RANOREX

INTERFACES AND CONNECTIVITY USERGUIDE

TABLE OF CONTENTS

INTERFACES AND CONNECTIVITY 4

Ranorex Instrumentation Wizard 5

Mozilla Firefox 15

Troubleshooting 19

Google Chrome/Microsoft Edge 20

Install Ranorex Automation Add-on in Edge 22

Manual installation 26

Important notes for testing with Google Chrome/MS Edge 26

Test offline/local HTML files ("Allow access to file URLs") 26

Google Chrome 27

MS Edge 27

Android apps 28

Prepare the app and the mobile device 28

Prepare the app 28

Prepare the mobile device 28

Instrumentation example 29

Select technology 29

Specify APK and settings 30

Add APK file 30

Set instrumentation parameters 31

Result 33

Advanced Android instrumentation settings 34

iOS apps 38

Prepare the app and the mobile device 38

Select technology 39

Specify IPA and settings 40

Configure Manually 42

Result 44

Technology instrumentation 47

Flash/Flex/Air 48

Qt testing 57

Legacy applications 59

SAP applications 60

CEF applications 62

General Troubleshooting 64

Source and revision control 71

Git 71

Subversion 79

Team Foundation Version Control (TFVC) 89

Ranorex Magic Merger 98

Azure DevOps Integration 102

Create ADO project and Git repo 103

Add Ranorex Studio solution to DevOps project 109

Set up an Azure Pipelines agent 113

Create and execute a pipeline 116

Continuous integration & test management 133

Jira integration 136

TestRail integration 147

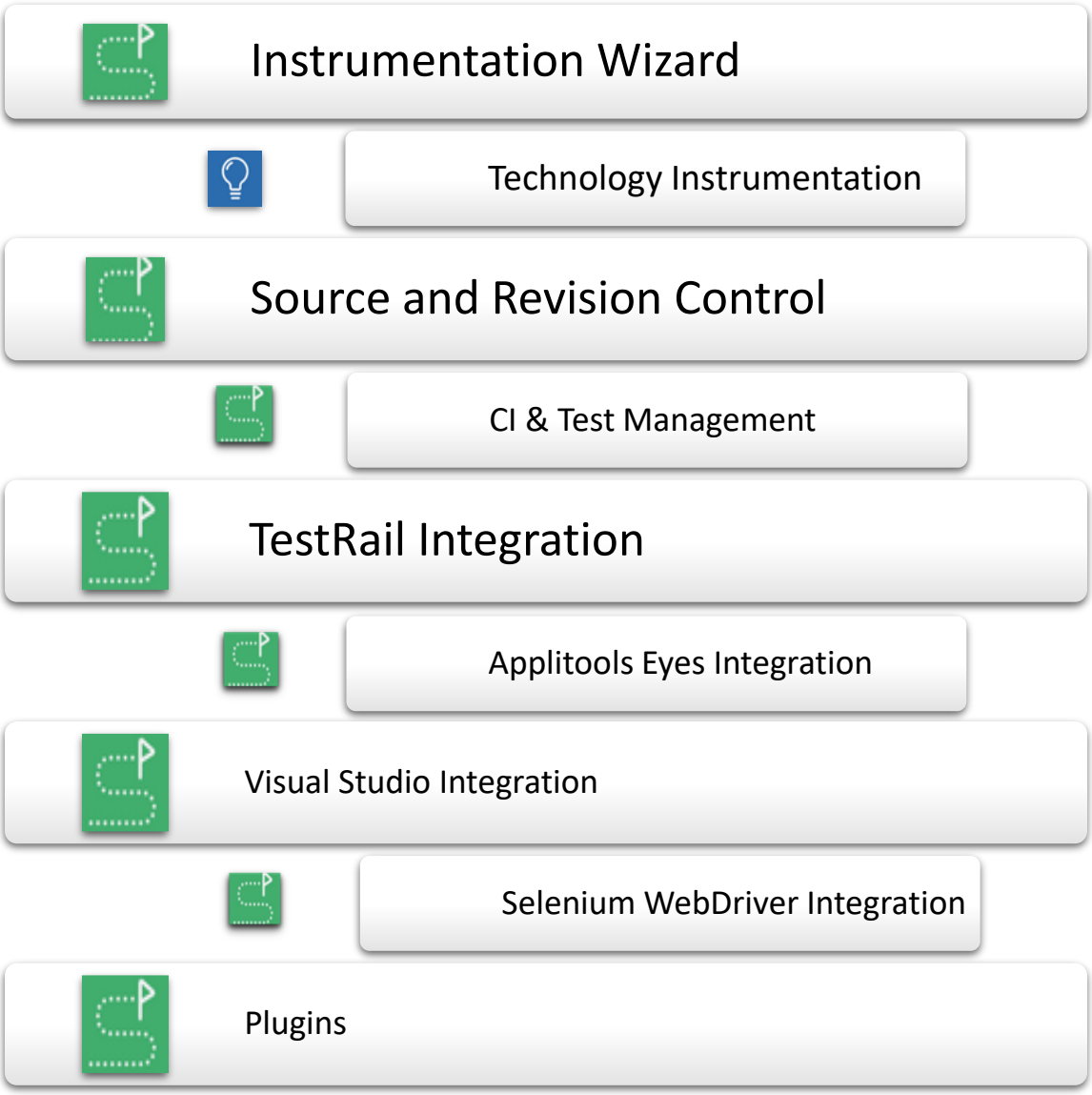
Importing, exporting, synchronizing 150

Running tests and reporting to TestRail 156

Collaboration *159*
APPLITOOLS EYES INTEGRATION 161
Create tests *161*
Run tests *166*
JENKINS INTEGRATION 168
Install and configure plugins *171*
Create and set up a Jenkins project *176*
Build a Jenkins project *186*
VISUAL STUDIO INTEGRATION 190
SELENIUM WEBDRIVER INTEGRATION 198
PLUGINGS 201
Improved WPF plugin *201*
Delphi plugin *204*

Interfaces and connectivity

Interfaces and connectivity contain all tools, methods, and concepts which are needed to connect Ranorex to third-party software, hardware systems, and different technologies. A powerful instrumentation wizard supports you in integrating different technologies. Test deployment and remote work with Ranorex are connectivity topics as well as the concept of technology instrumentation.

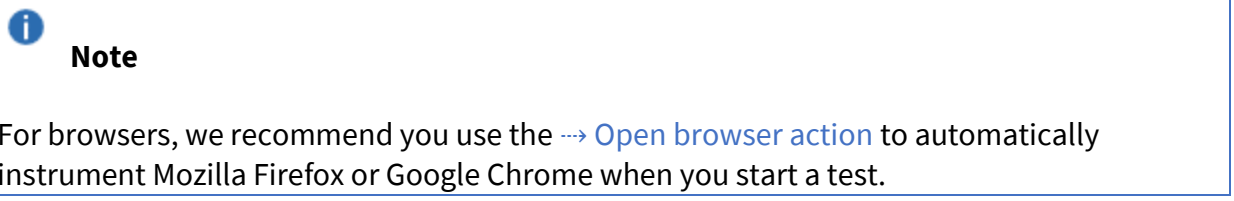


Ranorex Instrumentation Wizard

Ranorex Studio supports test automation for many different UI technologies. Some of these technologies must be instrumented for best automation results with Ranorex Studio. Instrumenting means that Ranorex Studio installs an add-on or inserts code so it can identify the UI elements of a particular technology.

The Ranorex Instrumentation Wizard simplifies this process for the following technologies:

- Mozilla Firefox
- Google Chrome
- Microsoft Edge
- Chromium
- Android/iOS apps



Note

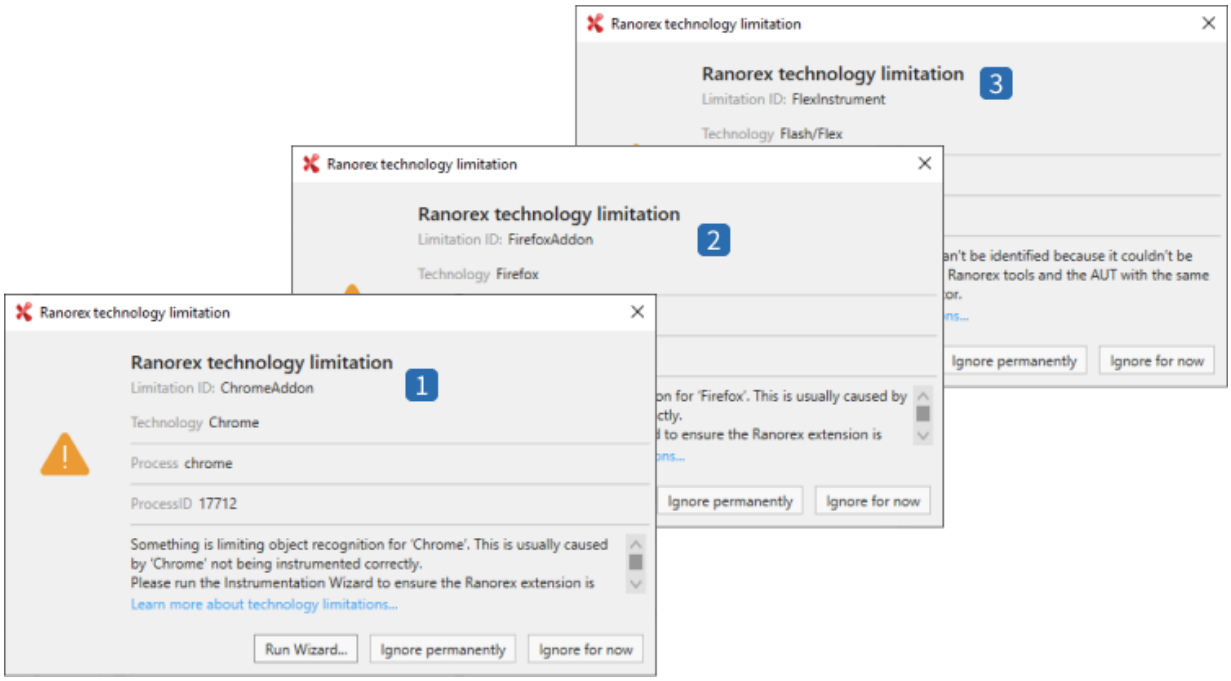
For browsers, we recommend you use the ⇢ Open browser action to automatically instrument Mozilla Firefox or Google Chrome when you start a test.

Start the Instrumentation Wizard

Here we'll show you the different ways that you can start the Instrumentation Wizard.

From technology limitation warnings

Whenever you try to automate one of the above technologies without having instrumented them, a technology limitation warning will appear. It will include a button to directly run the Instrumentation Wizard.



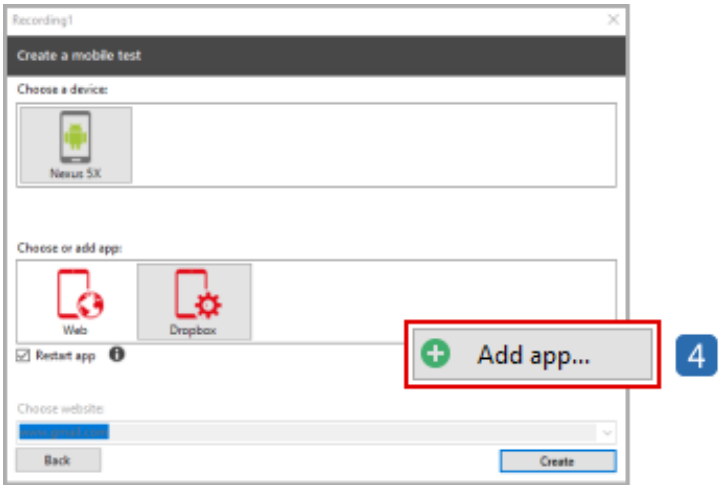
1 Warning for Google Chrome

2 Warning for Mozilla Firefox

3 Warning for Flash/Flex

When you create a mobile test

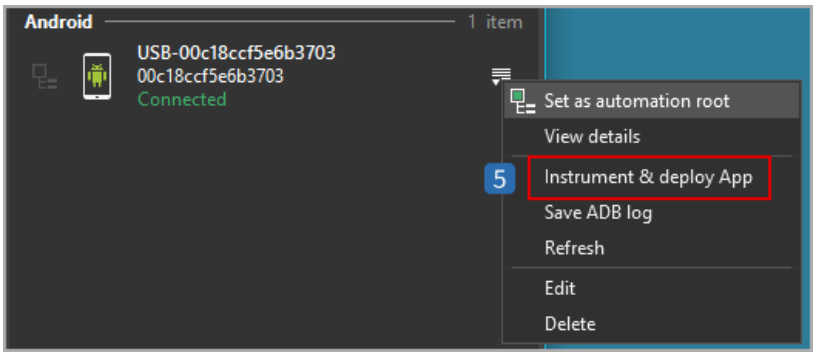
When you click Add app… in the mobile test creation dialog, the Instrumentation Wizard starts automatically. This is because apps need to be instrumented before you can deploy them to your mobile device.



4 Add app… button that automatically starts the Instrumentation Wizard.

From the settings of an iOS/Android endpoint

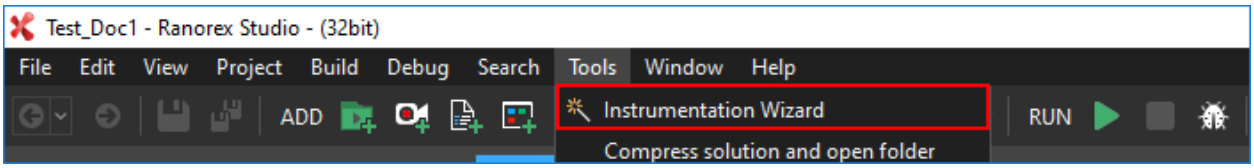
You can also start the Instrumentation Wizard from the settings of an Android/iOS endpoint. This works the same way as the Add app… button above.



5 Instrument/deploy button in the settings of an Android endpoint

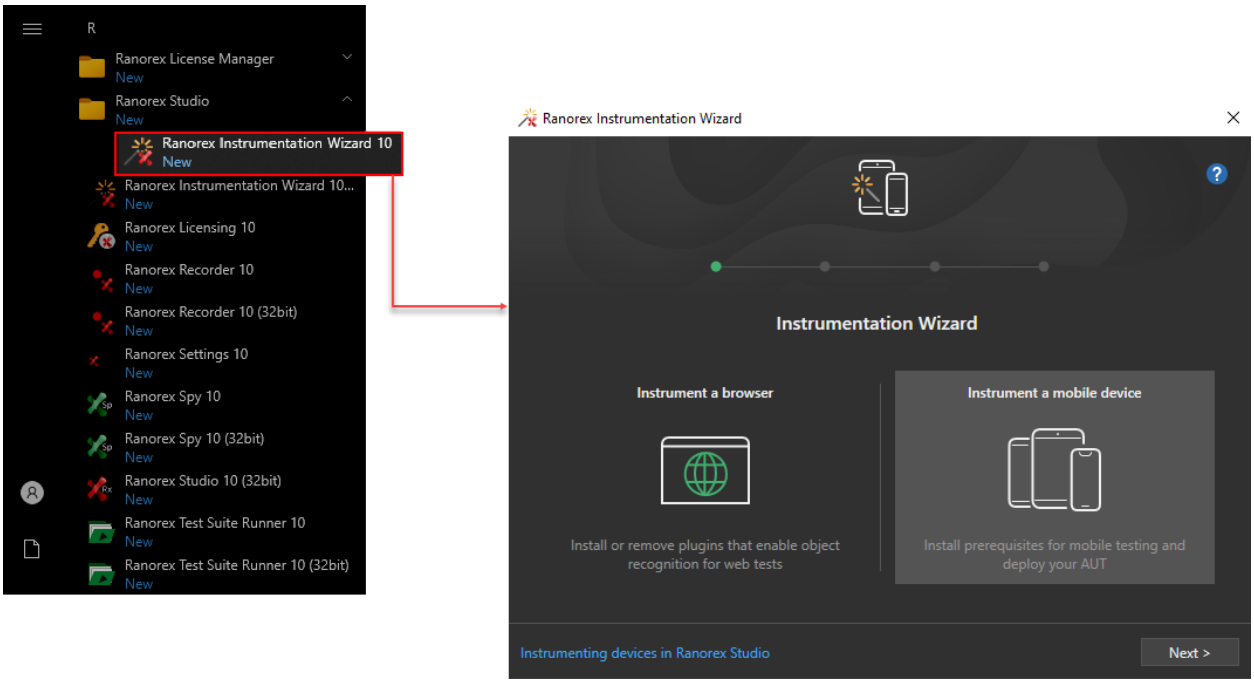
From the menu

You can also start the Instrumentation Wizard **from Tools > Ranorex Instrumentation Wizard.**



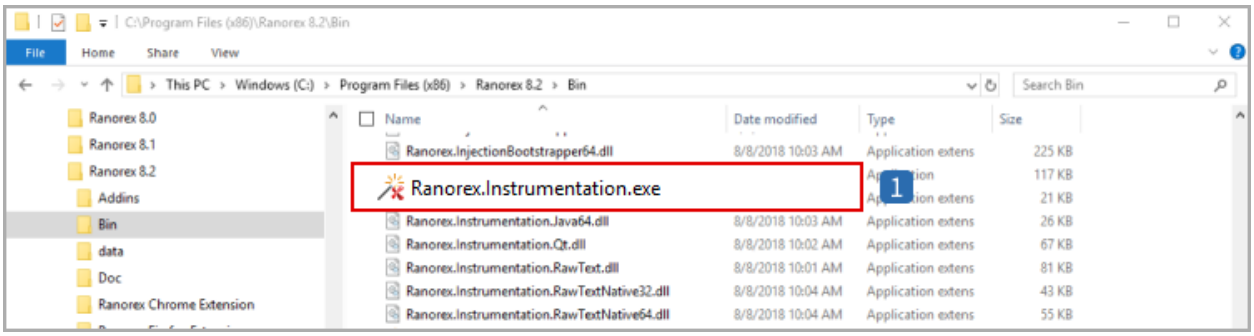
From the Windows Start menu

The Instrumentation Wizard is a standalone program and can therefore be started from the Windows Start menu.



Command line instrumentation

You can also instrument technologies from the command line. The available arguments are listed below. The image shows the Instrumentation Wizard executable in the /bin/ folder of the Ranorex Studio installation.

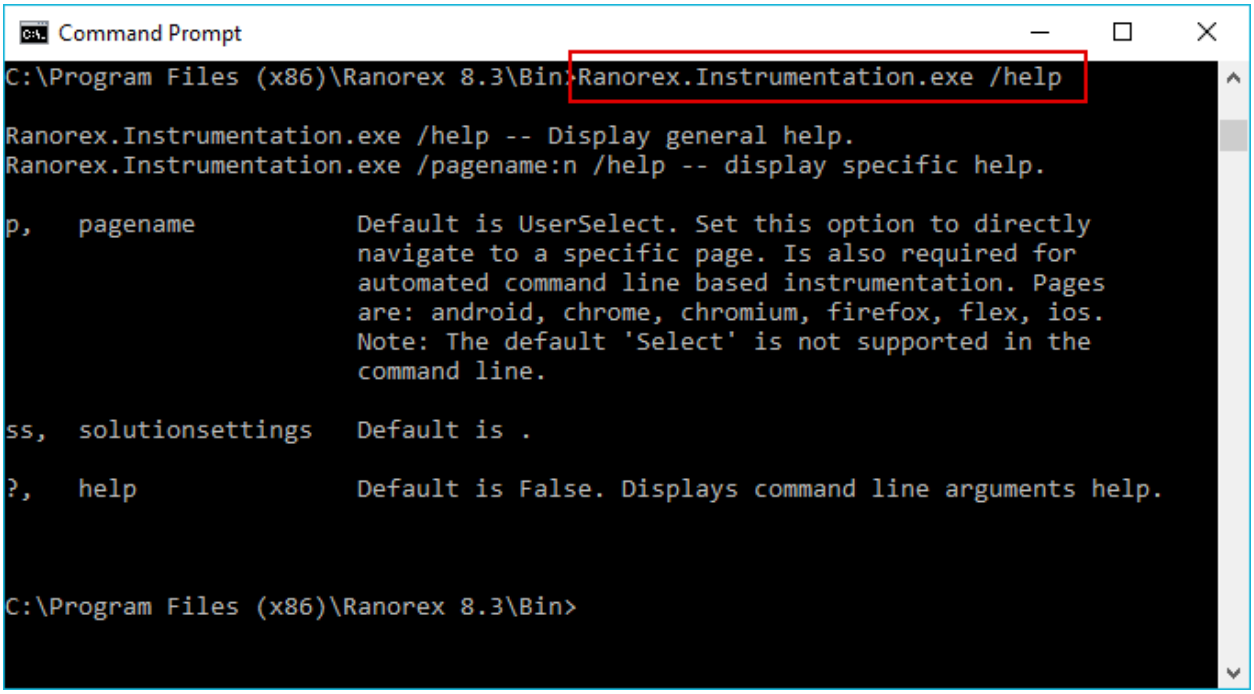


1 Instrumentation Wizard executable

?|help

Displays the general help. Add a **page** parameter to show the help for the respective technology.

Possible page names: **android, chrome, firefox, flex, ios**.



Android arguments (basics)

d|device

- Name of the mobile device to which you want to deploy an APK.
- Not required if **deploymode** is set to **NoDeploy**.

a|apkfile

- Path to the APK that you want to deploy

s|skip

- **True** = no instrumentation: only deploys the APK to the device.
- **False** (default) = instruments and then deploys the APK to the device.

dm|deploymode

- Configures how the APK is deployed.
- Parameters: **WiFi** | **Usb** | **NoDeploy** | **Auto** (default)
- **NoDeploy** = Does not deploy the APK.
- **Auto** = Deploys the APK using the same mode as used for device configuration.

o|outfile

- Optional path or folder to save the instrumented APK.
- You must have read and write permissions for the path/folder.
- Adds a suffix to the saved APK to indicate that it's been instrumented. Overwrites existing instrumented APKs.
- If **skip** is set to **true**, does nothing.

Android arguments (signing APKs)

JdkBinPath

- Complete installation path of the JDK environment.

KeyAlias

- Alias of the **keystore entity** of **jarsigner**.

KeyPass

- Password for **jarsigner**.
- **Attention:** The password is stored as plain text.

KeyStore

- Complete path to the associated **jarsigner keystore**.

KeyStorePass

- Password to the **jarsigner keystore.**
- **Attention:** The password is stored as plain text.

> **Note**
>
> For custom signing, **KeyStore, KeyAlias, KeyStorePass, KeyPass** and **JDKBinPath** must be set.

Android arguments (instrumentation settings)

RIdClass

- Optional class name for locating resource Ids (e.g. com.ranorex.demo.R)
- By default, Ranorex will search in .**R**

EnableWebTesting

- True (default) = shows DOM content of web views in the UI hierarchy.
- False = does not show DOM content.
- Important: Very high performance impact on your app when set to **true**.

FullImageComparison

- Uses more robust image comparison to determine resource IDs for images.
- Important: Decreases startup performance.

TreeSimplification

- **True** (default) = Postprocesses the generated UI tree internally to make it simpler.
- **False** = No postprocessing. **False** generates a larger UI tree. This can be useful for automating 3rd-party Android controls.

Chrome arguments

e|enable

- **True** = Activates the Ranorex Chrome add-on.
- **False** = Deactivates the Ranorex Chrome add-on.

Firefox arguments

enew|enablenew

- **True** = Activates the Ranorex Firefox add-on.
- **False** = Deactivates the Ranorex Firefox add-on.

Flex arguments

pl|preloader

- **True** = Activates the Ranorex preloader.
- **False** = Deactivates the Ranorex preloader.

ie

- **True** = Activates the Flex debug player for Internet Explorer.
- **False** = Deactivates the Flex debug player for Internet Explorer.

ff|firefox

- **True** = Activates the NPAPI Flex debug player for Firefox/Safari.
- **False** = Deactivates the NPAPI Flex debug player for Firefox/Safari.

cr|chrome

- **True** = Activates the Flex PPAPI debug player for Chrome.
- **False** = Deactivates the Flex PPAPI debug player for Chrome.

o|other

- **True** = Activates **cr** and **ff**
- **False** = Deactivates **cr** and **ff**

ft|flextrace

- True = Activates Flash tracelog.
- False (default) = Deactivates Flash tracelog.

iOS arguments

u|udid

- Unique ID of the device to which you want to deploy an app.
- Alternative argument: **dn|devicename**

> **Note**
>
> You can find the UDID in the details of the respective endpoint.

dn|devicename

- Name of the mobile device to which you want to deploy an app.

ip|inputpath

- Complete path of the IPA file you want to instrument/deploy.

o|outfile

- Optional path or folder to save the instrumented IPA.
- You must have read and write permissions for the path/folder.
- Adds a suffix to the saved IPA to indicate that it's been instrumented. Overwrites existing instrumented IPAs.
- If **skipinstrumentation** is set to **true**, does nothing.

si|skipinstrumentation

- True = no instrumentation, only deploys the IPA to the device.
- **False** (default) = instruments and then deploys the IPA to the device.

sd|skipdeployment

- **True** = Does not deploy the IPA.
- **False** (default)= Deploys the IPA.

k|keypath

- Path to the p12 certification file (*.p12).

pw|password

- Password to the p12 certification file (*.p12).

pp|provisionpath

- Path to the embedded mobile provision profile (*.mobileprovision).

ai|appid

- The application ID (bundle identifier of the app).
- Required so that an application can be uninstalled prior to installing the new archive.
- By default, requires you to enter the ID.
- Can also be set to **auto** to resolve the ID automatically during instrumentation.

db|deploybrowser

- True = deploys the preinstrumented Ranorex Web Browser App to the device.
- False (default) = Does not deploy it.

ds|deployservice

- True = deploys the preinstrumented Ranorex Service App to the device.
- False (default) = Does not deploy it.

ub|uninstallbrowser

- True = uninstalls the preinstrumented Ranorex Web Browser App before deployment.
- False (default) = No prior uninstalling.

ub|uninstallservice

- True = uninstalls the preinstrumented Ranorex Service App before deployment.
- False (default) = No prior uninstalling.

uf|uninstallfail

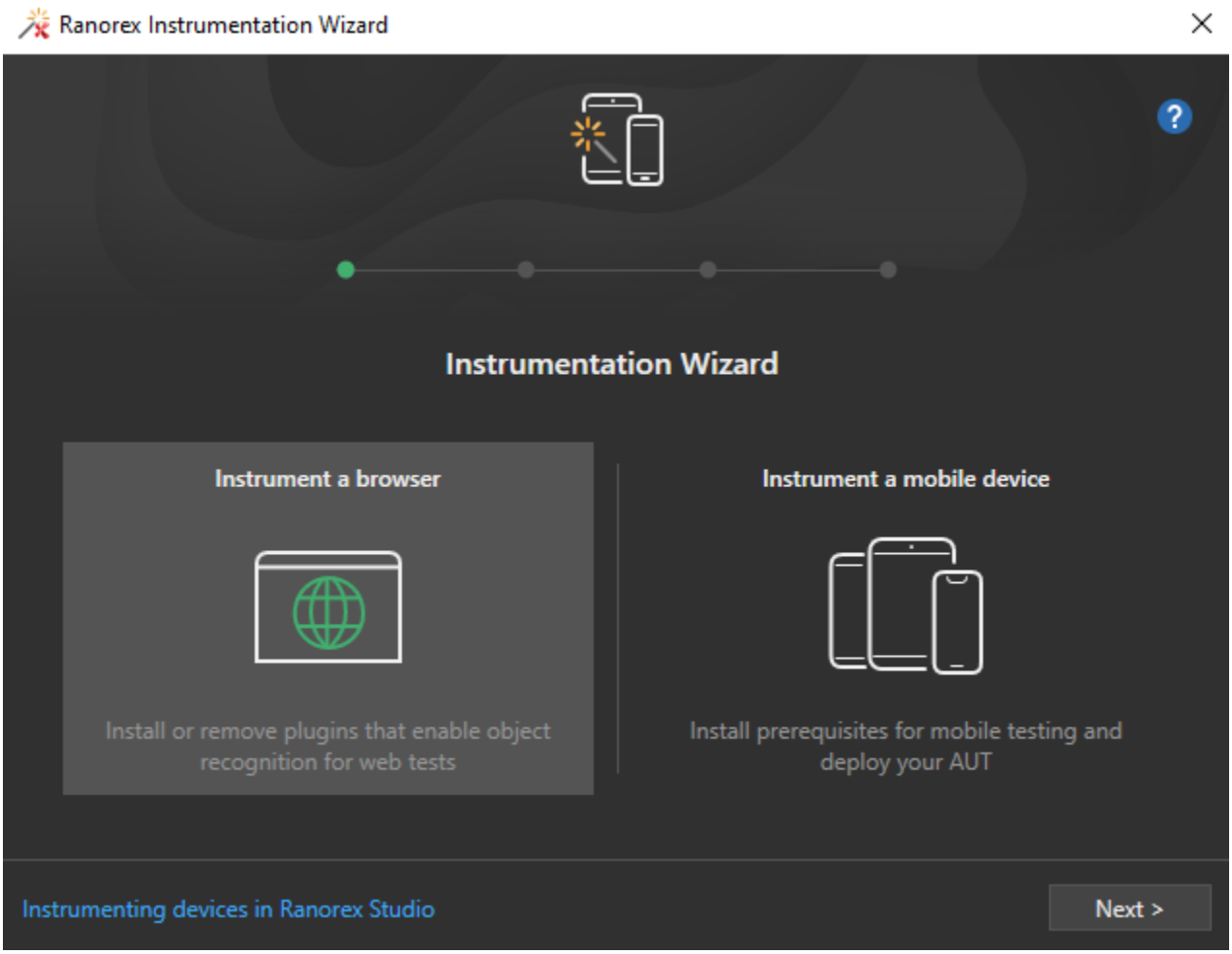
- True = Displays an error message if uninstalling an app fails.
- False (default) = Does not display an error message.

Mozilla Firefox

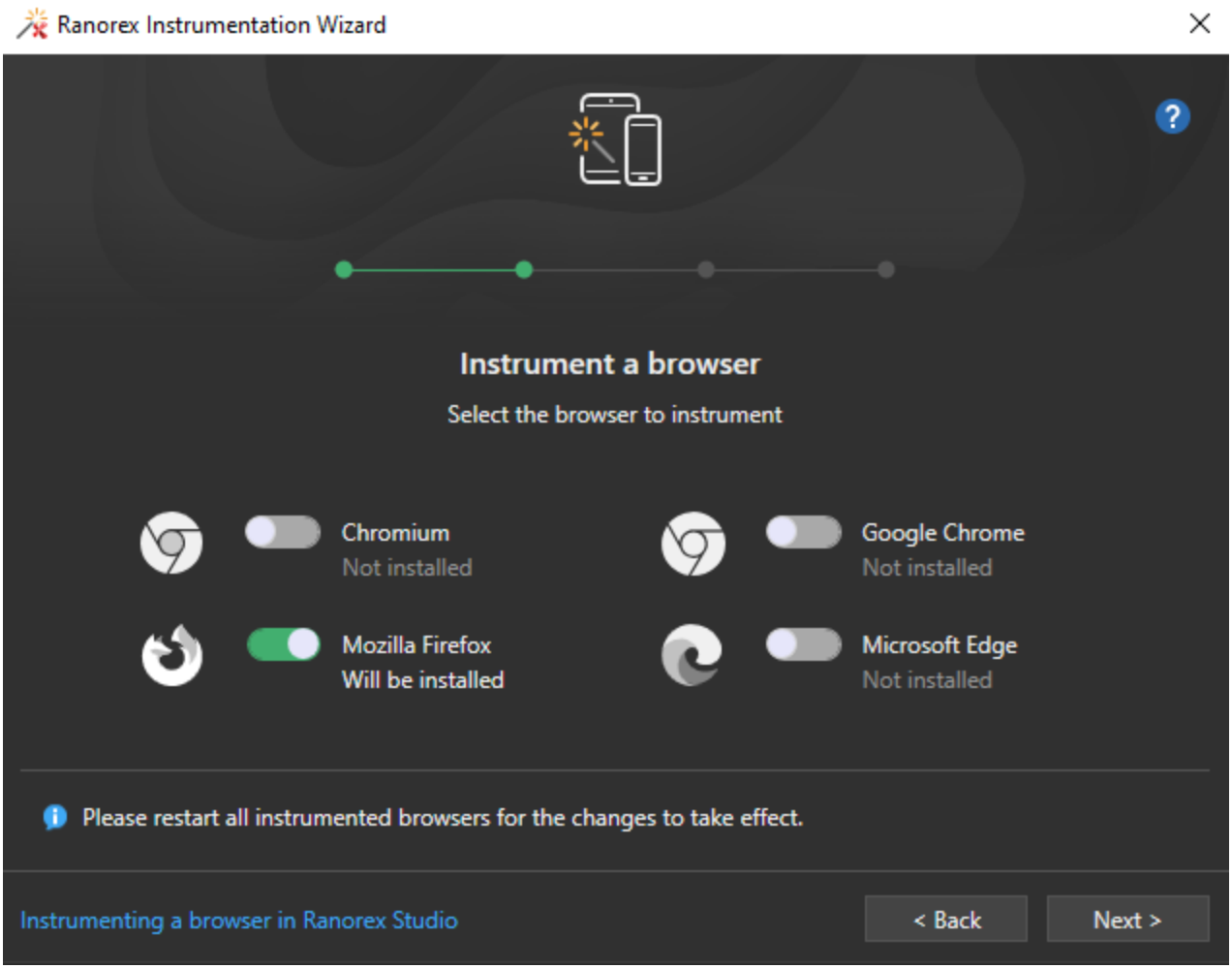
Install Ranorex Automation Add-on

To install the Ranorex Automation add-on, open the instrumentation wizard:

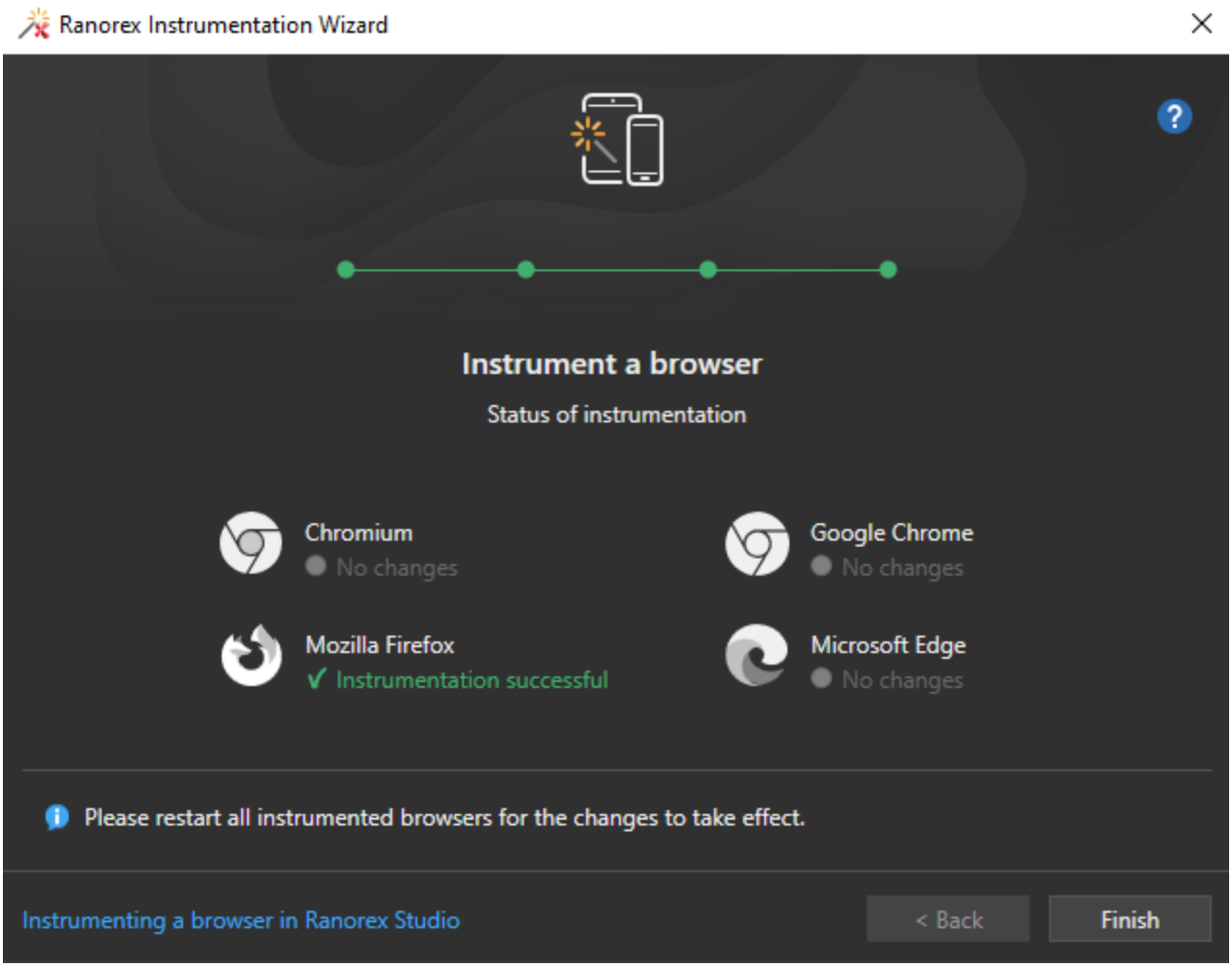
1. **Select Instrument a browser.**



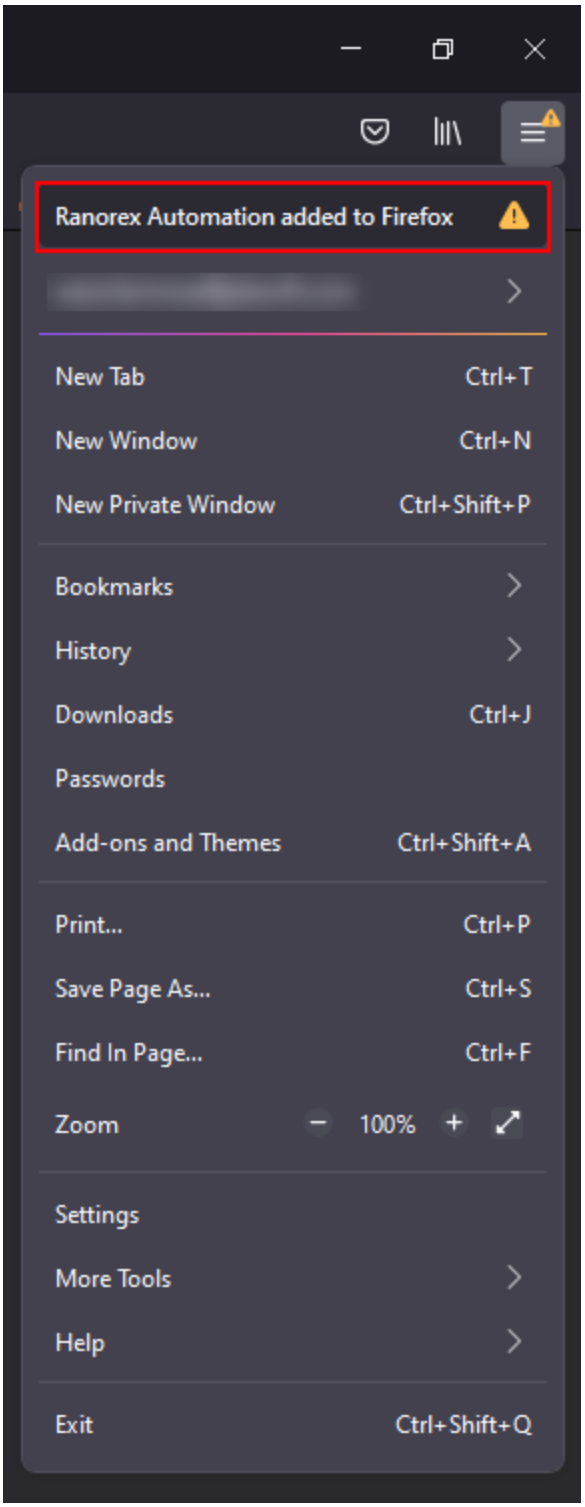
2. Enable the **Mozilla Firefox** slider option.



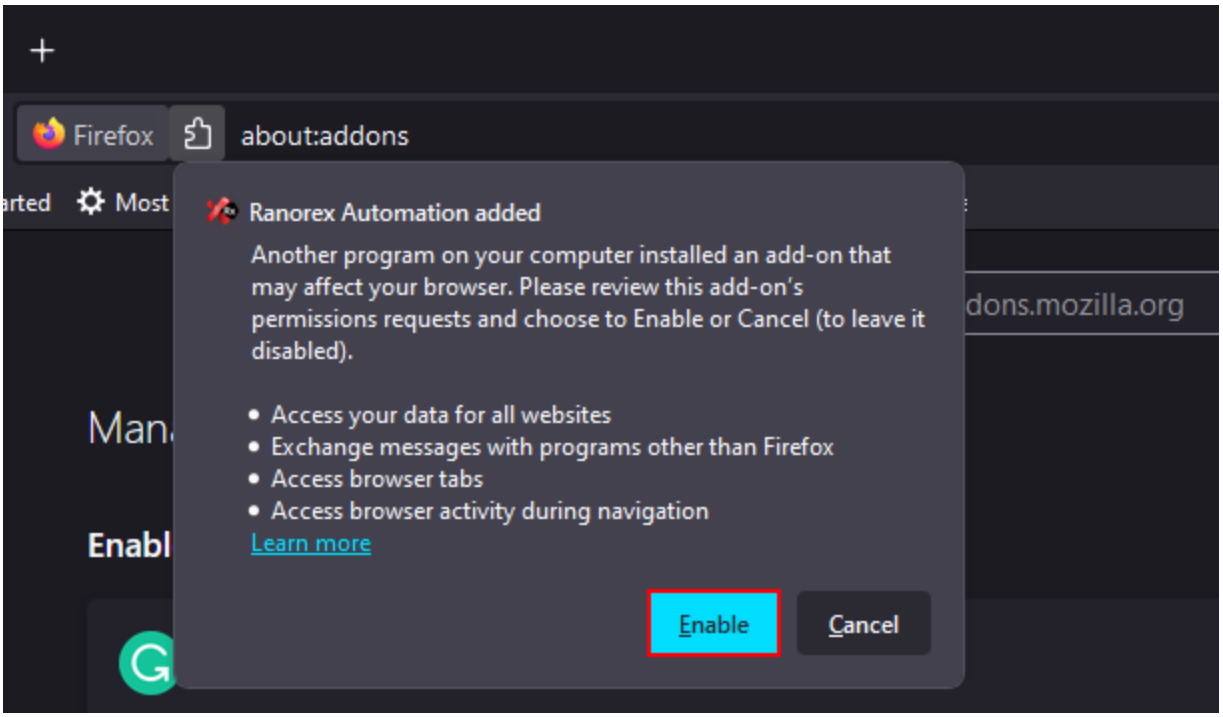
3 After the wizard displays the **Instrumentation successful** message, **click Finish**.



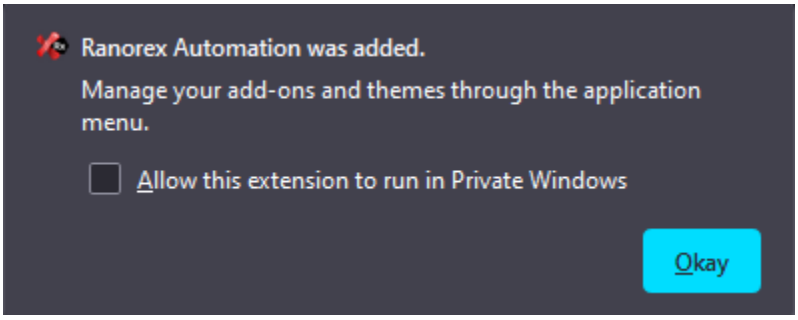
4 Restart your browser. **Click** the browser menu icon that shows a warning symbol and select the **Ranorex Automation added to Firefox** message.



