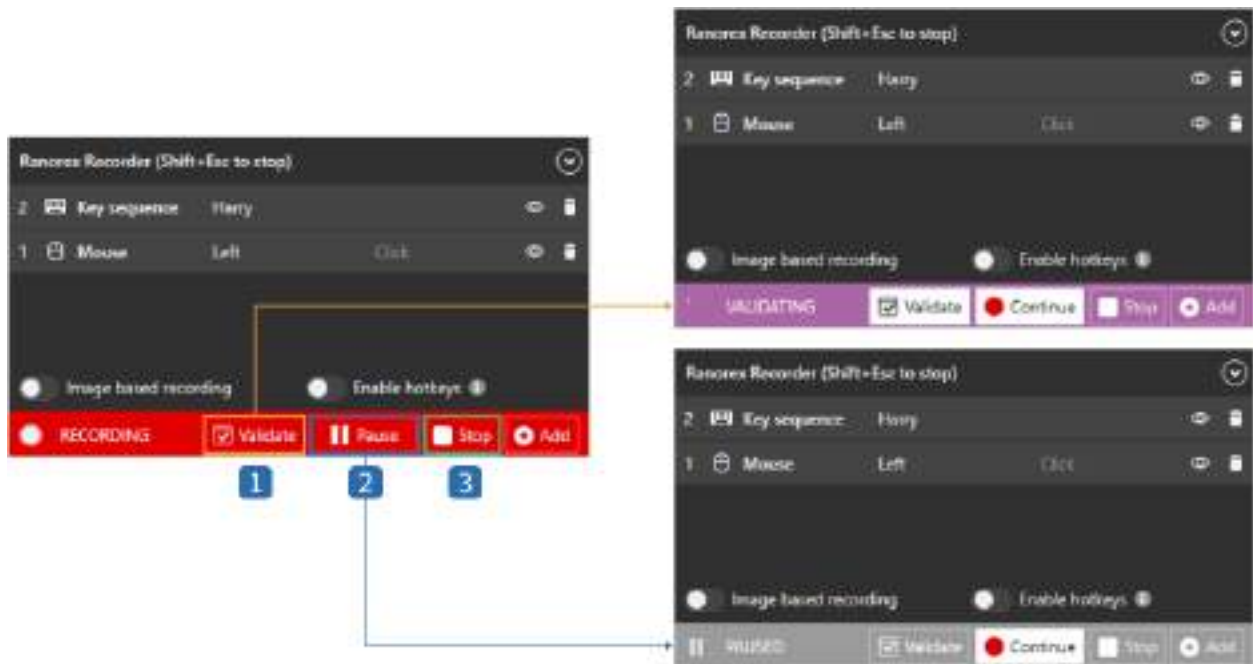
The Recorder control center is available in a full and a minimized view. Switch between them by clicking the button in the top right of the window.



- 1 Full view displaying all controls and an actions table that shows the last four recorded actions.
- 2 Minimized view that only shows the controls in the bottom ribbon.

## Standard functions

The Recorder control center has three button-activated controls.



### 1 Validation mode

- Inserts a test validation action into the recording.

### 2 Pause/continue

- Pauses/continues the recording.
- No user interaction is recorded while paused.

### 3 Stop

- Stops the recording.
- Returns to Ranorex Studio.

## Add/delete actions during recording

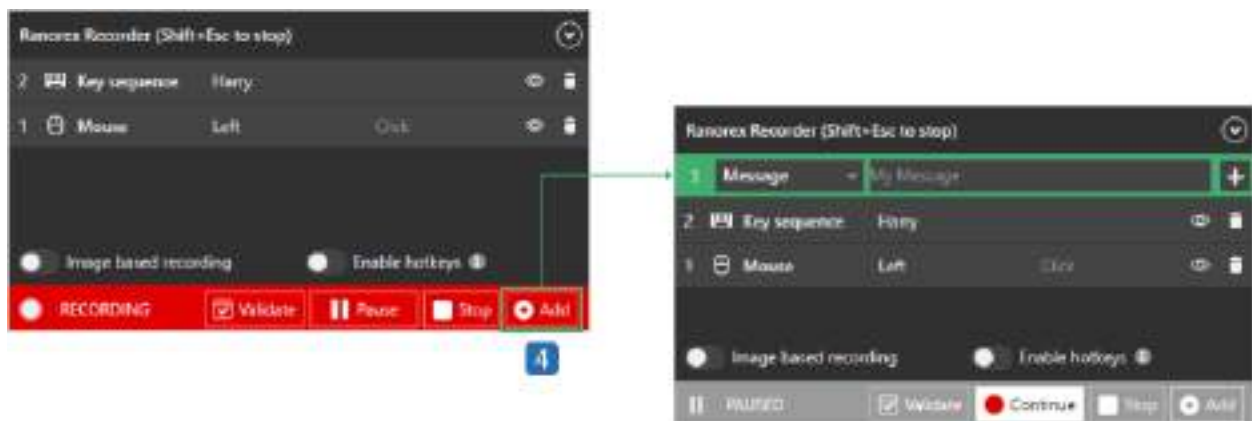
In the full view, the control center displays the last four recorded actions. From there, you can delete them or view their respective screenshots.





- 1 List displaying the last four recorded actions.
- 2 Mouse over the eye to display a screenshot of the action.
- 3 Click the trash can to delete an action.

You can also add one of three action types by clicking **Add** in the bottom right of the control center.



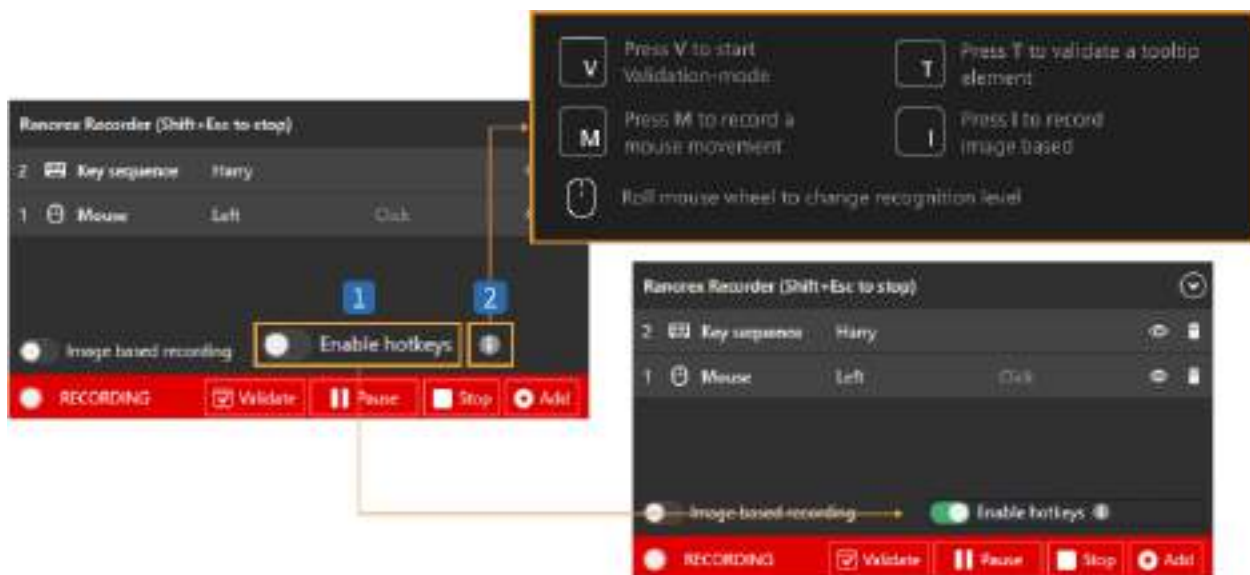
- 4 Click **Add** to directly add one of three action types. Recording pauses while you do so.



- 5 The available actions are Message, Screenshot, and Delay. For more information on what these do, refer to → [Action properties](#).
- 6 This is where you configure the selected action. Changes depending on the action.
- 7 Click this button to add the action and resume recording.

## Enable/disable hotkeys

Turn hotkeys on and off by setting the corresponding switch to on or off.



- 1 Switch to turn hotkeys on and off.
- 2 Mouse over this info element to display the available hotkeys.

### **Note**

While hotkeys are enabled, you won't be able to use the hotkey characters ( **v**, **m**, **t**, **i** ) in your recording, e.g. to enter a text.

The uppercase characters **V**, **M**, **T**, **I** can be used normally



## Validation mode hotkey

Pressing **V** on the keyboard activates validation mode in the same way clicking the **Validate** button would.

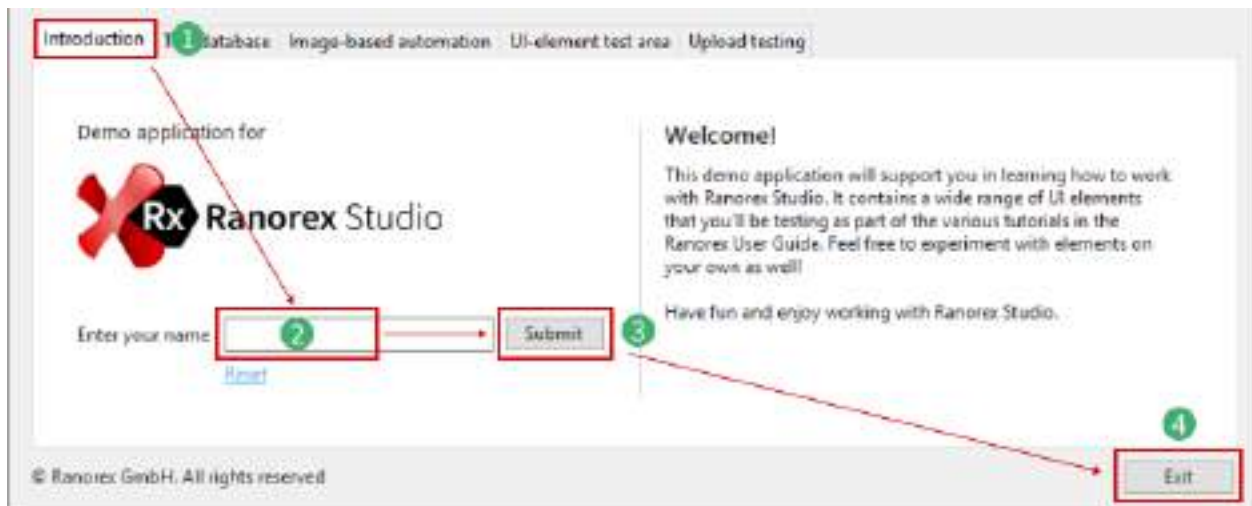


### Further reading

The concept of test validation is described in Ranorex Studio fundamentals >  
→ [Test validation](#).

## Mouse movement recording hotkey

Usually, mouse movements **are not recorded**. If you need to record mouse movements, you can do so by pressing **M**. Follow the instructions below to practice recording mouse movements.



- 1 **Move** the mouse over the **Introduction** tab and press **M**
- 2 **Move** the mouse over the name text field and press **M**
- 3 **Move** the mouse over the **Submit** button and press **M**
- 4 **Move** the mouse over the **Exit** button and press **M**

## Result(s):

Once you've stopped recording, the action table will list four mouse movement actions. Each movement is linked to a UI element.

+ Add new action ▾   ✂   📄   🗑   ↺   ↻   ▲ ▼   ⚡ Turbo mode   📷 Screenshot					
#	Action				
⌚ 1	Mouse	Move	Relative		📄 Introduction
⌚ 2	Mouse	Move	Relative		📄 EnterYourName
⌚ 3	Mouse	Move	Relative		📄 BtnSubmitUserName
⌚ 4	Mouse	Move	Relative		📄 RxButtonExit

## Tooltip validation hotkey

Press **T** to start a tooltip validation.



### Further reading

Tooltip validation is explained in Ranorex Studio fundamentals > Test validation > [Validation of tool-tips](#).

## Image-based recording hotkey

Press **I** to activate image-based recording.



### Further reading

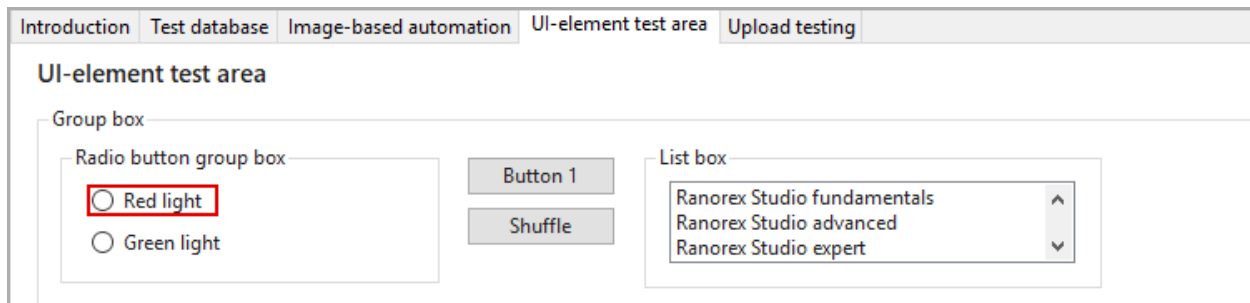
Image-based automation is an advanced topic and is explained in Ranorex Studio advanced > [Image-based automation](#).

## Change recognition level

When hotkeys are activated, the **mouse wheel** controls the recognition level of the UI element detection. Let's take a look at this with a simple example.

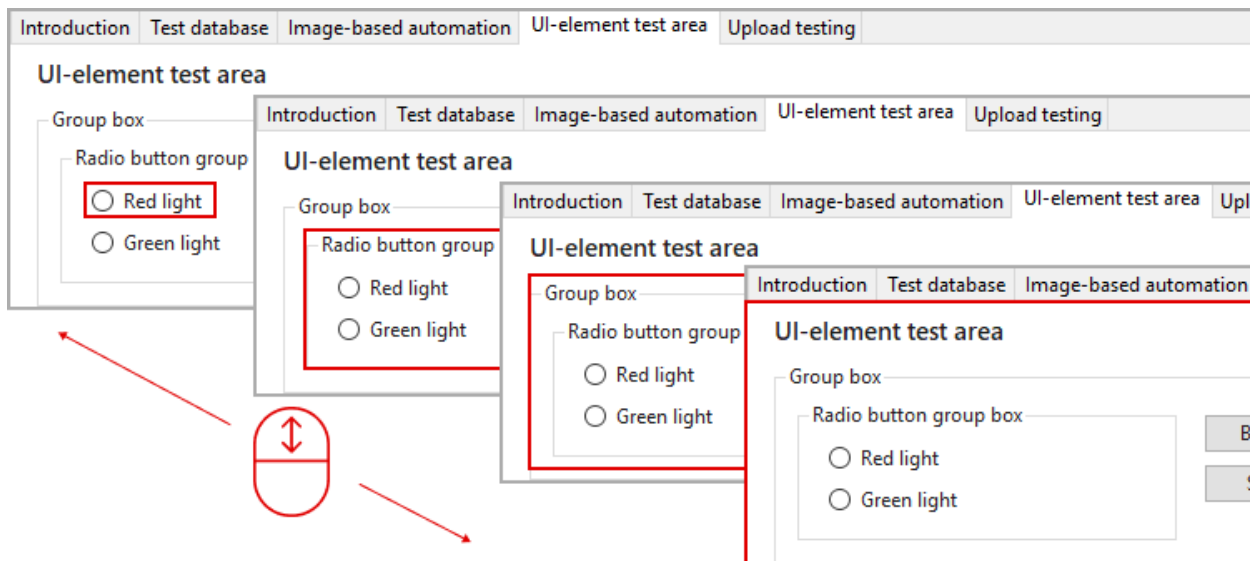
### Initial recognition level:

- Ranorex Recorder identifies every UI element during the recording.
- This UI element identification is made visible by a **red frame** that follows your mouse movement during the recording (see image).



### Change the recognition level:

**Scroll the mouse wheel** to move through the UI element levels.



# Test suite

The test suite is where you **build, organize, and run your tests** in Ranorex Studio. A test suite consists primarily of **test cases**, which are in turn built from **modules**. You can add **smart folders** to **structure** your test. The test suite is also where you **configure variables and data binding** for [data-driven testing](#). You can run test suites in Ranorex Studio or the stand-alone Ranorex Test Suite Runner.



## Screencast

The screencast “introduction to the Ranorex Studio test suite” walks you through information found in this chapter.

[Watch the screencast now](#)

## Download the sample solution

The explanations in the test suite chapter are based on a sample solution. You can download it below.



## Sample solution

**Theme:** Test Suite

**Time:** Less than 10min

[Download sample solution](#)

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `RxDatabase.rxsln`



## Hint

The sample solution is available for Ranorex Studio versions 8.0 or higher. You must agree the automatic solution upgrade for versions 8.2 and higher.

## The test suite file

Each test suite is contained in a **special file** with the extension **.rxtst** (for Ranorex test suite). Test suite files are always part of a project, and you can find them **in the projects view**. A project may contain **multiple test suite files**. On your **hard drive**, test suite files are **stored** in the **corresponding project's folder**, e.g.:

```
%USERPROFILE%\RanorexRanorexStudio
```

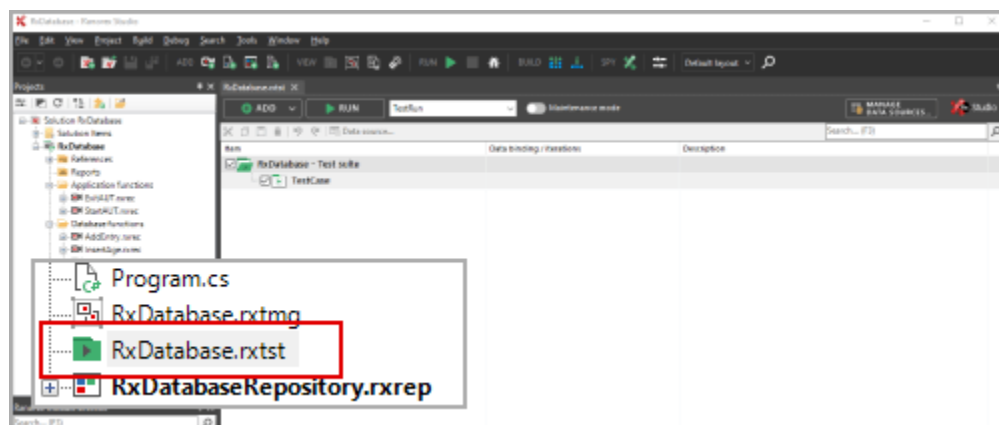
```
Projects\RxDatabase\RxDatabase.rxtst
```

Test suite files are **XML files** and can be opened in any XML viewer or editor.



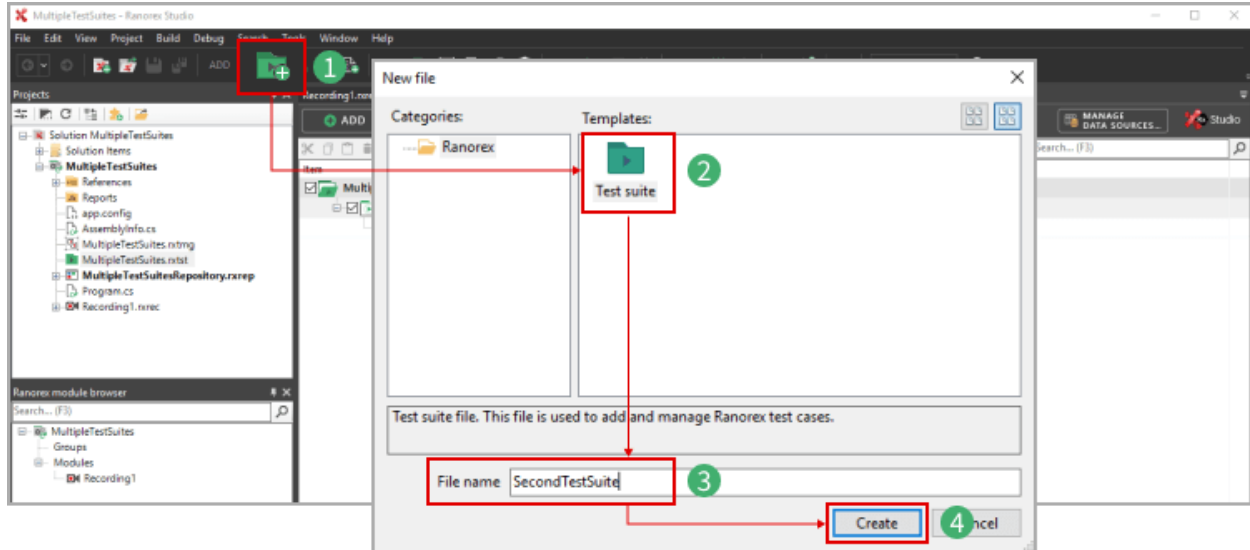
## Note

In the User Guide, we'll use the term "test suite" for both test suite itself and the **.rxtst** file. We'll point it out when we only mean one or the other.



## Add a test suite

Some projects don't contain a test suite by default, or you may want to add multiple test suites to your project. Follow the instructions below to do so.



- 1 In the Studio toolbar, **click** the **Add test suite** button.
- 2 If your solution contains multiple projects, **select** the desired project and **click OK**.
- 3 In the next dialog, **give** the test suite a name and
- 4 **Click Create**.

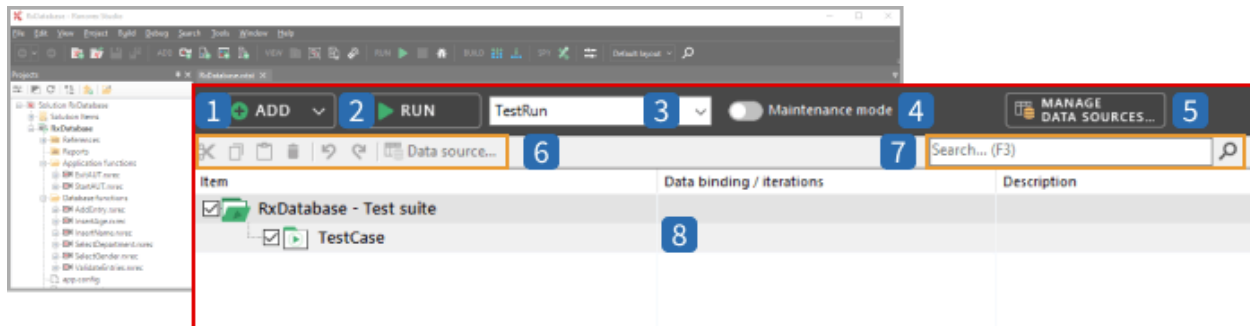


## Reference

Multiple test suites are explained in: [Ranorex Studio fundamentals > Test suite > Manage multiple test suites](#).

## The test suite view

To open the test suite, **double click** the test suite file. The **test suite view** appears. This is where you work on your test suite.



- 1 The **ADD** button. Click this to display a drop-down list of items to add to the test suite, such as a test case or smart folder. You can also add items by right-clicking within the test suite hierarchy. Items are greyed out in the drop-down list if they cannot be added at the current level of the test suite hierarchy.
- 2 The **RUN** button. Click this button to run the test suite in the selected run configuration.
- 3 The **run configuration** selector. This is where you can select and manage run configurations (see below).
- 4 The **maintenance mode** switch. Enables or disables the maintenance mode (see below).
- 5 The **MANAGE DATA SOURCES...** button. Add and manage test data sources. Ranorex Studio supports using a simple data table, as well as connectors to CSV, Excel, or SQL data files.
- 6 The **test suite toolbar**, which includes:
  - The **cut/copy/paste/delete** and **undo/redo** buttons.
  - The **Data source...** button brings up the data source dialog for a test case or smart folder.
- 7 The **search** box. Use the search box to locate items in the test suite.
- 8 The test suite **workspace**, which includes:
  - The **Item** column. Displays the test suite and the items it contains. Build your test here.
  - The **Data binding/ iterations** column. Displays data bindings and iterations that apply to the respective item (see below).
  - The **Description** column. Enter an optional description for an item here, such as what aspect of the AUT a test case covers.



## Reference

---

Running test suites, run configurations and iterations are explained in Ranorex Studio fundamentals > Test suite > → [Execute a test suite](#).



## Reference

---

**Data binding** and **data sources** are advanced topics explained in Ranorex Studio advanced > → [Data-driven testing](#).



## Reference

---

**Maintenance mode** is an advanced topic that's explained in Ranorex Studio advanced > → [Maintenance mode](#).

## The test suite structure

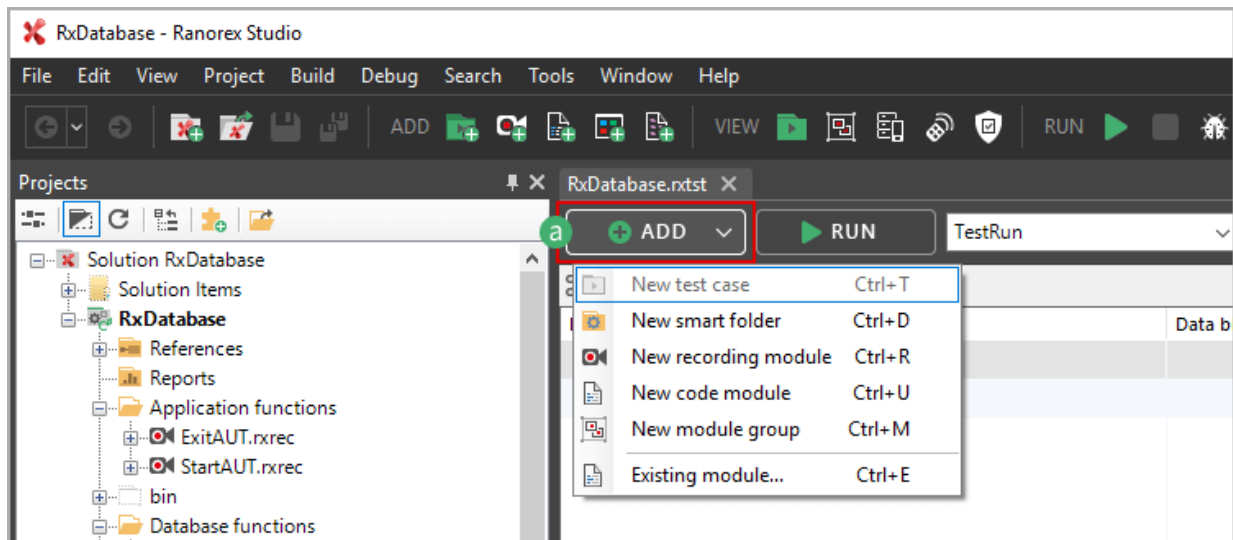
In this chapter, you'll learn about the items that make up a test suite, how to add them, their **purpose** and **functions**, and the **hierarchy** they follow.

### Add test suite items

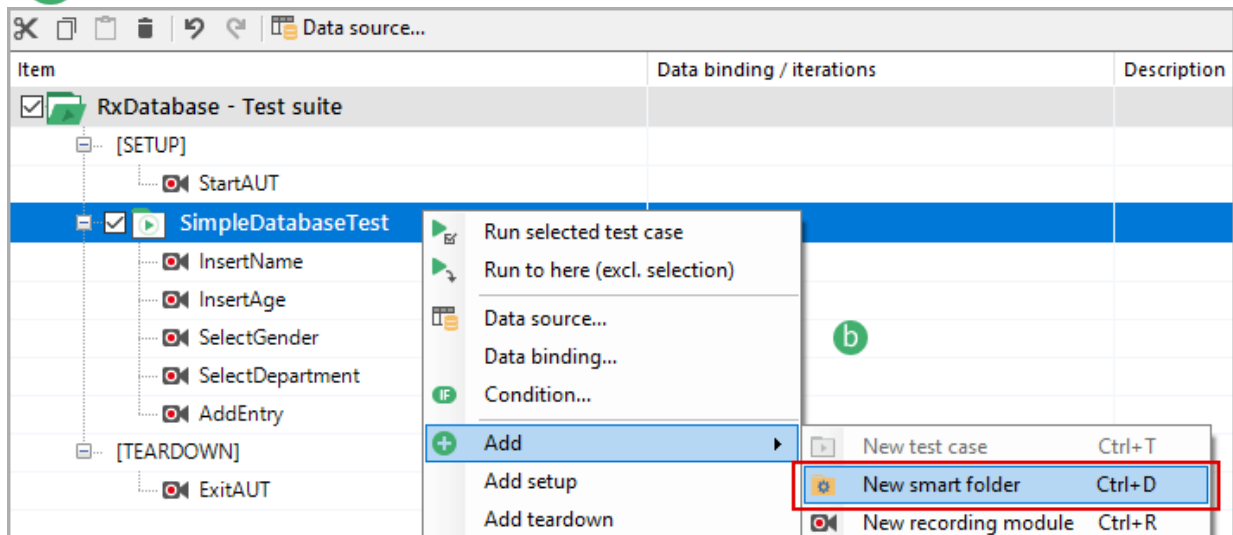
There are three ways to add items to a test suite.



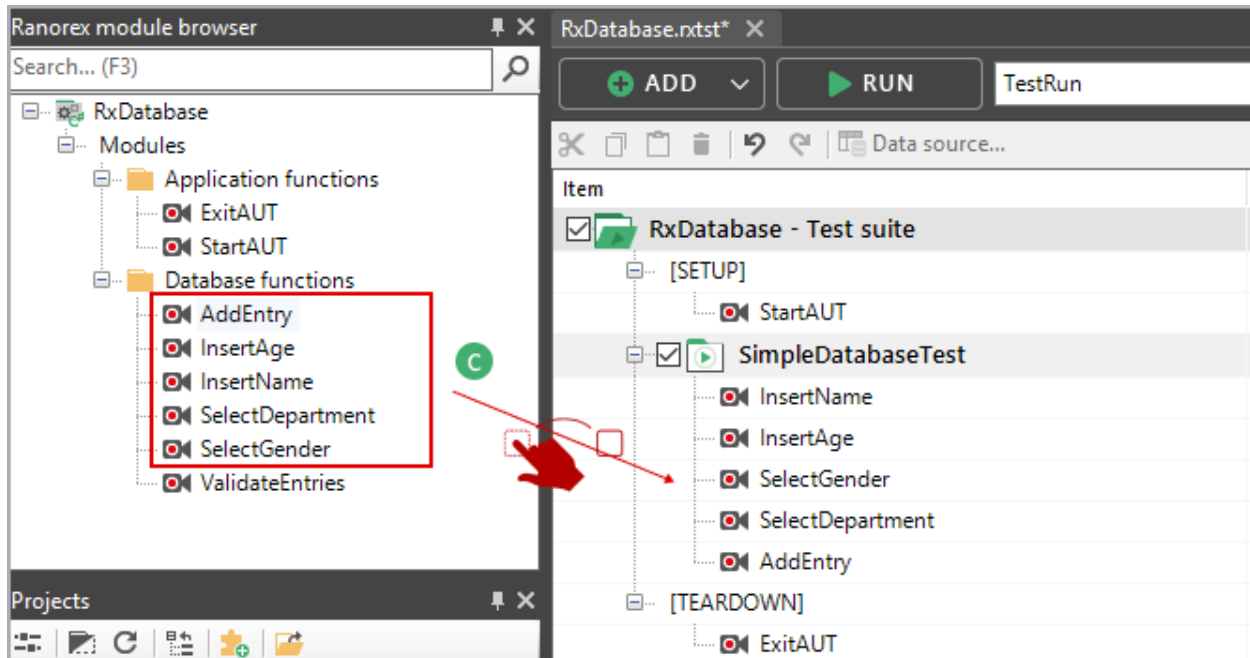
**a** Use the **ADD** button.



**b** Use the context menu in the test suite workspace.



**c** **Drag and drop** from the module browser. This only works for modules and module groups.



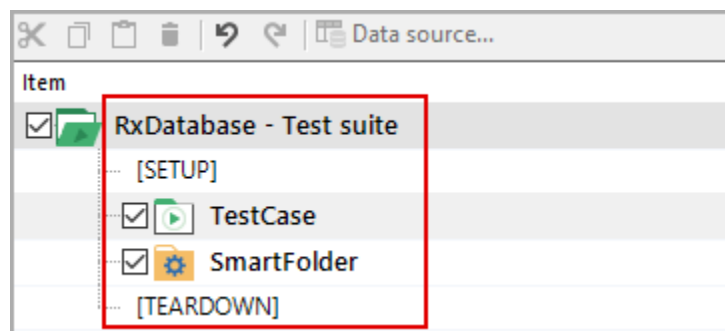
### Note

You can only add items to the test suite as the test suite hierarchy permits. If an item can't be added at the desired position, it is **grayed out** in the menu. The test suite hierarchy is explained at the end of this chapter.

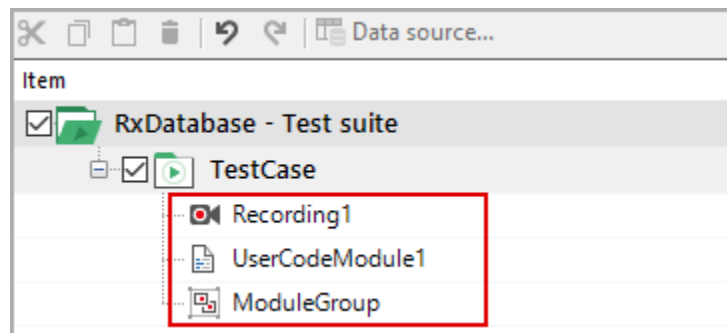
## Test suite items

In this section, you'll learn about the items that make up a test suite. Test suite items can be separated into two groups, **structure items** and **test action items**.

**Structure items** form the framework of your test.



**Test action items** contain the actions to perform during your test.



## The test suite

The test suite item serves as the **root structuring item**. It **contains all other items** and **cannot be cut, copied, pasted, or deleted**. It can contain **only** other structuring items as **direct children**, i.e. no modules or module groups.

The test suite item is where you configure global parameters and report settings for the test suite. Access these through the context menu.

## Test cases

A test case is a **structuring item**. Each test case represents a primary **function of your test suite**, such as adding an entry to a database. **Build** test cases by **populating them with modules** and **module groups**. You can also add **smart folders** to test cases to organize them further. Test cases **cannot contain other test cases**.

You can configure error behavior, data bindings, and conditions for test cases through their context menu.

Test cases are **the only items counted** in the test report success counter.

## Smart folders

A smart folder is a **structuring item**. Use smart folders to **organize your test suite** as you would in a folder hierarchy. You can **populate** smart folders **with modules and module groups**, but only **if** the smart folder is a **child of a test case**.

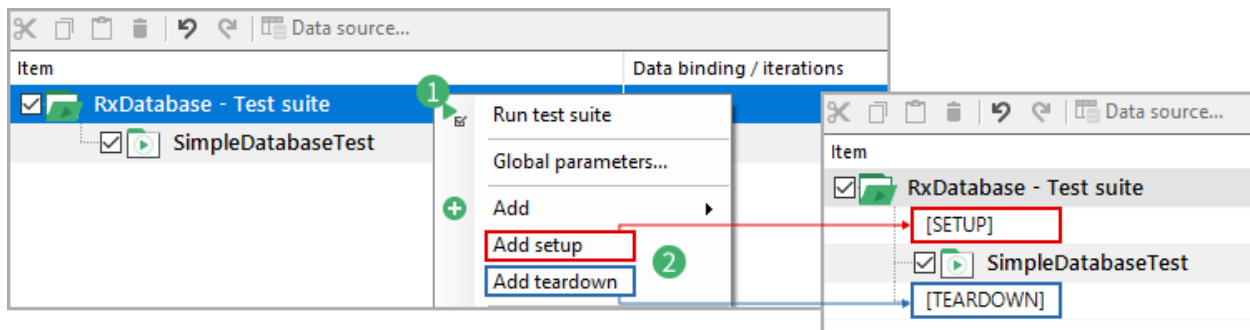
You can configure error behavior, data bindings, and conditions for smart folders through their context menu.

## Setup/teardown regions

Setup and teardown regions are **structuring items**. You can **add one of each** to the test suite item, test cases, and smart folders. The **setup** region will always be placed **at the beginning** of an item and the **teardown** region **at the end**.

To add a setup or teardown region:

- 1 **Right-click** the desired item.
- 2 **Click Add setup** or **Add teardown**.



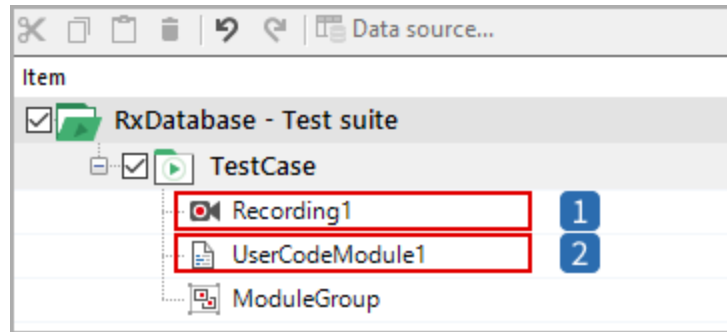
**Setup regions** are always **executed before anything else** in the direct parent item. Populate setup regions with the modules and module groups needed to **bring the AUT to the state required for the following modules to run**, such as starting the AUT and logging in.

**Teardown regions** are always **executed after everything else**, or **when an error occurs** in the direct parent item. Populate teardown regions with the modules and module groups needed to **clean up the AUT after a test run**, such as deleting all entered data and closing the AUT.

## Modules

Modules are test data items. They **contain** the **actions performed during a test**. You can **add** modules to **all** structure items **except** the test suite item and smart folders that aren't the child of a test case.

There are two types of modules:



## 1 Recording modules

- Contain actions you've recorded or added using the Ranorex Recorder and its action table.

## 2 Code modules

- Contain actions programmed in code.

You can configure data bindings for modules through their context menu. You can also group several modules to create **module groups**, as explained below.



### Note

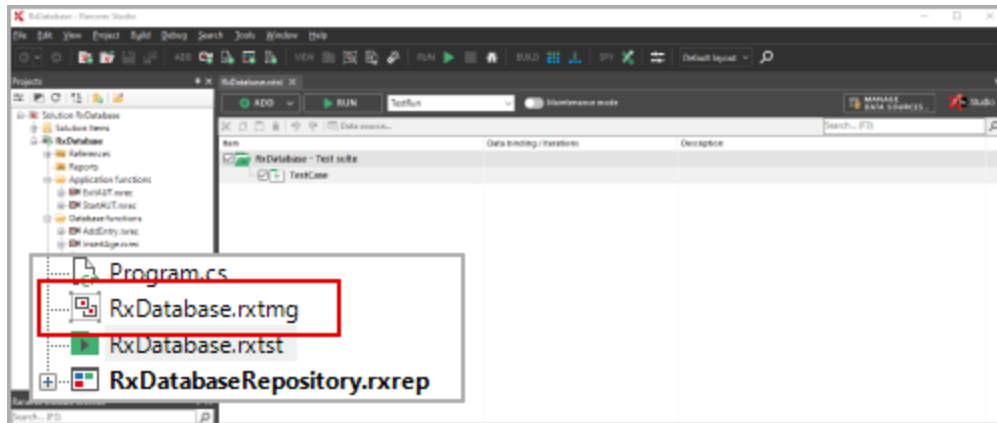
When you add a module to a test suite, you only add a **reference** to it. In the same way, when you delete a module from the test suite, you only delete that reference and not the module itself.

## Module groups

Module groups are test data items. Use them to group modules that logically belong together, such as data validations that usually occur together. You can add module groups to all structure items **except** the test suite item and smart folders that aren't the child of a test case.

All module groups are stored in a separate file. You can locate this file under the project in the projects view. By default, it's named

`[Project name].rxtmg` (for **R**anorex **t**est **m**odule **g**roup).

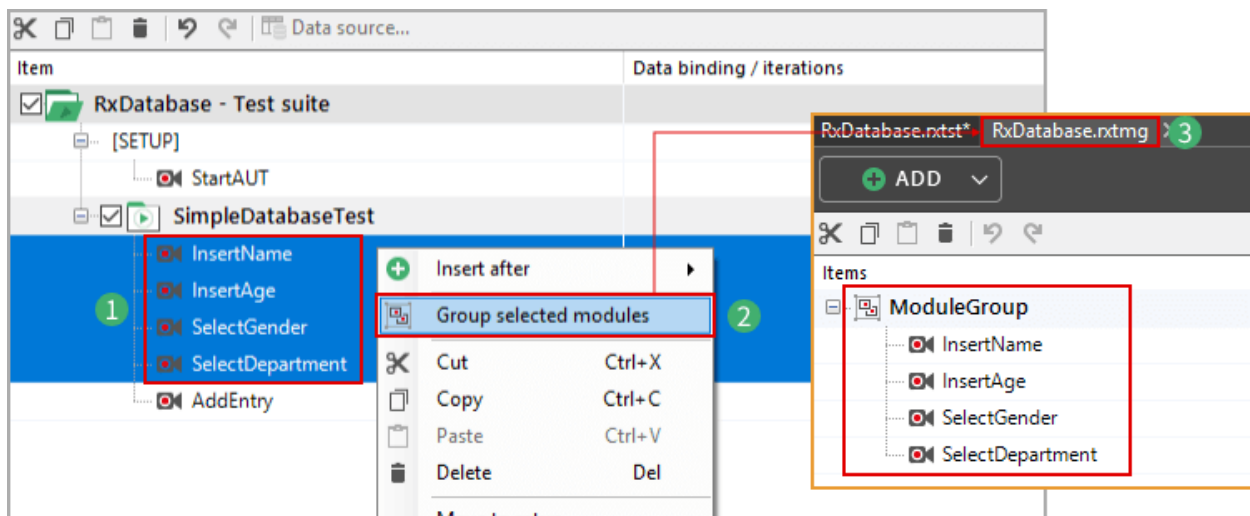


There are two ways to add module groups:

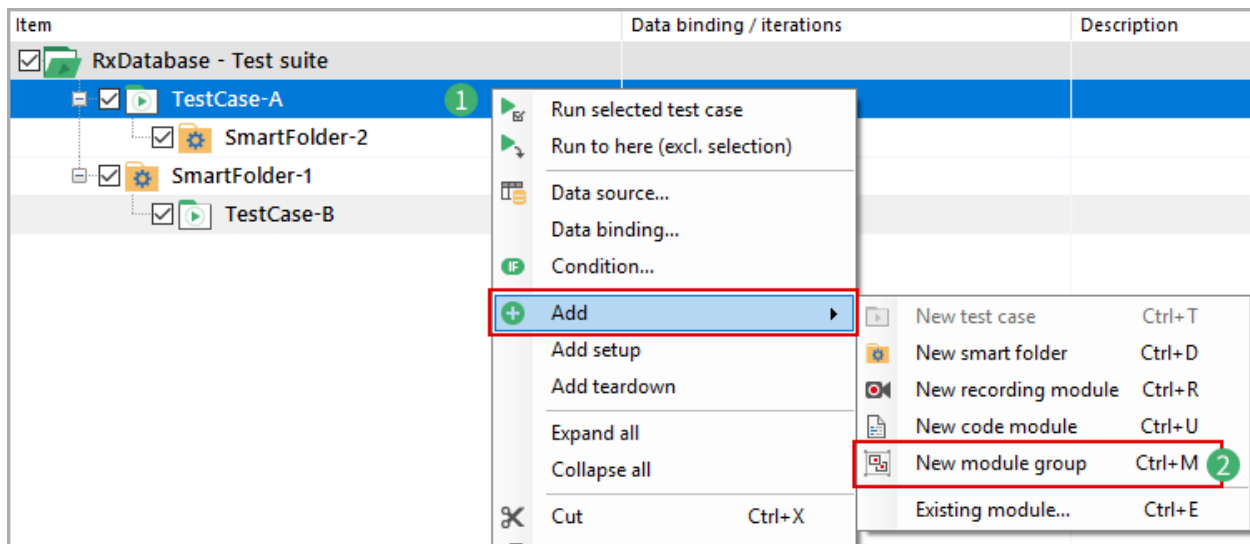
### Direct grouping

- 1 In the test suite view, **select** one or more modules that you want to group.
- 2 **Right-click** one of them and **click Group selected modules**.
- 3 The newly created module group opens in the **module group view**.

The **module group view** opens with the new module group selected. The new module group also appears in the module browser.



- 1 In the test suite view, **right-click** the structure item to contain the new module group.
- 2 **Click Add > New module group**. The module group view opens with the new module group selected.



**3** Drag and drop the desired modules from the module browser to the module group.

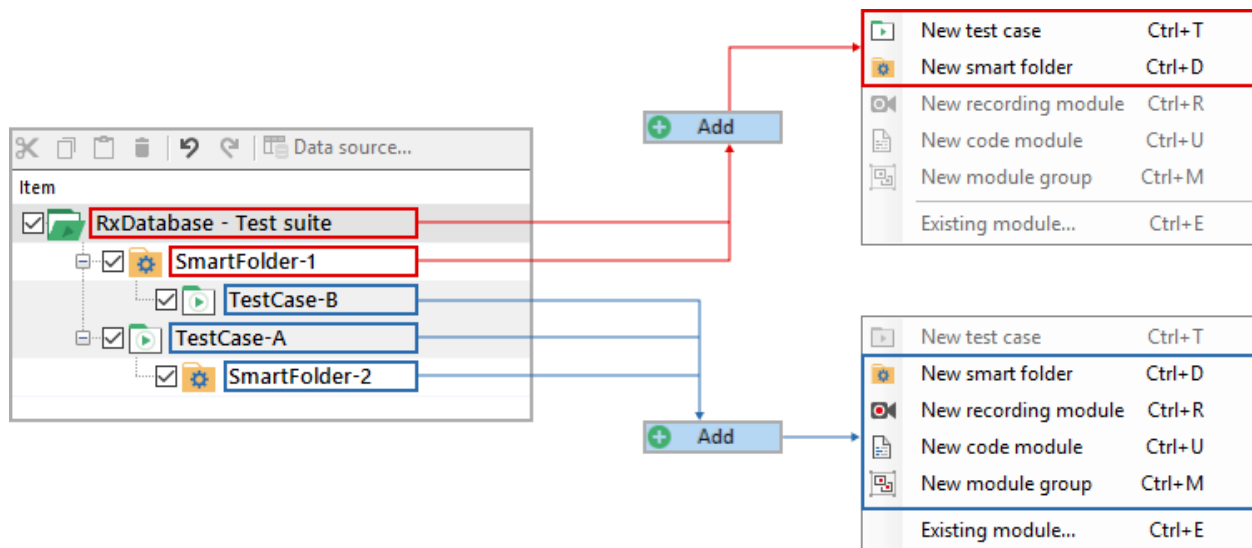
The new module group appears in the module browser. You can **rename** module groups and **organize** them **in folders** in the module group view.

### **Note**

When you add a module group to the test suite, you only add a **reference** to it. In the same way, when you delete a module group from the test suite, you only delete that reference and not the module group itself.

## **Test suite hierarchy**

Test suite items are organized in a hierarchy. This hierarchy controls where items can be added or moved, as illustrated below:



### Note

- Test cases can never be a child of another test case.
- When you move a test case into another test case, Ranorex will automatically convert it to a smart folder for you.
- A smart folder can contain modules or module groups only when it is the child of a test case.

## Build a test

In this chapter, you'll apply what you learned about test suites in the previous chapters. To do so, you will build a simple test using the various test suite items.

### Screencast

The screencast “build a Ranorex Studio test suite manually” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Download the sample solution



To follow along with this tutorial, download the sample solution file from the link below.



## Sample solution

**Theme:** Build a test

**Time:** 15 minutes

[Download sample file](#)

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `RxDatabase.rxsln`



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must accept the automatic solution upgrade for versions 8.2 and higher.

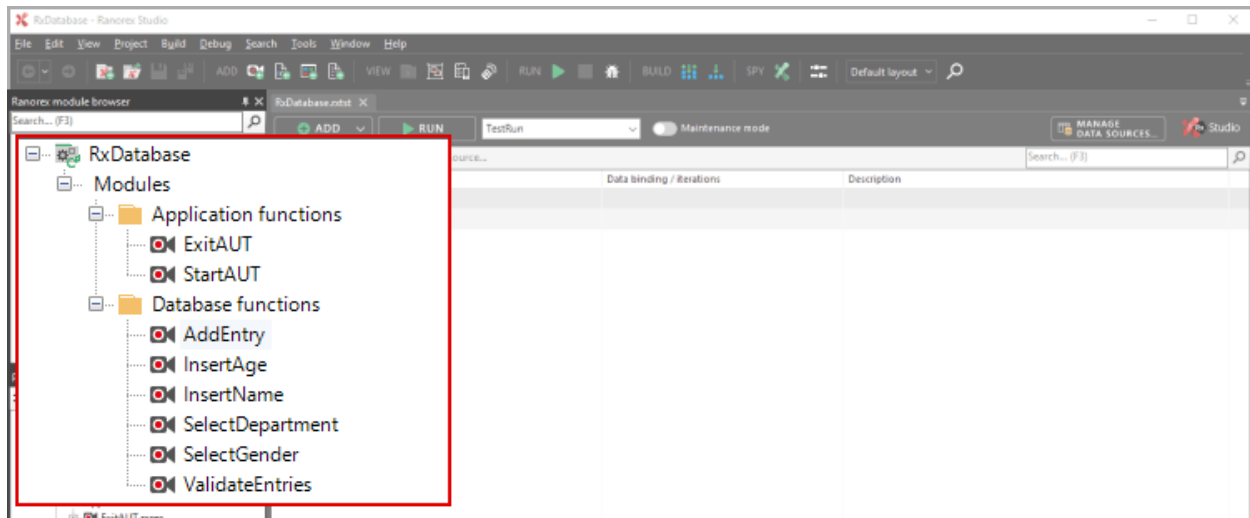
### Define the test

Before we start building our test, we need to define what it'll do. We're keeping it simple, so our test will only add an entry to a database and then validate whether it's been added. Our AUT will be the Ranorex Demo Application. The required steps are:

1. Start the Ranorex Demo Application.
2. Click the **Test database** tab.
3. Enter the first name in the **First name** field.
4. Enter the last name in the **Last name** field.
5. Select the department from the **Department** drop-down.
6. Enter the age in the **Age** field.
7. Select a gender from the **Gender** box.
8. Click **Add Entry**.
9. Validate that **Number of entries** has changed from 0 to 1.
10. Exit the Ranorex Demo Application.

### Review the recording modules

Our sample solution already contains the required recording modules. Let's take a look at them.



The modules are organized in two folders. The **application functions** folder contains the modules needed to control the application itself, i.e. starting and exiting the Ranorex Demo Application. The **database functions** folder contains the modules related to adding an entry to the database.

The modules in these folders are quite specific, as you can see from their names. They each contain only the actions necessary for an individual step in our test definition. Modules built in this way are easier to **reuse**. This gives you more **flexibility** when building tests.

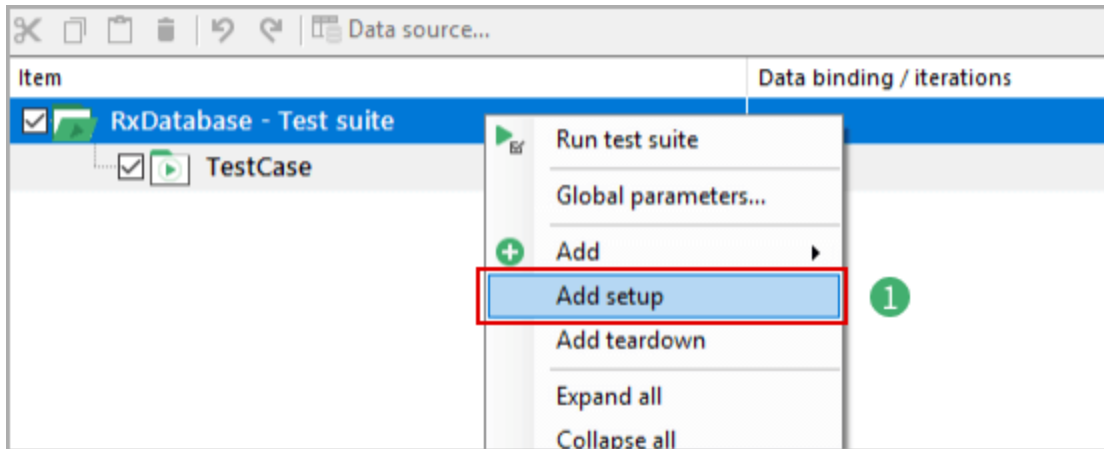
## Assemble the test

We've got everything we need, so let's get to business and start building our test. There is one test suite in our project, `RxDatabase.rxtst`. It should already be open. If it isn't, simply **double-click** the file in the project view.

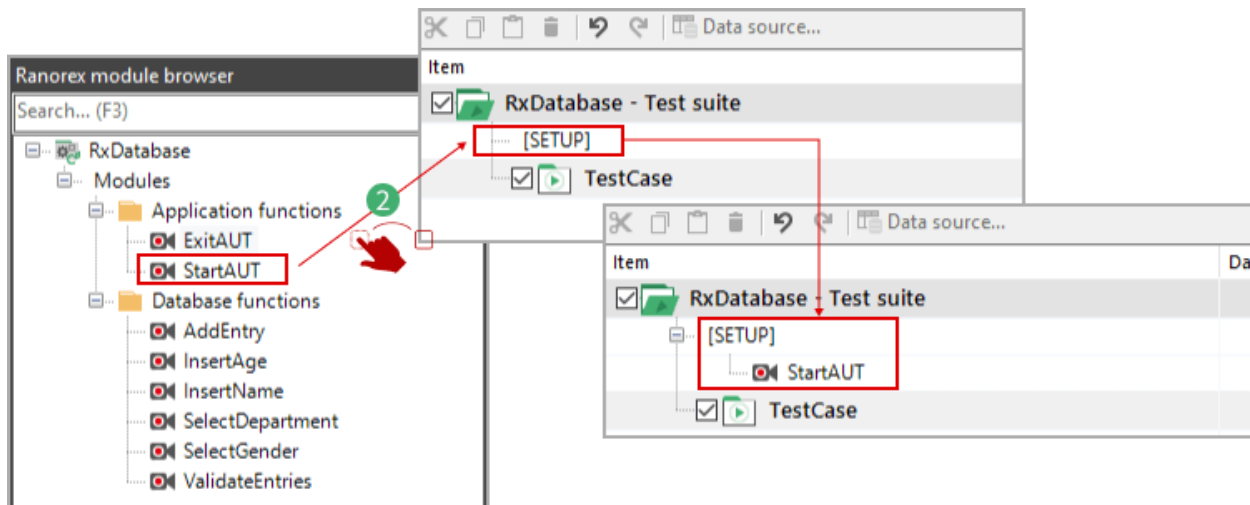
## Add a setup region

In the test suite view, you'll see that the test suite is mostly empty, except for a single test case with the default name. Our test definition lists **starting the AUT** as the first step, so that's what we'll add first. Starting the AUT is a perfect example of a module to include in a global **setup region** because, without it, no other test steps will work. To add a global setup region:

- 1 In the test suite view, **right-click** the test suite and **click Add setup**.



- 2 From the module browser, **drag** the module **StartAUT** to the setup region. In addition to starting the demo application, the StartAUT module also clicks on the Test database tab, which is step 2 from our test definition.

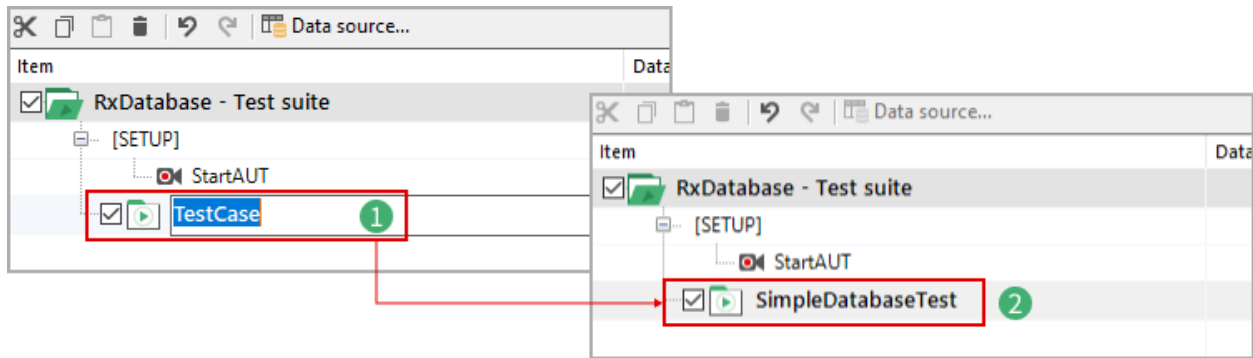


## Add the database test

Steps 3 to 9 in our test definition represent adding an entry to the database and validating it. This is the **core** of our test, so it should get its own **test case**. We can use the existing test case, but let's first give it a more **meaningful name**.

To rename it:

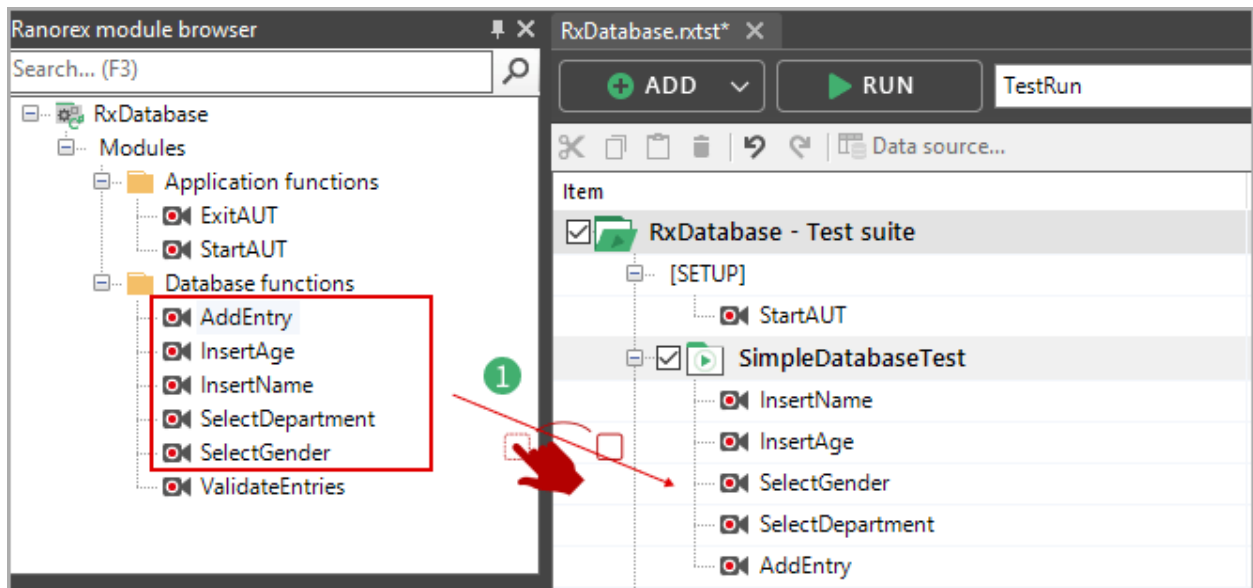
- 1 **Click** the test case and press F2.
- 2 **Rename** it to **SimpleDatabaseTest** and **press** Enter.



Now we can fill the test case with the required modules. These are, in test definition order: **InsertName**, **SelectDepartment**, **InsertAge**, **SelectGender**, **AddEntry**, and **ValidateEntries**. You can add the first four modules in any order you like, but **AddEntry** must be second-to-last, and **ValidateEntries** must be last.

To add the modules:

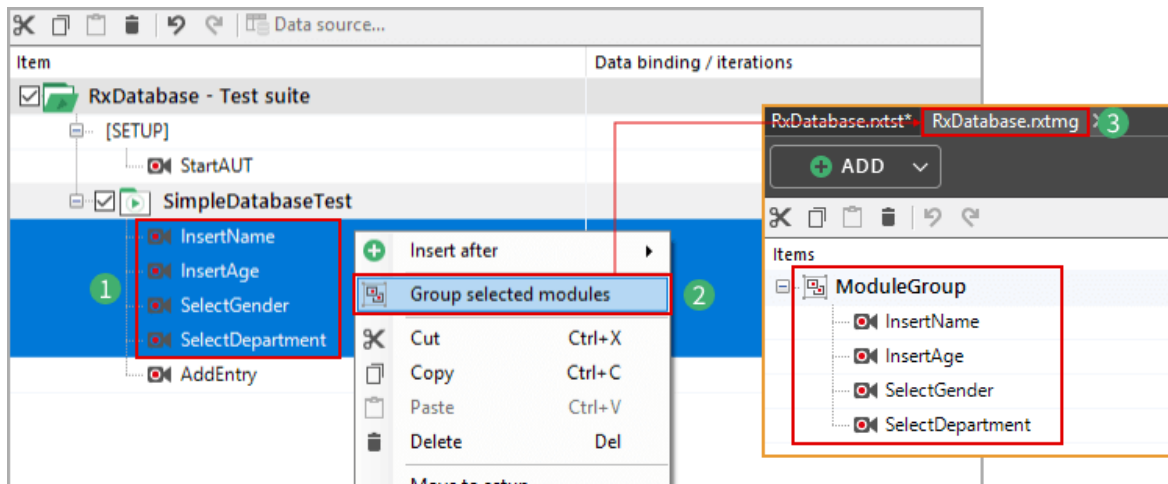
- 1 From the module browser, **drag** the modules to the test case.
  - You can add them individually or select several modules at once using **Ctrl** + Click.
  - If you misplace a module in the test suite view, simply drag it to its correct place.



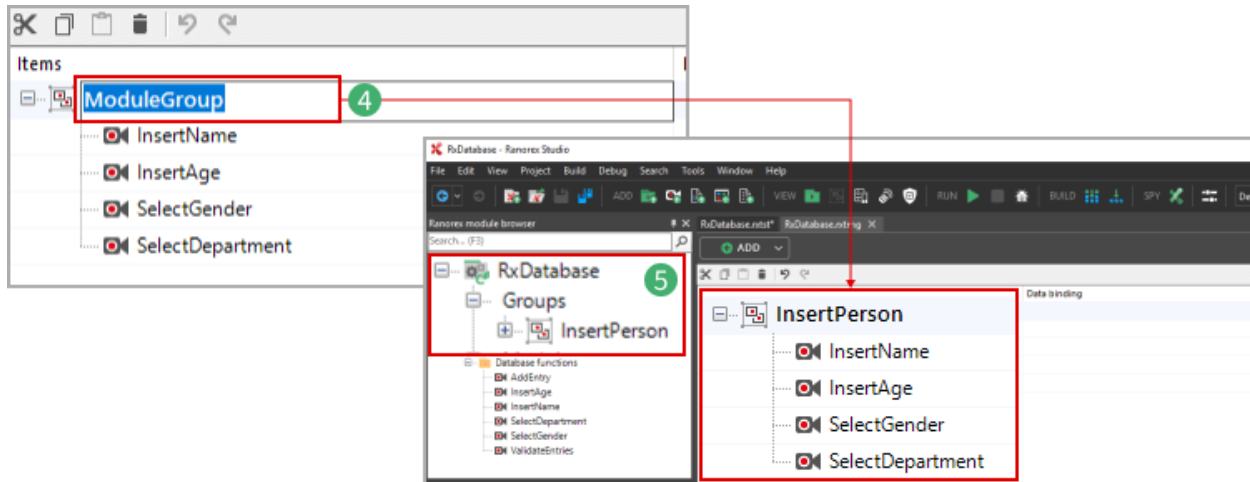
## Create a module group

The modules **InsertName**, **InsertAge**, **SelectGender**, and **SelectDepartment** are all part of the same process: defining the data that will be added to the database, in other words, inserting a person. This is why it makes sense to organize them in a **module group**. This way, you won't have to add all four modules over and over again when you create more test cases where this process is needed. To add the modules to a module group:

- 1 In the **test suite** view, **select** the four modules using Ctrl + Click or Shift + Click.
- 2 **Right-click** the modules; and then **click Group selected modules**.
- 3 The newly created module group opens in the **module group** view.



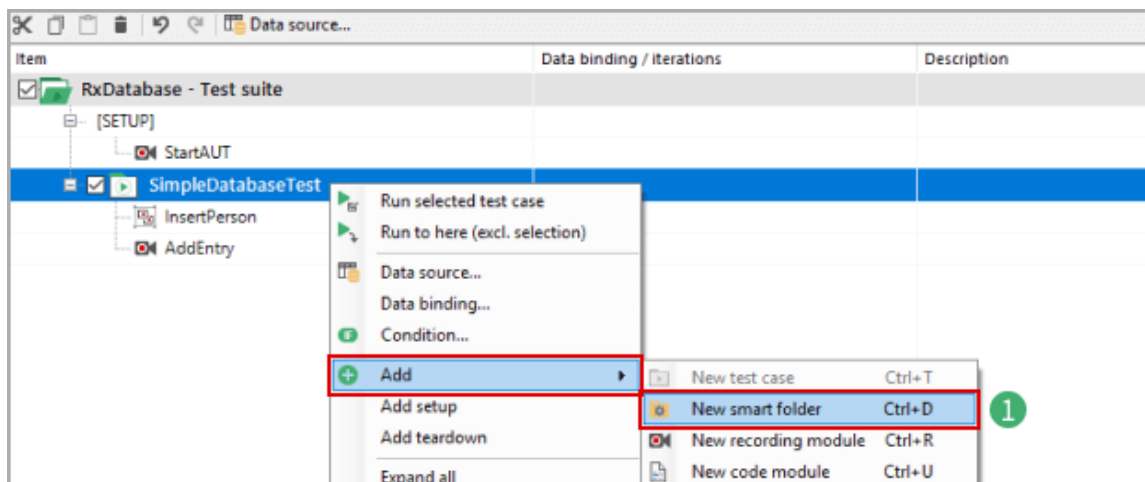
- 4 **Rename** the module group to **InsertPerson** and **close** it.
- 5 The test case now contains the module group InsertPerson and the module group also appears in the module browser for reuse.