5 **Click Enable** to activate the add-on.



6 The browser displays the following message after the add-on was enabled correctly.



Troubleshooting

If installing the add-on fails, Ranorex Studio will show a message.

Please try the following steps in order, retrying after each step.

1. Run the Instrumentation Wizard as administrator. If starting the wizard from Ranorex Studio, restart Ranorex Studio as administrator first.
2. Close Firefox and try again.
3. Reinstall Firefox.
4. Uninstall Ranorex Studio, restart your computer, then reinstall Ranorex Studio.

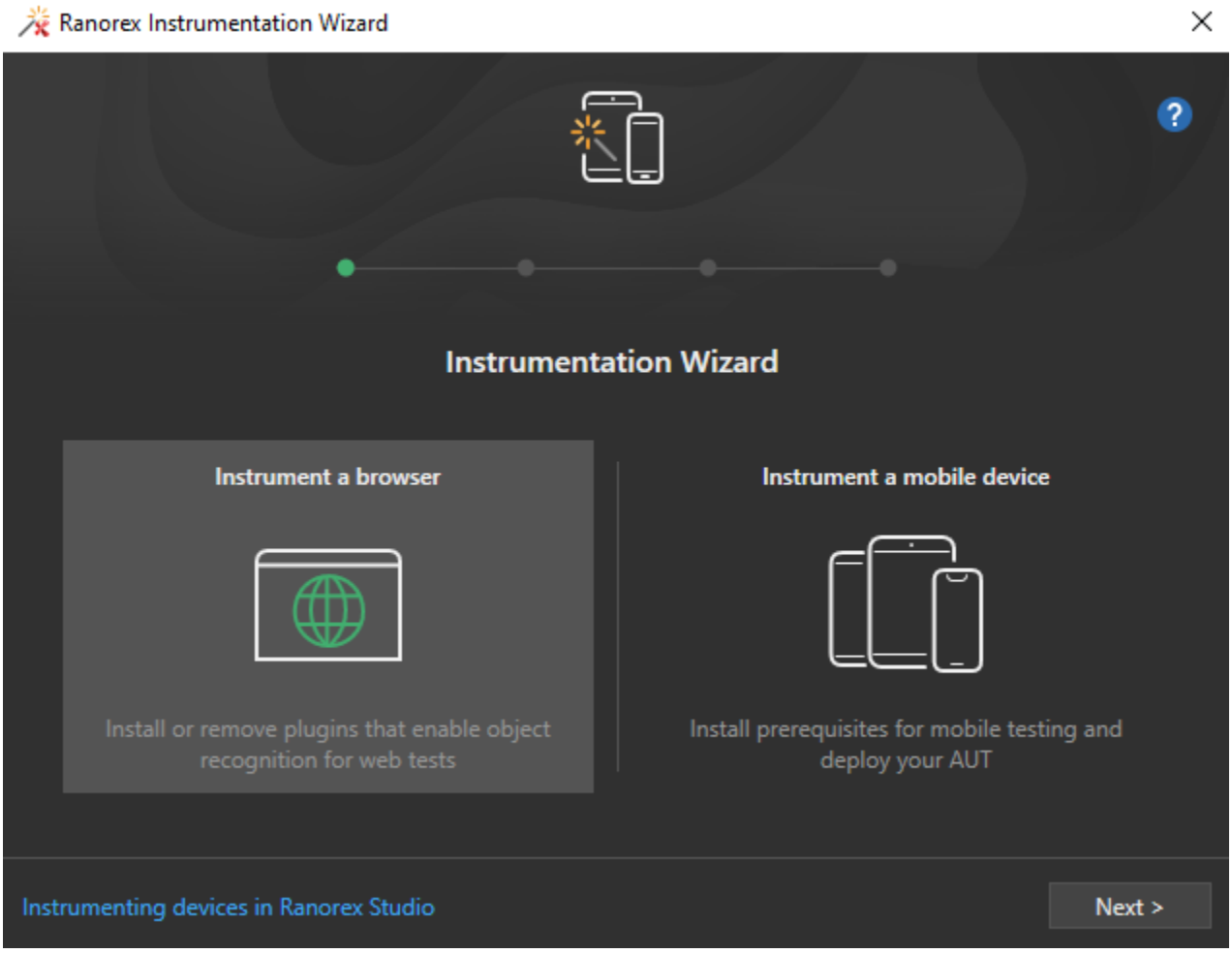
Google Chrome/Microsoft Edge

On this page you'll find out how to instrument Google Chrome and MS Edge based on Chromium with the Instrumentation Wizard. The process is essentially the same. You'll also find some recommendations for settings when automating with these browsers at the end of the page.

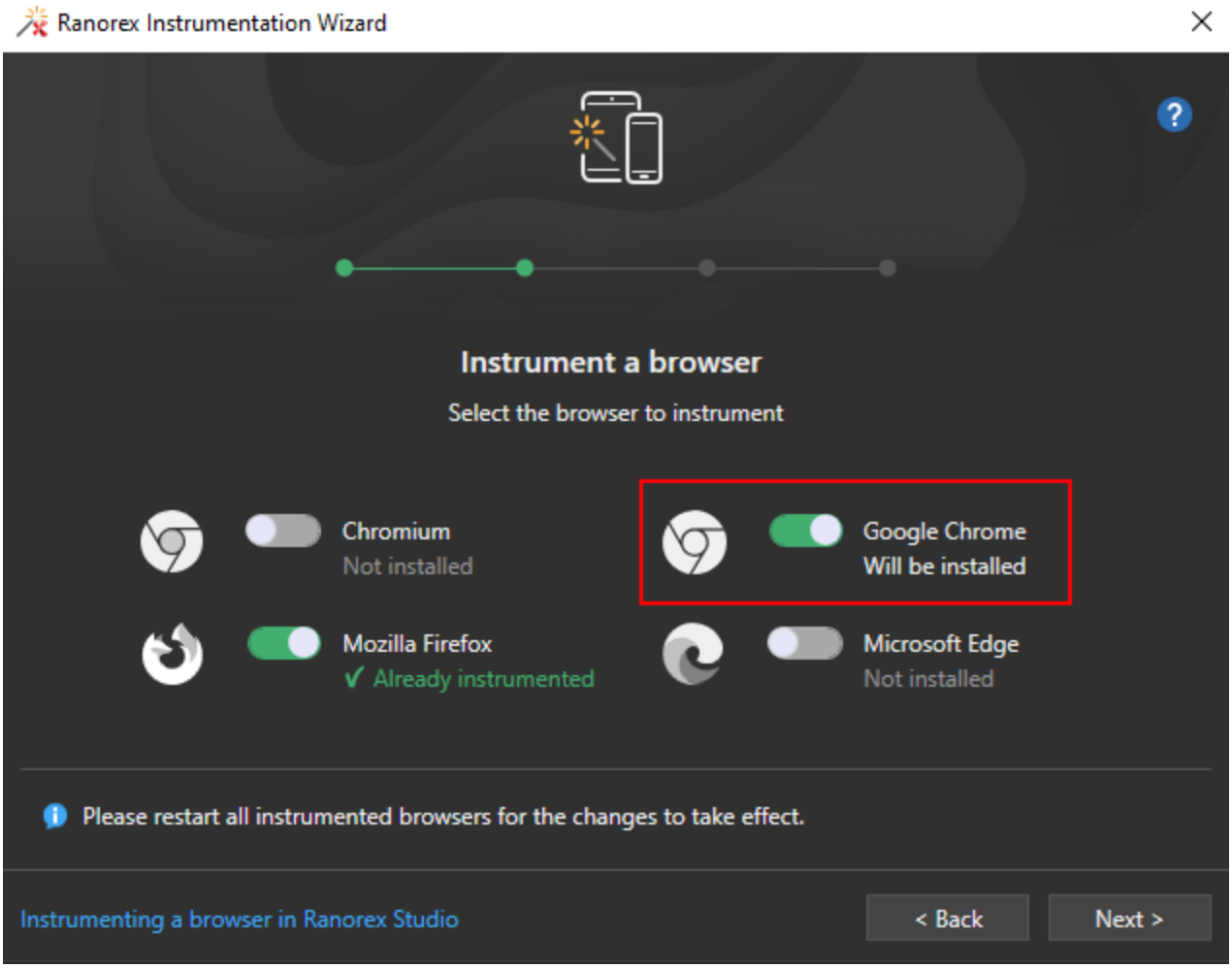
Install the Ranorex Automation Add-on

To install the Ranorex Automation add-on, open the instrumentation wizard:

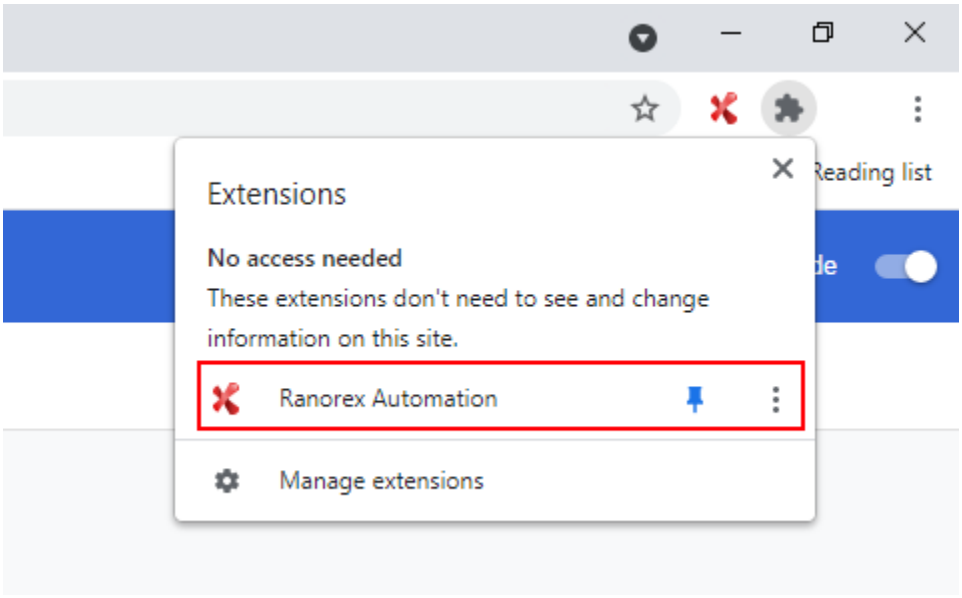
1. **Select Instrument a browser.**



2. Enable the **Chrome** slider option.



3 After the wizard displays the **Instrumentation successful** message, **click Finish**.

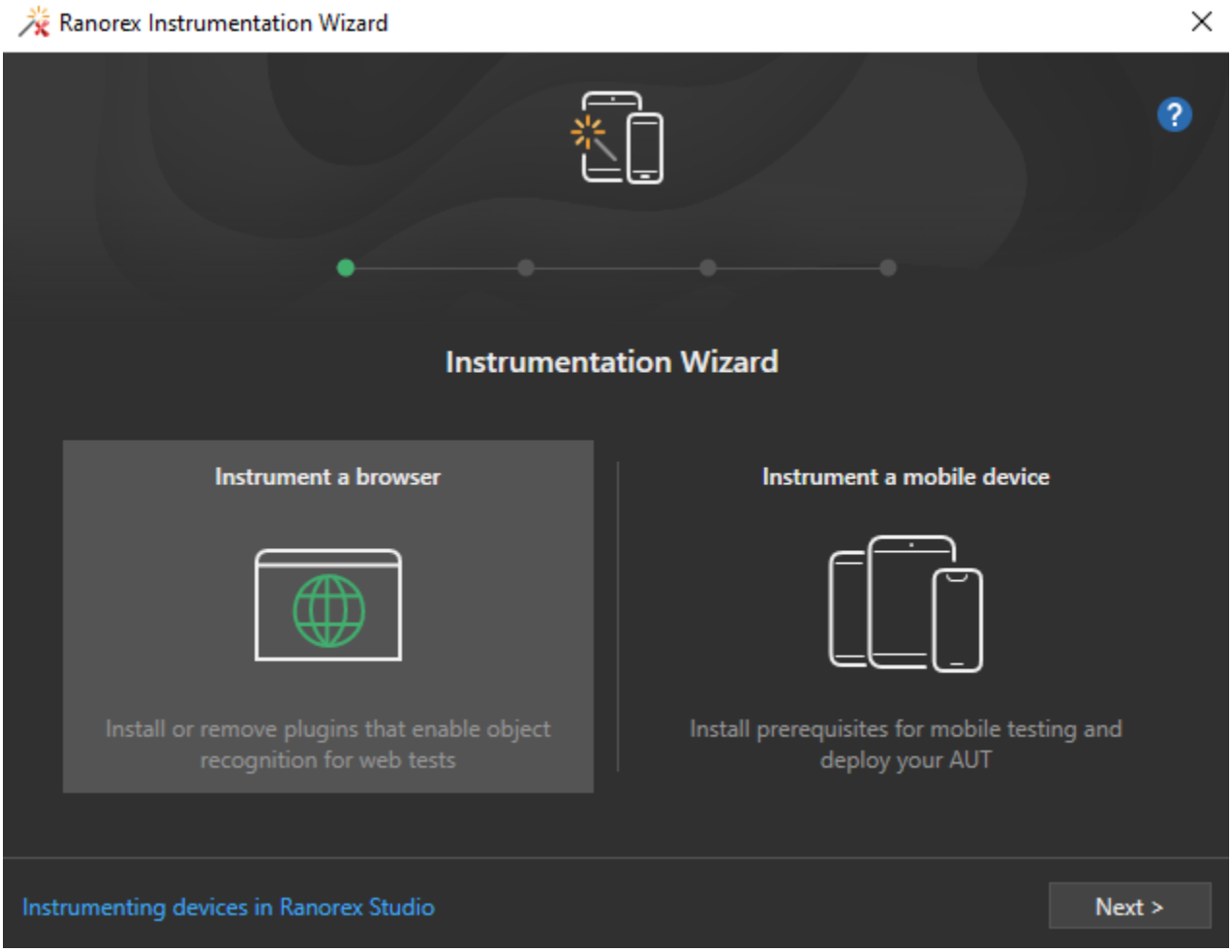


4. Restart your browser. Review that the Ranorex add-on has been installed correctly by clicking the Extensions icon located on the toolbar.

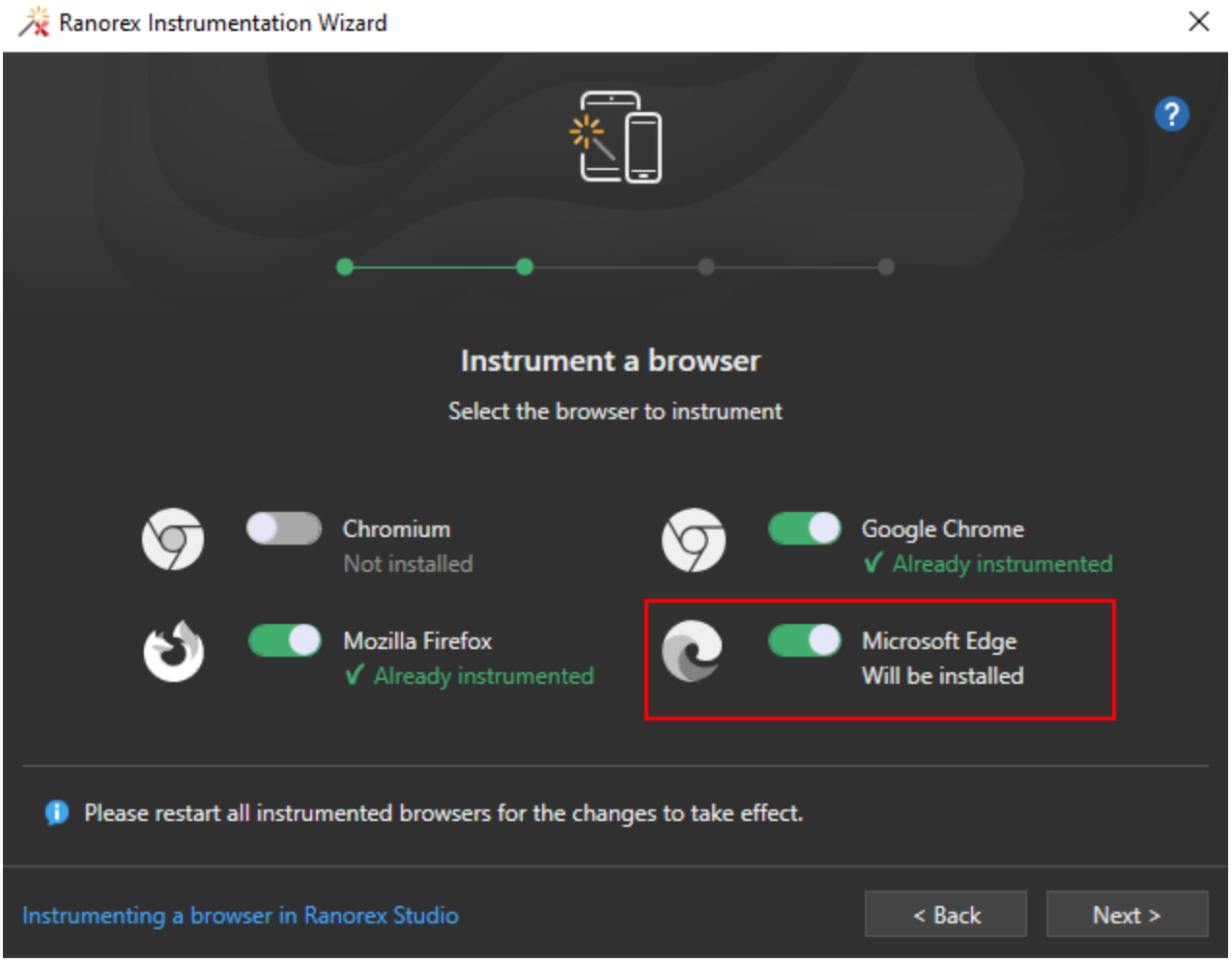
Install Ranorex Automation Add-on in Edge

To install the Ranorex Automation add-on, open the instrumentation wizard:

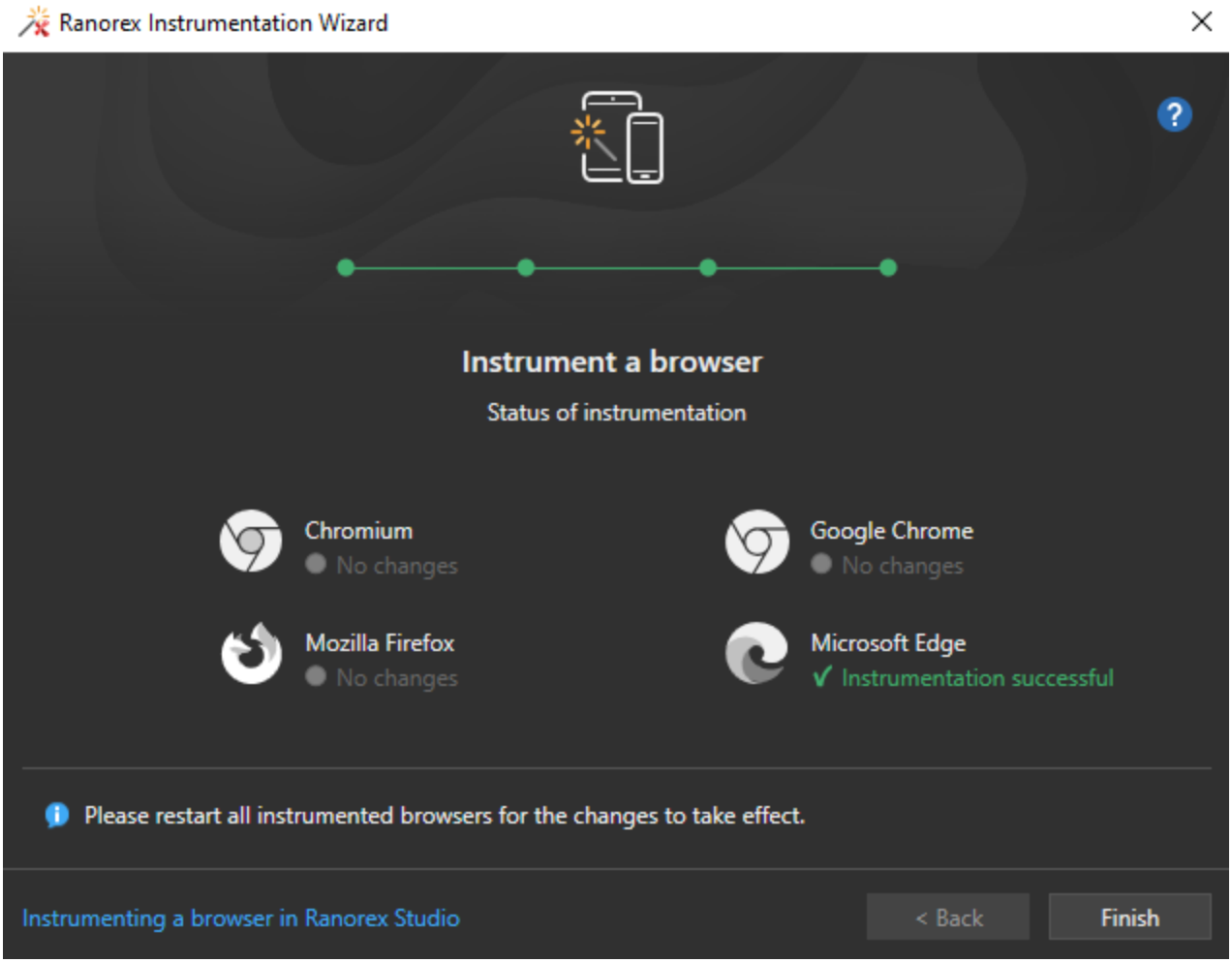
1. **Select Instrument a browser.**



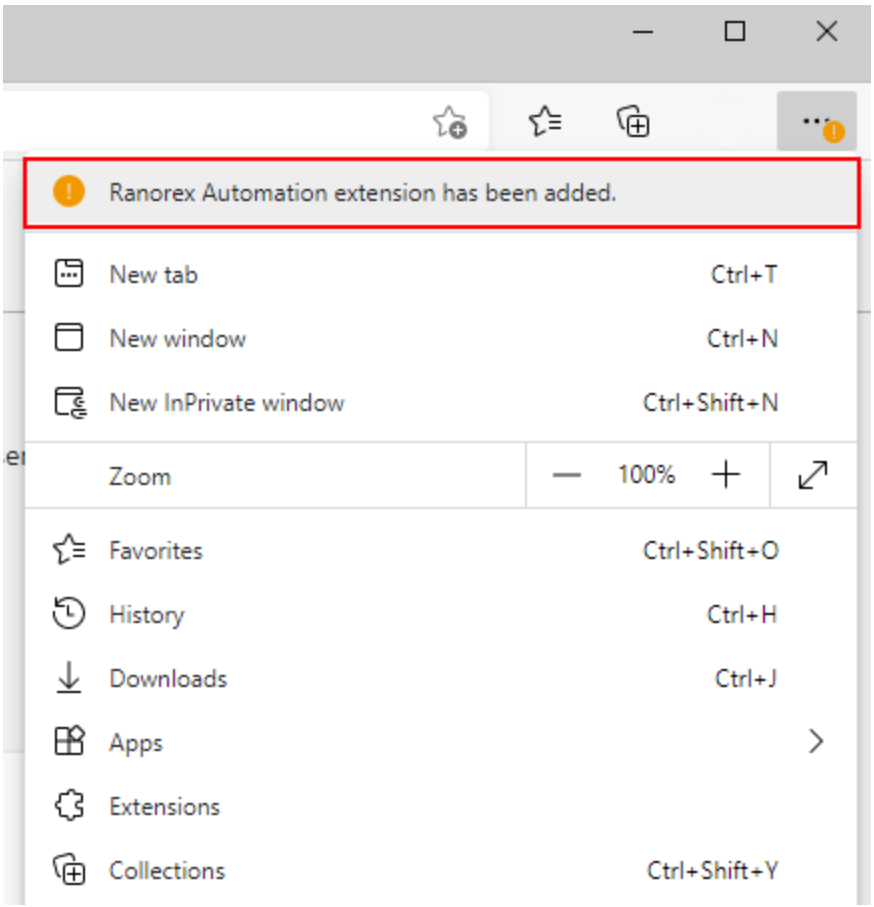
2. Enable the **Microsoft Edge** slider option.



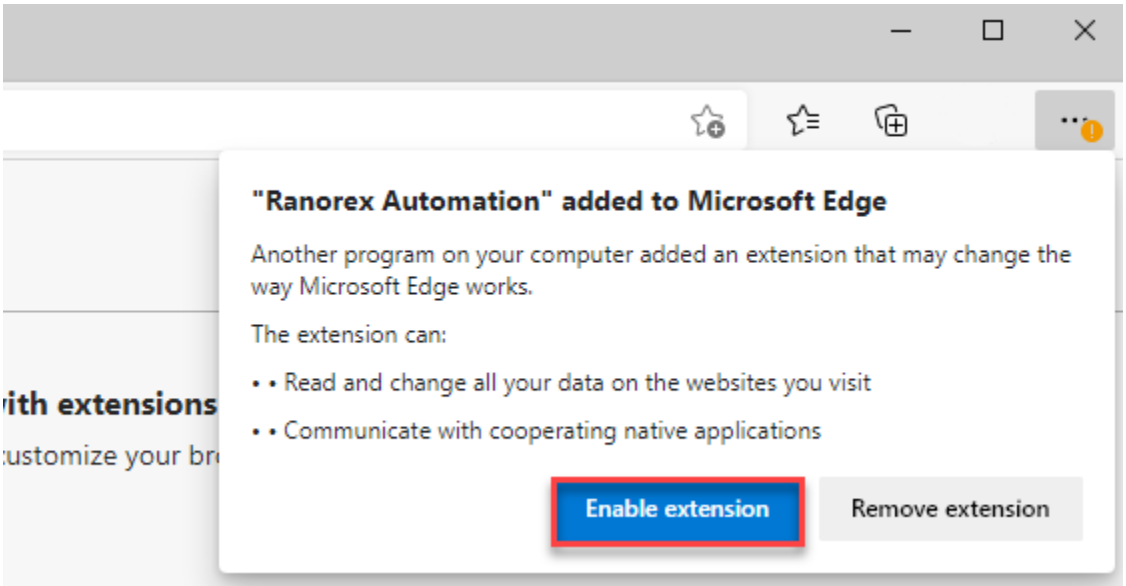
3 After the wizard displays the **Instrumentation successful** message, **click Finish**.



4 Restart your browser. **Click** the browser menu icon that shows a warning symbol and select the **Ranorex Automation extension has been added** message.



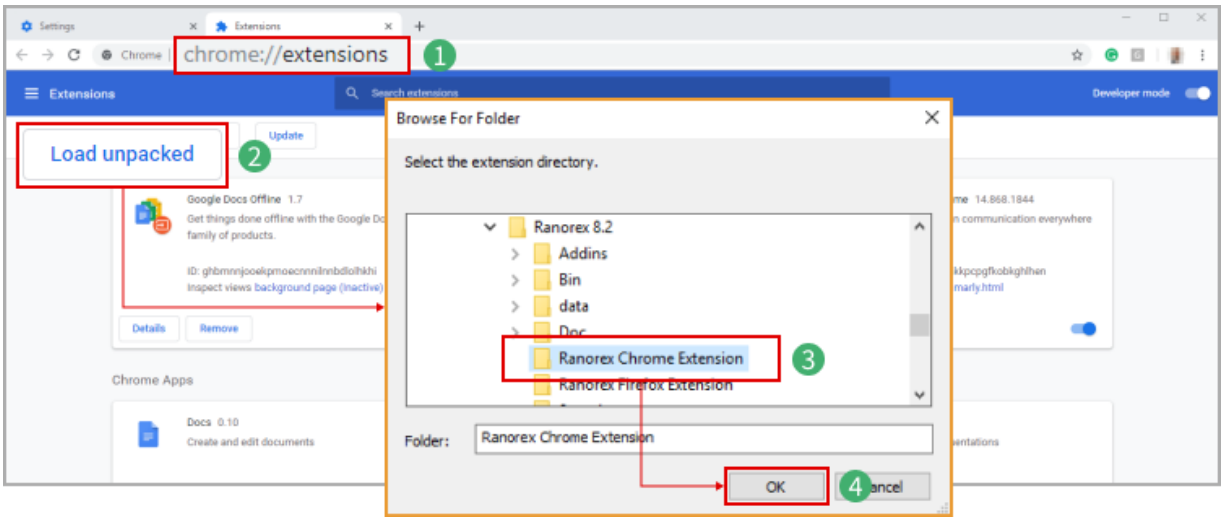
5 **Click Enable** to activate the add-on.



Manual installation

You can also install the add-on manually without the Instrumentation Wizard.

1. **Type chrome://extensions** (Google Chrome) or **edge://extensions** (MS Edge) into the address bar.
2. **Click Load unpacked**.
3. **Browse** to the folder **Ranorex Chrome Extension** in your Ranorex Studio installation folder.
4. **Click OK.**



Important notes for testing with Google Chrome/MS Edge

- If you want to test cross-domain iframe scripting, start the browser with the command line argument **–disable-web-security –user-data-dir=**""
- Make sure that the extension ID of the Ranorex Add-on is listed in the "allowed_origins" section of the **binRanorex.Plugin.ChromeMsgHost.manifest** file.
- If the **NoScript Extension** is installed, disable it.

Test offline/local HTML files ("Allow access to file URLs")

To test offline/local HTML files, the option Allow access to file URLs must be enabled for the Ranorex extension. Unfortunately, due to some technical restrictions, this doesn't work when instrumenting Google Chrome/MS Edge (Chromium) with the Instrumentation Wizard or the Open browser action.

Here are the workarounds for both browsers.

Google Chrome

1. **Make sure** the extension is installed (through any method except directly through the Google Chrome store.)
2. **Go** to the extension's installation folder:

%LOCALAPPDATA%\Google\Chrome\User Data\Default\Extensions\egdlgaljianpgdlmfijpphbadibfncdm

3. **Copy** the folder named **1.2.9_0** to another location.
4. **Open** this copied folder and **find** the file **manifest.json**.
5. **Open** this file in a text editor and under **"permissions"**, **add "storage"**, so the line looks like this:

"permissions": ["background", "tabs", "nativeMessaging", "http://*/*", "https://*/*", "storage"],

6. **Save** and **close** the file.
7. **Uninstall** the Ranorex extension with the Instrumentation Wizard.
8. **Open** Chrome, **go** to Extensions, and **ensure** Developer Mode is active.
9. **Click Load unpacked extension** and **point** to the copied 1.2.9_0 folder with the edited manifest file.
10. **Install** the Chrome extension again through the Instrumentation Wizard to resolve any errors displayed in Chrome.

MS Edge

If the extension isn't installed, start with step 2.

1. **Uninstall** the extension with the Instrumentation Wizard.
2. In Edge, **go** to the Google Chrome store.
3. **Find** the Ranorex extension and **install** it.
4. Once installed and enabled, **make sure Allow access to file URLs** is enabled in the extension's settings.

Android apps

In this chapter, you'll learn how to instrument an Android app.

Attention

For Android developers: As part of the instrumentation process, the app will be compiled with the Ranorex automation library. This library adds additional functions and permissions to your APK. This is why we recommend you **do NOT publish** instrumented apps to Google Play.

Prepare the app and the mobile device

Before you can instrument and deploy an app for automation with Ranorex, complete the following preparations:

Prepare the app

Make sure you can access the APK from the computer on which Ranorex Studio is installed. We recommend that you store a copy of the APK in a folder on that computer.

Prepare the mobile device

- Apply the device settings.
- Connect your device to the computer on which Ranorex Studio is installed.
- Start the Ranorex Service App on the device.

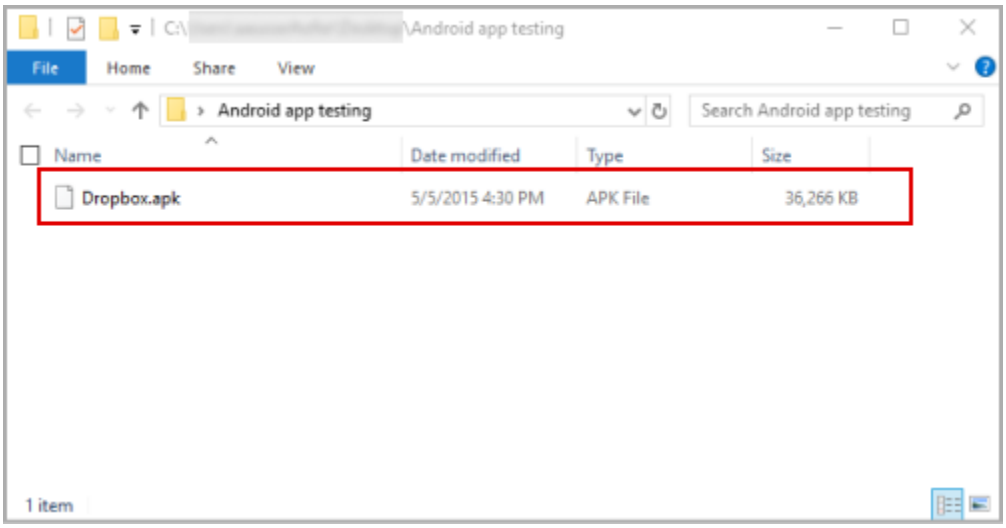
Note

You can instrument and then immediately deploy the app to your mobile device with the Instrumentation Wizard. If you want to do so, you must connect the mobile device to your computer via USB (recommended).

Instrumentation example

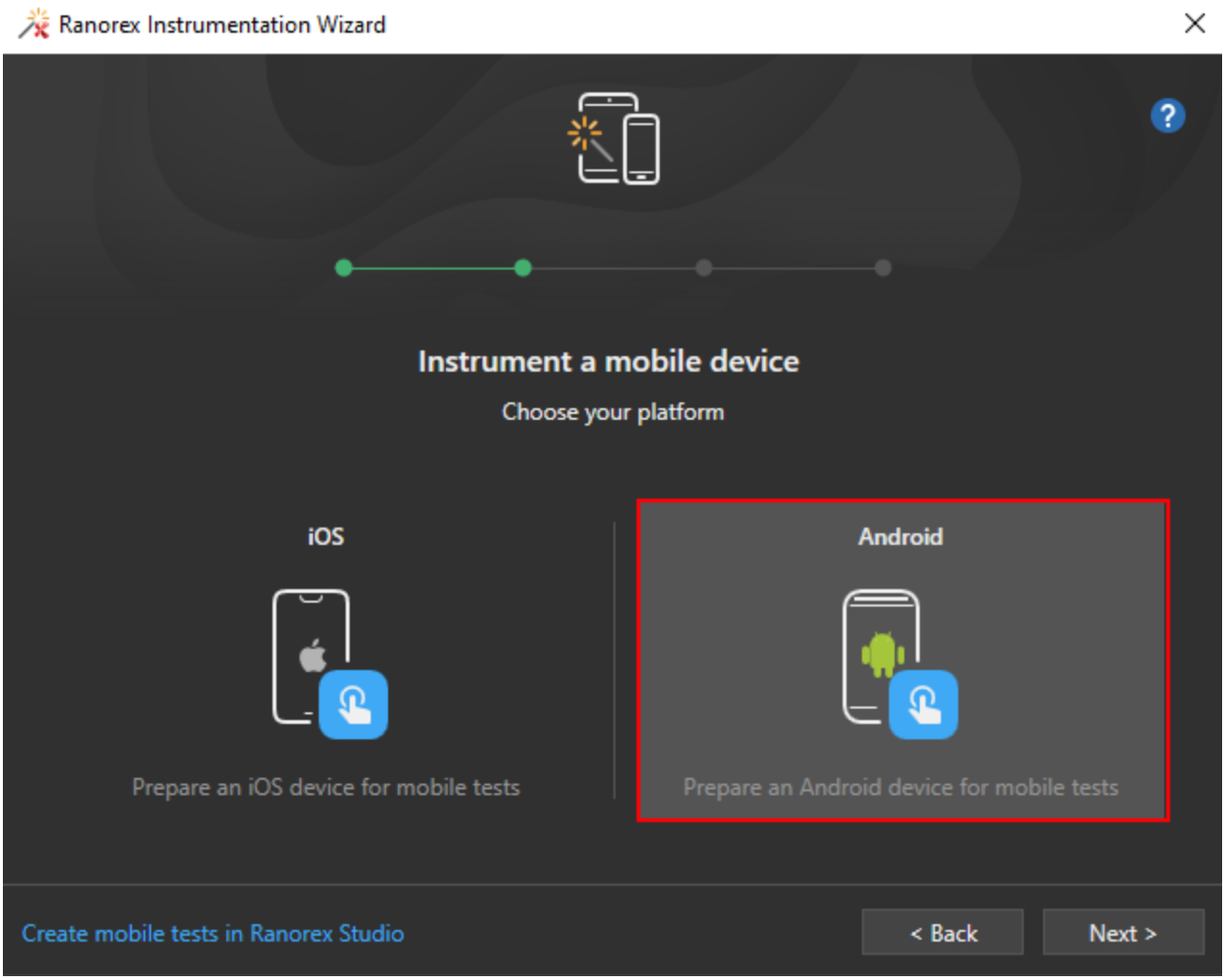
For our example, we'll instrument the Dropbox app. The app is a product of Dropbox, Inc, whose terms and conditions for downloading and using the app apply.

In preparation for instrumentation, we've stored the app in the folder **Android app testing** on the computer Ranorex Studio is installed on.



Select technology

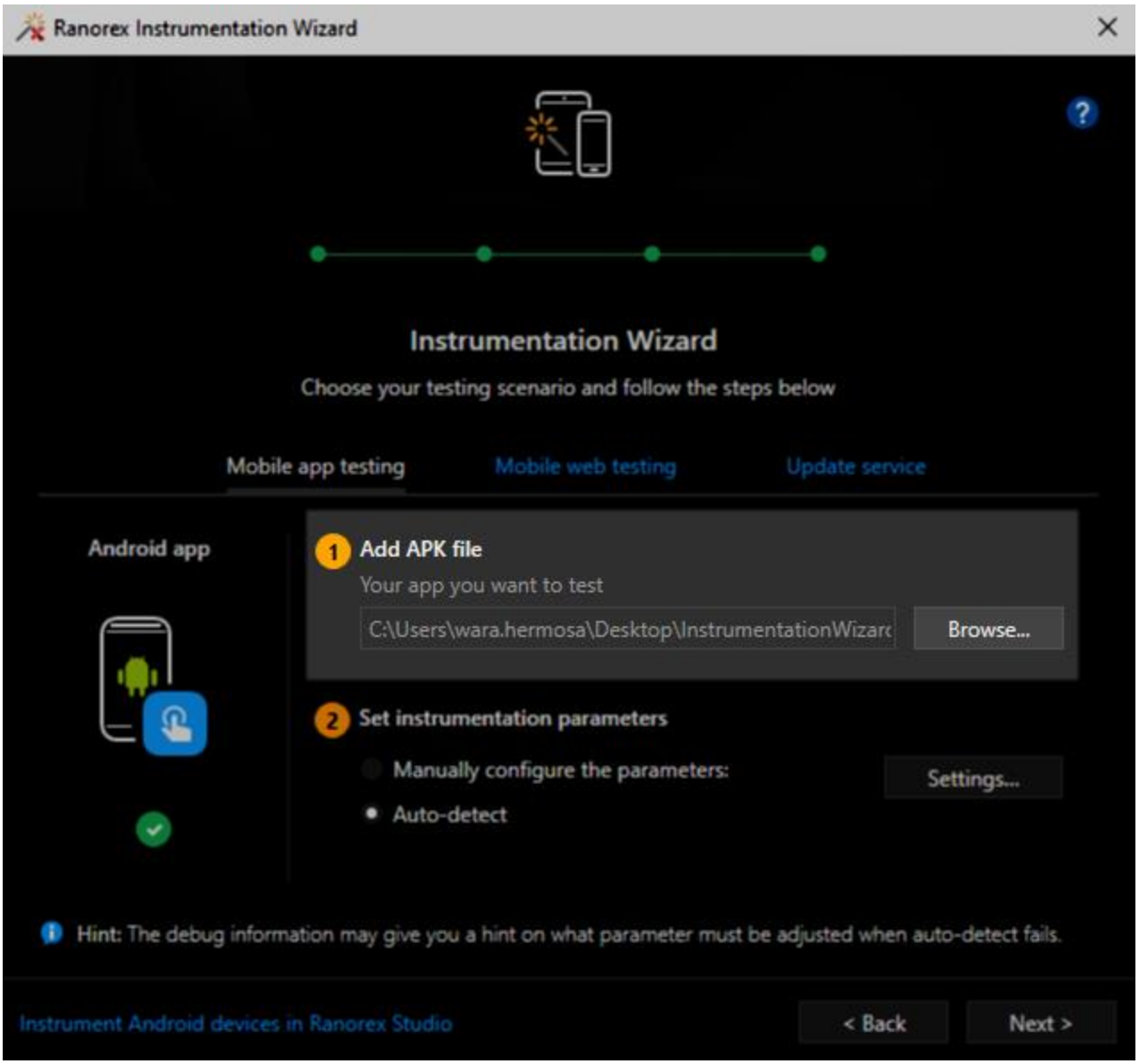
1. **Start** the Instrumentation Wizard and confirm the Windows security dialog.
2. **Click Android** and **Click Next**.



Specify APK and settings

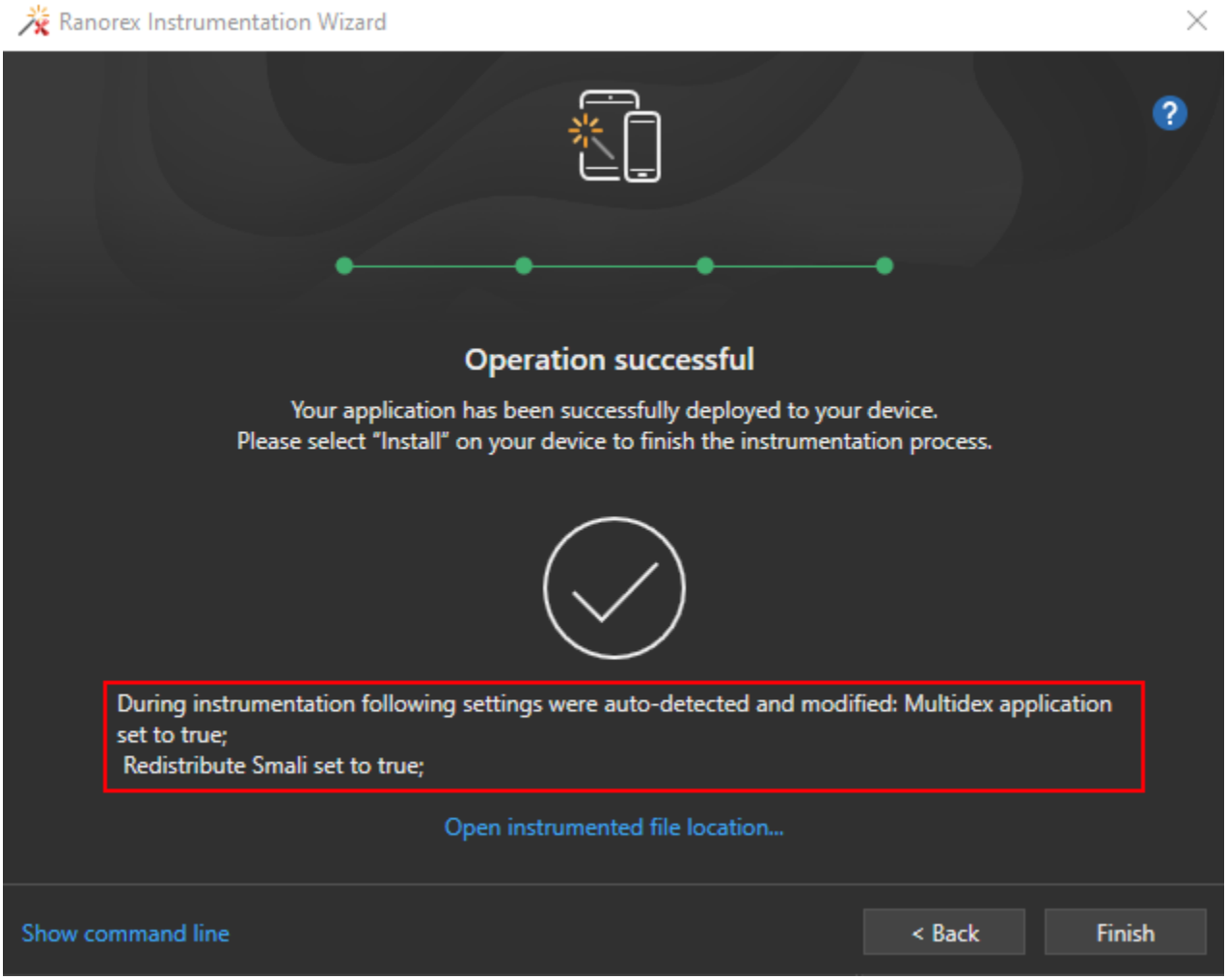
Add APK file

1. **Click Browse...** to select the APK.
2. The APK is loaded in the Instrumentation Wizard.



Set instrumentation parameters

3. **Select Manually Configure the parameters** and **click Settings...** to apply your selected advanced settings. These are explained at the end of this page under **Advanced Android instrumentation settings**.
4. Auto-detect is selected by default. This option automatically adjusts instrumentation settings and re-runs instrumentation process a couple of times.
5. **Click Next** to start the Instrumentation process.
6. When the instrumentation process completes the **Operation successful** message displays, note that this screen details settings changes that were auto-detected during the instrumentation wizard. **Click Finish** to close the wizard.



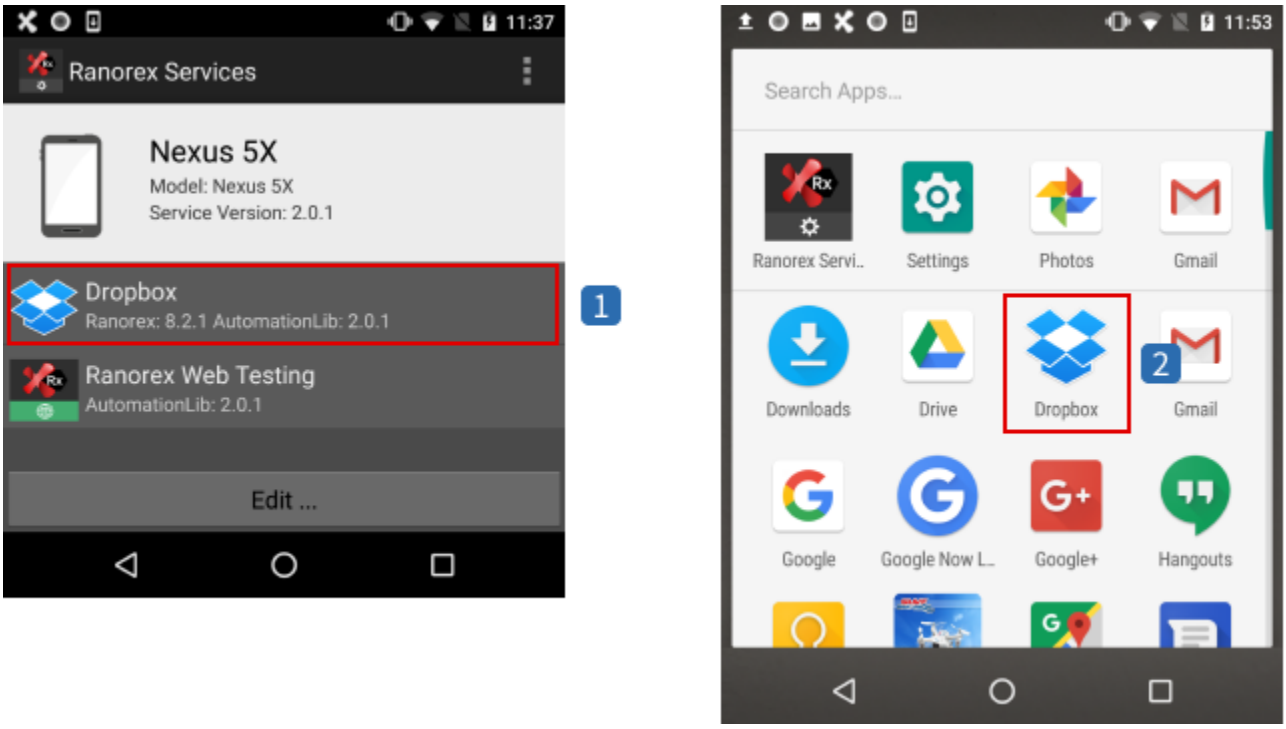
Note

If the process fails, please first check you've applied the ⇢ device settings correctly and ⇢ connected the device properly. A common problem is also that another program is consuming too m any resources, slowing down instrumentation to the point of a timeout error. Either close all other programs or increase the Instrumentation timeout in the settings.

Web sites can have highly complex structures. This is why it may sometimes be a good idea or even necessary to extend the above process by manually tweaking the recorded test with additional code. We'll cover this in the later subchapters.

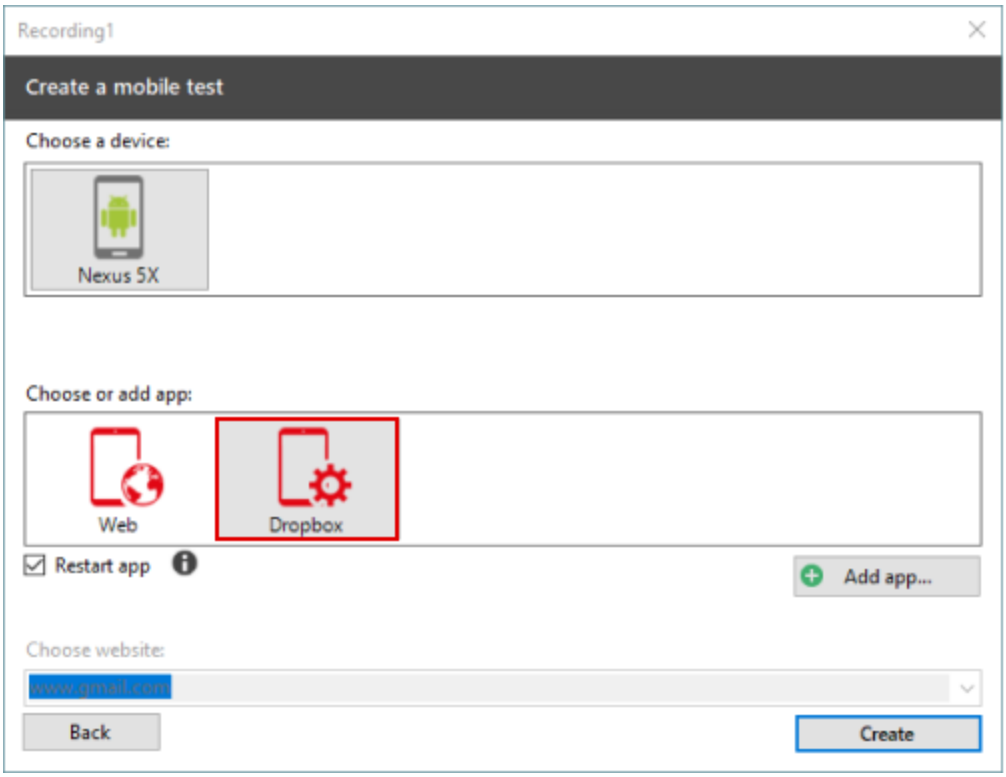
Result

If the app has been instrumented and deployed correctly, you will see the following on your device:



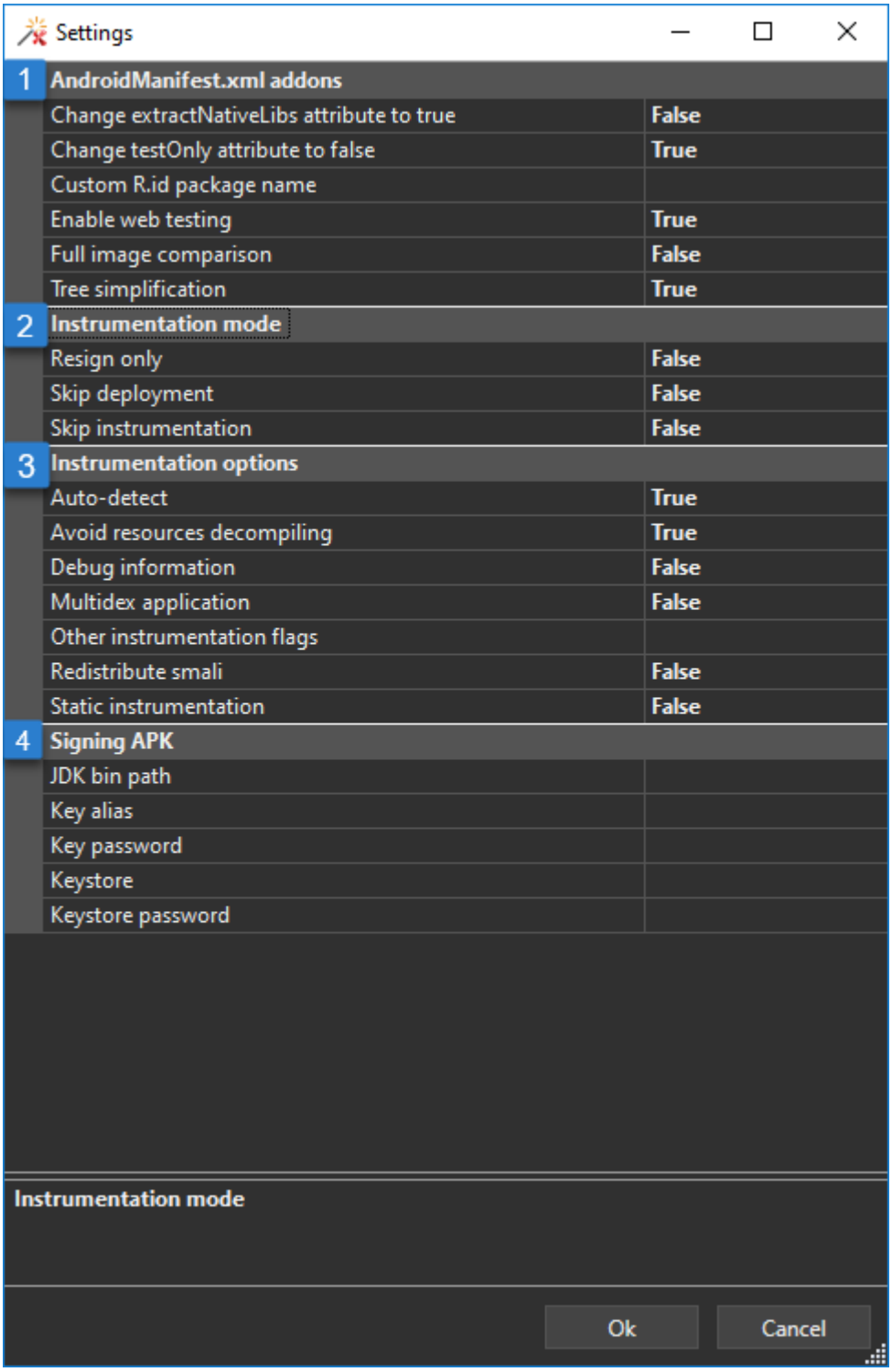
1. The instrumented app appears in the Ranorex Service App.
2. The instrumented app appears on your device's homescreen.

In Ranorex Studio, you can see the app has been instrumented correctly if it appears for selection when creating a mobile test.



Advanced Android instrumentation settings

These settings are available as part of the instrumentation dialog, as explained on this page under **Specify APK and settings**. They contain options for configuring instrumentation for special types of Android apps and are also useful for troubleshooting.



1 **AndroidManifest.xml addon settings**

Change extractNativeLibs attribute to true: When set to true, Ranorex changes the extractNativeLibs attribute to true if it is defined in AndroidManifest.xml.

Change testOnly attribute to false: When set to true, Ranorex sets the testOnly attribute to false if it is defined in AndroidManifest.xml

Custom R.id package name: Allows you to specify a custom class name for locating resource IDs (e.g. com.ranorex.demo.R). By default, Ranorex Studio will search for IDs in <manifestpackage>.R.

Enable web testing: When set to true (default), shows DOM content of WebViews in the UI element tree in Ranorex Spy. Has a significant performance impact.

Full image comparison: When set to true, uses a more robust image comparison mechanism to determine resource IDs for images. Decreases startup performance.

Tree simplification: When set to false, no postprocessing of the UI element tree takes place. This means a more complex element tree is displayed in Ranorex Spy, which may be useful for automating third-party Android controls.

2 Instrumentation mode

Resign only: When set to True, Ranorex will perform only changing the signature of the chosen APK file. By default, should be set to False.

Skip deployment: When set to True, Ranorex will not perform deployment process. It means that Ranorex will not attempt to upload the chosen IPA file to chosen endpoint. It may be useful, if one wants to deploy the IPA using another device. By default, this option is set to False.

Skip instrumentation: When set to True, Ranorex will not perform default instrumentation process for the chosen APK file, including signing of the APK. This option may be useful for deploying any APK. By default, this option is set to False.

3 Instrumentation options

Auto-detect: When set to True (by default), automatically adjusts instrumentation settings and re-run instrumentation process up to four times.

Avoid resources decompiling: If set to True (by default), resources are not decompiled and recompiled during instrumentation. This means the following:

- Resources will not be decompiled and the arsc is kept intact without any decoding.

- The AndroidManifest.xml is decoded by Ranorex Studio and not by the default Android tools embedded in the APK.

When enabled, **speeds up the instrumentation process** significantly. Only set to *False* if you need to make changes to the resources in the instrumented APK.

Debug information: When set to True, adds additional information to the instrumentation log file. Useful for troubleshooting instrumentation issues.

Multidex application: If your mobile app and the libraries it references exceed 65,536 methods and this option is set to False, you will encounter a build error during instrumentation. This is because your app has reached the limit of the Android build architecture. To resolve this, set this option to True and retry instrumenting.

Other instrumentation flags: Here you can add individual flags to apply to the instrumentation process. These are simply the settings you find here, except in parameter form, e.g. -**nores** for the option Avoid resources decompiling.

Redistribute smali: If you receive an error message that indicates issues with the number of methods in the decompiled smali code, set this option to *True*. Ranorex Studio will then distribute the code across multiple files.

Static instrumentation: If you receive an error message of the type "Register count exceeds…" or "Method count exceeds 65k", set this option to True. Ranorex Studio will then use a special instrumentation mechanism that highly reduces the amount of code that needs to be added to user activities.

Try the option "Redistribute smali" before activating this option!

4 Signing APK

If you want to sign your APK, you need to specify all of these options.

iOS apps

In this chapter, you'll learn how to instrument an iOS app.

> **Attention**
>
> For iOS developers: As part of the instrumentation process, the app will be compiled with the Ranorex automation library. This library adds additional functions and permissions to your IPA.
>
> **Do NOT publish** instrumented apps to the App Store.
>
> Instrumented apps **do not work** with TestFlight.

Prepare the app and the mobile device

Before you can instrument and deploy an app for automation with Ranorex Studio, complete the following preparations:

Prepare the app

To instrument an iOS app, you will need the IPA file, the associated P12 certificate and password, and the proper mobile provisioning profile.

Make sure you can access the IPA, P12, and mobile provisioning profile files from the computer on which Ranorex Studio is installed. We recommend you store a copy of them in a folder on that computer.

Prepare the mobile device

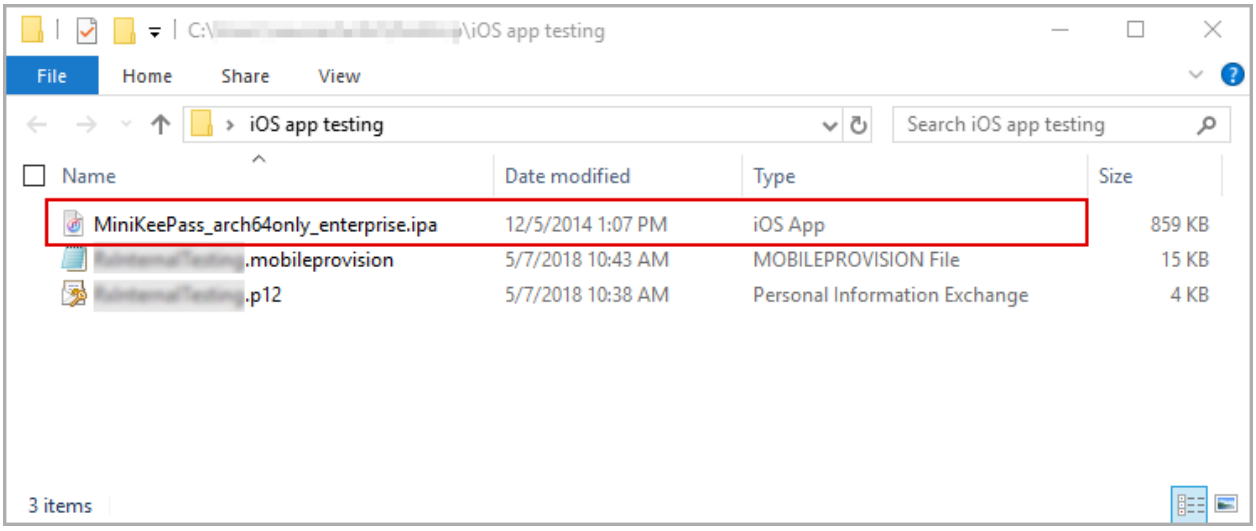
- Apply the device settings.
- Connect your device to the computer on which Ranorex Studio is installed.
- Start the Ranorex Service App on the device.

> **Note**
>
> You can instrument and then immediately deploy the app to your mobile device with the Instrumentation Wizard. If you want to do so, you must connect the mobile device to your computer via USB (recommended).

Instrumentation example

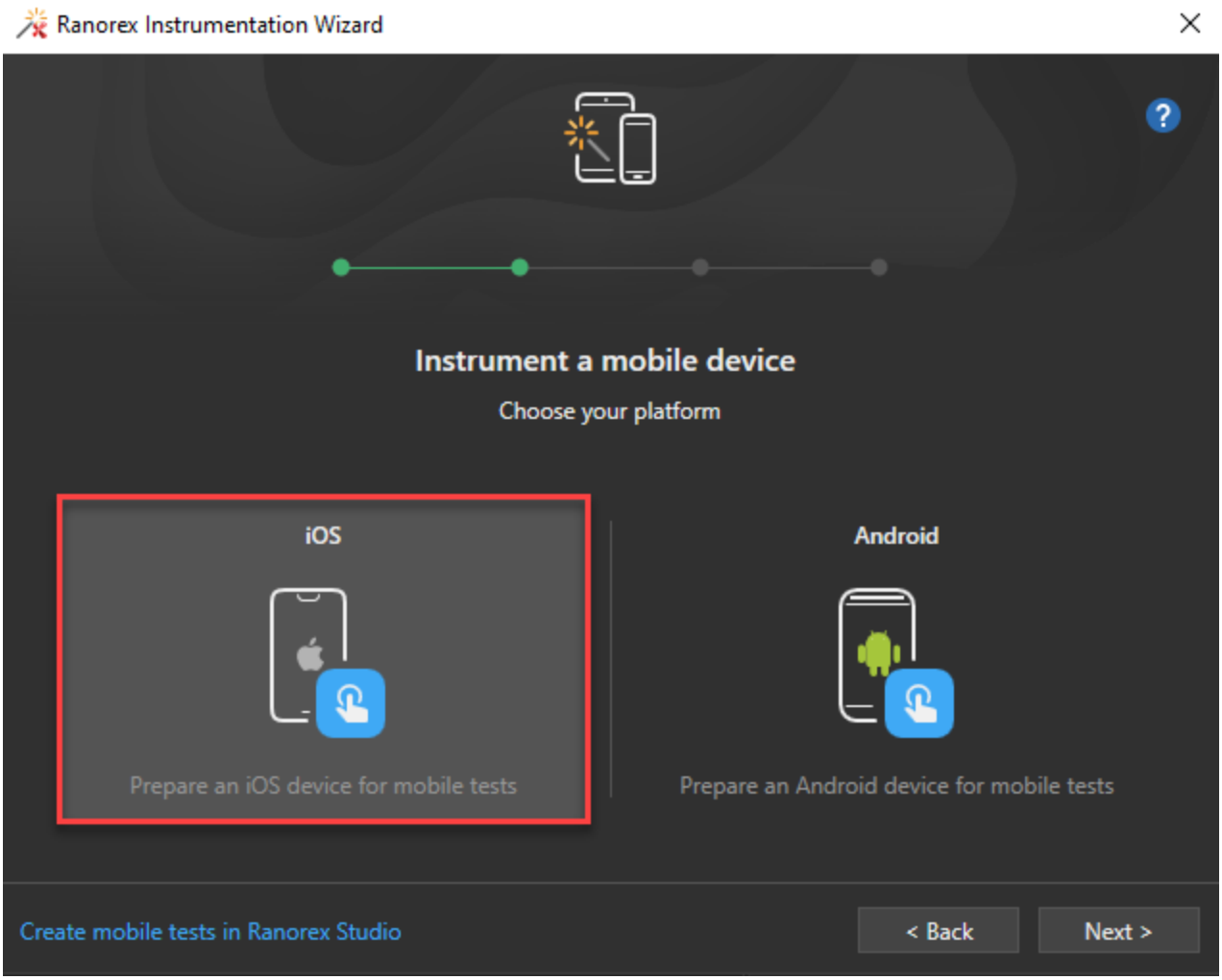
For our example, we'll instrument the KeyPass app. The KeyPass app is available under the GNU General Publice License. For information regarding the copyright holder and the license agreement, go to https://keepass.info/help/v1/license.html.

In preparation for the instrumentation process, we've stored the IPA, the P12 certificate and the mobile provisioning profile in the folder iOS app testing on the computer Ranorex Studio is installed on.



Select technology

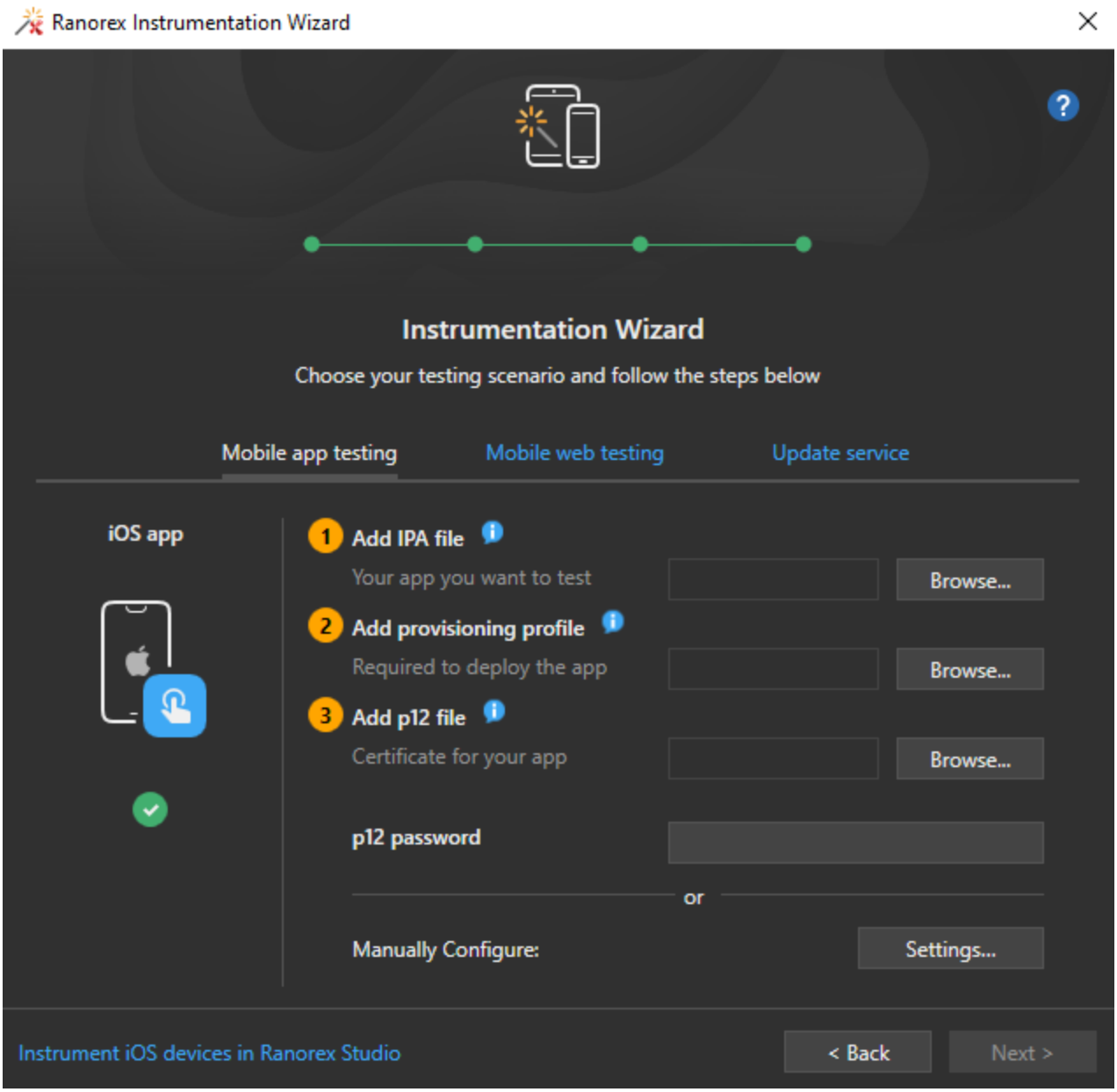
1. **Start** the Instrumentation Wizard and confirm the Windows security dialog.
2. **Select iOS** and **click Next**.



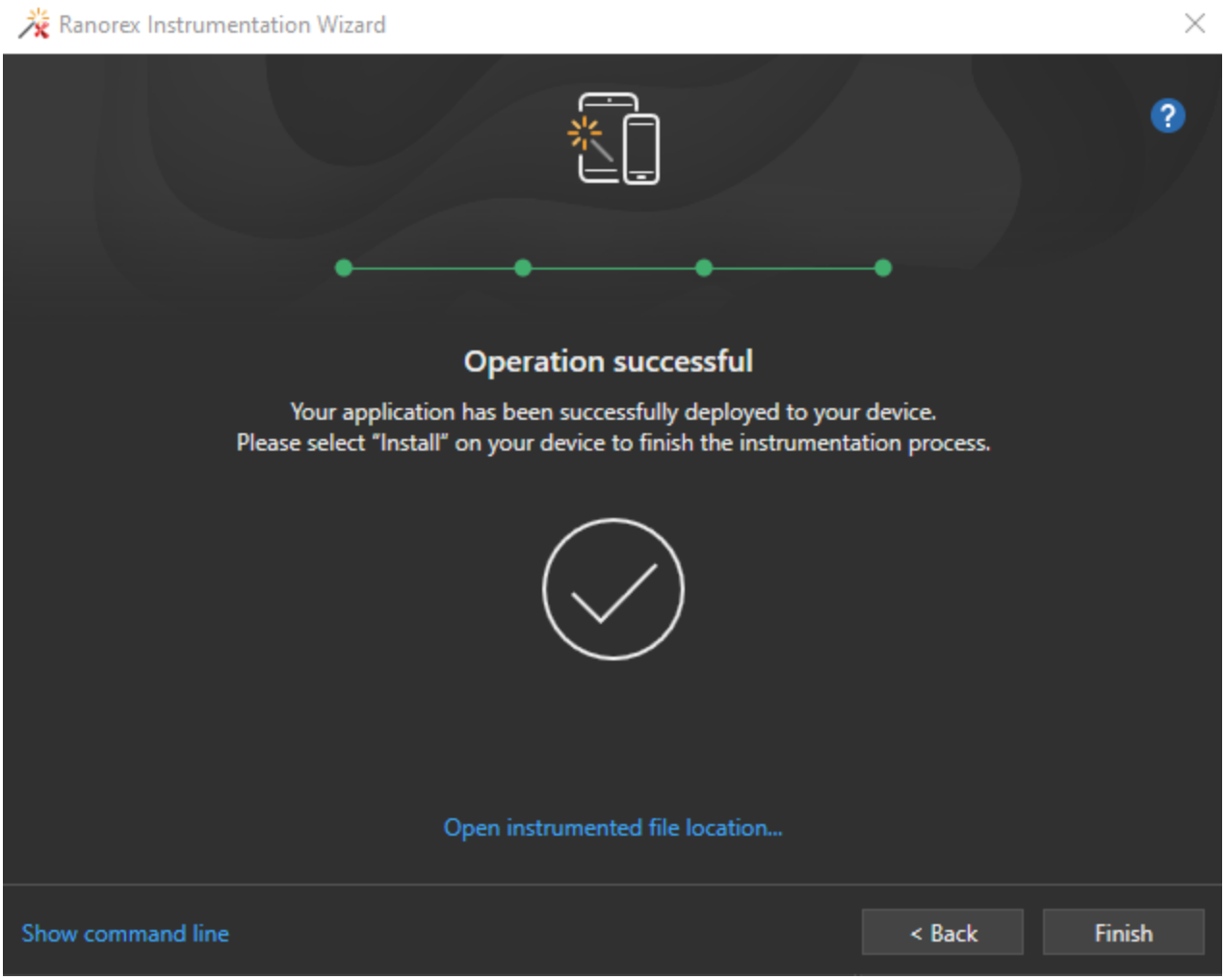
Specify IPA and settings

1. **Do not change** the default setting **Instrument & deploy custom IPA, this** instruments and deploys the app to the mobile device. **Click Next**.
2. **Click Browse...** and browse the IPA file.
3. Add the **provisioning profile** and the **p12 file** by clicking **Browse...** The **provisioning profile** and the **p12 file** can be obtained from the Apple Developer portal.
4. **Enter** the **password** for the **p12 file** and **click Next**.

Also, you can manually configure by clicking **Settings...**

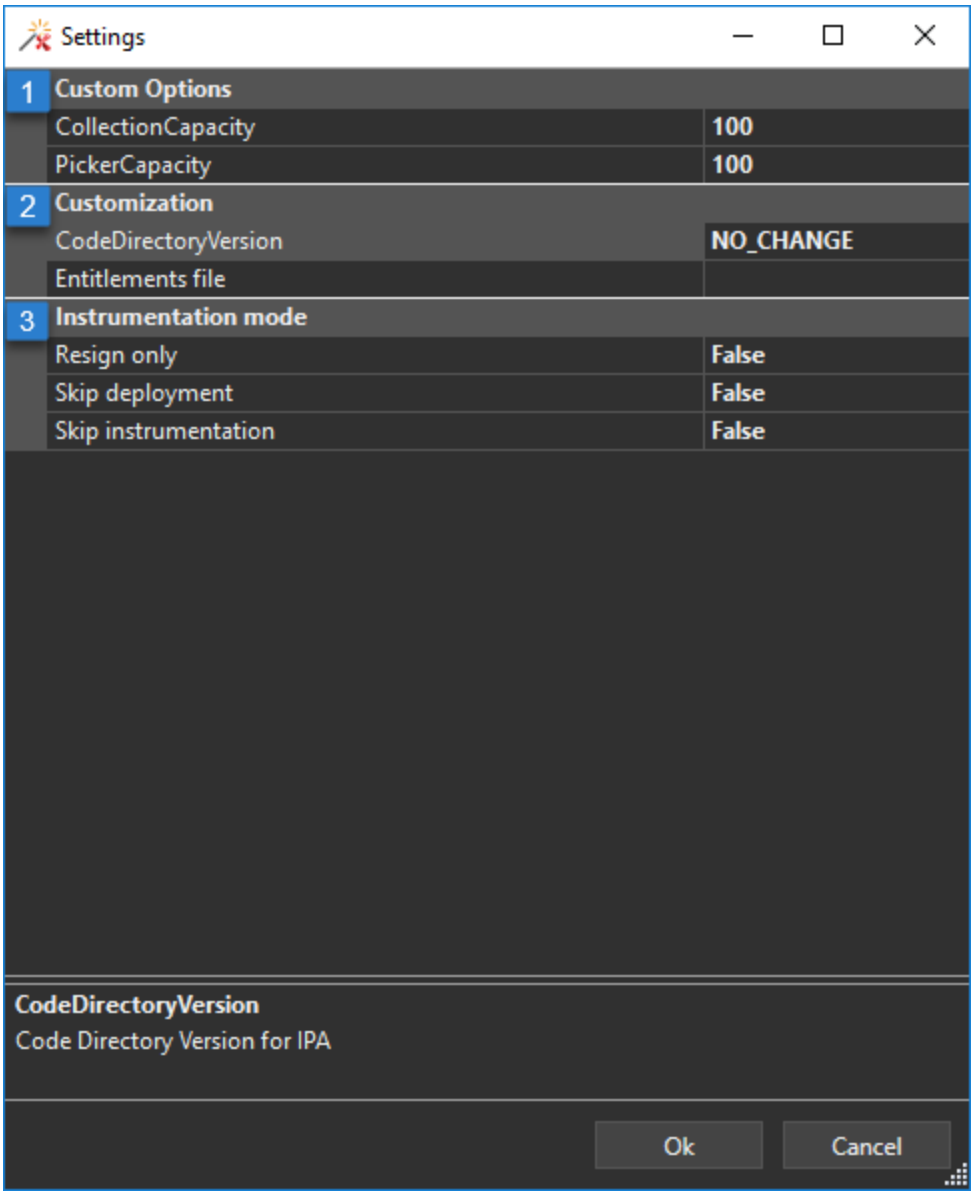


5 The app has been instrumented successfully.



Configure Manually

1. **Click Settings**.
2. In the **Settings** window, configure the options for the instrumentation according to your needs.


Settings
1 Custom Options
CollectionCapacity 100
PickerCapacity 100
2 Customization
CodeDirectoryVersion NO_CHANGE
Entitlements file
3 Instrumentation mode
Resign only False
Skip deployment False
Skip instrumentation False
CodeDirectoryVersion
Code Directory Version for IPA
Ok
Cancel

1 Custom Options

CollectionCapacity: Sets the maximum number of collection list items.

PickerCapacity: Sets the maximum number of picker list items.

2 Customization

CodeDirectoryVersion: Sets code directory version for IPA. Use the drop-down list to select an option.

Entitlements file: The path to optional entitlement customized file. Look for codesign entitlements file usage for more details.

3 Instrumentation mode

Resign only: When set to *True*, Ranorex will perform only changing the signature of the chosen API. By default, should be set to *False*.

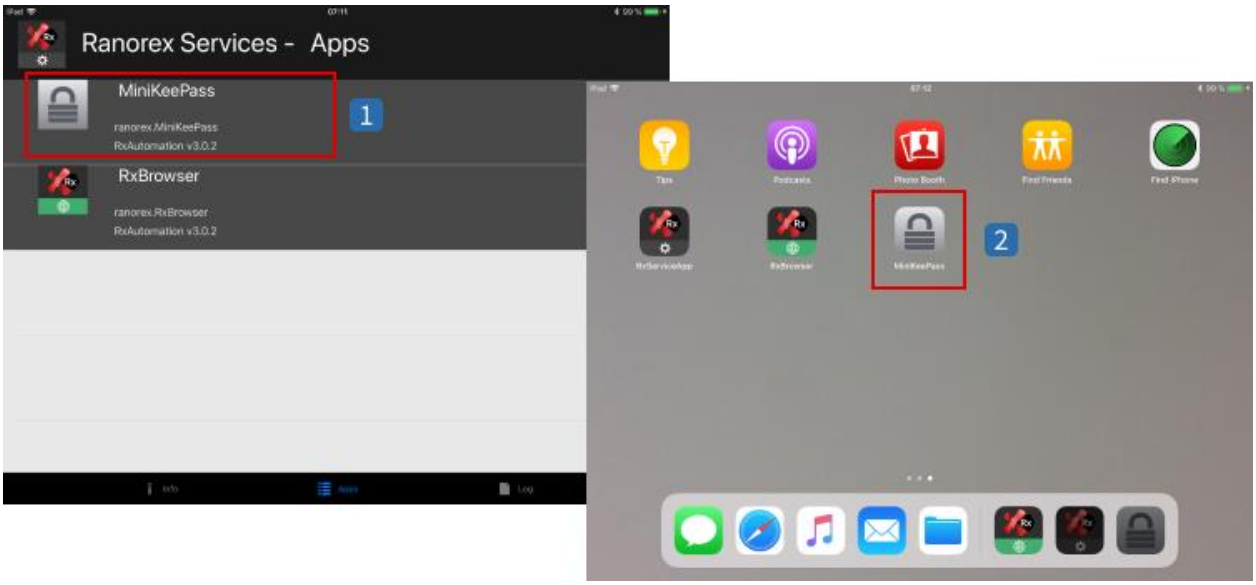
Skip deployment: When set to *True*, Ranorex will not perform deployment process. It means that Ranorex will not attempt to upload the chosen IPA file to chosen endpoint. It may be useful, if one wants to deploy the IPA using other device. By default, this option should be set to False

Skip instrumentation: When set to *True*, Ranorex will not perform default instrumentation process for the chosen API file, including signing of the API. This option may be useful for deploying any IPA. By default, the value for this option is *False*.

> **Note**
>
> If the process fails, please first check you've applied the device settings correctly and connected the device properly. A common problem is also that another program is consuming too many resources, slowing down instrumentation to the point of a timeout error. Either close all other programs or increase the Instrumentation timeout in the settings.

Result

If the app has been instrumented and deployed correctly, you will see the following on your device:



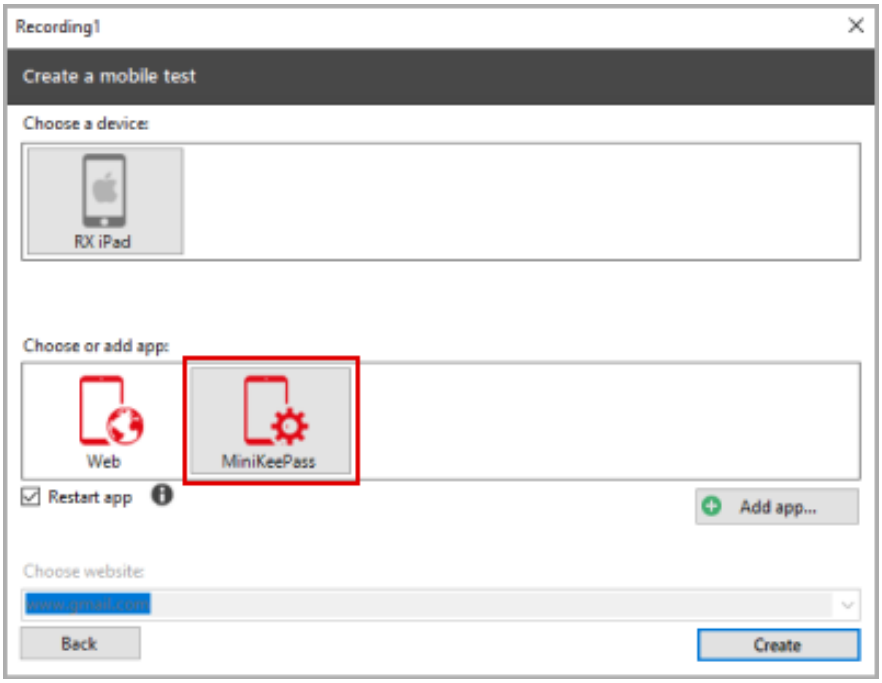
1 The instrumented app appears in the Ranorex Service App.

2 The instrumented app appears on your device's home screen.

Attention

After instrumenting and deploying the iOS app for the first time, **start the app manually once!** Otherwise it will not be visible within the Ranorex Service App.

In Ranorex Studio, you can see the app has been instrumented correctly if it appears for selection when creating a mobile test.


Recording1
Create a mobile test
Choose a device:
RX iPad
Choose or add apps:
Web
MiniKeePass
Restart app
Add app...
Choose website:
Back
Create

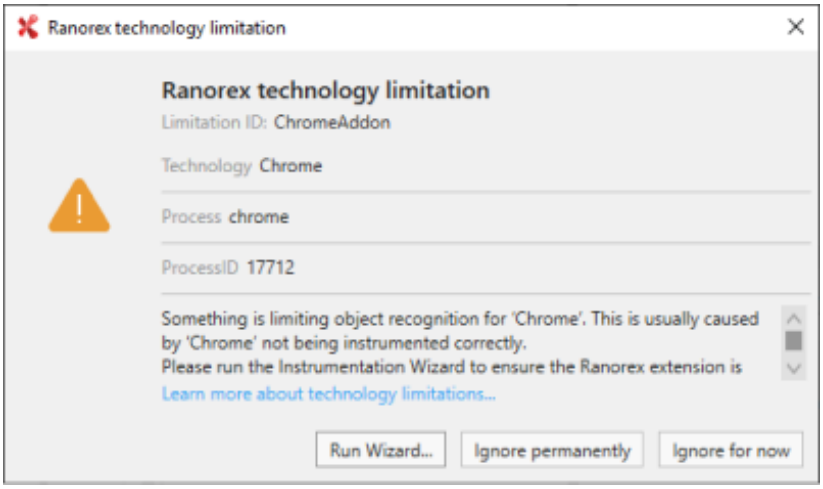
Technology instrumentation

Ranorex Studio supports test automation for many different UI technologies. Some of these technologies must be instrumented for best automation results. Instrumenting can mean installing an add-on, configuring the AUT in a certain way, or adding code so Ranorex Studio can identify UI elements for that technology. The capability to identify UI elements is also called **object recognition.**

Ranorex Studio can instrument many technologies automatically, but in some cases, this won't work. Some technologies simply require manual instrumentation, either with the ⇢ Ranorex Instrumentation Wizard or by hand.

This chapter covers the cases where automatic instrumentation or the Ranorex Instrumentation Wizard don't work. It is structured according to the different technologies, with a separate page for troubleshooting object recognition issues for technologies not covered on the other pages.

In many cases, Ranorex Studio will warn you about object recognition issues with a technology instrumentation message, which will look like this:



Technology limitation warning for Chrome – in this case, the Instrumentation Wizard can be used.

Flash/Flex/Air

The **Flash/Flex/Air plugin** in Ranorex Studio allows it to interface with the controls in such applications. This means you can record Flash/Flex/Air UI elements and work with them in Ranorex Spy.

However, these technologies require some manual instrumentation. Flash and Flex can be instrumented with the Ranorex Instrumentation Wizard in most use cases. Air needs to be instrumented entirely by hand.

This page explains the various instrumentation methods for all the technologies.

Prerequisites

The automation of Flash/Flex applications requires inclusion of the Ranorex Flex Automation Lib.