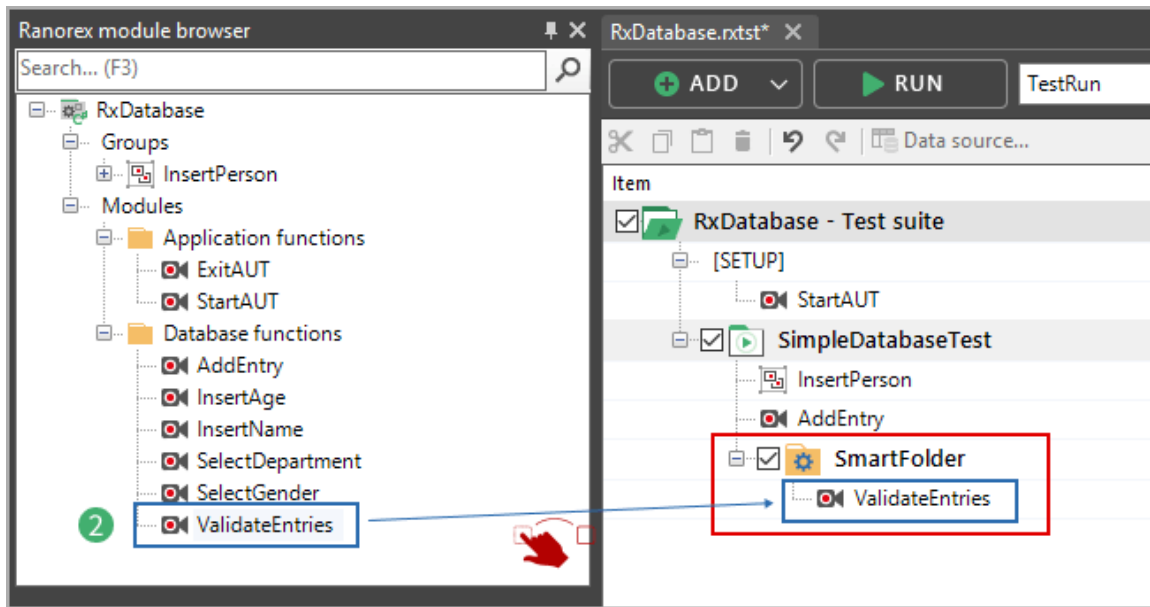
## Add a smart folder

As tests grow larger and more complex, it can become difficult to manage the test suite. **Smart folders** are a useful structuring item to overcome this issue. Instead of filling a test case with 50 modules that all logically belong to the same test, add smart folders to the test case and organize the modules in them. Smart folders also make it easy to **exclude** particular parts of a test case from being run during test execution. This is exactly what we're going to use a smart folder for in our example. We might not always want the validation to be carried out, so it will get its own smart folder. To add **ValidateEntries** to a smart folder:

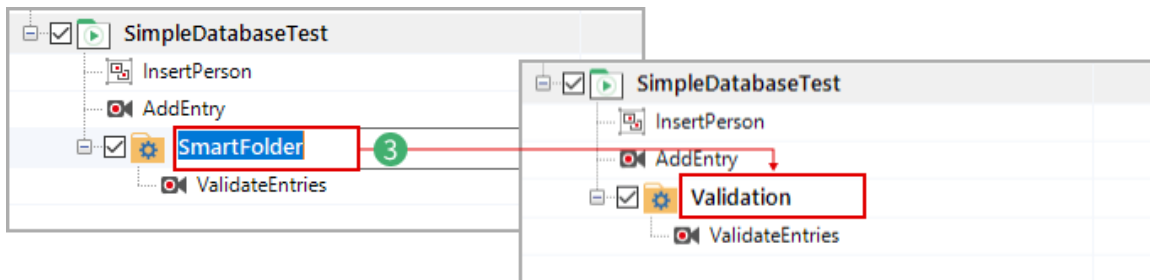
- 1 **Right-click** the test case **SimpleDatabaseTest** and **click Add > New smart folder**.



- 2 **Drag ValidateEntries** to the new smart folder.



### 3 Rename the smart folder to **Validation**.



## Reference

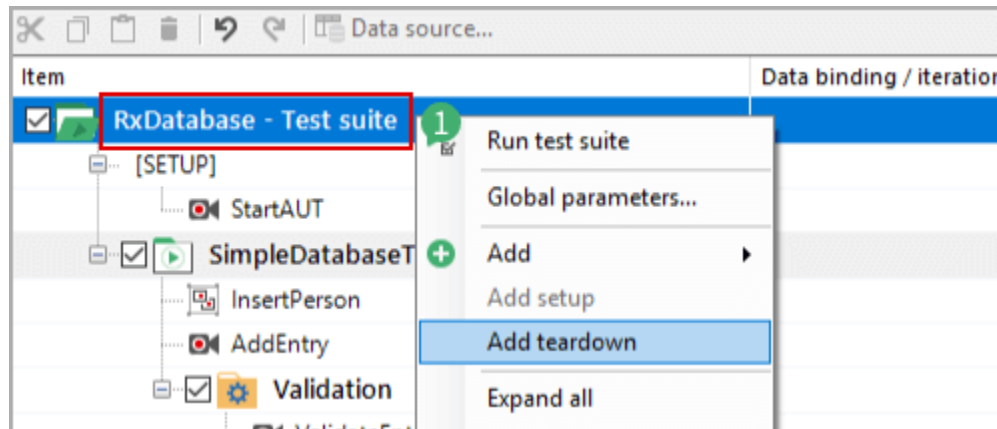
Excluding items from a test run is explained in Ranorex Studio fundamentals > Test suite > → [Execute a test suite](#).

## Add a teardown region

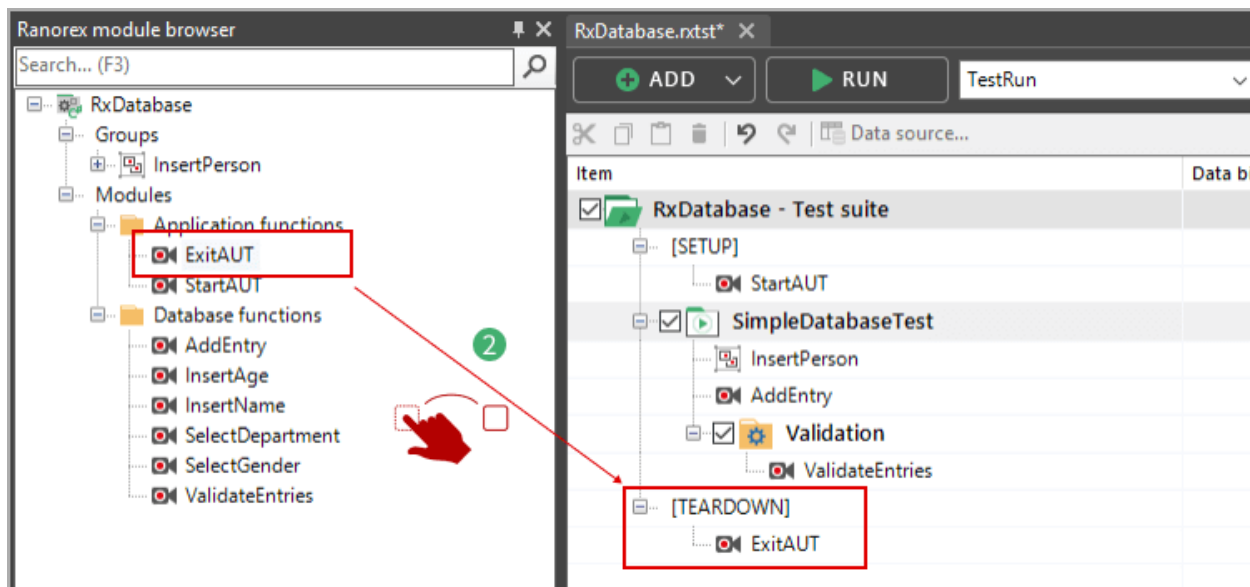
Our test is almost complete. We only need to add the **last step**, exiting the Ranorex Demo Application. This is a perfect example of a step that should be in a global **teardown** region because, after it, no other test actions can be performed. The system is restored to its state before the test. Normally, this would also include deleting all entries made, but our database

doesn't save any entries when exiting, so we don't need to include this action. To add the teardown region:

- 1 In the test suite view, **right-click** the test suite and **click Add teardown**.



- 2 From the module browser, **drag** the module **ExitAUT** to the teardown region.



## Run the test

Congratulations, you've just built a test in the test suite. You can apply the basic principles of this tutorial to any other test you'll create.



This is the point where you'll **run your test**. We'll cover the details and advanced options of running tests in the next chapter. For now, just press the large **RUN** button in the test suite view and enjoy your database test executing!



## Reference

Running tests is explained in detail in Ranorex Studio fundamentals > Test suite > [Execute a test suite](#).

## Execute a test suite

In this chapter, you'll learn about the various options to control test runs and execute tests in the test suite view.



## Screencast

The screencast “test suite execution options” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Download the sample solution

The explanations in this chapter are based on a sample solution. You can download it below.



## Sample solution

**Theme:** Test suite run

**Time:** Less than 10min

[Download sample file](#)

## Installation:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `Introduction.rxsln`.

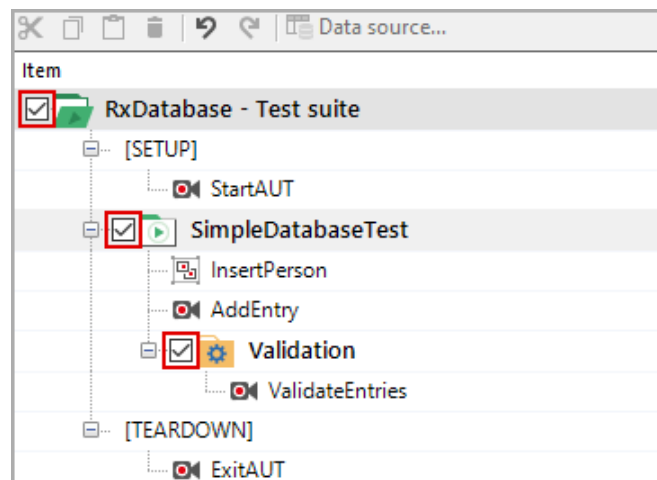


### Hint

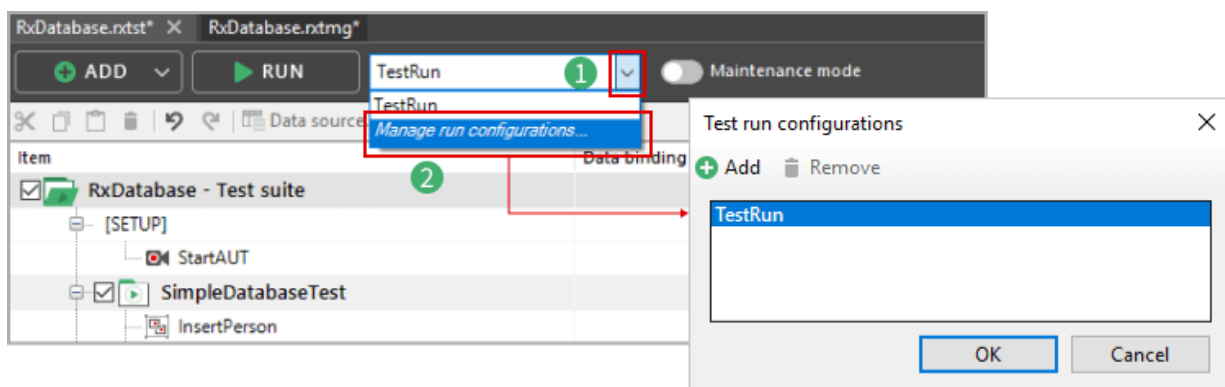
The sample solution is available for Ranorex versions 8.0 or higher. You must accept the automatic solution upgrade for versions 8.2 and higher.

## Manage run configurations

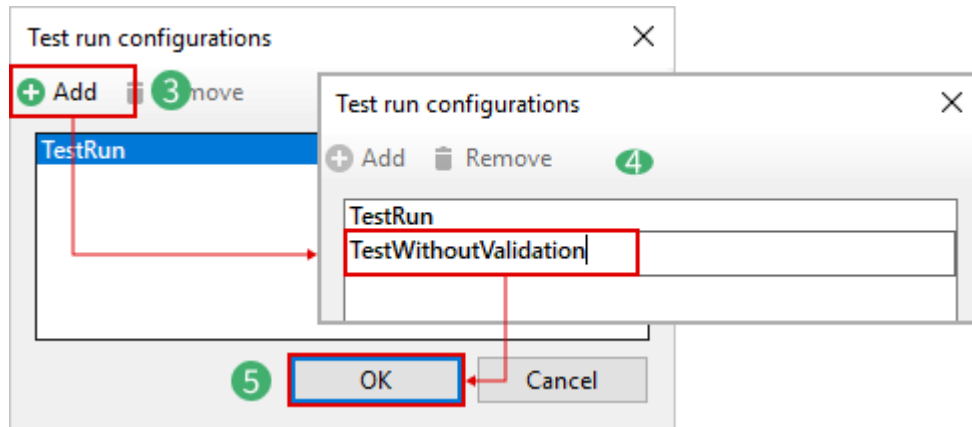
You can **include and exclude** test cases and smart folders from a test run. To do so, simply **check or uncheck** them in the test suite.



The current state of checked/unchecked test cases and smart folders is called a **run configuration**. You can **save** run configurations for reuse and **switch** between them using the drop-down menu in the test suite view.



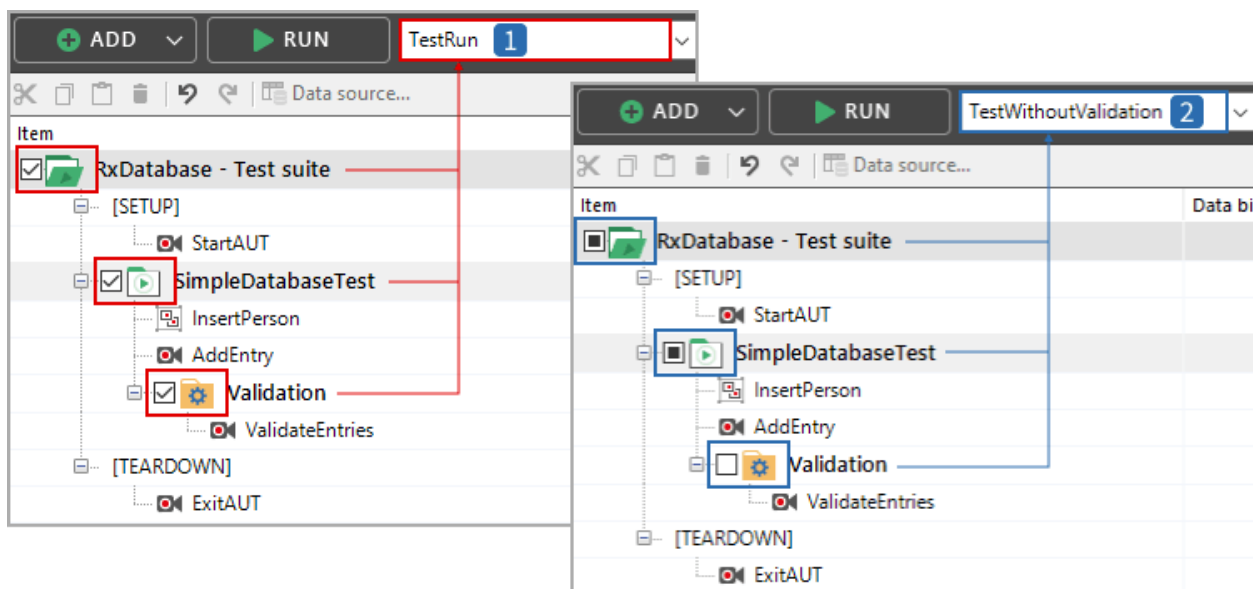
- 1 Click the run configurations drop-down menu.
- 2 Click **Manage run configurations...**



- 3 Click **Add**.
- 4 Give the new run configuration a meaningful name.
- 5 Click **OK**.

**Result(s):**

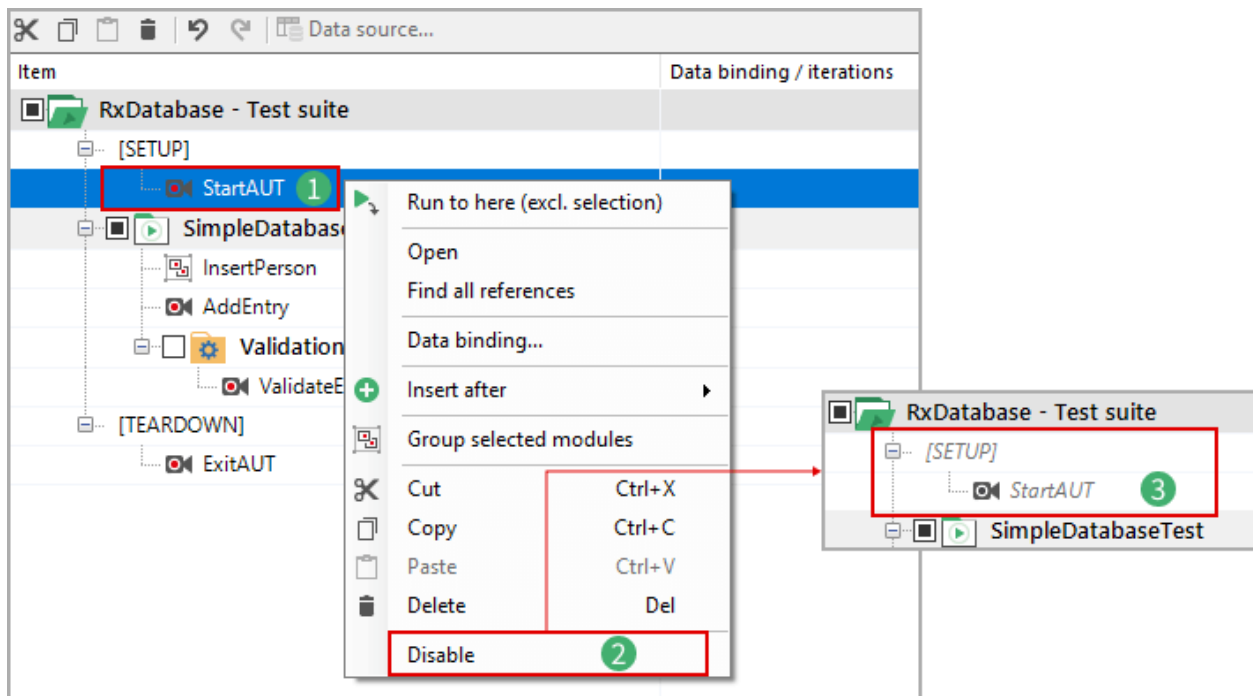
- The run configuration can be selected from the drop-down menu.



- 1 **TestRun** configuration including all test suite elements
- 2 **TestWithoutValidation** configuration excluding the validation smart folder

## Disable/enable test suite items

Similar to including/excluding test cases and smart folders, you can also **enable and disable** recording modules, code modules, module groups, and setup and teardown regions. Disabled items are **not executed** during a test run.



- 1 **Right-click** the item you want to disable.
- 2 **Click Disable.** The item will be grayed out.

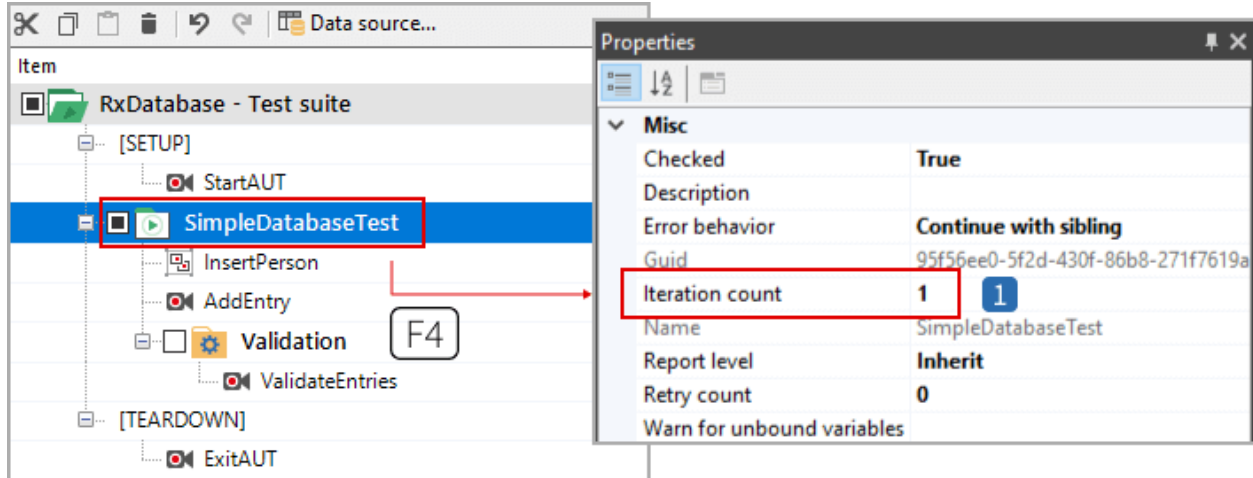


### Note

- Disabling a **module group** disables all modules in this module group.
- Disabling a **setup/teardown region** disables all items in it.

## Configure run iterations

By default, test cases and smart folders are run **once** during test execution. However, you may want to **run them several times**. You can do so with **run iterations**.

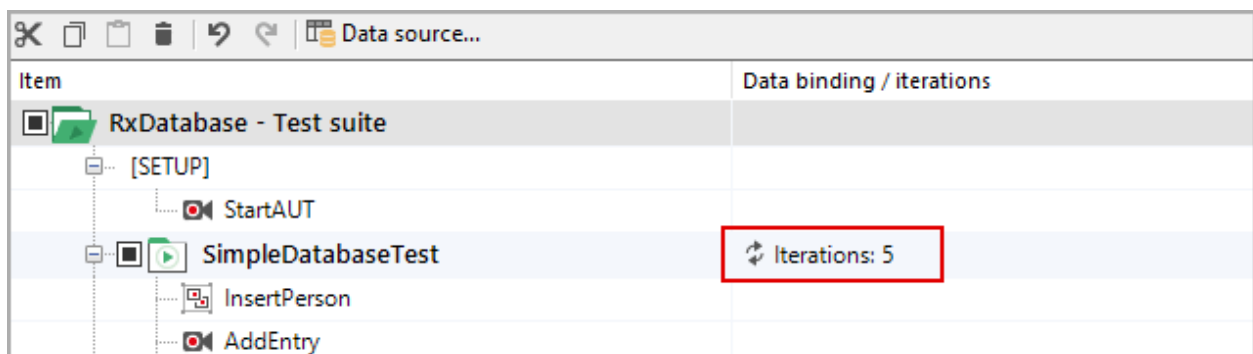


- 1 With a test case or smart folder selected, **press F4**.
- 2 The **Properties** pad appears to the right of the test suite.
- 3 Next to **Iteration count**, **set** the desired number of iterations.

### 1 Iteration count in the Properties.

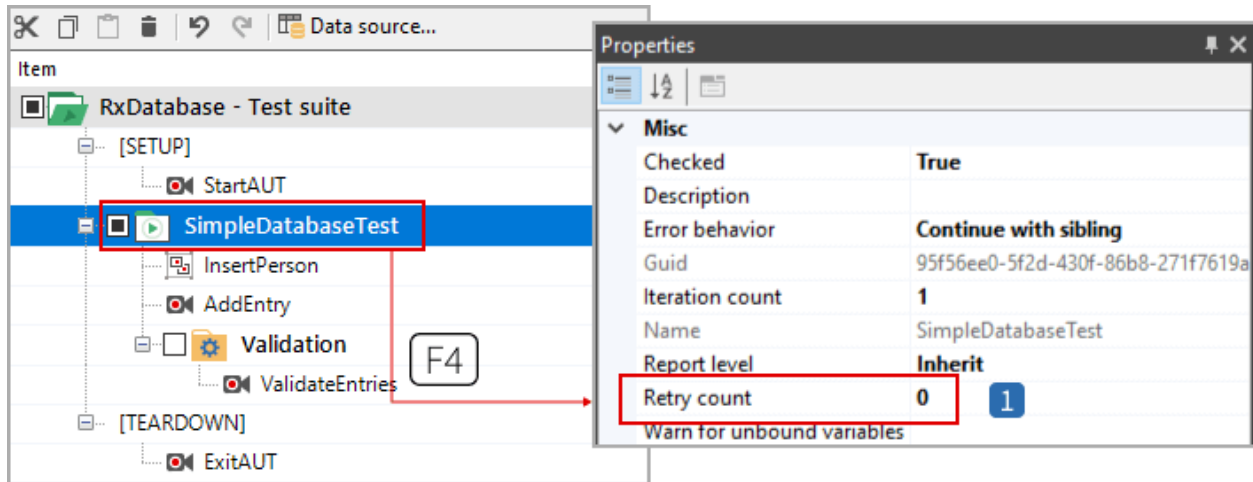
#### Result(s):

- The number of iterations appears next to the test suite item.
- In our example, the test case is iterated 5 times.



## Configure auto-retry

In UI testing, errors will sometimes occur simply because the application under test was **unresponsive**. In these cases, one solution is to **simply re-run parts of the test**. You can do so with **auto-retry**. Test cases or smart folders with an auto-retry count will be **rerun until** they are either **successful or all retries have been used up**.



- 1 **Select** the desired test case or smart folder.
- 2 **Press** **F4**. The **Properties** pad appears to the right of the test suite.
- 3 Next to **Retry count**, **set** the desired number of retries.

### 1 **Retry count** in the **Properties**.



#### Note

If there are data bindings or run iterations, the retries will start at the point of failure. For example, if the failure occurred at iteration 3 of 5, that's where the retry will start

Only test cases or smart folders that failed every single retry will be marked as failed in the report.

## Configure report levels

You can also set the **report level** of test cases and smart folders in the test suite. Report levels allow you to **control what information appears** in the report, and where it appears. This is especially useful for complex tests with many test cases and smart folders, to keep the report structured.



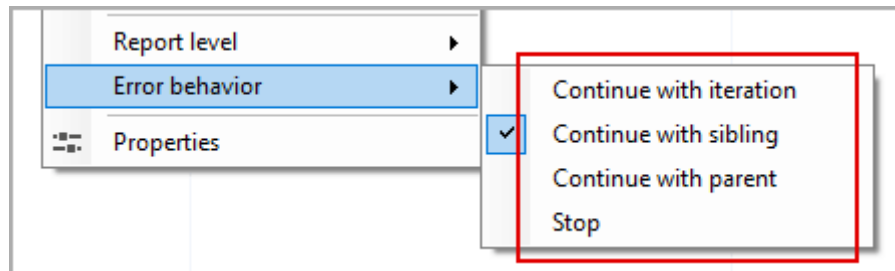
### Further reading

Report levels are beyond the scope of this chapter. They are explained in Ranorex Studio fundamentals > Reporting > [Report levels](#).

## Configure error behavior

Errors are part of testing. This is why it's important to tell Ranorex **what to do when an error occurs**. You can do so by configuring the **error behavior** of test cases and smart folders in the test suite.

The error behavior is set up in the context menu of a test case or a smart folder. There are **four different types**, explained below.



### Hint

The default error behavior is **Continue with sibling**.

## Continue with iteration

Item	Data binding / iterations
✓ RxDatabase - Test suite	
[SETUP]	
StartAUT	
✓ SimpleDatabaseTest	
InsertPerson	
AddEntry	
✓ Validation	Iterations: 5
ValidateEntries	Error
DataBaseCleanUp	
✓ CompleteDatabaseTest	
[TEARDOWN]	
ExitAUT	

### 1 Continue with iteration.

The test run will continue with the next iteration of the smart folder **Validation**.

## Continue with sibling

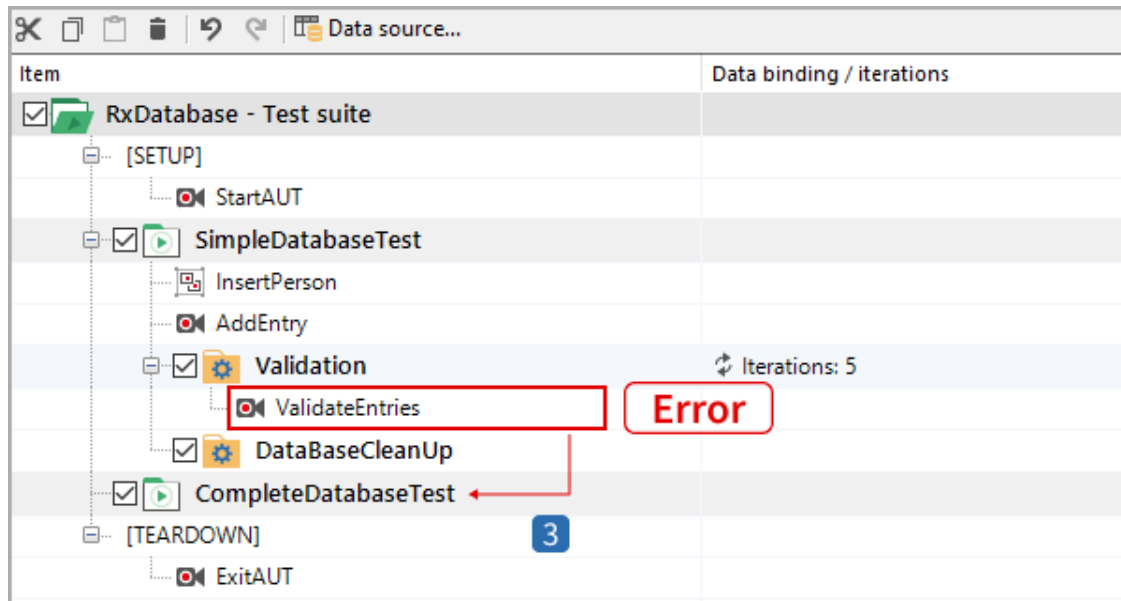
Item	Data binding / iterations
✓ RxDatabase - Test suite	
[SETUP]	
StartAUT	
✓ SimpleDatabaseTest	
InsertPerson	
AddEntry	
✓ Validation	Iterations: 5
ValidateEntries	Error
DataBaseCleanUp	
✓ CompleteDatabaseTest	
[TEARDOWN]	
ExitAUT	

### 2 Continue with iteration.



The test run will continue with the next sibling test case or smart folder. In our case, this is the smart folder **DatabaseCleanUp**.

### Continue with parent



### 3 Continue with parent

The test run will continue with the next parent test case or smart folder. In our case, this is the test case **CompleteDatabaseTest**.

### Stop

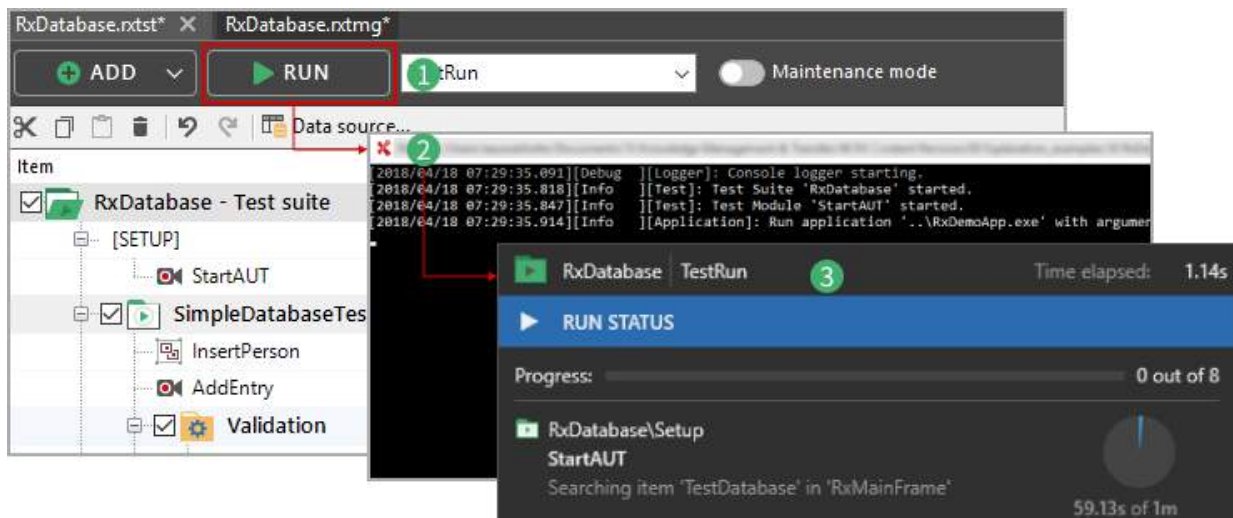
When the **Stop** error behavior is selected, an error immediately stops the entire test run.

### Error behavior of setup/teardown regions

Setup and teardown regions follow a special, fixed error behavior.

- An error in the **setup** region immediately stops the test.
- If a module fails in the **teardown** region, the next module is run.

### Run a test from the test suite view



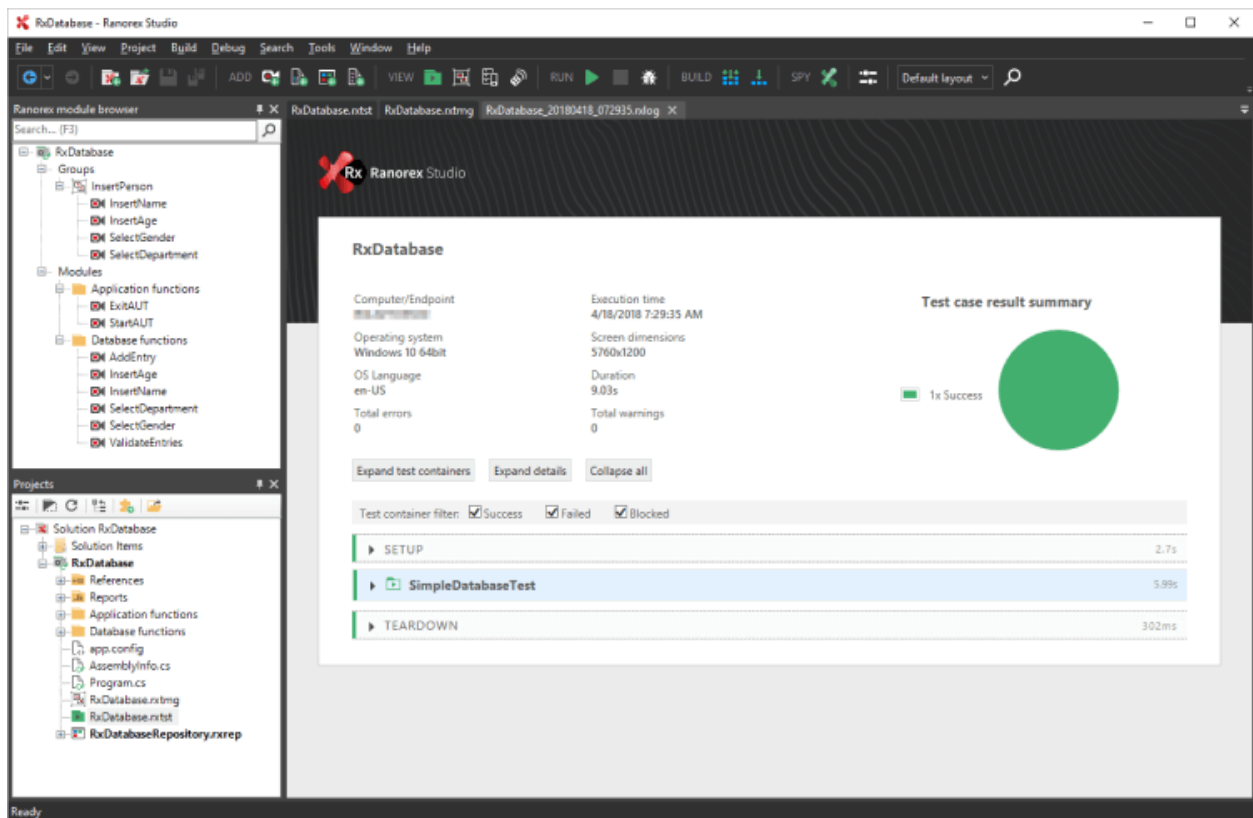
- 1 Click **RUN**.
- 2 Watch Ranorex Studio execute the test.
- 3 Observe the progress information during the test.

### Attention

After you've clicked **RUN**, **do not use the keyboard or mouse**. Doing so would interfere with the test actions and **cause a test failure**.

### Result(s):

- Once the test run finishes, the report appears.



## Further reading

Reports are described in

Ranorex Studio fundamentals > Reporting > [Introduction](#).



## Reference

You can also pause and resume test runs. This is explained in

Ranorex Studio fundamentals > Ranorex Recorder > [Run and debug recordings](#).

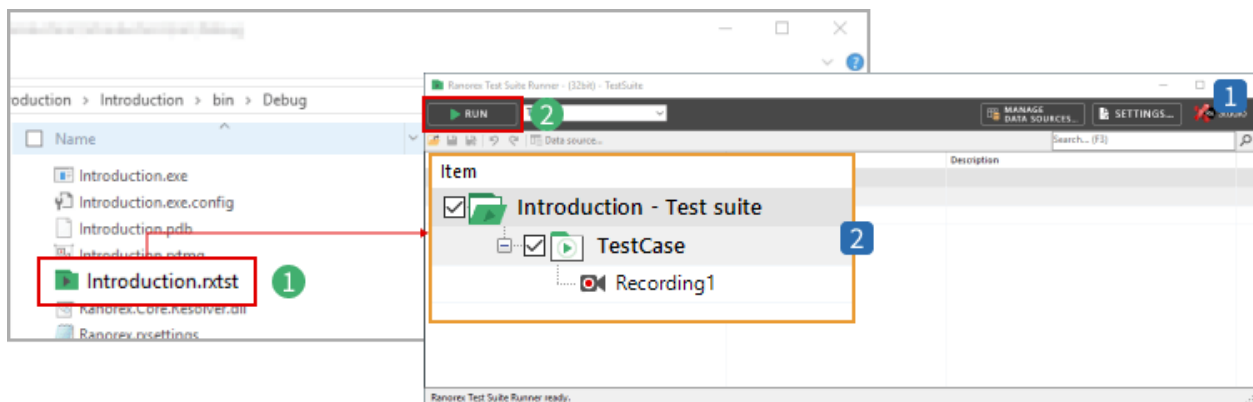
## Ranorex Test Suite Runner

The Test Suite Runner is a stand-alone program which executes test suites without Ranorex Studio. It opens automatically when you double-click a test suite file in Windows.

You can use the Ranorex Test Suite Runner to execute entire test suites, run certain test cases and smart folders, or just run a specific module.

In addition, you can create new run configurations the same way as in Ranorex Studio.

You can't make changes to the test suite itself, however.



1 In Windows, **double-click** a test suite file. The file opens in the Test Suite Runner.

2 Click **RUN**.

1 Stand-alone **Test Suite Runner**

2 **Currently loaded test suite**

## Manage multiple test suites

You can add multiple test suites per project and run them in sequence. This is especially useful when collaborating with others.

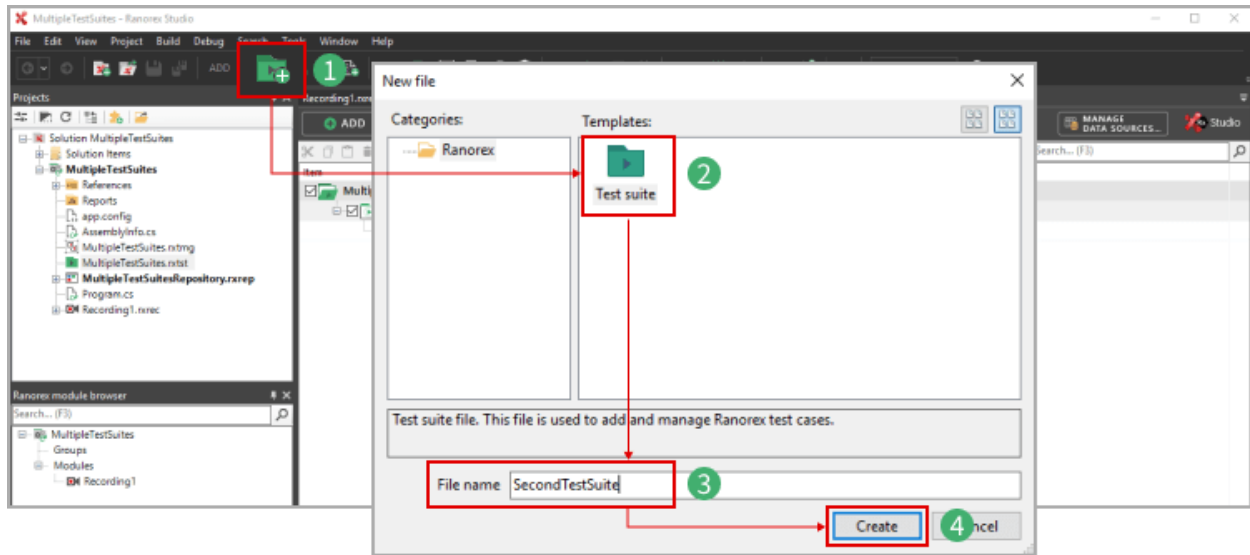
### **Screencast**

The screencast “Manage multiple test suites” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Create a new test suite

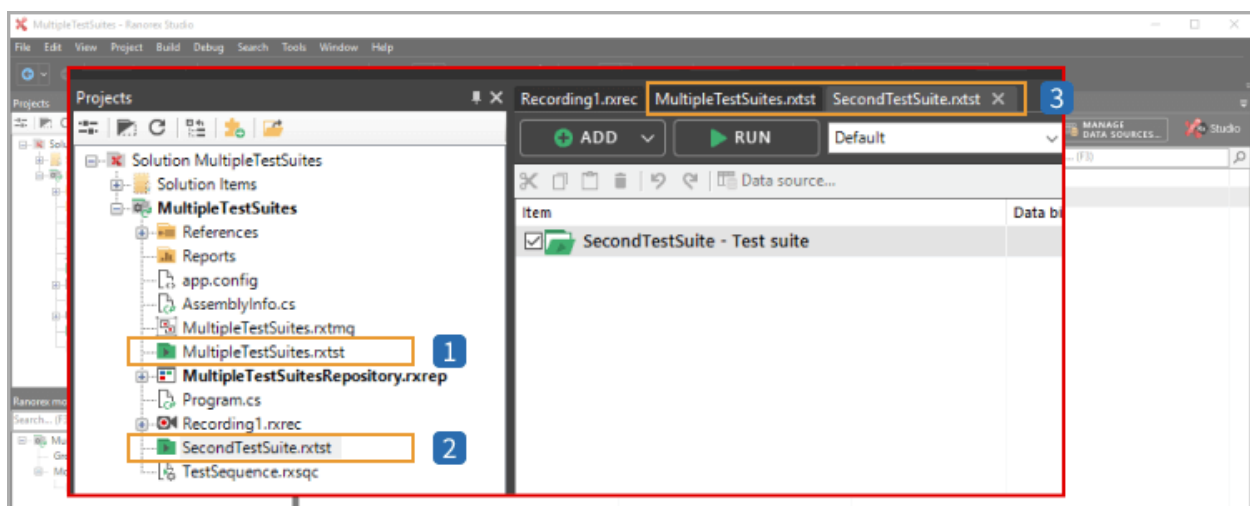
To add a test suite to an existing project, follow the instructions below:



- 1 In the Studio toolbar, **click the Add test suite** button.
- 2 In the dialog that appears, **verify that Test suite** is selected as the template.
- 3 **Name** the test suite.
- 4 **Click Create.**

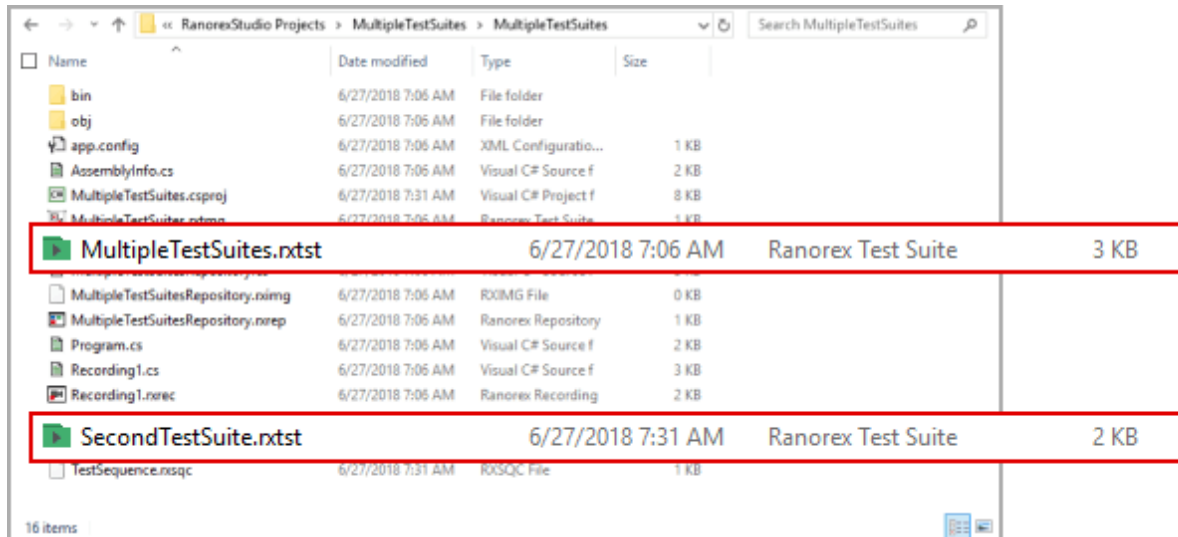
### Result(s):

The new test suite appears in the project view. Usually, it also opens automatically in the test suite view.



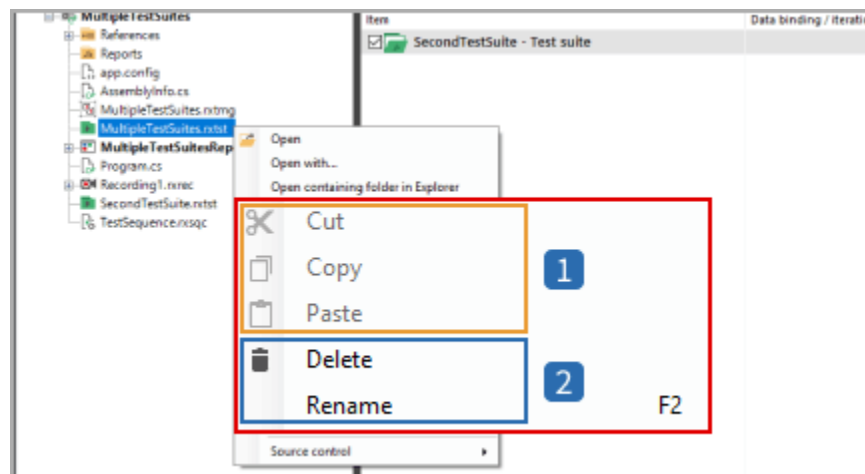
- 1 The **initial test suite** in the projects view.
- 2 The **newly added test suite** in the projects view.
- 3 The **test suite tabs** below the Ranorex Studio toolbar.

The new test suite file also appears in the **Ranorex Studio Projects** folder of your test solution.



## Cut/copy/paste/delete/move multiple test suites

To ensure the integrity of your project, you cannot cut, copy, or paste an entire test suite. However, you can delete and rename test suites. Select a test suite in the project view and open the context menu to use the delete and rename options:



## 1 Cut, Copy, Paste

- An entire test suite **cannot** be cut, copied, or pasted.
- Similarly, you can't **move** an entire test suite to another project.

## 2 Delete, Rename

- Test suites can be deleted.
- Test suites can be renamed.



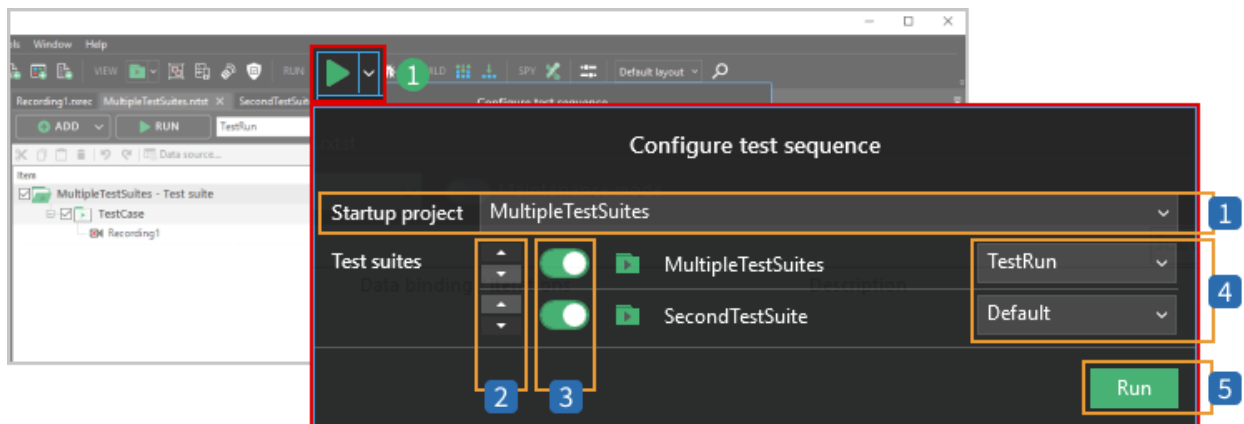
### Note

While you can't move an entire test suite, **you can move test suite content between test suites**. Simply cut/copy and paste the desired test cases, smart folders, modules, etc.

## Set the test sequence

Once a project contains at least two test suites, you can specify a sequence for the test suites.

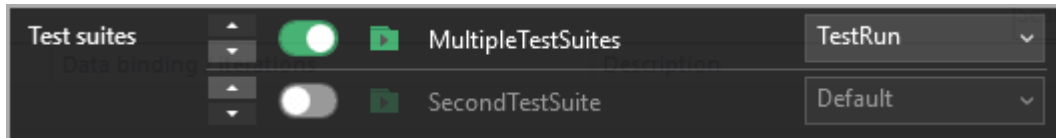
- 1 If your project contains at least two test suites, a context menu will be added to the **RUN** button. Click it to open the test sequence configuration dialog.



## 1 Startup project selection

- If you have multiple projects with multiple test suites, select the project to configure here.
- By default, the current test project is pre-selected.

- 2 Use the **order buttons on the left** to change the position of the test suites in the test sequence.
- 3 Use the **ON/OFF** buttons to exclude selected test suites from the test run.



- 4 Select the **run configuration** for each test suite.



### Further reading

Run configurations are described in [Ranorex Studio fundamentals > Test suite > Execute a test suite](#).

- 5 Click the **Run** button to start the specified test sequence.

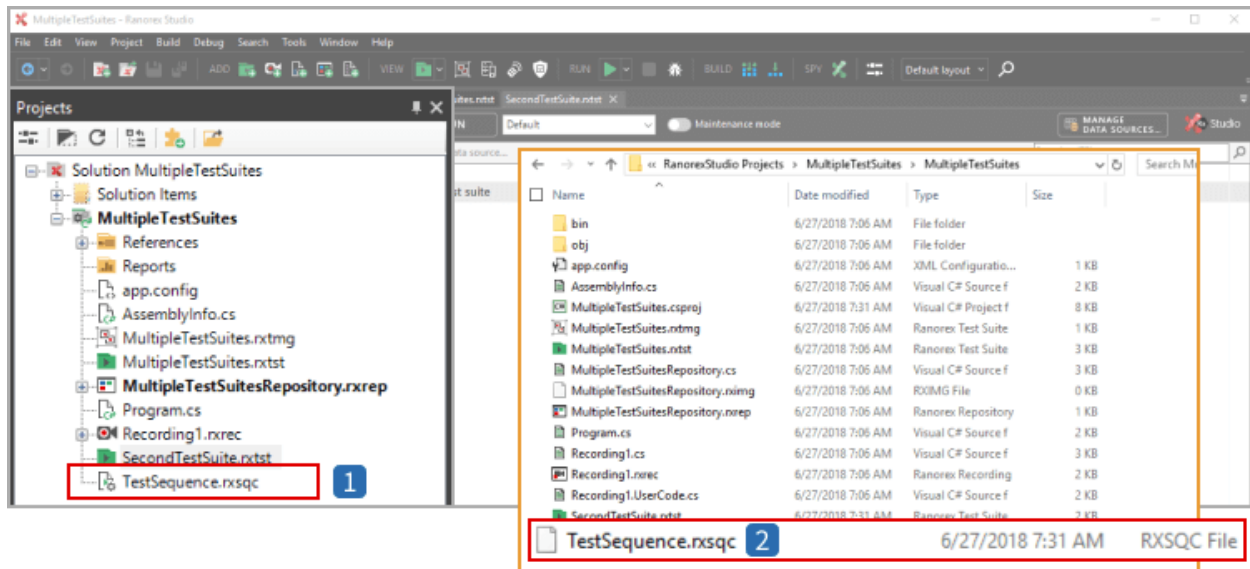
## Manage the test sequence file and parameters

This section introduces the configuration file where the test sequence is stored and describes how to add test suite run parameters to this file.

### Test sequence file

When a project contains at least two test suites, a test sequence file with all configuration information is created automatically and added to the project folder.



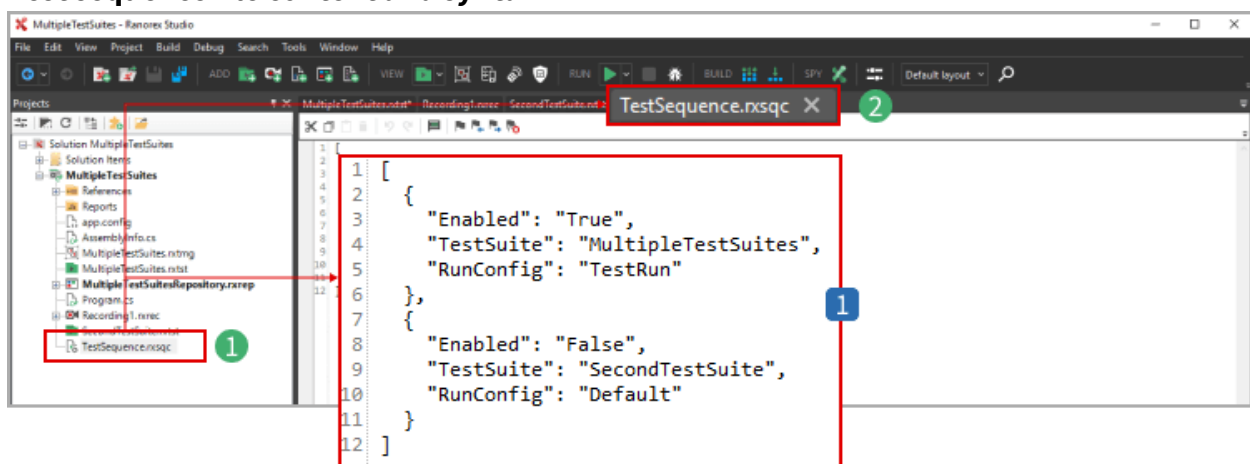


- 1 Test sequence file in the **project view**
- 2 Test sequence file in the **project folder**

### Note

The test sequence file has the file ending **.rxsqc** and can be edited with any text editor.

## Test sequence file content and syntax



1 To open the test sequence file, **double-click** it. It appears in its own tab.

1 **Text content** of the test sequence file:

- `[ ]` enclose the test sequence file
- `{ }` enclose all configurations for one test suite
- Multiple test suites are separated by a `,` (JSON formatting syntax)
- Configuration arguments follow the syntax `" " : " "`
- Multiple parameters are separated by a `,`



### Hint

You can configure your test sequence in the configuration dialog while the test sequence file is open. The changes will be reflected in the file in real time. Inserted command line arguments will remain intact.

## Adding command line arguments

Command line arguments can be added to the test sequence file and will be executed during the test run.

```
1 [
2   {
3     "Enabled": "True",
4     "TestSuite": "MultipleTestSuites",
5     "RunConfig": "TestRun"
6   },
7   {
8     "Enabled": "True",
9     "TestSuite": "SecondTestSuite",
10    "RunConfig": "Default",
11    "testcase": "InsertName"
12  }
13 ]
```

1 Example command line argument `/testcase`

- Command line arguments follow the previously introduced JSON syntax definition
- Two or more arguments must be separated by a `,`
- The example above calls the `/testcase` command line argument with a test case named `InsertName` as parameter value



## Further reading

Command line arguments for test suites are explained in Ranorex Studio fundamentals > Test suite > [Run tests without Ranorex Studio](#).



## Hint

While command line arguments in the runtime environment require **file endings** (i.e. `.rxtst`) for file parameters, these endings are discarded in the test sequence file.

## Multiple test suites – feature list

This section describes the **main features of multiple test suites** within Ranorex Studio. It lists **what can and can't be done** by means of multiple test suites.

### Transferring test suite contents

The contents of one test suite can be transferred (copied, moved) to any other test suite of the same project or another project. This means that test cases, smart folders, setup and teardown regions, modules, etc. can easily be copied or moved from one test suite to another.

### Data integrity

When you copy or move test suite contents to another test suite, data integrity with respect to variables and data binding will remain intact. No data binding is lost.



## Attention

Data-binding will remain intact if either of the following conditions is true:

- data is bound to global parameters which are also available in the destination test suite
- the complete data environment is copied.

If data is bound to parent test cases or parent smart folders which are not copied, data binding will be lost!

## TestRail integration

TestRail can also export/synchronize multiple test suites per project.



### Further reading

The TestRail integration with Ranorex is described in Interfaces & Connectivity > TestRail integration > [Introduction](#).

## Ranorex Remote

Test sequences can also be run on Ranorex Agents from the remote pad.



### Further reading

Ranorex Remote is explained in Ranorex Studio advanced > Ranorex Remote > [Introduction](#)

# Actions

An action is an individual activity performed by a module. In Ranorex, there are two groups of actions: basic and smart actions. **Basic actions** represent direct user input, such as mouse clicks. **Smart actions** represent more complex input or special functions, such as running a browser or carrying out a validation. Most actions are performed on repository items. Actions are managed in the actions table in the Recorder view.










## Screencast

The screencast “Introduction to actions” walks you through information found in this chapter.

[Watch the screencast now](#)

## Basic actions

Basic actions represent direct user input with or on a device such as mouse clicks or keyboard input. These actions are automatically recorded when you perform them during a recording, but can also be added manually in the actions table. The table below lists the icon and name of each basic action. For more details about each basic action, refer to the following page: [Action properties](#).



















	Mouse
	Mouse wheel
	Touch
	Swipe gesture
	Key shortcut
	Key sequence
	Mobile key press

## Smart actions

Smart actions represent more complex UI interactions and functions. For example, the **Run application** action directly runs an executable from a specific path without the mouse clicks

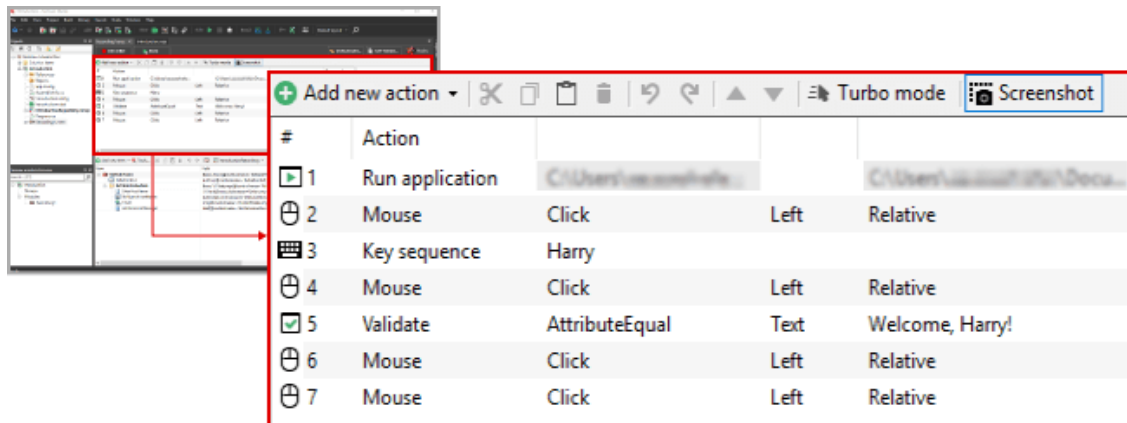
or keyboard interactions normally required to do so. During a recording, none of these actions are available, except for the Validation action. For more details about each smart action, refer to the following page:

→ [Action properties](#).

	Validation
	Invoke action
	Get value
	Set value
	Open browser
	Run application
	Run mobile application
	Deploy Android app
	Deploy iOS app
	Set device orientation
	Close application
	Wait for
	Log message
	Capture screenshot
	Create snapshot
	Separator
	Delay
	User code

## Actions in the actions table

Actions are managed in the actions table in the Recorder view.



## Note

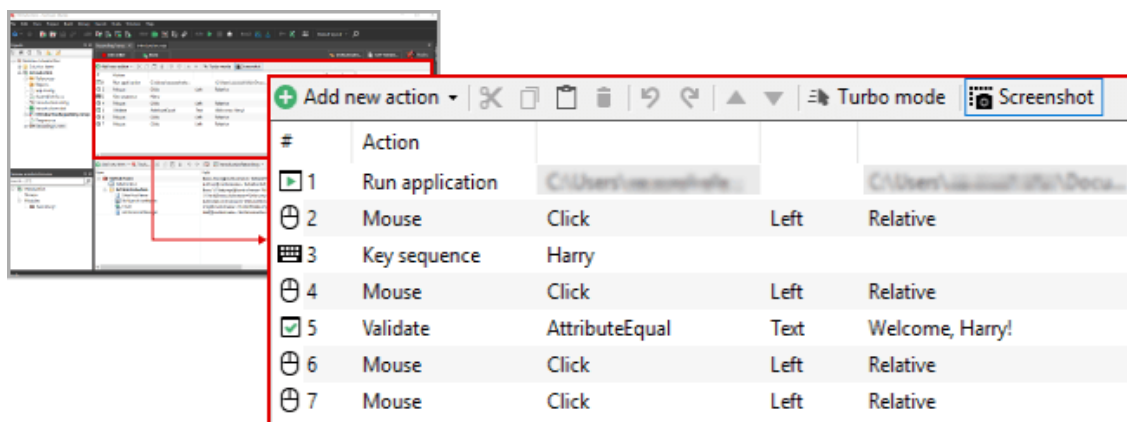
Actions are always listed chronologically, i.e. action #4 is executed before action #5.

## Actions and repository items

Most actions are performed on repository items, which in turn represent UI elements. Actions are managed in the actions table, while repository items are managed in the repository. They are therefore separate, but linked to each other. This chapter explains the relationship between actions and repository items.

## Actions and the actions table

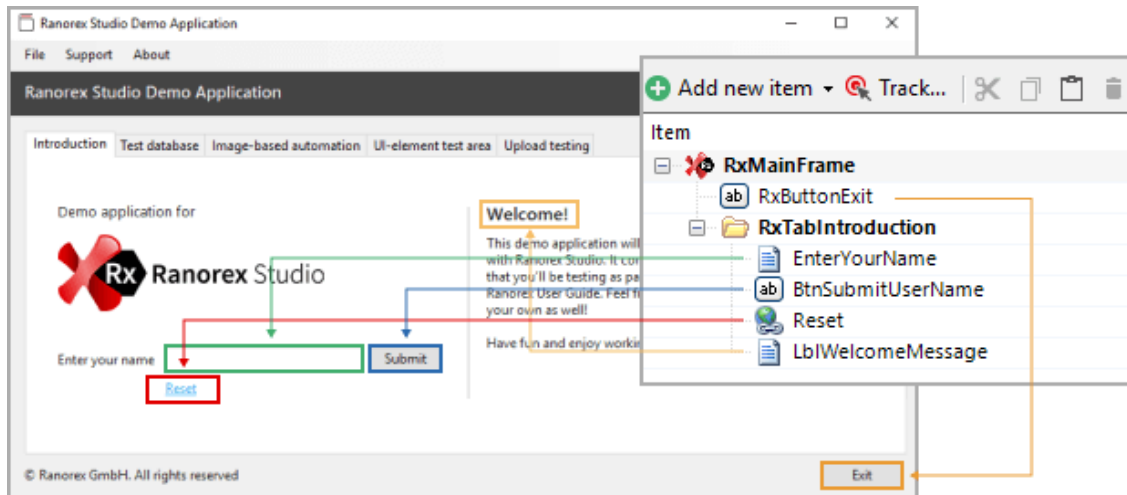
Actions are managed in the actions table of a recording module in the Recorder view.



## UI elements and repository items

Repository items are **representations** of UI elements. Ranorex uses repository items to **store** the structure of a UI and **access** it. This way, Ranorex can **perform actions** on repository items and, consequently, on UI elements. Repository items are stored and managed **separately** from actions in the repository in the Recorder view. Repositories are organized in a folder-like hierarchy.

For example, look at the welcome screen of the Ranorex demo app. Each UI element is represented by a repository item. Note how the items are organized in the repository.



## Further reading

The repository is described in Ranorex Studio fundamentals > → [Repository](#).

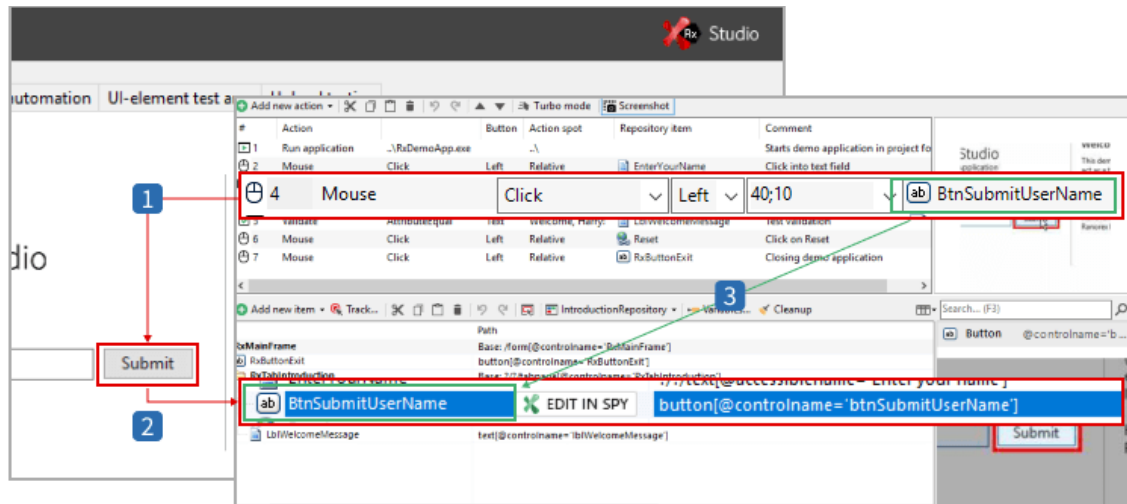
## Actions and repository items

Repository items by themselves don't do anything. It's only when they are **linked** to actions that they fulfill their function. While there are a few actions that can't be linked to repository items, such as the **Run application** action, the majority of actions can and in most cases should be linked to a repository item. Imagine that you had a **Key sequence** action that wasn't linked to a repository item—the key sequence would always be entered in whatever field was in focus at the moment. This would make your test unreliable.

Actions tell Ranorex **what** to do; repository items tell it **where in the UI** to do it.



The image below illustrates the link between a **Mouse click** action and a repository item representing a button.



## 1 Action

- Action #4 is a mouse click on a UI element, the **Submit** button.
- This action requires a link to the repository item that represents this button.

## 2 Repository item

- The repository item **BtnSubmitUserName** represents a UI element, the **Submit** button.
- Whenever an action should be carried out on this button, this repository item is invoked.

## 3 Link between an action and a repository item

- The link between the action and the UI element is established through the corresponding repository item.
- This is indicated in column 6 of the action, where the repository item is referenced.

## Manage actions

In this chapter, you'll learn to manually add actions, configure actions, and manage them.



### Screencast

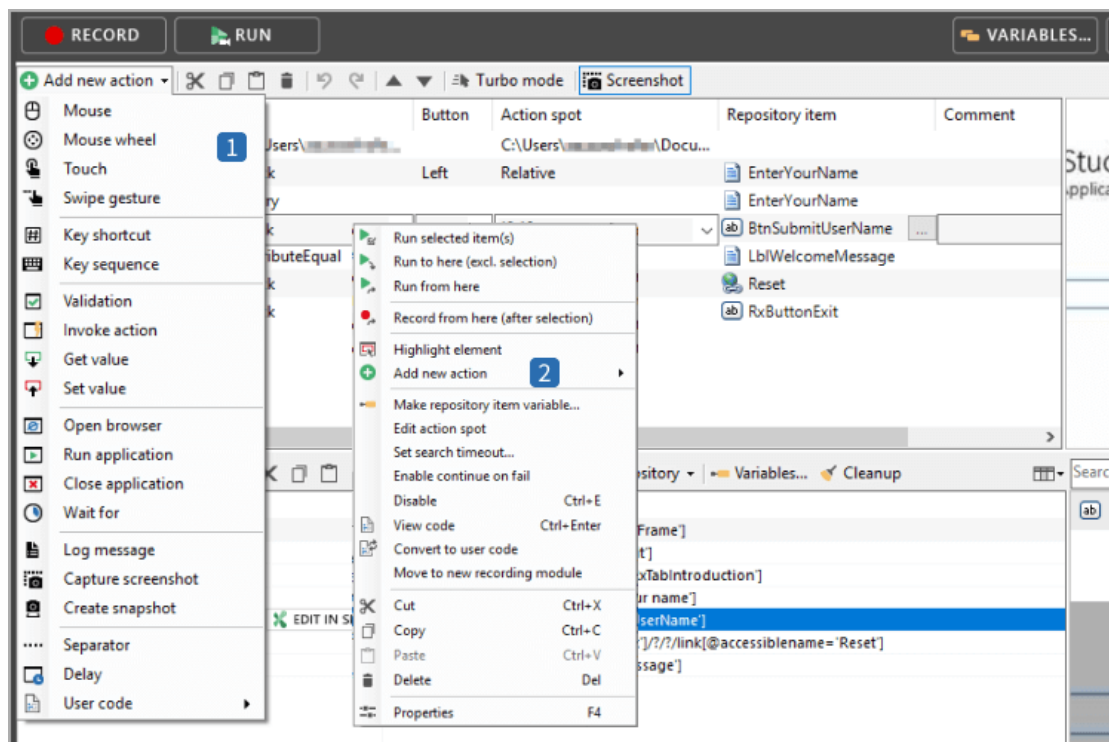
The screencast “Manage actions” walks you through information found in this chapter.

[Watch the screencast now](#)

## Add a new action

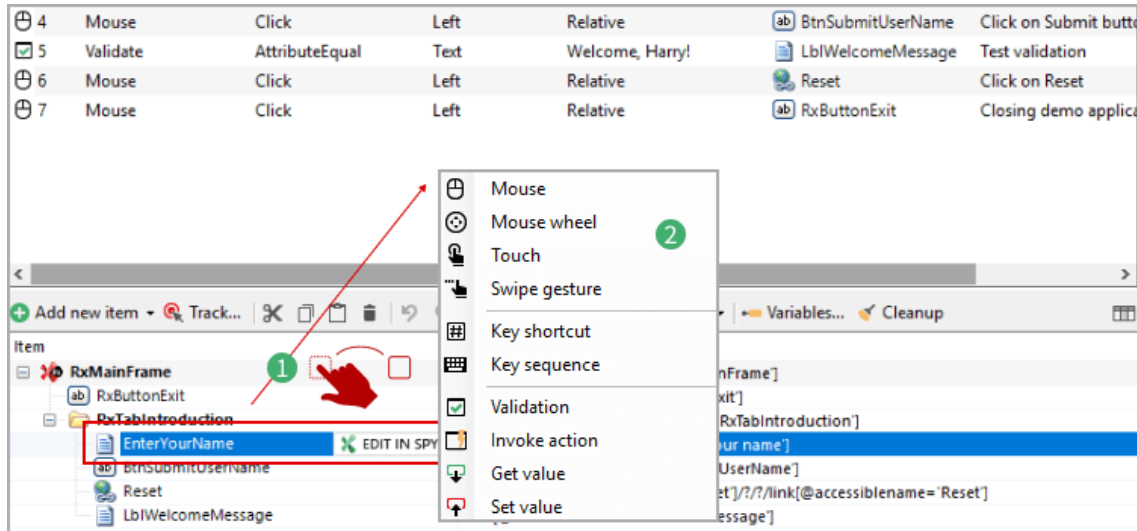
Many actions are generated automatically during recording. However, some actions must be added manually. There are two ways to add an action manually: by using a menu option or by dragging and dropping a repository item to the actions table.

### Add an action from a menu



- 1 Add new action menu in the actions table toolbar.
- 2 Add new action menu option in the actions table context menu.

### Drag and drop a repository item to the actions table



- 1 Select a repository item and **drop** it in the actions table.
- 2 A context menu opens with all possible actions for this repository item. **Click** the desired action.



#### Note

The new action is automatically linked to the repository item.

### Configure actions

The actions table is where you **configure** your actions. You can enter different settings through an action's **components**, its **context menu**, and its **properties**, some of which overlap.

## Action components

The action table contains seven different columns. These columns represent the five different components of an action. Some actions have fewer components, others more. The components are grouped as follows:

#	Action	Repository item	Action specifics	Comment
1	Run application	..\RxDemoApp.exe	..\	Starts demo application in project folder
2	Mouse	Click	Left Relative	Click into text field
3	Key sequence	Harry		Insertion of name
4	Mouse	Click	Left Relative	Click on Submit button
5	Validate	AttributeEqual	Text Welcome, Harry!	Test validation
6	Mouse	Click	Left Relative	Click on Reset
7	Mouse	Click	Left Relative	Closing demo application

### 1 Action symbol and sequence number

- Each action has its own symbol.
- The sequence number tells you the order in which actions are executed.

### 2 Action type

- The action type indicates what the action does or which part of a UI device it manipulates.

### 3 Action specifics

- The contents of columns 3–5 change depending on the action type.
- They define what the action does in detail: for example, the exact text string of a key sequence or the type of mouse click.

### 4 Repository item

- The repository item the action is performed on, i.e. the target of the action.
- Repository items represent UI elements in the AUT.
- Repository items are managed separately from actions, but they are linked to each other.



## Further reading

Repositories and repository items are introduced in Ranorex Studio fundamentals > [Repository](#).

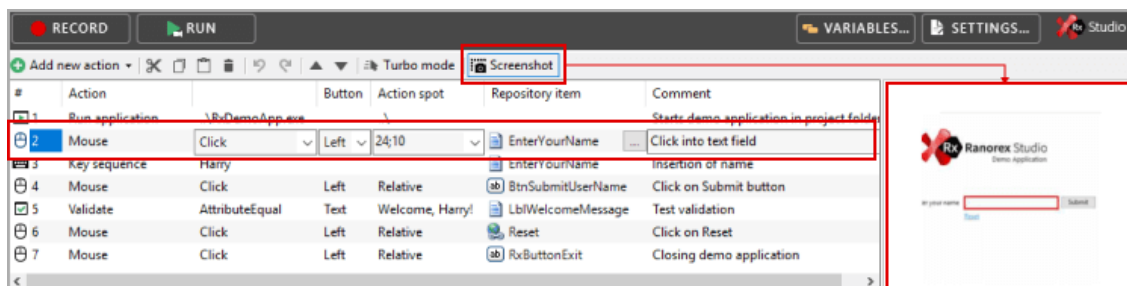
5

### Comment

- Here you can add comments to document what an action does. This has no effect on the action itself.

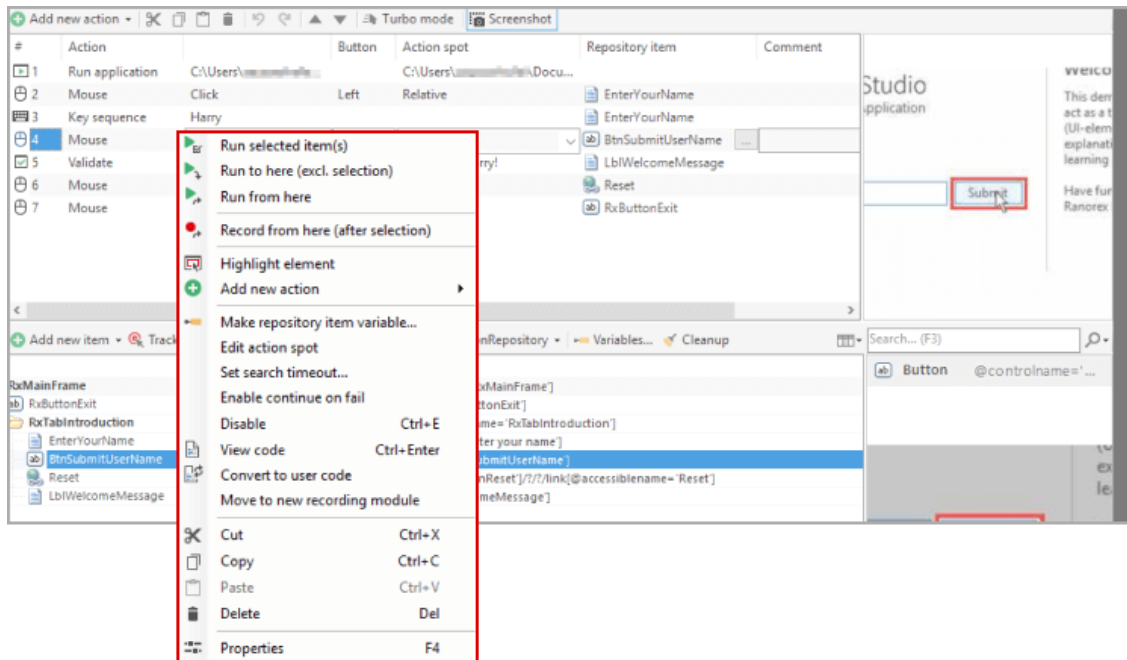
### Action screenshots

When you use the Ranorex Recorder to record an action that targets a UI element, a screenshot will be saved of the action. You can view this screenshot next to the action table. **Click Screenshot** to view or hide it.



### Context menu settings

An action's context menu gives you quick access to several settings, some of which are only available for certain actions. You can also select several actions at once to apply context-menu settings to them. Different settings are available when more than one action is selected.



## Note

The first four items are called **run options**. They are explained in Ranorex Studio Fundamentals > Ranorex Recorder > [Run and debug recordings](#).

## Single-action settings

**Highlight element:** Highlights the linked repository item in the AUT. The AUT must be running. Only available for actions linked to a repository item.

**Add new action:** Adds a new action after the current action.

**Make repository item variable...:** Makes the linked repository item a [variable](#). Only available for actions linked to a repository item.

**Set search timeout...:** Defines for how long Ranorex will attempt to find the linked repository item before the action fails. Only available for actions linked to a repository item.

**Enable/Disable continue on fail:** Sets whether Ranorex will progress to the next action or abort the test if this one fails. The default setting is **disabled**. If this option is enabled, the affected action will appear in *italics* in the actions table.

**Enable/Disable:** Sets whether the action will be run or skipped during test execution. The default setting is **enabled**. Disabled actions are grayed out in the actions table.

**View code:** Displays the underlying code of the action in the code editor. Useful for expert configurations and [→ user code actions](#).

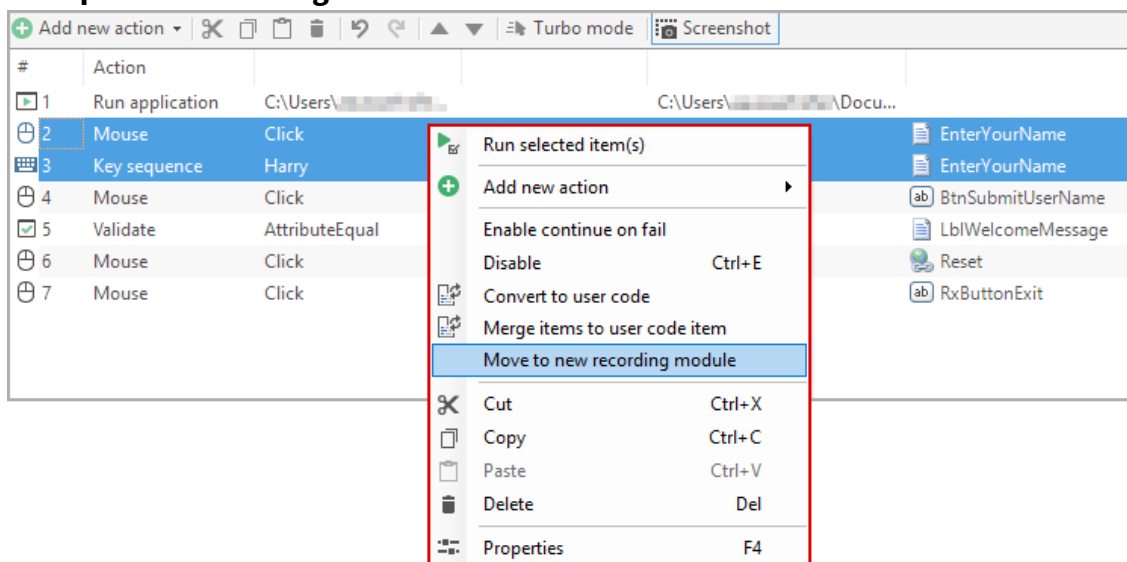
**Convert to user code:** Converts the action to a [→ user code action](#).

**Move to new recording module:** Creates a new recording module containing the selected action.

**Cut/Copy/Paste/Delete:** cuts, copies, pastes, or deletes the selected action.

**Properties:** Opens the action's properties, which are explained later in this chapter.

### Multiple-action settings



**Merge items to user code item:** Merges the code of the actions to a single [→ user code action](#). Useful for combining several actions for advanced scenarios.

**Merge selected keyboard items:** When two or more **Key sequence** actions or **Key shortcut** actions are linked to the same repository item, you can merge them to combine the key presses into a single action. This is useful for combining keyboard input that was unintentionally split up into several actions.

## Properties

Action properties are the various options with which you can control the behavior of actions. There are two groups of action properties: **Standard properties** and **action-specific properties**. Standard properties define behavior that applies to most action types and are explained below. Action-specific properties define behavior that applies only to particular action types.



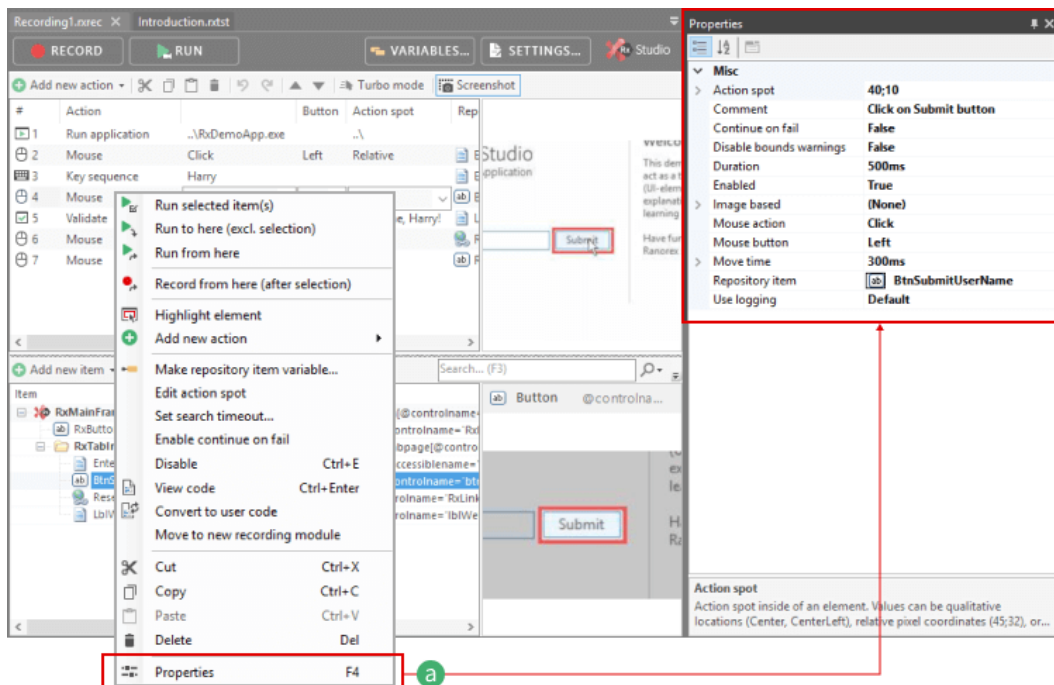
### Reference

Action-specific properties are described along with their associated actions in Ranorex Studio Fundamentals > Actions > [Action properties](#).

There are two ways to access an action's properties:

- a **Right-click** an action and **click Properties**.
- b **Click** an action and **press** **F4**.

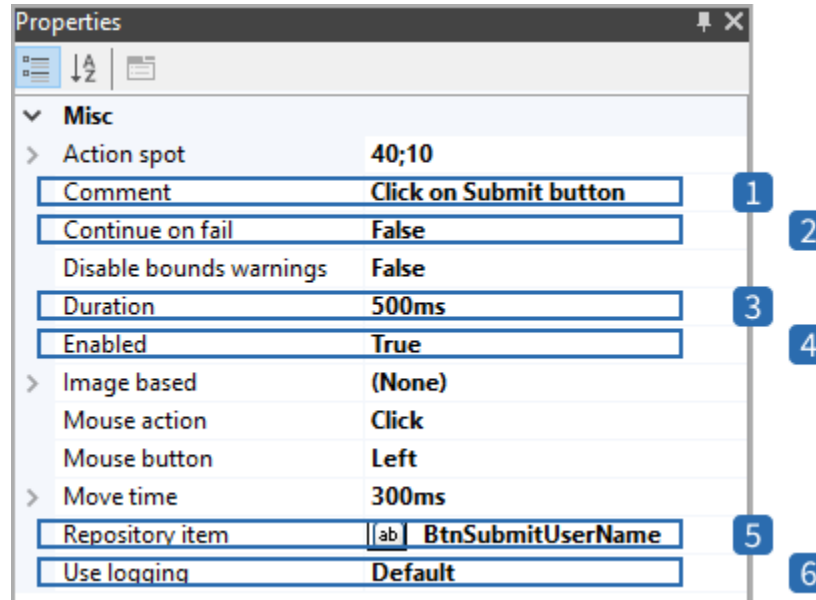
These also work for multiple actions. The properties panel will then display all properties shared among the selected actions.





## Standard properties

Here you'll find a list of all standard properties and details about them. They can be configured for almost all action types. **Default values** appear in **bold**.



1

### Comment

- Possible values: **empty** or text.
- Adds a comment. Can also be set directly in the action table.
- Available for all actions except the Separator action

2

### Continue on fail

- Possible values: **false** or true
- If set to false, test execution stops when this action fails.
- If set to true, test execution continues with the next action when this action fails.

3

### Duration

- Possible values: time in milliseconds (default varies by action)
- Defined action duration
- Available for all actions except the Report and Separator actions

4

### Enabled

- Possible values: **true** or false

- Enables or disables the action. You can also select an action and press **ctrl** + **E** to enable or disable it.

5

### Repository item

- Possible values: A reference to a repository item
- The repository item the action is linked to.
- Available for all actions that can be linked to repository items.

6

### Use logging

- Possible values: **Default**, true, or false
- Sets whether the action will be logged to the report or not.
- When set to **Default**, this value is inherited from the **Use item logging by default** setting in the → [Recorder defaults](#), where it is enabled by default. Therefore, all actions are logged by default.
- Available for all actions except the **Separator** action, which is always logged.

## All actions and their properties

In this chapter, you'll find a list of all actions with details on their components and properties. All components that can be configured in the actions table are also available in the properties of the action. We'll explain only the **action-specific properties** in this chapter. The **standard properties** are explained in → [Manage actions](#).

### Mouse

Summary: Performs a mouse action.

Type: Basic action

Variables possible: Yes

Linkable to repository: Yes

### Description

Presses a button on the mouse or moves the mouse to a defined position.

## Components

### Subactions

- Up/Down: Down presses and holds the specified mouse button, up releases it.
- Click/Double-click
- Move: Moves the mouse to the specified action spot.

### Button

Specifies which button is pressed. XButtons refer to the side buttons on a mouse.

### Action spot

Defines where on the screen the button press will be made or where the mouse will be moved. Can be a relative position, x;y pixel coordinates, or a variable.

### Action-specific properties

- Disable bound warnings: Disables the Out of bounds warning in the report. Default is False.
- Image based: Configure parameters for [image-based automation](#).
- Move time: How long a Mouse move action takes. Default is 300 ms. Can be bound to a variable.

## Mouse wheel

Summary: Performs a mouse wheel action

Type: Basic action

Variables possible: Yes

Linkable to repository: No

### Description

Performs a mouse wheel action in vertical or horizontal direction.

## Components

### Orientation

Horizontal, Vertical

## Delta

How far the wheel is moved vertically or horizontally as a positive or negative integer. Can be bound to a variable.

## Touch

Summary: Performs a touch screen action

Type: Basic action

Variables allowed: No

Linkable to repository: Yes, required

## Description

Performs different kinds of touch actions on a touchscreen device. Must be linked to a repository item.

## Components

### Touch type

- Touch
- DoubleTap
- LongTouch
- TouchStart/TouchMove/TouchEnd

### Action spot

Defines where on the screen the touch action will be made or, for the TouchMove type, where to move. Can be a relative position, x;y pixel coordinates, or a variable.

### Action-specific properties

- Disable bound warnings: Disables the Out of bounds warning in the report. Default is False.
- Image based: Configure parameters for [image-based automation](#).
- Long touch duration: How long the long touch will be held. Default is 1s. Can be bound to a variable.
- Move time: How long a touch move action takes. Default is 100 ms. Can be bound to a variable.

- Touch duration: How long the touch will be held. Default is 100 ms. Can be bound to a variable.

## Swipe gesture

Summary: Performs a touch screen swipe gesture

Type: Basic action

Variables allowed: Yes

Linkable to repository: Yes, required

## Description

Performs a swipe action on a specific repository item. You can configure the direction, distance, and duration of the swipe.

## Components

### Swipe direction

Direction of the swipe in degrees. Can be set to any value or bound to a variable.

### Distance

Distance the swipe covers in pixels. Can be set to any value or bound to a variable.

### Swipe duration

How long the swipe takes. Can be set to any value or bound to a variable.

## Action-specific properties

- Disable bound warnings: Disables the Out of bounds warning in the report. Default is False.
- Image based: Configure parameters for [image-based automation](#).
- Start location: Where the swipe starts. Default is Center. Various predefined values. Can also be an x;y pixel value. Can be bound to a variable.
- Steps: Number of times the swipe gesture is executed. Can be bound to a variable.

## Key shortcut

Summary: Performs a keyboard shortcut

Type: Basic action

Variables possible: Yes

Linkable to repository: Yes

## Description

Performs a keyboard shortcut using one or multiple keys.

## Components

### Key code

- Press: Presses and releases the keyboard shortcut.
- Up, Down: Down presses and holds, Up releases the keyboard shortcut.

### Shortcut

The keyboard shortcut itself. Click ... next to the field to open an assistant. You can also enter the shortcut directly. For example, to perform the copy shortcut, enter: `ctrl+c` and press Enter. You can also use the shortcut configurator in the action's properties. Can also be a variable.

### Action-specific properties

- Key data: The shortcut to perform. Contains a simple shortcut configurator with all possible combinations. Can be bound to a variable.
- Press time: How long the keyboard shortcut is held after being pressed. Can be bound to a variable.

## Key sequence

Summary: Enters a key sequence

Type: Basic action

Variables possible: Yes

Linkable to repository: Yes

## Description

Enters a key sequence of any length. You can mask the entered sequence in the action's properties.

## Components

### Sequence

The key sequence that will be entered. Click ... next to the field to open an assistant.

### Action-specific properties

- Mask sequence: Masks the entered sequence. Default is False.
- Press time: How long each key in the sequence is pressed. Can be bound to a variable.

## Mobile key press

Summary: Performs a mobile key press.

Type: Basic action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Presses a mobile action key, e.g. the Home or Back buttons.

## Components

### Key

The action key to press.

### Action-specific properties

- Image based: Configure parameters for [image-based automation](#).
- Key: The action key to press. Can be bound to a variable.

## Validation

Summary: Performs a validation.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Performs a validation on a repository item, i.e. checks whether expected and actual states match. Depending on the result, this action will log a success or failure to the report. Several different types of validation are available, each with their own properties.

The validation action is complex. The different validation types share a set of action-specific properties, but most have their own additional properties.

### Shared validation properties

Action: The type of validation. The rest of the properties change depending on this property.

Message: The message that will be logged to the report for the validation. Leave empty to use the default Ranorex message.

Report level on failure: The → [report level](#) the failure message will have.

Report level on success: The → [report level](#) the success message will have.

Report screenshot: Whether a screenshot will be included in the report message.

### Validation types and special properties

#### Exists

Checks whether a repository item exists. Logs a success if the item exists and a failure if it doesn't.



## **NotExists**

Checks whether a repository item does **not** exist. Logs a success if the item does not exist and a failure if it does.

## **AttributeEqual:**

Checks whether an attribute (column **Match name**) **equals** a defined value (column **Match value**) within a defined time (property **Grace period**). The available attributes change depending on the linked repository item. Similarly, the possible match values also change depending on the attribute. For example, the attribute Text only takes text strings, while the attribute Valid takes true/false values. Both match name and match value can be bound to variables.

## **Special properties**

Match name and Match value: explained above.

Grace period: Amount of time in seconds the action waits for an attribute value to be equal to the match value. The grace period starts after the action has found the referenced UI element. If the UI element stops existing during the grace period, the action starts looking for it again, but the grace period does not stop or restart.

## **AttributeNotEqual:**

Checks whether an attribute (column **Match name**) **does not equal** a defined value (column **Match value**) within a defined time (property **Grace period**). The available attributes change depending on the linked repository item. Similarly, the possible match values also change depending on the attribute. For example, the attribute Text only takes text strings, while the attribute Valid takes true/false values. Both match name and match value can be bound to variables.

## **Special properties**

Match name and Match value: explained above.

Grace period: Amount of time in seconds the action waits for an attribute value to be equal to the match value. The grace period starts after the action has found the referenced UI element. If the UI element stops existing during the grace period, the action starts looking for it again, but the grace period does not stop or restart.

### **AttributeRegEx:**

Checks whether an attribute (column **Match name**) matches a defined **regular expression** (column **Match value**) within a defined time (property **Grace period**). The available attributes change depending on the linked repository item. Both match name and match value can be bound to variables.

### **Special properties**

Match name and Match value: explained above.

Grace period: Amount of time in seconds the action waits for an attribute value to be equal to the match value. The grace period starts after the action has found the referenced UI element. If the UI element stops existing during the grace period, the action starts looking for it again, but the grace period does not stop or restart.

### **AttributeContains**

Checks whether an attribute (column **Match name**) **contains** a defined value (column **Match value**) within a defined time (property **Grace period**). The available attributes change depending on the linked repository item. Both match name and match value can be bound to variables.

Special properties

Match name and Match value: explained above.

Grace period: Amount of time in seconds the action waits for an attribute value to be equal to the match value. The grace period starts after the action has found the referenced UI element. If the UI element stops existing during the grace period, the action starts looking for it again, but the grace period does not stop or restart.

### **AttributeNotContains**

Checks whether an attribute (column **Match name**) **does not contain** a defined value (column **Match value**) within a defined time (property **Grace period**). The available attributes change depending on the linked repository item. Both match name and match value can be bound to variables.

Special properties

Match name and Match value: explained above.

Grace period: Amount of time in seconds the action waits for an attribute value to be equal to the match value. The grace period starts after the action has found the referenced UI element. If the UI element stops existing during the grace period, the action starts looking for it again, but the grace period does not stop or restart.

## ContainsImage

This is an → [image-based automation](#) validation.

When the validation executes, it creates a screenshot of the repository item's current state. Then it checks whether this new execution screenshot contains a previously defined validation screenshot/the image-find region defined in the validation screenshot (column **Screenshot name**). This validation type is more flexible than CompareImage, but becomes exponentially slower with increasing image sizes.

The validation screenshot and execution screenshot don't need to have the same dimensions, but the execution screenshot must be larger than the validation screenshot/the defined image-find region. Else, the validation will fail. Therefore, make sure the size of all relevant UI elements during test execution is the same or smaller than when you created the reference image.

**Click the ellipsis ...** in the **Screenshot name** column to open an → [image editor](#) where you can define the image-find region and also image-ignore regions.

**Image-ignore regions** for ContainsImage should be outside the image-find region. The validation will then ignore image features that look similar to/are the same as the image-find region, avoiding false positives.



### Attention

For ContainsImage, an overlap warning will appear if image-find and image-ignore regions overlap, as this will likely cause a test failure. Redefine the regions so the overlap is removed, unless the overlap is intentional and necessary for your test.

## Special properties:

Image based: Configure properties for → [image-based automation](#).

Report difference images: Defines whether a difference mask showing the differing pixels as well as the differential image is logged to the report.

Report expected and actual images: When the actual and expected images should be included in the report.

Report similarity: Defines whether the similarity of the two images is logged to the report.

## **CompareImage**

This is an [image-based automation](#) validation.

When the validation is executed, it creates a screenshot of the repository item's current state. Then it checks whether this new screenshot is the same as a previously defined validation screenshot/image-find region (column **Screenshot name**). This validation type is faster than ContainsImage, but less flexible.

The validation screenshot and execution screenshot must have the same dimensions. Else, the validation will fail. Therefore, make sure that the size of all relevant UI elements during test execution is the same or smaller than when you created the validation screenshot. If you defined an image-find region that is smaller than the validation screenshot, the action will automatically crop the validation screenshot and the execution screenshot to these dimensions during test execution.

**Click the ellipsis ...** in the **Screenshot name** column to open an [image editor](#) where you can define the image-find region and also image-ignore regions.

**Image-ignore regions** for CompareImage must be inside the image-find region. The validation will then ignore differences in these regions.

### **Special properties:**

Image based: Configure parameters for [image-based automation](#).

Report difference image: Defines whether a difference mask showing the differing pixels as well as the differential image is logged to the report.

Report expected and actual images: When the actual and expected images should be included in the report.

Report similarity: Defines whether the similarity of the two images is logged to the report.



## Further reading

The concept of test validation is explained in Ranorex Studio fundamentals > Test validation > → [Introduction](#).

## Invoke action

Summary: Invokes an action

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Directly invokes a particular action on the referenced repository item without any simulated UI interaction through mouse clicks, key presses, etc. Especially useful for accessing UI elements that are not immediately visible, such as items in lists; or drop-down menus or buttons in windows that are not in focus.

## Components

### Action name

The action that is invoked. Changes depending on the referenced repository item.

### Various arguments

Most actions don't require any arguments. Some do, however. Arguments appear in parentheses after the action name, e.g. InvokeMethod(Name). Define them in the actions table or the properties. They can be bound to variables.



## Further reading

For examples on using the Invoke action, see Ranorex Studio fundamentals > Actions > → [Invoke actions](#).

## Get value

Summary: Retrieves a value from a repository item attribute.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Retrieves a value from a repository item attribute and passes it to a variable. Depending on the assigned repository item, the available attributes change. The retrieved value can also be modified using RegEx before being passed to the variable.

A common scenario for using this action is retrieving a value that is the result of a particular interaction. You then pass this value to a variable that's used in a validation that checks whether the result of the interaction is correct.

## Components

#	Action	Name	Variable	Capture regex	Repository item
1	Get value	ControlText	\$WelcomeMessage		LblWelcomeMessage

- 1 The **attribute** whose value will be retrieved.
- 2 The **variable** the retrieved value will be passed to.
- 3 The **regular expression** according to which the value will be modified before it is passed to the variable.
- 4 The referenced **repository item**.

## Set value

Summary: Sets an attribute of a repository item to a defined value.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Sets a repository item attribute to a defined value. Depending on the assigned repository item, the available attributes change. The value can also be masked.

## Components

#	Action	Name	Value	Repository item
1	Set value	Text	Harry	EnterYourName

- 1 The **attribute** that will be set to a defined value.
- 2 The **value** the attribute will be set to. Can be bound to a variable.
- 3 The referenced **repository item**.

## Action-specific properties

- Mask value: Masks the value. Default is False.

## Open browser

Summary: Opens a browser and navigates to the specified URL.

Type: Smart action

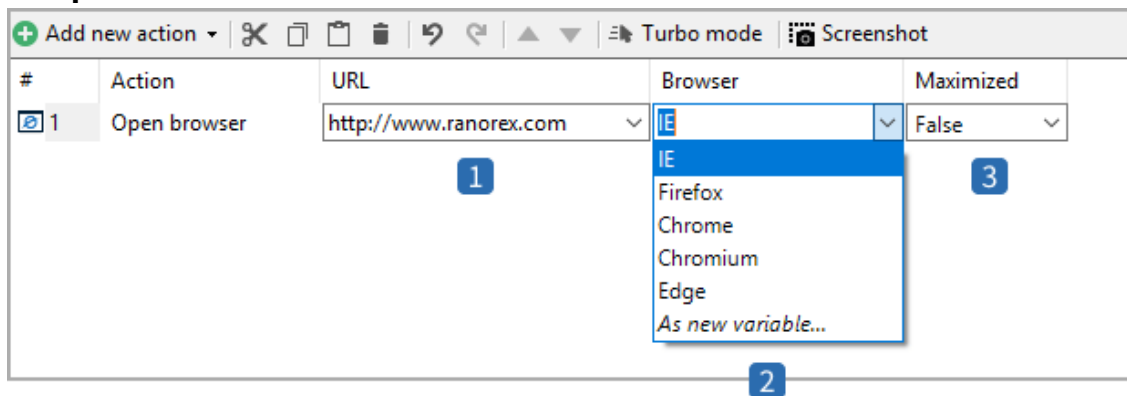
Variables possible: Yes

Linkable to repository: No

## Description

Starts a browser, instruments it, and opens the specified website.

## Components



- 1 The **URL** the browser will navigate to once started. Optional. Can be bound to a variable.
- 2 The **browser** to start. Can be bound to a variable.
- 3 Whether the browser window will be **maximized** or not.

## Action-specific properties

- Arguments: Define command line arguments to use when starting the browser, such as -ProfileManager to start Firefox with the profile manager opened. Can be bound to a variable.
- Clear cache: Whether the browser's cache will be cleared upon starting. Default is False. Can be bound to a variable.
- Clear cookies: Whether the browser's cookies will be cleared upon starting. Default is False. Can be bound to a variable.
- Incognito mode: Whether the browser will be started in incognito mode. Default is False. Can be bound to a variable.
- Instrument: Automatically instruments the browser. Particularly useful when deploying tests in runtime environments via XCOPY or Ranorex Agent. Default is True.
- Kill existing: Whether previously opened instances of the browser will be closed upon starting. Default is False. Can be bound to a variable.

## Run application

Summary: Runs an application from the given directory

Type: Smart action

Variables possible: Yes



Linkable to repository: No

## Description

Directly runs an application from the specified directory.

## Components

#	Action	File name	Arguments	Working directory
1	Run application	.\RxDemoApp.exe		.\

- 1** The **path** to the application. Can be relative if the application is located in the same folder as the test suite executable. Can be bound to a variable.
- 2** **Command line arguments** the application will be run with. Possible arguments depend on the application. Can be bound to a variable.
- 3** **Working directory:**
  - Optional information for the working directory of the application
  - Can be set as a constant value, or as variable
  - If not changed, similar to a directory of the application executable

## Action-specific properties

- Return value type/variable: These two fields are for working with the return value of the Run application action. The return value is always the process ID of the executed application. The type is therefore always System.Int32, i.e. an integer. Return value variable lets you select the variable to which you want to pass the process ID.
- Run flags: Allows you to set the NoElevation flag. This flag runs the application without Windows administrator privileges, e.g. as a normal user. Can be bound to a variable.
- Run maximized: Whether the application will be run maximized. Default is False.

## Run mobile app

Summary: Runs an application on a mobile device.

Type: Smart action

Variables possible: Yes

Linkable to repository: No

## Description

Runs an app on a mobile device.

## Components

### Endpoint

The endpoint (i.e. the mobile device) the app will be run on. Can be bound to a variable.

### Startup arguments

Command line arguments the app will be run with. Possible arguments depend on the app. Can be bound to a variable.

### Restart app

If the app is already running, restarts it. Can be bound to a variable.



### Further reading

Mobile testing is explained in detail starting with Web & mobile testing > Mobile testing > → [Introduction](#).

## Deploy Android app

Summary: Instruments an Android app and deploys it to a mobile device.

Type: Smart action

Variables possible: Yes

Linkable to repository: No

## Description

Instruments an Android app and deploys it to a specified mobile device.

## Components

### Endpoint

The endpoint (i.e. the mobile device) the app will be deployed to. Can be bound to a variable.

### File name

The path to the app to instrument and deploy. Can be bound to a variable.

### Action-specific properties

- Instrument APK: Whether the app will be instrumented before deployment. Default is True. Can be bound to a variable.
- Instrumentation options: Define various instrumentation options.
- Timeout: Maximum amount of time the instrumentation and deployment may take before the action fails. Can be bound to a variable.



### Further reading

Mobile testing is explained in detail starting with Web & mobile testing > Mobile testing > → [Introduction](#).

## Deploy iOS app

Summary: Instruments an iOS app and deploys it to a mobile device.

Type: Smart action

Variables possible: Yes

Linkable to repository: No

### Description

Instruments an iOS app and deploys it to a specified mobile device.

## Components

### Endpoint

The endpoint (i.e. the mobile device) the app will be deployed to. Can be bound to a variable.

### App archive

The path to the app archive to instrument and deploy. Can be bound to a variable.

### App ID

The app ID of the app to instrument and deploy. Can be bound to a variable.

### Action-specific properties

- Instrumentation settings: Define various instrumentation settings.



### Further reading

Mobile testing is explained in detail starting with Web & mobile testing > Mobile testing > → [Introduction](#).

## Set device orientation

Summary: Sets the orientation of a mobile device for a repository item.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

### Description

Sets the orientation of a mobile device for a specified repository item.

## Components

### Orientation

The orientation the device will be set to. Portrait and landscape orientations are available. Landscape left means the top of the device is rotated left. Vice-versa for Landscape right. Can be bound to a variable.



#### Further reading

Mobile testing is explained in detail starting with Web & mobile testing > Mobile testing > → [Introduction](#).

### Close application

Summary: Closes an application or website.

Type: Smart action

Variables possible: Yes

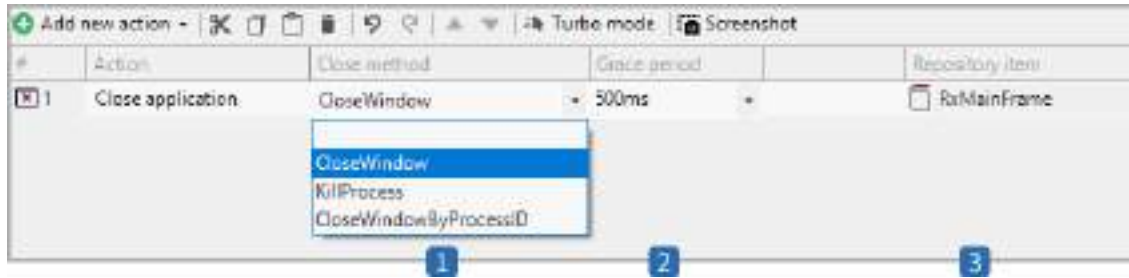
Linkable to repository: Yes, required

### Description

Closes an application or website.

This action can be used for closing applications and websites. If the 'Close Method' is set to 'CloseWindow' or 'CloseWindowByProcessID', an attempt is made to close the application. If the parameter 'Grace Period' is set to a value greater than 0 ms, the process will be killed after the 'Grace Period' if closing the application failed. If 'Close Method' is set to 'KillProcess' the application's process is killed immediately and the grace period is ignored.

## Components



- 1 The **close method**. **CloseWindow** and **CloseWindowByProcessID** attempt to close the window. If this fails, it will kill the process after the grace period. **KillProcess** ignores the grace period and immediately kills the process. **CloseWindowByProcessID** works with the ReturnValue property of the Run application action.
- 2 If normal closing of the application failed, the **grace period** is the time Ranorex will wait before killing the process. Can be bound to a variable.
- 3 The referenced **repository item** whose parent application will be closed.

## Wait for

Summary: Waits for a specified UI state to occur.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes, required

## Description

Waits until a defined state is reached within a specific time. Several different Wait-for types are available.

## Exists

Waits for the referenced repository item to **exist** within a specified timeout (column **Timeout**). The timeout can be bound to a variable.

## **NotExists**

Waits for the referenced repository item to **stop existing** within a specified timeout (column **Timeout**). The timeout can be bound to a variable.

## **AttributeEqual**

Waits for an attribute (column **Match name**) of the referenced repository item to equal a specified value (column **Match value**) within the specified timeout. Match name and match value can be bound to variables. Available attributes changed depending on the repository item. If the values don't match within the timeout, the action fails.

Configure the timeout in the action's properties (**Wait timeout**). It can be bound to a variable.

## **AttributeNotEqual**

Waits for an attribute (column **Match name**) of the referenced repository item to stop being equal to a specified value (column **Match value**) within the specified timeout. Match name and match value can be bound to variables. Available attributes changed depending on the repository item. If the values don't stop matching within the timeout, the action fails.

Configure the timeout in the action's properties (**Wait timeout**). It can be bound to a variable.

## **AttributeContains**

Waits for an attribute (column **Match name**) of the referenced repository item to contain a specified value (column **Match value**) within the specified timeout. Match name and match value can be bound to variables. Available attributes changed depending on the repository item. If the attribute doesn't contain the match value within the timeout, the action fails.

Configure the timeout in the action's properties (**Wait timeout**). It can be bound to a variable.

## **AttributeNotContains**

Waits for an attribute (column **Match name**) of the referenced repository item to stop containing a specified value (column **Match value**) within the specified timeout. Match name and match value can be bound to variables. Available attributes changed depending on the repository item. If the attribute doesn't stop containing the match value within the timeout, the action fails.

Configure the timeout in the action's properties (**Wait timeout**). It can be bound to a variable.

## Log message

Summary: Logs an item to the report.

Type: Smart action

Variables possible: Yes

Linkable to repository: Yes

## Description

Logs a message to the report, or captures a screenshot or Ranorex snapshot.

## Components

### Subaction

Log: Logs a message to the report.

Screenshot: Captures a screenshot and adds it to the report.

Snapshot: Captures a Ranorex snapshot and adds it to the report.

## Message

The message that will appear in the report. Also applies to screenshot/snapshot captures. Can be bound to a variable.

## Level

The [report level](#) at which the item will be logged.

## Action-specific properties

- Category: Define under which category the item will appear in the report. Default is User.

## Separator

Summary: Inserts a separation line at the current position in the actions table.

Type: Smart action



Variables possible: No

Linkable to repository: No

## Description

Inserts a separator line at the current position in the actions table. Use it to **visually** separate or group related actions. You can add a **header text** to describe the separator. The separator and its header text will also **appear in the report**. This action has no influence on test execution itself. It's **purely cosmetic**.



### Further reading

The separator action is also explained in Ranorex Studio fundamentals > Reporting > [Basic report characteristics & data](#).

## Delay

Summary: Delays execution of the next action.

Type: Smart action

Variables possible: Yes

Linkable to repository: No

## Description

Delays execution of the next action for a specified amount of time. In essence, this action **pauses test execution** for a specified time (the AUT will still run normally, however). The time value can be bound to a variable. The delay is not affected by turbo mode.

## User code

Summary: Adds a user code action.

Type: Smart action

Variables possible: Yes

Linkable epository:      Yes

## Description

Adds a user code action. You can add an existing user code action from the user code library or code one yourself.



### Further reading

User code actions are explained in Ranorex Studio fundamentals > Actions > [User code actions](#).

The user code library is explained in Ranorex Studio expert > User code library > [Introduction](#).

## Invoke actions

The Invoke action allows you to perform an action on a repository item **without any direct interaction** like mouse clicks or keyboard input. This is particularly useful for manipulating UI elements that **aren't visible**, such as windows that are out of focus or list items to which you need to scroll. In this chapter, you'll learn how to use the Invoke action through two examples.



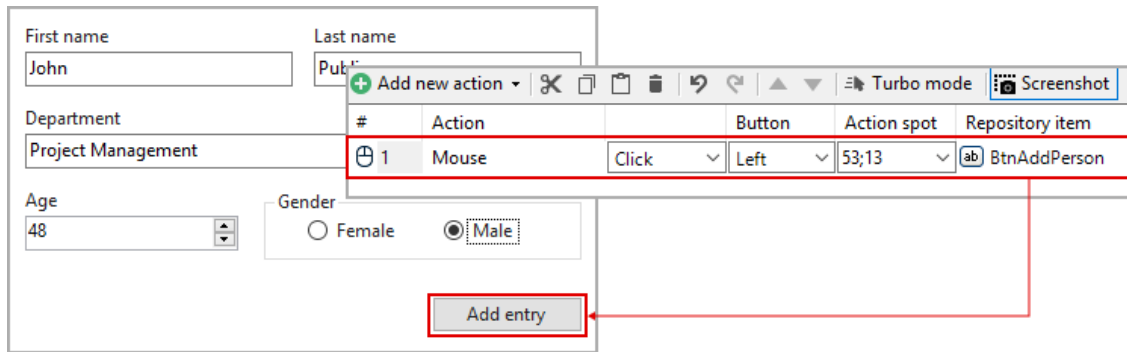
### Screencast

The screencast "Invoke action" walks you through information found in this chapter.

[Watch the screencast now](#)

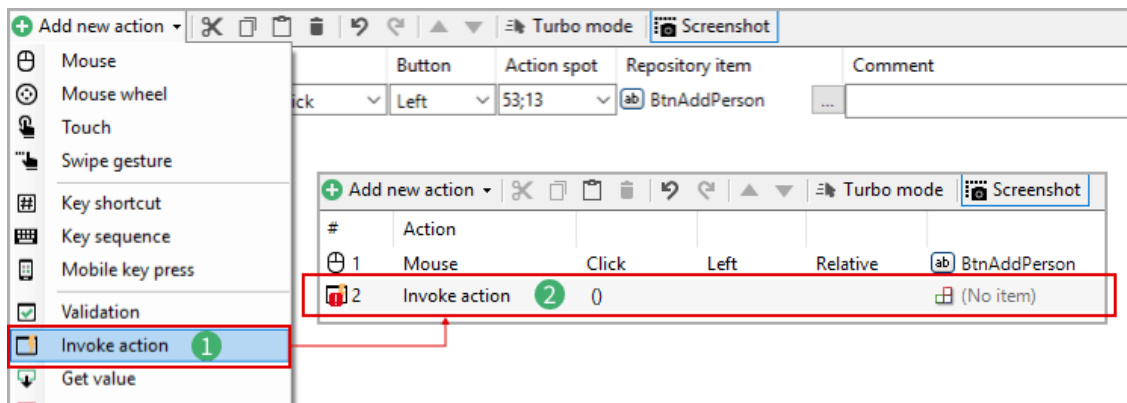
## Simple button click

In this example, you'll perform a simple button click with an Invoke action. Action #1 in the image below performs a mouse click on the **Add entry** button of the AUT.

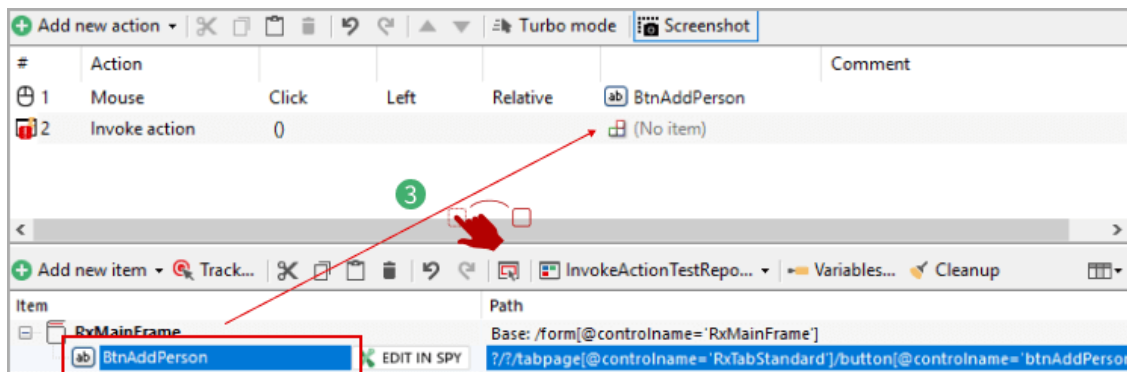


To replace it with an Invoke action that does the same:

- 1 Click **Add new action > Invoke action**.
- 2 An empty Invoke action appears in the actions table.



- 3 From the repository, **drag** the item **BtnAddPerson** that represents the **Add entry** button to the Invoke action.



- 4 In the Action name column of the Invoke action, select PerformClick().

+ Add new action   ✂   📄   🗑   ↺   ⬆   ⬇   ⚙ Turbo mode   📷 Screenshot					
#	Action	Action name			Repository item
1	Mouse	Click	Left	Relative	ab BtnAddPerson
2	Invoke action	PerformClick()	4		ab BtnAddPerson

- 5 Finally, **right-click** the regular Mouse click action and **click Disable** to disable it.

### Result(s):

- The click on the **Add entry** button is now executed directly without any mouse interaction.

### List item selection

Automating list item selection can be challenging because often, certain items won't immediately be visible. Using an Invoke action instead of a regular mouse interaction can often make your test more robust. The image below shows a standard list item selection using Mouse click actions.

The screenshot shows a test automation interface with a table of actions and a corresponding UI element. The table has columns for #, Action, and details. Action 1 is a Mouse Click (Left, Relative) on 'Open'. Action 2 is a Mouse Click (Left, Relative) on 'ProjectManagement'. The UI element is a 'Department' drop-down menu with a list of items: 'Project Management', 'Development Management', 'Marketing Office', 'Project Management', 'Sales', 'Support', and 'Testing'. A red box highlights the 'Open' button in the table, and a red arrow points to the drop-down arrow in the UI. A green box highlights the 'Project Management' item in the table, and a green arrow points to the selected item in the UI.

#	Action				
1	Mouse	Click	Left	Relative	ab Open
2	Mouse	Click	Left	Relative	ProjectManagement

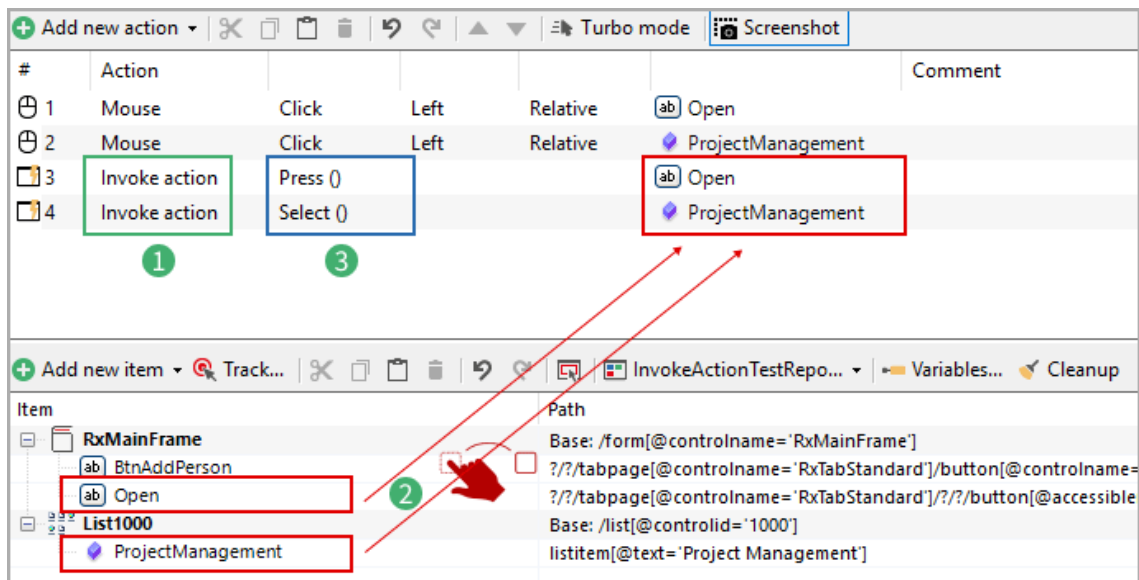
Department

- Project Management
- Development Management
- Marketing Office
- Project Management
- Sales
- Support
- Testing

- 1 Mouse click action that opens the **Department** drop-down.
- 2 Mouse click action that selects **Project Management** from the list items.

To perform this with an Invoke action instead:

- 1 Click **Add action > Invoke action** twice to add two empty Invoke actions.
- 2 From the repository, **drag** the item **Open** representing the drop-down button to the first Invoke action. **Repeat** for the second Invoke action with the item **ProjectManagement** representing the Project Management list item.
- 3 In the **Action name** column of the first Invoke action, select **Press ()**. **Repeat** for the second Invoke action with **Select ()**.
- 4 Finally, **right-click** the regular Mouse click actions and **click Disable** to disable them.



## Result(s):

- The list item selection is now executed directly without any mouse interaction.

## User code actions

User code actions extend the functionality of basic and smart actions. Examples include custom validations, accessing test-case related data and parameters, and extended reporting.

In this chapter, you'll learn how to **manage** user code actions in the **actions table**. Writing your own user code requires **programming knowledge** and is therefore **outside the scope** of this chapter. You can however make use of user code actions programmed by others through the **user code library**.



## Further reading

The user code library is an expert topic explained in Ranorex Studio expert > User code library > [Introduction](#).



## Screencast

The screencast “User code actions” walks you through information found in this chapter.

[Watch the screencast now](#)

## Locate the user code file

Each recording module is made up of three files. You can find these files in the projects view in Ranorex Studio or directly on your hard drive in the project folder. **Double-click** to open them.

The screenshot displays the Ranorex Studio interface. At the top, the 'Recorder' view shows a table of actions. Below it, the 'Projects' view shows the files for the recording module: Recording1.rrec, Recording1.cs, and Recording1.UserCode.cs. The 'Recording1.UserCode.cs' file is open in the code editor, showing the following code:

```
1 // This file was automatically generated by RANOREX.
2 // DO NOT MODIFY THIS FILE! It is regenerated by the designer.
3 // All your modifications will be lost!
4 // http://www.ranorex.com
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 namespace UserCodeAction
24 {
25     public partial class Recording1
26     {
27         /// <summary>
28         /// This method gets called right after the recording has been started.
29         /// It can be used to execute recording specific initialization code.
30         /// </summary>
31         private void Init()
32         {
33             // Your recording specific initialization code goes here.
34         }
35     }
36 }
```

1

Actions table with repository in the Recorder view.

2

**Code representation** of the recording module.

- Machine-generated code representation of the recording module.

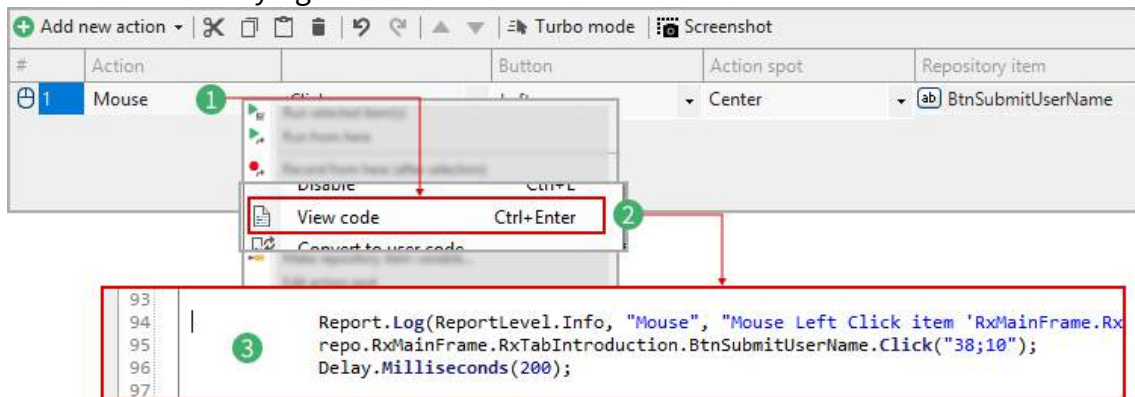
- Automatically updated by Ranorex. Any modifications you make are discarded.
- Read-only.

### 3 User code representation of the recording module.

- The user code file for the recording module and user code actions.
- Write all your user code to this file.
- User code methods from the user code library will automatically be added to this file.

## View the underlying action code

To view the underlying code of an action:



- 1 **Select** an action.
- 2 **Right-click** it and **click View Code**, or **press** `ctrl` + `Enter`.
- 3 The action's code appears



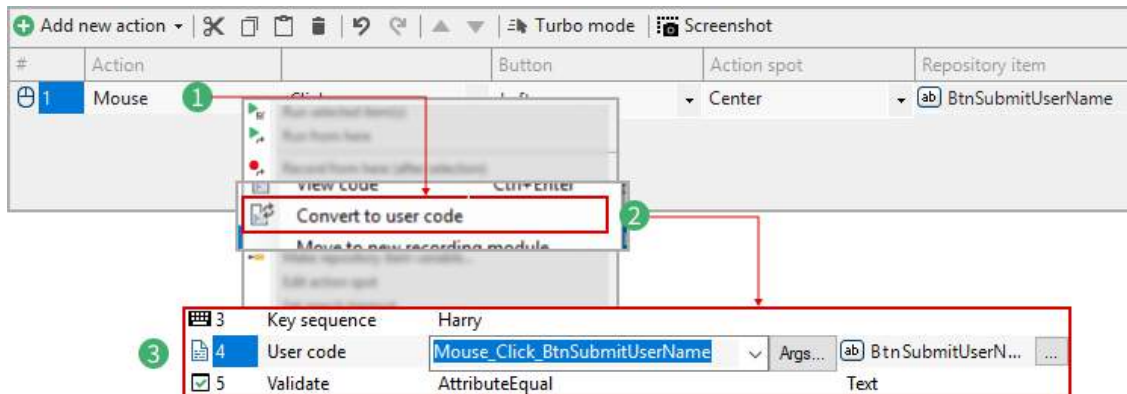
### Note

The file is locked, so you can't change it. It's updated automatically by Ranorex.

## Convert a standard action to user code

You can convert standard actions to User code actions.

To do so:



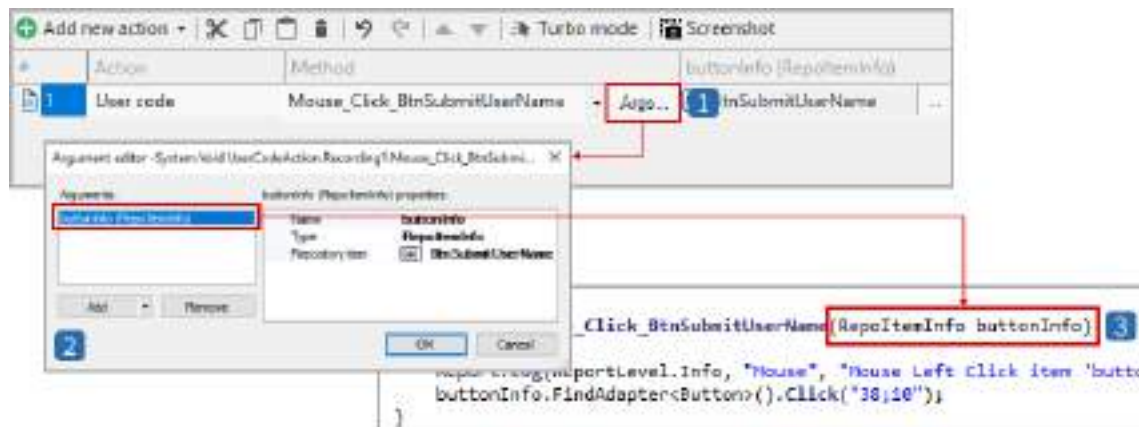
- 1 **Right-click** an action.
- 2 **Click Convert to user code.**
- 3 The action appears as a user code action with a default method name.
- 4 **Double-click** the user code action to view its code in the code editor.

```
UserCodeAction.Recording1
22
23 namespace UserCodeAction
24 {
25     public partial class Recording1
26     {
27         /// <summary>
28         /// This method gets called right after the recording has been started.
29         /// It can be used to execute recording specific initialization code.
30         /// </summary>
31         private void Init()
32         {
33             // Your recording specific initialization code goes here.
34         }
35
36         public void Mouse_Click_BtnSubmitUserName(RepoItemInfo buttonInfo)
37         {
38             Report.Log(ReportLevel.Info, "Mouse", "Mouse Left Click item 'buttonInfo' at 38;10.", buttonInfo);
39             buttonInfo.FindAdapter<Button>().Click("38;10");
40         }
41     }
42 }
```

## User code actions and arguments

You can configure arguments for user code actions in the argument editor or in code.