- **⇢ Using the Ranorex PreLoader**
- **OR including the Ranorex Lib** into Adobe Flex or Adobe Flash:
  ⇢ How to load the Ranorex Lib into a Flex application
  ⇢ How to load the Ranorex Lib into a Flash application

Methods for loading the Automation Lib

Depending on your flash/flex application, you can choose to use one of the following flexible instrumentation methods:

⇢ PreLoader (recommended)

The Ranorex PreLoader enables automation of your flash/flex application without modifying the application itself.

- ⇢ Automatic installation using the Instrumentation Wizard
- No modifications to your application needed
- Adobe Flash Debug Player has to be installed on the runtime machine

⇢ Automation Lib

The AutomationLib swc file has to be included into your flash/flex application (by adding a compiler argument).

- AutomationLib will be loaded in background (will not affect the functionality of your flash/flex application)

- No modifications to your website needed

⇢ Module

The Module swf file has to be loaded by adding a code snippet to your ⇢ ACTIONS (for flash) or to your ⇢ 'applicationCompleteHandler' function (for flex).

- Module will be loaded in background (will not affect the functionality of your flash/flex application)
- No modifications to your website needed
- Module swf has to be copied to the web server

The following table shows you the different instrumentation options available for the supported versions of flash/flex:

	Flash			Flex		AIR*		
	CS3	CS4	CS5	3.x	4.x	2.x	3.x	4.x
Preloader	✓	✓	✓	✓	✓			
Automation Lib		✓	✓	✓	✓	✓	✓	✓
Module	✓	✓	✓	✓	✓	✓	✓	✓

* Support for Adobe AIR2 release in combination with Flex 3.5 and Flex 4 release

✓ =supported/ ✓ = recommended

How to read/set attributes and styles

You can read/set common attributes (e.g. the checkstate of a checkbox) directly by using the adapter as follows:

Flash/Flex

```
// Load the slider as a Flex element to access flex attributes

FlexElement sliderFlex = repo.FlexTestPageDom.ContainerHSlider_Control.SliderHSlider.As<flexelement>();

sliderFlex["value"] = "100";
```

```
FlexElement containerFlex = repo.FlexTestPageDom.Self.FindSingle(".//container[@caption='
Tree Control']");

containerFlex.SetStyle("borderColor","#ee0000");

</flexelement>
```

To access custom attributes and styles, you first have to load your Flex item as a FlexElement in order to read your attributes. Here's an example:

Flash/Flex

```
// Load the slider as a Flex element to access flex attributes

FlexElement sliderFlex = repo.FlexTestPageDom.ContainerHSlider_Control.SliderHSlider.As<f
lexelement>();

sliderFlex["value"] = "100";

 FlexElement containerFlex = repo.FlexTestPageDom.Self.FindSingle(".//container[@caption=
'Tree Control']");

containerFlex.SetStyle("borderColor","#ee0000");

</flexelement>
```

Using the Ranorex PreLoader

The Instrumentation Wizard automatically activates Flash/Flex automation for your machine as ⇢ described here.

If the automatic instrumentation did not work, you can follow these steps to achieve what the Instrumentation Wizard would have done:

1. **Install the Adobe Debug Flash Player** on the machine on which you would like to record and execute Ranorex test scripts and for all browsers you would like to test with. The respective installers are available on Adobe's download page.
2. **Open your user profile directory %UserProfile%**, e.g. by opening Windows Explorer and copying the string *%UserProfile%* into the address bar.

3 **Create a new file called 'mm.cfg' in your user profile directory** and insert the following line of code:

```
C:\Program Files\Ranorex X.X\Bin\RanorexAutomation.swf
```

where 'C:\Program Files\Ranorex X.X\Bin\RanorexAutomation.swf' needs to be replaced by the preloader location of your Ranorex installation. If the 'mm.cfg' file already exists, check if the location is correctly set to your current preloader location (Ranorex X.X might change after upgrading to a new version!). In addition to this configuration entry you can enable the logging mechanisms of the Adobe Debug Flash Player, which might be helpful if you have any problems with this kind of instrumentation. In such a case you can then simply provide the logging information to our support team. To enable logging you have to add (or modify) the following line of code to your 'mm.cfg' file:

```
TraceOutputFileEnable=1
```

This additional configuration entry forces the Debug Player to create a log file at following location:

```
C:\Users\[username]\AppData\Roaming\Macromedia Flash Player\Logs
```

where [username] needs to be replaced by the username that is logged on.

4 If you run your application from the local drive, add your output directory to the trusted locations of Flash Player as follows:

1. Open following link http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html#119065
2. Add your project output directory

Adobe Flex: Load the Ranorex Lib into your Flex application

1. Start Adobe Flash Builder and open your workspace
2. Right-click on your project and choose Properties
3. Under Flex Compiler add the **RanorexAutomation.swc** file (located in the Bin directory of your Ranorex installation) to the the compiler argument as follows:

```
-include-libraries "C:/Program Files/Ranorex X.X/Bin/RanorexAutomation.swc"
```

4. Append following code to your 'applicationCompleteHandler' function:

```
Flash/Flex
```

```
import Ranorex.RxAutomation;

RxAutomation.Init();
```

5. Save and compile your application
6. Run your application in a supported browser (Internet Explorer, Firefox, Chrome, Safari)
7. If you run your Flex application from the local drive, add your output directory to the trusted locations of Flash Player as follows:

1. Open following link http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html#119065
2. Add your project output directory

Adobe Flash: Load the Ranorex Lib into your Flash application

1. Start Adobe Flash CS4/CS5 and open your application
2. Open the 'Publish Settings' dialog (File->Publish Settings)

3. Include the Ranorex Flash Library in your Flash application under Flash->Script->Settings…->Library Path and choose 'Browse to SWC file'
4. Select 'RanorexAutomation.swc' file in the browse dialog (RanorexAutomation.swc is located in the Bin directory of your Ranorex installation)
   Insert following code to your ACTIONS (by pressing F9):

Flash/Flex
```
import Ranorex.RxAutomation;
RxAutomation.InitFromApp(stage);
```

OR add the code to your ActionScript source file as follows:

Flash/Flex
```
import mx.controls.*;
import flash.events.*;
import flash.display.*;
import flash.ui.Keyboard;
import flash.geom.Rectangle;
import fl.events.SliderEvent;
// Add Ranorex library
import Ranorex.RxAutomation;

public class Simple extends MovieClip
{
 public function Simple()
 {
 // Add to your constructor
 RxAutomation.InitFromApp(stage);
 }
}
```

5. Run your application in a supported browser (Internet Explorer, Firefox, Chrome, Safari)
6. If you run your Flash application from the local drive, add your output directory to the trusted locations of Flash Player as follows:

   1. Open following link
      http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html#119065
   2. Add your project output directory

How to load the Ranorex Module into your Flash application

1. Start Adobe Flash CS3/CS4/CS5 and open your application
2. Insert following code to your ACTIONS (by pressing F9):

Flash/Flex

```
import flash.net.URLRequest;
var rxloader : Loader = new Loader();
stage.addChild(rxloader);
rxloader.name = "__rxloader";
rxloader.width = 0; rxloader.height = 0;
rxloader.load(new URLRequest("RanorexAutomation.swf"));
```

3. Copy the **RanorexAutomation.swf** file, located in the Bin directory of your Ranorex installation, to your web server (where your .swf file is located)
4. Run your application in a supported browser (Internet Explorer, Firefox, Chrome, Safari)
5. If you run your Flash application from the local drive, add your output directory to the trusted locations of Flash Player as follows:

1. Open following link http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html#119065
2. Add your project output directory

How to load the Ranorex Module into your Flex/AIR application

1. Start Adobe Flash Builder and open your workspace
2. Append following code to your 'applicationCompleteHandler' function or initialization code:

For Flex 3:

Flash/Flex

```
var rxloader : Loader = new Loader();
Application.application.rawChildren.addChild(rxloader);
rxloader.name = "__rxloader";
rxloader.width = 0; rxloader.height = 0;
rxloader.load(new URLRequest("RanorexAutomation.swf"));
```

For Flex 4:

```
Flash/Flex
var rxloader : Loader = new Loader();
FlexGlobals.topLevelApplication.parent.addChild(rxloader);
rxloader.name = "__rxloader";
rxloader.width = 0; rxloader.height = 0;
rxloader.load(new URLRequest("RanorexAutomation.swf"));
```

3. Copy the **RanorexAutomation.swf** file, located in the Bin directory of your Ranorex installation, to your web server (where your .swf file is located) or where your AIR application is located.
4. Run your application in a supported browser (Internet Explorer, Firefox, Chrome, Safari)
5. If you run your Flash application from the local drive, add your output directory to the trusted locations of Flash Player as follows:

1. Open following link http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html#119065
2. Add your project output directory

Troubleshooting

After instrumenting your Flash/Flex/Air application, Ranorex Studio is still not able to recognize the respective UI elements in your application.

AllowScriptAccess parameter issues

The AllowScriptAccess parameter in the HTML code that loads a SWF file controls the ability to perform outbound URL access from within the SWF file.

Solution: Set this parameter inside the PARAM or EMBED tag to ‘always’.

AUT instrumented twice

Your Flash/Flex application was instrumented twice using different versions of the Flash components in provided by Ranorex Studio.

Solution: Revert to an uninstrumented version of your AUT and use only one of the instrumentation approaches described on this page to instrument your AUT.

Qt testing

Ranorex Studio supports out-of-the-box test automation of Qt applications.

Supported Qt versions:

- **Qt Widgets**: Qt version 4.5.3 or higher
- **Qt Quick:** Qt version 5.0.2 or higher
- **Qt WebKit:** Qt version 4.6.4 or higher

Only dynamically linked Qt applications are supported.

There is no need for manual instrumentation, Ranorex Studio will do it automatically.

> **Note**
>
> Mobile Qt apps are not supported.

Reverting to MSAA/Qt Accessibility recognition

If you don't want to use the default Ranorex Studio Qt object recognition, you can revert back to MSAA (Microsoft Active Accessibility) and Qt Accessibility recognition.

To do so:

1. In Ranorex Studio, **go** to **Settings > Plugins > Qt**.
2. **Set Use Qt legacy automation mode** to **True**.

Loading the accessibility plugin into your Qt application

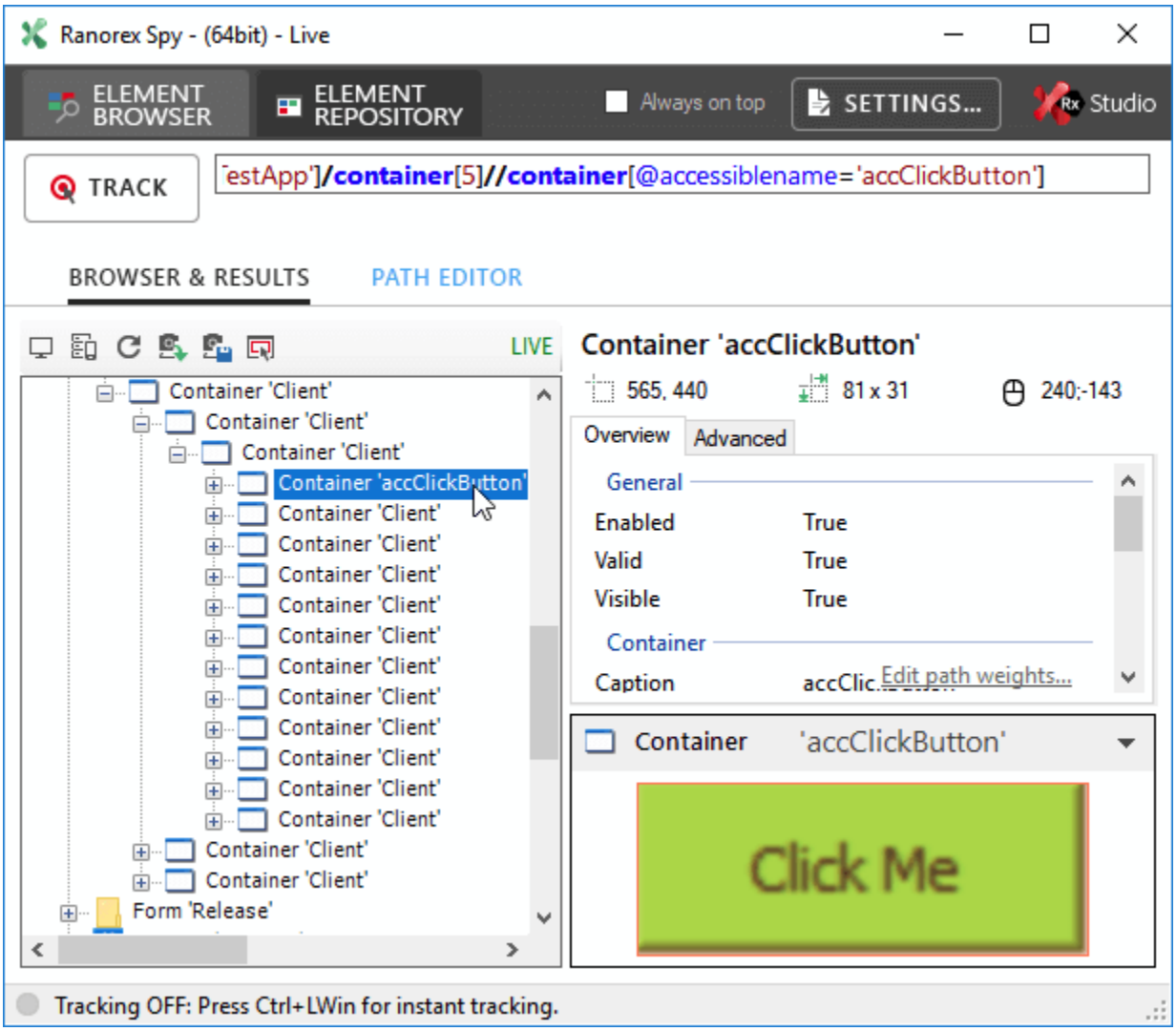
You also need to ensure your Qt application supports MSAA, or Ranorex Studio won't be able to identify UI elements in it. To do so, you need to **load the Qt accessibility plugin** in the Qt application under test and ship it with it. The accessibility plugin is included in the Qt SDK.

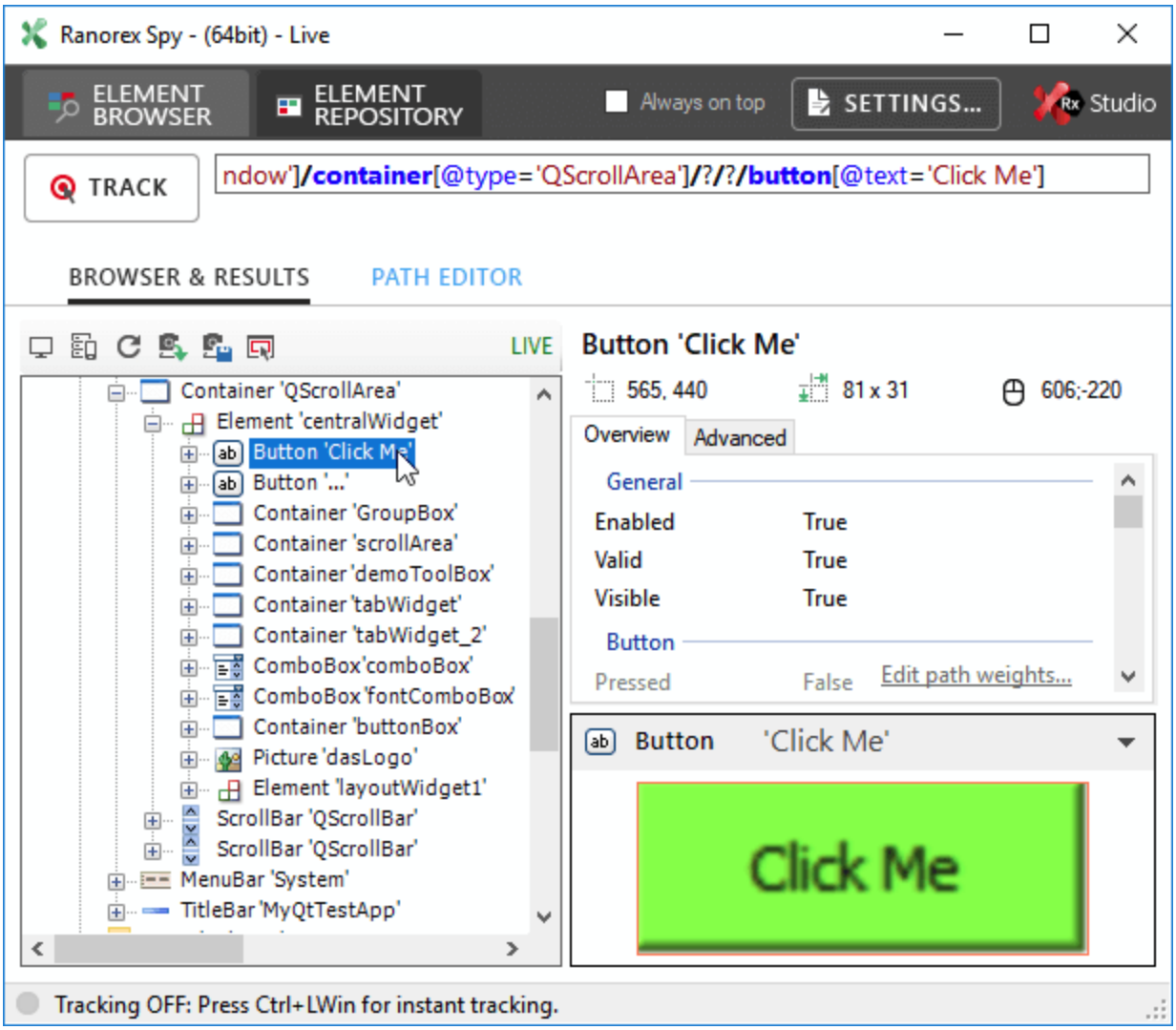
1. Go to the installation folder of the Qt SDK and locate the folder named **accessible**. If you don't have access to the Qt SDK, contact the developer of your application and ask them to send you the **accessible** folder.

2. **Copy** the **accessible** folder and all its contents to the **plugins** folder in your application's directory (e.g. Program Files\YourApplication\plugins).
3. In the root directory of your application (e.g. Program Files\YourApplication), **create** a new file called **qt.conf** and **insert** the following code. If the file already exists, simply **add** the following code to it.

```
[Paths]

Plugins = plugins
```

This will ensure Ranorex Studio can identify UI elements correctly, as shown in the two screenshots below:





Legacy applications

For some older technologies like VB6.0, Delphi or MFC based applications, Ranorex Studio has limited support for robustly identifying UI elements. However, the GDI RawText plugin increases object recognition for controls that use the Windows GDI drawing mechanism to display text on screen.

Reference

Using the GDI plugin for certain controls or classes is explained in Ranorex Studio advanced > Ranorex Spy > ⇢ GDI capture feature

SAP applications

On this page, you'll find out what steps you need to take before you can automate SAP applications.

Deselect Ranorex EXE manifest files

When installing Ranorex Studio, deselect the Ranorex EXE manifest files under Ranorex Main Components during setup.

If you've already installed them:

1. In Windows, **go** to **Add or remove programs**.
2. **Find** the Ranorex Studio entry.
3. **Select** it and **click Modify.**
4. **Select Change** in the dialogue that appears.
5. **Deselect Ranorex EXE Manifest Files** under **Ranorex Main Components**.
6. **Click Next** and **Finish** once setup is done.

Note

If this doesn't work, you can also turn off User Account Control (UAC) in Windows as a last resort. Please refer to the Microsoft Windows documentation for how to do so.

Enable scripting in SAP

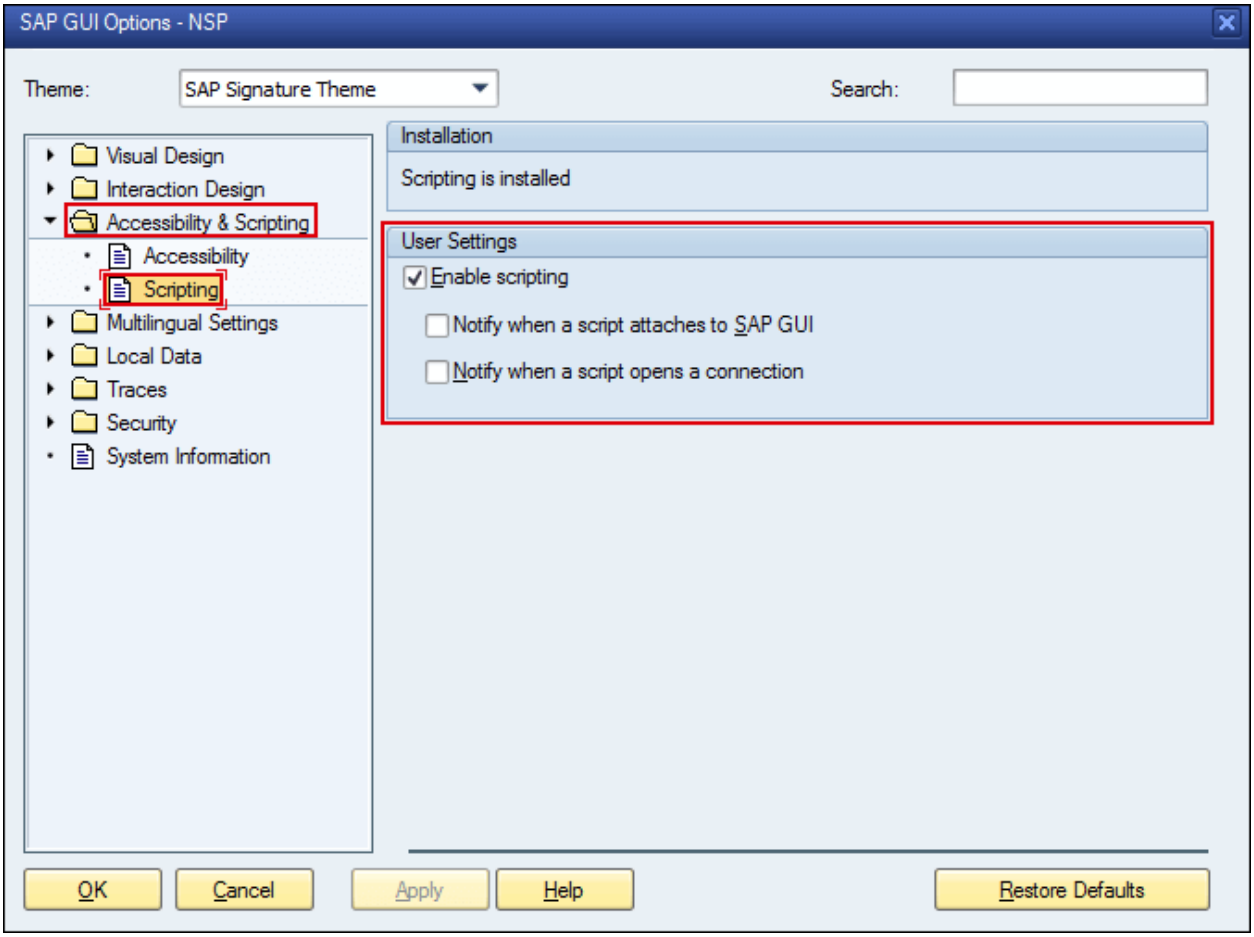
You need to enable scripting for both server and client in SAP for automation to work properly.

To enable scripting on the server:

1. **Set** the profile parameter **sapgui/user_scripting** to **TRUE** on the application server. Administrator privileges are required to do so.

To enable scripting in the client:

1. In SAP, **open** the **SAP GUI Options** dialog.
2. Under **Accesibility & Scripting > User Settings**, enable scripting and disable all notification options as shown below.



Troubleshooting

Insufficient privileges

Solution: Restart all Ranorex tools and the AUT with the same privileges, preferably as an administrator.

CEF applications

Ranorex Studio supports testing of applications based on the following CEF technologies:

- Chromium Embedded Framework (CEF)
- CefSharp
- Electron
- NW.Js
- Qt WebEngine
- JxBrowser

To enable CEF testing, you need to enable the remote debugging port in the AUT. This page explains how to do so.

Enable the remote debugging port

To test CEF applications with Ranorex Studio, you have to **enable the remote debugging port** in the AUT.

For most CEF technologies, **adding the following command line argument** (e.g. to the Run application action in a recording module) when starting the AUT is sufficient:

MyApplicationUnderTest.exe **–remote-debugging-port=8081**

> **Attention**
>
> The **port must not be used by another application or service** on the machine where you're running the test.

However, some technologies require a different approach.

CefSharp

If your AUT is based on **CefSharp**, the command line argument does not work. Instead, enable the remote debugging port in the source code of the AUT.

To set the remote debugging port in CefSharp, you have to specify it with help of the CefSettings class before initializing the control as follows:

```
var settings = new CefSettings();

settings.RemoteDebuggingPort = 8081;

Cef.Initialize(settings);
```

JxBrowser

If your AUT is based on **JxBrowser**, the command line argument does not work. Instead, enable the remote debugging port in the source code of the AUT by calling the **BrowserPreferences.setChromiumSwitches(String...)** method.

Please refer to this article in the official JxBrowser documentation to see how.

Troubleshooting

When testing CEF applications, you may come across a technology limitation warning. Here are potential causes and solutions for these technology warnings.

Another application is already listening on the remote debugging port

UI elements in your AUT can't be identified because another application is blocking the remote debugging port (e.g. Chrome DevTools). Close the blocking program and restart the AUT.

Solution:

Ranorex Studio normally tries to close the blocking application automatically. In some cases, like for the Chrome Developer Tools, this doesn't work. In this case, simply close all applications that may be connected to the AUT at the remote debugging port you specified.

If this fails, you can also try to change the remote debugging port number for the AUT.

Remote debugging not available

UI elements in your CEF AUT can't be identified because the remote debugging port isn't enabled in your AUT or can't be found.

Solution:

Make sure you've activated the remote debugging port correctly (i.e. in source code if required) and that no other applications are using it. If you've set the remote debugging port number to something else than 8081, try resetting it to this value.

General Troubleshooting

On this page, you'll find causes and solutions for limited object recognition that weren't covered by the other technology-specific pages of this chapter.

The issues are sorted by technology.

.NET WinForms

There are various reasons why object recognition may be limited when working with WinForms.

Insufficient privileges

Solution: Restart all Ranorex tools and the AUT with the same privileges, preferably as an administrator.

Insufficient security rights

This can be the case when you start the test executable and/or the AUT from a network drive or encrypted folder.

Solution: Copy both the executable build and the AUT to a folder on the desktop of the machine you want to run the test on.

Mixed-mode executable/obfuscated controls

The controls that can't be identified are implemented in a mixed-mode executable (not DLL). These are often created by obfuscating applications or assembly merging utilities. The .NET Framework doesn't support loading such mixed-mode executables in other processes, which is why Ranorex Studio and its tools cannot recognize such controls.

Solution: As a workaround, you can try to automate the non-obfuscated version of your application.

AUT and test executable target different platforms/processors

The controls that cannot be identified are implemented in an assembly that targets a particular platform/processor, while the test executable targets a different platform/processor. This causes isses on 64-bit operating systems, as the test executable can't load such assemblies.

Solution: Try changing the **Target CPU** (Ranorex Studio) or **Platform Target** (Visual Studio) option in the automation project settings to match the target of the automated application as described ⇢ on this page. If you don't know the target CPU/platform of the automated application, first try **Any processor** or **Any CPU**, respectively. If this doesn't work, try the remaining options.

.NET WPF

There are various reasons why object recognition may be limited when working with WPF.

UIAutomation element selected instead of WPF element

While selecting an element of a WPF application in the element tree of Spy, you selected the UIAutomation tree of that application.

Solution: The preceding sibling of the selected window will identify the same window, but provide the "WpfElement" capability instead of "UiAutomation". When browsing elements in Spy, select elements below the "WpfElement" window.

Insufficient privileges

Solution: Restart all Ranorex tools and the AUT with the same privileges, preferably as an administrator.

Active legacy UIA plugin

Using the legacy UIA plugin to identify WPF elements limits object recognition.

Solution: If you experience problems, switch to the native WPF plugin in Ranorex Studio under **Settings > Plugins > WPF > WPF Legacy/UIA interaction > WpfImprovedOnly**.

.NET Silverlight

Windowless property set to True

The **Windowless** property of your Silverlight AUT is set to **True**.

Solution: Set it to **False** and restart all Ranorex tools and the AUT.

Java

Insufficient privileges

Solution: Restart all Ranorex tools and the AUT with the same privileges, preferably as an administrator.

Delphi

Irretrievable remote object pointer

The started Ranorex tool(s) or the test executable couldn't retrieve the remote object pointer for a particular item or classname. This is usually caused by an outdated Delphi version.

Solution: Check if you're using a ⇢ supported Delphi version. If you are, please contact our support team for assistance.

Win32-based applications

There are various reasons why object recognition may be limited when working with Win32-based applications.

Insufficient privileges

Solution: Restart all Ranorex tools and the AUT with the same privileges, preferably as an administrator.

Bit bridge couldn't be started

The started Ranorex tool(s) or the test executable and the automated process don't have the same bit width and the 32/64 bit bridge couldn't be started.

Solution: Please contact our support team for further assistance.

Bit bridge is disabled

The started Ranorex tool(s) or the test executable and the automated process don't have the same bit width and the 32/64 bit bridge is disabled.

Solution: For full functionality, enable the 32/64 bit bridge in Ranorex Studio under **Settings > Advanced > Enable 32/64 Bit Bridge**.

Microsoft Internet Explorer

Slow automation

The Ranorex addon for Internet Explorer isn't installed. This makes object recognition significantly slower.

Solution: Install the addon by running the Ranorex Studio setup again and only select the addon.

> **Note**
>
> In Internet Explorer:
>
> - The "Enhanced Protected Mode" has to be disabled.
> - The "Internet Explorer Enhanced Security Configuration" has to be disabled on Windows Server machines.

Image-based automation

Asynchronous dispatching disabled

The setting **Asynchronous dispatching of mouse and keyboard events** is disabled. In most cases, this will cause errors in image-based automation.

Solution: Enable the setting in Ranorex Studio under Settings > Advanced.

Geometry-related issues

Controls may be displaced, remain empty or scrolled out of view during automation.

Solution: Set the **Use ensure visible** property of the corresponding repository items (and their parent folders) to **False** as described in the chapter ⇢ Manage repository items.

Windows 8 UI apps

There are various reasons why object recognition for applications based on the Windows 8 UI may be limited.

Disabled setting "Use UiaLauncher…"

The setting **Use UiaLauncher to elevate privileges for processes started by tools** is disabled. This limits object recognition.

Solution: Enable the setting in Ranorex Studio under **Settings > Advanced**.

Ranorex Studio not installed in a secure location

Ranorex Studio is not installed in a secure location.

Solution: Install Ranorex Studio to \Program Files\ or \Program Files (x86)\.

Test executable not started by a Ranorex tool

The test executable wasn't started by a Ranorex tool.

Solution #1: Start the test suite from Ranorex Test Suite Runner.

Solution #2: Start the test executable with the Ranorex UiaLauncher (\bin\Ranorex.UiaLauncher32.exe) by passing the executable name as the first argument, e.g. Ranorex.UiaLauncher32.exe\bin\debug\MyTest.exe.

Solution #3: Add the following code to the Program.cs/Program.vb of the executable build of your test project:

C#

```
if (Util.IsRestartRequiredForWinAppAccess)
  return Util.RestartWithUiAccess();
```

VB.NET

```
If Util.IsRestartRequiredForWinAppAccess
 Then Return Util.RestartWithUiAccess()
```

"Run this program as an administrator" enabled

You enabled "Run this program as an administrator" in the compatibility settings of a Ranorex tool executable, e.g. for the RanorexStudio.exe file. Enabling this options prevents access to Windows 8 UI apps.

Solution:

Disable the setting for any Ranorex tool executables. If you want to run a tool with administrator rights, create a shortcut to the respective executable and enable the option in the shortcut properties under Shortcut > Advanced.

Mobile

There are various reasons why object recognition may be limited when working with WinForms.

Ranorex Service App not running

The Ranorex Service App isn't running on the mobile device. This limits automation functionality, e.g. starting or stopping apps won't be possible.

Solution: Make sure the ⇢ Service App is installed and running.

iTunes not running

iTunes couldn't be detected. It's required to connect to iOS devices via USB.

Solution: Make sure iTunes is ⇢ installed and started and then restart all Ranorex tools.

Trying to record in an emulator

You are using an emulator and trying to record a mobile test in it. Object recognition is limited in this case and Ranorex Studio reverts to image-based recording, i.e. UI elements will only be recognized by their coordinates. The test will not be robust against changes in the UI or even the window size of the emulator.

Solution: For full functionality, treat the emulator as if it was a real mobile device. Add it as an endpoint and build a mobile test as ⇢ explained here.

Miscellaneous

Anti-virus or security software

Antivirus or security software can sometimes affect object recognition.

Solution: Add an exception for the respective Ranorex process or temporarily switch off the specific security application.

RanoreXPath identifying multiple UI elements

A particular RanoreXPath identifies multiple elements when you only want it to identify one. This is because several instances of the same process are running, e.g. multiple instances of Mozilla Firefox.

Solution: If you will only have one instance running during the test, ignore this limitation. Otherwise, make the path more specific, so it resolves to only one UI element, even with multiple instances of the containing process running.

Different DPI scaling values

When testing with Ranorex Studio or one of its tools on a multi-display setup, different DPI scaling values can lead to object recognition issues.

Solution: Set the scaling of all connected displays to 100 %.

Failed EnsureVisible action

This warning occurs when Ranorex Studio tries to execute an action that has an EnsureVisible operator, i.e. requires that the targeted UI element is visible, but can't bring the window that contains the UI element to the foreground. Usually, this is caused by insufficient privileges.

Solution: Restart Ranorex Studio as administrator or use the Ranorex.UiaLauncher to start your AUT.

If you start the AUT through Ranorex Studio, e.g. with a Run application action, activate **Settings > Advanced > Use UiaLauncher to elevate privileges for processes started by tools.**

If you start the AUT separately, you need to call Ranorex.UiaLauncher32.exe from the command line. It is located in the \Bin\ folder of your Ranorex Studio installation folder, C:\Program Files (x86)\Ranorex\Studio\Bin by default.

The syntax for running an application through the UiaLauncher is:

```
Ranorex.UiaLauncher32.exe <path to application>
```

Source and revision control

Introduction to source control

A source control system, also called revision control or version control system (VCS), keeps track of changes to a file or set of files. One can thus see who made changes and when. It allows reverting single files, or an entire project, to a previous state and comparing changes over time. It provides an easy way to recover previous versions of a file or project.

There are two types of version control systems: centralized version control systems, such as Subversion and Team Foundation Version Control, and distributed version control systems like Git.

Important notice

This User Guide section provides information on the software prerequisites for each of the supported version control systems and how they can be used within Ranorex Studio.

As a precondition we assume you have knowledge about version control in general, as well as your specific version control system. Setting up and configuring the version control system of your choice is not in the scope of this documentation.

Source control in Ranorex Studio

Ranorex Studio supports three version control systems: Git, Subversion, and Team Foundation Version Control. Use the menu on the left to get to the corresponding chapters for each system.

Git

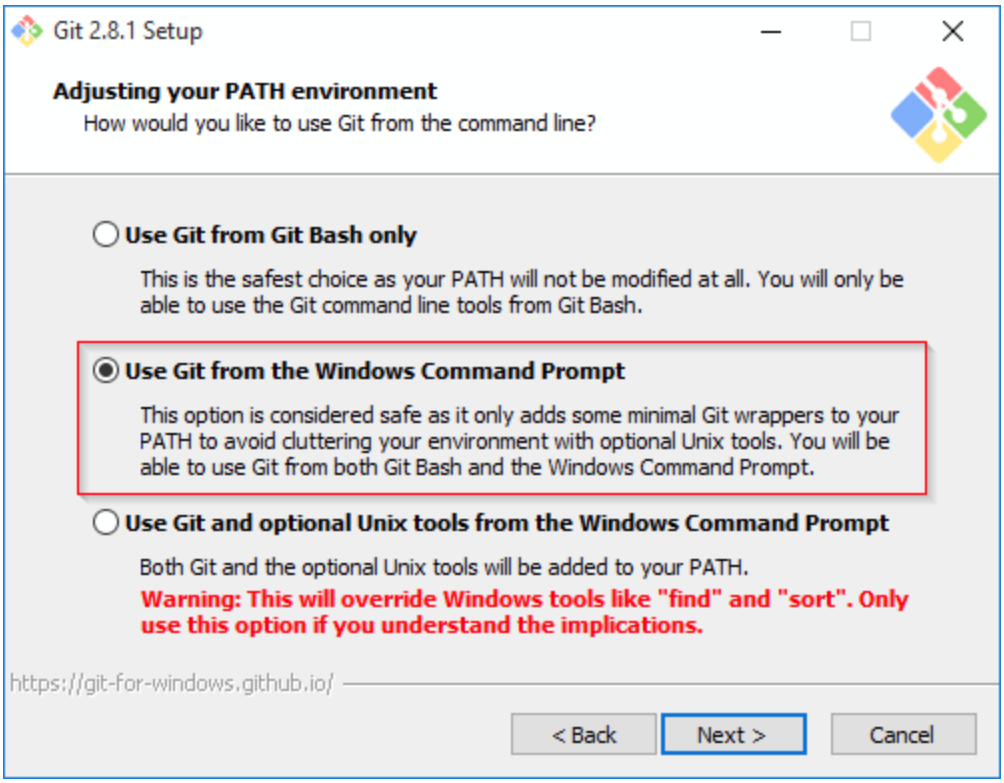
About Git and Ranorex Studio

Git is a distributed version control system which is free and open source.

In order to use Git as a source control provider in Ranorex Studio, the following preconditions have to be fulfilled in the given order:

- **Git** needs to be installed
  Download Git for Windows here: https://git-scm.com/download/win

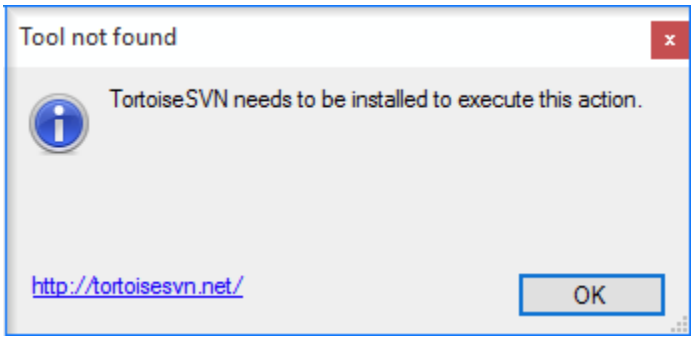
  Please make sure you choose to 'Use Git from the Windows Command Prompt' during Git installation!



- **TortoiseGit** needs to be installed
  Download TortoiseGit here: https://tortoisegit.org/download

TortoiseGit is a Windows shell interface to Git and is needed, for example, to overlay icons which show the file status.

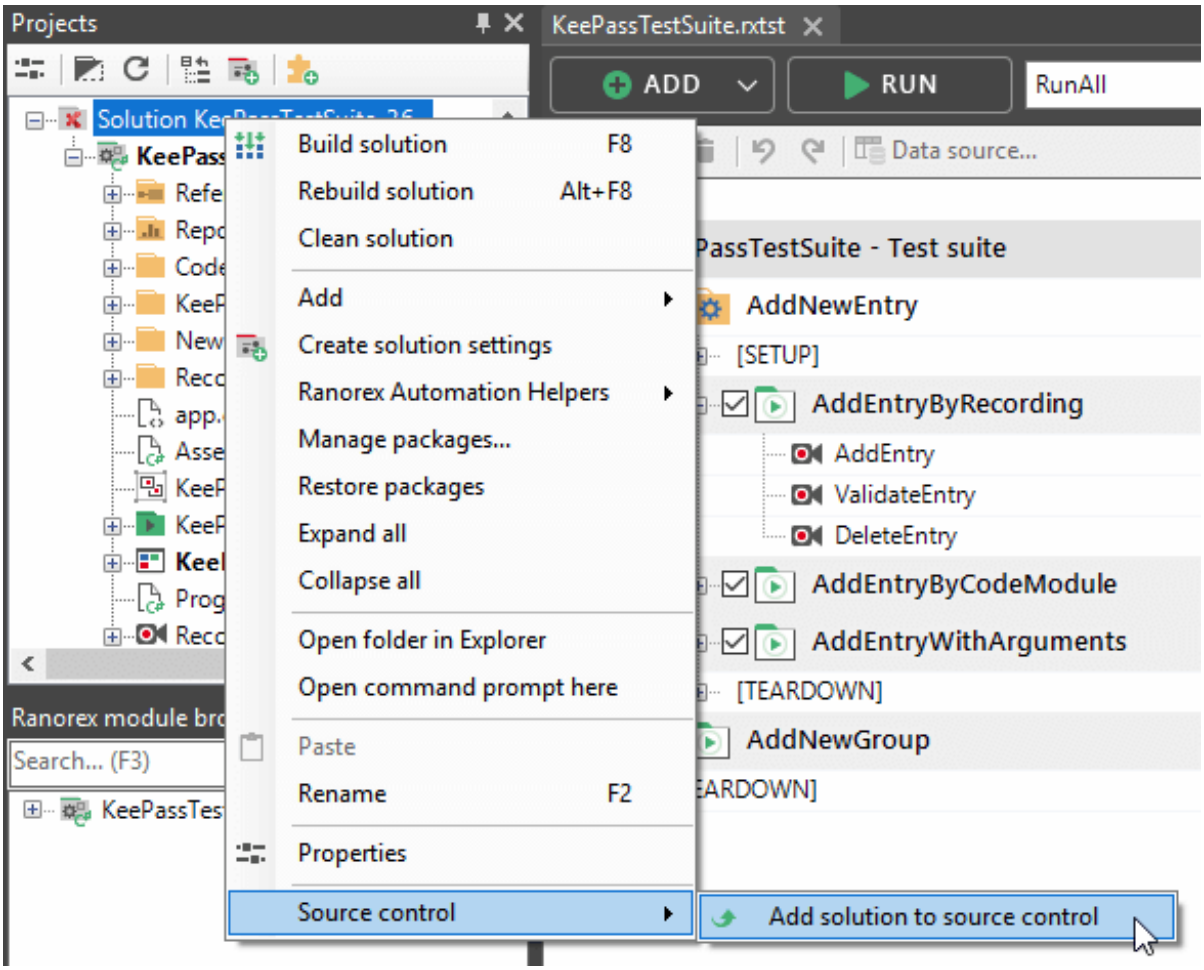
Ranorex Studio will assist you with this dialog in case the required prerequisite is not present on the machine:



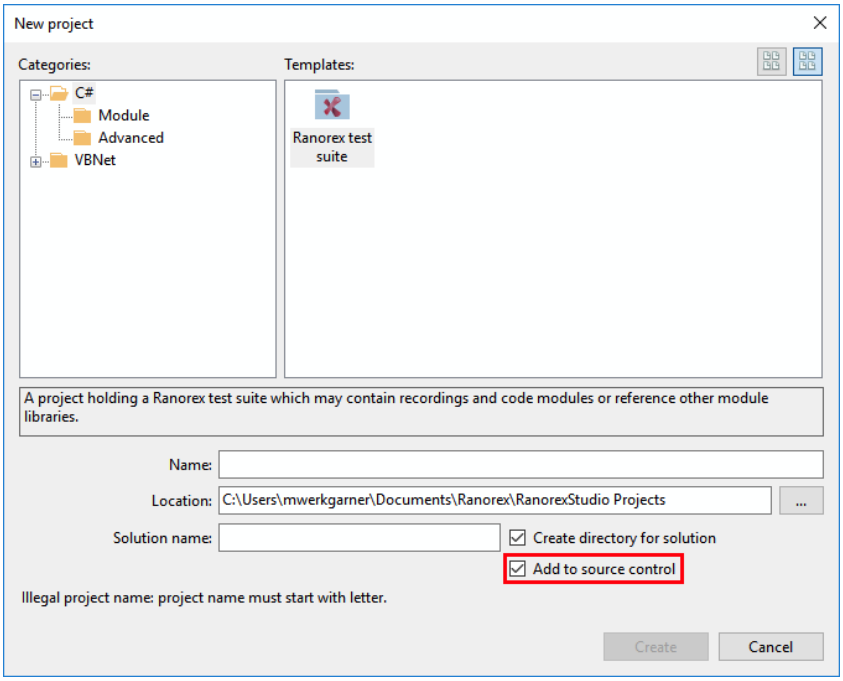
Add a Ranorex Solution to Git

Please make sure your Git infrastructure is set up and working.