- 1 Button to open the argument editor of the code method.
- 2 The argument editor.
- 3 The argument in the action's code.

#### Argument name:

- Any name is possible.
- It is a best practice to use naming conventions, especially when working in a team.

#### Available argument type(s):

- **String** (value), can be bound to a variable
- **Boolean** (true, false), can be bound to a variable
- **DateTime** (value "YYYY/DD/MM"), can be bound to a variable
- **Duration** (value in ms), can be bound to a variable
- **Int32** (integer value), can be bound to a variable
- **Location** (values center, middle, ...), can be bound to a variable
- **Point** (coordinate values), can be bound to a variable
- **Rectangle** (dimension value), can be bound to a variable
- **TimeSpan** (time format), can be bound to a variable
- **Double** (double value), can be bound to a variable
- **Adapter** (Ranorex adapter), linked to a repository item
- **RepoItemInfo** (repository item), linked to a repository item

#### Important to know:

- Method parameters can be used within the method.
- Use the argument types 'Adapter' or 'RepoItemInfo' to pass a reference to a repository item into an action method.

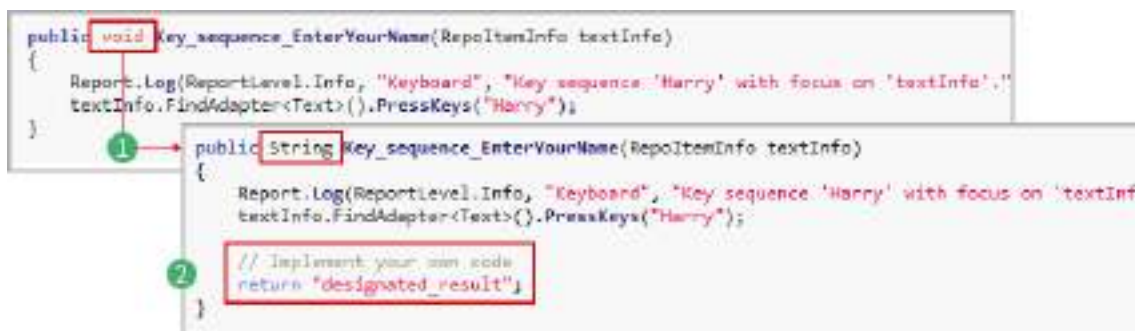
- Using repository items as arguments for user code actions enables a variety of possibilities such as providing a framework of smart test actions, defining generic technology-independent get/set value actions, combining several related actions to one user code action, implementing complex validations, and many more.

## User code return values

User code methods do not return a value by default. However, you can configure specify one of the available data types as method return value.

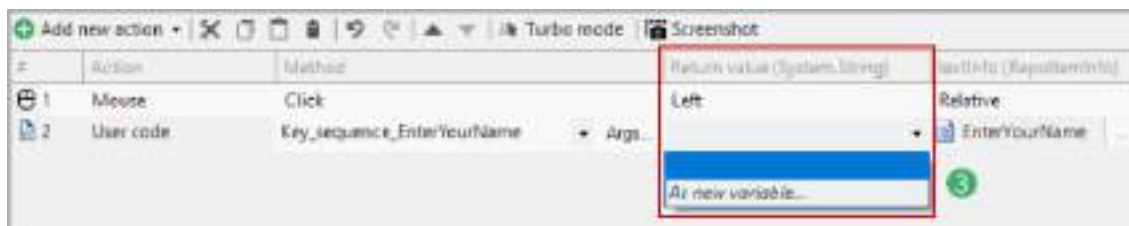
To do so:

- 1 Replace** the default `void` return value with your desired data type.
- 2 Implement** your individual user code and **add** a line to return the specified value.



The image shows two code snippets. The first snippet shows a method `key_sequence_EnterYourName` with a `void` return type. A red box highlights the `void` keyword, and a green circle with the number 1 points to it. The second snippet shows the same method with a `String` return type. A red box highlights the `String` keyword, and a green circle with the number 2 points to it. Below the `String` declaration, there is a comment `// Implement your own code` and a line `return "designated_result";` enclosed in a red box.

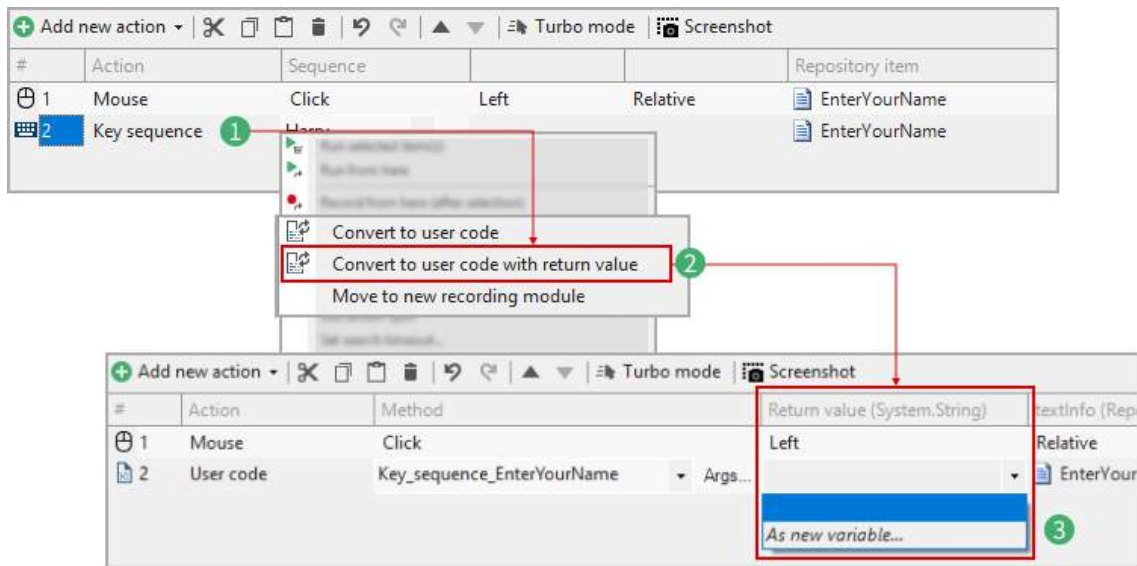
- 3** This causes a **Return value** column to appear in the action table. It contains the **return value** of the user code method and allows you to make it variable.



**Alternatively**, you can use the context menu to create a user code method with return value:

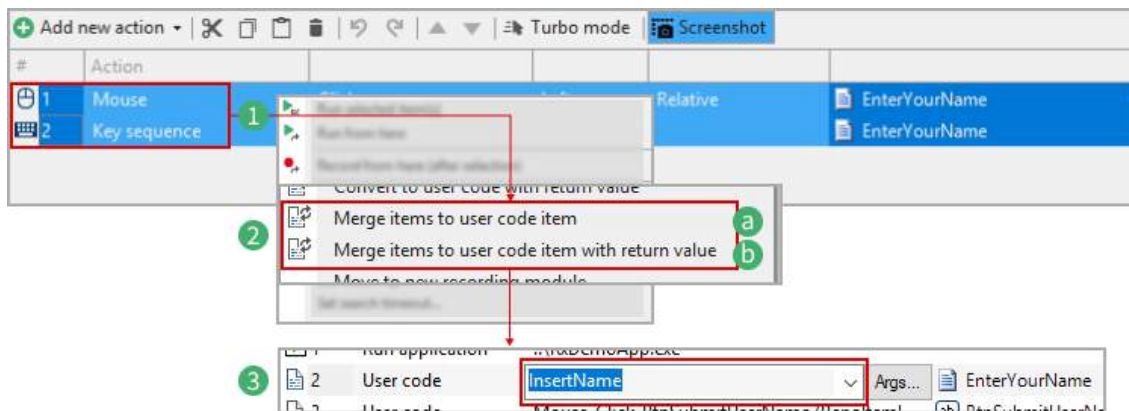
- 1 Right-click** an action.
- 2 Click Convert to user code with return value.**

- 3 The new action appears in the action table, including a **Return value** column.



## Merge actions into a user code action

Two or more actions can be merged into a single user code action, as shown below:

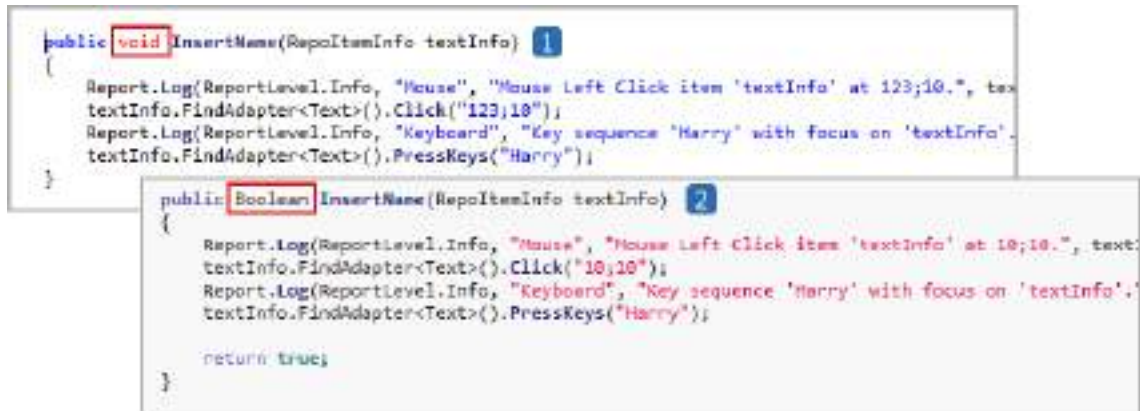


- 1 **Select** the actions you want to merge.
- 2 **Right-click** them to open the context menu.
  - a **Click Merge items to user code item** to create a default user code method without return value.
  - b **Click Merge items to user code item with return value** to create a user code method with a return value.

**3** Name the new user code action and press **Enter** .

### Result(s):

The code of the actions appears in the code of the new user code action.



```
public void InsertName(RepoItemInfo textInfo) 1
{
    Report.Log(ReportLevel.Info, "Mouse", "Mouse Left Click item 'textInfo' at 123;10.", textInfo.FindAdapter<Text>().Click("123;10"));
    Report.Log(ReportLevel.Info, "Keyboard", "Key sequence 'Harry' with focus on 'textInfo'.", textInfo.FindAdapter<Text>().PressKeys("Harry"));
}

public Boolean InsertName(RepoItemInfo textInfo) 2
{
    Report.Log(ReportLevel.Info, "Mouse", "Mouse Left Click item 'textInfo' at 10;10.", textInfo.FindAdapter<Text>().Click("10;10"));
    Report.Log(ReportLevel.Info, "Keyboard", "Key sequence 'Harry' with focus on 'textInfo'.", textInfo.FindAdapter<Text>().PressKeys("Harry"));

    return true;
}
```

**1** Default merged user code method without return value

**2** Merged user code method with Boolean return value.

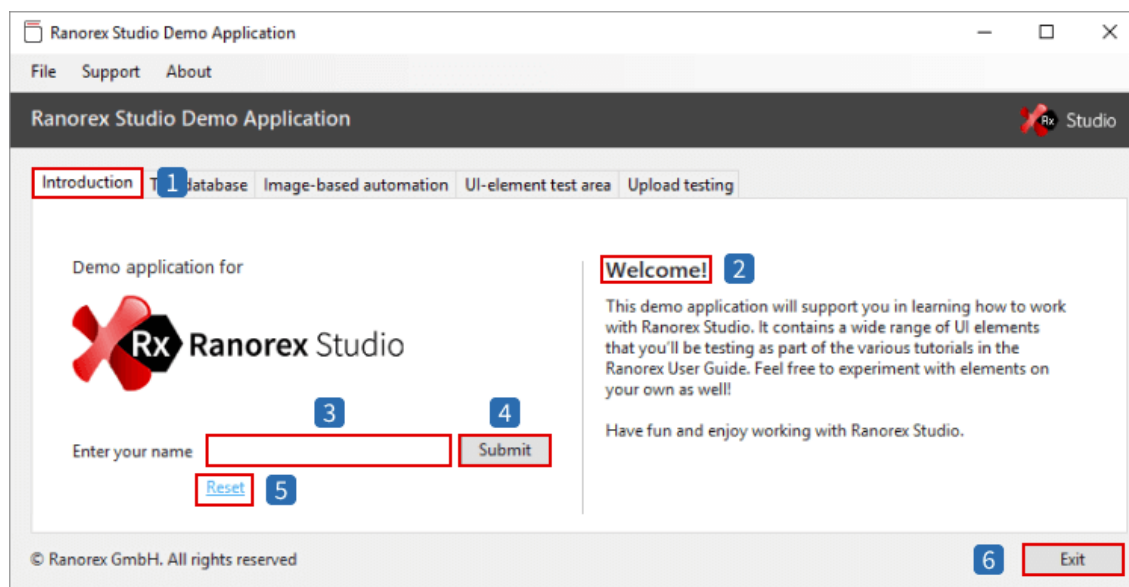
# Repository

A repository contains representations of user interface (UI) elements used in a test, called “repository items.” Repository items are organized in a tree-like structure. Each item has a RanoreXPath that uniquely identifies it and allows Ranorex to link the item with the corresponding UI element of the AUT.

In this chapter, you’ll learn how to create, manage and structure repository items. You will also learn how to use multiple repositories and about special use cases such as embedding repositories.

## UI elements

Ranorex Studio tests consist of a series of interactions with elements in an application’s user interface. UI elements include buttons, tabs, drop-down menus, text fields, and so on. Ranorex Studio represents UI elements as repository items in order to work with them.



**1** **Introduction** register tab

**2** **Welcome** text label

**3** Text input field

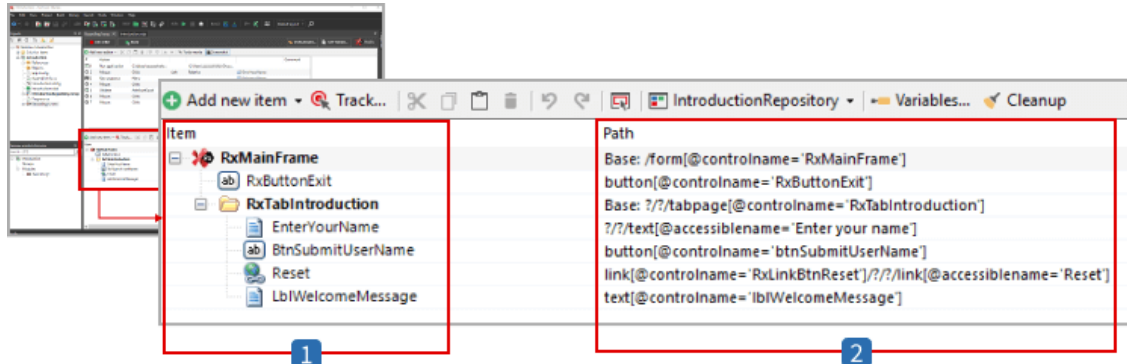
**4** **Submit** button

**5** **Reset** link

**6** **Exit** button

## The repository and repository items

Repository items are stored and managed in the repository in a tree-like structure. To access the repository, open the repository file from the projects view. The repository file also opens automatically when you open a recording module. By default, the repository appears in the lower section of the Ranorex Studio work area.



### 1 Repository items in tree-like structure

- Each repository item has a name. By default, this name is inherited from what the UI element is called in the internal structure of the AUT.
- You can rename a repository item as desired.
- A repository item's position in the tree-like structure (i.e. which folder it's in) suggests where the corresponding UI element is located in the AUT.

### 2 Paths of repository items

- Each repository item is defined by its path.
- The path is derived from the physical position of the UI element in the AUT.
- The path follows a special syntax, the RanoreXPath.

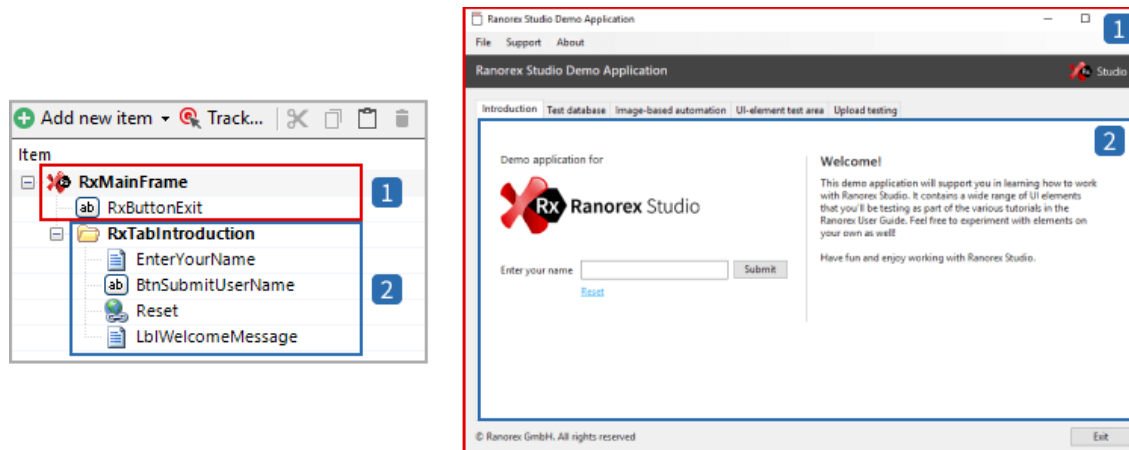


## Further reading

The RanoreXPath is described in Ranorex Studio advanced > [RanoreXPath](#).

## Repository structure

Repositories are organized in a tree-like structure that follows the internal structure of the AUT's UI. UI elements that contain other UI elements are represented as folders in the repository, with app folders acting as top-level elements and rooted folders as children.



1

### RxMainFrame

- **RxMainFrame** is an app folder. It represents the complete program window of the application that contains all other UI elements.
- In the example, its only direct child item that doesn't contain any other items is the **Exit** button in the lower right corner.

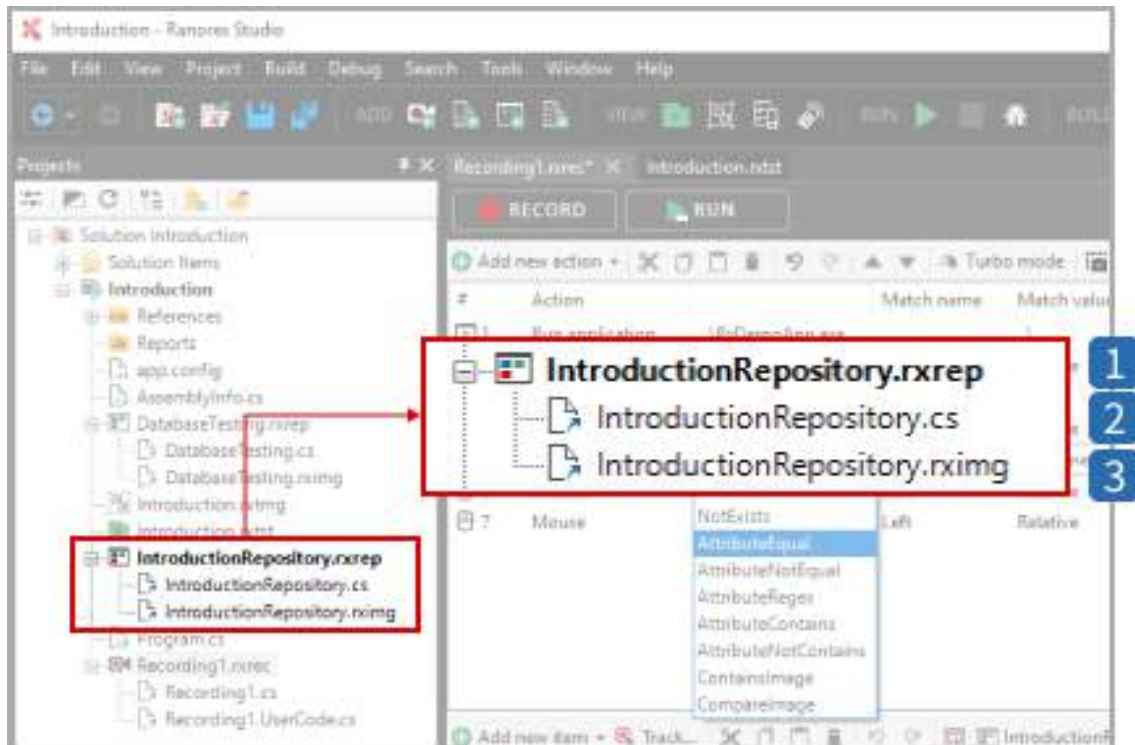
2

### RxTabIntroduction

- **RxTabIntroduction** is a rooted folder. It represents the screen area of the **Introduction** tab.
- In the example, it is a direct child of **RxMainFrame** and contains all UI elements in the **Introduction** tab.

## The repository file

The contents of the repository are stored in a repository file. As with other Ranorex files, you can access it from the projects view or in the project's folder in Windows. Each repository file also has two subfiles.



### 1 Repository file

- The repository file has the file ending **.rxrep**, which stands for **Ranorex repository**.
- The default file name is a combination of the parent test solution (i.e. **Introduction**) and **Repository**. You can rename the file to your liking.

### 2 Repository code representation






- The file with the ending **.cs** is the code representation of the repository in C#.

### 3 Image hosting file

- The file with the ending **.rximng** hosts all screenshots linked to repository items.

All of these files are also available in the respective project's folder in Windows.



 Introduction.txtst	7/2/2018 11:08 AM	Ranorex Test Suite	2 KB
 IntroductionRepository.cs	7/2/2018 11:08 AM	Visual C# Source f	11 KB
 IntroductionRepository.rximg	4/13/2018 6:35 AM	RXIMG File	41 KB
 IntroductionRepository.rxrep	7/2/2018 11:08 AM	Ranorex Repository	5 KB
 Program.cs	2/9/2018 12:40 PM	Visual C# Source f	2 KB

## Add a repository

If your project doesn't contain a repository, you can add one. You can also add [multiple repositories](#) to a project.



- 1 In the Studio toolbar, **click** the **Add repository** button.
- 2 A dialog opens with the repository template preselected.
- 3 **Name** the repository.
- 4 **Click Create.**

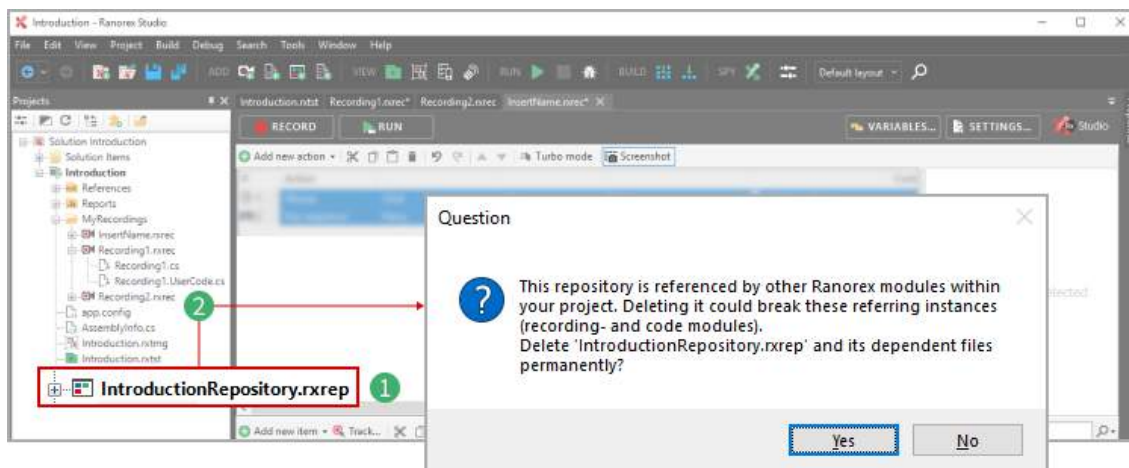
## Delete a repository

- 1 In the projects view, **select** the main repository file (.rxrep).
- 2 **Press DEL.**
  - If the repository doesn't contain any items that are linked to actions, it is simply deleted.
  - If at least one repository item is linked to an action, a dialog informs you about the consequences and you must confirm the deletion.



## Attention

Be careful when deleting repositories. Ranorex Studio can only locate references for repository items in the **currently open solution**. If you use the repository in another solution, tests that use its repository items will **break**.



## Repository items and actions



### Reference

The link between actions and repository items is explained in:

Ranorex Studio fundamentals > Actions > [Actions and repository items](#)

## Create repository items

In this chapter, you'll learn about the various methods for creating repository items and adding them to your repository.



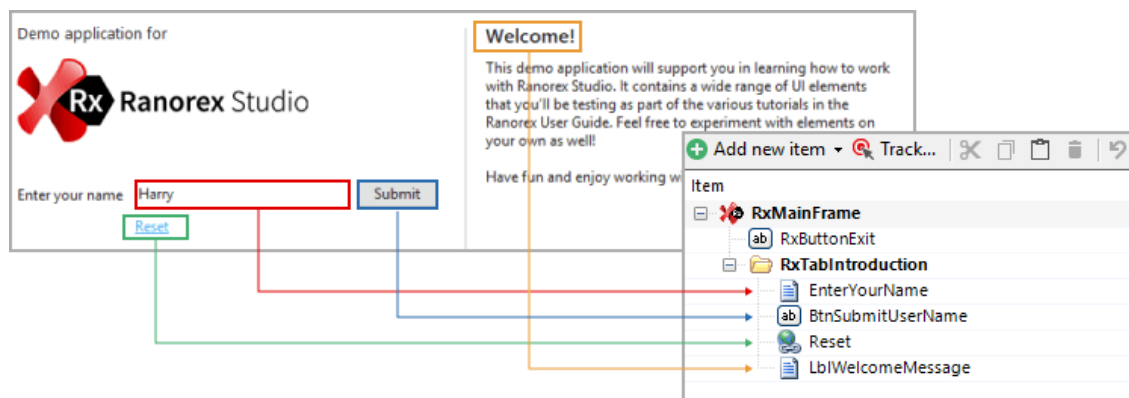
### Screencast

The screencast “Create repository items” walks you through information found in this chapter.

[Watch the screencast now](#)

## Automatic creation during a recording

When you're recording a test with Ranorex Recorder and you perform an action on a UI element, the corresponding repository element is created and added to your repository automatically.



## Create a repository item with the Track button

The Track button is available in the repository view and in Ranorex Spy. This button lets you track UI elements manually and create repository items. This is particularly useful when you need to update the definition of a UI element that has changed.



## Reference

---

Using the Track button is explained in detail in

Ranorex Studio advanced > Tracking UI-elements > → [Track button](#)

## Create a repository item with instant tracking

Instant tracking is another way of creating repository items manually. It's especially helpful for manually identifying UI elements that aren't easily accessible, like items in a drop-down menu. To activate instant tracking, mouse over the desired UI element and press the shortcut

**Ctrl** + **WIN** .



## Reference

---

Using instant tracking is explained in detail in

Ranorex Studio advanced > Tracking UI-elements > → [Instant tracking](#).

## Create a repository item with Ranorex Spy

You can also add repository items manually from Ranorex Spy, a dedicated tool for mapping and identifying UI elements.



## Further Reading

---

Ranorex Spy is introduced in Ranorex Studio advanced > → [Ranorex Spy](#).

## Create a repository item manually in the repository view

Finally, you can create repository items manually in the repository view itself.

To do so:

- 1 **Right-click** an app folder, a rooted folder, or a simple folder.
- 2 **Click Add new item > Item.**
- 3 Manually **specify** the item's path and **name** it.



## Reference

The RanoreXPath used to identify repository items is explained in

Ranorex Studio advanced > RanoreXPath > [Introduction](#)

## Manage repository items

Once they're part of your repository, you have several ways to edit and manage repository items can be edited and managed. This chapter describes the options available.



## Screencast

The screencast “Manage repository items” walks you through information found in this chapter.

[Watch the screencast now](#)

## Rename repository items

Ranorex Studio assigns repository items names automatically based on what the corresponding UI elements are called within the tracked application. Sometimes, these names can be cryptic or too generic, making it hard to navigate the repository. You can rename repository items to make it clearer what they do.



- 1 Select the repository item and press **F2**.
- 2 Rename the repository item and press **Enter**.
- 3 The name of the repository item has been changed.

## Copy/move/delete repository items

Repository items can be copied, moved, and deleted as usual. However, moving and deleting can cause issues if:

- you move repository items across logical boundaries in the repository.
- you delete repository items that are linked to one or more actions.

## Moving across logical boundaries

Ranorex organizes repository items in logical groups called folders. A folder always contains references to UI elements which share the same base position in the UI. This concept is explained in detail in the chapter [Structure repository items](#).

When you move a repository item from one folder to one with a different base path, you also alter the repository item's base path. As a result, Ranorex likely **won't be able to find the UI element** anymore and any actions linked to this repository item **will fail** during test execution. Ranorex will ask you for confirmation when you attempt to move an item in such a way.



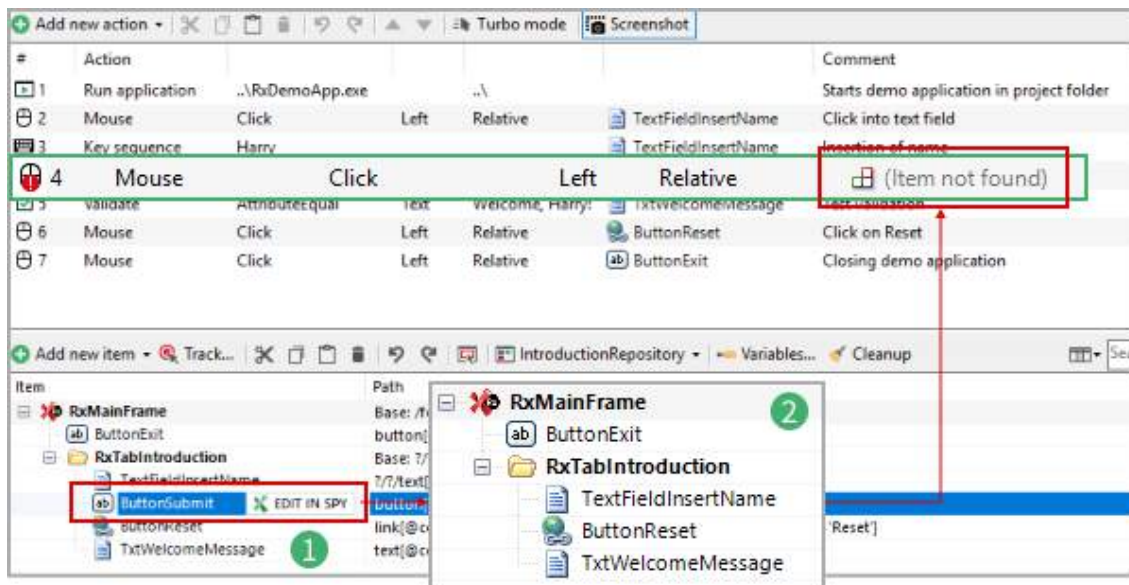
- 1 When you move a repository item from one folder to another with a different base path...
- 2 ...Ranorex will inform you about the consequences and ask you for confirmation.

### Note

This only **applies to app folders and rooted folders**. Simple folders are not affected because they don't have a base path.

## Deleting a linked repository item

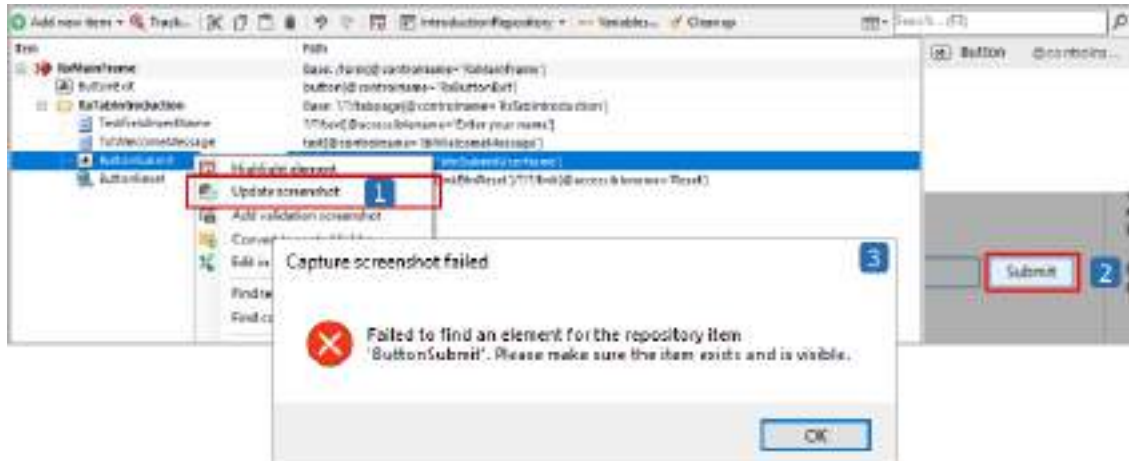
When you delete a repository item that's linked to one or more actions, the **actions will stop working**. If you want to **remove unused repository items**, use the **Cleanup** function instead



- 1 **Select** a repository item and **delete** it.
- 2 The repository item is removed from actions it was linked to. The actions won't work anymore.

## Update screenshot

When a repository item is created, a screenshot of the corresponding UI element is also saved. It can be viewed in the repository. When the UI element changes in appearance, you may need to update it. The option **Update screenshot** allows you to do this quickly without having to retrack the UI element.



### 1 Update screenshot option

- **Right-click** the repository item whose screenshot you want to update
- **Click Update screenshot**

### 2 The screenshot

### 3 Failure message

- The screenshot can only be updated if the path for the UI element is still valid and the containing application is running, with the UI element visible. If this isn't the case, an error message is displayed.

## Highlight element

Sometimes it can be hard to determine exactly which UI element a repository item references. When you click **Highlight element**, Ranorex Studio will highlight the UI element in the containing application. Of course, this will only work if Ranorex Studio can find the element, i.e. if the containing application is running and the path to the element is correct.

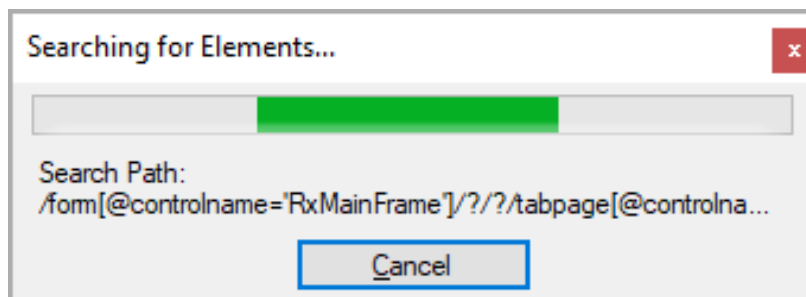




### 1 Highlight element

- **Right-click** the desired repository item.
- **Click Highlight element.**

2 The **UI element is highlighted** in the application for a few seconds with a blinking red frame.

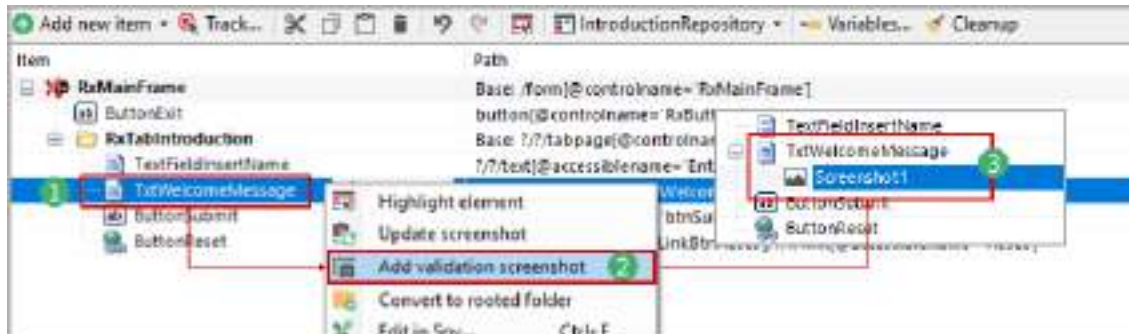


### 1 Search for the corresponding UI element

- Sometimes Ranorex may take a while to find the corresponding UI element.
- If the path is incorrect or the containing application hasn't been started, Ranorex won't be able to find the element.

## Add validation screenshot

This option is useful for [image validations](#). Normally, when you record an image validation, the validation screenshot will be created automatically. However, you can also add the validation screenshot manually. You can also add several validation screenshots to a single repository item. This is useful when image-validating the same UI element in different states (for example, a traffic light status field). The option **Add validation** screenshot lets you do this.



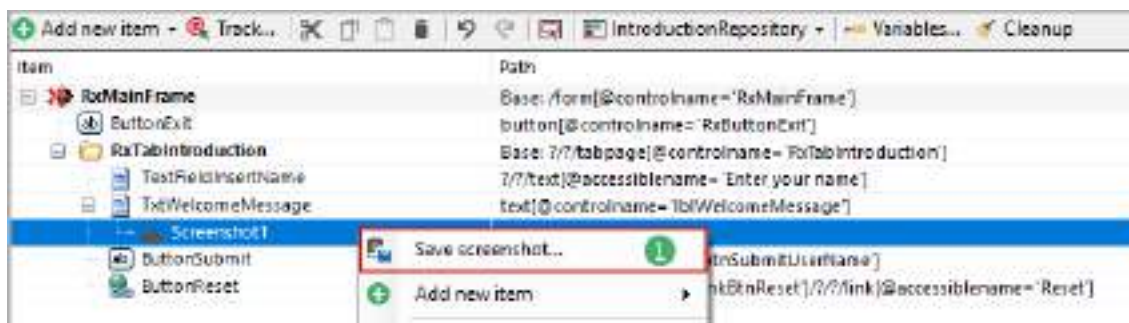
- 1 **Right-click** the desired repository item.
- 2 **Click Add validation screenshot.**
- 3 A screenshot of the current state of the UI element is added to the corresponding repository item.

### **Note**

This option only works if the AUT is opened and the respective UI element is visible.

## **Save screenshot**

Once a validation screenshot has been added to a repository item, you can save it to any destination on your system. You can then restore the saved screenshot using the [image editor](#).



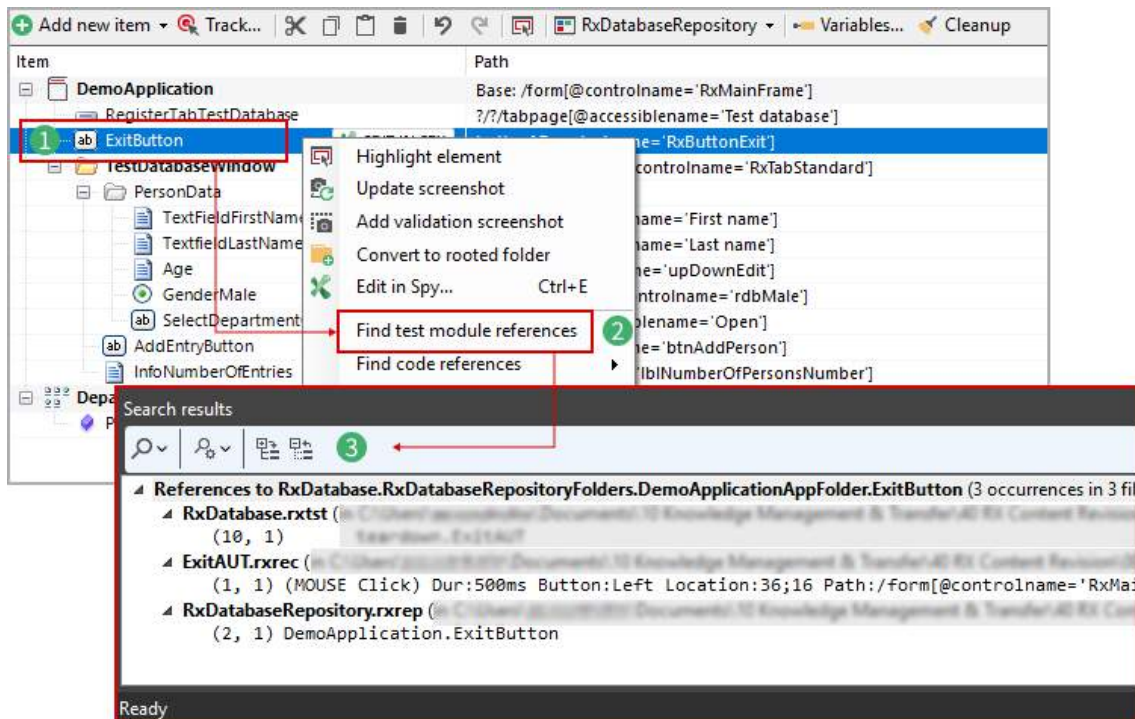
- 1 **Right-click** the screenshot and click **Save screenshot...**

## Find references

In large test projects, a repository item may be referenced in many modules and/or pieces of code. That's why it can make sense to check where exactly the repository item is used before you make changes to it. The two context menu options **Find test module references** and **Find code references** allow you to do so.

### Find test module references

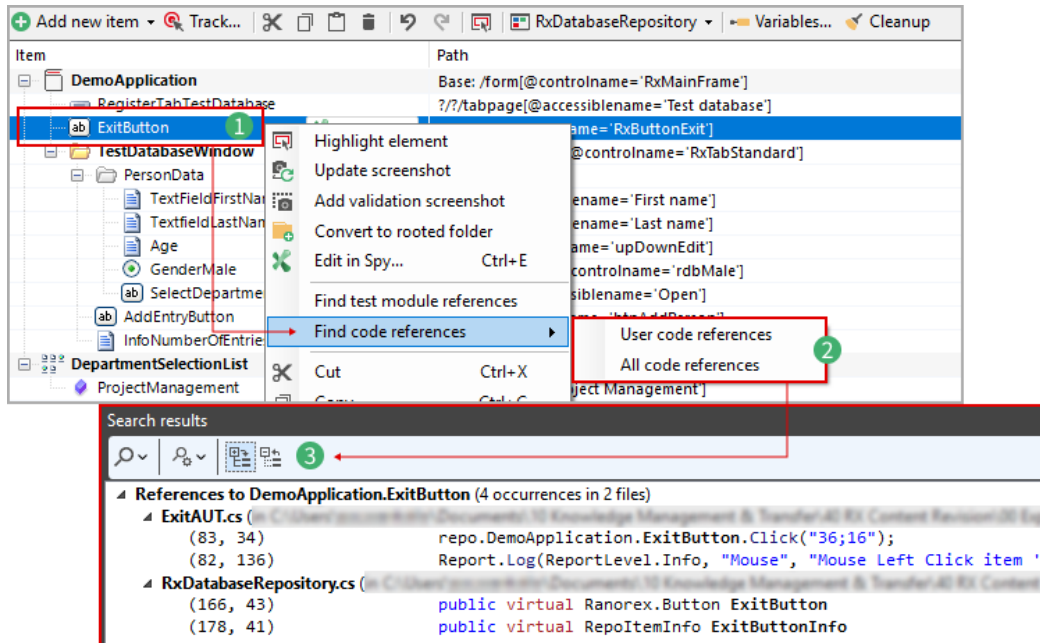
This option shows you all test modules where the repository item is referenced.



- 1 **Right-click** the desired repository item.
- 2 **Click Find test module references.**
- 3 The references appear in the search results at the bottom of Ranorex Studio.

### Finding code references

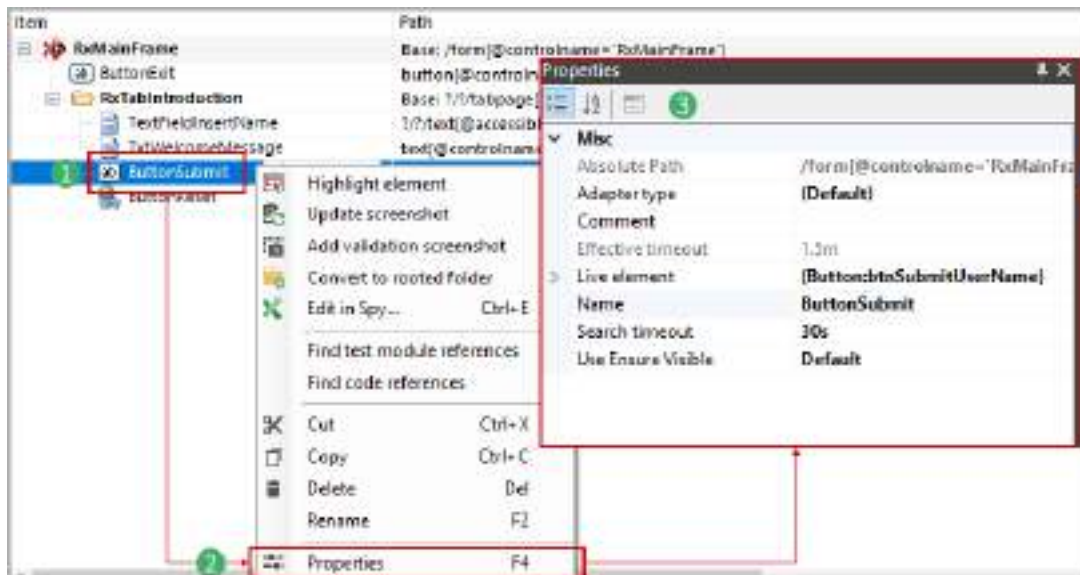
This option shows you all code instances where the repository item is referenced.



- 1 **Right-click** the desired repository item.
- 2 **Click Find code references** and **select** one of the two options.
- 3 The code references are displayed in the search results at the bottom of Ranorex Studio.

## Repository item properties

Each repository item also has a set of properties that can be configured in its Properties panel.



- 1 **Right-click** the desired repository item.
- 2 **Click Properties.**
- 3 The properties open to the right of the repository.

#### List of properties

<b>Absolute path</b>	Represents the path to the repository item including the paths of all parent folders. Read-only.
<b>Adapter type</b>	Lets you select the adapter type. By default, Ranorex will automatically select the most appropriate adapter.
<b>Effective timeout</b>	The sum of all search timeouts applied to the specific repository item and all its parent folders. Read-only.
<b>Comment</b>	Lets you add a comment to the repository item.
<b>Live element</b>	When the UI element represented by the repository item is live, i.e the AUT is opened and the element visible, this property displays a range of different parameters for it.
<b>Name</b>	The name of the repository item.
<b>Search timeout</b>	Defines the amount of time Ranorex will search for an element before an exception is thrown.
<b>Use ensure visible</b>	<p>When set to <b>Yes</b>, forces the UI element the repository item represents to become visible during a test run. For example, the UI element may be at the bottom of a webpage and you would need to scroll to it first to see it. With this property enabled, Ranorex will do this automatically.</p> <p>When set to <b>Default</b>, the property is inherited from the setting <b>Use 'Ensure visible'</b> by default in the <b>General</b> settings.</p>
<b>Use cache</b>	<p>Only available for app folders and rooted folders.</p> <p>When this is <b>True</b>, Ranorex will cache the UI element represented by the folder upon finding it. This can speed up automation because Ranorex won't search for the element's path any longer. However, we recommend you set this to False if the UI element changes regularly. In these cases, caching would slow down the test run and cause a warning message in the report. This is because the cached element would not match the actual element, and Ranorex would search for it using the absolute path. This also applies to folders that return two or more UI elements. Ranorex will always cache the first UI element it finds and the rest would again be searched for using the absolute path, causing slowdowns.</p> <p><b>Default values for manually created folders:</b> App folder: False</p>

	Rooted folder: False
--	----------------------

**Default values for automatically created folders** are set according to the technology of the UI element. In most cases, this means **False**.

## Structure repository items

As your tests grow larger and more complex, your repository will also contain an increasing number of items. Normally, they are structured automatically, i.e. Ranorex Studio will create app folders and rooted folders automatically. However, this can become confusing with many items. You may prefer to structure the repository yourself. This chapter describes the types of folders available and how to use them to structure your repository.



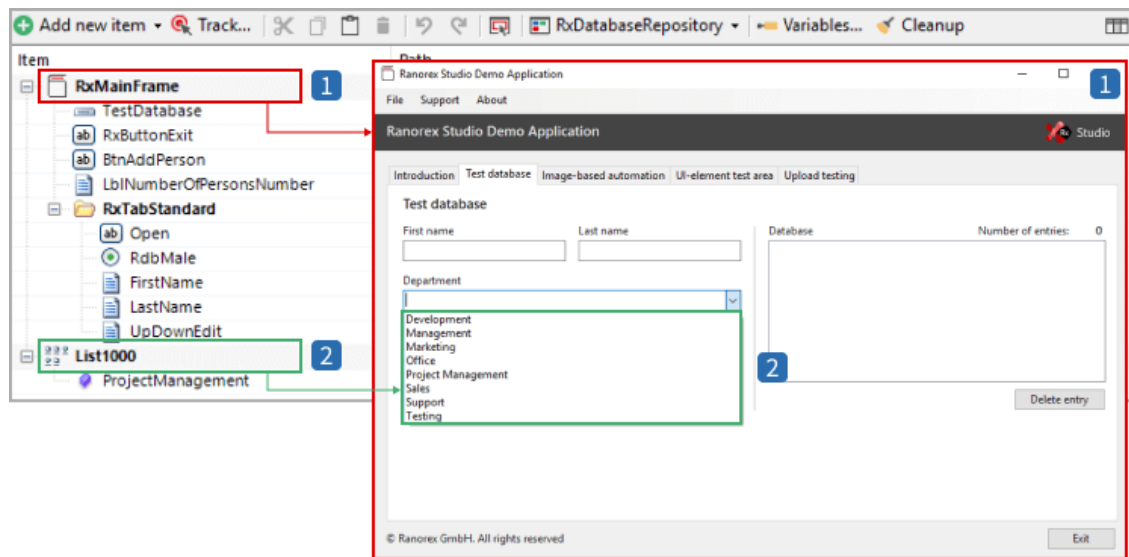
### Screencast

The screencast “Structure repository items” walks you through information found in this chapter:

[Watch the screencast now](#)

## App folder

App folders are **root** folders. They represent the top-level data container in the tree structure of repository elements. They can never be a child of another folder. In the example below, the folders **RxMainFrame** and **List1000** are app folders.



1

## RxMainFrame

- **RxMainFrame** is the top-level folder that contains all UI elements of the Ranorex Studio Demo Application.

2

## List1000

- **List1000** is the top-level folder that contains all selectable items in the Department drop-down.

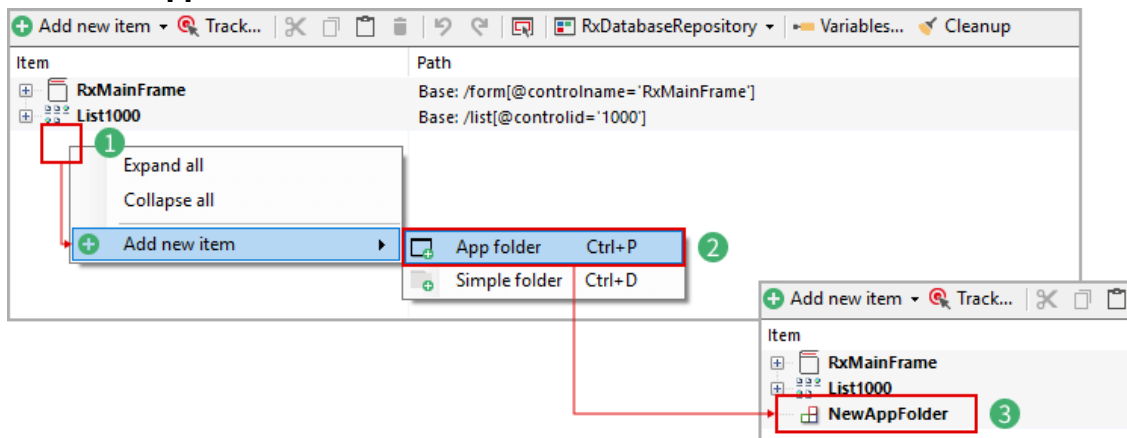


## Note

To make use of app folders, you will need to have a basic understanding of [UI elements](#) and the [RanoreXPath](#).



## Create an app folder



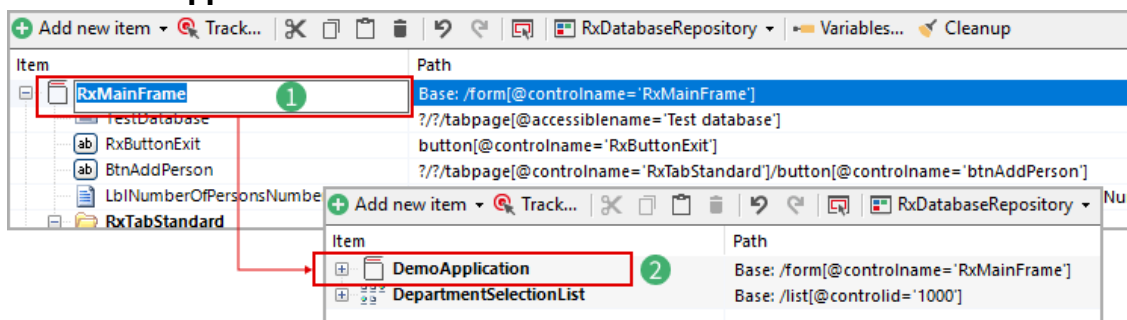
- 1 In the **Item** column, **right-click** in an empty area.
- 2 Click **Add new item** > **App folder** or press Ctrl+P.
- 3 The new app folder appears with the default name.



### Hint

You can also click **Add new item** > **App folder** in the menu bar of the repository view.

## Rename an app folder



- 1 **Select** the desired app folder and **press** **F2**.
- 2 **Enter** a name.

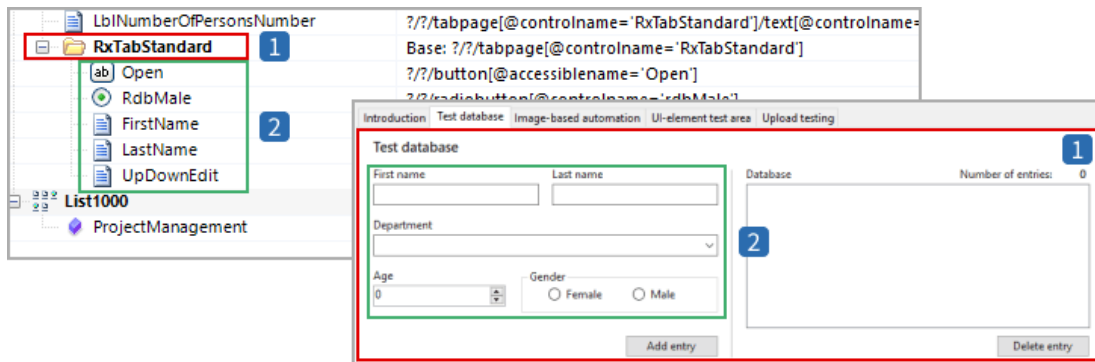


## Note

Note that the `controlname` in the path of the repository item remains unchanged. Only the display name in the repository is changed.

## Rooted folder

Rooted folders represent distinct parts of the UI that contain a set of individual UI elements. Repository items in a rooted folder all share the same base path, i.e. the path of the rooted folder. In the example below, the **RxTabStandard** is a rooted folder.



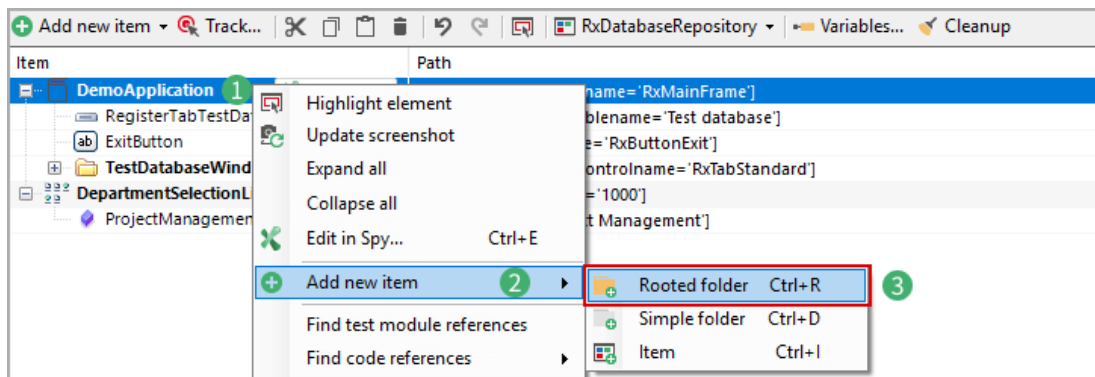
### 1 RxTabStandard

- The rooted folder and what it represents in the UI of the Demo Application, i.e. the Test database tab.
- All items in the rooted folder share the rooted folder's path.

### 2 Repository items in rooted folder

- Repository items representing the UI elements in the left part of the Test database tab, which in turn is represented by the rooted folder.

## Create a rooted folder



- 1 **Right-click** where you want to add the rooted folder.
- 2 **Click Add new item.**
- 3 **Click Rooted folder**, or press **Ctrl** + **R**.

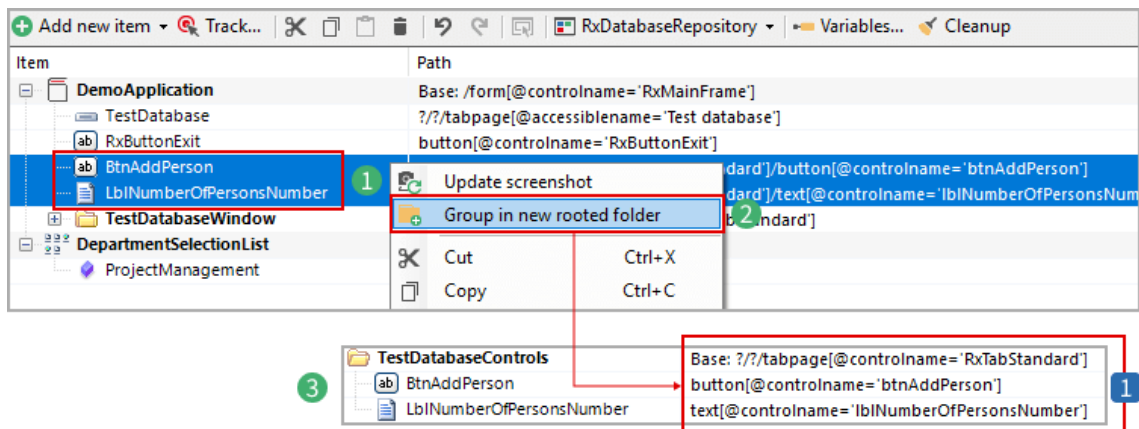


### Hint

You can also click **Add new item > Rooted folder** in the menu bar of the repository view.

## Group items in a new rooted folder

You can also group existing repository items in a new rooted folder.

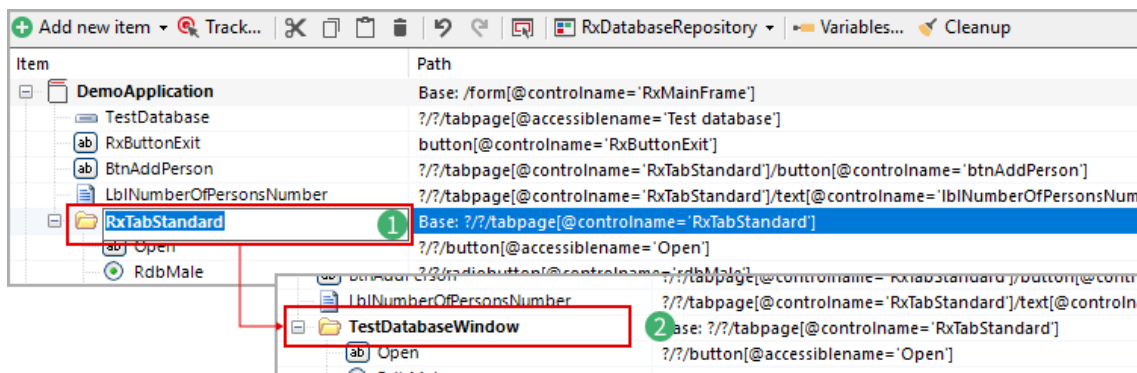


- 1 **Select** the repository items you want to group.
- 2 **Right-click** them and **click Group in new rooted folder**.
- 3 **Name** the folder. The repository items are grouped in the new folder.

### 1 **Path specification** of grouped repository items

- The rooted folder's path is the shared path of the selected repository items.
- The repository items are displayed with their individual path endings.

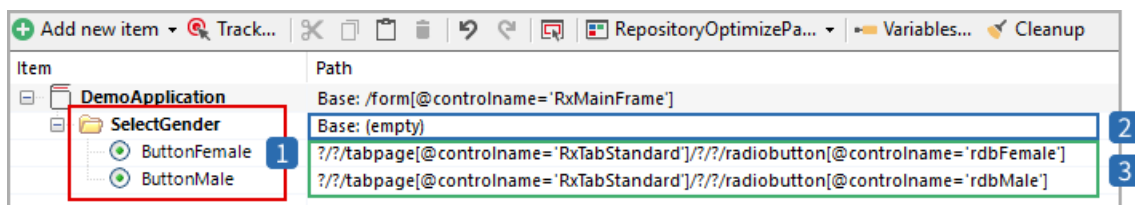
### Rename a rooted folder



- 1 **Click** the desired rooted folder and **press** **F2**.
- 2 **Enter** a name.

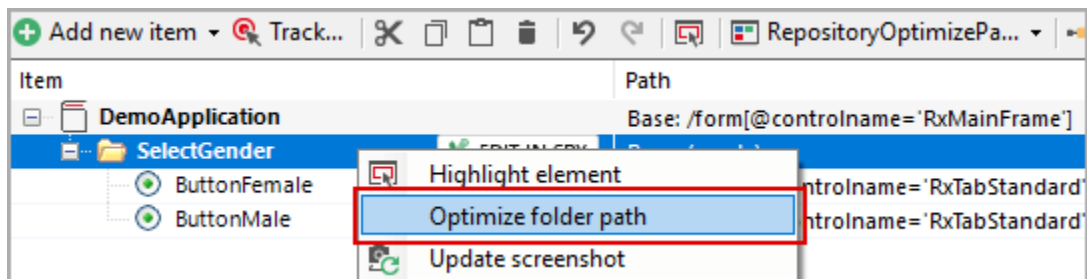
### Optimize a folder's path

A rooted folder's path should always be the longest possible path shared among all the repository items it contains. To ensure this is the case, use the **Optimize folder path** option.



- 1 The rooted folder **SelectGender** and its two repository items **ButtonFemale** and **ButtonMale**.
- 2 The base path of the rooted folder is empty.
- 3 Full paths of the two repository items. Note they only differ in the control name at the end.

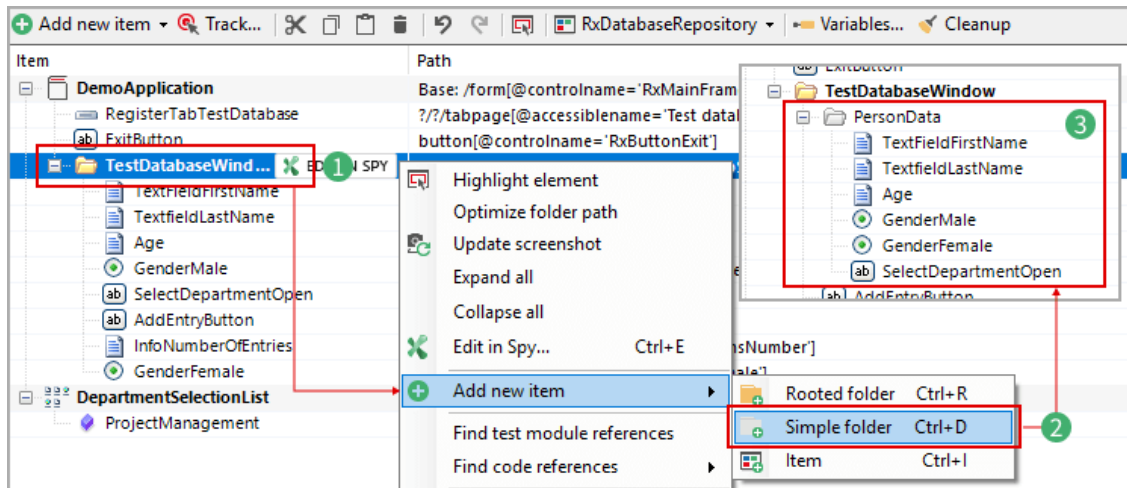
- 1 **Right-click** the rooted folder.
- 2 **Click Optimize folder path.**



- 1 Again, the same rooted folder **SelectGender** and its two repository items **ButtonFemale** and **ButtonMale**.
- 2 The base path of the rooted folder is now the longest shared path of the two repository items.
- 3 Only that portion of the path required to uniquely identify the two repository items is displayed.