To add an **existing Ranorex Solution** to Git, open the context menu of the solution. Go to ‘Source Control’ and select ‘Add Solution to Source Control’.

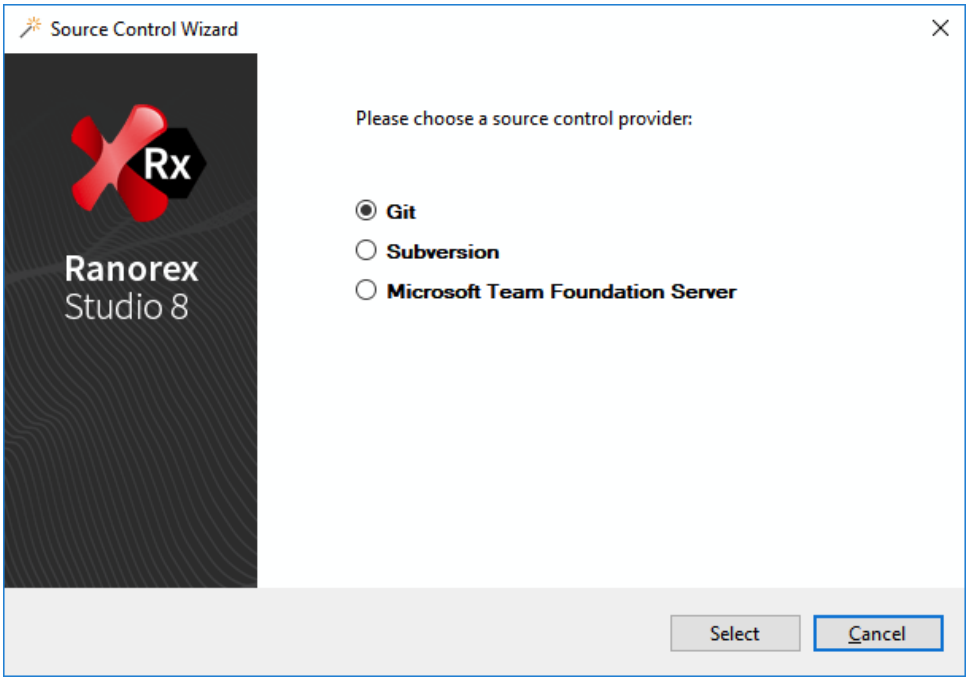


To add a **new Ranorex Solution** to Git, check the option ‘Add to Source Control’ in the ‘New Project’ dialog.

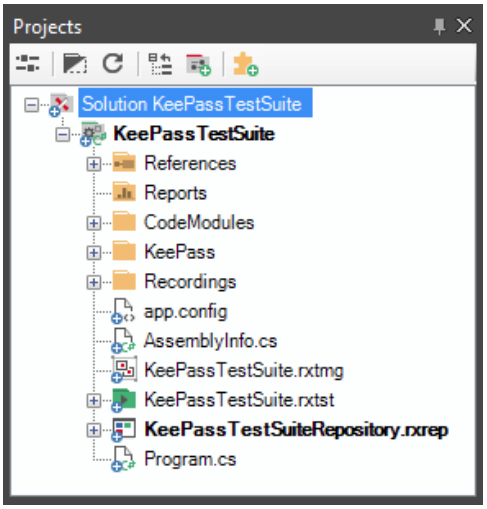


Please follow these instructions:

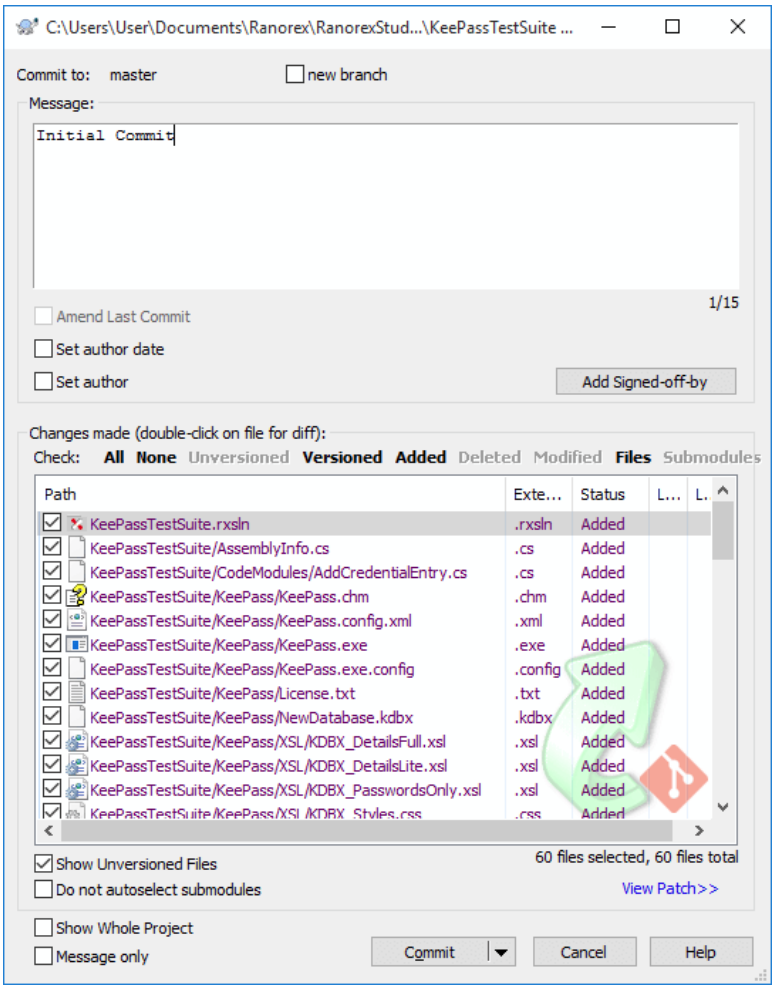
1. Choose **Git** as Source Control provider.



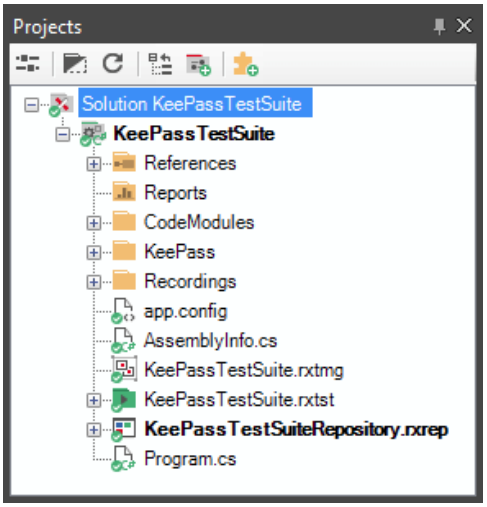
2. The Ranorex solution is automatically configured as a new local Git repository. All files are indicated with a plus sign in the projects view.



3 To commit the current state to the local repository click **Commit...** in the context menu of the solution.



4. After the commit a check symbol is shown in the projects view.

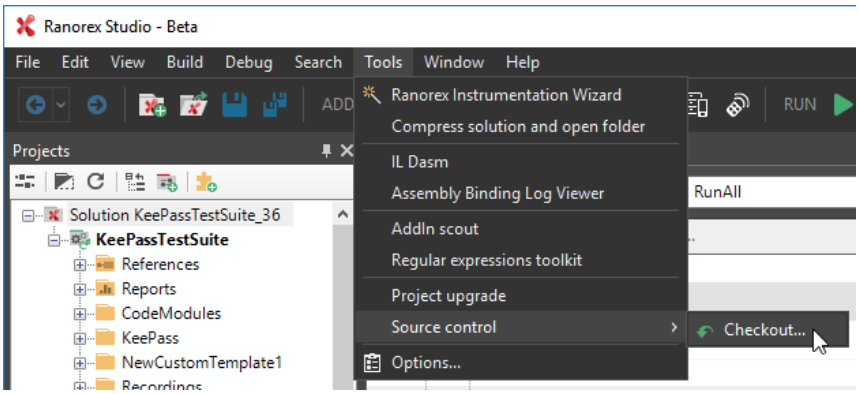


All further Git related steps should be defined in your workflow.

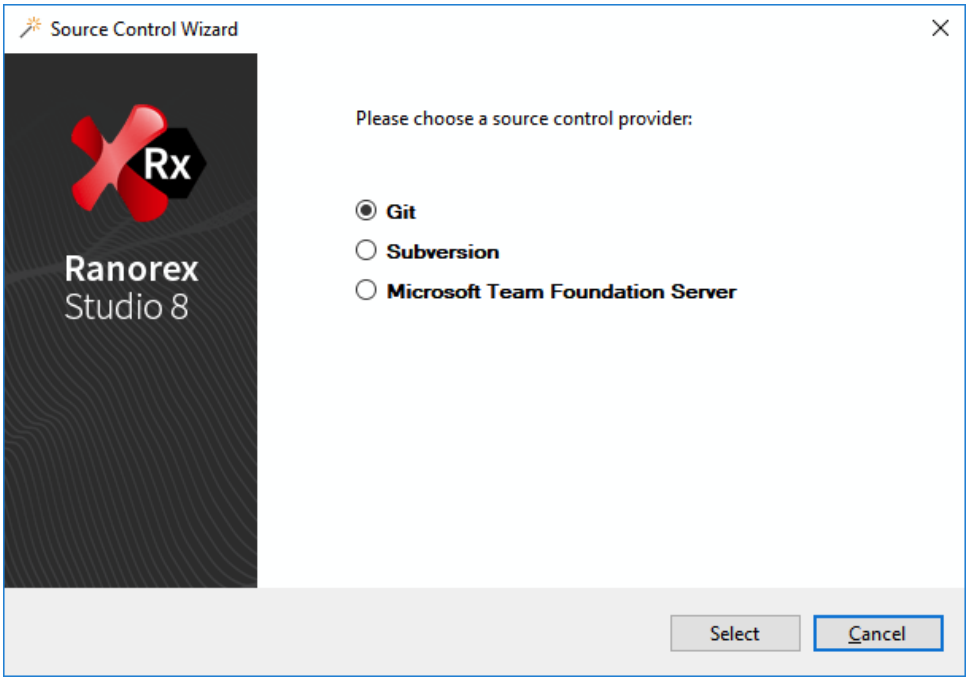
Check out a Ranorex Solution from Git

Please make sure your Git infrastructure is set up and working.

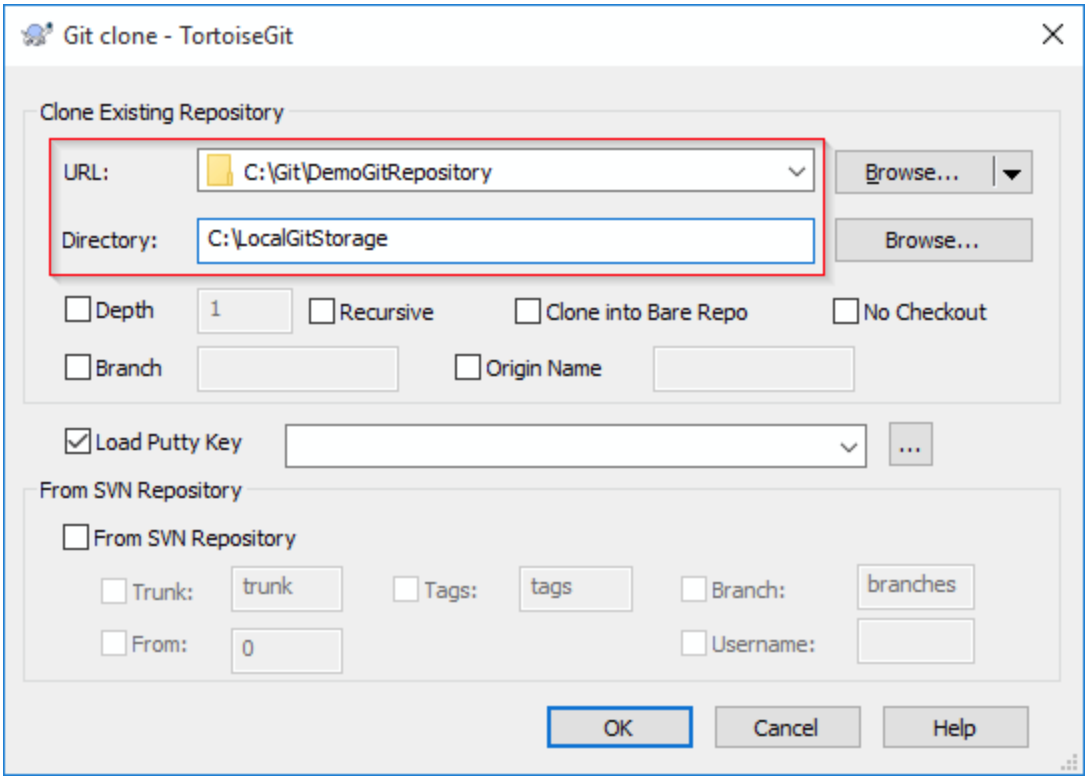
1. Go to **Tools > Source Control > Checkout...**



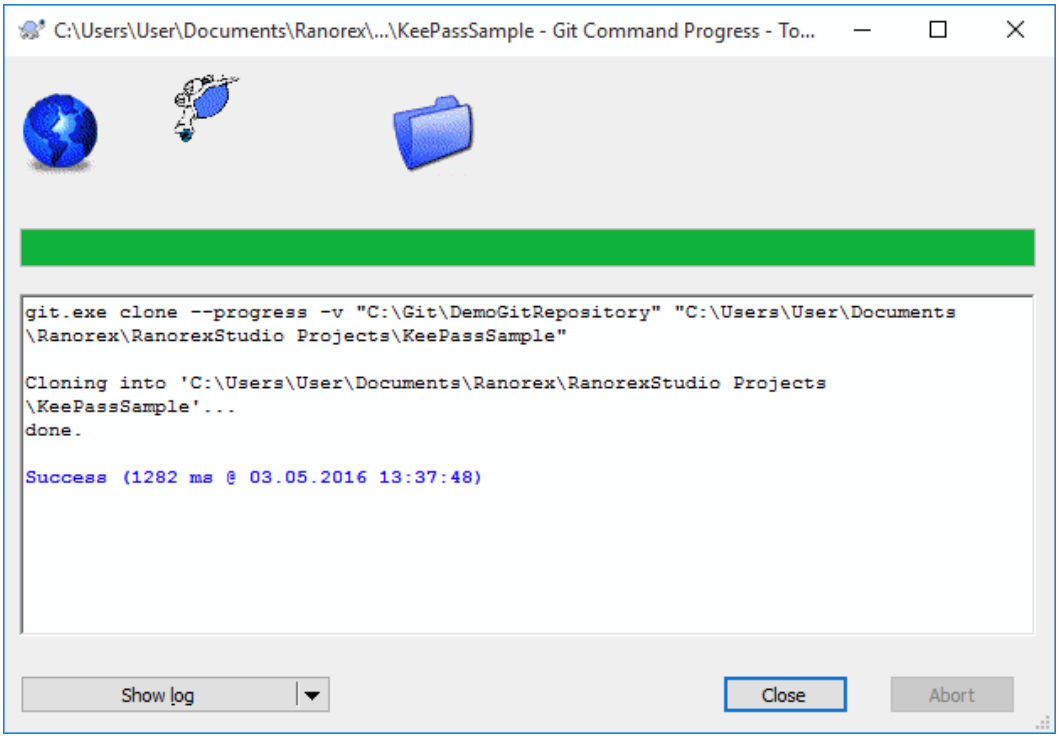
2. Choose **Git** in the Source Control Wizard.



3 TortoiseGit will ask you for the path to the repository and where you would like to store the files on your local machine. Fill in the URL to the Git repository as well as the path to the local directory and click **OK**.

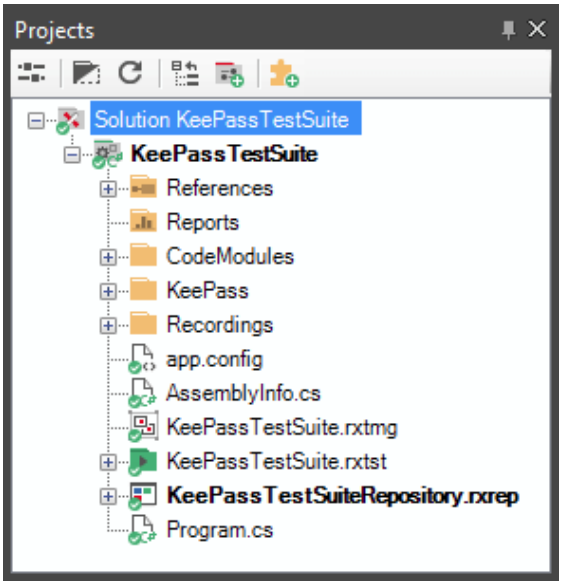


4. The whole repository will be cloned to the local folder. After a successful clone this dialog will be shown.



5. In case your repository includes a Ranorex Solution in its root, this solution will be opened automatically. If this is not the case, you have to open the Ranorex Solution manually from the file System.

After opening the Ranorex Solution from the local folder you see the icon overlays in Projects View.



All further Git related steps should be defined in your workflow.

Icon overlays in Projects View

Overlay icons are added to the items in the Projects View in Ranorex Studio, as the solution is under source control.

Icon overlays	
Normal	Not locally modified, no changes waiting for commit.
Conflicted	Indicates a conflict.
Modified	Modified, changes are waiting to be committed.
Added	Marked for addition, waiting to be committed.

Subversion

About Subversion and Ranorex Studio

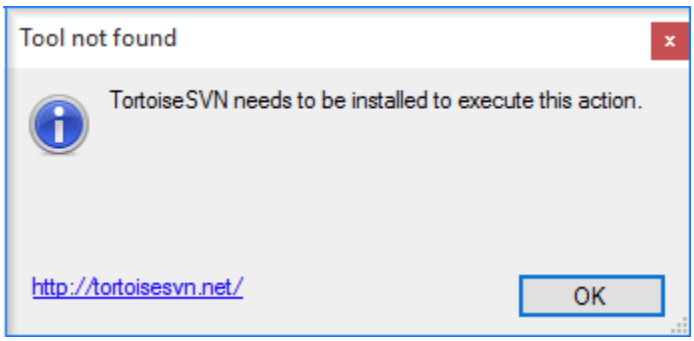
Subversion is a centralized version control system by Apache (https://subversion.apache.org/).

To get more information about Subversion, please have a look at the official Subversion online documentation: http://svnbook.org

Ranorex Studio uses two applications for working with subversion, which are installed with Ranorex Studio by default:

- SharpSvn
  Is a set of libraries for working with Subversion.
- TortoiseSVN
  It is a Windows shell extension for Subversion and provides the icon overlays showing the file status in Ranorex Studio.

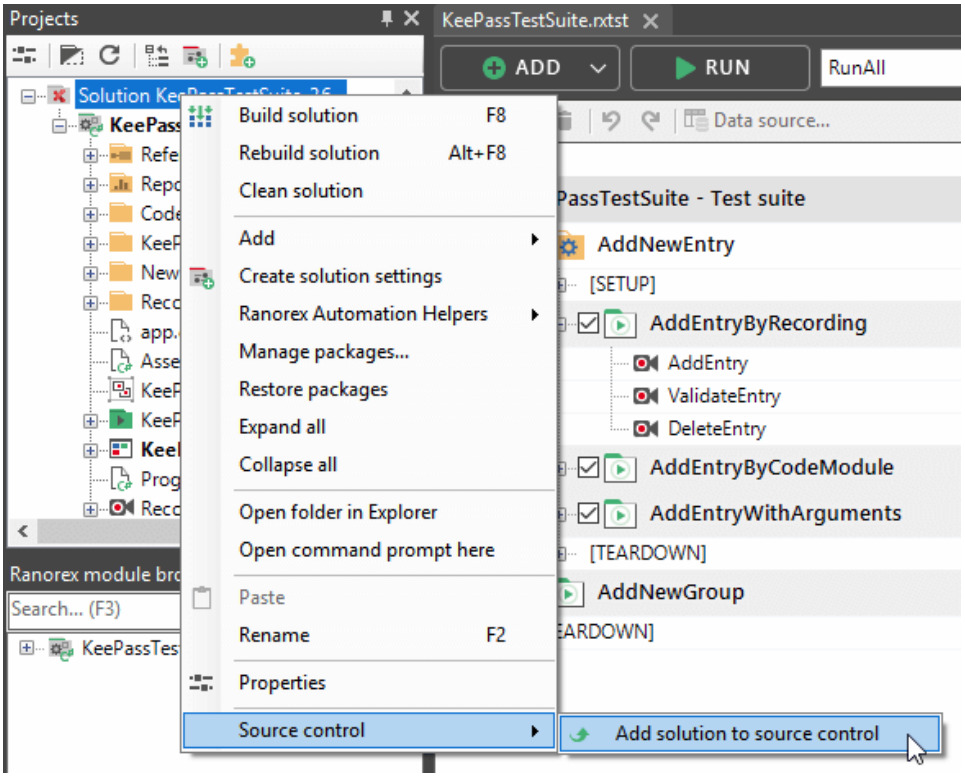
Ranorex Studio will assist you with this dialog in case the required prerequisite is not present on the machine:



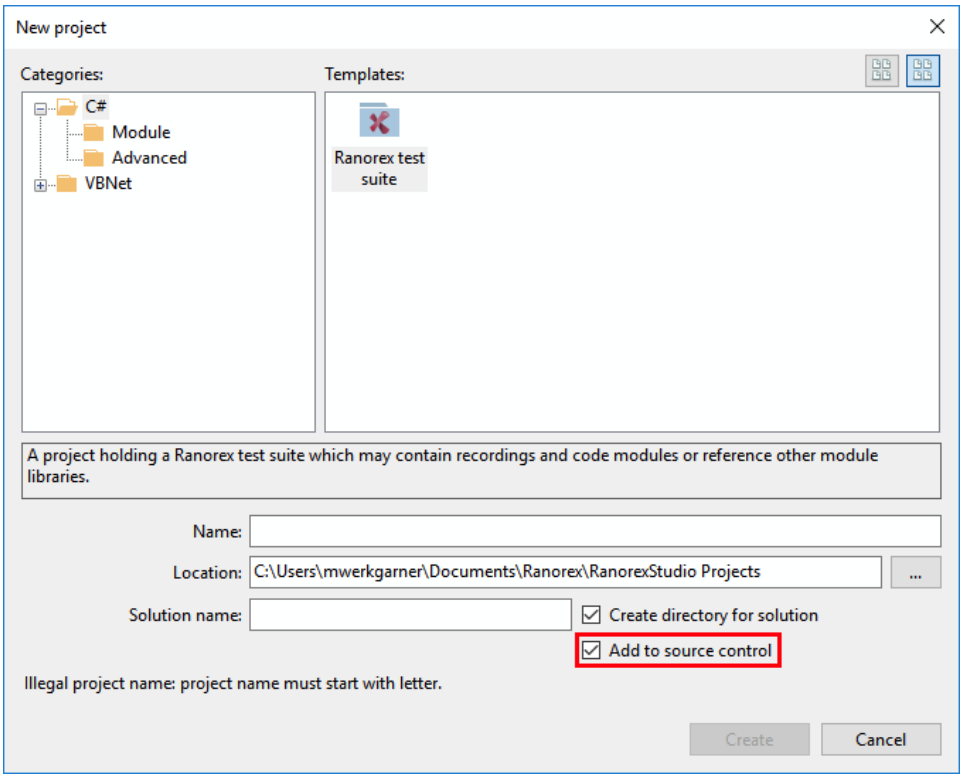
Add a Ranorex Solution to Subversion

Please make sure your Subversion infrastructure is set up and working.

To add an **existing Ranorex Solution** to Subversion, open the context menu of the solution. Go to 'Source Control' and click on 'Add Solution to Source Control'.

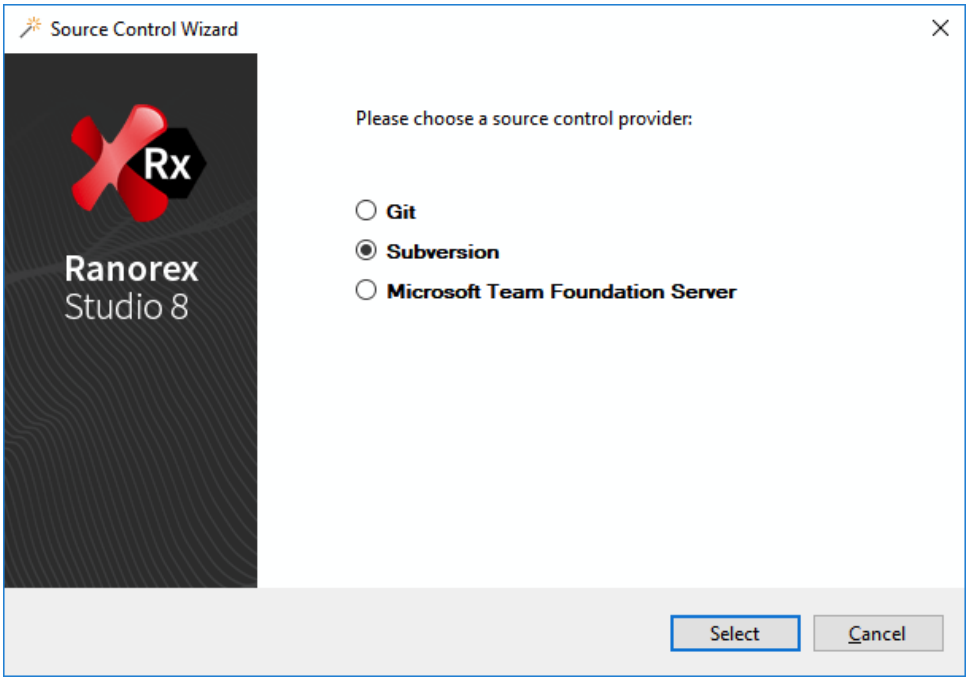


To add a **new Ranorex Solution** to Subversion, check the option 'Add to Source Control' in the 'New Project' dialog.



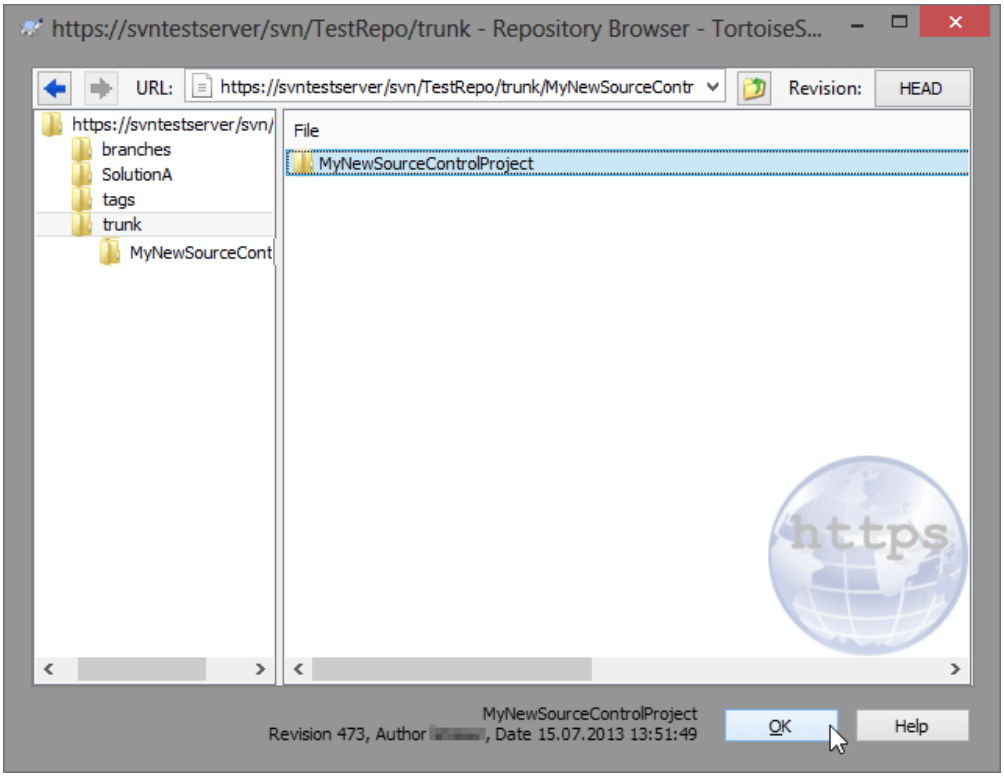
Please follow these instructions:

1. Choose ‘Subversion’ as Source Control provider.

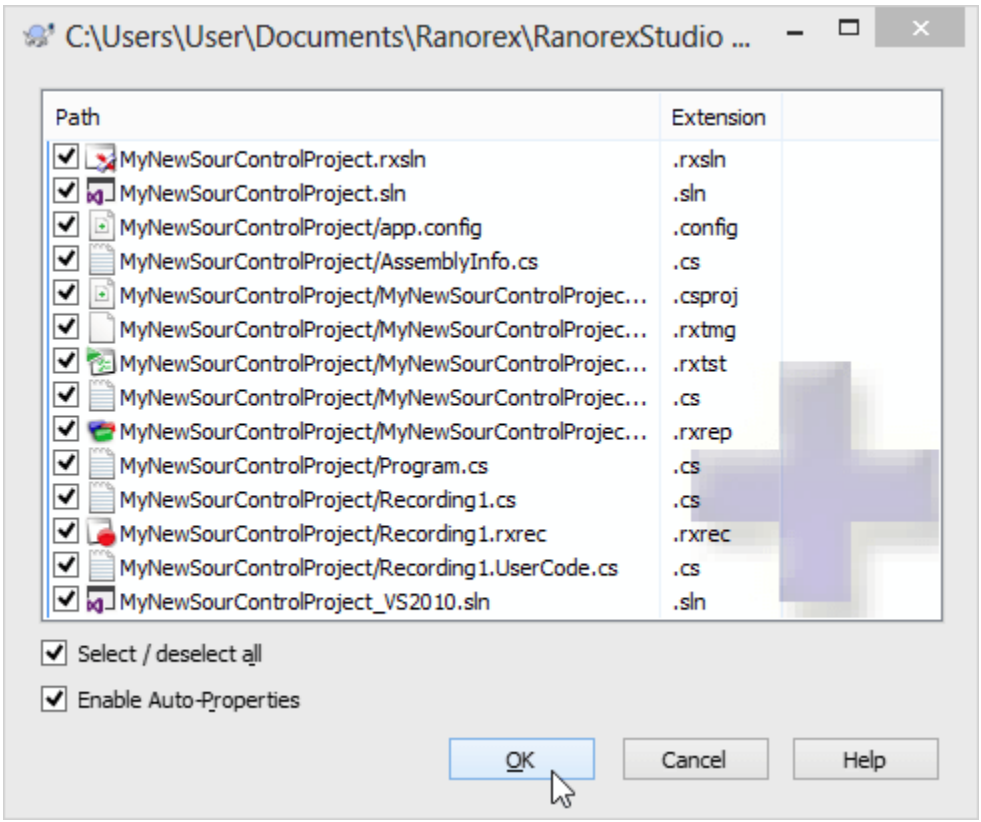


2. TortoiseSVN will ask for the URL of the repository.

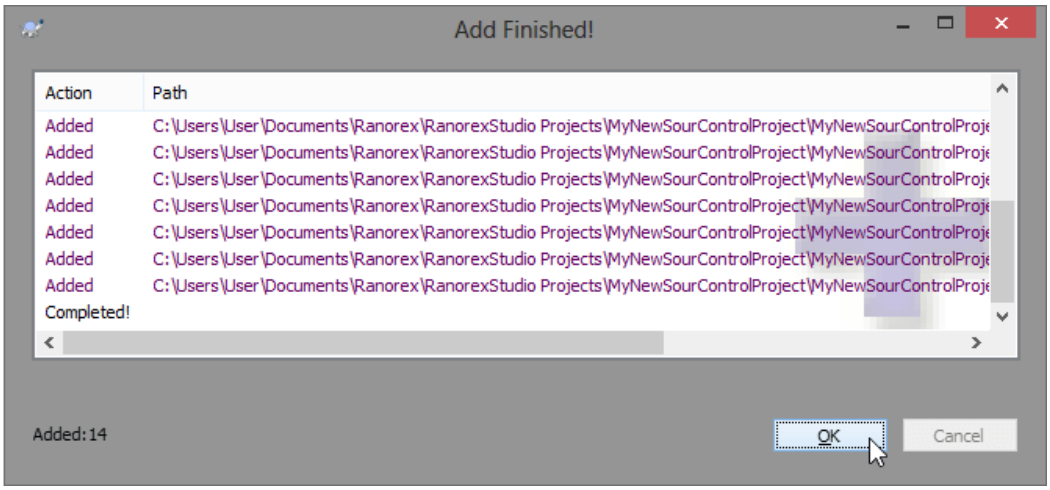
3. Authenticate yourself on the subversion server.



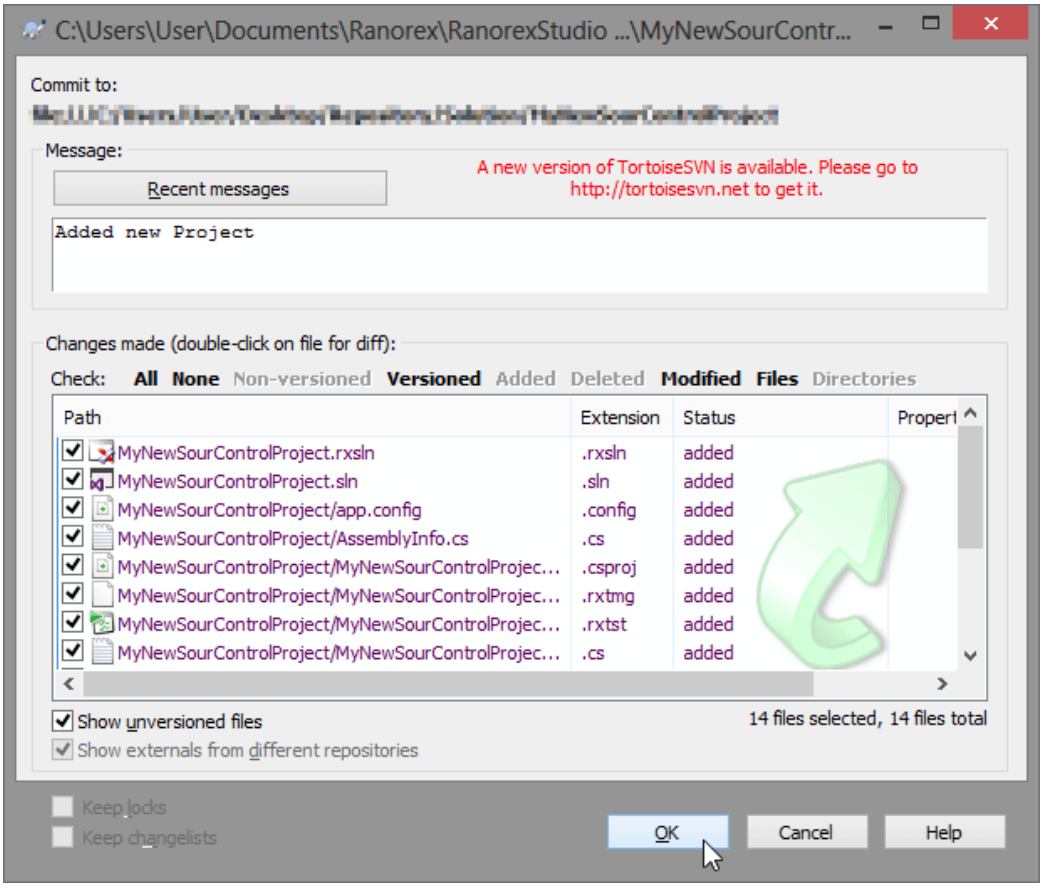
4 Choose the folder in the repository you want to add your solution to.



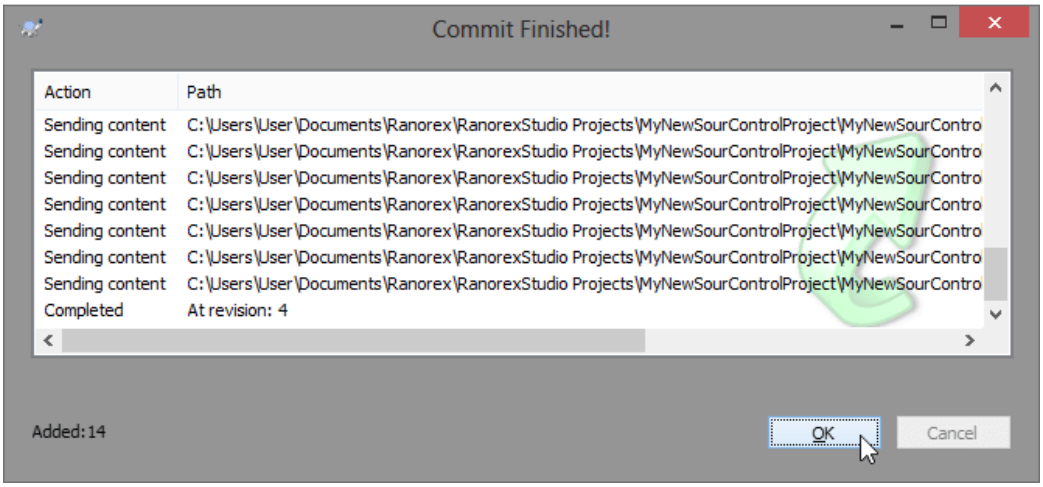
5 Check the files you want to have under source control and uncheck the files you don't want to have under source control respectively.



6 The chosen files will be added to source control.

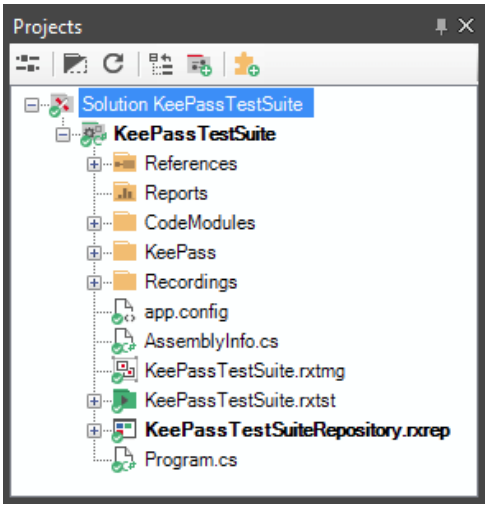


7 Now, commit the changes to the Subversion server.



8 The commit is done.

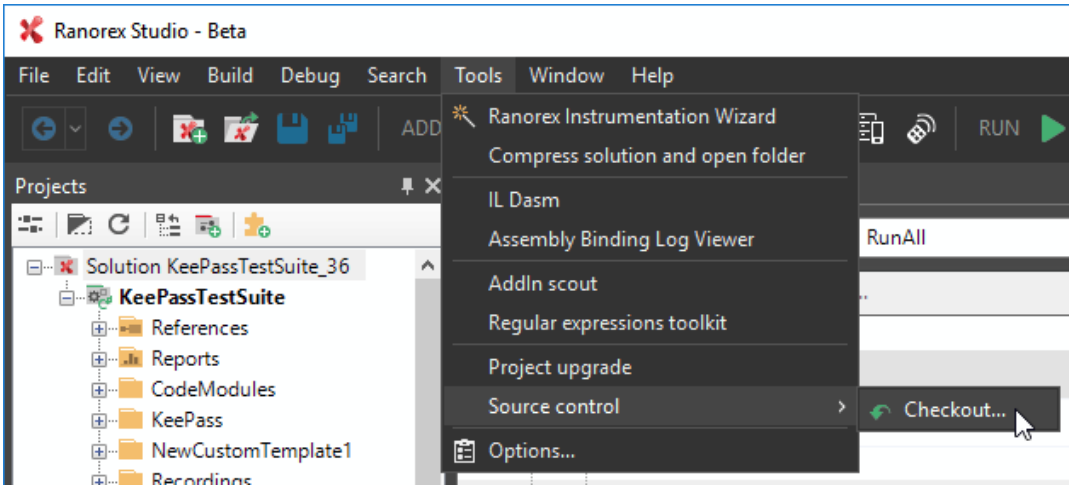
After performing these steps, your solution is under source control and your local copy is up to date.



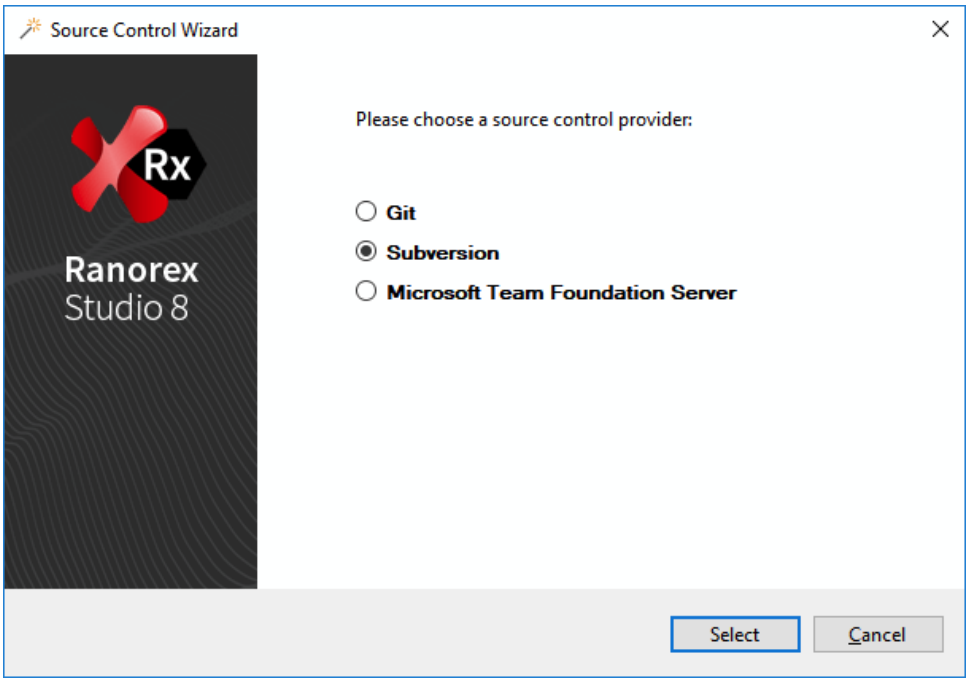
Check out a Ranorex Solution from Subversion

Please make sure your Subversion infrastructure is set up and working.