## Simple folder

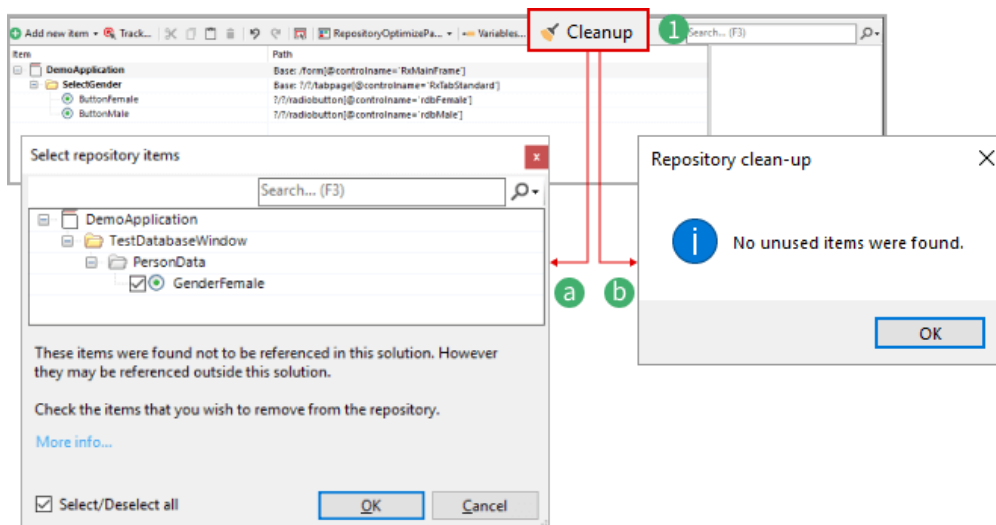
Simple folders do not have a base path. These folders allow you to use your own logical categories, like folders in Windows. Simple folders are never created automatically. They need to be created and managed manually.



- 1 **Right-click** where you want to add the simple folder.
- 2 **Click Add new item > Simple folder** or press **Ctrl + D**.
- 3 **Name** the folder and **fill** it with repository items.

## Clean up the repository

As you update actions in recordings and change your tests, repository items that were previously linked to actions may become unlinked and therefore unused. To keep your repository tidy, you can remove these unused items with the **Cleanup** button in the repository toolbar. This is usually a better idea than manually deleting items, as it is faster and less prone to errors.



- 1 **Click Cleanup** in the repository toolbar.
  - a A summary with all unused repository items appears and you are asked to confirm the deletion.
  - b If no unused repository items are found, a message appears.



### Attention

If you use a repository in **more than one project**, **be careful** with the cleanup function. Make sure all projects that use this repository are **loaded in the current solution** before you carry out a cleanup. Otherwise, you may inadvertently delete repository items that are still used in unloaded projects.

## Represent multiple elements with a single repository item

There are times when it may be helpful to have a single repository item represent multiple UI elements. Examples include elements such as radio buttons and checkboxes. This feature is commonly used with code modules.



### Screencast

The screencast “Represent multiple elements with a single repository item” walks you through information found in this chapter:

[Watch the screencast now](#)

## Necessary context

The concept covered in this chapter is often used with code modules, which is an expert topic. In addition, you will find the example easier to understand if you are familiar with the RanoreXPath syntax and the Ranorex Spy tool. Use the links below to learn more about these advanced topics.



## Further reading

Code modules are introduced in > Ranorex Studio expert > [Code modules](#).

The RanoreXPath syntax is described in > Ranorex Studio advanced > [RanoreXPath](#).

Learn about the Ranorex Spy tool in > Ranorex Studio advanced > [Ranorex Spy](#).

## Example definition

In the following example, **one** repository item represents **two** radio buttons. These radio buttons appear within the gender selection region of the database in the demo application.

The screenshot shows the 'Test database' tab in Ranorex Studio. It contains a form with the following elements:

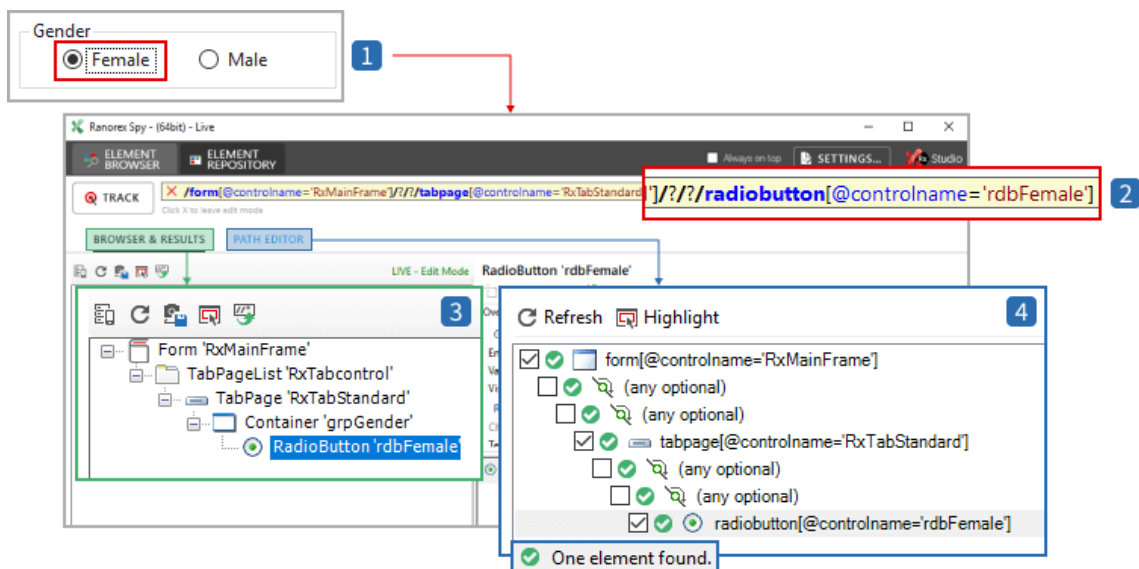
- First name**: A text input field.
- Last name**: A text input field.
- Department**: A dropdown menu.
- Age**: A numeric input field with a spinner, currently showing '0'.
- Gender**: A section containing two radio buttons, 'Female' and 'Male'. This section is highlighted with a red rectangle.
- Add entry**: A button at the bottom right.

## Track the repository item

The procedure for creating a repository item to represent multiple elements is the same as for a regular repository item. Refer to the instructions below:

- 1 **Open Ranorex Spy**
- 2 **Start the demo application** and click on the Test database register tab

### 3 Track one of the gender selection radio buttons

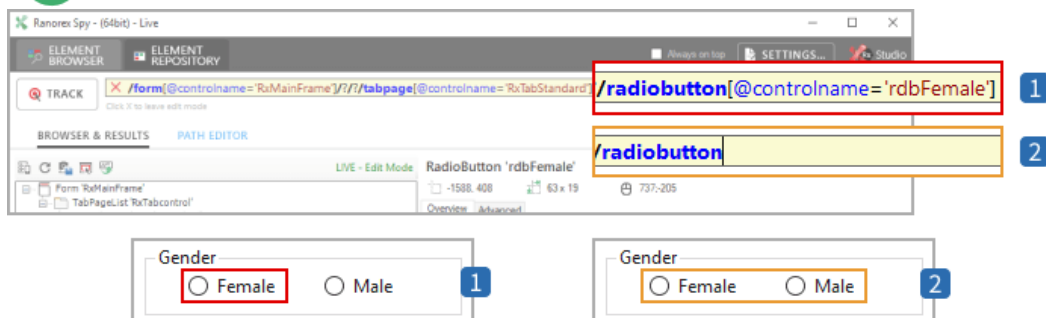


- 1 Tracked and identified **female radio button** UI element in the demo application
- 2 **Unique RanoreXPath specification** for the female radio button in Ranorex Spy
- 3 UI element browser tree displaying the **defined path for the female radio button in Ranorex Spy**
- 4 **Path editor** for the female radio button in Ranorex Spy

## Generalize the RanoreXPath specification

Follow the steps below to generalize the RanoreXPath specification so that it includes the second gender selection radio button of the demo application.

### 1 Change the RanoreXPath specification as shown here

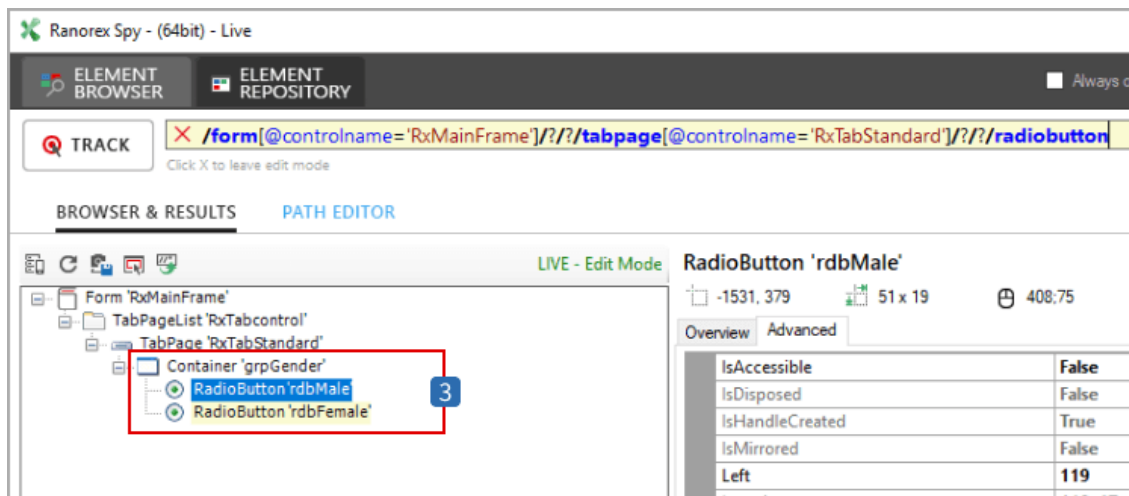




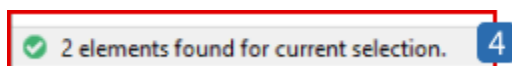
- 1 Initial RanoreXPath specification that includes only the female radio button
- 2 Modified, generalized RanoreXPath specification that includes both radio buttons

## Multiple repository items detection

Generalizing the RanoreXPath specification leads to the simultaneous tracking of both gender selection radio buttons in the demo application. This can be seen in Ranorex Spy.



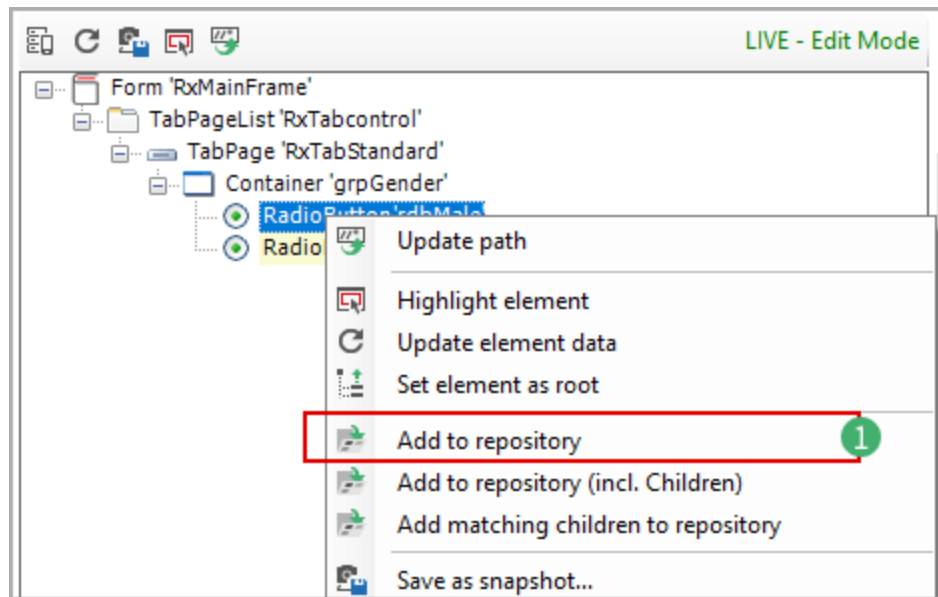
- 3 Positive match of two tracked radio buttons



- 4 Ranorex Spy displays the match result in the lower left corner of the working environment

## Add the tracked item to the repository

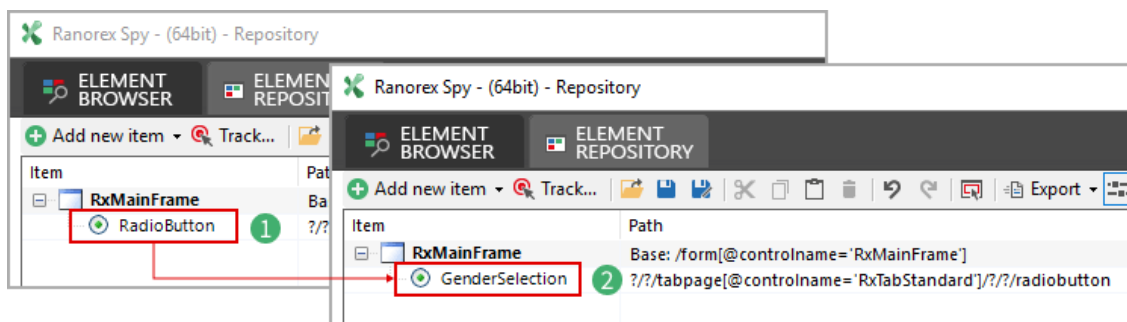
Finally, add the matched items to the repository.



- 1 Select one of the matched repository items, open the context menu and click **Add to repository**

## Name the generalized repository item

Like all repository items, the generalized repository item can be renamed to be more meaningful.



- 1 Select the repository item representing multiple UI elements.
- 2 Open the context menu to change the repository item name.

## Manage multiple repositories

By default, a Ranorex Studio project contains a single repository file that's used automatically for each new recording module. You can manage all repository items that your modules reference in a single repository, but there are also good reasons for having multiple repositories in a single project. This chapter explains these reasons and describes how to work with multiple repositories.



### Screencast

The screencast “Manage multiple repositories” walks you through information found in this chapter:

[Watch the screencast now](#)

## Reasons for multiple repositories

### Testing different user interfaces

Let's say your test suite contains test cases for a web application and test cases for a user interface of a client application. In this scenario, using two repositories would make sense. One would contain the repository items for the web application, while the other would contain those for the client application. You could also separate the repository items for both applications in a single repository by using simple folders, but using two repositories is more convenient, especially when working in a team.

### Modularizing repositories

It's a good idea to modularize repositories in a way similar to recording and code modules. For example, when you think about a rich client application with main menus, ribbons, or toolbars, you would create small reusable recordings for clicking UI elements in the main menu like **File > Open > Handle the Open File Dialog** etc.

All these reusable modules work with the main menu, the main toolbar or similar, i.e. shared controls. Therefore, it's a good idea to also base them on a repository which exclusively represents these shared controls.

## Advanced RanoreXPath Expressions

Another reason to build a separate repository could be to store advanced RanoreXPath expressions which should exclusively be used to create new actions manually instead of recording them.

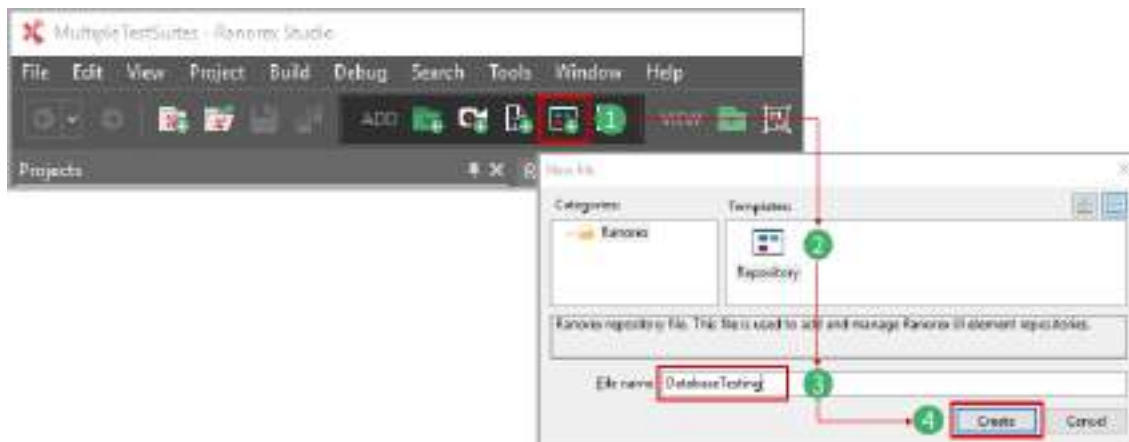
## Multiple testers on the same project

While working alone on a project, a single repository is often sufficient and usually won't be an issue. However, when working with others, this can quickly result in conflicts. This can be circumvented by using multiple repositories. Also make sure to set up rules and responsibilities, like: Who may rename repository items? Who may restructure repositories? Who may delete items?

## Complexity

Some of our customers have tests with thousands of repository items. It's quite obvious that keeping all of them in one repository would make it incredibly hard to maintain, not to mention the file size of such a repository. As your tests grow larger, think about how you can sensibly divide your repository items into different repositories to keep them maintainable and performant.

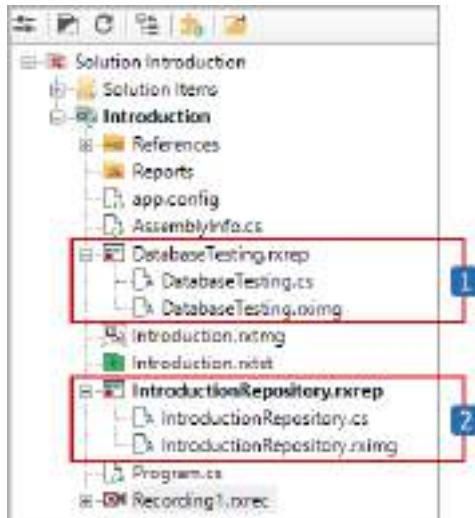
## Add a new repository



- 1 In the Studio toolbar, **click** the **Add repository** button.
- 2 The repository template is preselected.
- 3 **Name** the repository.
- 4 **Click Create**.

## Result(s):

- The new and empty repository is added and can be seen in the projects view.



1 Newly added repository file with code representation and image hosting file.

2 Default repository.

## Assign a repository to a module

Once you have more than one repository, you can assign them to different recording modules. You can then use the repository items of that repository in the recording.

## Currently assigned repository

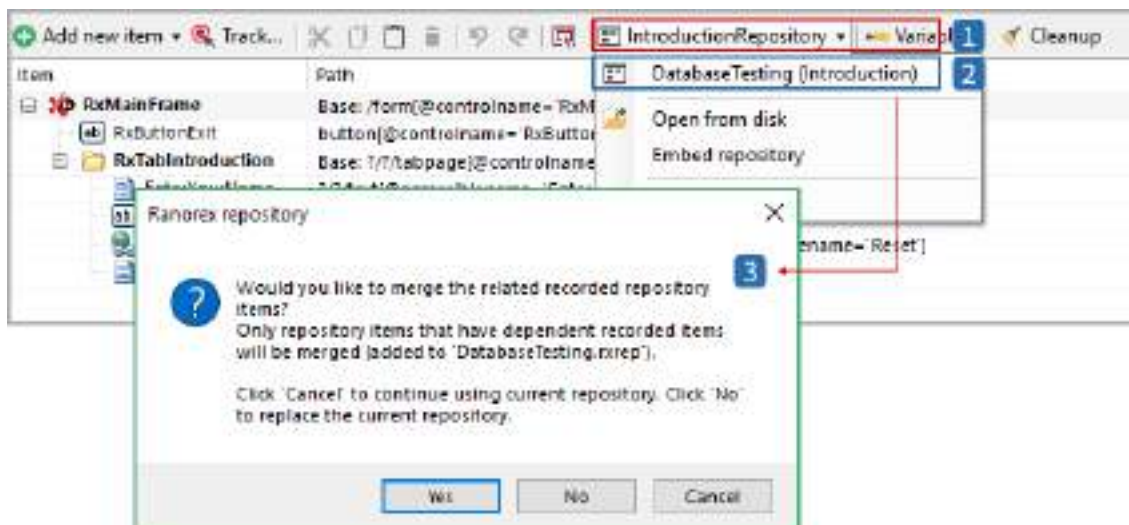
The currently assigned repository is displayed in the repository's toolbar when a recording module is opened.



- 1 The recording module `Recording1.rxrec` ...
- 2 ... has the repository `IntroductionRepository` assigned to it.

### Assigning a different repository

- 1 In the repository toolbar, **click** the currently assigned repository.
- 2 In the drop-down menu, **click** the repository you want to assign.



- 1 The currently assigned repository. In this case, it's the default repository.
- 2 The other available repository, `DatabaseTesting`.
- 3 If your recording module contains actions that are linked to repository items from the currently assigned repository, you can choose to have Ranorex transfer these repository items to the newly assigned repository.

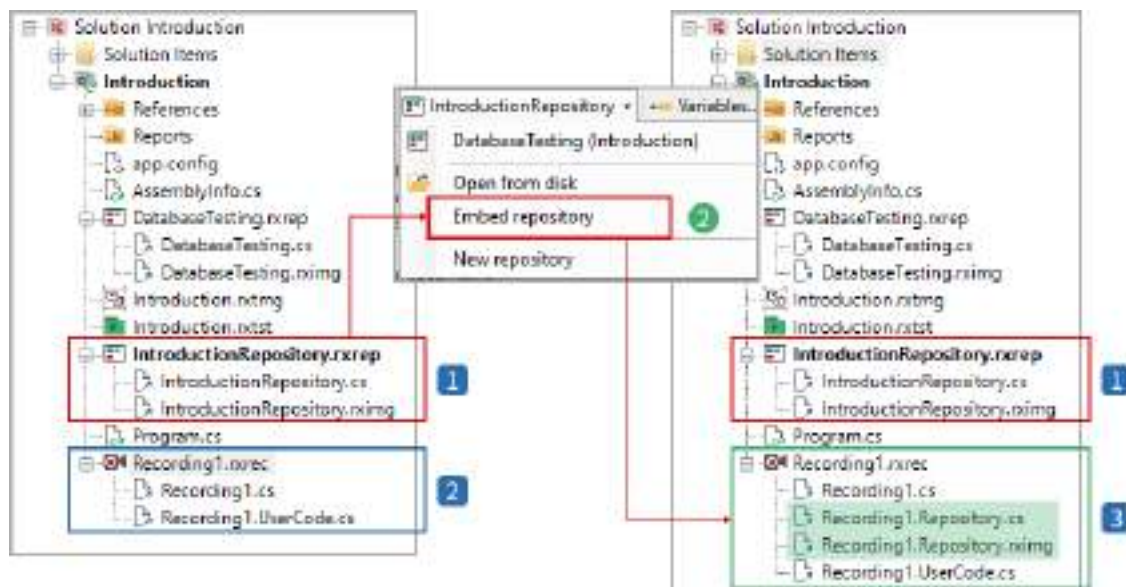
## Embed a repository

Usually, repositories are separated from recording modules. This separation has many advantages, easy re-use of repository items in different recording modules being just one of them.

However, there are cases when you may want to embed a repository in a recording module. Embedding is useful when you want to send someone else a recording you've made. Without it, you'd have to send both the recording and the repository separately. The standalone Ranorex Recorder also always creates recordings with embedded repositories for this reason.

## Embed a repository

- 1 **Open** the recording module in which to embed a repository.
- 2 In the repository toolbar, **click** the name of the currently assigned repository and **click Embed repository**. If you have multiple repositories, you can also assign a different repository first.



- 1 The repository file and its subfiles in the projects view.
- 2 Recording module **Recording1** and its subfiles.
- 3 Recording module **Recording1** with **embedded repository files**.

**Note**

Any changes you make to a recording module with an embedded repository are limited to that module and that repository. The original repository is not affected and other recording modules that use it will continue to work.



# Test validation

In software testing, validation is the process of verifying that the behavior and/or responses produced by the application under test match the expected results. This chapter describes the test validation approaches available in Ranorex Studio.

For simplicity, this chapter demonstrates validations that use constant values to determine whether the application returns an expected result. In practice, you will normally want to use variables for validation rather than constant values.

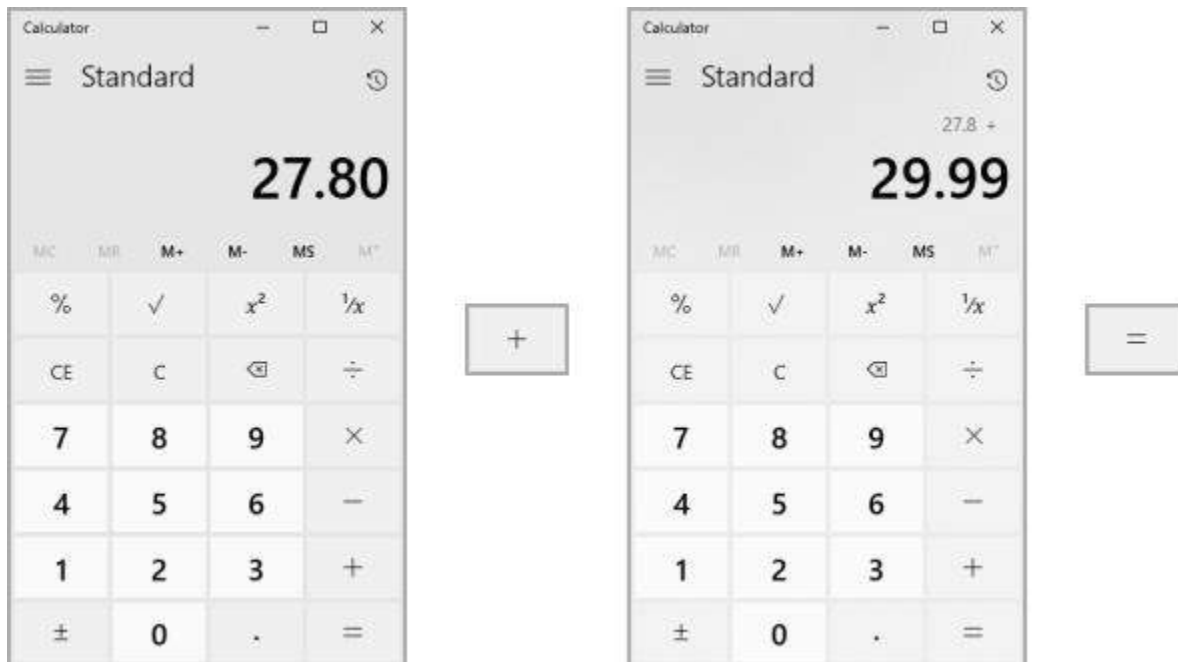


## Reference

To learn how to use variables in validation actions, refer to the following section:  
Ranorex Studio advanced > Variables and parameters > → [Validation variables](#).

## Understanding validation

Let's begin with a simple example of test validation. Imagine a standard pocket calculator that you use to add two numbers:



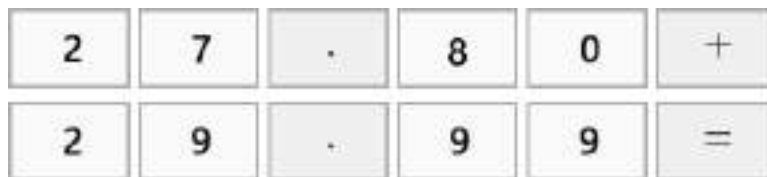
Test validation succeeds if the calculator returns the expected result, 57.79.

However, the manufacturer of the pocket calculator needs to guarantee that not only **this** result is correct, but also all other results, from the simplest addition to any other complex built-in mathematical formula.

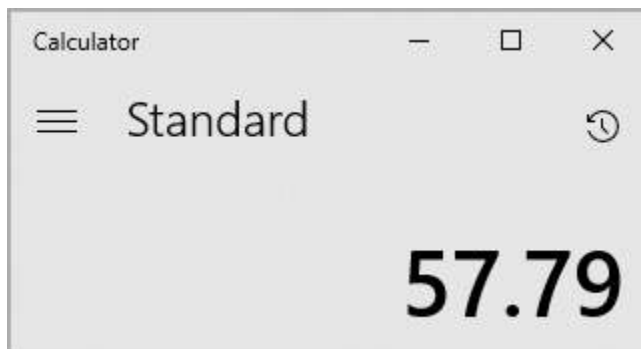
This is where test validation comes into play. Automated testing makes it possible to cover as many different calculations with different input data and results as necessary in order to prove that the calculator works correctly.

## Indirect validation

It is important to understand that GUI test automation provides *indirect test validation* only. This means that we verify whether the user input sequence of:



leads to the result of:



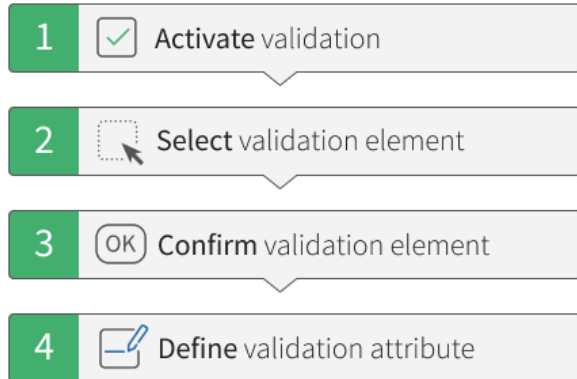
Indirect validation verifies that the **displayed result is similar to the expected result** based on the user input. It **does not ensure** whether **the calculator works correctly**.

- There might be many operating and processing steps between the user input and the display of the result on the screen.
- We are only able to match the displayed result against an initial sequence of user inputs actions and determine if the result or behavior is as expected.

In other words, if test validation fails, there is no evidence of what went wrong. It could be that the software processes everything correctly, but the displaying of the result went somehow wrong. Test data may contain erroneous data. You always need to be aware that **we are not able to validate the internal processing steps** within a computer system, but **only the 'outside' visible results**. Keep that in mind when designing your test cases.

## Validation Process

Test validation is a process with four consecutive steps which are listed below. You can create a test validation action during recording or add it manually later.



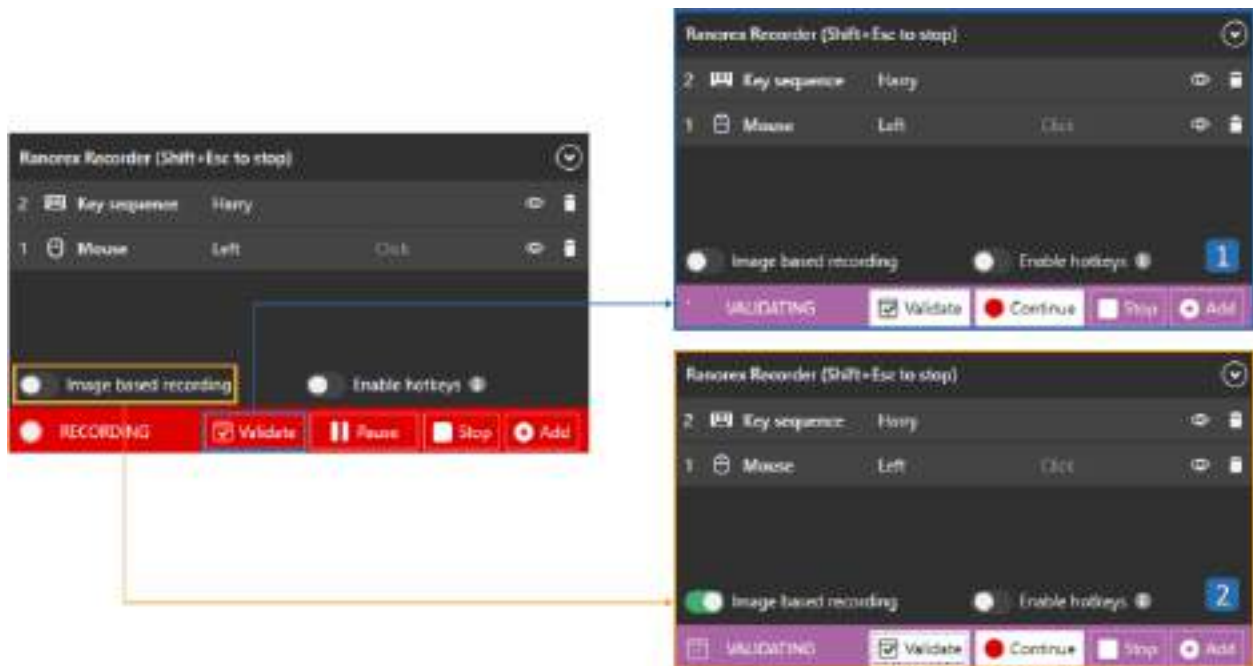
## Activate validation

Within Ranorex, validation is treated as a special type of action. So, to “activate” a test validation simply means to add a validation action to the current recording module. The instructions below describe adding a test validation during a recording session. As mentioned earlier, validation actions can also be added manually after recording.



### Reference

If you are not familiar with test actions, we recommend reading  
Ranorex Studio fundamentals → [Actions](#)



- 1 To configure **text-based validation**:
  - **Click Validate** to activate text-based validation mode.
- 2 To configure **image-based validation**:
  - **Activate** the switch **Image based recording**.
  - **Click Validate** to activate image-based validation mode.



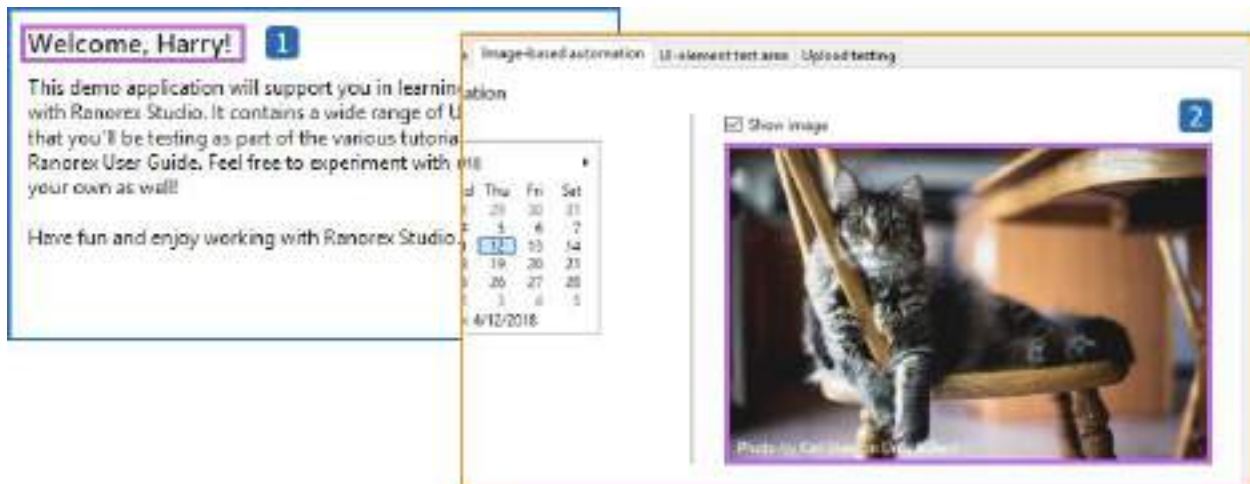
### Note

The recording session pauses automatically to allow you to insert the test validation action. This is indicated by the purple bar in the Recorder control center.

## Select the validation element

The second step of the validation process is to select the UI element to be validated. The selection process is independent of the type of validation (i.e. text/attribute/image-based validation).

Move the mouse over the UI element to be validated. Look for a purple frame to appear around the UI element, indicating that Ranorex Studio has identified it. Click to select the UI element.



- 1 Selection of a text-based validation element.
- 2 Selection of an image-based validation element.



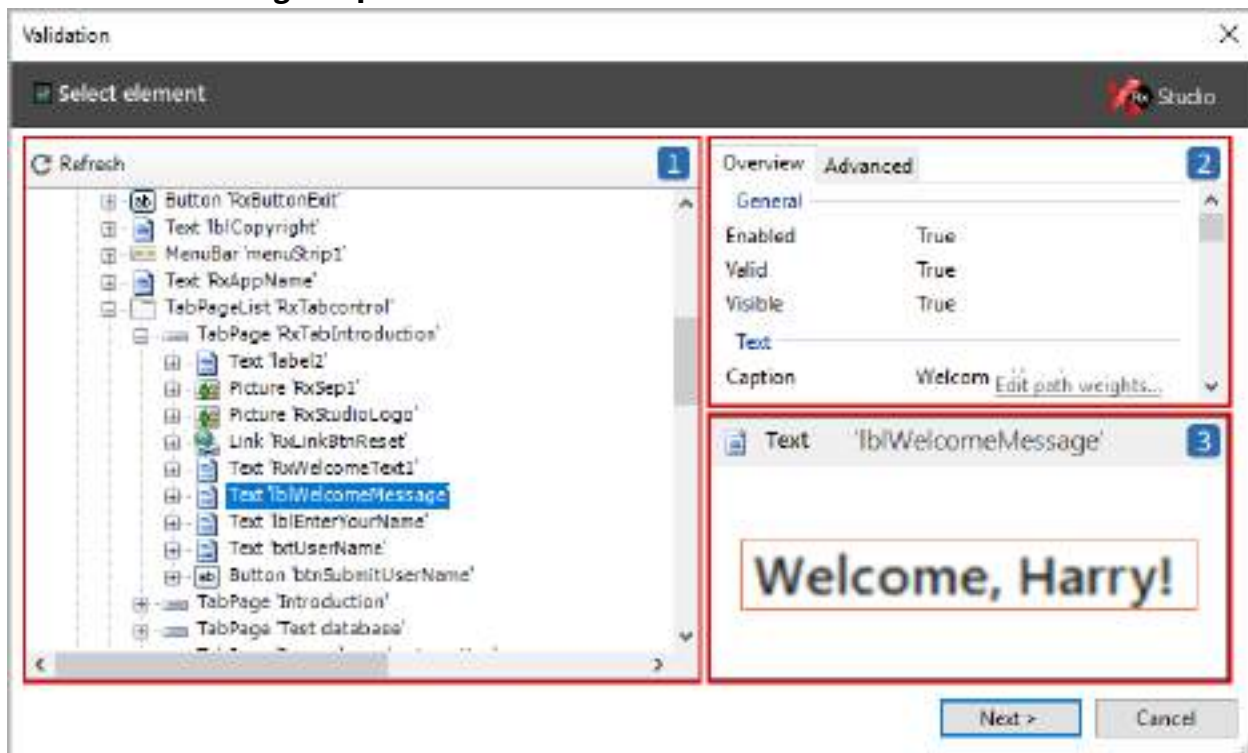
### Hint

- Identifying the correct UI element can be challenging, especially when it is nested closely with other UI elements.
- If you select the wrong UI element by mistake, you can easily change it during the next step.

### Confirm the validation element

The third step of the validation process is to confirm that the correct UI element was selected for validation. If so, click **Next**. Otherwise, correct the selection.

## Confirmation dialog components



### 1 UI element location within GUI

- Ranorex identifies every UI element within the application under test.
- All UI elements are managed in a hierarchical structure.
- The names of the UI elements are derived from their underlying type and content.

### 2 UI element states and attributes

- Each UI element is defined by a set of attributes and states.
- An example of a state is **Enabled** with a current value of **True**.
- An example of an attribute is the UI element's name. In the example above, the text attribute has the current value **lblWelcomeMessage**.

### 3 Screenshot of the UI element

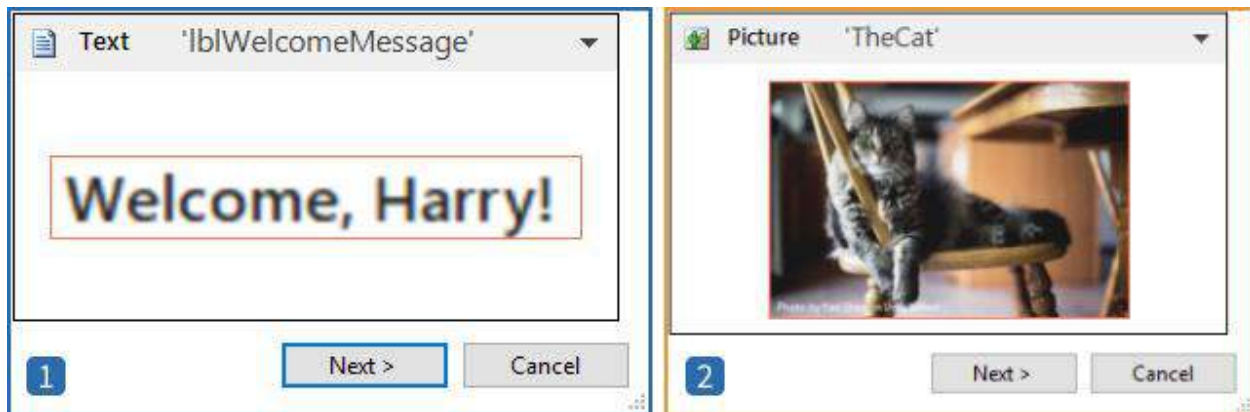
- A screenshot of the selected UI element is displayed for confirmation.



## Reference

UI elements, their representation within Ranorex and their states and attributes are covered in **Ranorex Studio advanced** > → **UI elements**

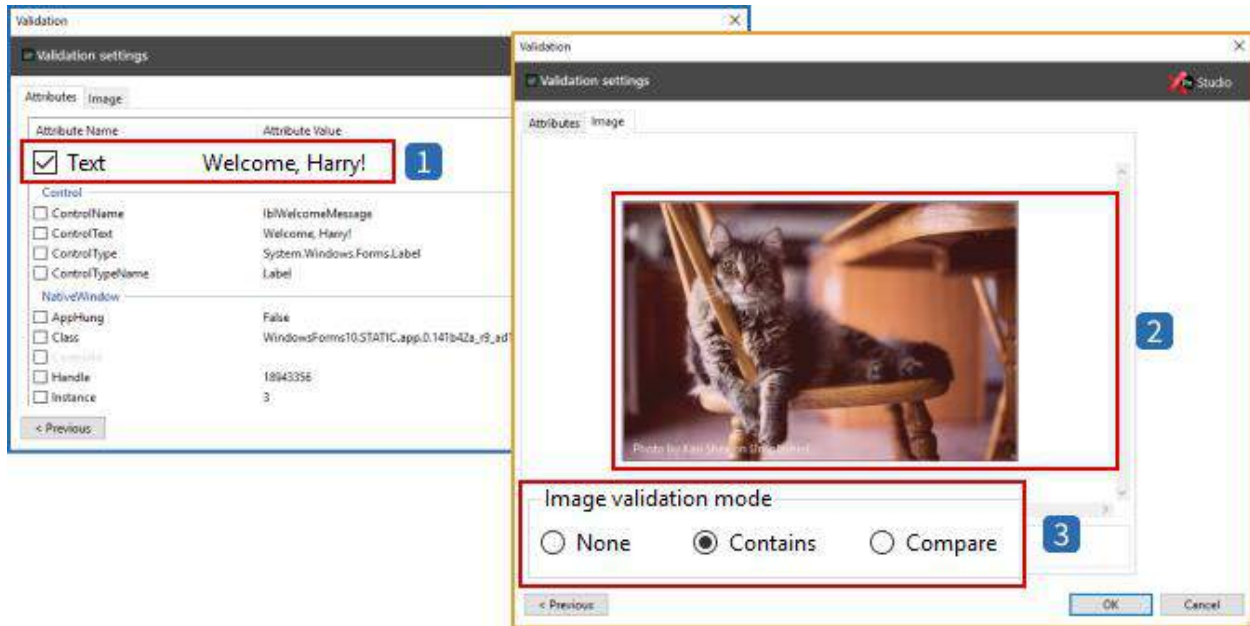
### Confirmation action



- 1 Confirmation of text-based validation element.
- 2 Confirmation of image-based validation element.

### Define validation attributes

Each UI element is defined by states and attributes. During the final step, specify the particular attribute or state to be used for validation, as shown below.



1

Attribute/state selection for **text/attribute-based** validation

- Any of the available attributes/states can be selected.
- The text attribute is the default attribute for text-based validation.
- **Select** the attribute and **click OK** to confirm.

2

(Sub-)image selection for **image-based** validation

- A sub-region of the image can be selected by drawing a pink rectangle within the image.
- If you do not select a sub-region, the complete image is used for test validation.

3

Image validation mode

- None = No image-based validation; validation discarded.
- Contains = Checks that a matched image is contained within a reference validation image.
- Compare = Checks a matched image against a complete reference validation image.

## Validation examples

To learn more about the three types of validation, refer to the examples linked below:





## Reference

---

Validation examples:

- [→ Text-based validation example](#)
- [→ Attribute-based validation example](#)
- [→ Image-based validation example](#)



## Reference

---

To learn more about UI elements, refer to

Ranorex Studio advanced > [→ UI elements](#).

To learn more about image-based automation, refer to

Ranorex Studio advanced > [→ Image-based automation](#).

## Text-based validation example

This chapter uses a simple example to explain the concept of **text-based validation**. Before working with this example, make sure you're familiar with the [→ basic concept of test validation](#).



### Screencast

The screencast “Text-based validation” walks you through information found in this chapter:

[Watch the screencast now](#)

## Download the sample solution

To follow along with the steps in this chapter, download the sample solution file from the following link: [Text-based validation example](#)

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `Introduction.rxsln`



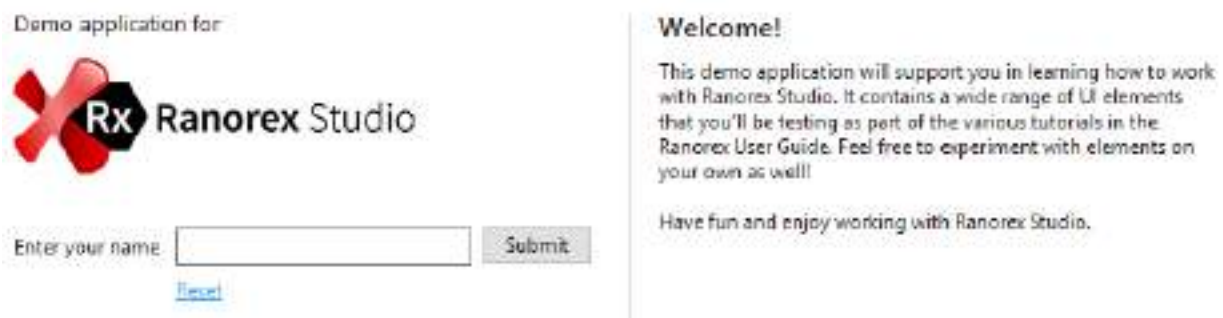
#### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

### Test definition

Before we start recording our test, let's define it. The test consists of 5 steps:

1. **Open** the Ranorex Demo Application.
2. In the "Enter your name" field, enter **Harry** and click **Submit**.



3. **Verify** that the welcome message changes accordingly.
4. **Reset** the welcome message.
5. **End** the demo application and **stop** the recording.

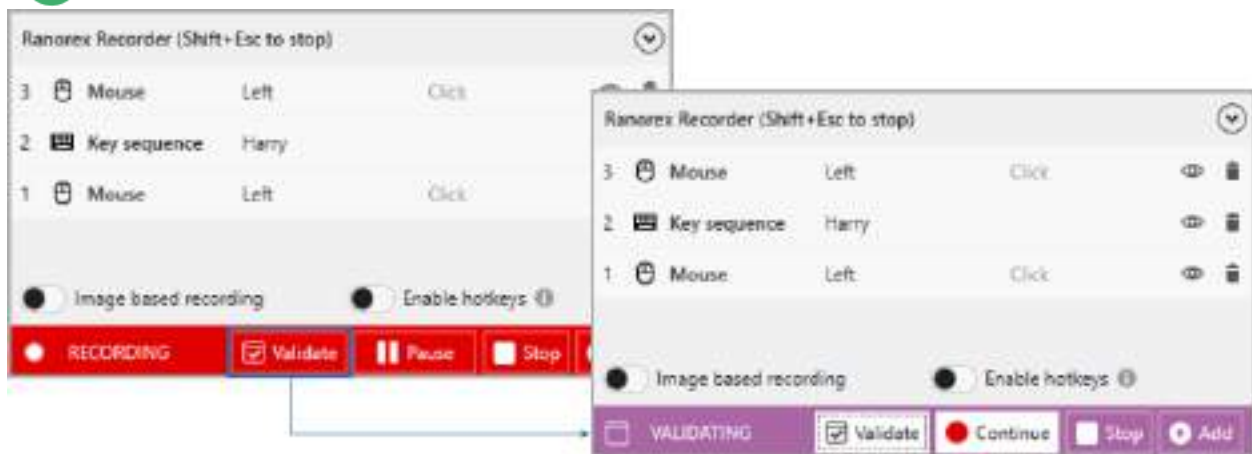
## Text-based validation

The purpose of the test validation in this example is to verify whether the interaction in step #2 of our test definition leads to the desired result, i.e. if the welcome message changes accordingly. Since this requires us to validate the text contained in a text field, we're carrying out a **text-based validation**.

Let's go through the steps:

### Activate validation

- 1 Click **Validate**. Recording pauses and the Recorder switches to validation mode.

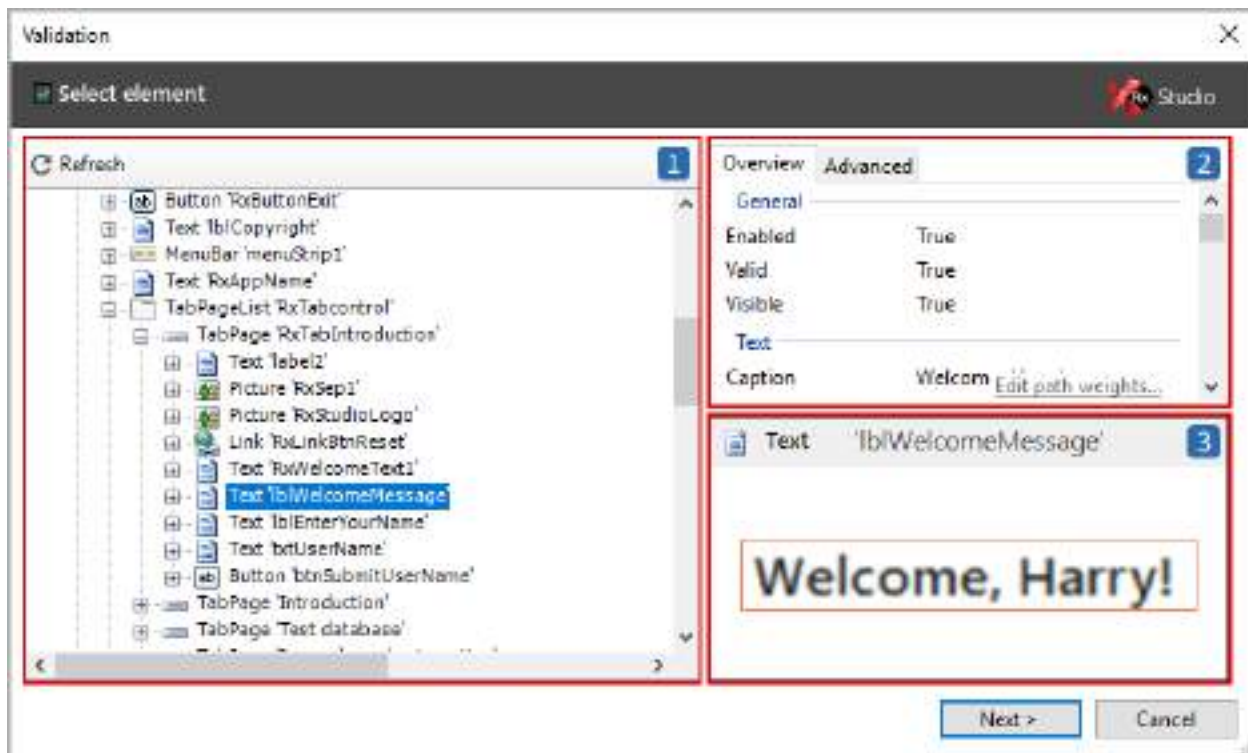


### Select validation element

- 2 **Select** the UI element to validate:
  - **Mouse over** the changed welcome message. A **purple frame** follows your mouse movement.
  - The purple frame indicates which element is currently selected for validation.
  - Once your selection matches the welcome message, **click it**.

### Confirm validation element

- 3 To confirm the UI element, **click Next**.



### 1 UI element location within GUI

- Here you can correct the previous choice by selecting any other UI element for validation.
- The UI element tree represents the hierarchical GUI structure of the application.

### 2 UI element states and attributes

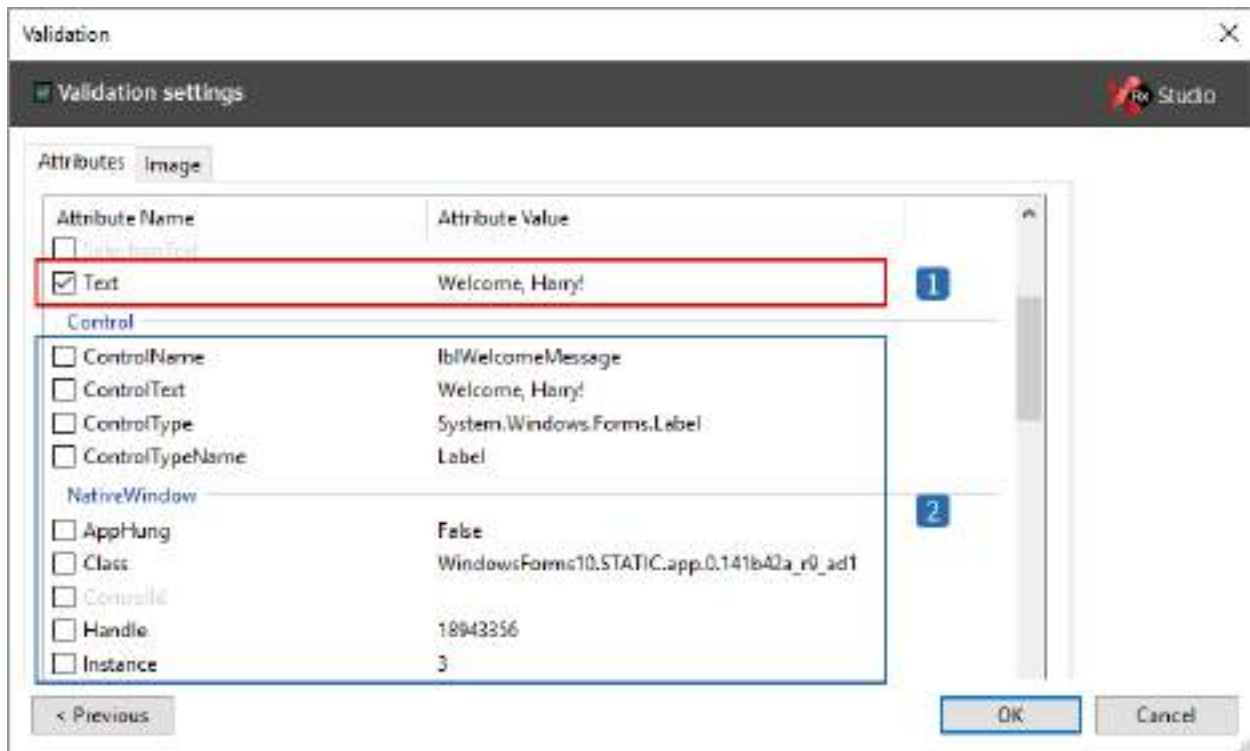
- Here, all the attributes of the selected UI element are displayed.

### 3 Screenshot of validation UI element

- Use the screenshot to quickly check whether you've selected the correct UI element.

## Define validation attributes

- 4 The attribute **Text** is usually preselected. If it isn't, **select** it. **Click OK** to confirm.



1 **Text** is selected as validation attribute.

2 Other attributes available for validation.



### Note

All attributes and states that describe and define a UI element can be used in validations. You can also validate several attributes and states at once to make a validation more specific.

## Result

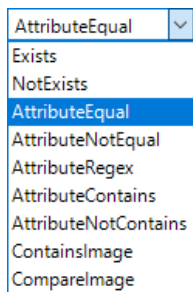
The finished recording contains five actions. **Action #4** is the validation action.

#	Action	Match name	Match value	Repository item
1	Mouse Click	Left	Relative	EnterYourName
2	Key sequence	Harry		EnterYourName
3	Mouse Click	Left	Relative	BtnSubmitUserName
4	Validate	AttributeEqual	Text	LblWelcomeMessage
5	Mouse Click	Left	Relative	Reset

1 Validation action

2 Validation type operator

- This operator defines what type of validation will be carried out.
- There are 9 different validation type operators.



## Further reading

A detailed description of all validation type operators can be found in Ranorex Studio fundamentals > Actions > [Action properties](#).

3 Validation attribute

- Column #4 shows the validation attribute.
- In our example, this is the **Text** attribute.
- You can select other attributes from the drop-down menu.

4 Match value

- The match value can be a constant (i.e. text, number, etc.) or a variable.

5 The linked repository item

- Column #6 shows the repository item the action is linked to, i.e. the UI element on which the validation is performed.

## Interpreting the validation

Spelled out, the validation reads as follows:

If the `Text` attribute of the UI element referenced by `LblWelcomeMessage` is equal (`AttributeEqual`) to “**Welcome, Harry!**”, THEN the validation returns the value `True`.

## Attribute-based validation example

In this chapter, we’ll demonstrate the concept of **attribute-based validation** based on a simple example. Before working with this example, make sure you’re familiar with the [→ basic concept of test validation](#).

Of course, [→ text-based validation](#) is also an example for attribute-based validation, but we’ve dedicated a separate chapter to it because it’s probably the most frequent type of validation.



### Screencast

The screencast “Attribute-based validation” walks you through the information found in this chapter.:

[Watch the screencast now](#)

## Download the sample solution

To follow along with this tutorial, download the sample solution file from the following link: [Sample Attribute Based Validation](#)

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `AttributeBasedValidation.rxsln`



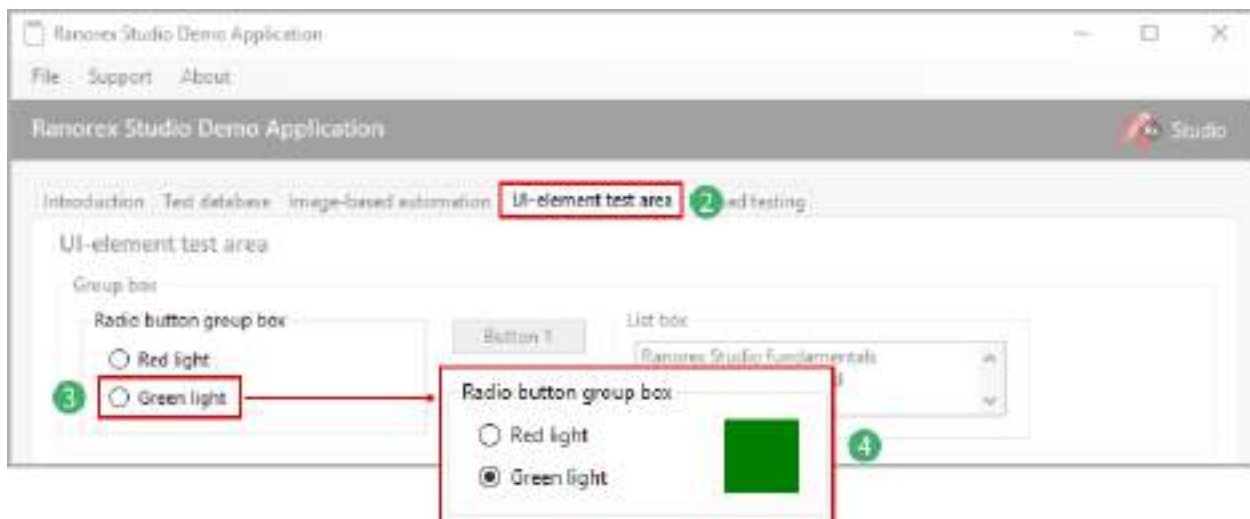
## Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

## Test definition

Before we start recording our test, let's define it. The test consists of 5 steps:

1. **Open** the Ranorex Studio Demo Application.
2. **Click** the tab **UI-element test area**.
3. **Click** the radio button **Green light** in the radio button group box.
4. **Verify** that a green square appears next to the radio button.
5. **End** the Demo Application and **stop** the recording.

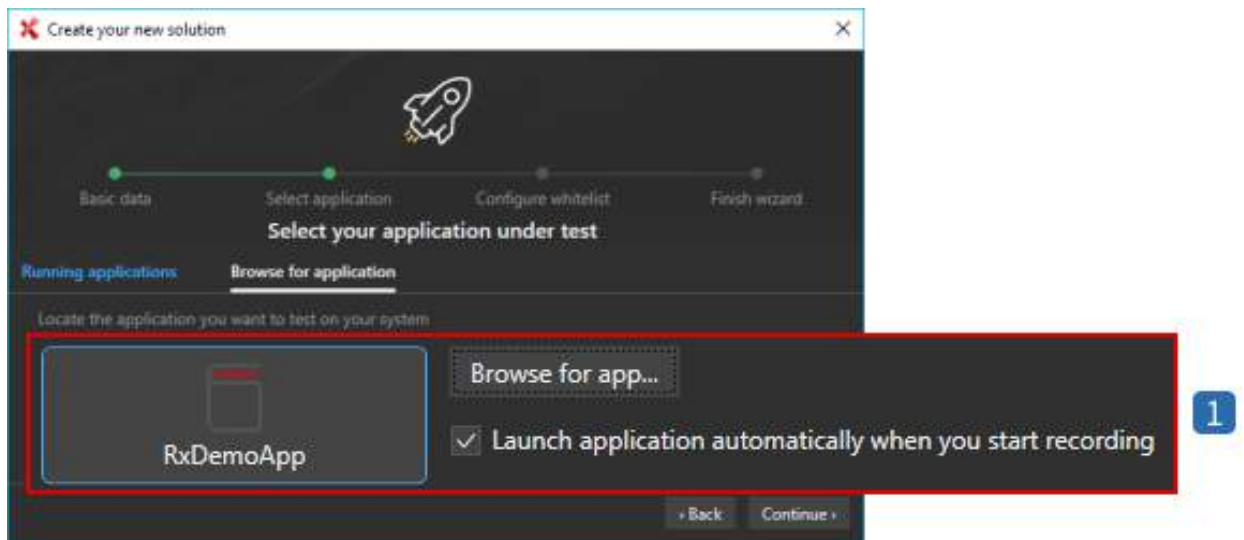


## Recording preparations

1

**Create** a desktop-test solution using the Solution Wizard and in step 2 of the setup, **select** the Demo Application as your AUT.





- 1 Demo App selected as AUT and is launched automatically when you start recording.
- 2 When the Solution Wizard has finished, **click** the **Recording1** tab in the Ranorex Studio working environment.

## Record the test: part one



### Hint

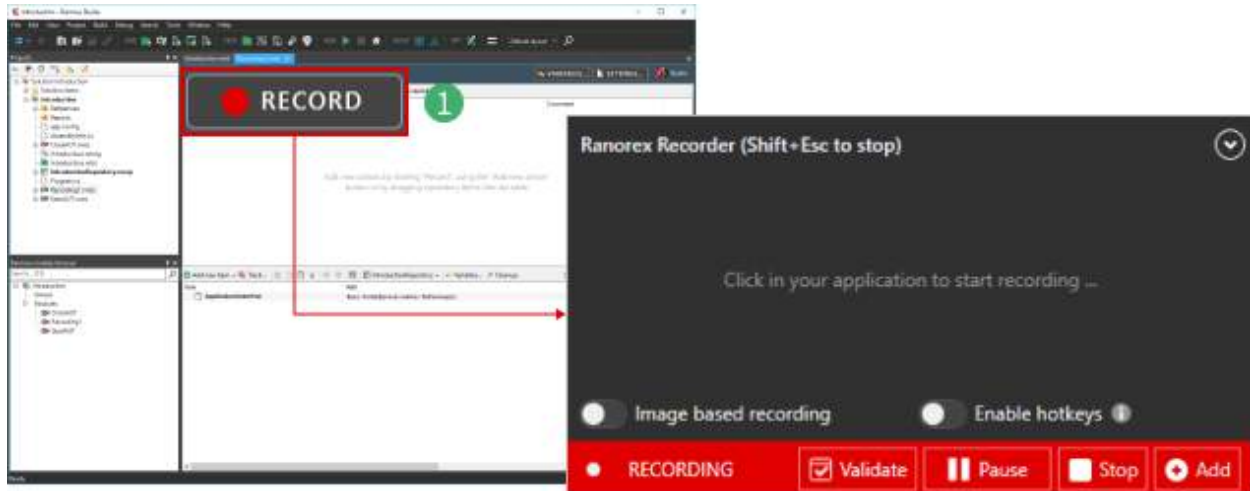
Remember, if you're not using whitelisting, all user interactions are captured once recording has started, even if they are not performed on the AUT.