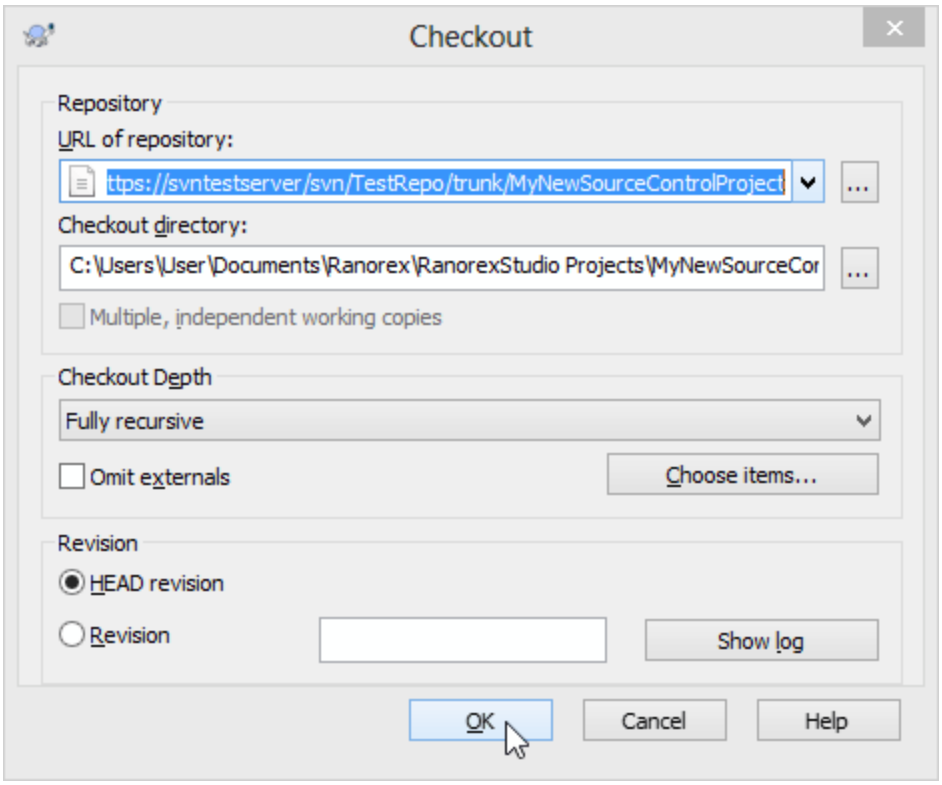
1 Open ‘Tools’ menu, move to ‘Source Control’ and click on ‘Checkout…’.



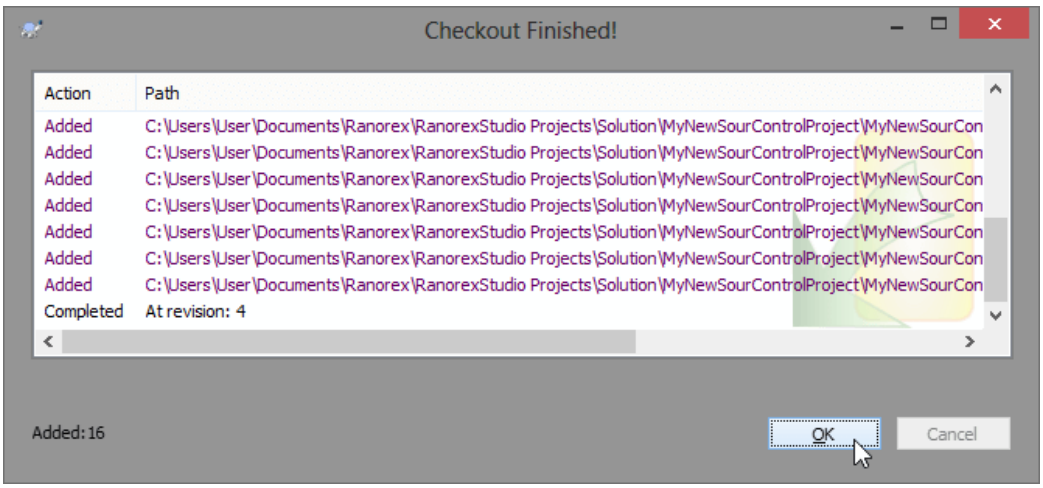
2 Choose ‘Subversion’ as Source Control provider in Source Control Wizard.



3 Enter the URL of your repository and specify the checkout directory.



4 The chosen project will be checked out.



Icon overlays in Projects View

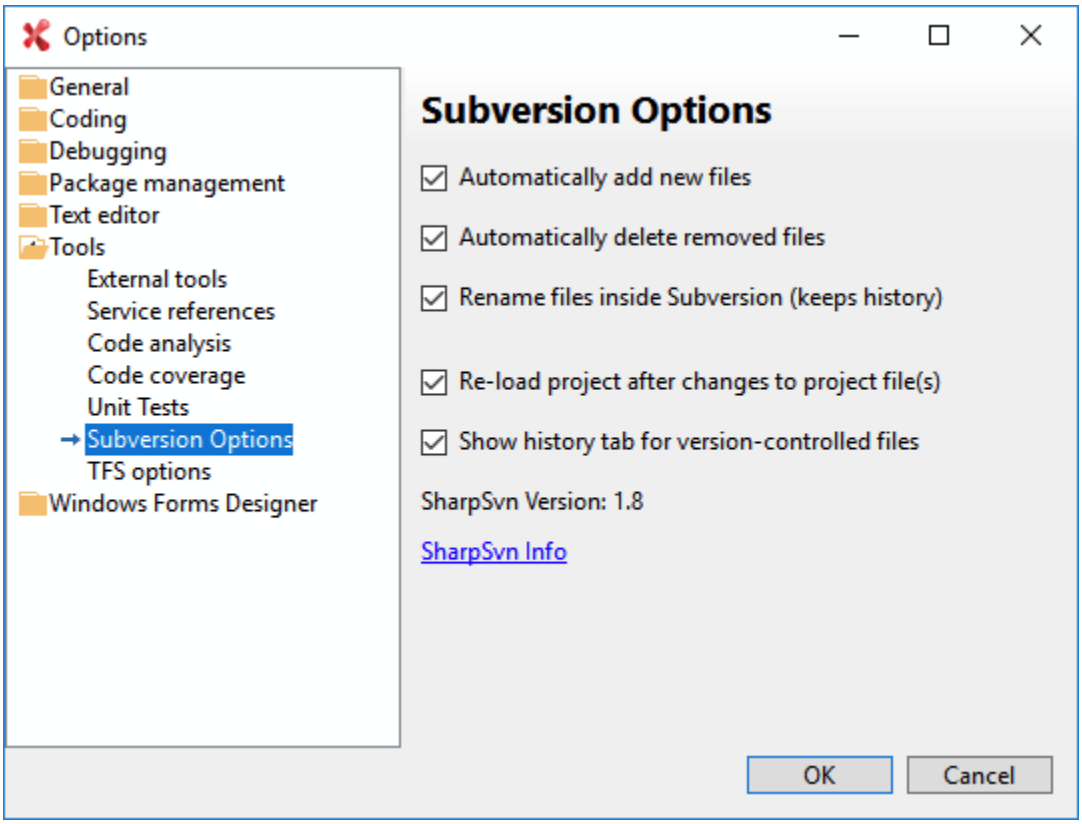
Overlay icons are added to the items in the Projects View in Ranorex Studio, as the solution is under source control.

Icon overlays	
Normal	Not locally modified, no changes waiting for commit.
Conflicted	Indicates a conflict.
Modified	Modified, changes are waiting to be committed.
Added	Marked for addition, waiting to be committed.

Options

In Ranorex Studio Options you can find options related to Subversion.

Open main menu item ‘Tools’ and choose ‘Options…’. In the following dialog select folder ‘Tools’ and sub-item ‘Subversion Options’. By default, all checkboxes are checked. The currently used SharpSvn version can be found here.



Using another version of SharpSvn and TortoiseSVN

In order to use another version of TortoiseSVN and SharpSvn than the preinstalled ones, you have to download and install the version you want to use:

- Download and install TortoiseSVN first from http://tortoisesvn.tigris.org.
- After that download, the corresponding package for SharpSvn from https://sharpsvn.open.collab.net.
  Please make sure you choose the right version of SharpSvn, which matches the TortoiseSVN version.
- Extract the downloaded Version.
- Copy the extracted folder to 'RanorexStudioAddInsSourceControlSubversionAddin'.
- Once you restart Ranorex Studio, the correct version of SharpSVN will be automatically chosen.

Team Foundation Version Control (TFVC)

About Team Foundation Version Control (TFVC) and Ranorex Studio

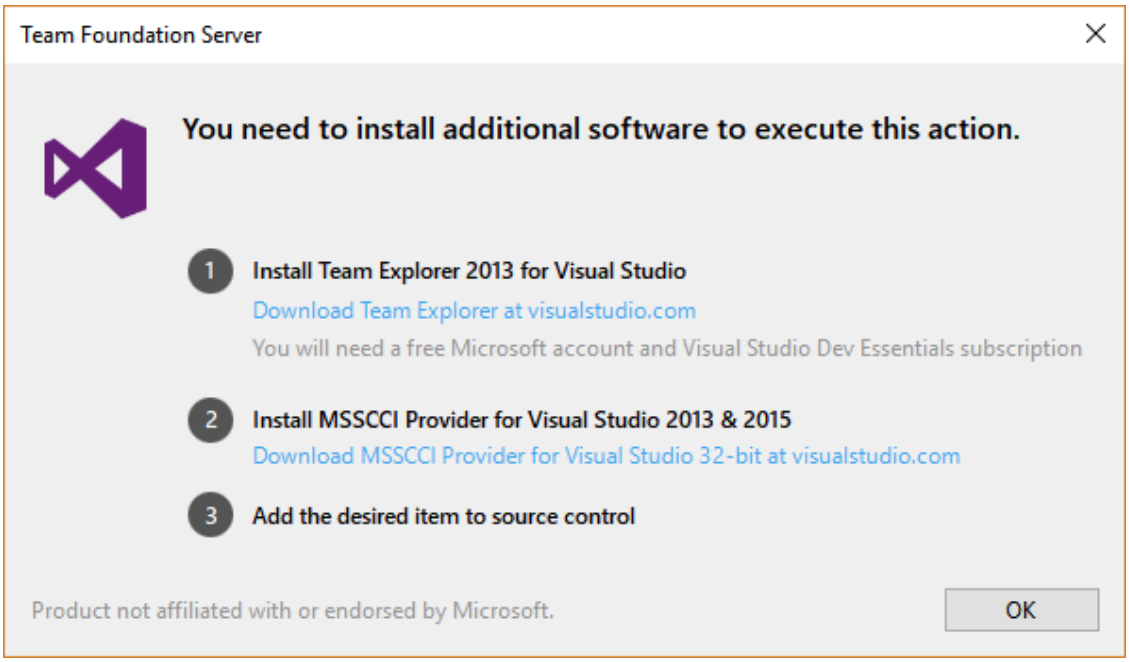
Team Foundation Version Control (TFVC) is a centralized version control system by Microsoft included in Team Foundation Server.You will need to install the following software to work with TFVC in Ranorex Studio:

Team Explorer 2013 for Visual Studio (Requires a free Microsoft account and Visual Studio Dev Essential subscription)

MSSCCI Provider for Visual Studio 2013 & 2015

Don't worry about the 2013/2015 designations in the software names if you're using later versions (e.g. 2017) of Visual Studio or TFS. They are compatible and the TFVC integration will work.

Ranorex Studio will assist you with this dialog in case the required software is not present on the machine:

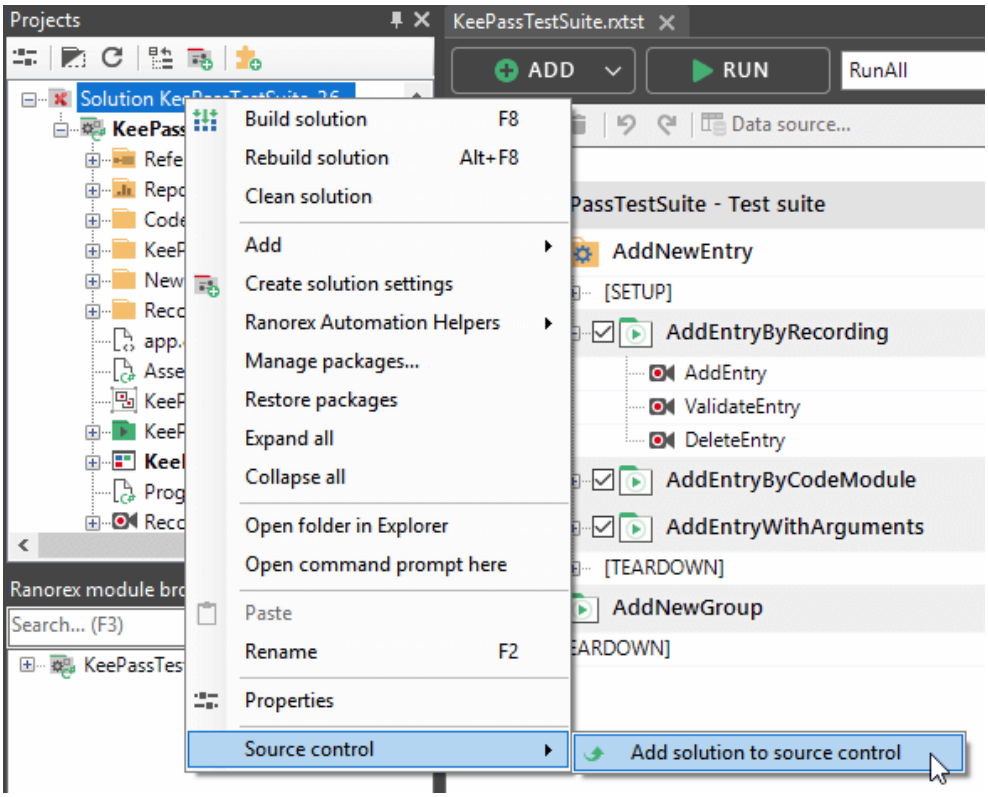


The MSSCCI interface supports single byte character sets (SBCS) only. Here you can find a description on how to change the code page. This means there is no MBCS support.

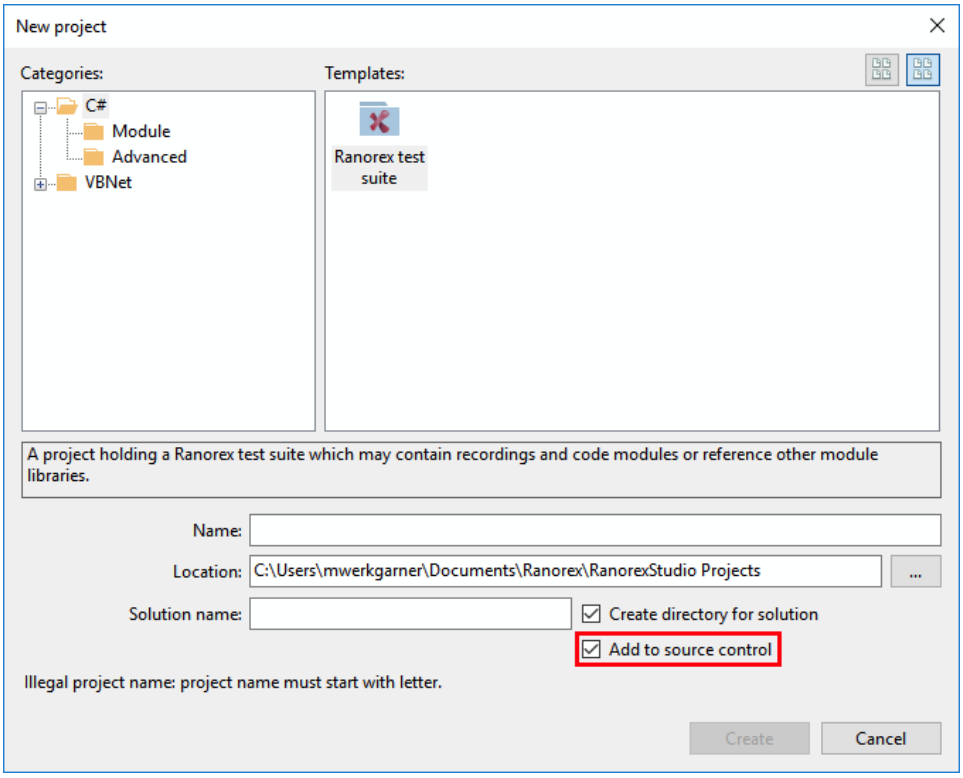
Add a Ranorex Solution to TFVC

Please make sure your Team Foundation Server infrastructure is set up and working.

For adding an **existing Ranorex Solution** to TFVC open the context menu of the solution. Go to ‘Source Control’ and click on ‘Add Solution to Source Control’.

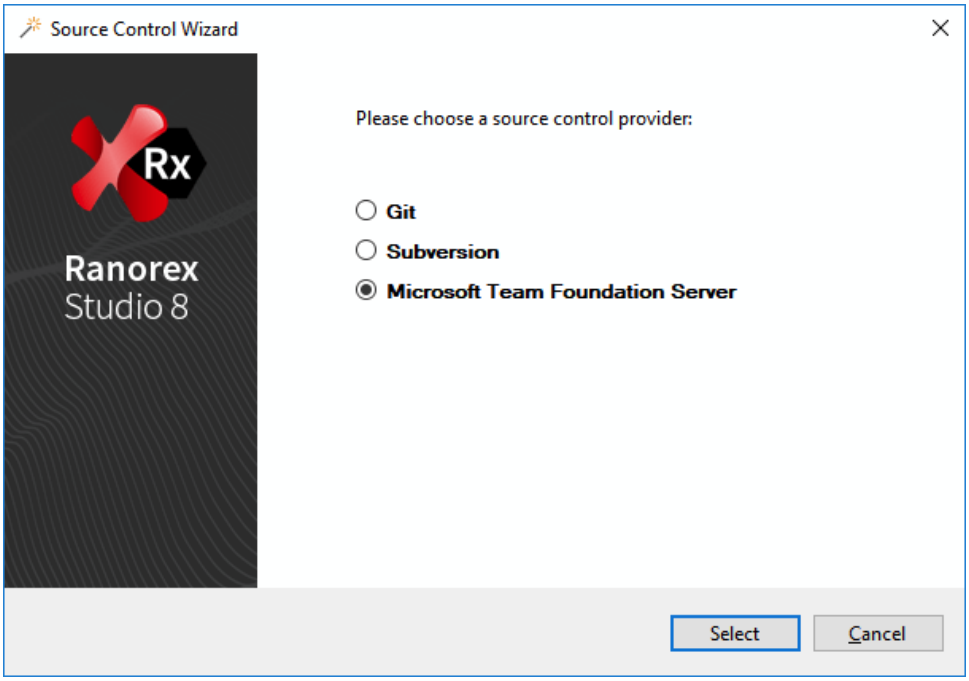


To add a **new Ranorex Solution** right from the beginning to TFVC check the option 'Add to Source Control' in the 'New Project' dialog.

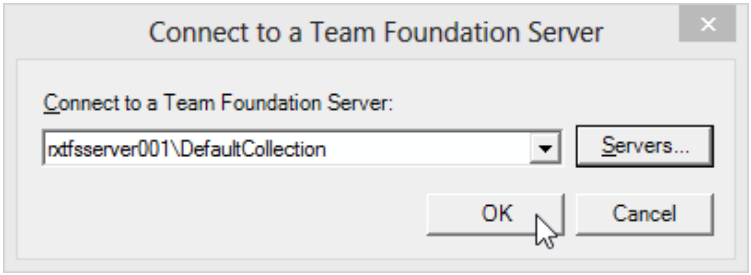


Please follow these instructions:

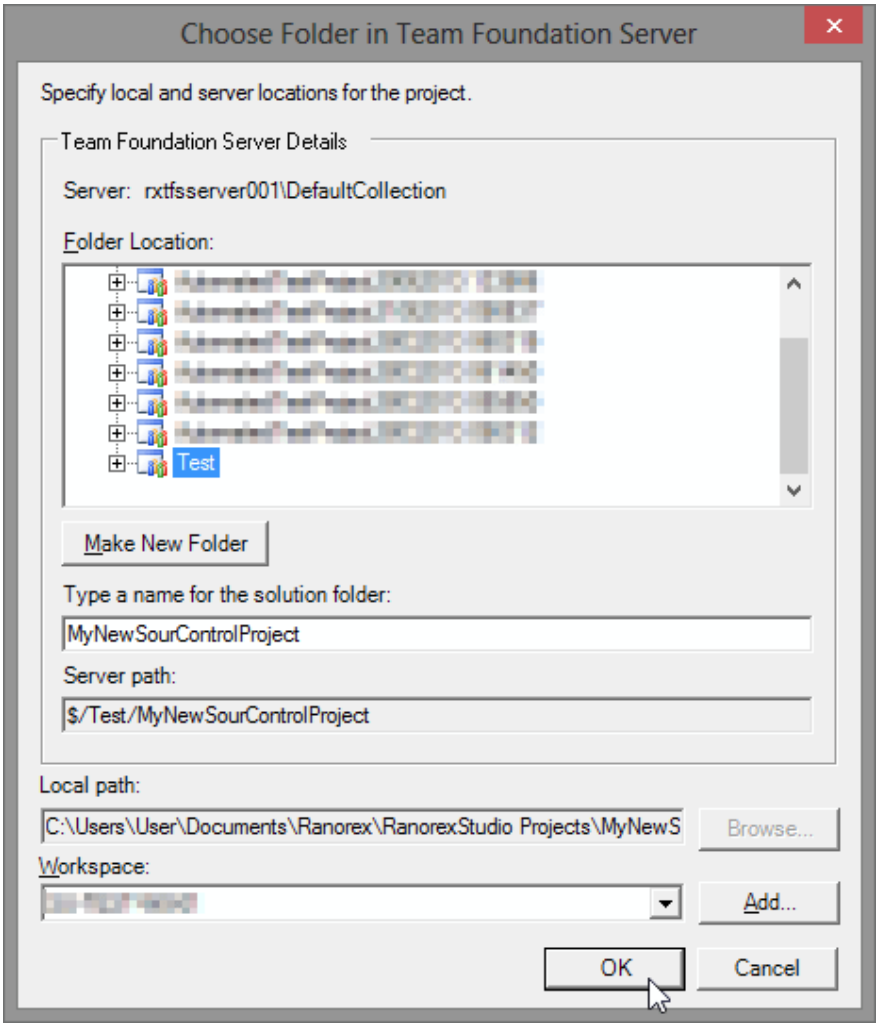
1. Choose ‘Microsoft Team Foundation Server’ as Source Control provider.



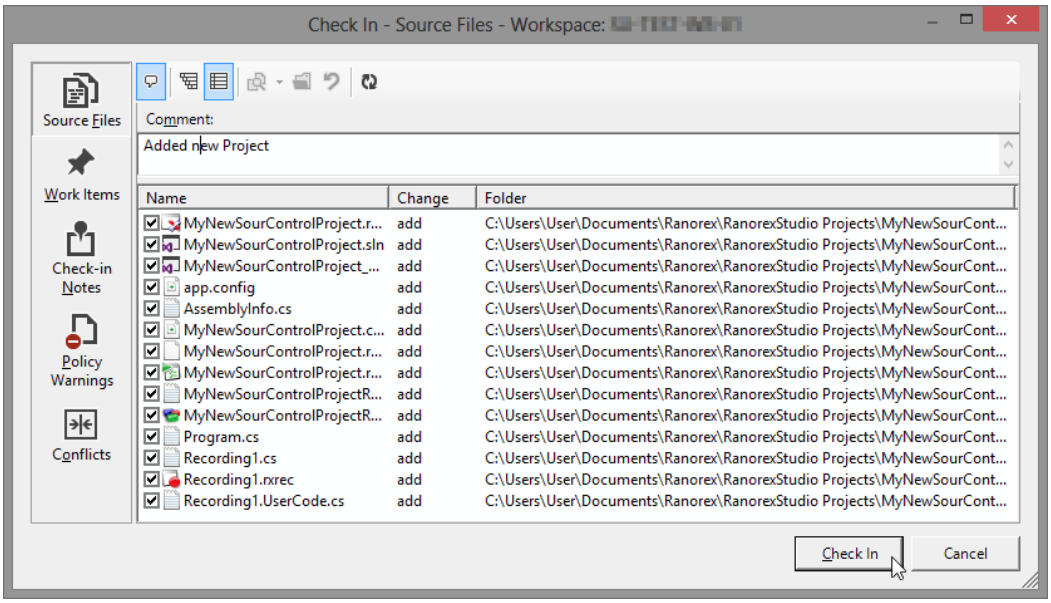
2. Select your Team Foundation Server.



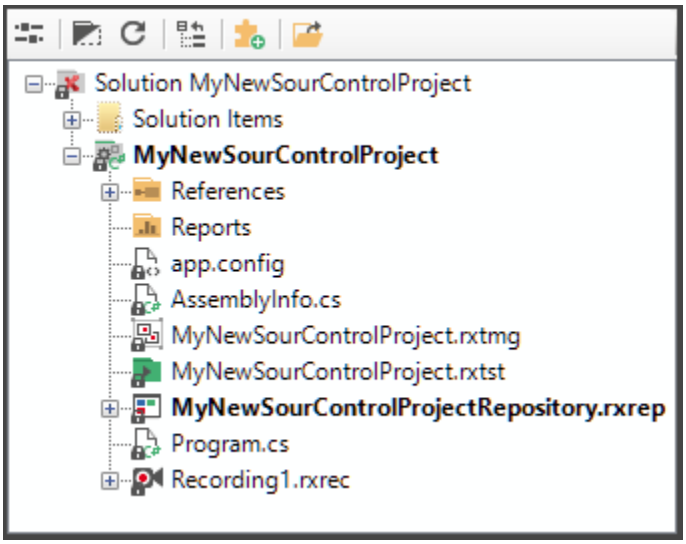
3. Choose the folder you want to add your solution to.



4 Select the files you want to have under source control and deselect the files you don't want to have under source control respectively and perform a check in.



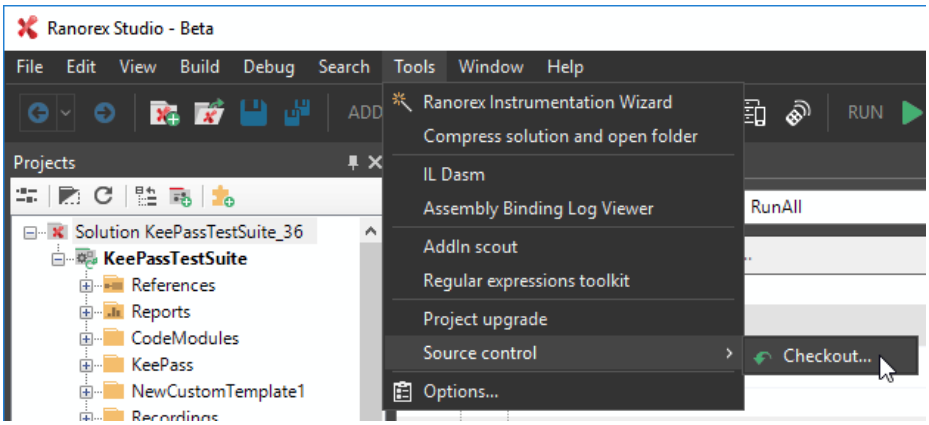
After performing these steps, your solution is under source control and your local copy is up to date.



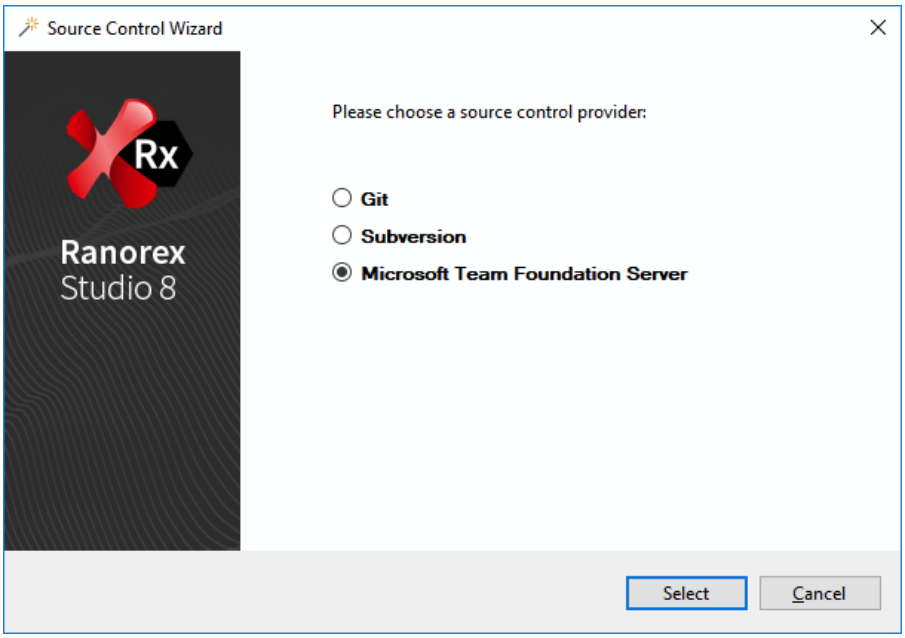
Check out a Ranorex Solution from TFVC

Please make sure your Team Foundation Server infrastructure is set up and working.

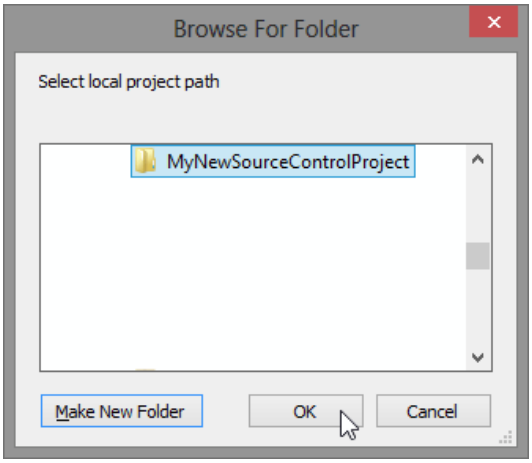
1. Open ‘Tools’ menu, move to ‘Source Control’ and click on ‘Checkout…’.



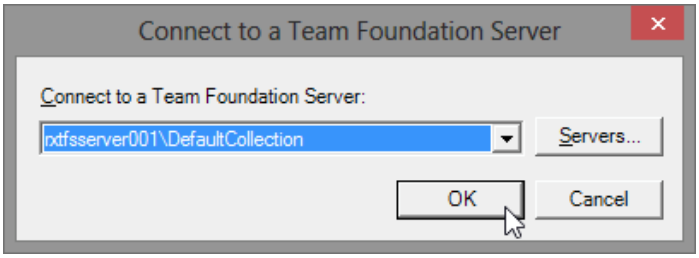
2 Choose ‘Microsoft Team Foundation Server’ as Source Control provider in Source Control Wizard.



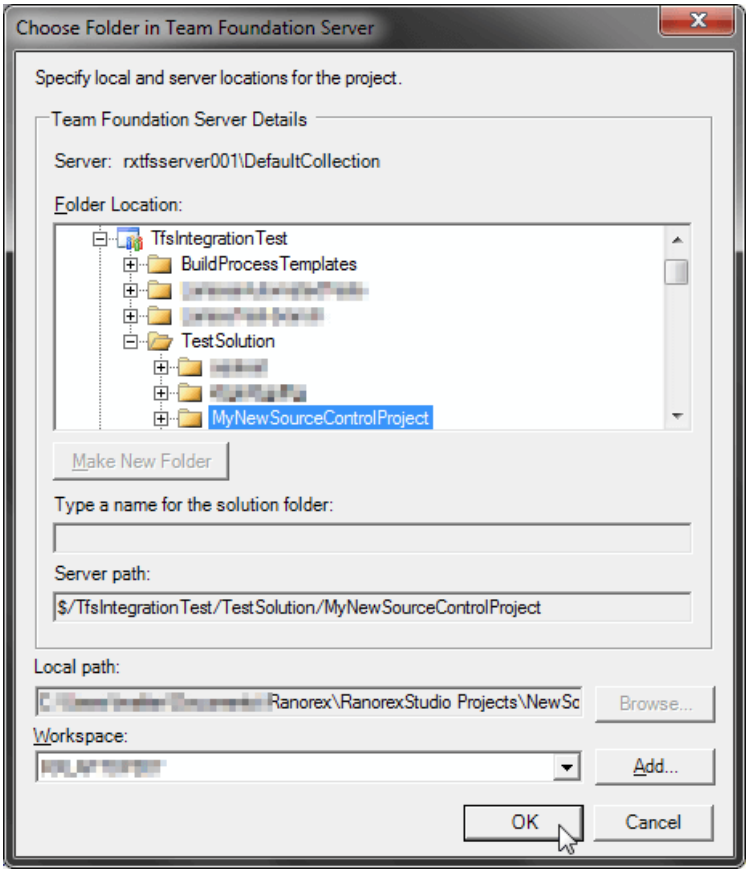
3 Specify the local check out Directory



4 Select your Team Foundation Server.



5 Choose the folder in the repository you want to check out from server.



6 The chosen solution will be checked out.

Icon overlays in Projects View

Overlay icons are added to the items in the Projects view in Ranorex Studio as the solution is under source control.

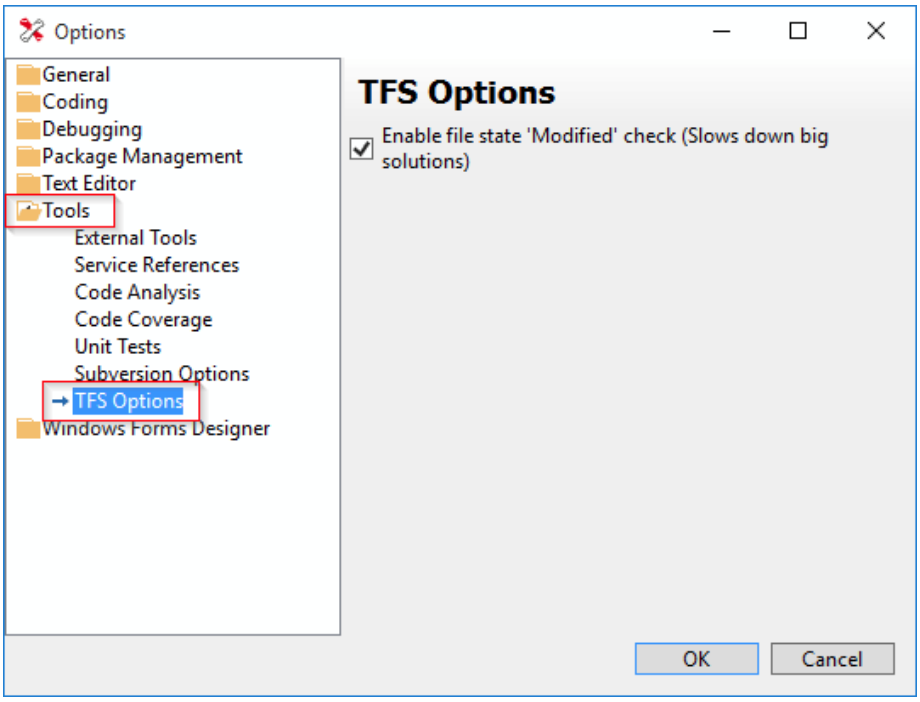
Icon overlays	
Normal	Not locally modified, no changes waiting for commit.
Conflicted	Indicates a conflict.
Modified	Modified, changes are waiting to be committed.
Added	Marked for addition, waiting to be committed.

Options

In Ranorex Studio Options you can find options related to TFVC.

Open main menu item ‘Tools’ and choose ‘Options…’. In the following dialog select folder ‘Tools’ and sub-item ‘TFS Options’. By default, all checkboxes are checked.

If you are working with TFVC and running into performance problems with big solutions, uncheck ‘Enable file state ‘Modified’ check’.



Ranorex Magic Merger

Ranorex Magic Merger makes it easy to **auto-merge XML-based Ranorex files** under source control and **resolve manual conflicts on the fly**.

It handles the following file types:

- Ranorex test suite files .rxtst
- Ranorex repository files .rxrep
- Project files .csproj or .vbproj

Attention

When a repository is merged, this also affects the **automatically generated code files** (.cs or .vb) associated with the repository. These files **must be regenerated after the merge** process for everything to work correctly. To do so, simply **rebuild your solution in Ranorex Studio** after the merge.

Magic Merger will display a message with instructions if automatically generated code files were affected.

Setting up Ranorex Magic Merger

Ranorex Magic Merger is designed to be **integrated into your source control system as the default merging tool** for the above-mentioned file types. The setup process is different for the three source control systems supported by Ranorex.

Git

When you open a solution that's under Git source control or add one to Git, Ranorex Magic Merger will show a dialog to confirm that you want to use it as the default merging tool for the listed Ranorex file types. If you select **Yes**, everything will be **set up automatically** and you're good to go. The dialog will be shown for each Git repository, meaning the merge tool is not set as default globally.

Note

When you select **Don't use Ranorex Magic Merger** and check **Do not show again**, Ranorex Magic Merger will not be used for any Git repositories except those that are already using it. The dialog will not appear again. Get it back by using the **Restore ignored dialogs** option in Settings.

SVN

You will need to **manually** set Ranorex Magic Merger as the default merging tool for the listed Ranorex file types in SVN. For TortoiseSVN, open its settings and go to **External Programs > Merge Tools > Advanced** and add all three Ranorex file types along with the following

command line (change path if required): C:\Program Files (x86)\Ranorex 8.2\Bin\Ranorex.MagicMerger.exe -base:%base -mine:%mine -theirs:%theirs -output:%merged

TFVS

You will need to **manually** set Ranorex Magic Merger as the default merging tool for the listed Ranorex file type in TFVS. Follow this description and add all three Ranorex file types along with the following command line (change path if required): C:\Program Files (x86)\Ranorex 8.2\Bin\Ranorex.MagicMerger.exe -base:%3 -mine:%1 -theirs:%2 -output:%4

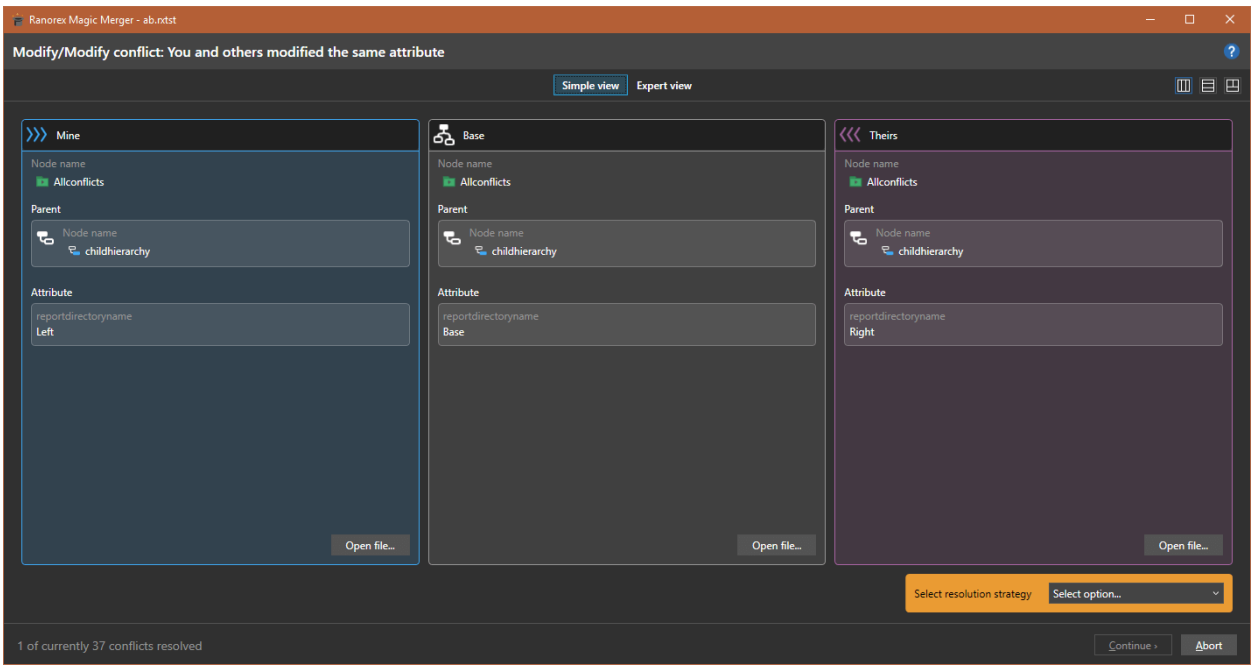
Local use

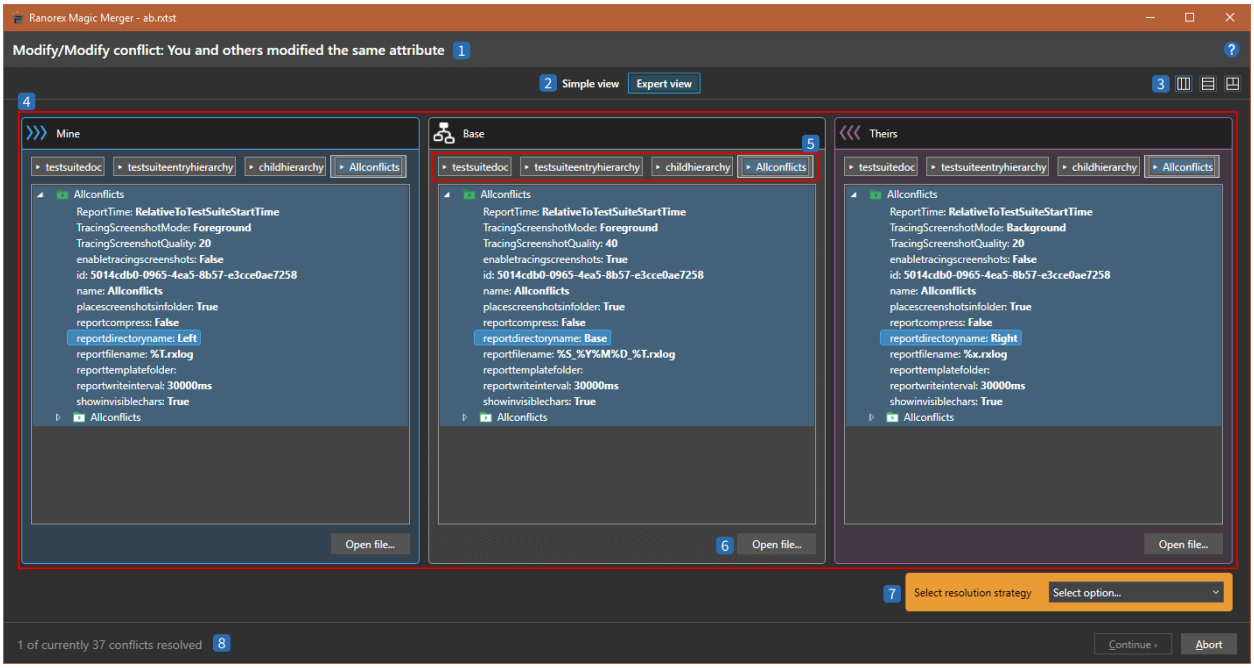
In some edge cases you may want to use Magic Merger locally. You can do so by running the executable from the command line with the required parameters (change path if required): C:\Program Files (x86)\Ranorex 8.2\Bin\Ranorex.MagicMerger.exe -base: -mine: -theirs: -output: The output file is generated automatically as a result of the merge process.

Merging

In the majority of cases, most of the merging will occur automatically, followed by a screen notifying you that the merge was completed successfully. However, as with all merging tools, sometimes user intervention will be required.

In this case, the manual merge screen will appear:





1. The type of conflict and a short explanation
2. Switch between **Simple** and **Expert view**. Expert view provides a more detailed tree structure.
3. Choose one of three layouts for the current view
4. The three sections representing the different file versions (Expert view only)
5. Select one of the tabs to view the hierarchy from this node onwards. The node containing the conflict is marked in blue.
6. Click to open the file in Ranorex Test Suite Runner
7. Resolution strategy selector
8. Number of conflicts resolved and pending. Note that the total number of conflicts may increase or decrease depending on the resolution strategy picked.

To resolve a conflict, simply take a look at the changes, choose your desired resolution strategy, and then click Continue. Once you've resolved all manual conflicts, a success message will appear and the merge is complete.