- Click **Pause** to pause recording. Click **Continue** to resume recording.
- Click **Stop** to end recording.

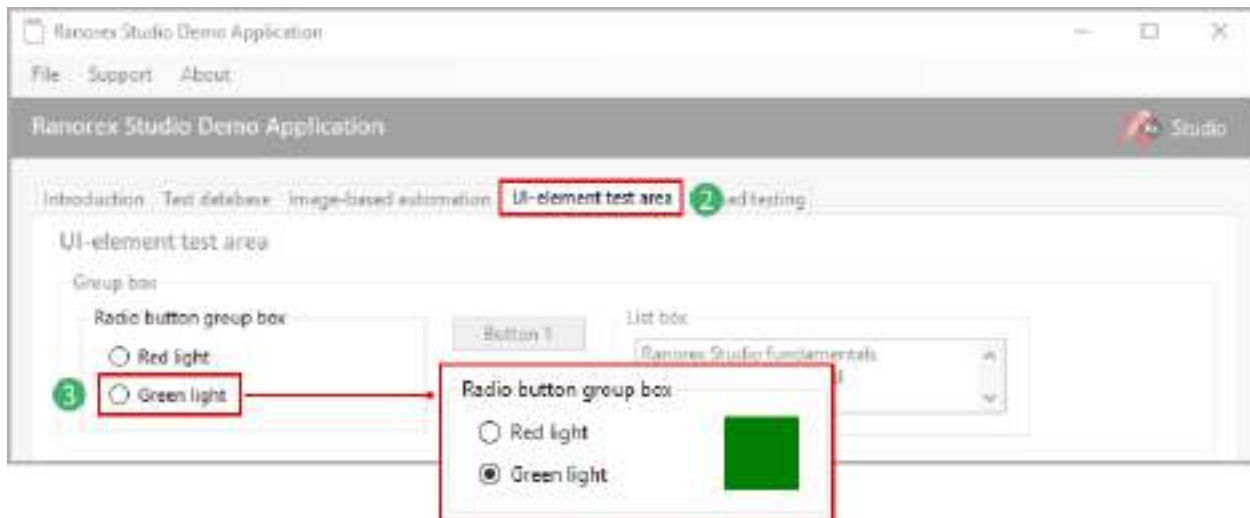
Learn more about the Recorder control center in Ranorex Studio fundamentals > Ranorex Recorder > → [Recorder control center & hotkeys](#).

Read about whitelisting in Ranorex Studio fundamentals > → [Whitelisting](#).

- 1 In the recording module view of **Recording1**, **click RECORD**. Ranorex Studio is minimized to the taskbar automatically.  
The Recorder control center shows that recording is active.



- 2 The application under test comes into focus.
- 3 In the radio button group box, **click** the radio button **Green light**.



## Attribute-based validation

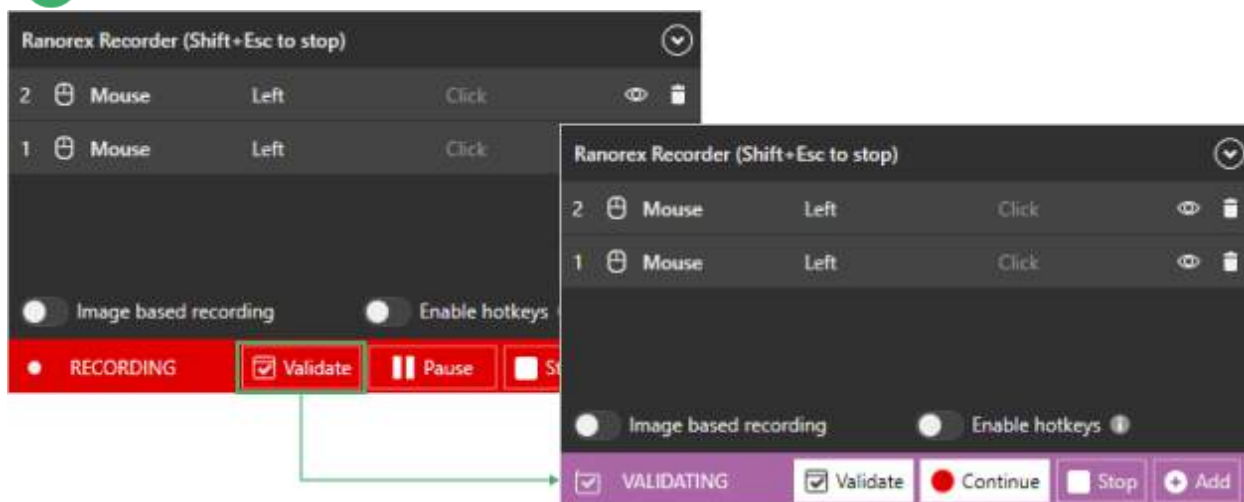
The purpose of the test validation in this example is to verify whether a colored square appears in the correct color after clicking the radio button.

Since this requires us to validate the **color attribute** of a UI element, we're carrying out an **attribute-based validation**.

Let's go through the steps:

## Activate validation

- 1 Click **Validate**. Recording pauses and the Recorder switches to validation mode.



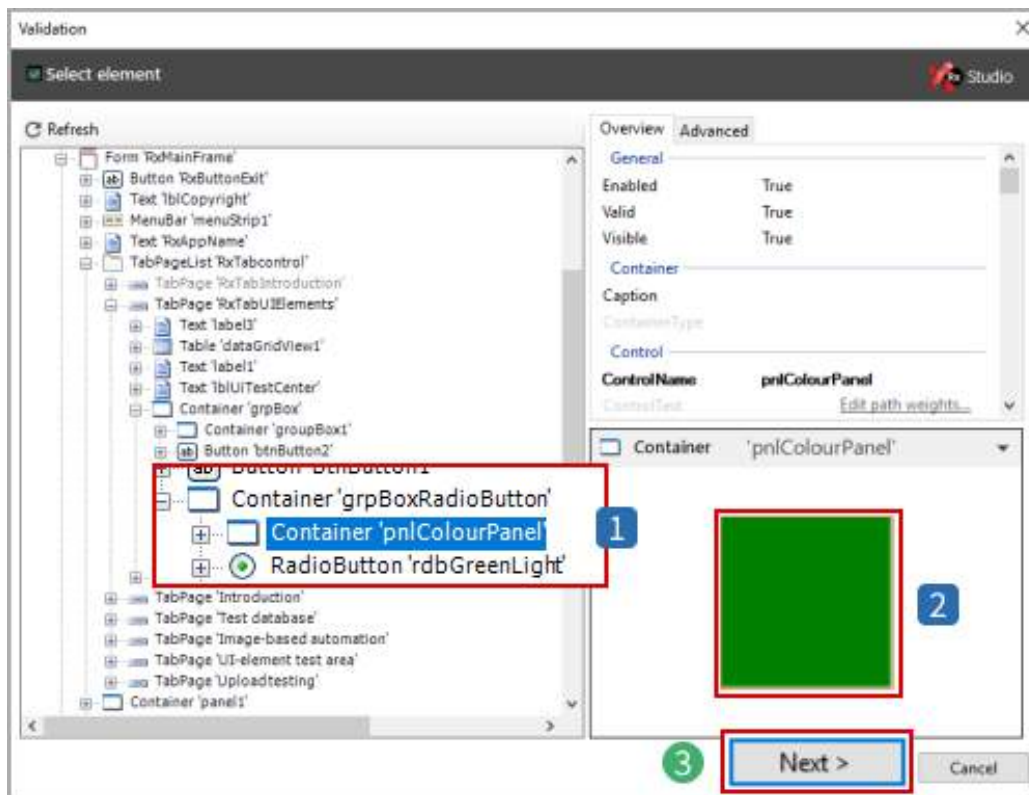
## Select validation element

- 2 **Select** the UI element to validate:
  - **Mouse over** the green square. A **purple frame** follows your mouse movement.
  - The purple frame indicates which element is currently selected for validation.
  - Once your selection matches the green square, **click it**.



## Confirm validation element

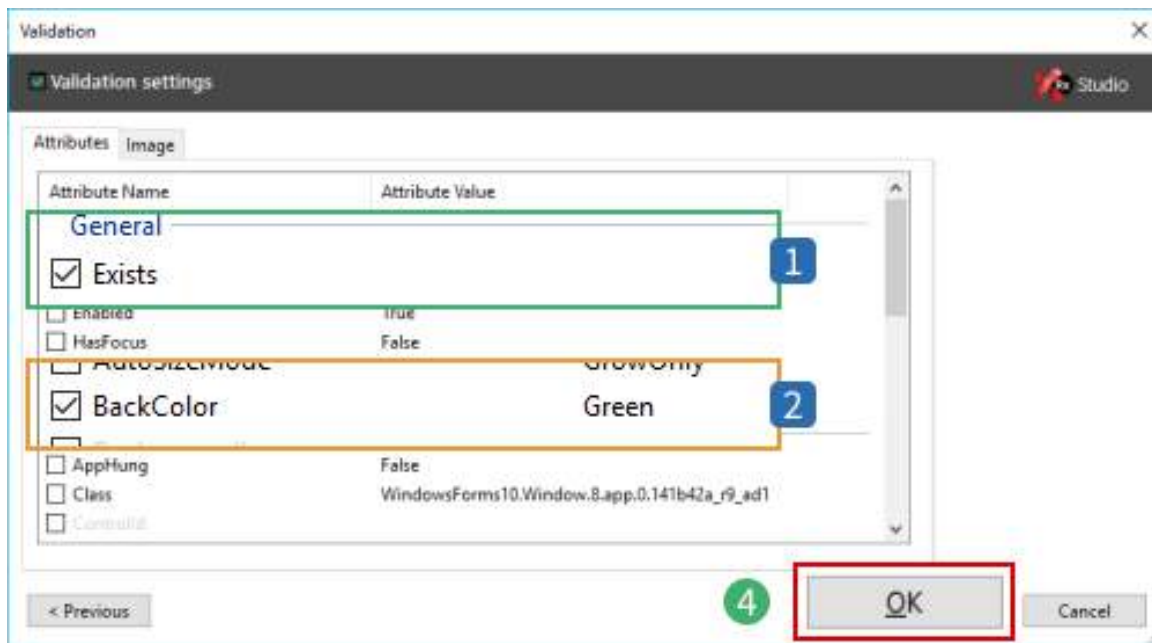
- 3 To confirm the UI element, **click Next**.



- 1 The selected **validation element**:
  - The UI element has the role **Container** and the **ControlName** **pnlColourPanel**
- 2 The **screenshot** of the selected UI element, showing the green square.

## Define validation attributes

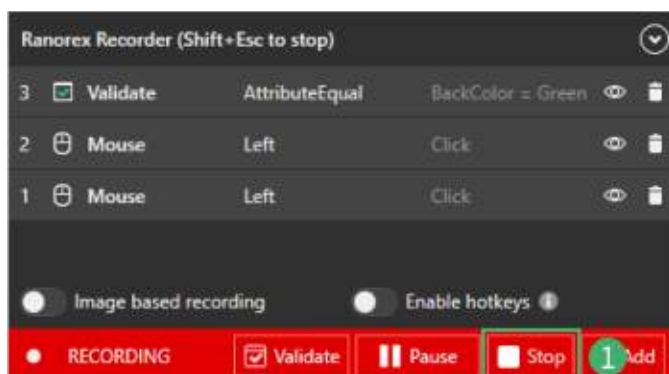
- 4 **Select** the attributes **Exists** (usually preselected) and **BackColor** and **click OK** to confirm.



- 1 The **general attribute** **Exists** is usually preselected.
- 2 The **dynamic attribute** **BackColor** with the value **Green** needs to be selected.

## Finish recording

Once you've finished the validation action, Ranorex automatically continues recording. The next step is to end the test recording.



- 1 In the Recorder control center, **click Stop** to end the recording.

## Result

After recording stops, you are returned to Ranorex Studio. The actions table contains three recorded actions. **Action #3** is the validation.

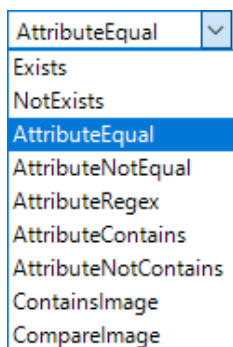
+ Add new action   ✂   📄   🗑   ↺   ⬆   ⬇   ⚙ Turbo mode   🖼 Screenshot					
#	Action		Match name	Match value	Repository item
1	Mouse	Click	Left	Relative	UIElementTestArea
2	Mouse	Click	Left	Relative	RdbGreenLight
3	Validate	AttributeEqual	BackColor	Green	PnlColourPanel

1 2 3 4 5

1 The validation action

2 **Validation type operator**

- This operator identifies the type of validation to perform.
- There are nine different validation type operators.



### Further reading

A detailed description of all validation match operators can be found in Ranorex Studio fundamentals > Actions > [Action properties](#).

3 **Validation attribute:**

- This column displays the validation attribute.
- In our example, this is the **BackColor** attribute.

- You can select other attributes from the drop-down menu.

#### 4 Match value:

- The match value can be a constant (i.e. text, number, etc.) or a variable.

#### 5 Repository item:

- This column shows the repository item linked to the action, i.e. the UI element on which the validation is performed.

## Interpreting the validation

If written as a statement, the validation would read as follows:

If the attribute `BackColor` of the UI-element `PnlColourPanel` is **EQUAL** to the value `Green`,  
**THEN** the validation returns the value `True`.

## Image-based validation example

This chapter contains a simple tutorial for **image-based validation**. Before working with this example, make sure you're familiar with the → [basic concept of test validation](#).



### Screencast

The screencast “Basic image-based validation” walks you through the information found in this chapter.:

[Watch the screencast now](#)

## Download the sample solution

To follow along with this tutorial, download the sample solution file from the following link:  
[Sample Image Based Validation](#)

## Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `ImagebasedValidation.rxsln`



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

## Test definition

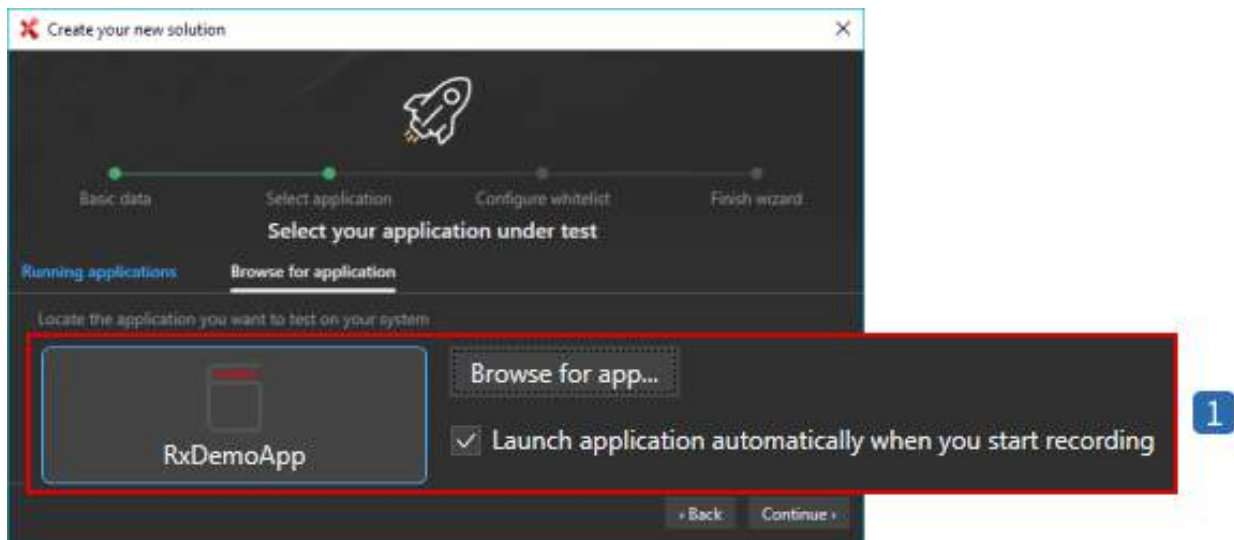
Before you start recording, let's define the test. It consists of 5 steps:

1. **Start** the demo application.
2. **Click** the **Image-based automation** tab.
3. **Make** the image visible.
4. **Verify** that the image is displayed.
5. **Exit** the Demo Application.

## Recording preparation

- 1 **Create** a desktop-test solution using the Solution Wizard and in step 2 of the setup, **select** the Demo Application as your AUT.





- 1 Demo App selected as AUT and is launched automatically when you start recording.
- 2 When the Solution Wizard has finished, **click** the **Recording1** tab in the Ranorex Studio working environment.

## Record the test: part one

### Hint

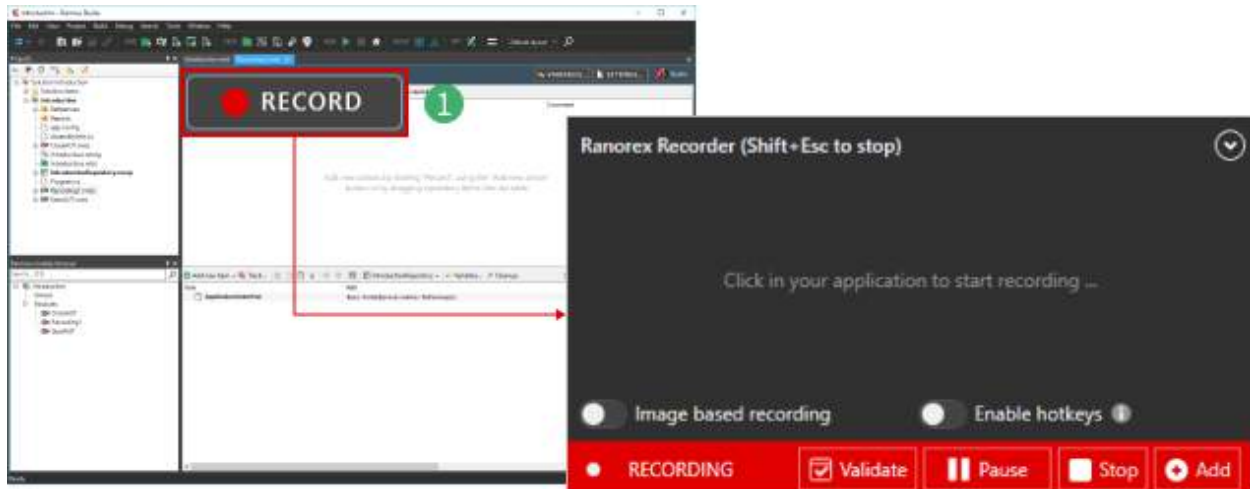
Remember, if you're not using whitelisting, all user interactions are captured once recording has started, even if they are not performed on the AUT.

- Click **Pause** to pause recording. Click **Continue** to resume recording.
- Click **Stop** to end recording.

Learn more about the Recorder control center in Ranorex Studio fundamentals > Ranorex Recorder > → [Recorder control center & hotkeys](#).

Read about whitelisting in Ranorex Studio fundamentals > → [Whitelisting](#).

- 1 In the recording module view of **Recording1**, click **RECORD**. Ranorex Studio is minimized to the taskbar automatically.  
The Recorder control center shows that recording is active.



- 2 The application under test comes into focus. Click the tab **Image-based automation**.
- 3 To make the image appear, click the checkbox **Show image**.

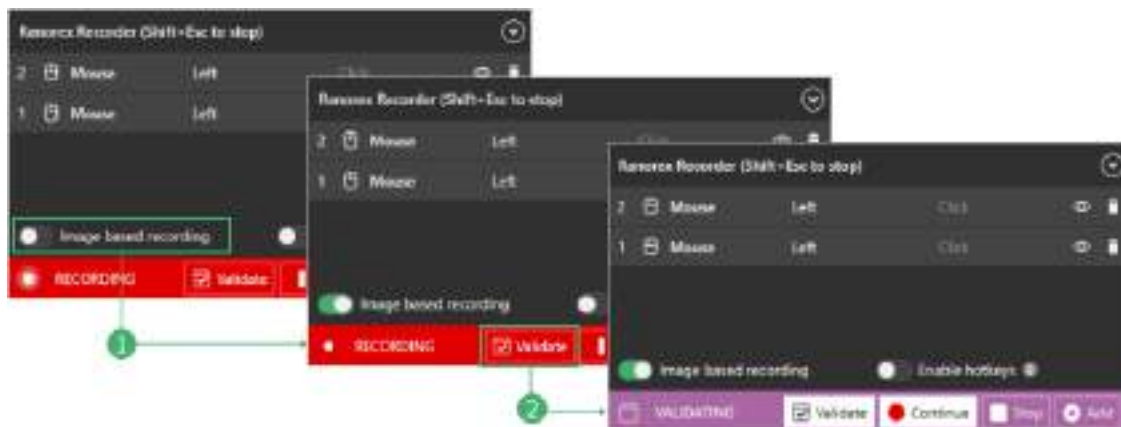
## Image-based validation

The goal of the example validation is to verify whether the image of the cat appears when the checkbox is selected. Since this requires us to validate an image, it's an **image-based validation**.

Follow the steps below:

### Activate image-based validation

- 1 In the Recorder controls, **activate** the switch **Image based recording**.
- 2 **Click Validate**. The Recorder pauses recording and switches to image-based validation mode.



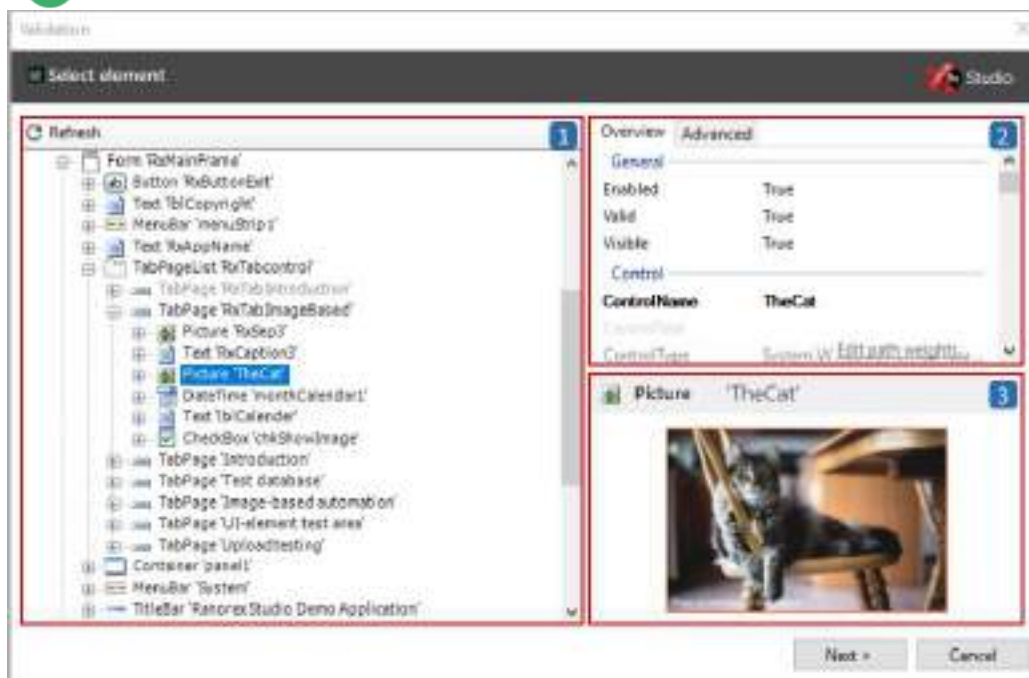
## Select validation element

### 3 Select the UI element to validate:

- **Mouse over** the image of the cat. A **purple frame** follows your mouse movement.
- The purple frame indicates which element is currently selected for validation.
- Once your selection matches the image, **click it**.

## Confirm validation element

### 4 To confirm the UI element, **click Next**.



### 1 UI element location within GUI

- The UI element tree represents the hierarchical GUI structure of the application.
- You can click a different UI element to select it for validation, if desired.

### 2 UI element states and attributes

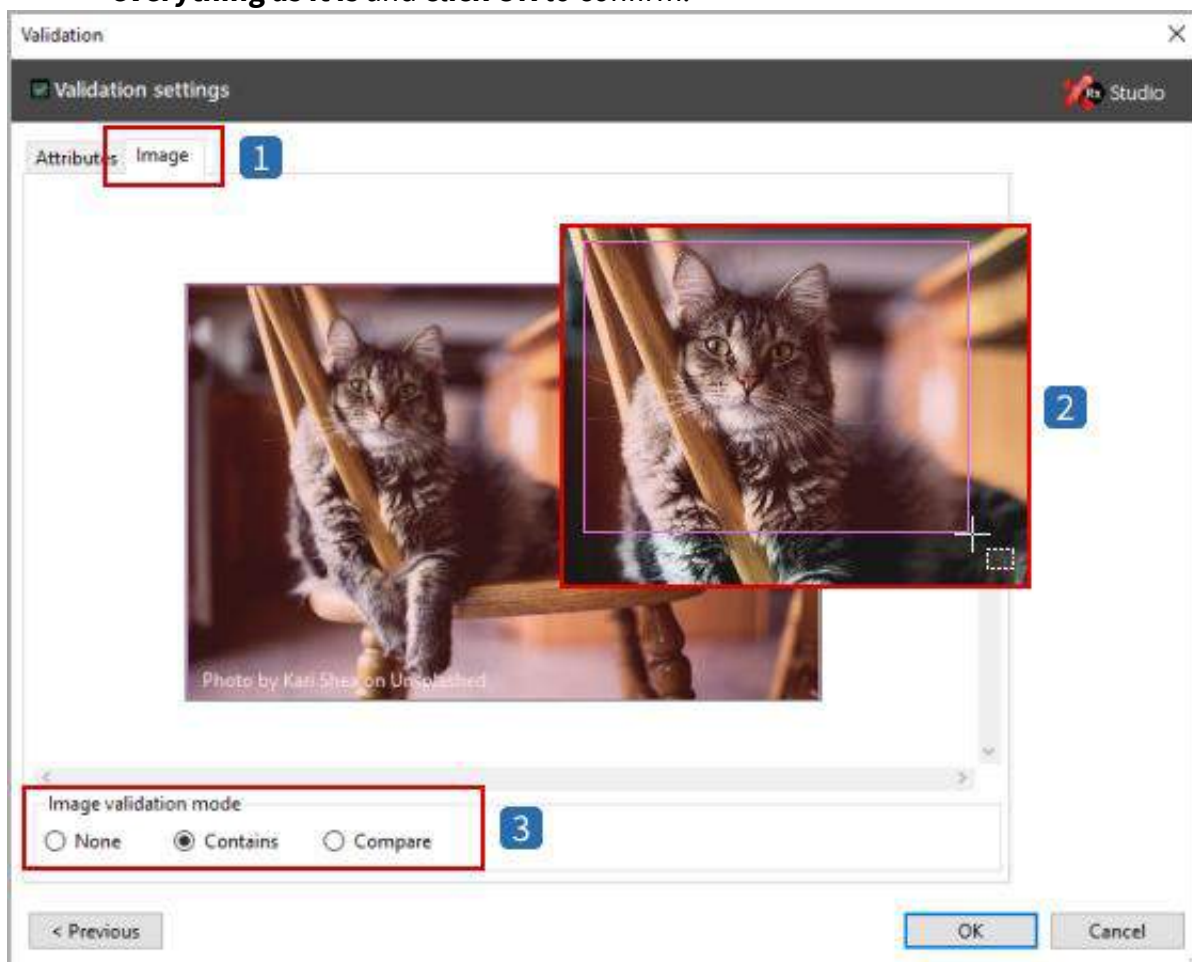
- All the attributes of the selected UI element appear in this area.

### 3 Screenshot of validation UI element

- Use the screenshot to quickly check whether you've selected the correct UI element.

## Define validation attributes

- 5 **Define** the image that will be used for the validation. For our example, simply **leave everything as it is** and **click OK** to confirm.

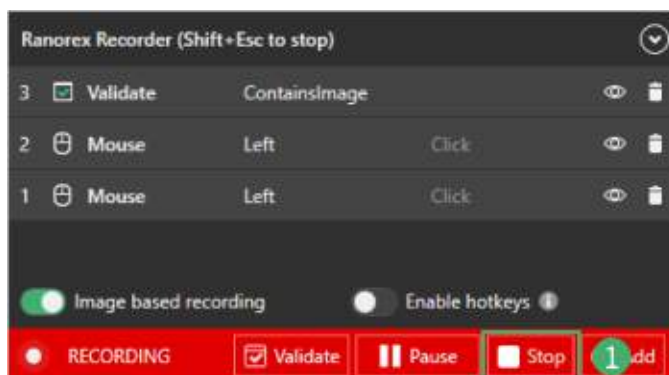


- 1 The **Image** tab is selected in the validation dialog instead of the **Attributes** tab.
- 2 Draw a rectangle to select a part of the image when using the “Contains” validation mode.
- 3 **Different image validation modes are available:**
  - **None:** deactivates image validation.
  - **Contains:** determines whether the validation image is contained in the actual image found during the test.
  - **Compare:** determines whether the validation image and the actual image found during the test are the same.

## Finish recording

Once you’ve finished the validation action, Ranorex automatically continues recording. The next step is to end the test recording.

- 1 In the Recorder control center, **click Stop** to end the recording.



## Result

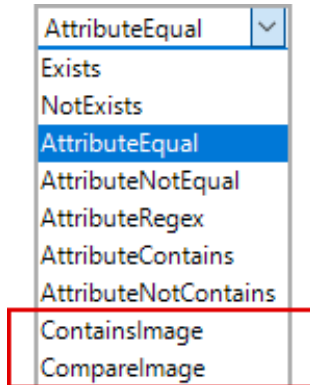
After recording stops, you are returned to Ranorex Studio. You’ll see the actions table with three recorded actions. **Action #3** is the validation action.

+ Add new action ▾ ✂ 📄 🗑️ ↺ 🔍 Turbo mode 🖼️ Screenshot					
#	Action		Screenshot name		Repository item
1	Mouse	Click	Left	Relative	ImageBasedAutomation
2	Mouse	Click	Left	Relative	ChkShowImage
3	Validate	ContainsImage ▾	Screenshot1	...	TheCat

1
2
3

## 1 Validation type operator

- This operator identifies the type of validation to perform.
- There are nine different validation type operators.



### Further reading

A detailed description of all validation match operators can be found in > Ranorex Studio fundamentals > Actions > [Action properties](#).

2 **The validation screenshot** to which the actual image found during testing will be matched.

3 **The repository item** linked to the action, i.e. the UI element on which the validation is performed.

## Interpreting the validation

If written as a statement, the validation would read as follows:

If Screenshot1 is contained (**ContainsImage**) in the UI-element referenced by the repository item **TheCat**, THEN the validation returns the value '**True**'.

## Validation of tooltips

Validating tooltips is a bit more challenging than validating other types of UI elements. In this chapter, you'll learn how to do it.

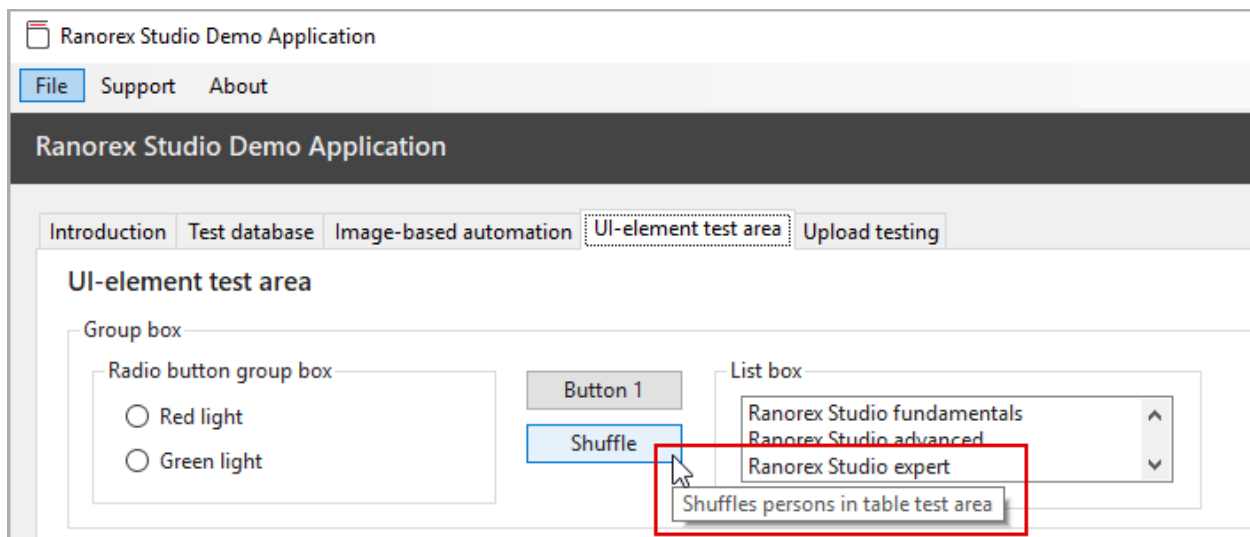
## Screencast

The screencast “Tooltip validation” walks you through the information found in this chapter.

[Watch the screencast now](#)

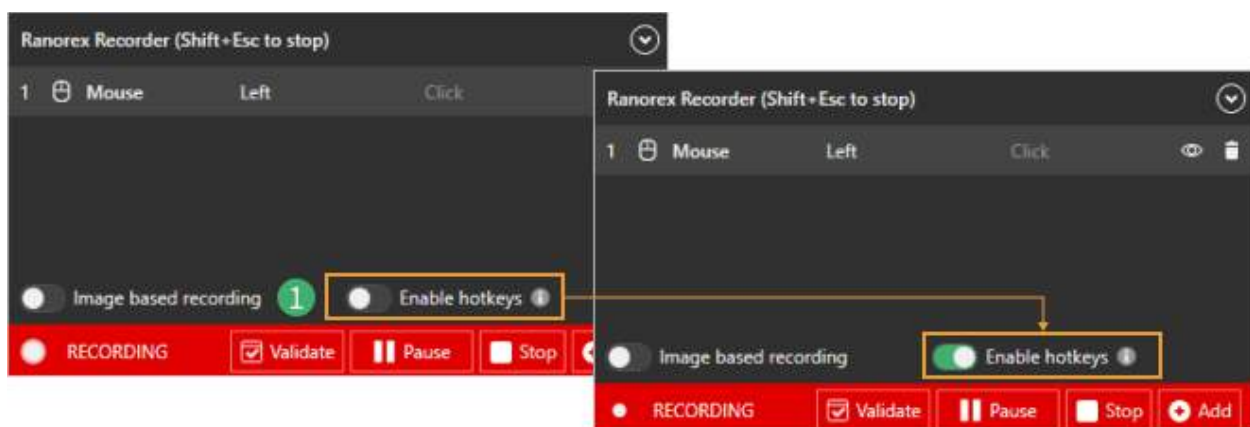
## Test definition

We want to validate that the tooltip shown in the image below appears properly when a user hovers the mouse over the **Shuffle** button.



## Activate tooltip validation

To activate tooltip validation:



## 1 Activate Enable hotkeys.

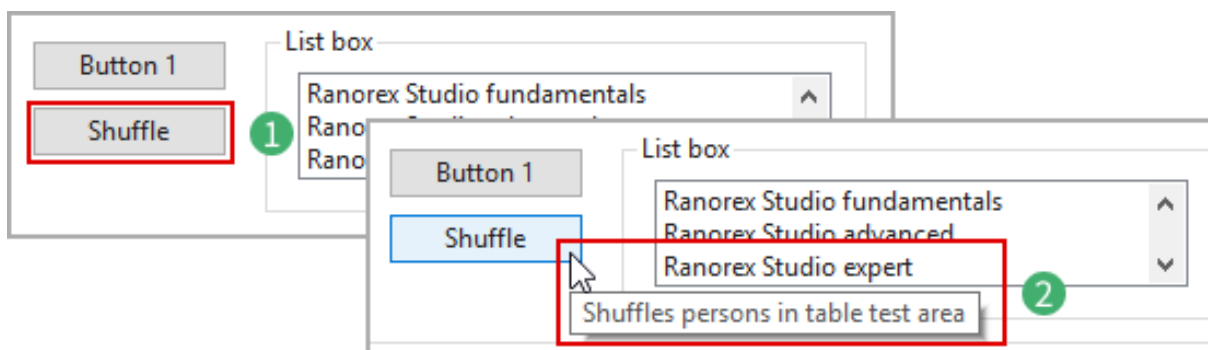


### Further reading

Recorder hotkeys are explained in Ranorex Studio fundamentals > Ranorex Recorder> → [Recorder control center & hotkeys](#).

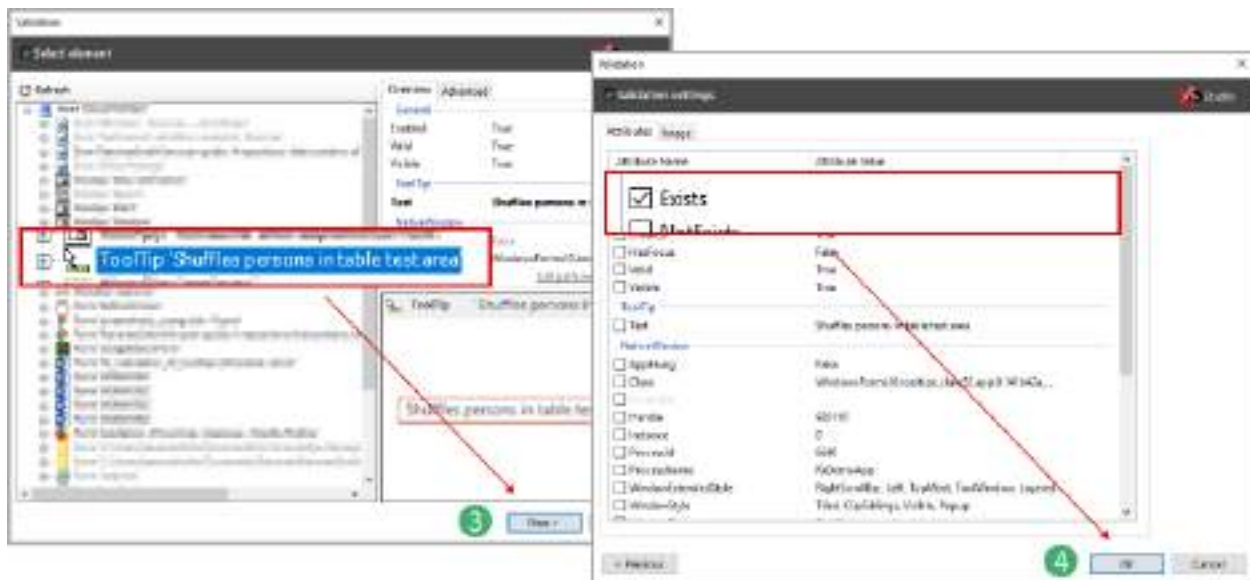
## Validate the tooltip

- 1 **Place** the cursor over the UI element whose tooltip you want to validate.
- 2 When the tooltip appears, **press** **T**.



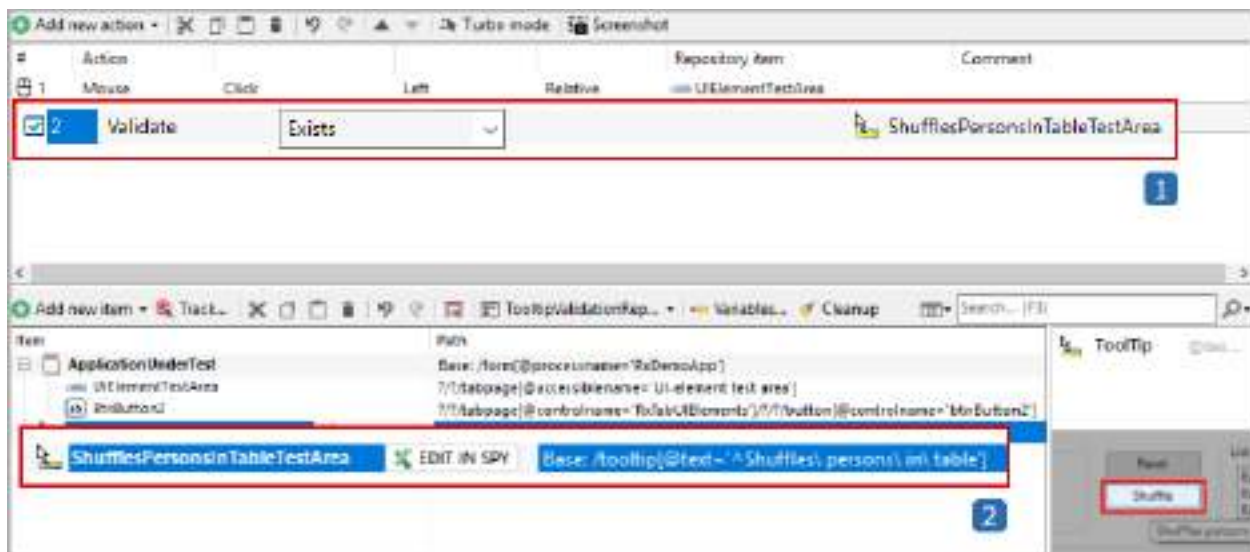
- 3 In the next screen, **check** whether the correct tooltip element has been selected and **click Next** to confirm.
- 4 **Ensure** the attribute **Exists** is checked and **click OK** to confirm.





## Result


There are two recorded actions in the actions table. **Action #2** is the tooltip validation.



- 1 Actions table and tooltip validation action.
- 2 Linked repository item representing the tooltip UI element.

## Download the sample solution

You can download the sample solution for this chapter here. It contains the finished solution with all recorded actions.

 **Sample solution**

---

**Theme:** Tooltip validation  
**Time:** 15 minutes

Download sample file

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `TooltipValidation.rxsln`



#### Hint

The sample solution is available for Ranorex Studio versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher

# Whitelisting

When identifying UI elements, Ranorex Studio and its components scan all running processes on your computer by default. With whitelisting, you can **focus Ranorex Studio** on only **the processes relevant for your test**.

This has two advantages:

- It **increases performance** during test recording, test execution, and in Ranorex Spy
- It **helps you create clean recordings** since you can interact only with whitelisted processes.



## Hint

Plan ahead which processes will be part of your test and add them to your whitelist.

## Editing your whitelist

In Ranorex Studio, **go** to **View > Whitelist**. The whitelist pad opens.

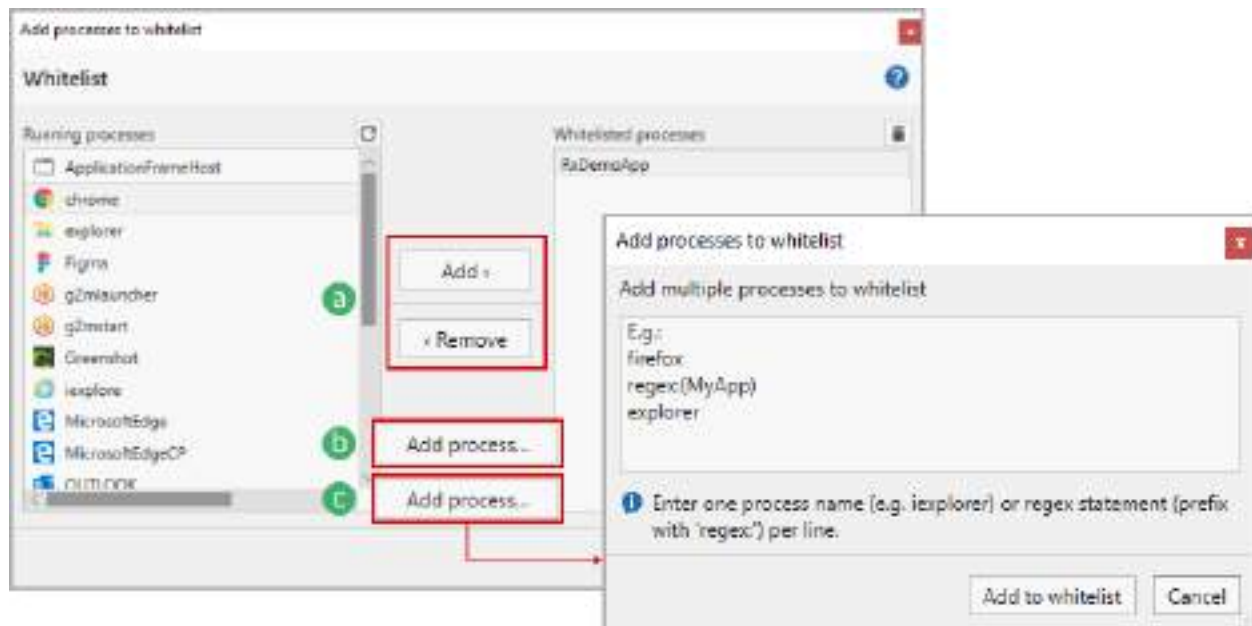
**Click Edit whitelist...** if your whitelist is empty or **click** the whitelist icon in the top right of the pad.

The edit window appears. In it:

**Add** processes to the whitelist or **remove** them using the buttons.

If a process isn't listed on the left, **click Add process...** to browse to it and add it directly.

**Click Advanced...** to open a dialog where you can paste a list of process names or specify a range of process names with a regex statement.



When you're done, **click Apply whitelist**. Ranorex Studio and its components will now recognize UI elements only in the whitelisted processes.



### Note

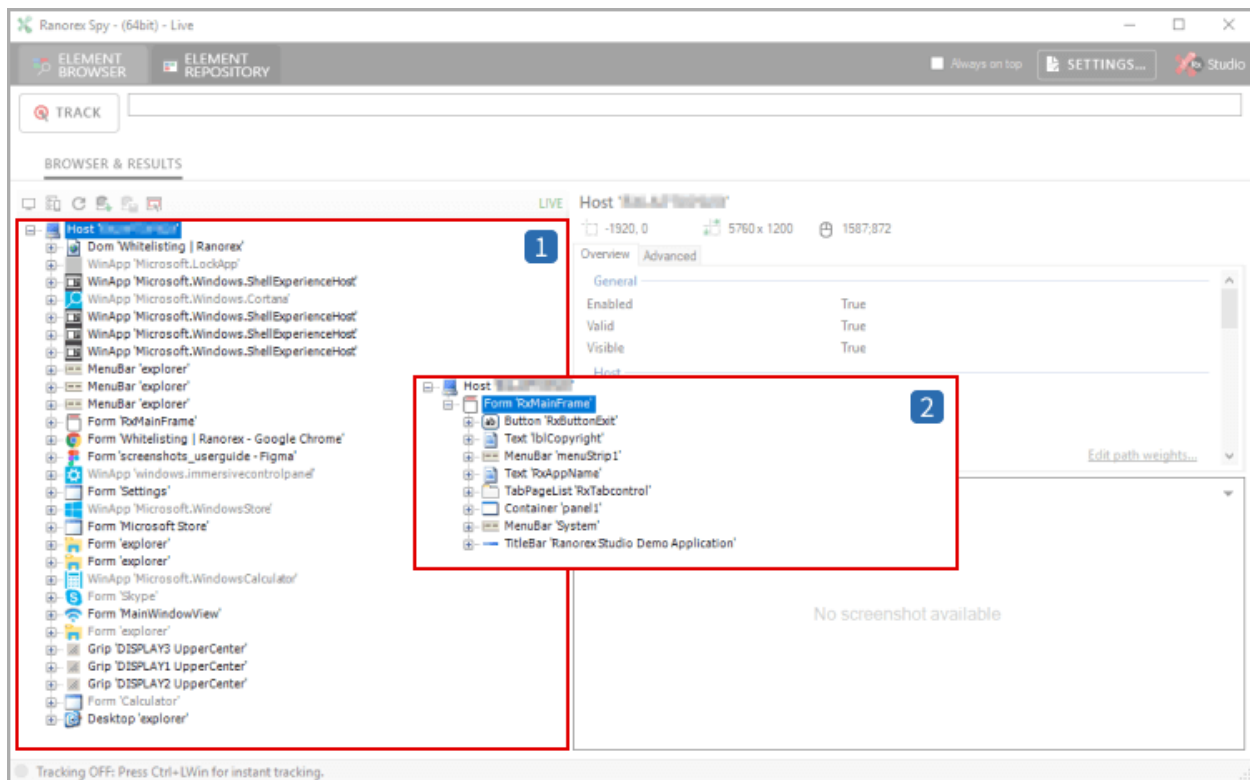
Endpoints are not affected by whitelisting, and so will still appear in Spy.



### Reference

Find out more about endpoints in  
Web and mobile testing > Endpoints > [Getting started](#)

## Result



1

Spy browser window **without whitelisting**

2

Spy browser window **with active whitelisting** and the Ranorex Studio Demo Application in the whitelist

# Ranorex Coach

The Ranorex Coach helps you stick to best practices in designing your tests. It does so by giving you short, non-intrusive hints when it detects certain behaviors (its triggers).

These hints contain tips how to improve your test and can help you avoid issues in the future.

On this page, you'll find out how the Coach works, how you can enable/disable it or specific triggers, and you'll find short descriptions of the hints it gives.



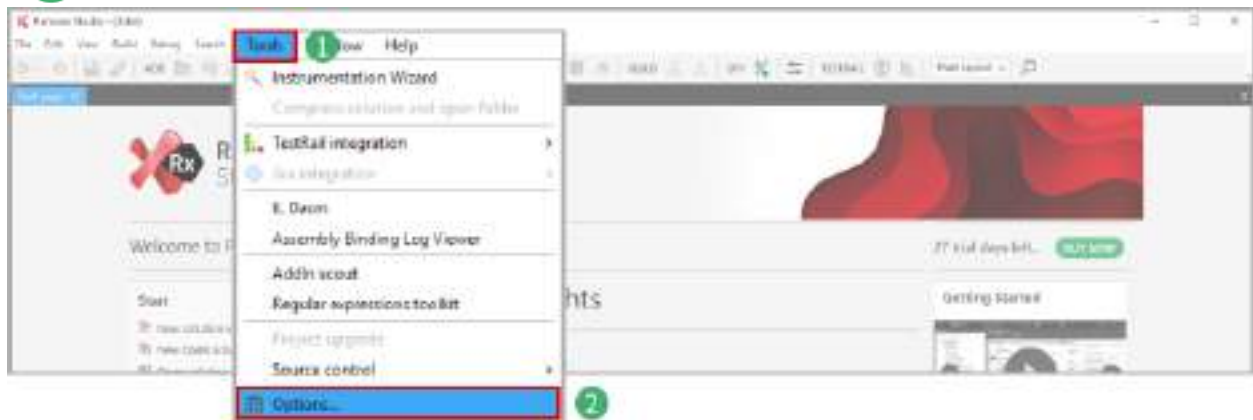
## Reference

For more best practices in designing your test structure, also take a look at Hands-on application topics > → [Structure your tests](#).

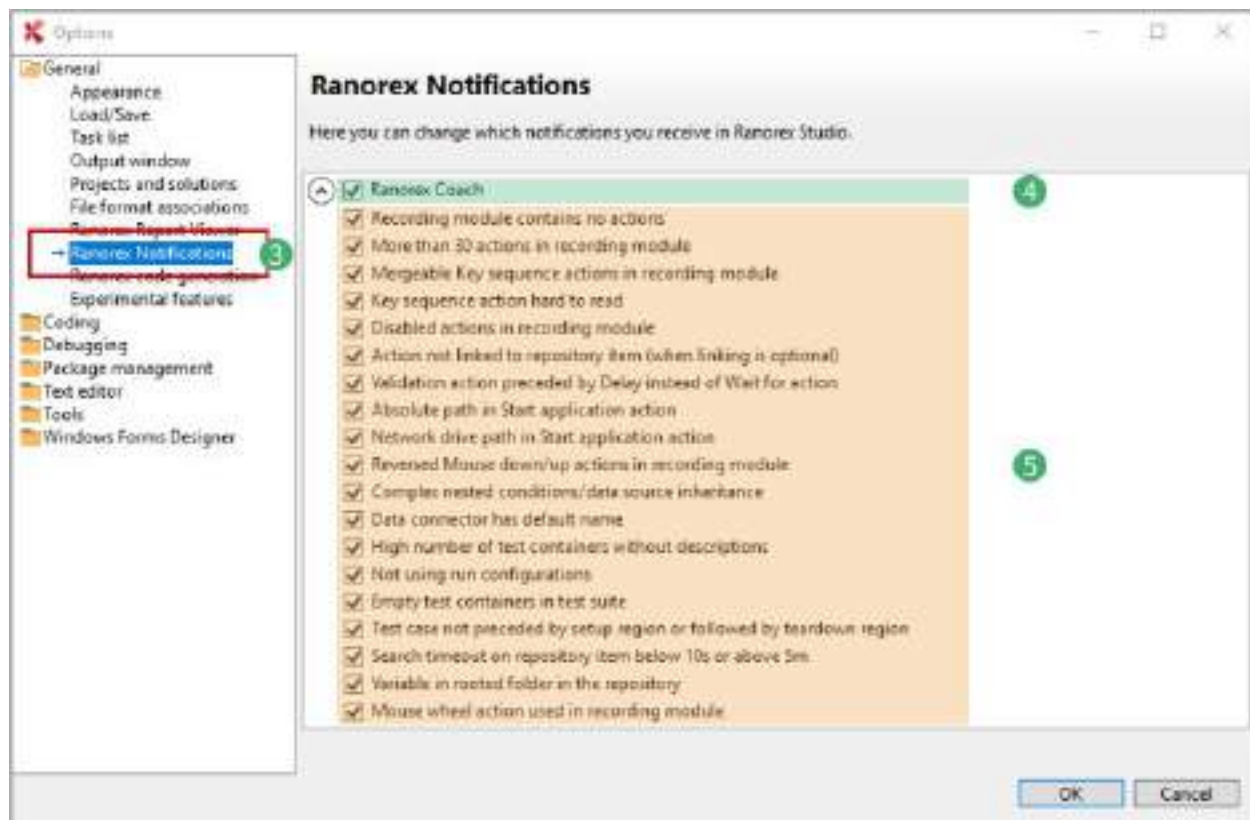
## Enable/disable the Coach

By default, the Coach is enabled, including all hints. You can disable the Coach as a whole or only specific hint. To enable/disable:

- 1 In Ranorex Studio, **click Tools**.
- 2 **Click Options**.



- 3 **Click Ranorex Notifications**.
- 4 **Click** to enable/disable the Coach entirely.
- 5 **Click** to enable/disable specific hints. See further below for a short explanation of all hints.



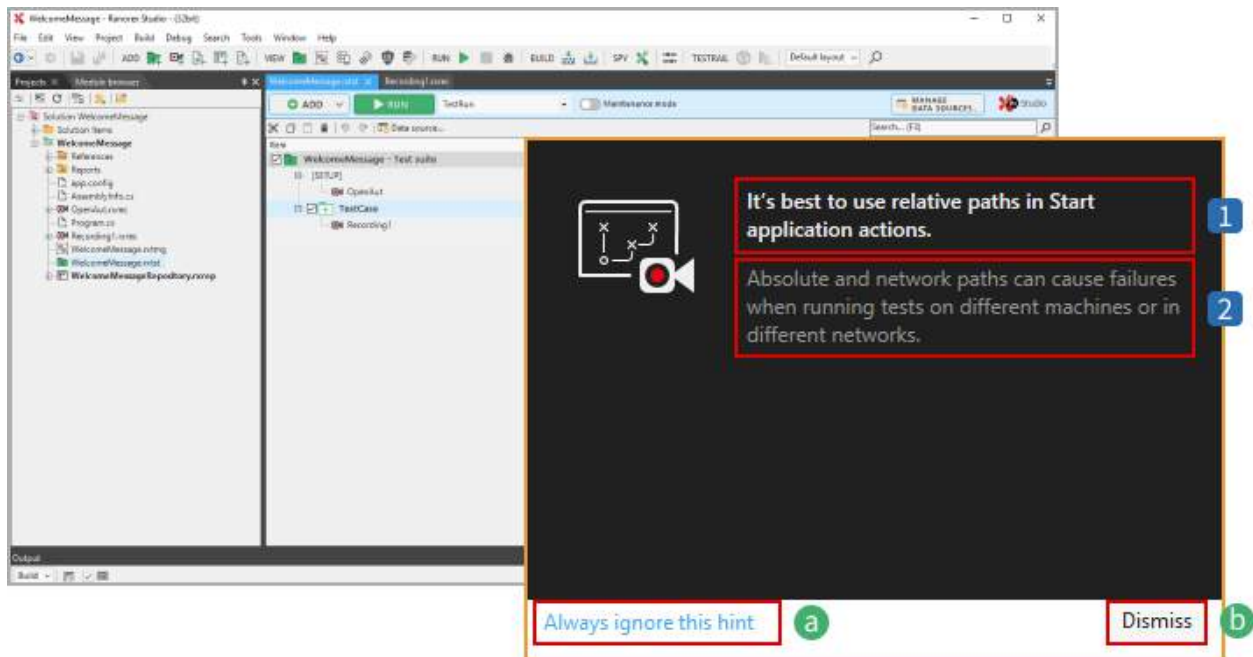
## How it works

When the Coach detects a trigger, it displays a hint. The Coach aims to be helpful without being intrusive. This is why:

- It only checks for triggers and displays a hint when you save or close a file
- It only shows one hint, even if multiple triggers are detected
- It won't show a new hint until you've dismissed an existing one and saved/closed a file again

Coach hints are displayed as a small popup window in the bottom right of Ranorex Studio, where they're least likely to be in the way. You can close them in two ways:

- To close a hint and deactivate its specific trigger, **click Always ignore this hint.**
- To only close a hint, **click Dismiss.**



- 1 Short best-practice suggestion
- 2 More information on the topic, e.g. why it's a best practice

## Test suite hints

### Complex nested conditions/data source inheritance

<b>Trigger:</b>	The test suite contains conditions or data source inheritance that extend over too many test container levels.
<b>What to do:</b>	Keep it simple. Use conditions sparingly and don't use multiple data source over several nested test container levels.
<b>More information:</b>	→ <a href="#">Best practices for structuring</a>

### Empty test containers in test suite

<b>Trigger:</b>	The test suite contains at least one empty test container.
<b>What to do:</b>	Fill empty test containers with modules or delete them if they're no longer needed.



## Test case not preceded by setup region or followed by teardown region

<b>Trigger:</b>	The test suite contains at least one test case that isn't wrapped in at least one setup/teardown region.
<b>What to do:</b>	Make sure your test cases are wrapped in at least one setup/teardown region.
<b>More information:</b>	→ <a href="#">The test suite structure</a>

## High number of test containers without descriptions

<b>Trigger:</b>	Fewer than half of your test containers have descriptions.
<b>What to do:</b>	Give all your test containers good descriptions.

## Data connector has default name

<b>Trigger:</b>	At least one data connector still has its default name, e.g. „NewConnector“.
<b>What to do:</b>	Give all data connectors good names.

## Not using run configurations

<b>Trigger:</b>	You have a test suite that has deselected test containers, but only one (likely the default) saved run configuration.
<b>What to do:</b>	Create and use run configurations if you want to regularly run your test with different test containers selected/deselected.
<b>More information:</b>	→ <a href="#">Execute a test suite</a>

## Recording module hints

### Recording module contains no actions

<b>Trigger:</b>	At least one recording module contains no actions..
<b>What to do:</b>	Fill empty recording modules with actions or delete them if they're no longer needed.

## More than 30 actions in a recording module

<b>Trigger:</b>	At least one recording module contains too many actions
<b>What to do:</b>	Keep recording modules focused on a particular task and only include actions that are necessary for it.

## Mergeable Key sequence actions

<b>Trigger:</b>	At least one recording module contains subsequent Key sequence actions that look like they should be merged.
<b>What to do:</b>	Merge Key sequence actions that are fragments of a larger string.

## Key sequence action hard to read

<b>Trigger:</b>	At least one recording module contains Key sequence actions that contain non-text input.
<b>What to do:</b>	Remove unnecessary non-text input or simply activate Keystroke optimization.
<b>More information:</b>	<a href="#">→ Action properties</a>

## Disabled actions in recording modules

<b>Trigger:</b>	At least one recording module contains disabled actions.
<b>What to do:</b>	Unless you're troubleshooting, activate disabled actions or delete them if they're no longer needed.

## Action not linked to repository item

<b>Trigger:</b>	At least one action isn't linked to a repository item when it could be.
<b>What to do:</b>	Link actions to repository items, even when optional.

## Validation action preceded by Delay instead of WaitFor

<b>Trigger:</b>	Using a fixed Delay action before a Validation action to ensure a UI element can be found.
<b>What to do:</b>	Use a flexible Wait for action instead.
<b>More information:</b>	<a href="#">→ Action properties</a>

## Absolute path in Start application action

<b>Trigger:</b>	A Start application action contains an absolute path.
<b>What to do:</b>	Use a relative path.

## Network drive path in Start application action

<b>Trigger:</b>	A Start application action points to a network drive.
<b>What to do:</b>	Use a relative path that points to a local drive.

### Reversed mouse down/up actions

<b>Trigger:</b>	At least one recording contains Mouse up/down actions in reverse order.
<b>What to do:</b>	Ensure Mouse up/down actions are in correct order or use Mouse click action instead.

### Mouse wheel action used in recording module

<b>Trigger:</b>	At least one recording module contains a Mouse wheel action.
<b>What to do:</b>	Keyboard and Mouse actions usually scroll automatically. If this doesn't work, try an Invoke action with the 'EnsureVisible' or 'Focus' argument. Alternatively, set the 'Use Ensure Visible' property to 'True' on repository items.
<b>More information:</b>	<a href="#">→ Action properties</a> <a href="#">→ Manage repository items</a>

### Repository hints

#### Search timeout too low or too high

<b>Trigger:</b>	At least one repository item has a search timeout that is below 10 seconds or over 5 minutes.
<b>What to do:</b>	Keep the search timeout between the above numbers. 30 seconds is often a good value.

#### Variable in rooted folder in the repository

<b>Trigger:</b>	At least one rooted folder in your repository contains a variable in its RanoreXPath.
<b>What to do:</b>	Avoid variables in rooted folder paths.

# Reporting

Every test run in Ranorex Studio ends with a report. After all, how else would you know whether the test run was successful or not? The report details the entire test run from beginning to start, i.e. how many test cases succeeded, failed, or were blocked; what errors and warnings were raised; the iterations of a given test container; and so on. You can customize reports to change what information is displayed or add your own logos, for example. Finally, you can also export reports.

## Basic characteristics

- The report starts collecting data as soon as a test run starts.
- → [Report levels](#) control what data is included in the report.
- The report is generated as the test run progresses and you can view it at any time during the run. This is especially useful for very large test suites that take hours to run.
- Reports can be customized.

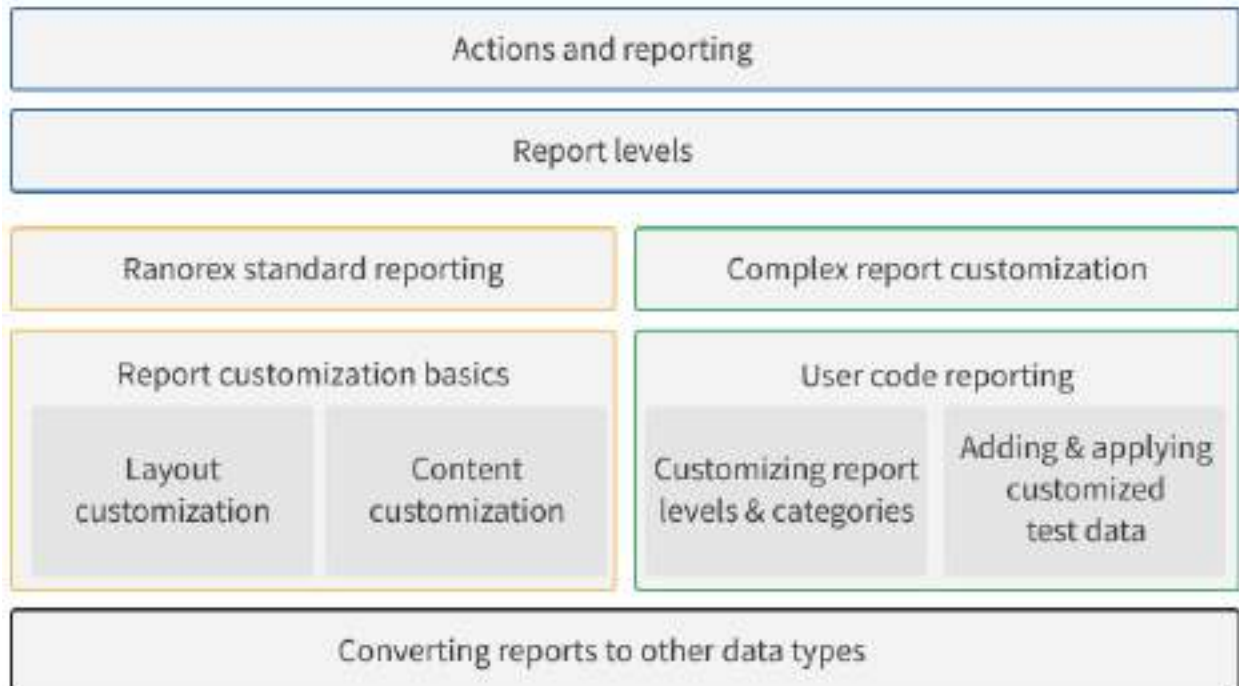


### Further reading

These basic characteristics are explored in Ranorex Studio fundamentals > Reporting → [Ranorex standard reporting](#), → [Report levels](#)., and → [Customization basics](#).