Azure DevOps Integration

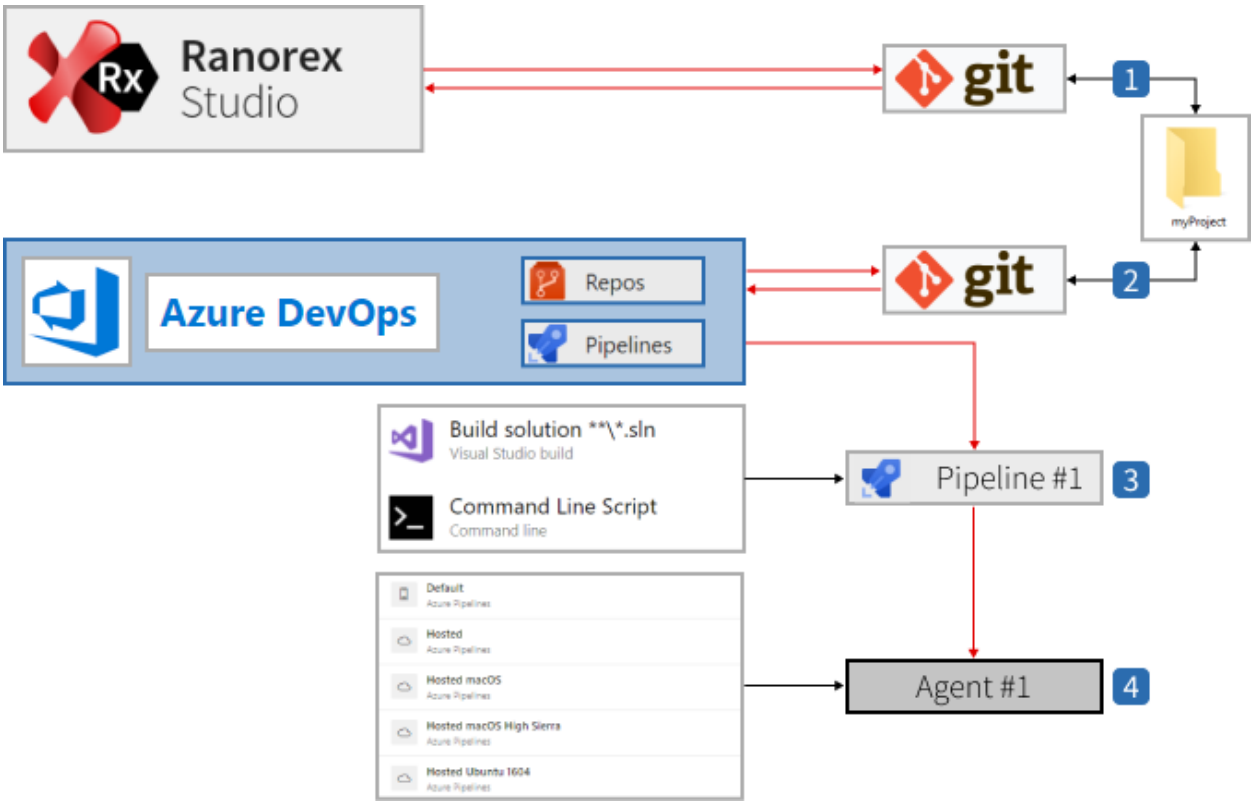
Microsoft Azure DevOps (abbreviated as ADO) is a cloud-based continuous-integration software, previously known as Visual Studio Team Services. It provides all the necessary functionality for managing software development projects.

For more details on Azure DevOps, go to https://azure.microsoft.com.

In this chapter, you'll find out how to use the Azure DevOps integration, i.e. running your Ranorex Studio tests as part of a build pipeline in Azure DevOps.

Overview

The integration relies on various systems. The image below illustrates how they work together:



1. The Ranorex Studio solution is stored in a folder which acts as a Git repository (the repository folder), and is therefore under Git version control.
2. The Azure DevOps project connects to the Ranorex Studio solution in the repository folder through the **Repos** functionality and also uses Git for version control.

3 The Azure DevOps project can contain one or more **pipelines**. These provide the CI functionality, i.e. building and executing a Ranorex Studio solution.

4 Finally, the test is executed on one or more Azure Pipelines agents.

Software requirements

You will need to have the following software installed to use the Azure DevOps integration.

Ranorex Studio

For creating and maintaining tests, you will need a full installation of Ranorex Studio and a Premium license. For only running a test on a machine with an Azure Pipelines agent, you will need a Runtime license.

For more information about the available license models, refer to https://www.ranorex.com/prices/.

Azure DevOps account

You will need an Azure DevOps account. A free account is enough to use the integration, but depending on your requirements, you may need to upgrade to a paid account. Go to https://azure.microsoft.com for more information about functionality and pricing.

Git

The integration relies on Git for versioning and providing the Ranorex Studio solution to the Azure DevOps project. Git is free under the GNU GPL. You can get it from https://git-scm.com/.

Microsoft Visual Studio

Pipelines in Azure DevOps rely on Microsoft Visual Studio to build Ranorex Studio solutions. You therefore need to have Microsoft Visual Studio installed. The free version is enough to use the integration, but depending on your requirements, you may need to upgrade to a paid version.

Go to https://visualstudio.microsoft.com/ for more information about different versions.

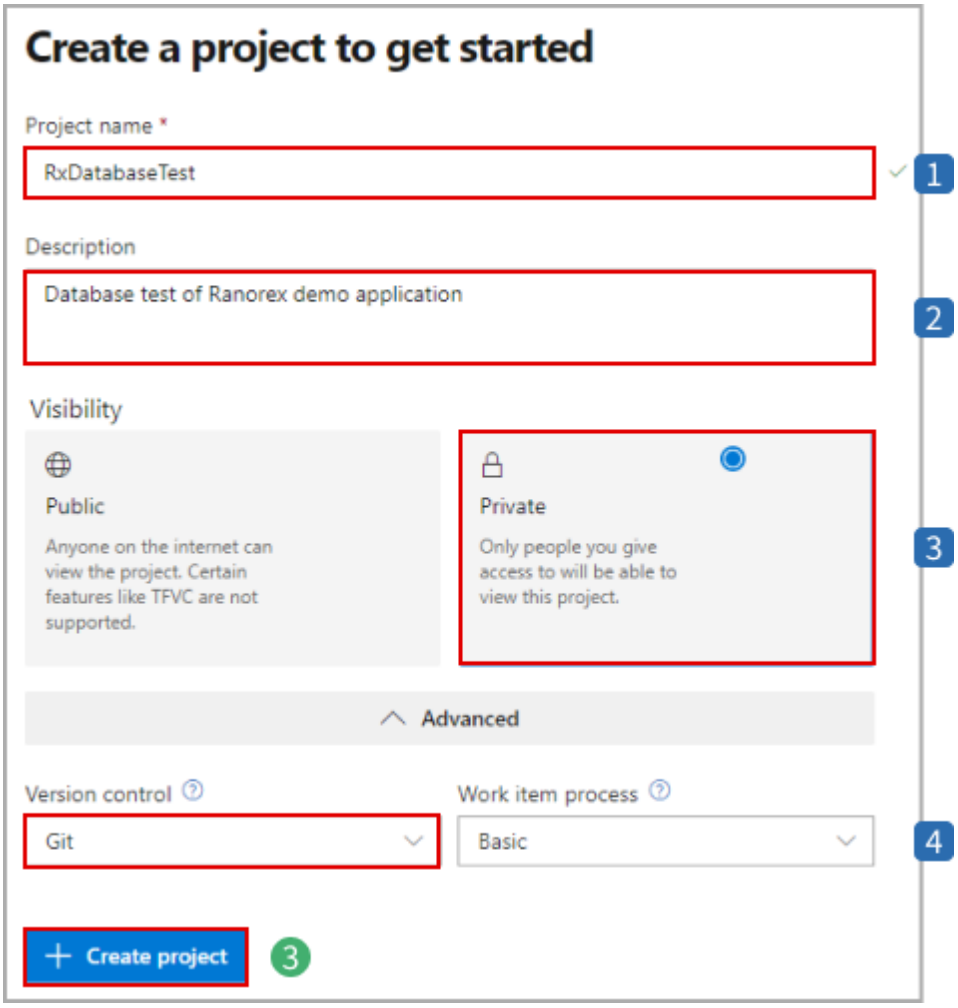
Create ADO project and Git repo

On this page, you'll find out how to create a project in Azure DevOps and then a folder that will serve as the Git repository. We'll also cover adding a custom .gitignore file to the repository folder.

Create a project in Azure DevOps

First, we'll create the Azure DevOps project.

1. **Log in** to your Azure DevOps account.
2. **Create** a new project and **enter** the details below.
3. **Click Create project**.



1. Project name
2. Project description (optional)
3. Visibility set to private
4. Git as version control

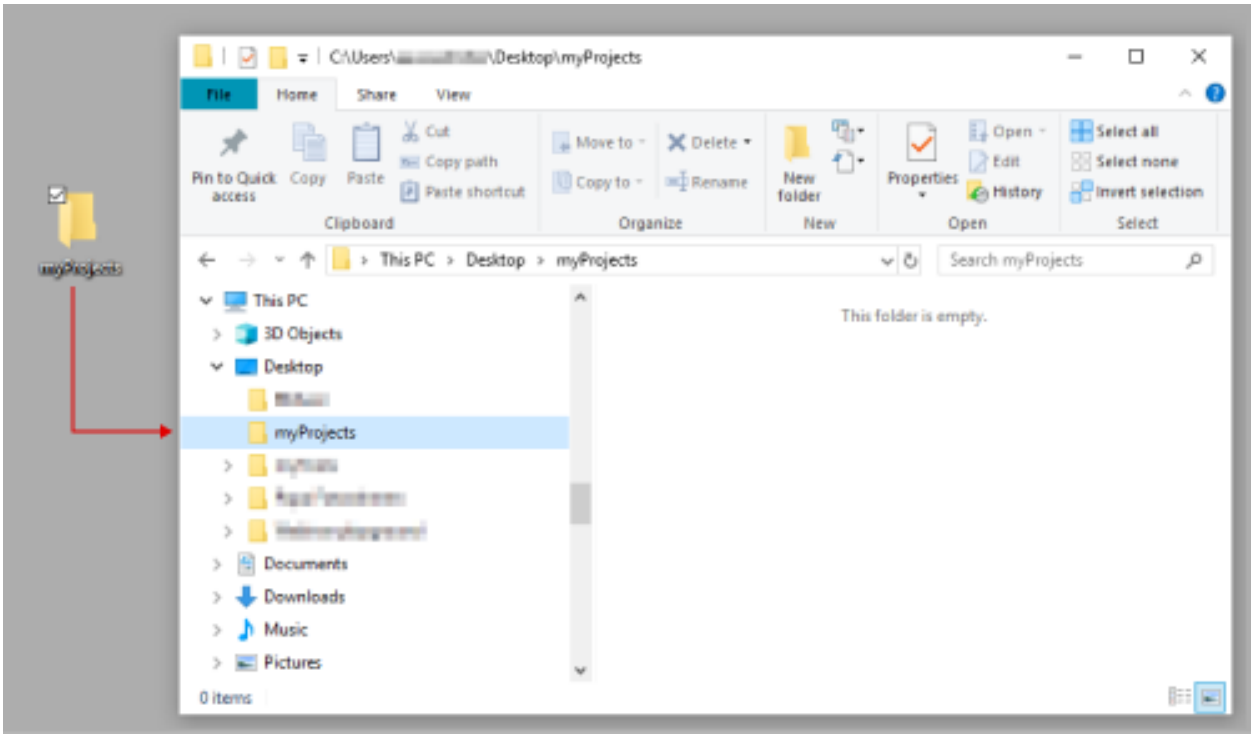
Create the repository folder

For the integration to work, the Azure DevOps project must have access to the Ranorex Studio solution. This is accomplished through a folder which acts as a Git repository. This repository folder can be local, e.g. on the machine with the Ranorex Studio installation, or in a network.

The following instructions assume a local repository folder, but they also work for a network folder with some adjustments.

Create the folder

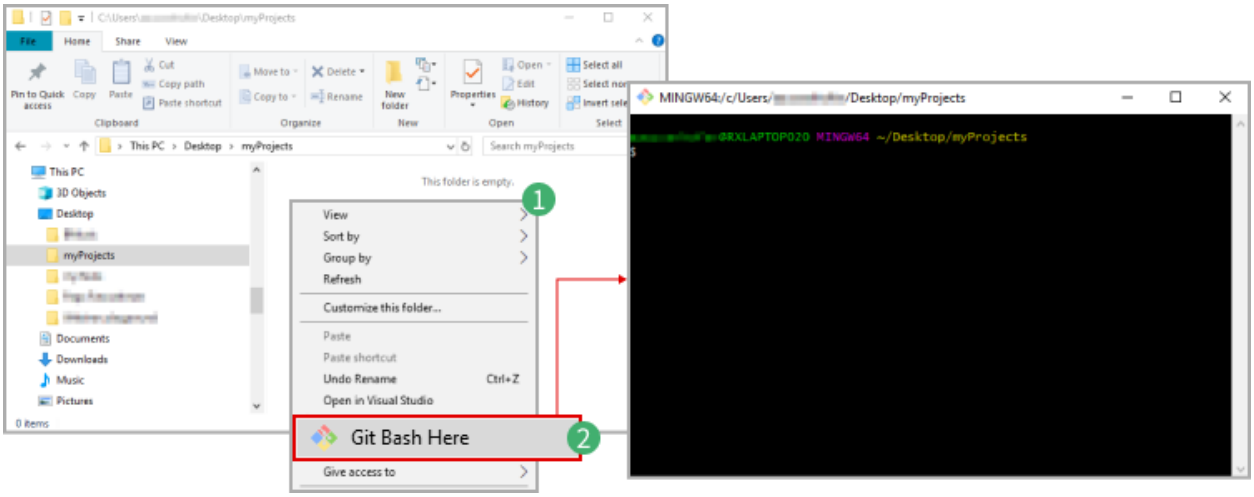
1. **Create** a folder called **myProjects** and **open** it.



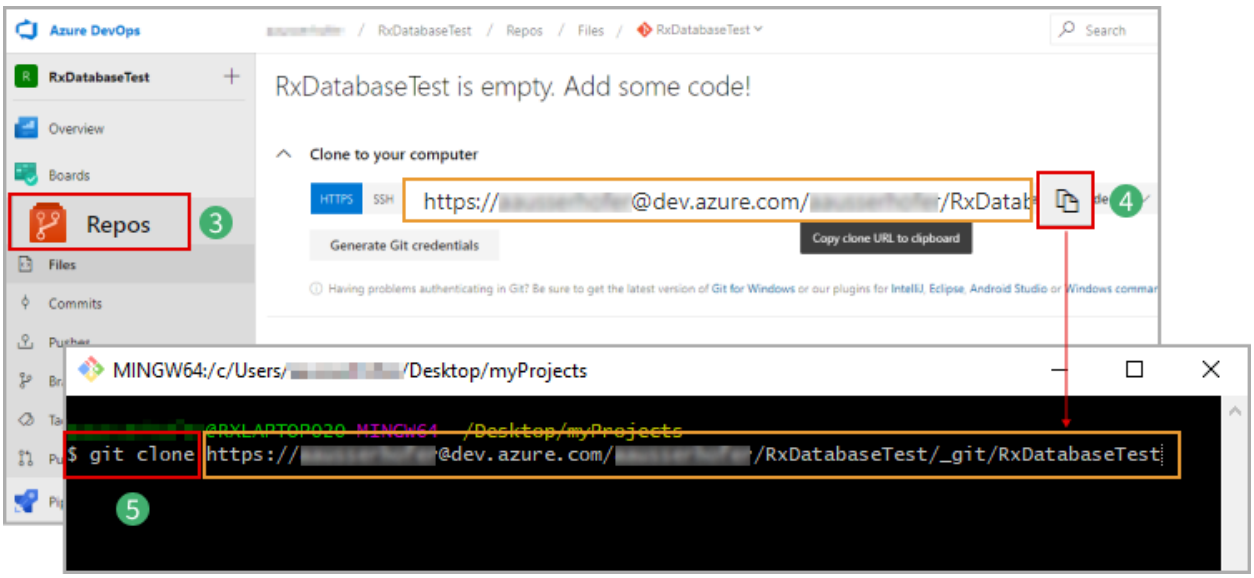
Clone the Azure DevOps project folder

We now clone the Azure DevOps project folder to the repository folder. To do so:

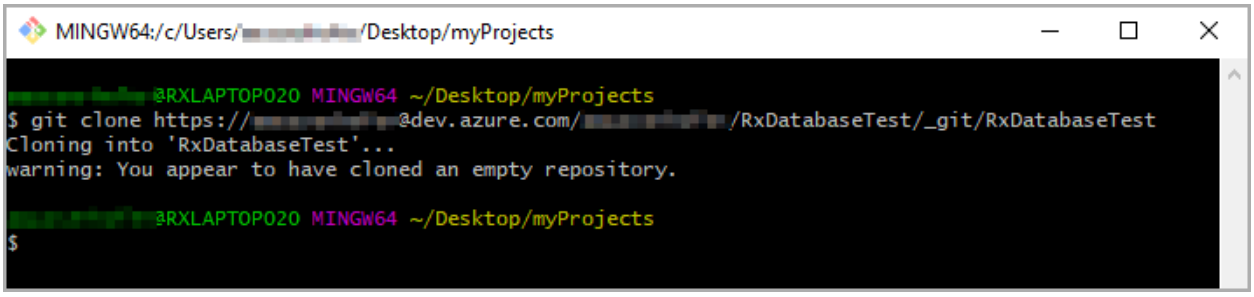
1. **Right-click** a free space in the repository folder.
2. **Click Git Bash Here** to open the command line.



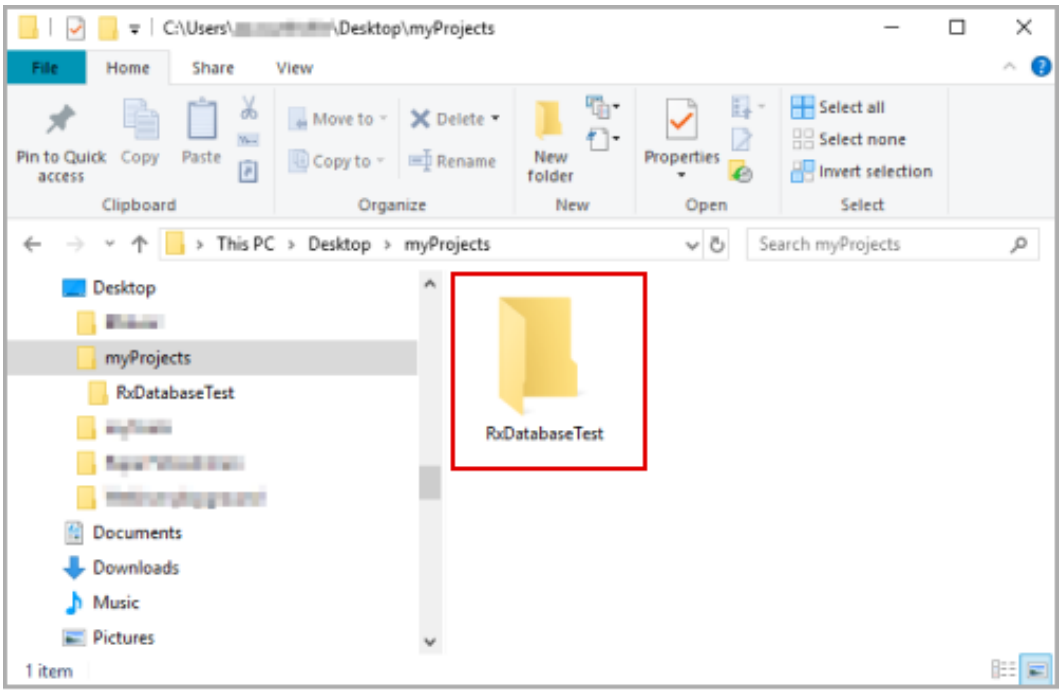
1. In the Azure DevOps project, **click Repos**.
2. Under **Clone to your computer**, **copy** the URL of the project folder.
3. In the command line, **enter git clone** and **paste** the project folder URL.



4. **Press Enter** to execute the clone command.



The repository folder now contains the cloned Azure DevOps project folder.

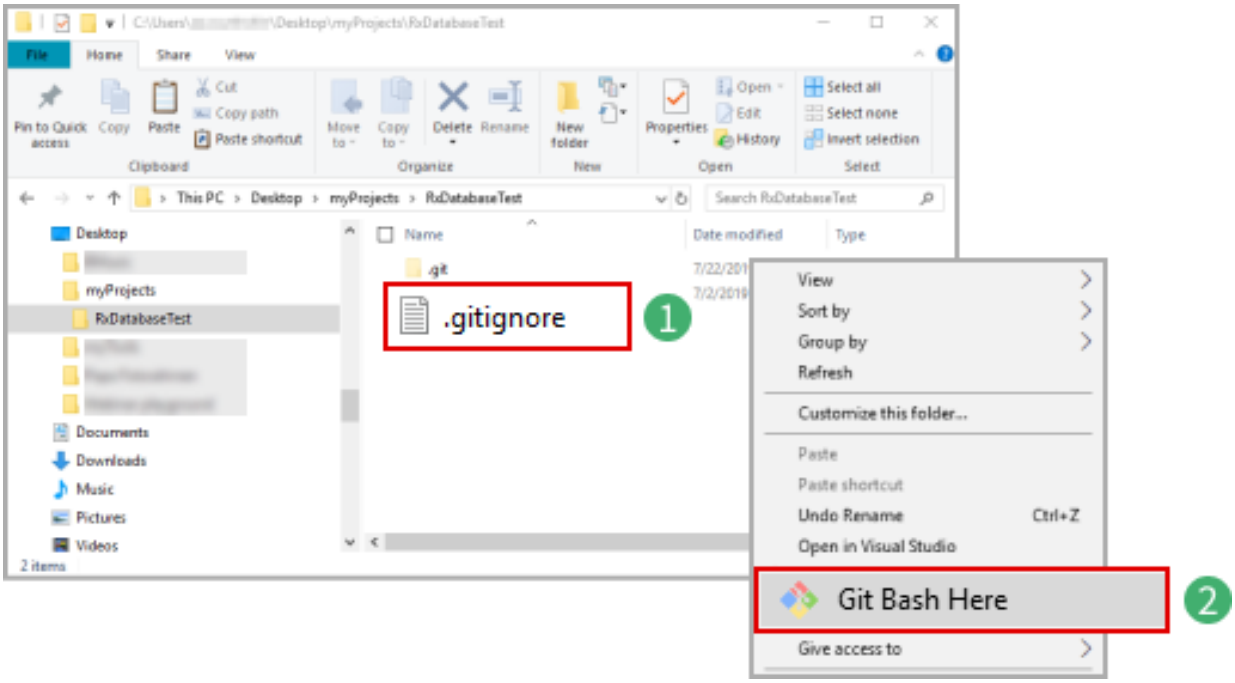


Add .gitignore file

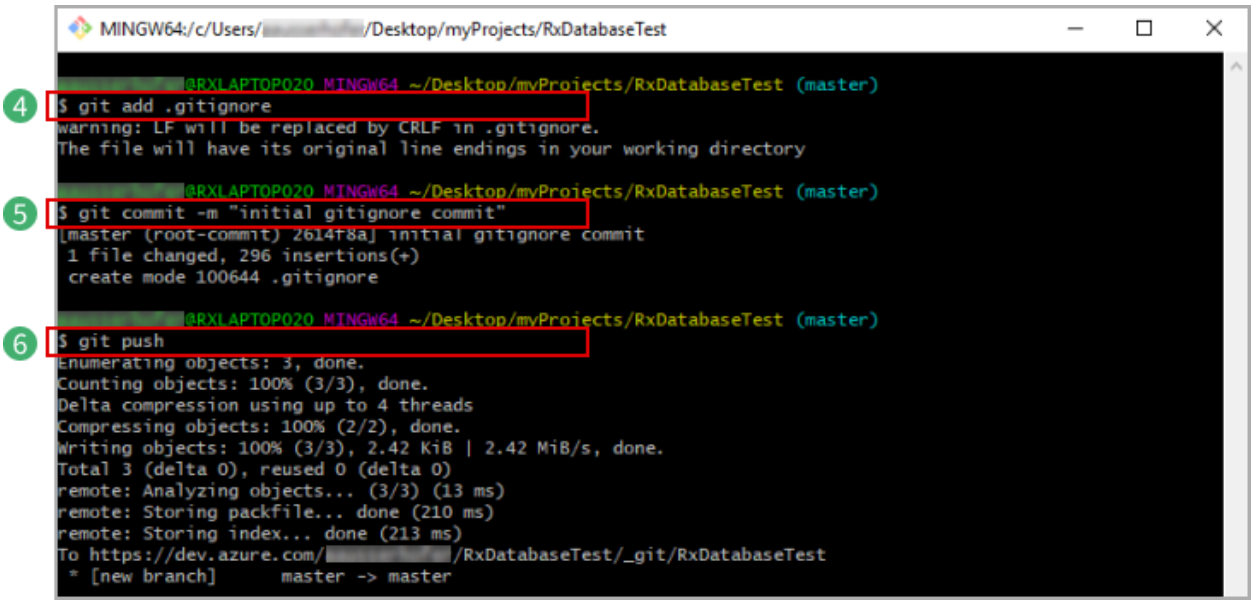
The .gitignore file in a Git repository tells Git to exclude certain files from commits. If you don't include this file, the integration will still work fine, but unnecessary files will be included in your commits.

> **Note**
>
> We've prepared a custom .gitignore file for you to download and use in your project. It excludes all the files in a Ranorex Studio solution not needed by Azure DevOps from commits.

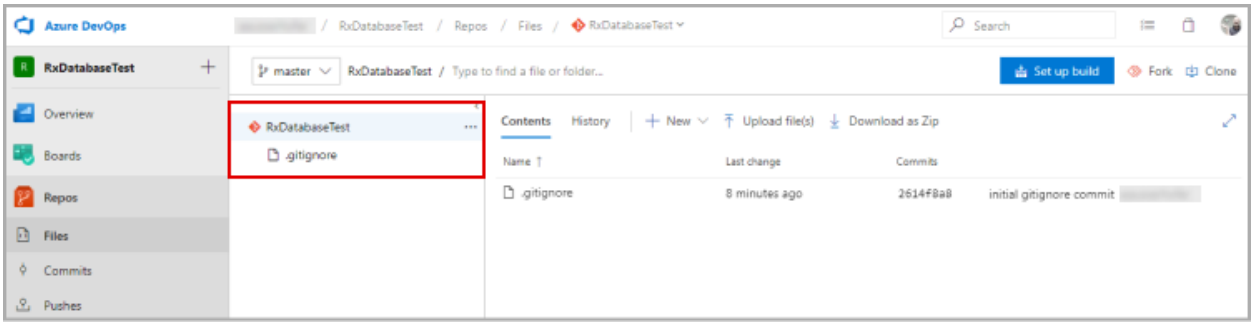
1. **Copy** the .gitignore file to the cloned Azure DevOps project folder.
2. **Right-click** in a free space in the project folder and **click Git Bash here**.



3. **Enter** the following commands, confirming each one with **Enter**:
4. **git add .gitignore** to add the file to the repository.
5. **git commit – m "initial gitignore commit"** to execute an initial commit.
6. **git push** to push the commit.



When you refresh the Repos view in Azure DevOps, you will see the. gitignore file in the projects folder.



Add Ranorex Studio solution to DevOps project

For the integration to work, the Azure DevOps project must have access to the Ranorex Studio solution. This is done through the Git-versioned repository folder we created in the previous step.

On this page, you'll find out how to add a Ranorex Studio solution to the repository folder and ensure that it's under Git version control there.

Add solution to repository folder

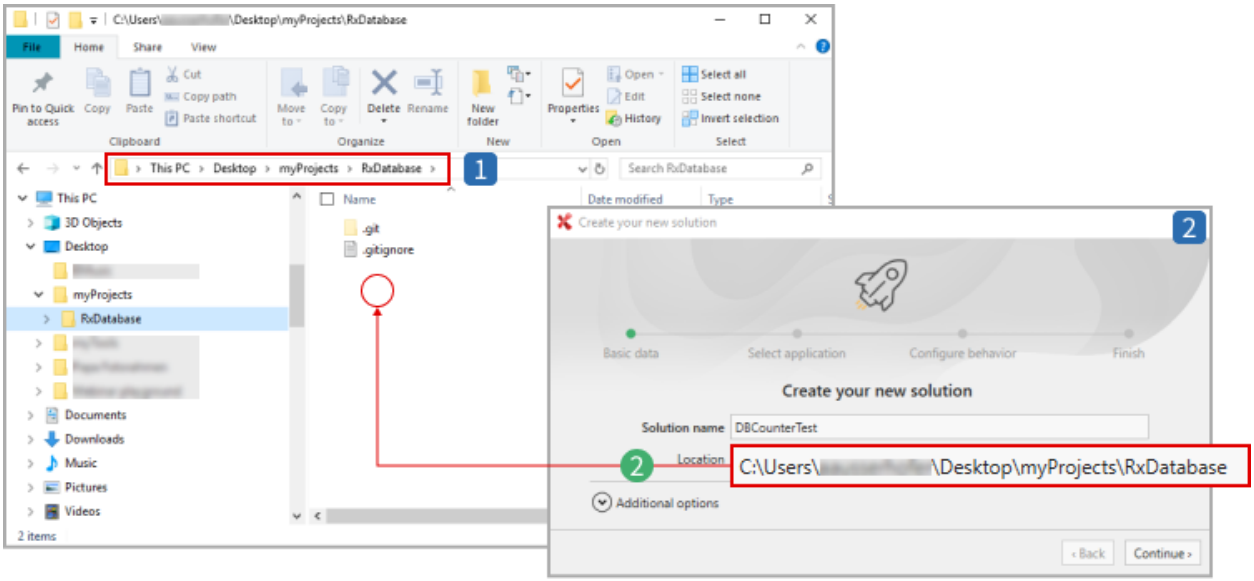
First, you need to place the Ranorex Studio solution you want to use in Azure DevOps in the repository folder.

For an existing solution

1. **Copy** the entire solution folder to the repository folder.

When creating a new solution

1. **Create** a new Ranorex Studio solution
2. **Specify** the repository folder as the location.



1. The repository folder.
2. Field to specify the solution's save location in the solution wizard.

Put solution under Git version control

For the integration to work, you also need to put the Ranorex Studio solution under Git version control.

> **Reference**
>
> For instructions on how to set up the Git integration in Ranorex Studio and put a Ranorex Studio solution under Git version control, please refer to
>
> Interfaces and connectivity > Source control & revision control > ⇢ Git

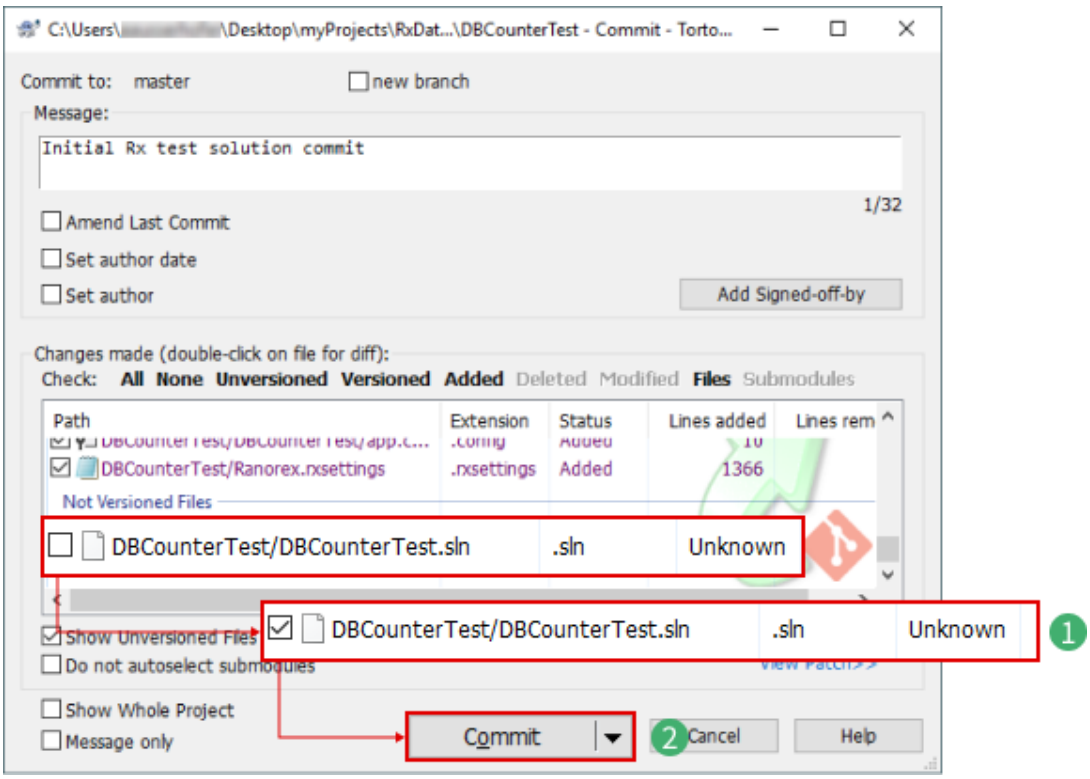
Attention

For the initial commit of the solution, you need to **manually include the Visual Studio solution file** with the file ending .sln.

This is because Ranorex Studio normally only requires the solution file ending in .rxsln for the build. Therefore, the .sln file isn't checked for commit by default. Azure DevOps, however, uses Visual Studio for the build, which requires the .sln file.

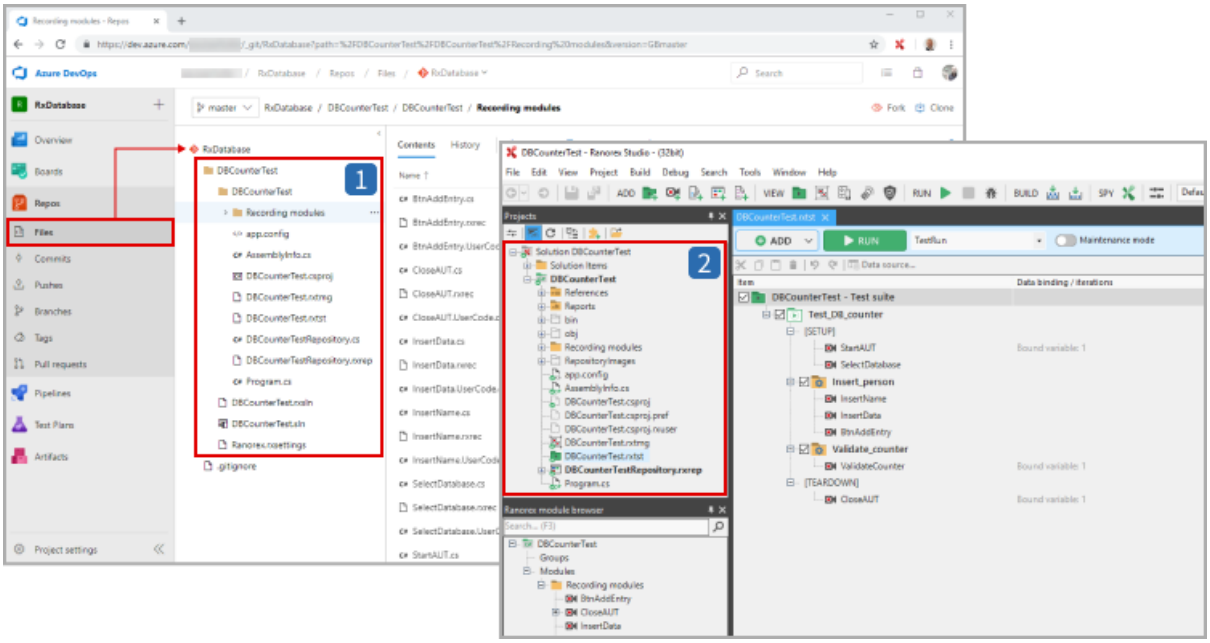
To manually include the .sln file in the inital commit:

1. In the Git commit dialog, **select** the solution file with the ending **.sln**.
2. **Click Commit**.



View the solution in Azure DevOps

Once the Ranorex Studio solution is stored in the repository folder and versioned with Git, you can see it in your project in Azure DevOps.



1. Ranorex Studio solution files in the Azure DevOps project.
2. Ranorex Studio solution files in the projects view in Ranorex Studio.

Note

In the image, the projects view shows all solution files, including temporary ones and report files. The Azure DevOps project, however, only shows the files required to build the solution. Everything else, like the subfolders \bin\, \References\, or \Reports\, was filtered by our >.gitignore list, which ensures that these are not included in the Git version control, and therefore not in the Azure DevOps project.

Set up an Azure Pipelines agent

To execute an Azure DevOps pipeline, you need an Azure Pipelines agent. This is a standalone program that's similar to a Ranorex Agent. You install it on a physical or virtual machine, connect it to Azure DevOps, and can then pass pipelines to the agent for execution.

On this page, you'll find out how to set up and start an agent. We'll only cover what's relevant for executing a Ranorex Studio solution through an Azure DevOps pipeline. For all other topics, please refer to the official Azure DevOps documentation.

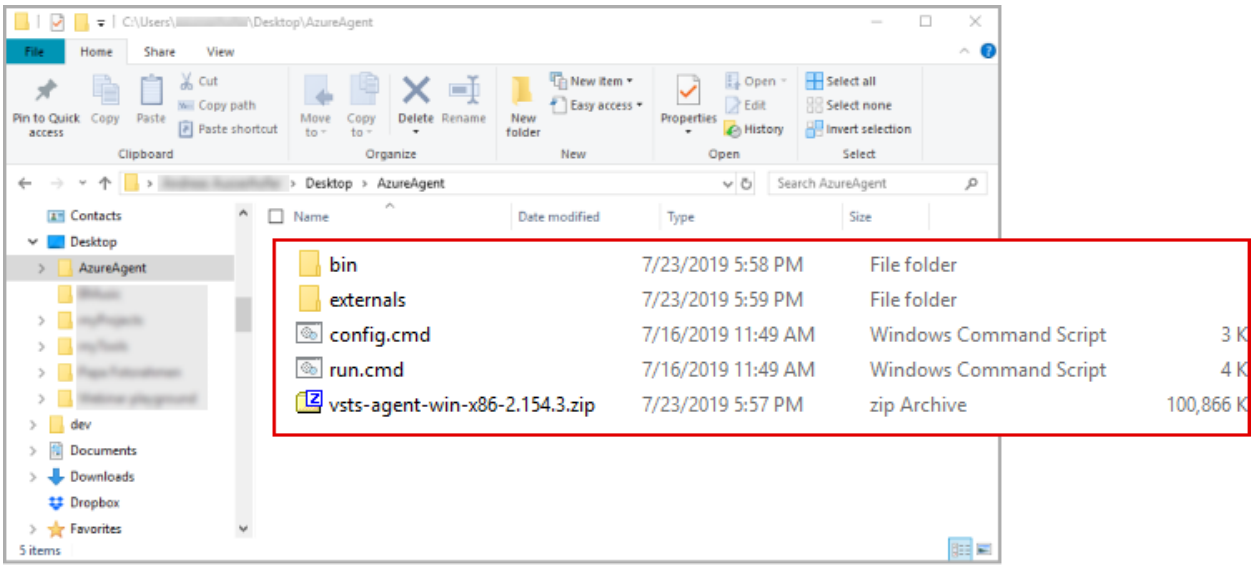
Download an agent

To download an agent:

1. In the bottom left of your Azure DevOps project, **click Project settings**.
2. **Click Agent pools**.
3. **Select** an available agent pool. You can select from a hosted pool or from a local pool (default).

> **Note**
>
> The following instructions assume a default local pool.

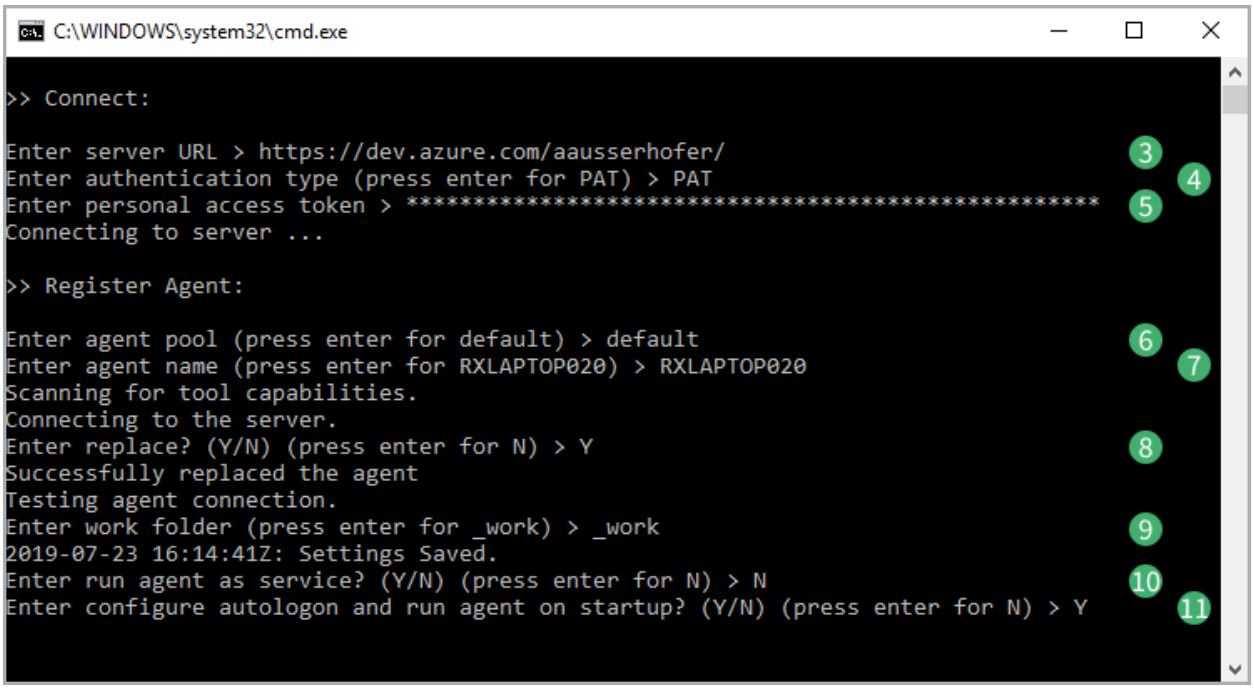
4. In the agent view that appears, **click New agent**.
5. **Select** Windows as OS, your OS's architecture (x64 or x86), and **click Download**.
6. **Unpack** the downloaded .zip file to a folder of your choice.



Configure the agent

To configure the downloaded agent:

1. **Double-click** the file **config.cmd**.
2. **Configure** the agent as follows:



This is a basic userguide text with all paddings and margins set correctly. Just use all text styles H2, H3 etc… as you want. If you need to define Key strokes like *CTRL* do it like this here.

3. **Paste** the Azure-DevOps project URL from your browser's address bar.
4. **Press Enter** to use PAT as authentication type.
5. In the Security menu of your Azure DevOps account, **create** a full-access PAT and **paste** it here.