## Chapter structure

Reporting is a fairly complex topic, which is why we've outlined the structure of this chapter below.



## Actions and the report

In this chapter, you'll learn how actions executed during a test run are represented in the report.



### Screencast

The screencast “Actions and the report” walks you through the information found in this chapter.

[Watch the screencast now](#)

## How actions are reported

By default, every action executed during a test run triggers a report message.

Test container filter: ☒ Success ☒ Failed ☒ Blocked

▼ Test Case 7.42s

▶ Recording! 7.38s

Filter: ☒ Info ☒ Success

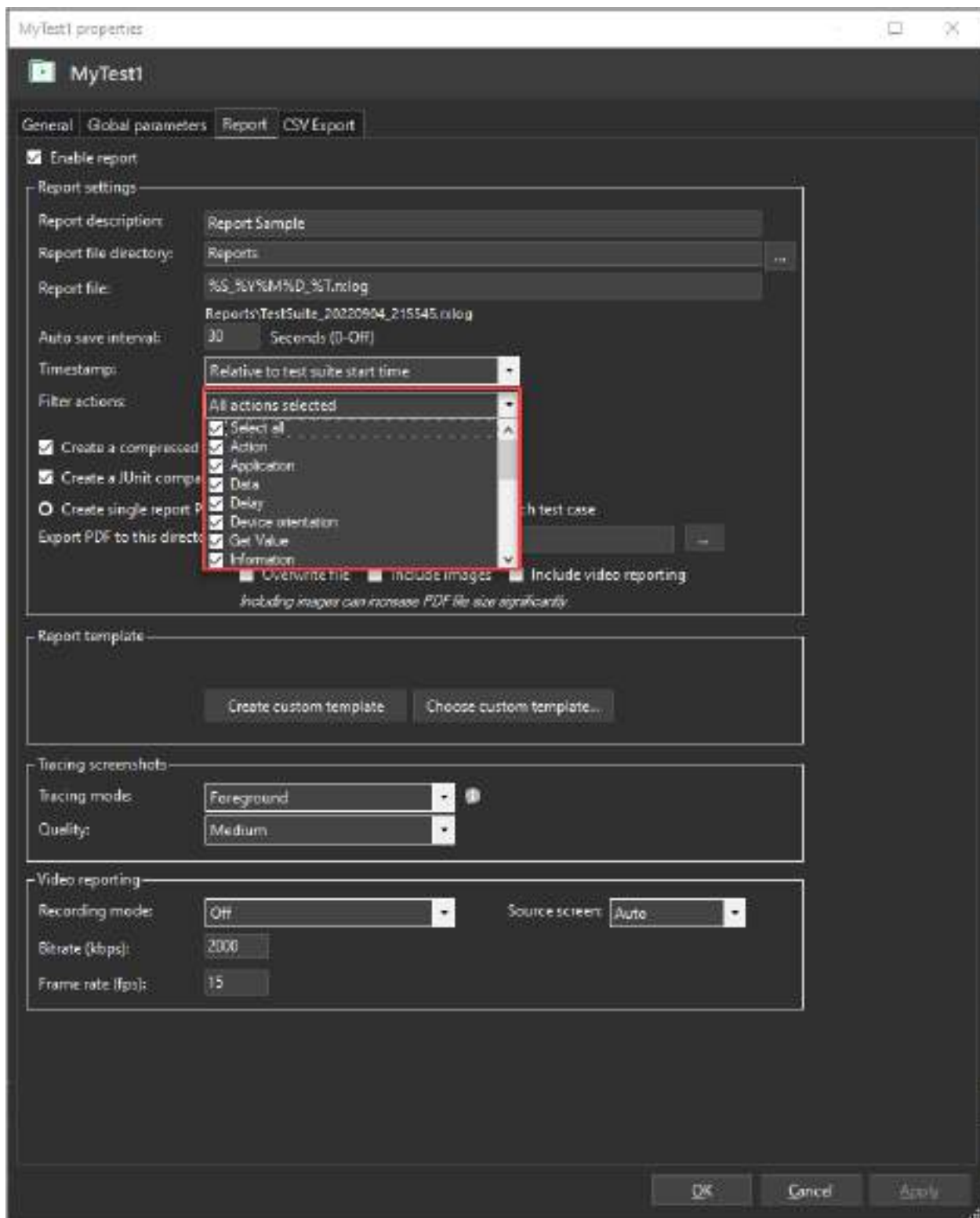
Time	Level	Category	Message
00:00.805	Info	Application	Run application 'C:\Users\pissonek\Documents\Sanoma\SanomaStudio\Projects\Sample\RxDemoApplication\RxDemoApplication.exe' with arguments "" in normal mode.
00:01.158	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.EnterYourName' at 24;10.
00:04.502	Info	Keyboard	Key sequence 'Harry' with focus on 'RxMainFrame.RxTabIntroduction.EnterYourName'.
00:05.185	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.BtnSubmitHarryName' at 40;10.
00:06.230	Info	Section	Validation action next.
00:06.277	Info	Validation	Validating AttributeEqual (Text: 'Welcome, Harry!') on item 'RxMainFrame.RxTabIntroduction.LblWelcomeMessage'.
00:06.437	Success	Validation	Attribute 'Text' of element for item 'IntroductionRepository.RxMainFrame.RxTabIntroduction.LblWelcomeMessage' does match the specified value.
00:06.580	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.Reset' at 11;6.
00:07.417	Info	Mouse	Mouse Left Click item 'RxMainFrame.BtnQuitTest' at 46;10.

## 1 Actions reported during a test run

As you can see in the image above, **one action means at least one report message**, and sometimes more. In large test suites that have thousands of actions, the report can get cluttered very quickly. To prevent this, you can use → [report levels](#) to control which report messages make it into the report.

You can reduce the number of actions recorded in a Report at the Test Suite level by using the **Filter Actions** in the Report **Properties**.

In the **Filter Actions** drop-down select which **Actions** you want to be recorded in the test suite report. By default, all actions are selected.



You can also completely disable reporting for actions through their properties.



## Reference

Disabling reporting for actions is explained in:

>Ranorex Studio fundamentals >Actions >→ [Manage actions](#)

## What actions look like in the report

Actions can be separated into four groups in terms of how they appear in the report:

- Failed or successful validations
- Separators
- User-defined log messages
- All other actions

## Validations

Validation report messages always have two parts. The first part has the report level **Info** and is just a notification that a validation has been performed.

The second, more interesting part, tells you whether the validation succeeded or failed.



1

### Successful validation

A successful validation has the report level **Success** and is printed in green by default.

2

### Failed validation



A failed validation has the report level **Failure**, has a red background, and usually comes with two screenshots to make it easier to find out what went wrong.



## Reference

To learn more about validations, refer to the chapters:

Ranorex Studio fundamentals > Actions > → [Action properties](#) and Ranorex Studio fundamentals > → [Test validation](#)

## Separators

Separators are useful for visually structuring actions in the actions table and the report.

#	Action	Object	Click	Relative	Buttons
1	Run application	C:\Users\... \bin\B...			
2	Mouse	Click	Left	Relative	EnterYourName
3	Key sequence	Harry			EnterYourName
4	Mouse	Click	Left	Relative	BtnSubmitUserName
Validation action next					
5	Validate	AttributeEqual	Test	Welcome, Harry!	UIWelcomeMessage
6	Mouse	Click	Left	Relative	Reset
7	Mouse	Click	Left	Relative	BtnButtonExit

Time	Level	Section	Message
00:05:158	Info	Mouse	Mouse Left Click now: 'BtnSubmitUserName' at (8, 10).
00:05:302	Info	Keyboard	Key sequence 'Harry' with focus on 'BtnSubmitUserName' at (8, 10).
00:05:305	Info	Mouse	Mouse Left Click now: 'BtnSubmitUserName' at (8, 10).
00:05:230	Info	Section	Section: Validation action next
00:05:231	Info	Validate	Validating attribute 'Text' on 'UIWelcomeMessage'.

1

A separator in the actions table with the header **Validation action next**.

2

In the report, the separator has the report level **Info** and its message displays the header defined in the actions table.



## Further reading

For more information on separators, refer to

Ranorex Studio fundamentals > Ranorex Recorder > [Managing recording modules](#) and Ranorex Studio fundamentals > Actions > [Action properties](#)

## User-defined log messages

With the Log message action, you can pass a custom message with a custom report level to the report.



1 Log message in the actions table.

2 The same log message in the report.

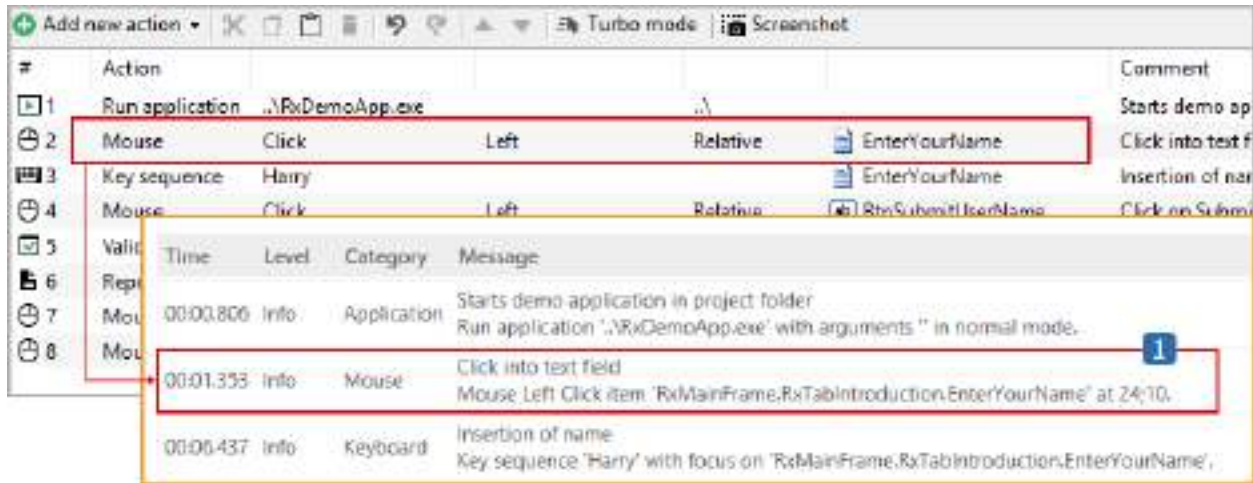


## Further reading

For more information on the Log message action, refer to Ranorex Studio fundamentals > Actions > [Action properties](#)

## All other actions

All other actions have the report level **Info** and their message describes what they are doing. If such an action fails, a separate message with the report level **Failure** will be passed to the report, in the same way as described further above for validations.



The screenshot shows a test runner interface with a list of actions and a detailed report for a mouse click action. The actions list includes:

#	Action	Path	Coordinates	Relative	Target	Comment
1	Run application	..\\RxDemoApp.exe				Starts demo ap
2	Mouse	Click	Left	Relative	EnterYourName	Click into text f
3	Key sequence	Harry			EnterYourName	Insertion of nar
4	Mouse	Click	Left	Relative	Tab1.RxSubmitUserName	Click on Subm

The detailed report for the mouse click action (Action 2) is shown below:

Time	Level	Category	Message
00:00.806	Info	Application	Starts demo application in project folder
			Run application '..\\RxDemoApp.exe' with arguments '' in normal mode.
00:01.353	Info	Mouse	Click into text field Mouse Left Click item 'RxMainFrame,RxTabIntroduction,EnterYourName' at 24;10.
00:00.437	Info	Keyboard	Insertion of name Key sequence 'Harry' with focus on 'RxMainFrame,RxTabIntroduction,EnterYourName'.

- 1 The Mouse click action has the report level **Info** and its text describes that a left click was performed on the specified repository item at the specified screen coordinates.

## Report levels

Report levels categorize events that happen during a test run and govern what information is included in the report. When your tests contain hundreds of test cases with thousands of modules, report levels are the key to striking the balance between detail and brevity. In this chapter, you'll find out how report levels work and how to use them.

### Screencast

The screencast “Report levels” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Download the sample solution

The explanations in this chapter are based on a sample solution that you can download here: [Sample Introduction](#)

## Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `Introduction.rxsln`



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

## The default report levels

Ranorex Studio comes with 6 default report levels, each with a corresponding color and integer value.



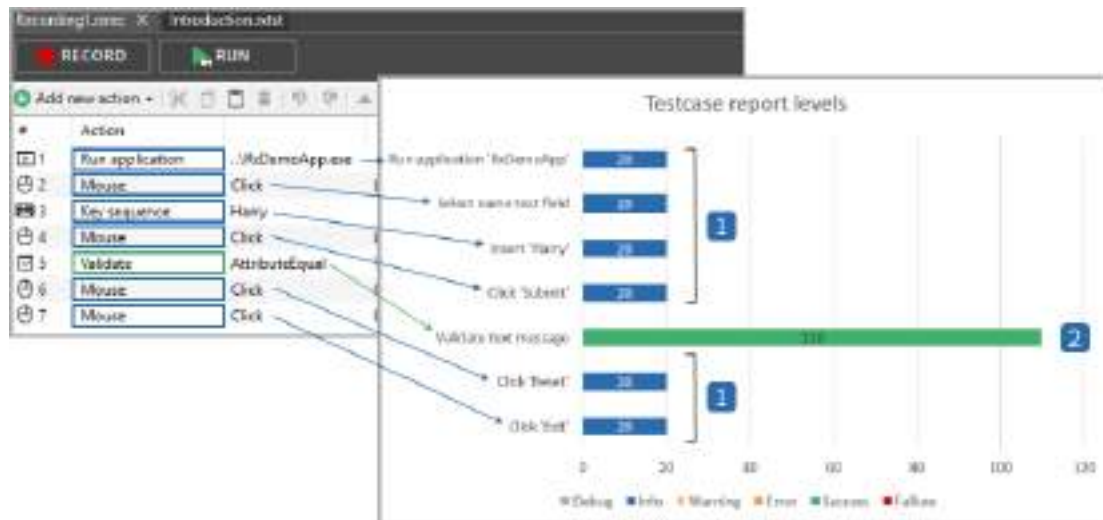
- The higher the integer value of a report level, the **more important it is**, i.e. a **Debug** notification is less important than an **Info** notification.
- Recording modules use the default report levels.
- In code modules, you can → [define your own report levels](#).

## The purpose of report levels

In a test report, **one executed action means at least one report message**. In large test suites with thousands of actions, the test report can quickly become cluttered. With report

levels, you can **control which messages are included in the report**. Report levels categorize report messages by importance and act as a filter.

The image below illustrates this.



- 1 Actions #1 through #4 and #6 through #7 have the report level **Info** (value = 20).
- 2 A validation always either succeeds or fails, and it's usually crucial to include this in a test run. Therefore, a **successful test validation** (action #5) has the report level **Success** (value = 110). A validation that **fails** has the report level **Failure** (value = 120).

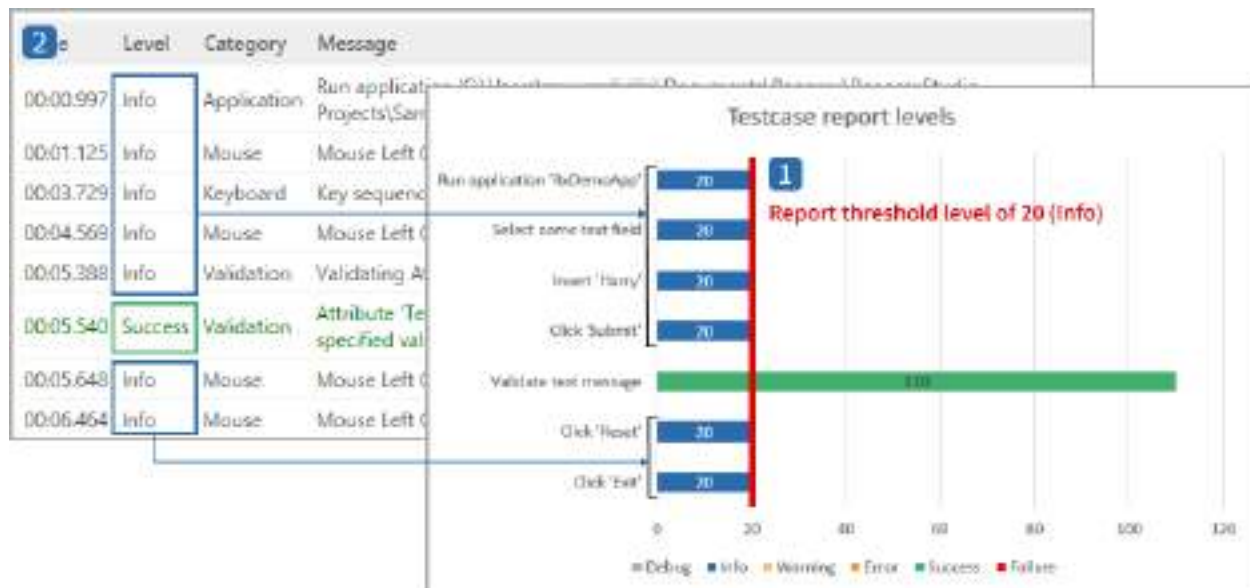
## Report level threshold

The report level threshold is the integer value that defines the minimum report level a message must have to be included in the report. If the report level is below this value, the message will not appear in the report.

The report level threshold is set in the test suite for structure items, i.e. the test suite item, test cases, and smart folders.

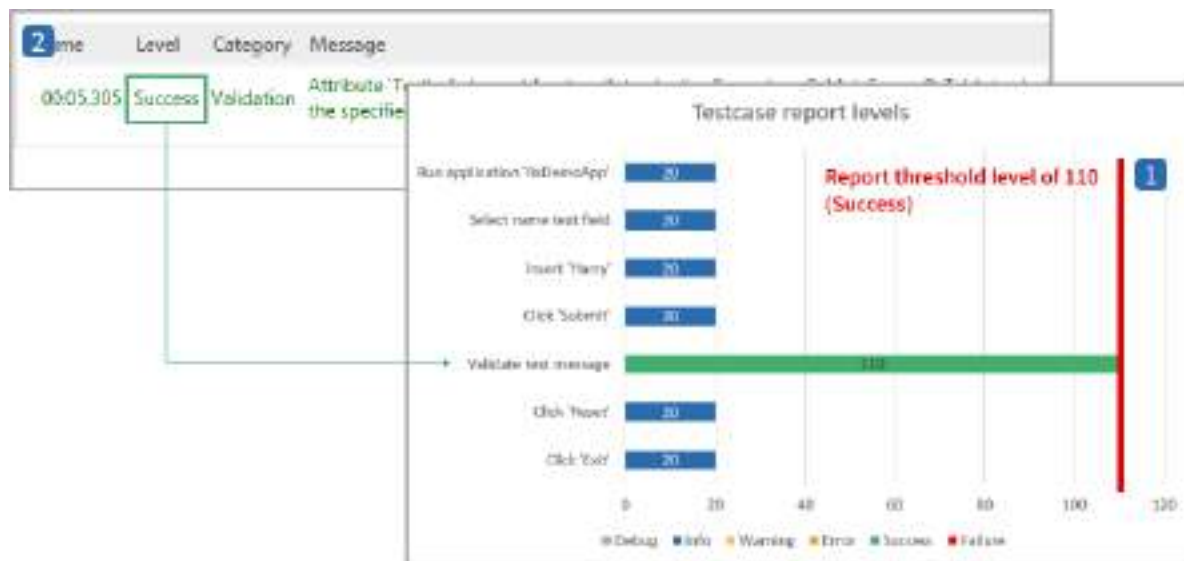
The images below illustrate this.

## Report threshold level of 20 (Info)



- 1 Threshold set to **Info** (value = 20).
- 2 In the report, all messages with a minimum report level of **Info** are included.

### Report level threshold of 110 (Success)

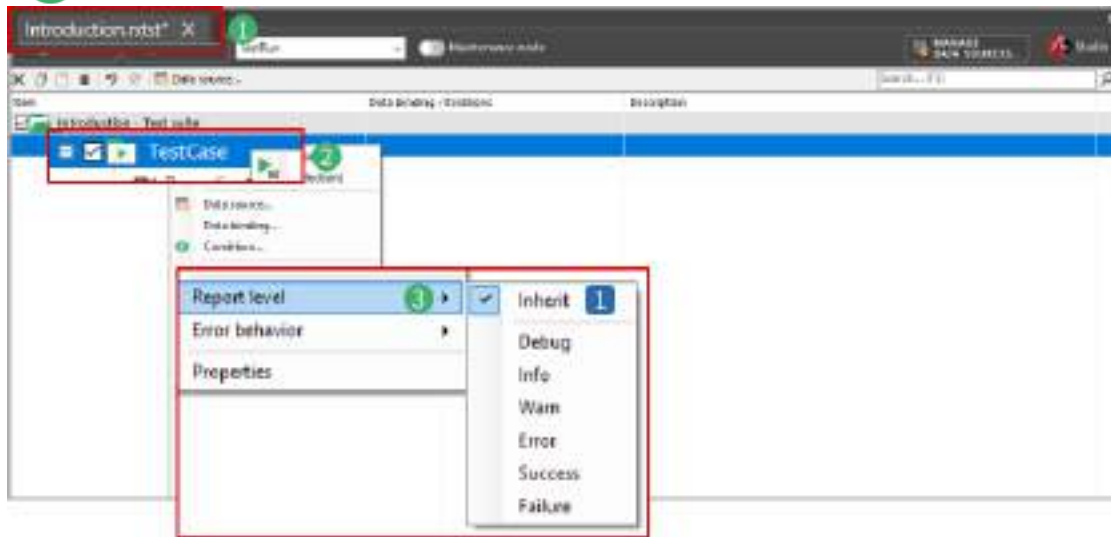


- 1 Threshold set to **Success** (value = 110).
- 2 In the report, all messages with a minimum report level of **Success** are included. In this case, that's only the message notifying you of a successful validation.

### Set report level threshold

The report level threshold is set in the test suite for structure items, i.e. the test suite item, test cases, and smart folders. By default, structure items inherit the setting from their direct parent.

- 1 **Change** to the test suite view.
- 2 **Right-click** the test suite, a test case, or a smart folder.
- 3 **Click Report level** and select the desired value.



## 1 Inherit

When this option is selected, the setting is inherited from the direct-parent structuring item.

## Example

Let's apply the above explanations to an example. We'll insert a log message and set its report level.

### Initial situation

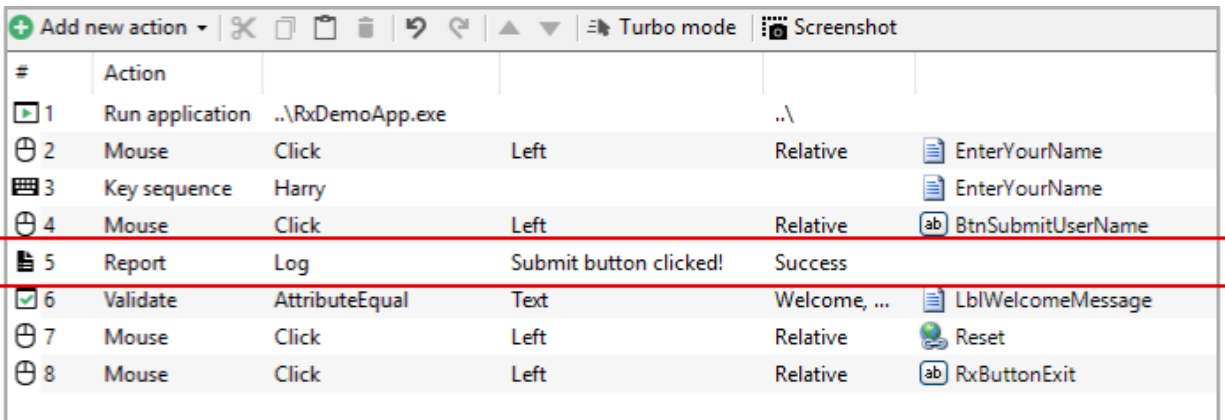
The recording module in the sample solution contains 7 recorded actions, one of them a validation action.

The recognition level of the test case is set to **Success**. This means that currently, only the successful or failed validation would be reported during a test run.

We now want to add a log message which tells us that the **Submit** button has been clicked, and of course we want it to show up in the report, whereas all other actions except for the validation shouldn't show up.

### Here's how:

- 1 **Add** a log message after action #4.



#	Action				
1	Run application	..\RxDemoApp.exe		..\	
2	Mouse	Click	Left	Relative	EnterYourName
3	Key sequence	Harry			EnterYourName
4	Mouse	Click	Left	Relative	BtnSubmitUserName
5	Report	Log	Submit button clicked!	Success	
6	Validate	AttributeEqual	Text	Welcome, ...	LblWelcomeMessage
7	Mouse	Click	Left	Relative	Reset
8	Mouse	Click	Left	Relative	RxButtonExit

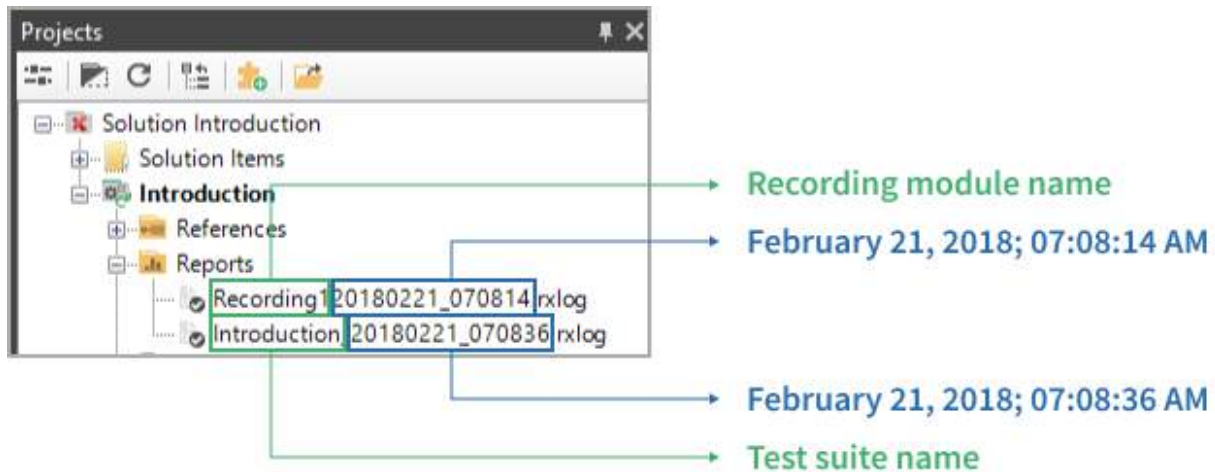
- 2 **Enter** a message and set its report level to **Success**.

### Result

The report levels for the actions in this recording and the report level threshold of the test case now look like this:







- For a run from a **test suite**, the file name starts with the test suite's name.
- For a run from a **recording**, the file name starts with the recording's name.
- The second part of the report name is a combination of the date and time the report is generated.
  - Test date = **21 February 2018** = **20180221**
  - Test time = **07:08:36 PM** = **190836**
- Date and time are separated by an underscore.
- The file ending is **.rxlog**, short for **Ranorex Log**

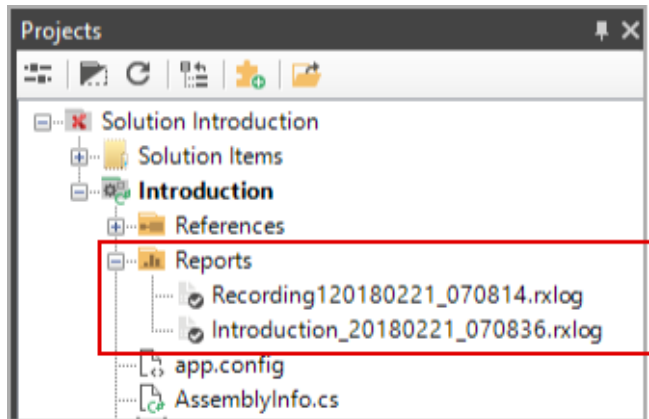


### Further reading

You can customize how report file names are generated. This is explained in Ranorex Studio system details > Settings & configuration > → [Report settings](#).

### Report file location

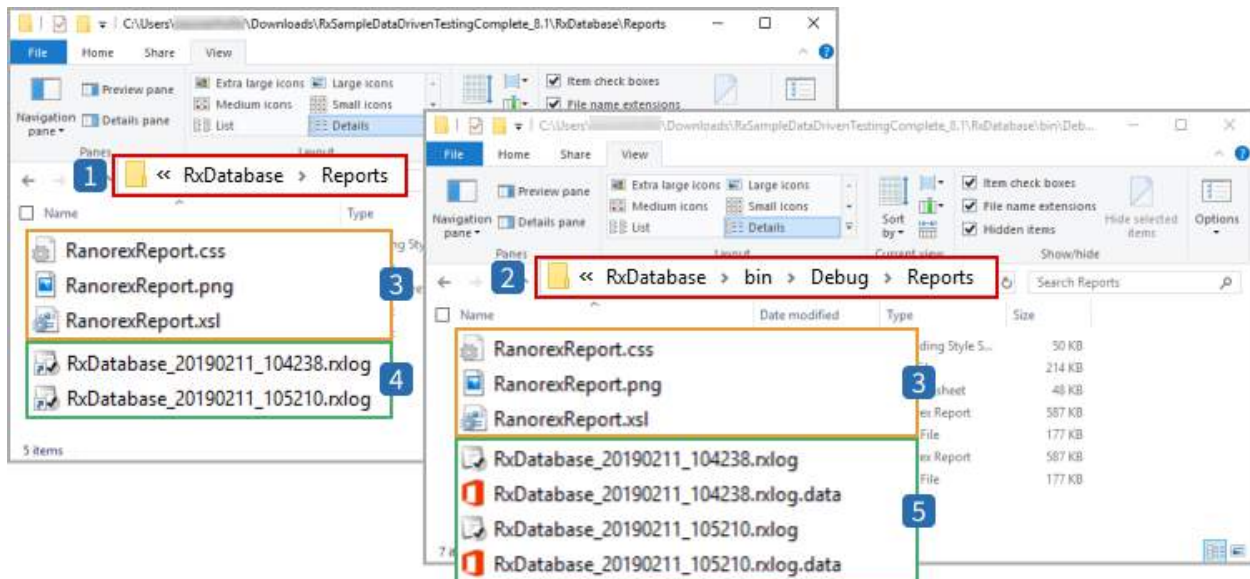
In the projects view, reports are listed in the **Reports** folder of your project.



In Windows, you can access reports from two different folders. The original reports along with the layout files and the raw data files of the report (explained later as part of customizations) are stored in the **Reports folder in the output directory**.

The **Reports folder in the project's folder** only contains a shortcut to the original reports, and does not contain any of the stylesheet or raw data files.

See here for an illustration:



- 1 **Reports** folder in the project's folder.
- 2 **Reports** folder in the output directory of the project.
- 3 **Layout** files for the report.
- 4 **Shortcuts** to the original report files.

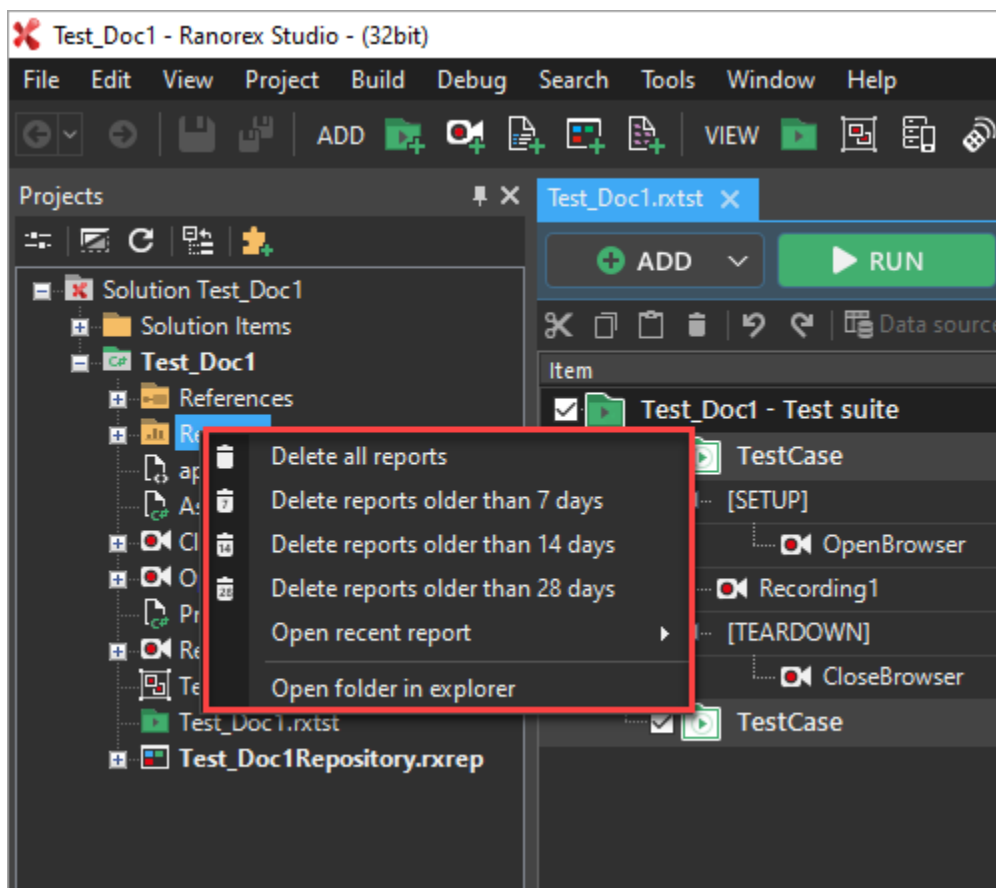
## 5 Original report files with raw data files.

### Report deletion

To ease the report file maintenance, right-click Reports in the Project View panel.

The following options are listed for report deletion which is based on the difference between the creation date and the current date:

- Delete all reports
- Delete reports older than 7 days
- Delete reports older than 14 days
- Delete reports older than 28 days



The reports are removed from the current configured directory.

Ranorex does not track configured report directories that are changed. If the user changes the configured report directory, the deletion does not affect any report files in previous directories.

The files with the following extensions that are not deleted:

- .css
- .png
- .xsl

A confirmation dialog displays to proceed with the removal.

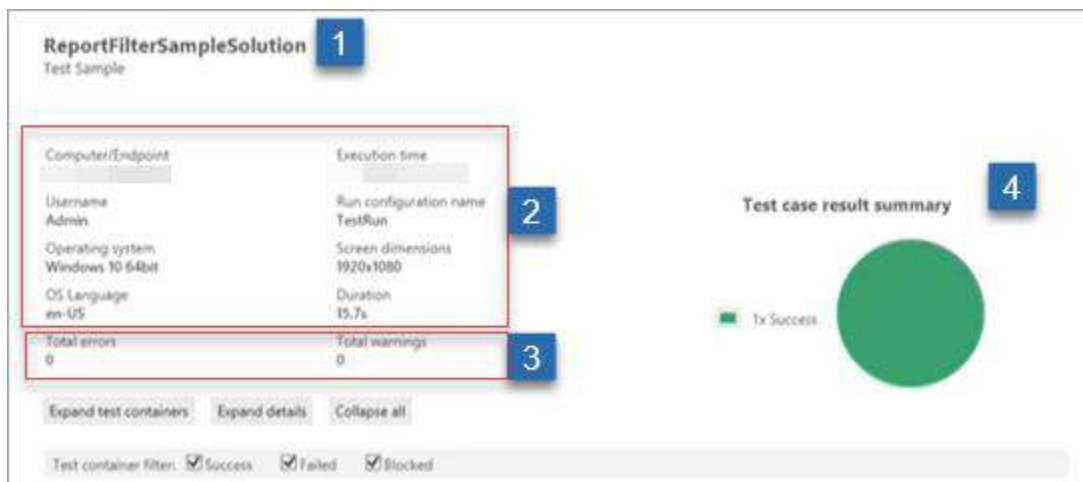
### Note

For solutions you use in Ranorex Studio 9 but created in an earlier version, the behavior is slightly different:

The original reports along with the layout and raw data files are stored directly in the output directory of your project, and not collected in a Reports folder.

## Report header

The report header displays a range of useful data and summarizes the test results.

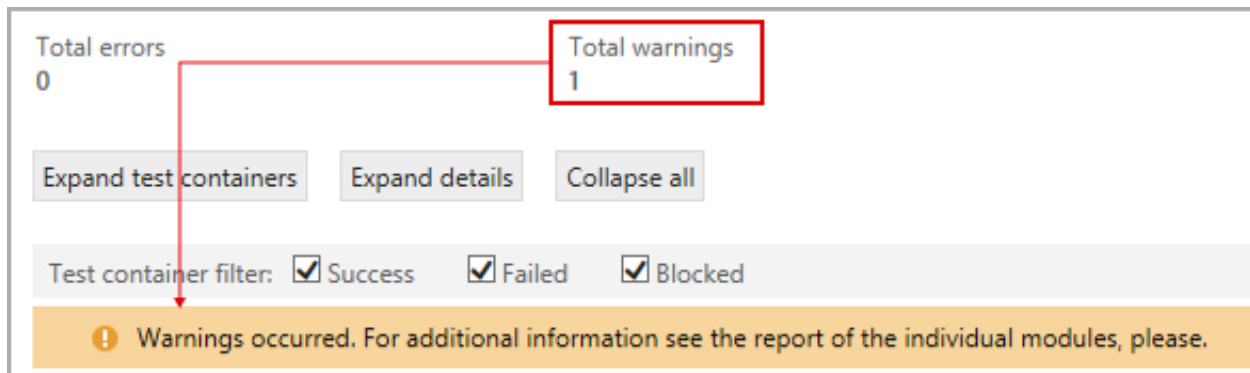


### **1** Test suite name/recording module name

The name of the test suite or recording module from which the test was run.

- 2 System and test data
- 3 Error/warning counters

If there's at least one warning in the report, an orange notification appears, as shown below.



#### 4 Test results

A pie chart summarizing the test results.

##### Note

**Only test cases** are counted for this chart. Smart folders are ignored; you can see their results in the report details further below.



#### 5 Successful test

A green pie chart means all test cases were successful.

## 6 Failed test

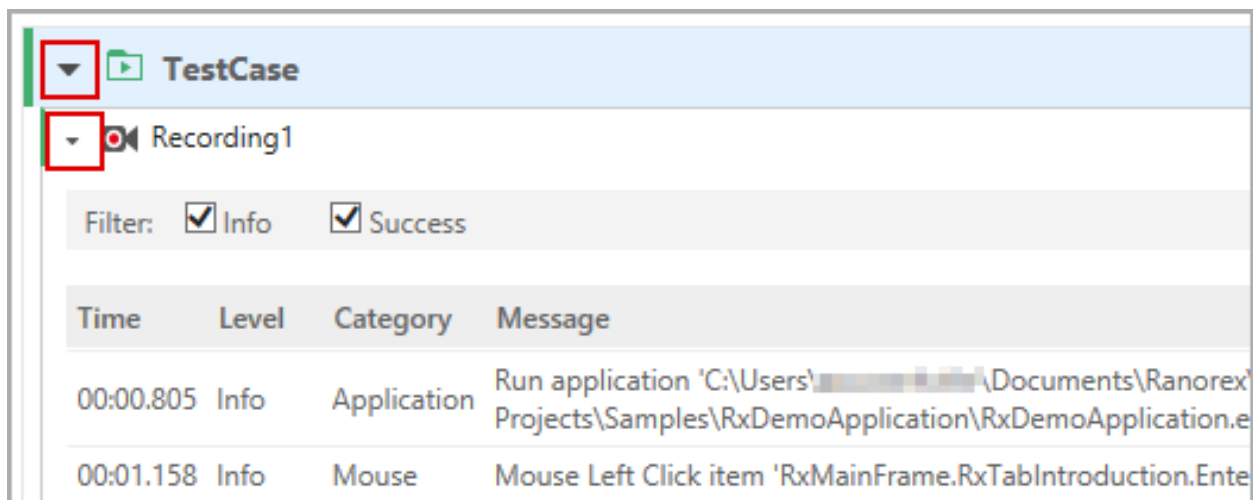
A red pie chart means all test cases failed.

## 7 Test summary

- Pie chart with successful, failed, and blocked test cases
- In the example, the run configuration contained 8 test cases. The first two test cases passed successfully. The third failed, after which the test was aborted, blocking the remaining 5 test cases.

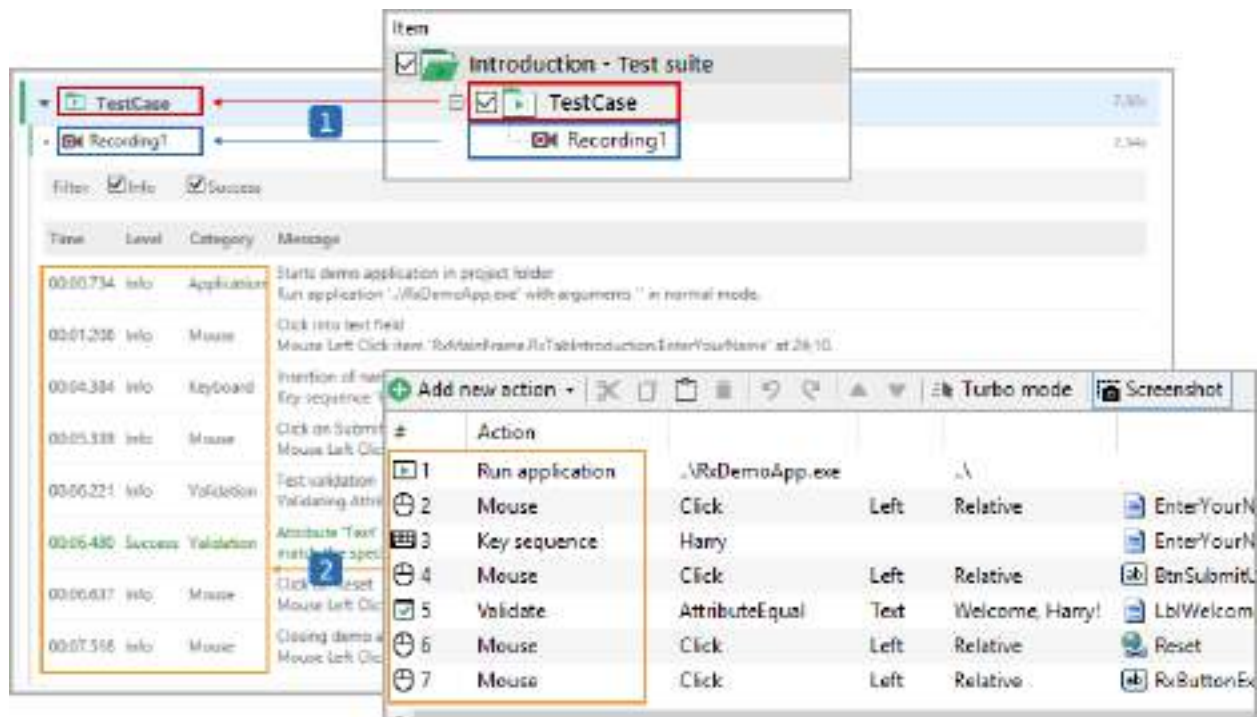
## Viewing report details

By default, the details for test cases, smart folders, etc. are collapsed. Simply click the arrow next to an item's name to show the details.



## Basic report structure

The report is structured in the same way as the test suite or recording module on which the test run was based. Each structure element in the test suite appears the report, as well as each test action item. Report messages appear for all actions in the order of execution provided that they met the report level threshold.



- 1 The structure is the same as in the test suite of this test run.
- 2 The **Recording1** item contains all the report messages for the actions in it.

## Detailed report contents

Time	Level	Category	Message
00:00:00.000	Info	Application	Run application 'C:\Users\james\Documents\Ranorex\RanorexStudio\Projects\Sample\VRxDemoApplication\VRxDemoApplication.exe' with arguments '' in normal mode.
00:01:15.000	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.EnterYourName' at 24.10.
00:04:50.000	Info	Keyboard	Key sequence 'Harry' with focus on 'RxMainFrame.RxTabIntroduction.EnterYourName'.
00:05:38.000	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.BtnSubmitUserName' at 40.10.
00:06:20.000	Info	Section	Validation action next:
00:06:27.000	Info	Validation	Validating AttributeEqual (Text='Welcome, Harry!') on item 'RxMainFrame.RxTabIntroduction.LblWelcomeMessage'.
00:06:43.000	Success	Validation	Attribute 'Text' of element for item 'IntroductionRepository.RxMainFrame.RxTabIntroduction.LblWelcomeMessage' does match the specified value.
00:06:58.000	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxTabIntroduction.Reset' at 11.6.
00:07:41.000	Info	Mouse	Mouse Left Click item 'RxMainFrame.RxButtonExit' at 48.10.



## 1 Time

By default, this column displays the relative execution time measured from the beginning of the test run. This can be changed in the → [report settings](#).

## 2 Level

The second column shows the report level of the action performed.

## 3 Category



This column shows the action type performed.

## 4 Message

The report message contains more detailed information as to what happened during the execution of the action.

Insertion of name  
Key sequence 'Harry' with focus on 'RxMainFrame.RxTabIntroduction.EnterYourName'.

In case of a failure, the report will also include two screenshots, one at the moment of failure and one right before the failure occurred.

Time	Level	Category	Message
00:00.752	Info	Mouse	Mouse Left Click item 'RxMainFrame.MonthCalendar1' at 4/4, 
00:02.220	Error	Module	Image not found in element {DateTime:monthCalendar1}. <a href="#">Show/Hide Stacktrace</a> 

1 Screenshot immediately before the failure.

2 Screenshot during failure.

## Data iterations in reports

In data-driven tests, a test case or smart folder bound to data goes through a number of iterations depending on the data source. In the report, each of these iterations is shown separately.

The screenshot displays the Ranorex Studio interface. On the left, a tree view shows a test suite 'RxDATABASE - Test suite' with a test case 'Data-driven\_DB\_Test' and its iterations 'Data-driven\_DB\_Test Data Row 1' and 'Data-driven\_DB\_Test Data Row 2'. A red box highlights the test case and its iterations, with a blue '2' indicating the number of iterations. On the right, the 'Data binding / Iterations' pane shows the test case 'Data-driven\_DB\_Test' with a red box around it and a blue '1' indicating the number of iterations. Below this, the 'Current variable values' pane shows the values for the test case: '\$txtFirstName = 'Mary', \$txtLastName = 'Smith''. On the bottom right, a table labeled 'Test data' shows the data used for the iterations:

	FirstName	LastName	Age	Gender	Department
1	John	Public	48	Male	Project Management
2	Mary	Smith	36	Female	Sales
3	Henry	Rogers	29	Male	Support
4	Thomas	Bach	42	Male	Development

### 1 Data iterations

- Data iterations are indicated by the **Rows: #** labels next to test containers.
- **Rows #** indicates how many times the data container will be iterated – one time for each row.

### 2 Data rows in test report

- Each data row/iteration and its details are shown in the report.

### 3 Variable and value

- Variables and values used in this iteration.



## Note

- Secret test data can be masked in the data source dialog. This will also apply to the report.
- If, for example, age and gender are masked, the data appears as shown below:

▼ Test data		
FirstName: Mary	LastName: Smith	Age: ●●
Gender: ●●●●●●	Department: ●●●●●	Num: ●



## Reference

Data iterations and masking data are explained in Ranorex Studio advanced > [Data-driven testing](#).

## Run iterations in test reports

If a test case or a smart folder is run iteratively, each run iteration appears in the report with its details.

Test container filter: <input checked="" type="checkbox"/> Success <input checked="" type="checkbox"/> Failed <input checked="" type="checkbox"/> Blocked		
▼  TestCase	Iterations: 4	16.53s
▶  TestCase	Run: 1	4.66s
▶  TestCase	Run: 2	3.99s
▶  TestCase	Run: 3	3.89s
▶  TestCase	Run: 4	3.98s

- Run iterations are indicated by the **Iterations: #** and **Run: #** labels next to test containers.

- **Iterations:** # tells you how many times the test container is run.



## Reference

Run iterations are explained in Ranorex Studio fundamentals > Test suite > [Running tests](#).

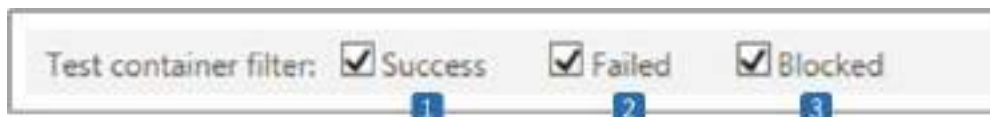
## Filter messages

There are three filters available in the report:

- One for filtering test containers
- One for filtering messages for actions
- One for filtering the action category

### Filter test containers

Use the checkboxes in the upper part of the report to filter test containers by their success status.



- 1 Success:** displays report messages with report level 'Success' if checked, and if these messages are included in the report based on report levels. Hides these messages if not checked.
- 2 Failed:** displays report messages with report level 'Failure'. Hides these messages if not checked.
- 3 Blocked:** displays blocked test cases of a test run when checked. Hides blocked test cases if not checked.

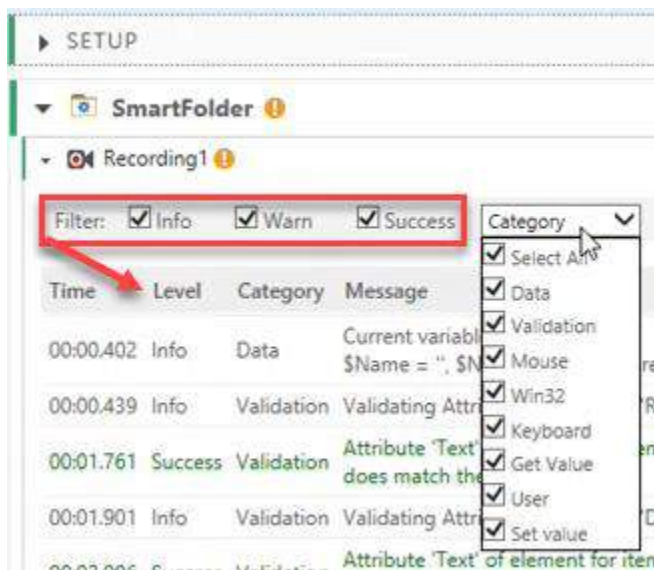


## Note

Blocked test cases or smart folders could not be executed during a test run, either because the test was aborted beforehand, a condition wasn't met, or they were deselected in the run configuration.

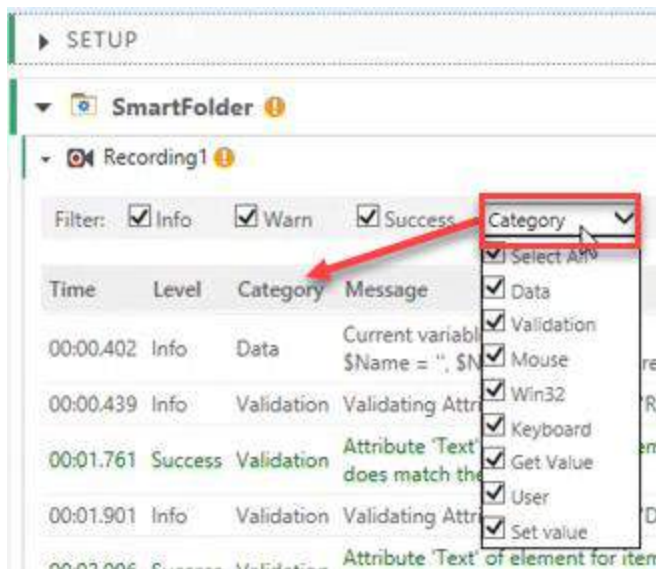
## Filter action messages

You can also filter action messages in modules by their report level.



## Filter category messages

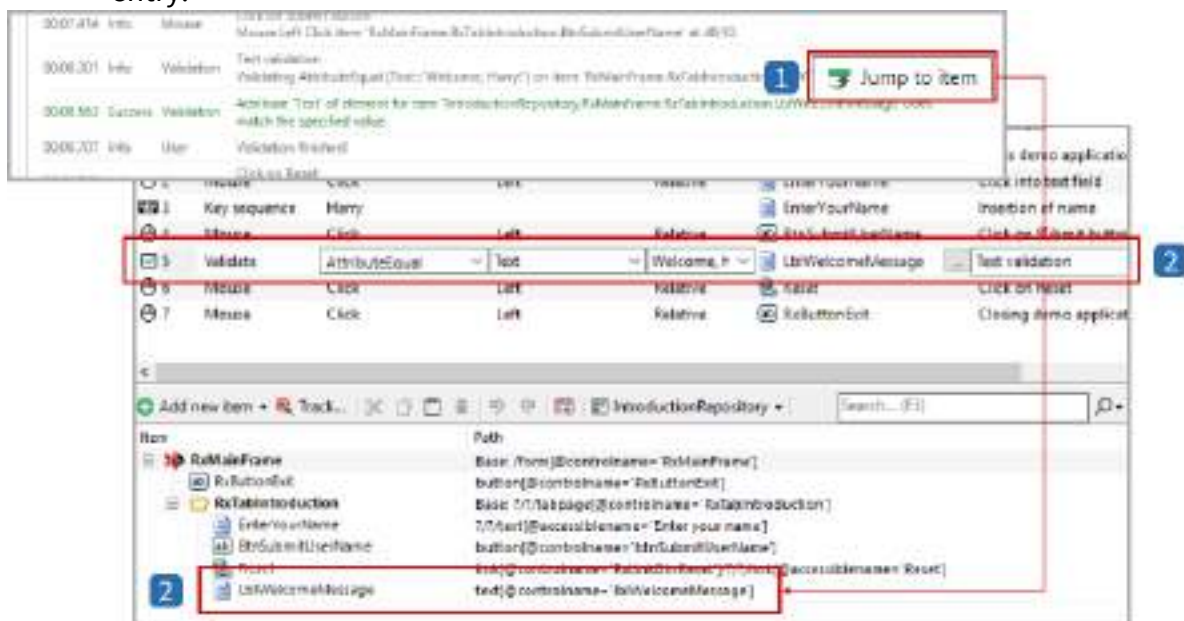
You can also filter the action messages in the modules by their action type using the **Category** filter.



## Jump to item

You can directly jump from the report message to the corresponding action or test suite item in Ranorex Studio.

- 1 **Mouse over** a report entry, i.e. a test container or a report message for an action.
- 2 Click the **Jump to item** button that appears in the right upper corner of the report entry.



- 1 **Jump to item** button.

- 2 Corresponding action with linked repository item.

## Open in Spy

When you open a report outside of Ranorex Studio, the **Open in Spy** button becomes available. Click the button to open the corresponding report item in Ranorex Spy.



- 1 **Open in Spy** report functionality.

## Video reporting

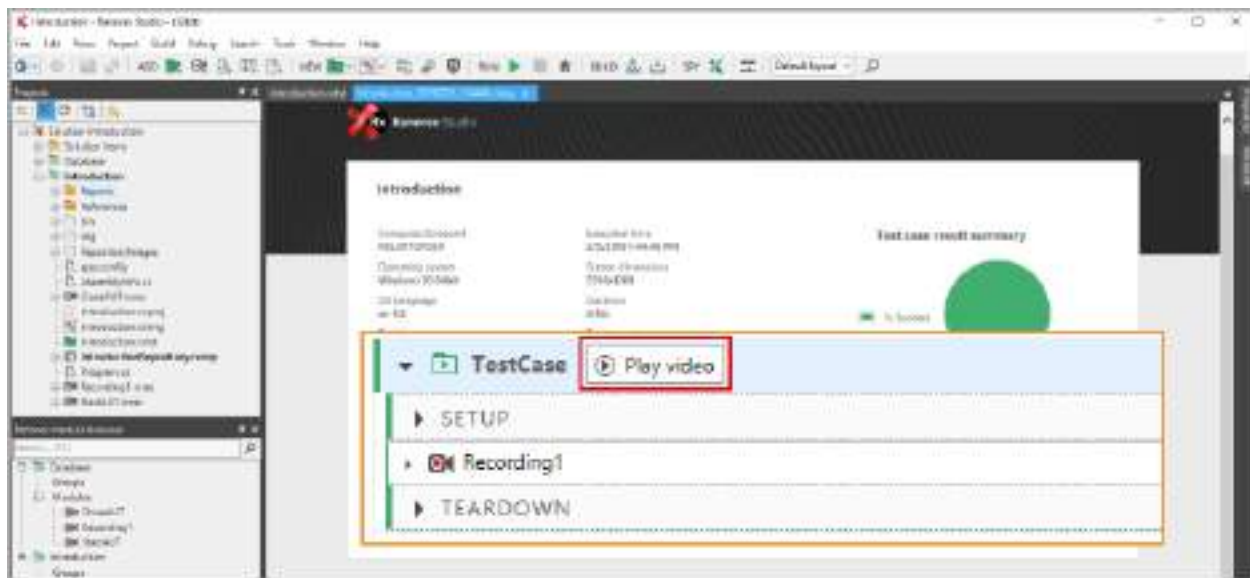
Video reporting lets you record a test run as a collection of videos. This also works on Ranorex Agents.

### Enable video reporting

Video reporting is disabled by default. You can enable and configure it in the [Report settings](#).

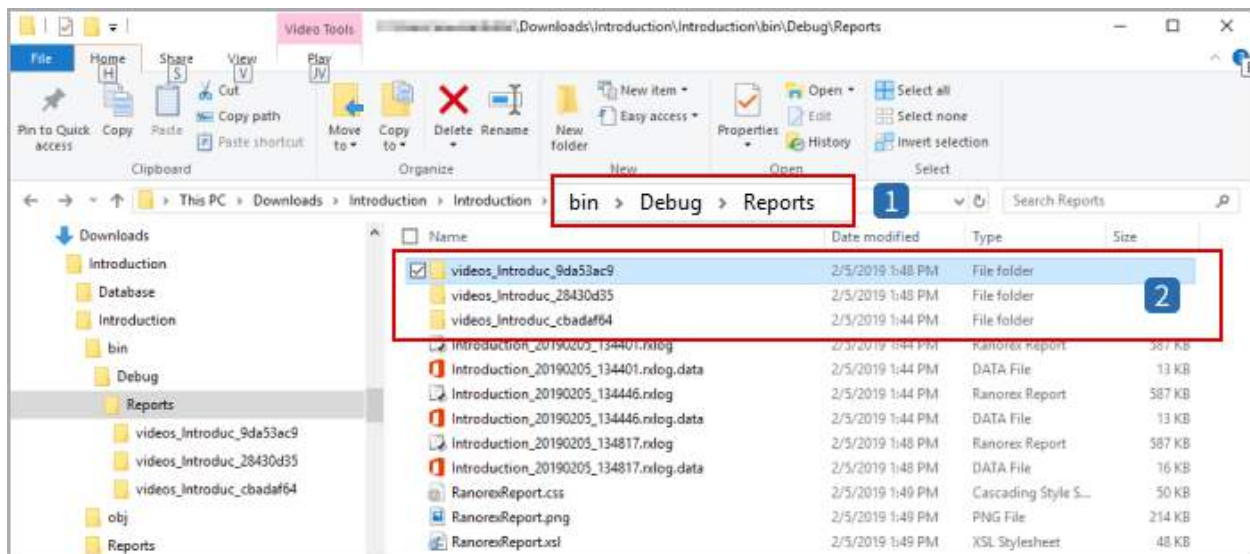
### Play video reports from the report

In the report, click Play video next to a test case to play the respective video.



## Video directory

The videos are saved in the output directory in the Reports folder. For each test run, a separate folder is created.



1 Reports folder in the output directory

2 Three video folders

## Progressive report preview

Ranorex generates the report as the test run progresses. You can view it at any time during the test run. This is especially useful for very long test runs.



- Ranorex starts generating the report as soon as the test run begins.
- The report file is saved once the defined auto-save time elapses (default = 30s).
- At this point, you can open the in-progress report simply by double-clicking the report file.



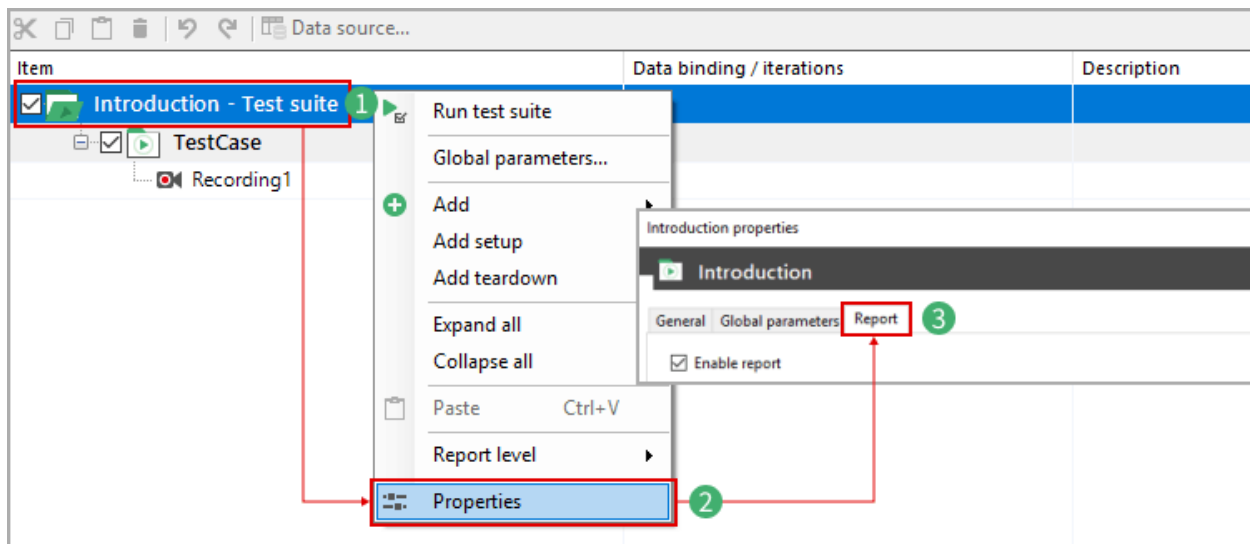
### Note

Naturally, if you try to open the report on the machine the test is running on, you will very likely cause a test failure because you will be interacting with the UI.

## Report settings and configuration

To access the report settings:

- 1 In the test suite view, **right-click** the test suite.
- 2 **Click Properties.**
- 3 **Click the Report tab.**



## Further reading

You can customize how report file names are generated. This is explained in [Ranorex Studio system details > Settings & configuration > Report settings](#).

## Customization basics

The standard report is usually appropriate for most scenarios. However, to suit individual needs, it's fully customizable. Because customization is a large topic, we've split it up into two chapters.

In this chapter, you'll learn about the basics of customization and go through a few example to customize the look of the report without coding.



## Reference

For more complex customizations that involve coding, refer to

[Ranorex Studio fundamentals > Reporting > Complex customizations](#).



### Screencast

The screencast “Introduction to customizing reports” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Download the sample solution

The explanations in this chapter are based on a sample solution that you can download here: [Sample Custom Report](#)

### Install the sample solution:

1

**Unzip** to any folder on your computer.

2

**Start** Ranorex Studio and **open** the solution file `Introduction.rxsln`



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

## From raw data to the report

The image below illustrates the process that prepares raw data to be displayed in an easily readable format in the report.



1. During the test run, the report engine collects data in an XML format.
2. The report engine converts this data to HTML and creates a report file based on CSS and XSL specifications.
3. Ranorex Studio displays this HTML-based report with its built-in HTML viewer.

## Collected test data

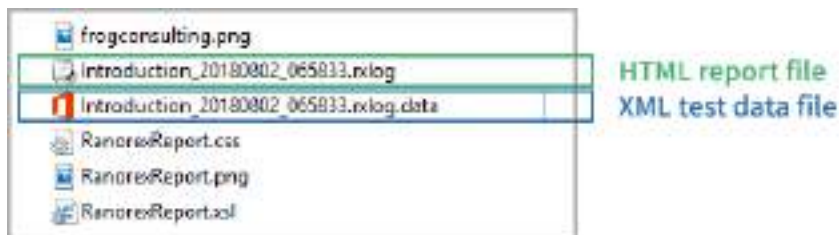
Data from a test run is collected and stored in an XML format. This raw data can also be read and used outside of Ranorex Studio with any XML viewer/editor.

```

<item>
  timeRelativeToTestSuiteStartTime="00:07.018"
  timeRelativeToTestModuleStartTime="00:06.318"
  timeWallClock="06:59:40"
  time="00:07.018"
  level="Info"
  category="Keyboard">
    <message>
      Insertion of name<br/>Key sequence 'Harry' with focus on
      'RxMainFrame.RxTabIntroduction.EnterYourName'.
    </message>
    <metaInfo>
      type="reportItem"
      path="/form[@controlname='RxMainFrame']/?/?/tabpanel[
        @controlname='RxTabIntroduction']/?/?/text[@accessiblename='Enter your name']"
      itempath="/?/?/text[@accessiblename='Enter your name']"
      fullname="IntroductionRepository.RxMainFrame.RxTabIntroduction.EnterYourName"
      id="8c857999555b473c9c63a6eda5aac6cf"
      timeout="90000"
      codefile="C:\\Users\\[username]\\Downloads\\RxSampleCustomReport\\Introduction\\Recording1.cs"
      codeLine="93"
      itemIndex="2"
      loglvl="Info">
    </metaInfo>
  </item>

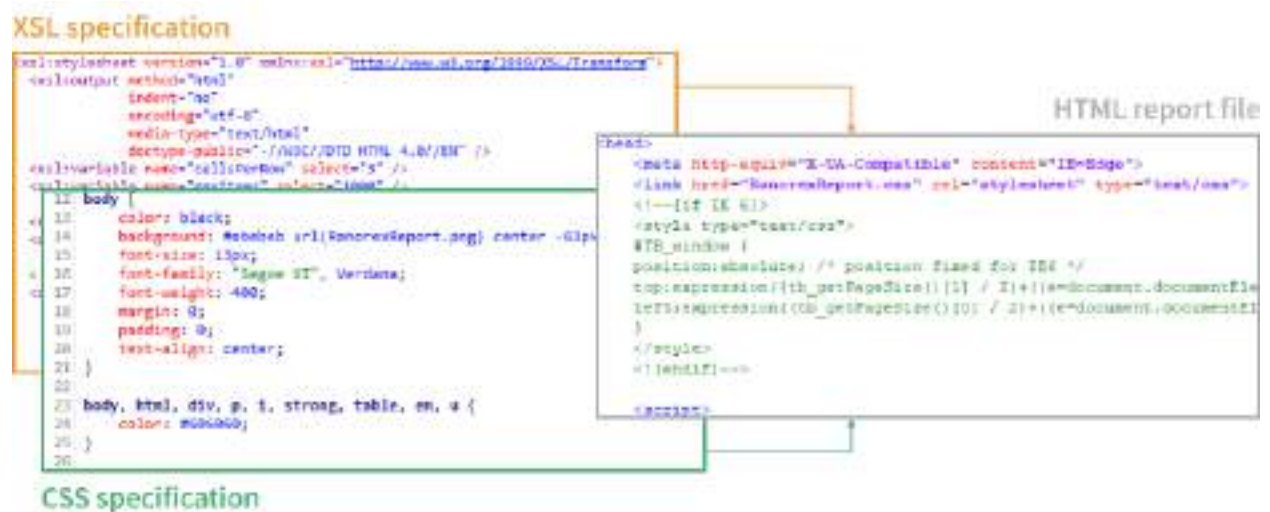
```

The raw XML data is always saved in the same folder as the corresponding original report file and has the same file name with the added suffix `.data`.

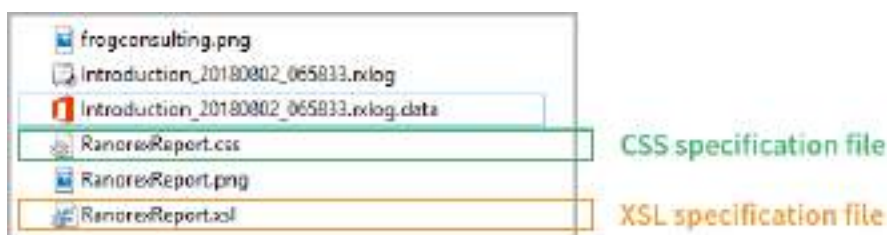


## CSS and XSL specifications

The raw XML data is used to create a HTML file based on CSS and XSL specifications.



The XSL and CSS specifications used to generate the HTML file are also stored in the same folder as the XML data and the actual report.



## Further reading

To be able to change the layout and content of Ranorex standard reports, a basic understanding of HTML, CSS, XSL, and XML is recommended. Refer to the World Wide Web Consortium (W3C) at [www.w3.org](http://www.w3.org) for further reading.

## HTML report file

The final report generated from XML by XSL and CSS is stored in a HTML-based format in the **Reports** folder of the project's output directory.



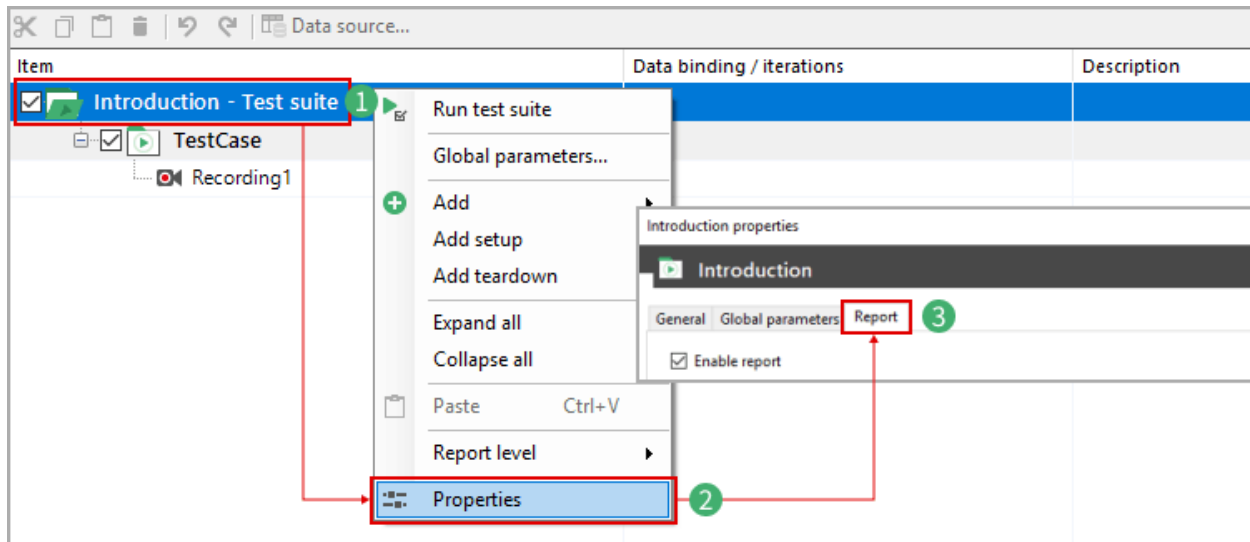
### Note

The report can be viewed in any web browser. The report file can be edited in any HTML editor.

## Create a custom report template

Now that we've established how reports are generated, we can start customizing the report. There's a convenient way to do this in Ranorex Studio without affecting the standard report:

- 1 In the test suite view, **right-click** the test suite.
- 2 **Click Properties.**
- 3 **Click the Report tab.**



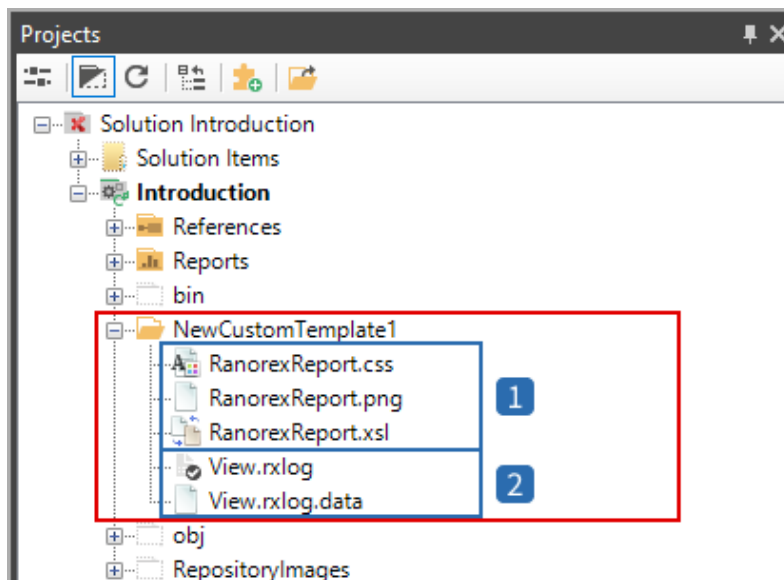
4 Click **Create custom template**.

5 A message appears, telling you where the new report template folder has been created. **Confirm** with **OK**.



## Result:

In the projects view, you can see the new template folder with the copied report files.



### 1 *CSS, XSL, and image files*

- These are the copied files that you can edit to customize the layout and content of the report.
- The PNG file contains the Ranorex logo as the default logo for the report as well as other images used in the standard report. You can replace these with your own images.

### 2 **Preview files**

- These files enable you to quickly check what your customizations will look like in an actual report. Just open View.rxlog to get a preview.
- You can also open it from Windows, so you don't have to start Ranorex Studio.

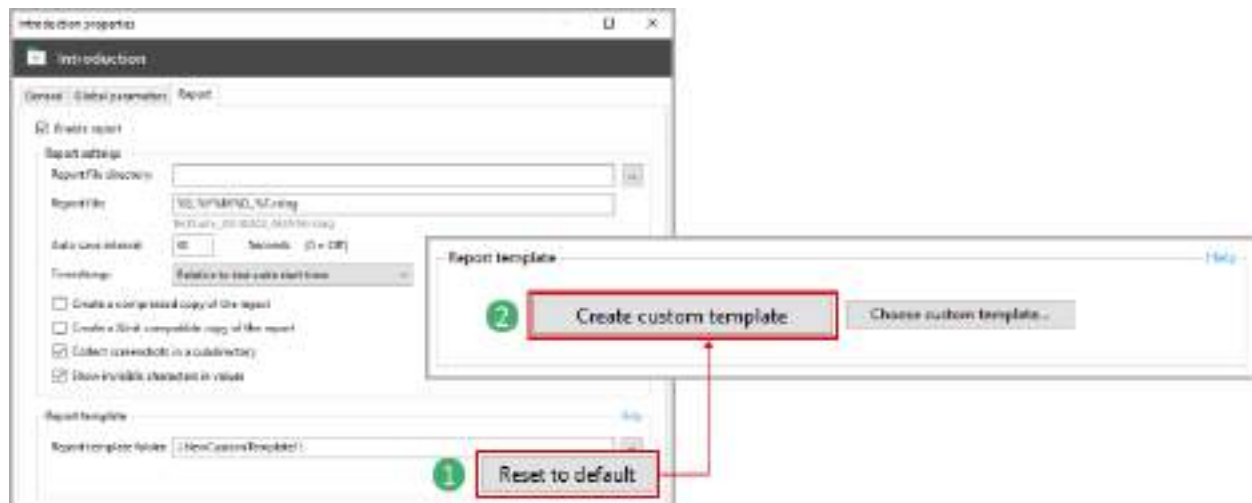
## Multiple custom report templates

You can create as many custom report templates as you want. However, only one template can be active at a time for a test suite.

To create additional custom templates:

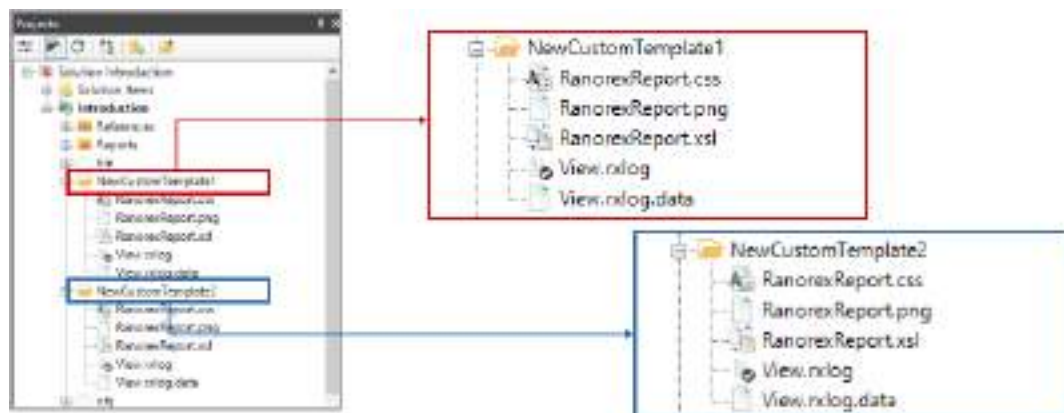
- 1 Click **Reset to default** in the **Report** tab of the test suite properties.
- 2 Click **Create custom template** again and **confirm** with **OK**.





## Result:

Another template folder appears in the projects view.



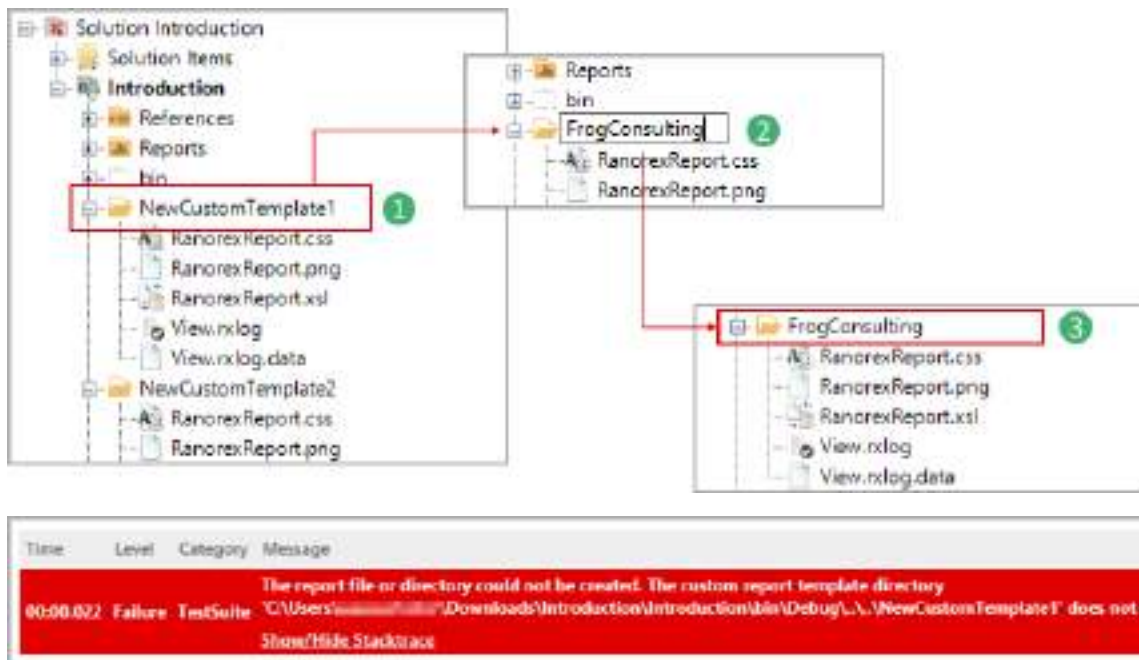
## Rename custom report templates

You can rename custom report templates.

- 1 In the projects view, **select** the template you want to rename.
- 2 Press **F2** and **rename** the template folder.
- 3 The renamed folder.

## ⚠ Attention

The report engine doesn't automatically recognize the name change. You will have to reapply the template after renaming it. This is explained in the next section.



## Choose/reapply a custom report template

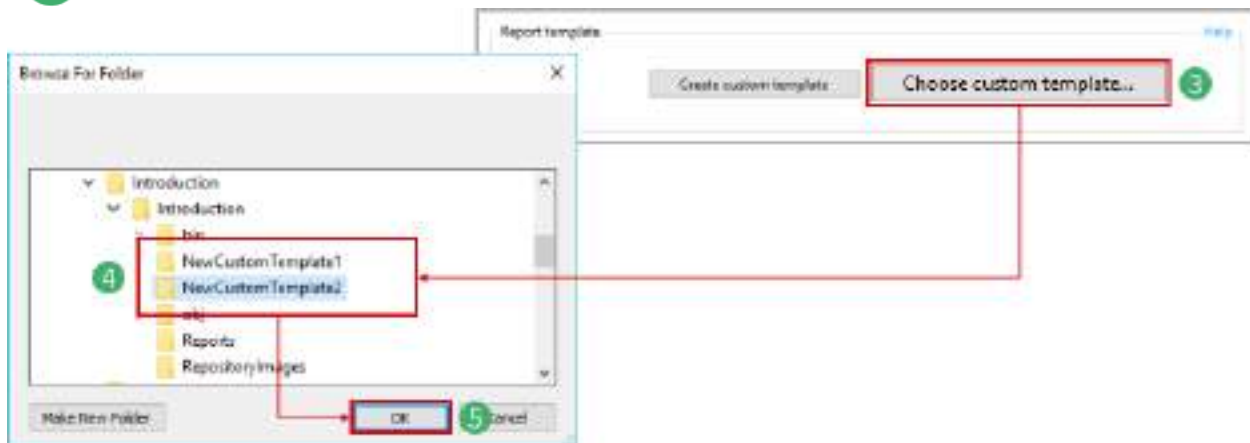
This function is used for three different purposes:

- To add an existing custom template from a folder.
- If you have multiple custom templates, to choose which one should be used.
- If you've renamed a template, to reapply it so Ranorex Studio can find it again.

## i Note

Only one custom report template can be active at a time for a test suite.

- 1 **Open** the **Report** tab in the test suite properties.
- 2 If there's a custom template currently active, **click Reset** to default template.

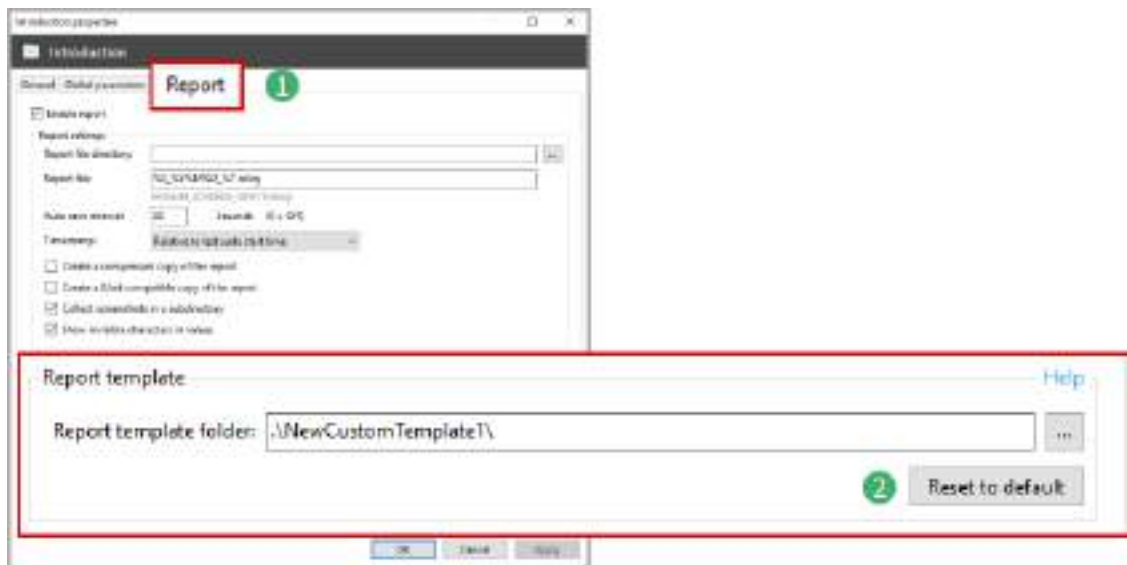


- 3 **Click Choose custom template....**
- 4 **Choose** a custom template in the browser window.
- 5 **Click OK.**

## Reset to default report template

If you want to stop using a custom template, you can reset to the default Ranorex Studio template at any time.

- 1 **Open** the **Reports** tab in the test suite properties.
- 2 **Click Reset to default** and **confirm** with **OK**.



### Note

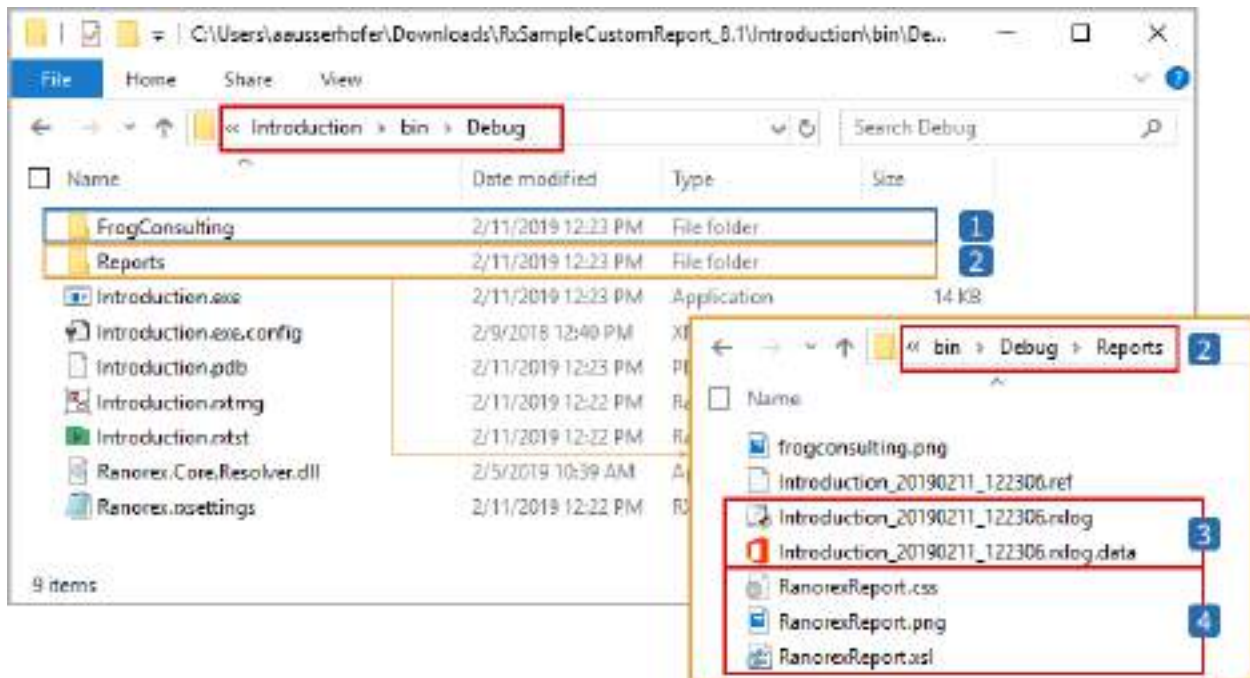
Custom report templates won't be deleted by this action. You can still reapply them as described in the section above.

## How Ranorex Studio processes custom templates

Ranorex Studio has a special mechanism for processing customized report templates, which includes the output folder and the reports folder for each project. **Understanding this process is important when you want to include external files in a report template**, e.g. a PNG containing your logo, because these files aren't included in the process by default.

### Output folder \bin\Debug\

When you execute a test, Ranorex Studio copies all files needed for the test run into a designated output folder called `\bin\Debug\`, which is located in your project's folder. The report files are also among the files copied. They are taken from the `\Reports\` folder. External files, such as a logo, aren't copied to the output folder unless they are configured to be and therefore won't appear in your report by default. How to do this is explained in the next section.



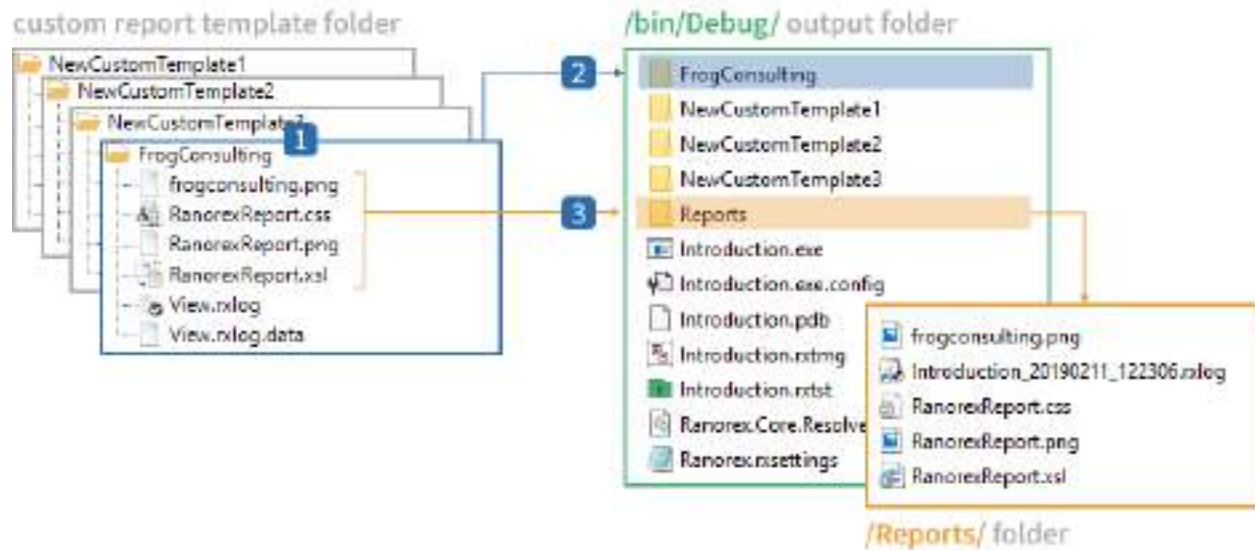
- 1 Copy of the customized report template folder (FrogConsulting in this example) in the output folder
- 2 Reports folder in the output folder
- 3 Report file (.rxlog) and corresponding raw data file (.rxlog.data)
- 4 CSS and XSL specification files and default logo file RanorexReport.png

### Note

Ranorex Studio synchronizes all the files of the output folder for every subsequent test run.

## Process summary

The image and descriptions below illustrates the process. As mentioned earlier, external files need special configuration to be included in the process. In this case, we assume that this has been done. You can learn how to do so further below.



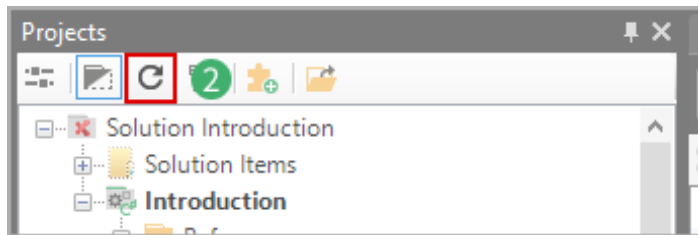
- 1 A custom report template has been applied to the test suite. The folder for the custom template is located in your project's folder, FrogConsulting in this example.
- 2 On the first test run, the custom template folder is copied directly to the output folder of the project.
- 3 The report layout files (CSS, XSL, and custom files) are also copied from the template folder directly to the Reports folder in the output folder of the project.

On subsequent runs, the custom template folders in the project folder and the output folder are synchronized, i.e. the output folder always contains the newest files from the custom template folder.

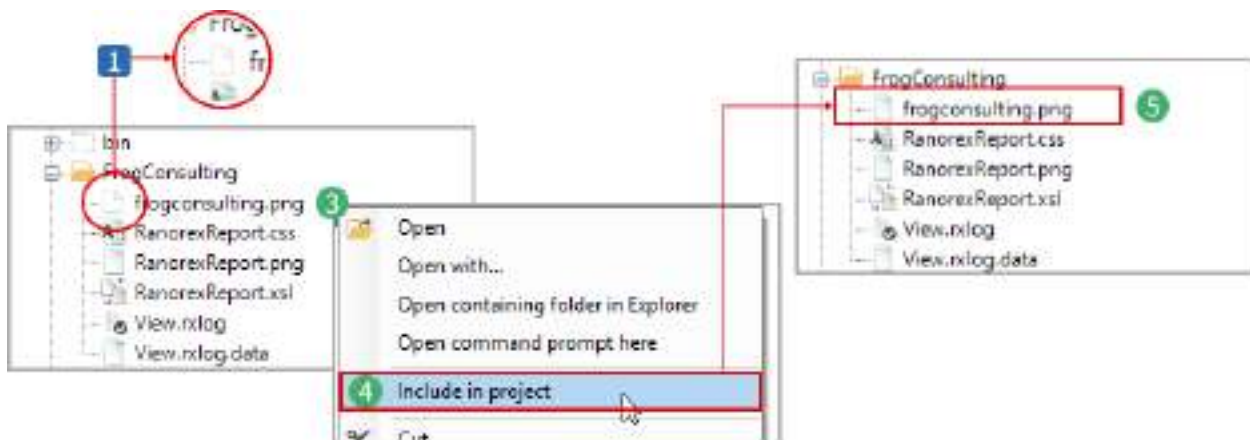
## Including external files in the report

By default, Ranorex Studio only includes internal files (.rxlog, rxlog.data, .css, .xml) in the reporting process, i.e. only these are copied to the output folder. For external files such as logos to show up in your custom report, you will need to include them in this process manually.

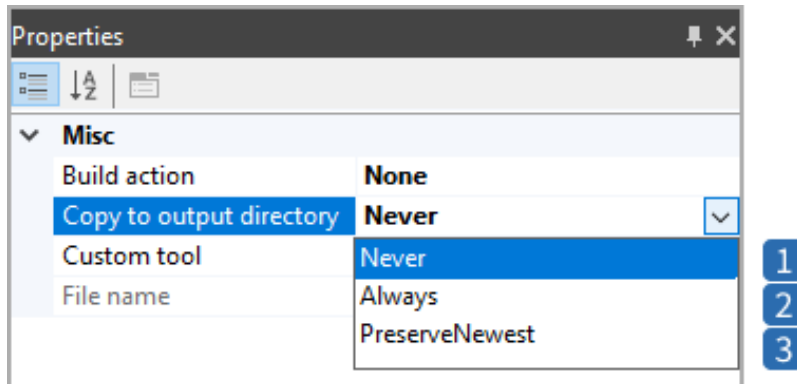
- 1 **Copy** the external file(s) to the folder of the custom report template you want them to be part of.
- 2 In the projects view in Ranorex Studio, **click** the **refresh button** to see the file in the template's folder.



- 3 **Right-click** the external file.
- 4 **Click Include in project.**
- 5 The file is now included in the project, but we're not finished yet.



- 1 Any file(s) copied into the custom report folder are initially **not** part of the project, meaning they aren't included in the reporting process either. They appear grayed out in the projects view.
- 6 **Select** the external file and **press** **F4** to open its properties.
- 7 You can see **Copy to output folder** is set to **Never**, so the file would be excluded from the reporting process. Change it to **Always** or **Preserve newest**.



- 1 **Never** is the default setting for any newly included external files.
- 2 **Always** means the file will be copied to the output folder for each test run.
- 3 **Preserve newest** means the file will only be copied if the version in the `\Reports\` folder is newer than the already existing one in the output folder. If no file exists yet in the output folder, it will always be copied.

## Background color and logo customization

In this example, you'll change the background color of the report and replace the Ranorex logo with a custom one. The example is based on the sample solution that's available for download at the beginning of this chapter.

### The logo

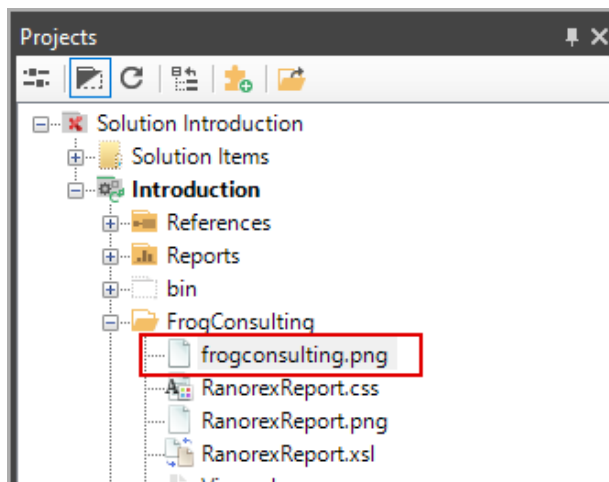
- Your logo doesn't have to be a specific size. Experiment with different dimensions.
- It's good to know the exact HEX values of the color(s) used in your logo.

We've prepared a sample logo. The green background has the HEX value `#ACDB6B`.



The **sample logo** is already in the FrogConsulting custom report template folder.

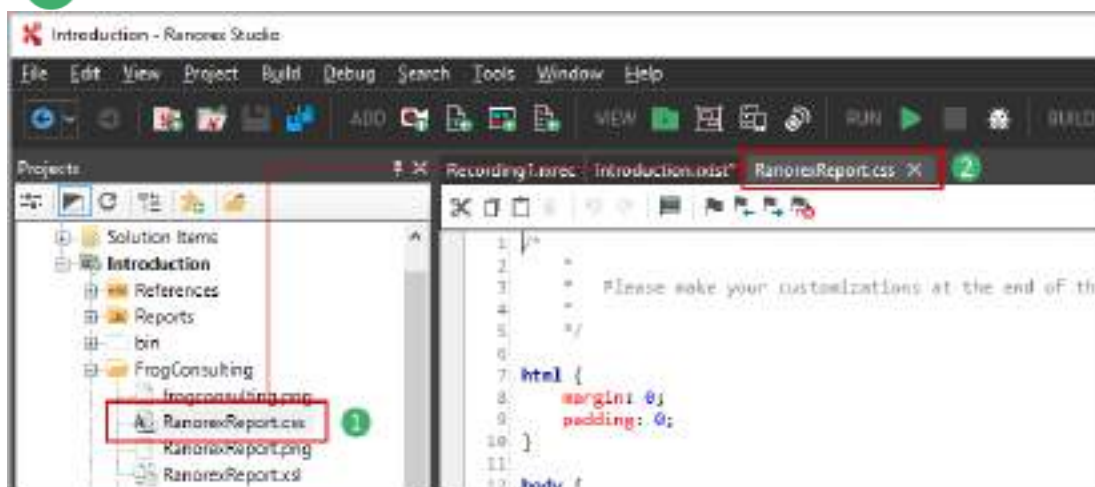




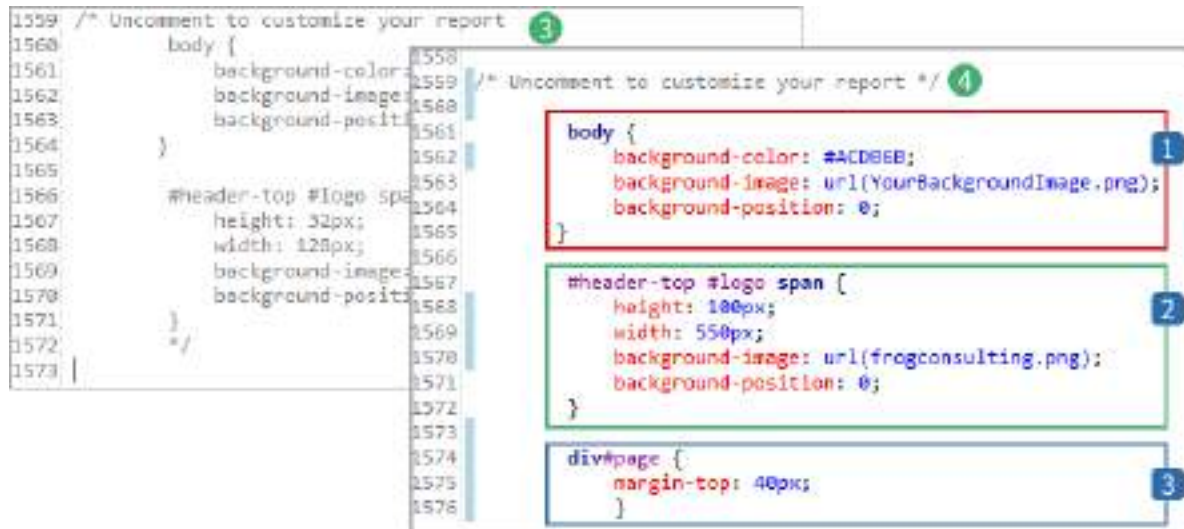
## Replace the logo

To replace the logo, you'll need to make changes to the CSS specification file.

- 1 In the projects view, **double-click** the CSS specification file.
- 2 It opens in a new tab.



- 3 Go to the end of the file and **find** the customization area.
- 4 **Delete** this line to uncomment the customization section and then **replace** its contents as described below.



### 1 Background customization

- The background color is set to the same green HEX value as the logo background color.
- All other settings are unchanged.

### 2 Custom logo

- Height and width of the logo are set to their respective values (see logo size).
- The default logo name is replaced with “frogconsulting.png”.
- The other settings are unchanged.

### 3 Report info box alignment

- Finally, a top margin of 40px is set for the general information box.

**Result:**

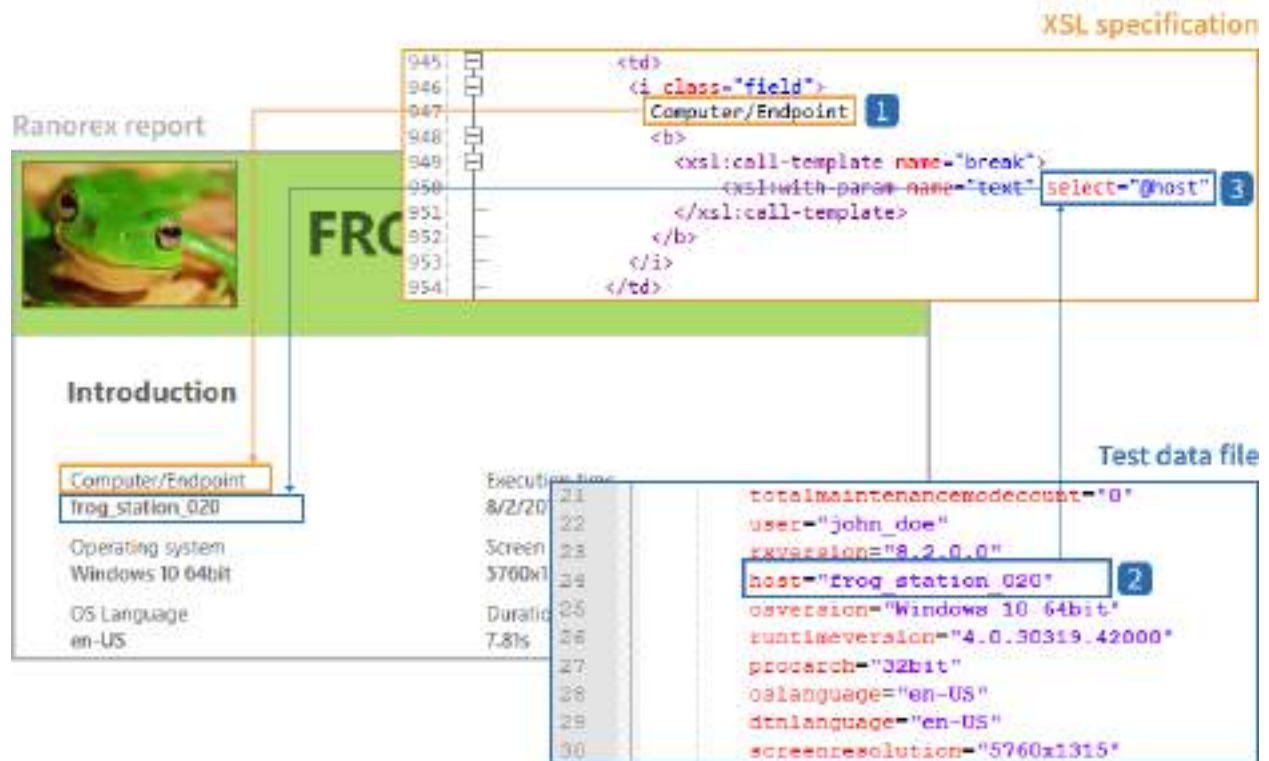


### Hint

If the logo is missing in your report, remember to include the logo file in the project and the reporting process as explained further above.

## Replace Computer/Endpoint with username

There are also various options for changing the contents of the report. We'll go through three examples that will introduce the basic principle and help you customize reports according to your own ideas. In the first example, you'll customize the report so that the entry for "Computer/Endpoint" reads "Username" and instead of the machine name displays the username.



- 1 The line in the XSL file that defines the title of the entry.
- 2 The “host” line in the raw data file. It contains the actual computer/endpoint name.
- 3 This line in the XSL file gets the computer/endpoint name from the “host” line in the raw data file and displays it below the “Computer/Endpoint” entry in the report.

To replace the “Computer/Endpoint” entry with “Username” and display the username:

- 1 In the XSL specification, **replace** “Computer/Endpoint” with “Frog user”.
- 2 Also, in the XSL specification, **replace** the variable `@host` with `@user`.



3 Save and close the file.

**Result:**



## Change report message formatting

In this example, you'll change the formatting of a specific type of report message.

As you can see in the image below, messages in the default report with the **Success** level are printed in green font. Let's change this to a **bold blue** font.

00:05:466	Info	Mouse	Mouse Left Click item 'RtMainFrame.RtTabIntroduction.BtnSubmitUserName' at 40:10
00:06:337	Info	Validation	Test validation Validating AttributeEqual (Text='Welcome, Harry!') on item 'RtMainFrame.RtTabIntroduction.LibWelcomeMessage'.
00:06:561	Success	Validation	1 Attribute 'Text' of element for item 'IntroductionRepository.RtMainFrame.RtTabIntroduction.LibWelcomeMessage' does match the specified value.
00:06:737	Info	Mouse	Click on Reset. Mouse Left Click item 'RtMainFrame.RtTabIntroduction.BtnReset' at 11:6

1 Default **Success** message in **green**

- 1 **Open** the CSS specification file.
- 2 **Find** the color definition for **Success** messages and copy it.
- 3 **Paste** it in the customization section at the end of the CSS file and **modify** it as shown in the image.
- 4 **Save** and **close** the file.

#### CSS specification

1596				
1597	tr.Success {			2
1598	color: green;			
1599	}			
...				
1574	div#page {			
1575	margin-top: 40px;			
1576	}			
1577				
1578	tr.Success {			3
1579	color: blue;			
1580	font-weight: bold;			
1581	}			

Initial success action specification  
(lines #1597-#1598 in CSS specification)

Overriding success action specification  
(end of CSS specification)

2 Default font color definition for **Success** messages

3 New definition in the customization section. Overrides the default definition.



Don't change the **default definition** (CSS lines #197 to #199). That way, you'll be able to revert to the default easily.

### Result:

[illegible]

## Remove information from report

In this example, you'll remove information from the report. This is useful when something's not relevant and you want to free up space in the report.

In our example, we want to remove the **Time** column for report messages.







comment start

```
<TABLE border="0" cellSpacing="0" class="reporttable">
  <thead>
    <!-- <th>
      <b>Time</b>
    -->
    <th>
      <b>Level</b>
    </th>
  </thead>
```

commented code

comment end

2 This code gets the time values for each action line and displays them in the correct place.

4 **Deactivate** it by commenting it out as below.

comment start

```
<tr class="{ $tablerowclass}" style="{ @style}"
  <!-- <td class="timeCell">
    <xsl:value-of select=" ./@time" />
  -->
  <td class="levelCell">
```

commented code

comment end

**Result:**

**FROG consulting, Inc.**

**Introduction**

Proj user: jake.john  
 Operating system: Windows 10-64bit  
 OS version: 1809  
 File system: NTFS

Execution time: 8/27/2018 1:02:00 PM  
 Error information: 00000000  
 Duration: 1.80s  
 Total warnings: 0

**Test case result summary**

100% Success

Expand test cases, Expand details, Collapse all

Test case: Test Case 1

Level	Category	Message
Info	Application	Starts demo application in project folder Run application '...\RxDemoApp.exe' with arguments '?' in normal mode.
Info	Mouse	Click into text field Mouse Left Click item 'RxMainFrame.RxTabIntroduction.EnterYourName' at 24/10.
Info	Keyboard	Insertion of name Key sequence 'Harry' with focus on 'RxMainFrame.RxTabIntroduction.EnterYourName'.

Log file: C:\Users\jake.john\AppData\Local\Frog Consulting\RxDemoApp\log.txt

## Complex report customization

In this chapter, you'll find instructions on how to perform a range of more complex report customizations. Since they require coding skills, you should be familiar with code modules in Ranorex Studio.



### Further reading

If you are not familiar with the concept of code modules and their application, refer to Ranorex Studio expert > [Code modules](#).

## Download the sample solution

The examples in this chapter are based on a sample solution that you can download from the following link: [Sample User Defined Report](#)

### Install the sample solution:

- 1 **Unzip** to any folder on your computer.
- 2 **Start** Ranorex Studio and **open** the solution file `RxDatabase.rxsln`



### Hint

The sample solution is available for Ranorex versions 8.0 or higher. You must agree to the automatic solution upgrade for versions 8.2 and higher.

## The standard report classes

The easiest way to create a report message in code is using one of the six different standard report classes.

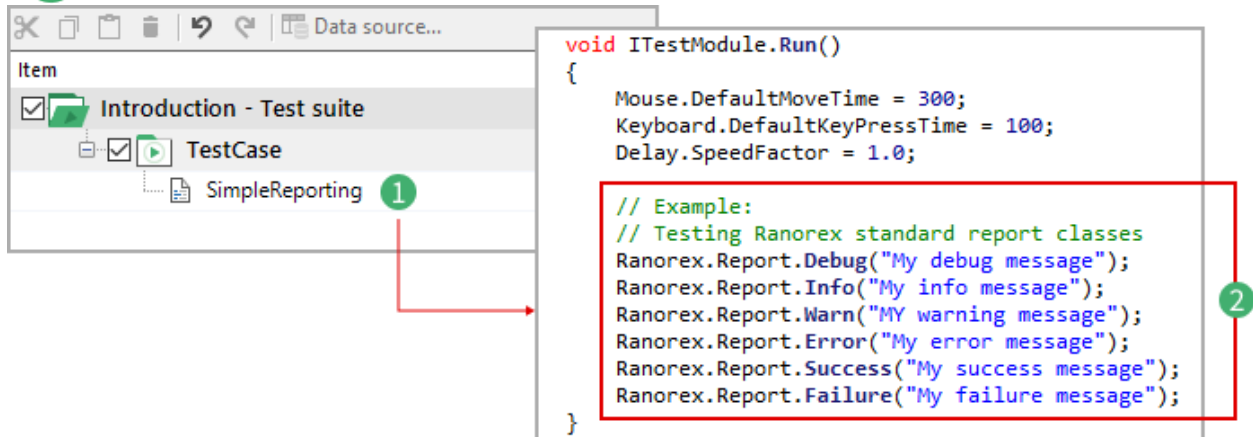
### Standard report classes

Ranorex Studio has six standard report classes, as shown below. These classes correspond to the standard report levels.

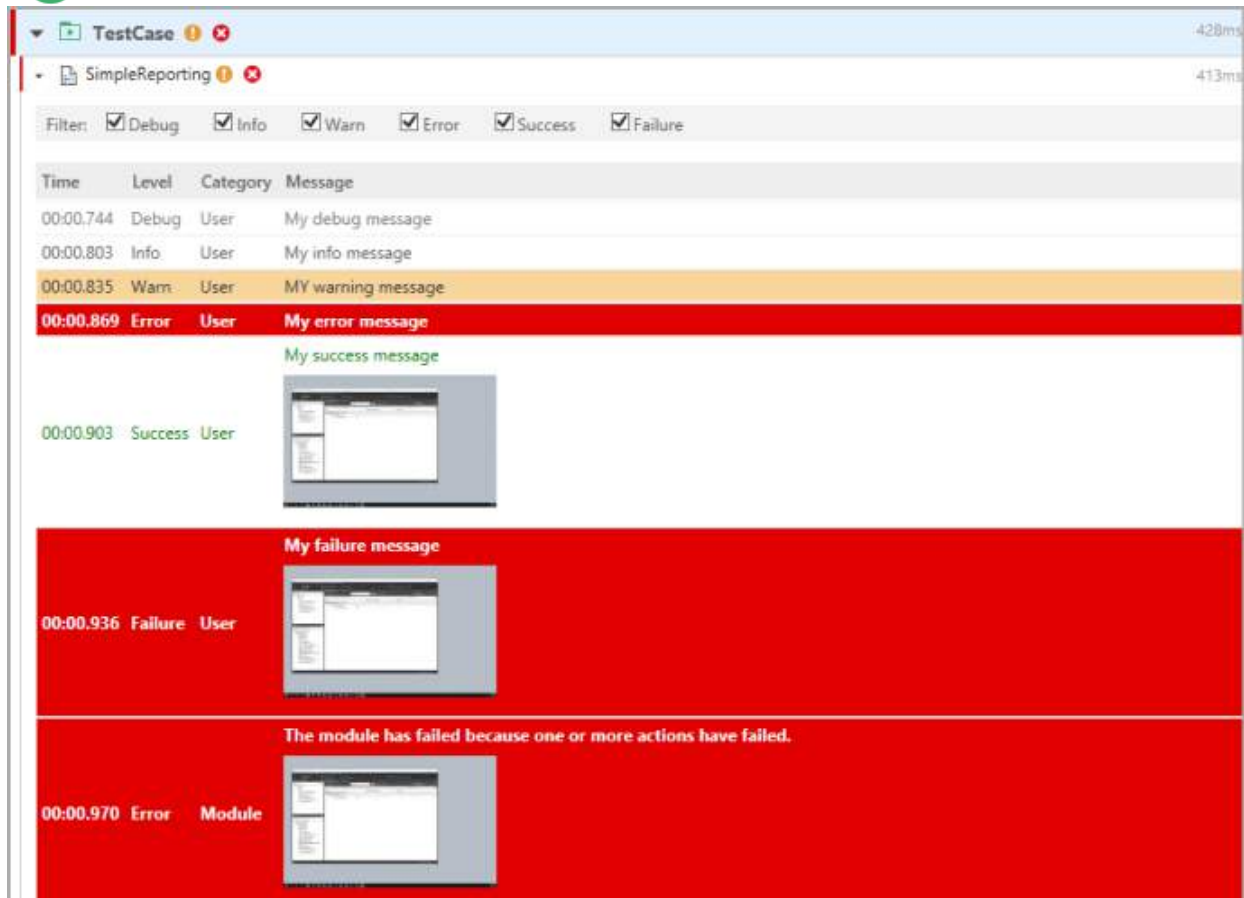
```
Ranorex.Report.Debug("Debug message");
```

```
Ranorex.Report.Info("Information message");
Ranorex.Report.Warn("Warning message");
Ranorex.Report.Error("Error message");
Ranorex.Report.Success("Success message");
Ranorex.Report.Failure("Failure message");
```

- 1 **Create** a code module and open it.
- 2 **Add** the class instantiations for the standard report classes in the Run() method.



- 3 **Run** the code module from the test suite to see the results in the report.





## Hint

Make sure the report level of the test suite is set to **Debug** to display all report messages.

## Report with user code actions

In addition to using code modules, you can also create report messages in recording modules with user code actions.



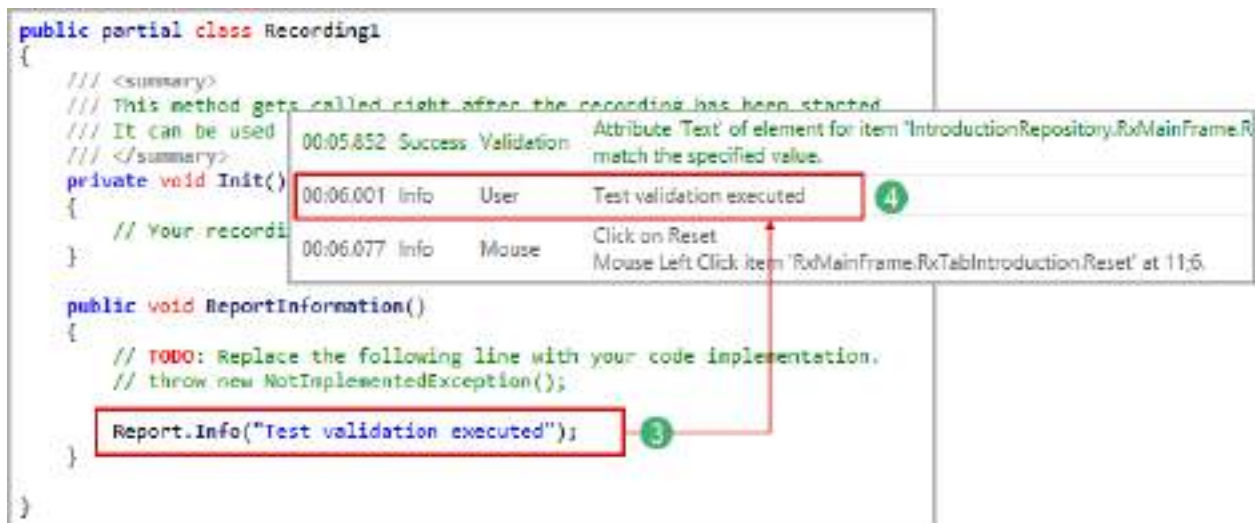
## Further reading

To learn more about user code actions, refer to:  
Ranorex Studio fundamentals > Actions > [User code actions](#).

- 1 **Open** a recording module.
- 2 **Insert** a new user code action called `ReportInformation()` and **open** it.

#	Action					Comment
1	Run application	..\RxDemoApp.exe				Starts demo application
2	Mouse	Click	Left	Relative	EnterYourName	Click into text field
3	Key sequence	Harry			EnterYourName	Insertion of name
4	Mouse	Click	Left	Relative	BtnSubmitUse...	Click on Submit button
5	Validate	AttributeEqual	Text	Welcome, H...	LblWelcomeM...	Test validation
6	User code	ReportInformation()				
7	Mouse	Click	Left	Relative	Reset	Click on Reset
8	Mouse	Click	Left	Relative	RxButtonExit	Closing demo applica...

- 3 **Add** a new report information message to the constructor of the user code action.
- 4 **Run** the recording module to see the report message in the test report.



## Customize the default report category

By default, the category of custom report messages is **User**. You can change this to a different value, either on a per-message basis or for all messages until you change it back.

### Per message

- 1 Create a new code module and open it.
- 2 Add the code shown below.



- 1 This report message appears under the category **Cat.A**.
- 2 This report message appears under the default category **User**.

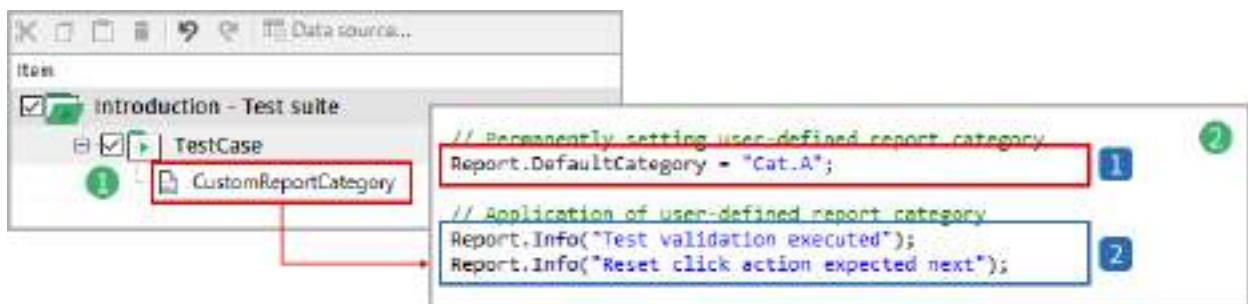
### Result:

Time	Level	Category	Message
00:00.764	Info	Cat.A	Test validation executed
00:00.826	Info	User	Reset click action expected next

- 3 The report messages with their categories as defined above.

### Change default category

- 1 Create a new code module and open it.
- 2 Add the code shown below.



- 1 This line defines **Cat.A** as the new value for the default category for user-defined report messages.
- 2 These report messages appear under the new default category.

### Result:

Time	Level	Category	Message
00:00.712	Info	Cat.A	Test validation executed
00:00.782	Info	Cat.A	Reset click action expected next

- 3 User-defined messages appear under the category **Cat.A** until you change it.

## Define custom report levels

You can define report levels with a custom name and value.

- 1 **Define** the custom report levels **MID** and **LOW** using the code below.
- 2 **Create** two user-defined report messages with the new report levels.

```
// Definition of user-defined report levels
// =====
ReportLevel ReportLow = new ReportLevel("LOW", 35, null);
ReportLevel ReportMid = new ReportLevel("MID", 55, null);

// Application of user-defined report levels
Report.Log(ReportMid, "Test validation executed");
Report.Log(ReportLow, "Reset click action expected next");
```

**Result:**

Time	Level	Category	Message
00:00.747	MID	User	Test validation executed
00:00.803	LOW	User	Reset click action expected next

- 1 The report messages appear with the new report levels.

## Format custom report levels

- 1 **Add** the code below to apply custom CSS formatting to a report level.

```
// Specifying report message style
// =====
ReportLevel ReportLow = new ReportLevel("LOW", 35, null);
ReportLevel ReportMid = new ReportLevel("MID", 55, "color:blue; font-weight:bold");

// Application of user-defined report levels
Report.Log(ReportMid, "Test validation executed");
Report.Log(ReportLow, "Reset click action expected next");
```

Time	Level	Category	Message
00:00.741	MID	User	Test validation executed
00:00.812	LOW	User	Reset click action expected next

- 1 See the formatted result in the test report

## Set threshold report level with custom report levels



In this example, you'll set the threshold report level for messages using a custom report level.

- 1 Define two custom report levels.
- 2 **Set** the current threshold report level to **MID and higher**.
- 3 **Create** two report messages, one with **LOW** and one with **MID** as report levels.

```
// Setting current report level
// =====
ReportLevel ReportLow = new ReportLevel("LOW", 35, null);
ReportLevel ReportMid = new ReportLevel("MID", 55, null);

// Setting current level to value = 55
Report.CurrentReportLevel = ReportLevel.Parse("Activate MID and higher;55");

// Application of user-defined report level
Report.Log(ReportMid, "Test validation executed");
Report.Log(ReportLow, "Reset click action expected next");
```

#### Result:



The screenshot shows the 'CustomReportLevel' window in a testing application. The 'Filter' is set to 'MID'. A table displays the report results, with one message highlighted by a red box and a blue '1' icon.

Time	Level	Category	Message
00:00.908	MID	User	Test validation executed

- 1 Only report messages with the report level **MID** and higher appear in the report.

#### Override current report level

You can override the current report level with the special report level **Always**.



```

// Overriding current report level
// =====
ReportLevel ReportLow = new ReportLevel("LOW", 35, null);
ReportLevel ReportMid = new ReportLevel("MID", 55, null);

// Setting current level to value = 55
Report.CurrentReportLevel = ReportLevel.Parse("Activate MID and higher;55");

// Application of user-defined report level
Report.Log(ReportMid, "Test validation executed");
Report.Log(ReportLevel.Always, "Frog-Admin", "Reset click action expected next");

```

- 1 **Define** two custom report levels.
- 2 **Set** the current threshold report level to **MID and higher**.
- 3 **Create** two report messages, one overriding the current report level.

#### Result:

Time	Level	Category	Message
00:00.755	MID	User	Test validation executed
00:00.822	Always	Frog-Admin	Reset click action expected next

- 1 The second message appears regardless of the current report level.

#### Report screenshots

- 1 **Add** the code below to send a screenshot to the report. If you don't specify a repository item, Ranorex Studio takes a screenshot of what's visible at the time of code execution.

```

// Inserting a desktop screenshot into the report
Report.Screenshot();

```

Time	Level	Category	Message
00:00.898	Info	Screenshot	

- 1 The desktop screenshot in the test report.



### Further reading

To learn how to take screenshots of specific repository items, refer to Ranorex Studio expert > [Code modules](#).

## Report a system summary

- 1 Add the code below to display a system summary in the report.

The screenshot shows a code editor with the following text:

```
// Inserting system summary into the report  
// -----  
Report.SystemSummary();
```

A red box highlights the `Report.SystemSummary();` line, with a green circle containing the number 1 next to it. An arrow points from this line to a message box in the report viewer.

The message box has the following columns: Time, Level, Category, and Message. The message is:

Time	Level	Category	Message
00:00.884	Info	System Summary	Host: 888-888-8888 Ranorex version: 8.1.0.0 OS version: Windows 10 64bit OS default locale: en-US .NET Runtime version: 4.0.30319.42000 Number of logical CPUs: 4 Number of displays: 3 Screen dimensions: 5760x1200 Memory: 45% load, phy 8802 MiB free/16239 MiB total, virt 1490 MiB free/2047 MiB total

A red box highlights the message content, with a blue circle containing the number 1 next to it.

- 1 The system summary appears as an **Info** message in the report.

## Add customized data

You can collect customized data during a test run and write it to the raw data file (.rxlog.data) that's used to generate the final report.

### Add custom data to raw data file

Customized data collection is triggered by a user code action with a method that tracks customized data.

- 1 **Define** an object that references the current activity stack of Ranorex Studio.  
The activity stack is where all activities during a test run are collected by means of a stack data structure.
- 2 **Add** an activity to the activity stack.  
The activity stack method **CustomProperties** adds a reported activity to the activity stack. It is defined by two strings, name (e.g. **myName**) and value (e.g. **myValue**).

```
void ITestModule.Run()
{
    Mouse.DefaultMoveTime = 300;
    Keyboard.DefaultKeyPressTime = 100;
    Delay.SpeedFactor = 1.0;

    // Define an object on the current activity stack of Ranorex
    var currentActivity = Ranorex.Core.Reporting.ActivityStack.Current;

    // Add customized data to the activity stack and therefore to the XML data file
    currentActivity.CustomProperties["myName"] = "myValue";
}
```

1

2

#### Result:

- The result is a custom data field pair in the corresponding data file

```
109 iteration="1"
110 activity-exectype="execute"
111 testentry-activity-type="testmodule">
112 <custom
113     myName="myValue">
114 </CUSTOM>
115 <varbindings>
116 </varbindings>
117 </activity>
118 </activity>
119 </activity>
120 </activity>
121 </report>
```

1

- 1 The raw data file contains the custom data.
  - If you want to apply the test data outside of Ranorex Studio report generation, you can parse the XML file as needed.
  - To include the custom data in a Ranorex Studio report, please refer to the next section.

## Include custom data in report



### Further reading

If you are not familiar with XML and the corresponding concepts of CSS and XSL specifications, we recommend the [website](#) of the World Wide Web Consortium (W3C) for more information.

To include custom data in Ranorex Studio reports, you must modify the corresponding XSL specification file of the report. You can do this in any XML editor or in Ranorex Studio.

- 1 **Add** the code in the picture below to the file.

```
588 <i>
589   <xsl:value-of select="./detail" />
590 </i>
591
592 <!-- CUSTOM DATA SECTION -->
593 <xsl:if test="./custom">
594   <i>
595     MyTestData:
596     <xsl:value-of select="./custom/@myName"/>
597   </i>
598 </xsl:if>
599
600 <span class="duration">
601   <xsl:value-of select="./@duration" />
602 </span>
603 </a>
```

- 1 This line identifies the field pair where the custom data is located in the raw data file.
- 2 This optional line adds a header that will be displayed before the custom data.
- 3 This line gets the field value of the corresponding field name.

## Result:

```
588 <i>
589   <xsl:value-of select="./detail" />
590 </i>
591
592 <!-- CUSTOM DATA SECTION -->
593 <xsl:if test="./custom"> 1
594   <i>
595     MyTestData: 2
596     <xsl:value-of select="./custom/@myName"/> 3
597   </i>
598 </xsl:if>
599
600 <span class="duration">
601   <xsl:value-of select="./@duration" />
602 </span>
603 </a>
```

- 1 The report contains the custom data.

## Convert existing reports to PDF

This article describes how to use the ReportToPDF tool in order to convert existing Ranorex Studio report files to PDF. This allows Ranorex Studio reports to be viewed on computers where Ranorex Studio is not installed.

### Note

If you want Ranorex Studio to convert reports to PDF as they are being generated during test execution, use the [ReportToPDFModule](#) contained in the Ranorex Studio Automation Helpers.

### Screencast

The screencast “Converting reports” walks you through the information found in this chapter.

[Watch the screencast now](#)

## Download the ReportToPDF tool

The ReportToPDF tool is available as a standalone executable:

- [Ranorex\\_PDF\\_Executable.zip](#): executable for conversion of existing report files (\*.rxzlog).
- [style.zip](#): stylesheet to generate the PDF.

## Convert an existing report

The ReportToPDF executable is a command line tool that allows you to convert existing Ranorex Studio reports to PDF. The report must be passed to the converter in its [compressed form](#), the so-called **rxzlog**. The **rxzlog** is a single archive (.rxzlog extension) including the report and all associated files.

Use the following call to execute the ReportToPDF tool from the command line:

```
Report.PDF.exe [input file] [output file] /[argument]
```

The input and output file are required; the arguments are optional.

**[input]**: Sets the file (\*.rxzlog) which is converted.

**[output]**: Sets the name of the PDF file.

## Allowed arguments

**style**: Sets a custom stylesheet.

**detail**: Specifies how much information is shown in the generated PDF:

- **none**: No module actions are shown.
- **failed**: Only actions from failed modules are shown.
- **all**: All actions are shown (default value).