> **Attention**
>
> The generated PAT is visible only once and can't be recreated. **Copy and paste it immediately after generating it.**

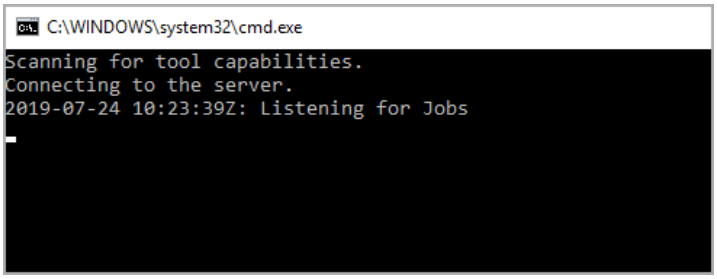
6. **Press Enter** to select the default local pool.
7. **Press Enter** to confirm the agent's name or change it.
8. **Press Y** if you have previously had an agent installed on this machine; **press Enter** if not.
9. **Press Enter** to confirm the work folder or change it.
10. **Press N**. The agent must not be run as a service.
11. **Press Y**.

Start the agent

To start the agent:

1. In the folder to which you unpacked the agent files, **double-click run.cmd**.

The agent starts, connects to the server, and idles until it receives a pipeline to execute.



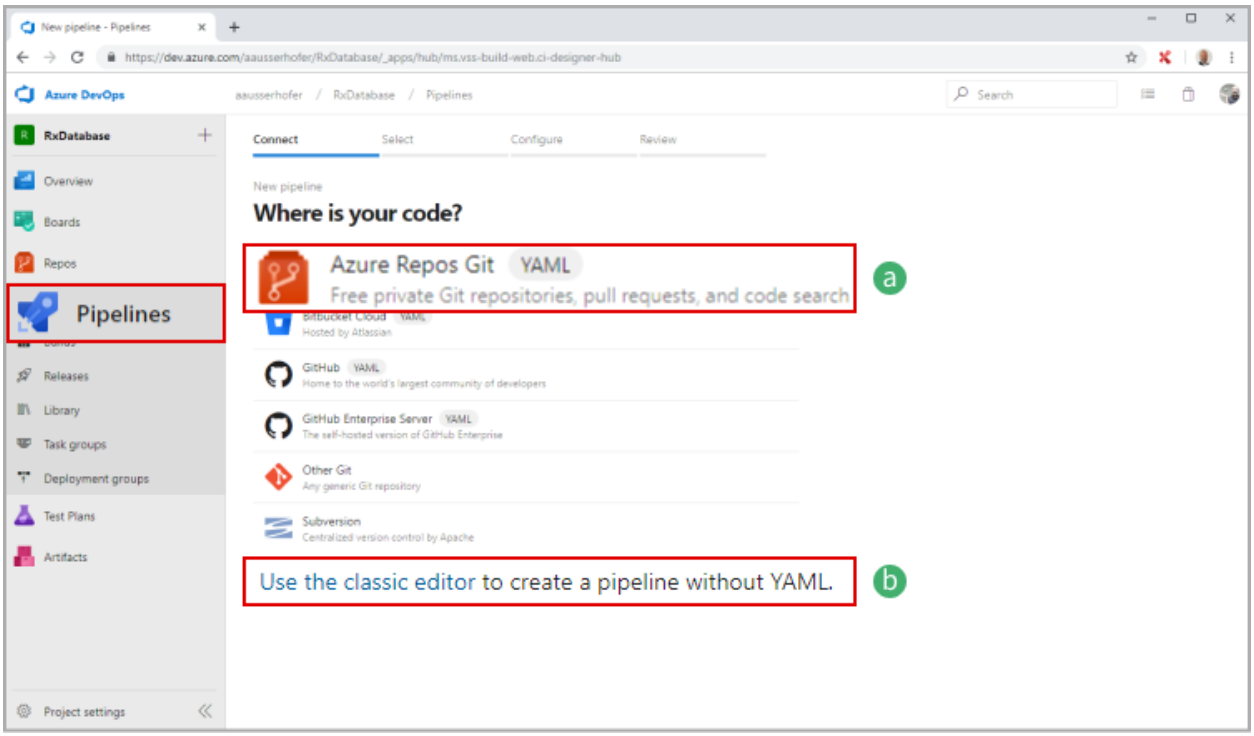
Create and execute a pipeline

Pipelines in Azure DevOps are how you build, test, and deploy your code. On this page, you'll find out how to create, configure, and execute pipelines in Azure DevOps to run your Ranorex Studio solution. For more details about pipelines, please refer to the official Azure Pipelines documentation.

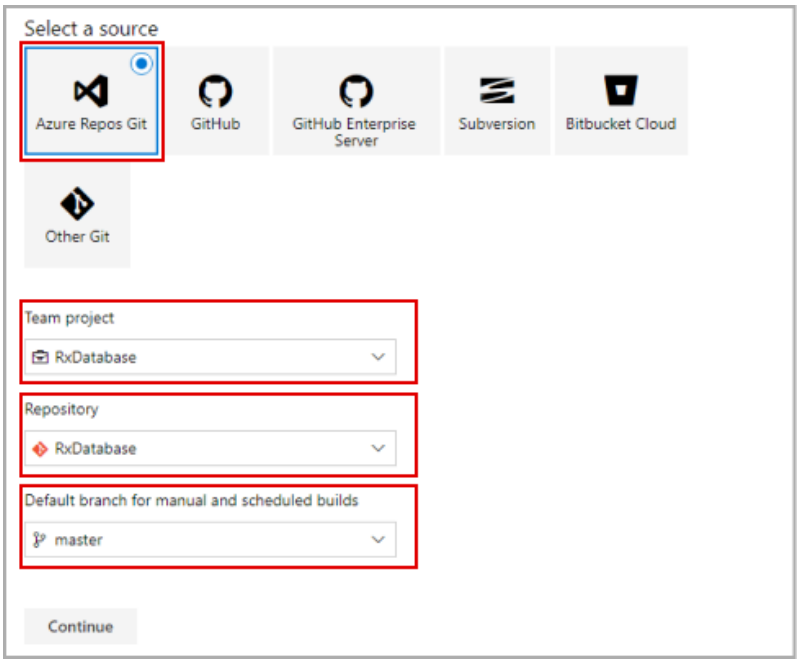
Create a pipeline

To create a new pipeline for your Azure DevOps project:

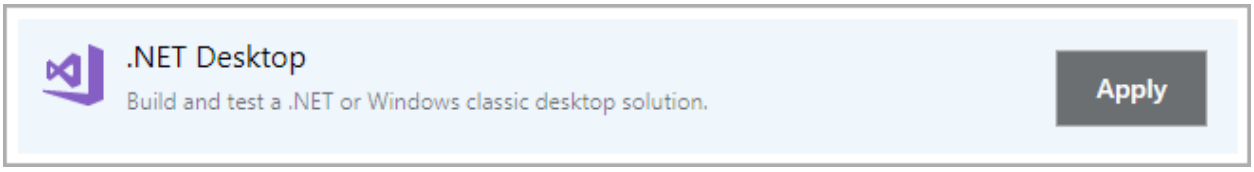
1. In your opened project, **click Pipelines** in the register on the left.
2. **Click New pipeline**.
3. Depending on whether you want to specify a pipeline using YAML or on your own:
   a. **Click Azure Repos Git (YAML)** (recommended for experts, not covered here).
   b. **Click Use the classic editor**.



4 **Configure** as shown below and **click Continue**.

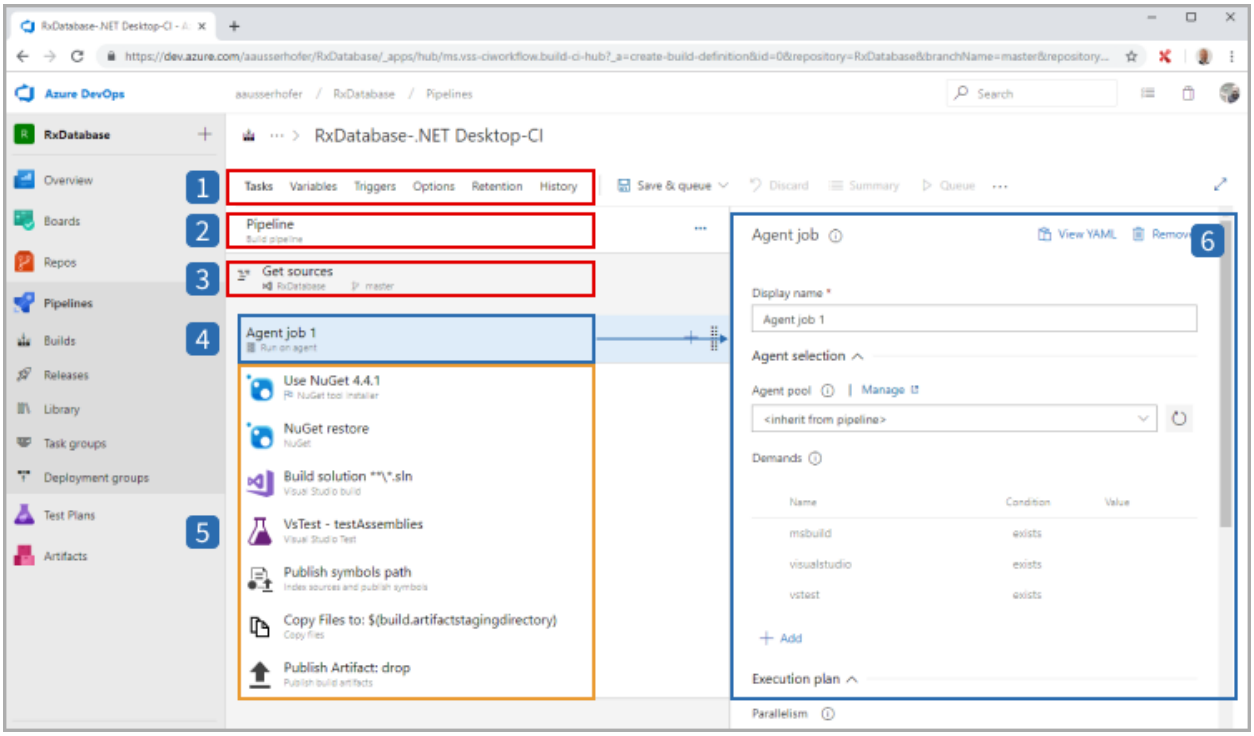


5 In the next step, **click Apply** for **.NET Desktop**. This is the framework Ranorex Studio solutions are based on.



Pipeline overview

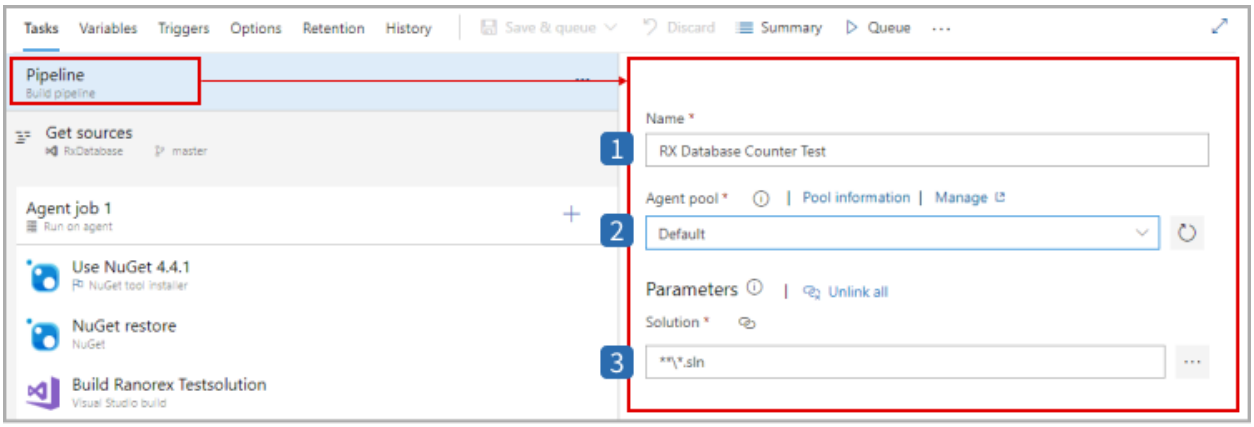
The pipeline overview shows the various components a pipeline consists of and offers many possibilities for configuration. Here we only explain what's relevant for the Ranorex Studio integration.



1. Configuration categories of a pipeline. In the image, **Tasks** is selected.
2. Basic pipeline configuration. Here you can give the pipeline a name, set the agent pool, and set optional parameters.
3. Here you can configure the sources (i.e. repositories) the pipeline uses. This corresponds to what we did in **Create a pipeline** above.
4. A pipeline always contains at least one agent job. This is the logical set of tasks that the pipeline carries out.
5. An agent job always contains at least one task. Tasks are carried out in the order they appear in under the agent job.
6. The configuration window. Configuring one of the above items brings up this window. In the image, the window for configuring the agent job is shown.

Basic pipeline configuration

1. Under **Tasks**, **click Pipeline** to bring up the basic pipeline configuration, where you can make the following settings:



1. Here you can change the pipeline name.
2. Here you select the agent pool for the pipeline. You can use the default local pool or choose from a range of hosted pools that have various images already set up.
3. Here you can set optional pipeline parameters. The preset parameter in the image builds the executable from the Visual Studio solution file (**.sln** file ending).

Pipeline tasks for Ranorex Studio tests

For Ranorex Studio tests to be executed correctly in an Azure DevOps pipeline, the agent job must contain the tasks listed below.

> **Note**
>
> The .NET pipeline already contains various default tasks. Do not delete them.

> **Hint**
>
> Instead of the following tasks, you can also use the official Ranorex Studio build task, explained below after this topic.

Build the Ranorex Studio solution

The task **Visual Studio build** builds the Ranorex Studio test solution. It is already included by default in the agent job and functional.

Configuration

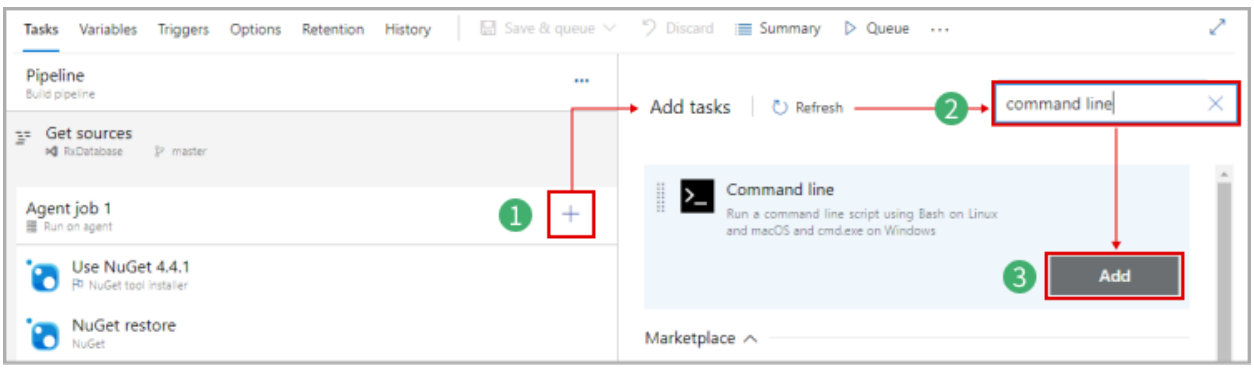
1. **Change** the display name for easier identification.
2. **Change** the platform to **x86**, if required.

Execute the build

To execute the build of the Ranorex Studio solution, you need to add the Command line task to the agent job.

To do so:

1. **Click the +** symbol next to the agent job.
2. In the search bar, **enter command line**.
3. **Click Add**.

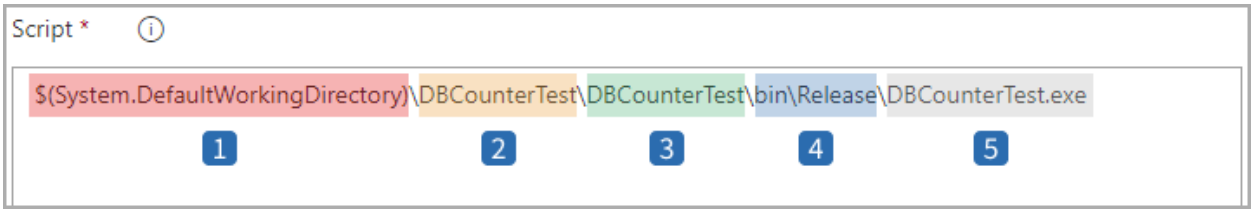


Note

Make sure the **Command line** task comes after the **Visual Studio build** task. Otherwise, execution will fail.

Configuration

4. **Change** the Display name for easier identification.
5. **Replace** the default script with the following:



1. System variable for the current working directory of the Azure DevOps project.
2. Folder for the Ranorex Studio solution.
3. Folder for the Ranorex Studio project in the solution.
4. Default output folder for Visual Studio projects.
5. File name of the executable build. You can add further optional arguments to the .exe.

Reference

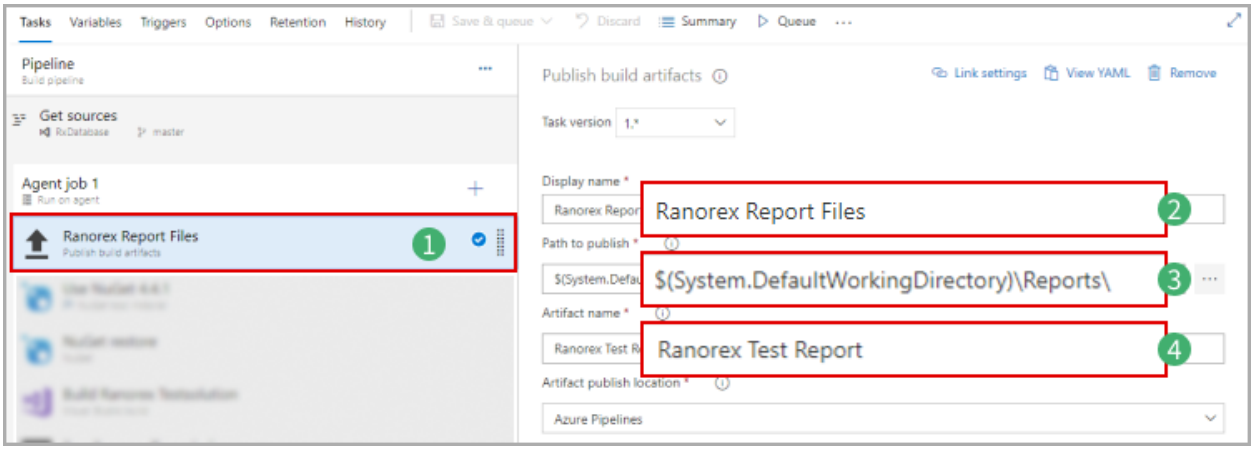
For a full list of command line arguments and how to use them, refer to

Ranorex Studio expert > Runtime and remote execution > ⇢ Command line execution

Reporting

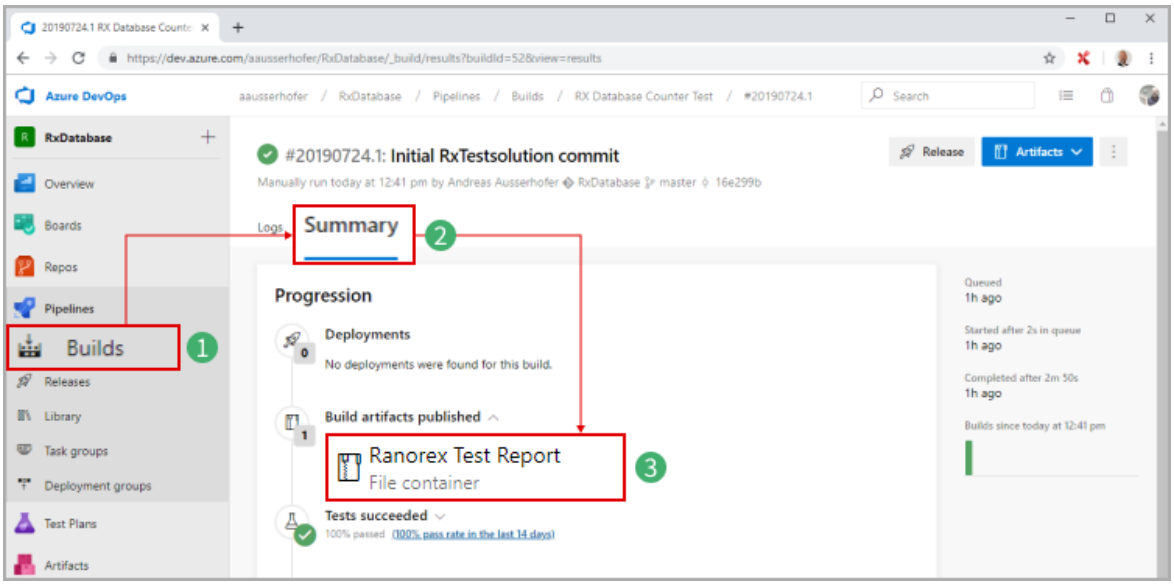
When running a Ranorex Studio test through an Azure DevOps pipeline, the associated Ranorex Studio report is not saved to the working directory of the build by default. To include it, you have to add a Publish build artifacts task in the agent job. Then:

1. **Click** the **Publish build artifacts** task.
2. **Change** the display name, if desired.
3. **Change** the directory as shown in the image.
4. **Change** the artifact name, if desired.



Result

Once a pipeline has finished, you can see the report in the summary and open it from there.



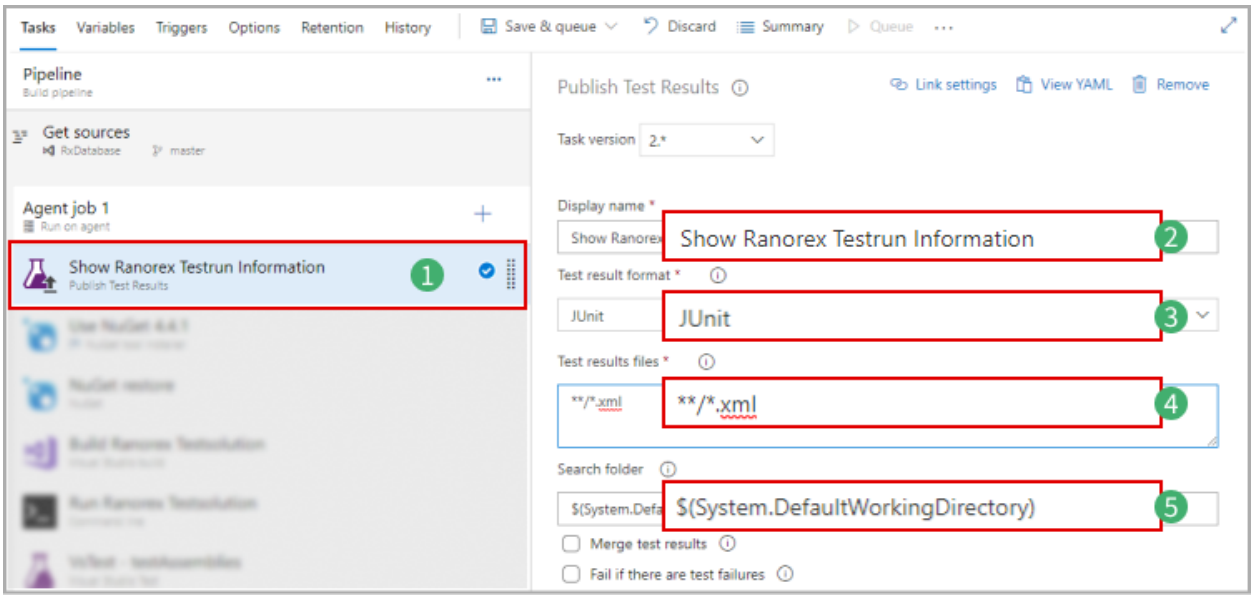
1. **Click Pipelines > Builds**.
2. **Click Summary**.
3. Under **Build artifacts published**, **click** the item. It will have the name you specified in the task.

Show test results in the build results

By default, the results of a Ranorex Studio test are not displayed in Azure DevOps. You can however have them displayed by using the JUnit format of Ranorex Studio reports.

To do so:

1. To the agent job, **add** a **Publish test results** task.
2. **Change** the task's display name, if desired.
3. **Set** the test result format to **JUnit**.
4. As test results files, **specify **/*.xml**.
5. As search folder, **specify** the current working directory.



6 Switch to the Command line task and extend the script as in the image:

1. The executable build of the Ranorex Studio solution.
2. Argument to create the report in the JUnit format.
3. Argument to zip the report.
4. Argument to specify a directory and file name for the zipped report.

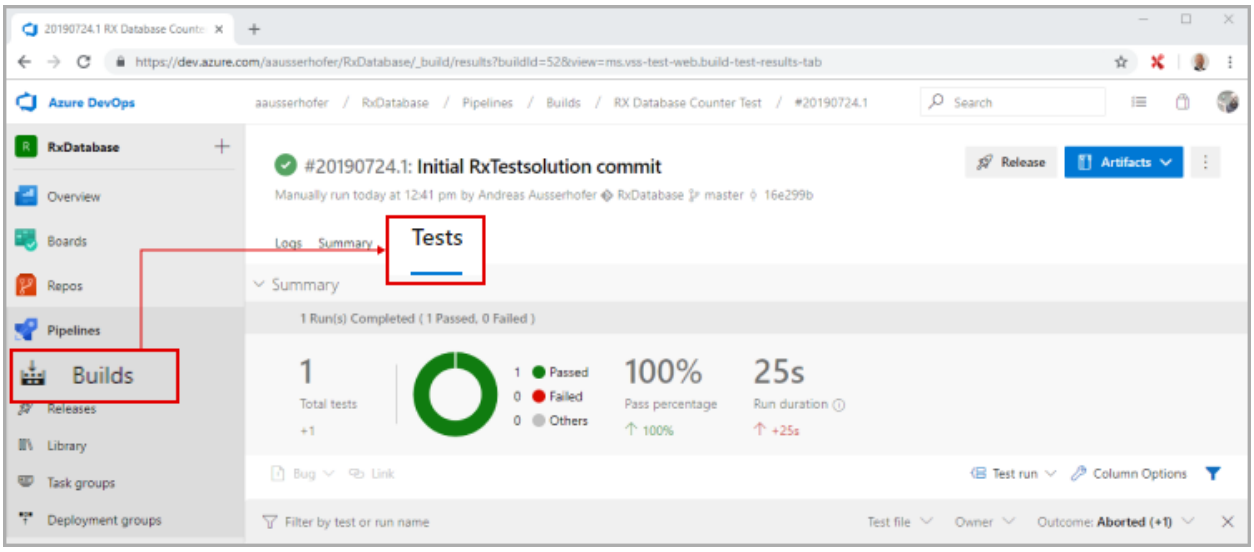
Reference

For a full list of command line arguments and how to use them, refer to

Ranorex Studio expert > Runtime and remote execution > ⇢ Command line execution

Result

Under **Pipelines > Builds > Tests**, you can see various test details after a build has been executed.

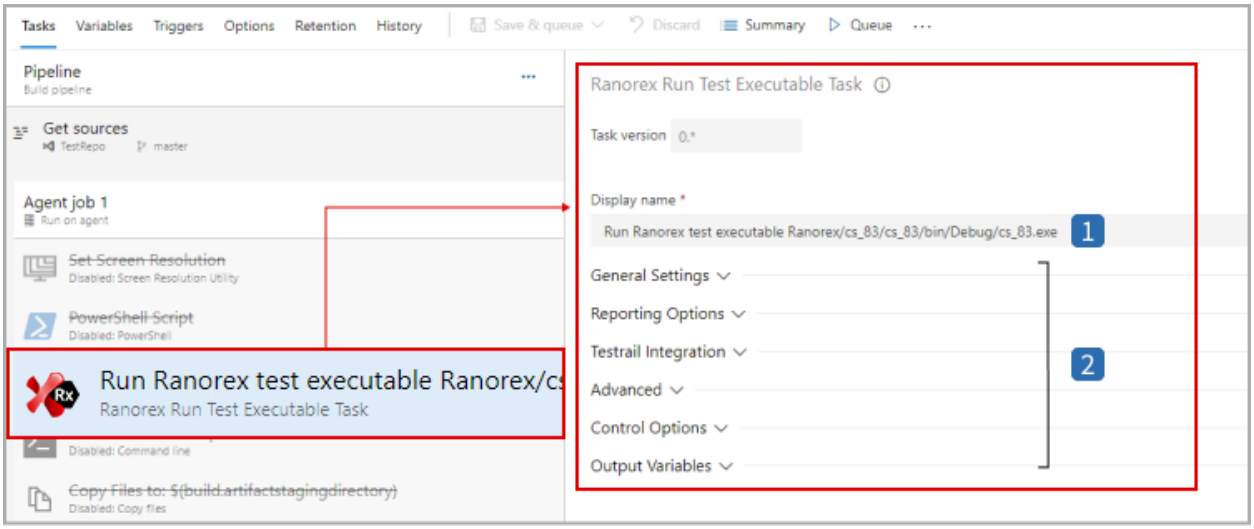


Official Ranorex Studio build task

The official Ranorex Studio build task collects all the tasks described above under Pipeline tasks for Ranorex Studio tests and more in one simple task with a straightforward UI to configure it. You can download it as a free extension from the Azure DevOps Marketplace here.

Simply add the task to a job to get started.

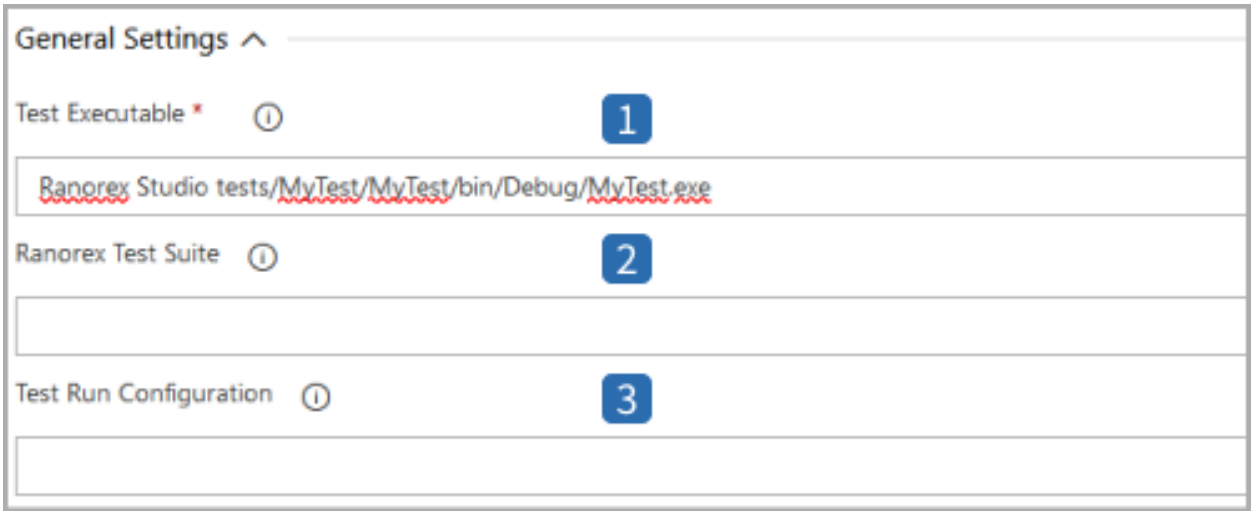
> **Note**
>
> The build task creates a set of command line arguments based on how you configure it. Therefore, it behaves in the same way as running a Ranorex Studio test from a command line interface and the arguments themselves also work the same way.
>
> All command line arguments are explained ⇢ here.



1 Display name of the task.

2 Task settings controlling the integration with Ranorex Studio. We explain these below.

General settings

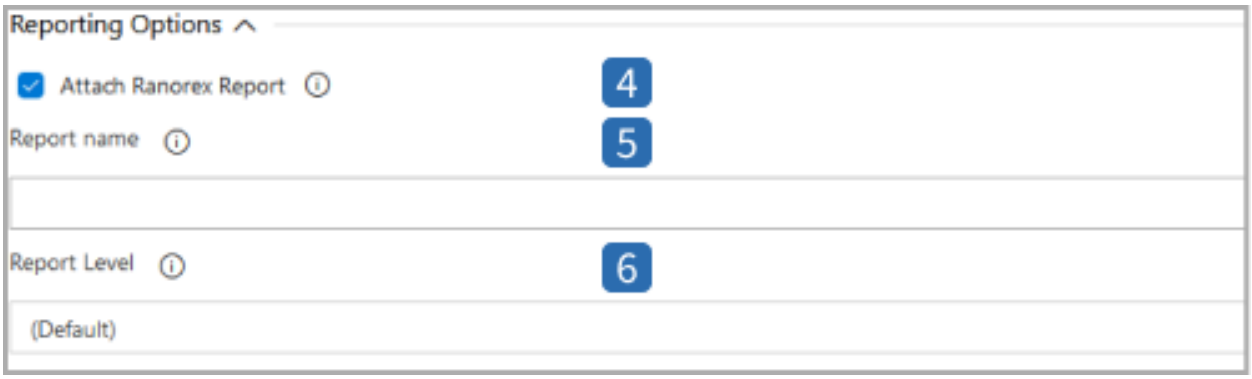


1 The relative path from the repository folder to the test executable in the output folder of your Ranorex Studio solution. Required.

2 The relative path from the repository folder to a specific test suite file (.rxtst) in the output folder of your Ranorex Studio solution. Useful if your test contains several test suites and you only want to run one of them.

3 The run configuration that will be used for the test run. Enter exactly as it is named in the run configuration manager in Ranorex Studio. Useful if you have more than one run configuration.

Reporting



4 Enables reporting, i.e. a Ranorex Studio report will be attached to the build.

5 Allows you to enter a custom report name. You can use report-specific placeholders, explained ⇢ here under Reporting.

6 The minimum report level that events must have to be included in the report. (Default) uses the report level setting that was active when the solution was built in Ranorex Studio. Report levels are explained ⇢ here.

TestRail integration

These options allow you to pass test results to a TestRail instance.

7 Enables reporting test results to TestRail.

8 Enter the service connection for your TestRail instance. If you don't have one yet, you need to configure it in Azure DevOps under **Project Settings > Service connections > New service connection > Generic**. There, enter your TestRail credentials. You can also click the **Manage** link next to the field to get to the Service connections menu.

9 The identifier of an existing test run in TestRail to report test results to. If you don't specify an ID or it doesn't exist, results will be reported to a new test run.

10 Creates a new test run in TestRail with the specified name to report test results to. Without this argument the default name is used.

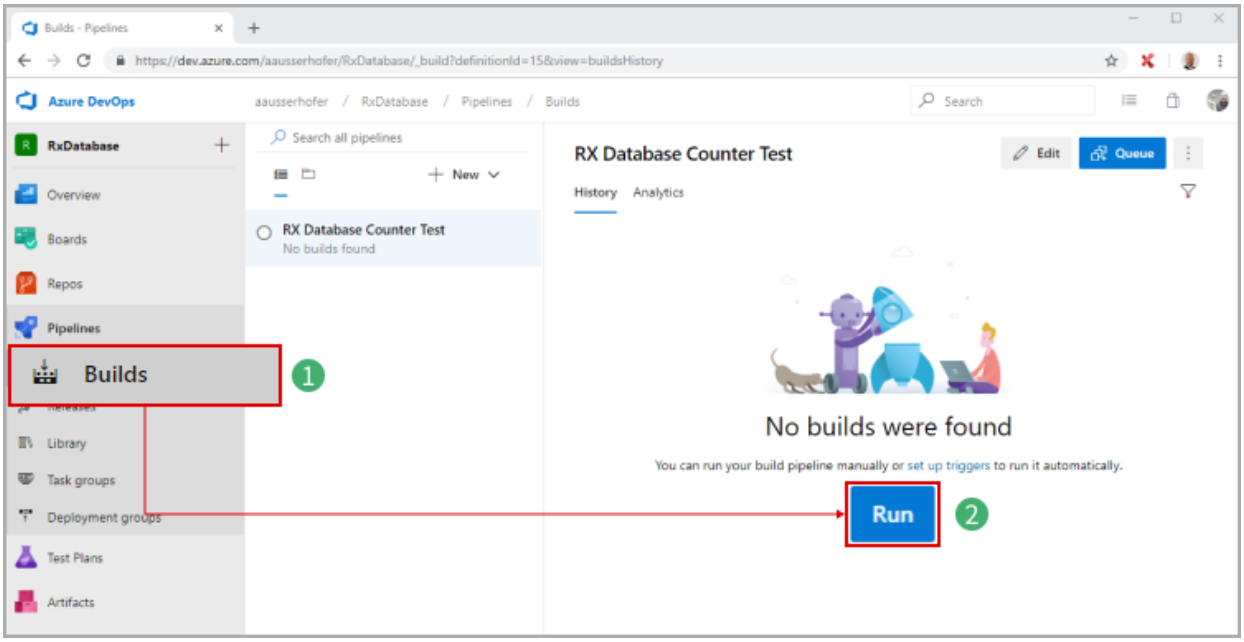
Advanced

Here you can specify additional command line arguments to customize the test run. All available command line arguments are explained ⇢ here.

Execute a pipeline

To execute a pipeline, you need a ⇢ configured Azure Pipelines agent.

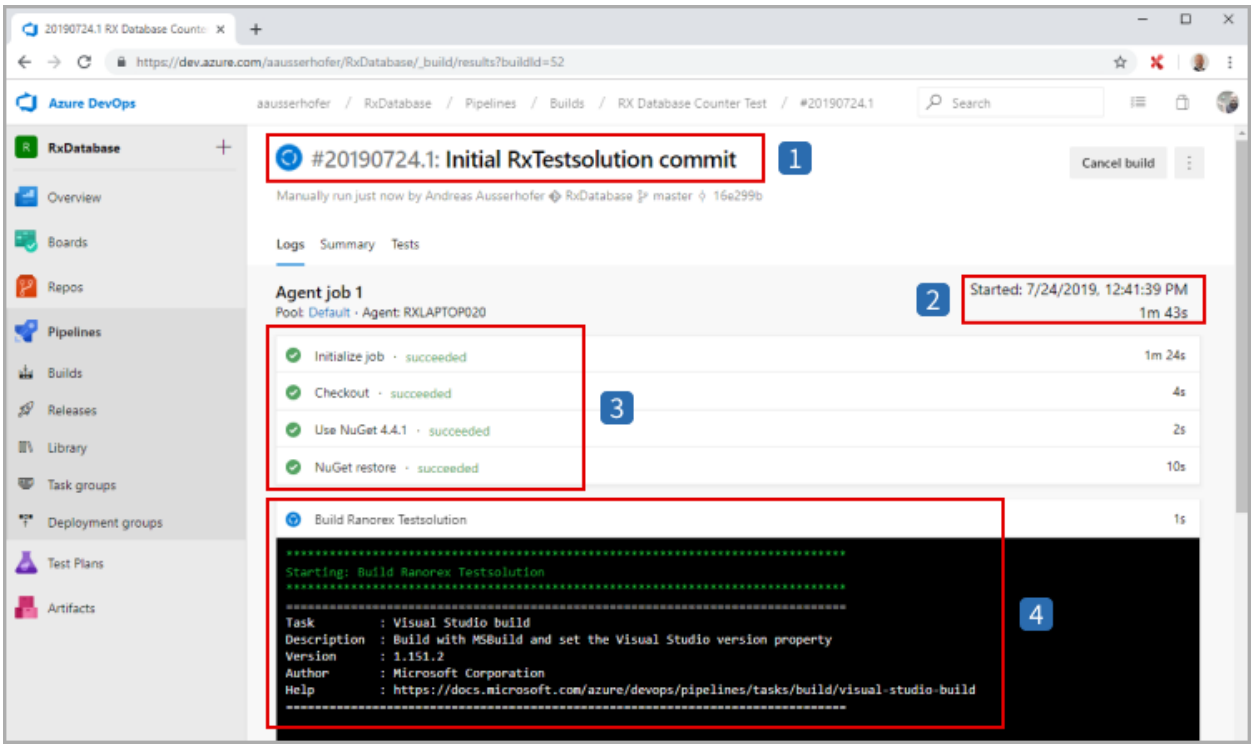
1. **Click Pipelines > Builds**.
2. **Click Run**.



3 In the dialog that appears, **review** the pipeline settings and **click Run**.

Pipeline progress view

The pipeline will carry out all tasks in the specified order. You can view the progress in Azure DevOps.



1 Indicates that a pipeline has been started.

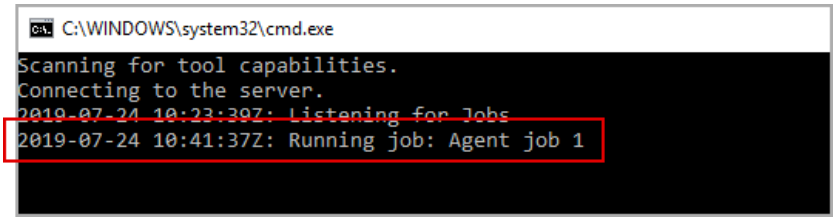
2 Start time and elapsed time.

3 Completed tasks.

4 Current task, including debug info.

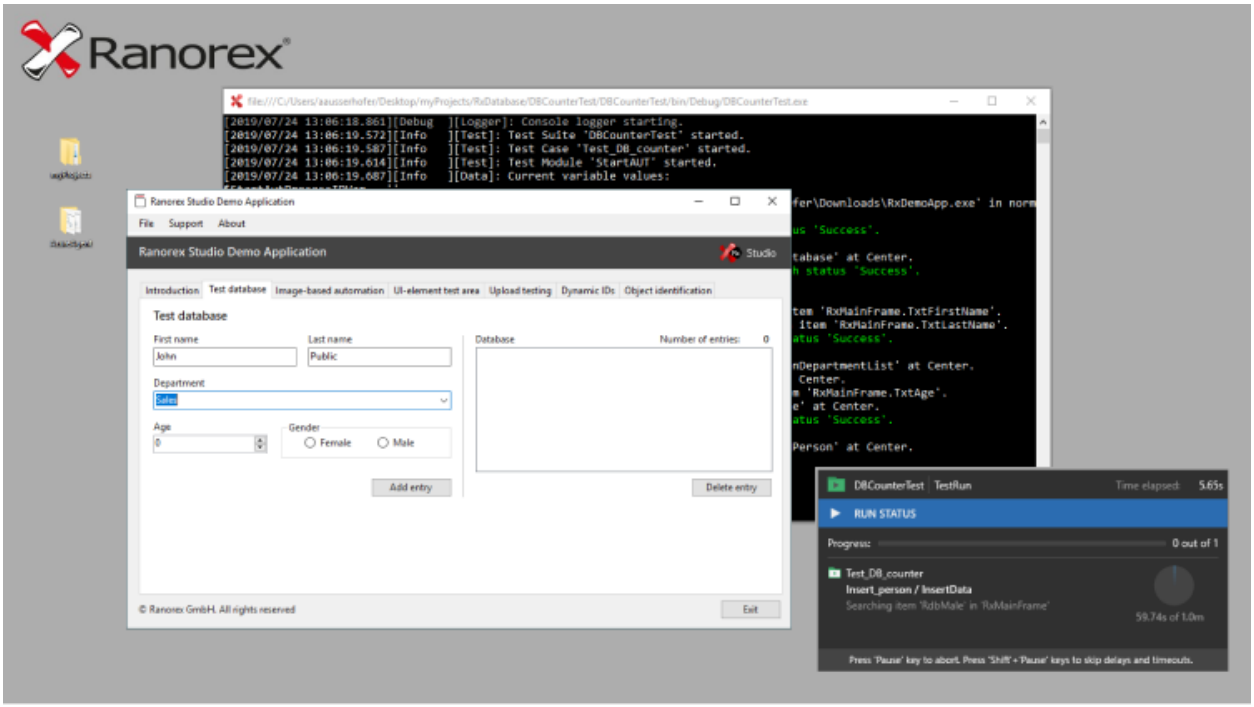
Agent window

Once you've started a pipeline, the agent job is passed to the agent, which then runs it. The agent window will show that a job has been started.



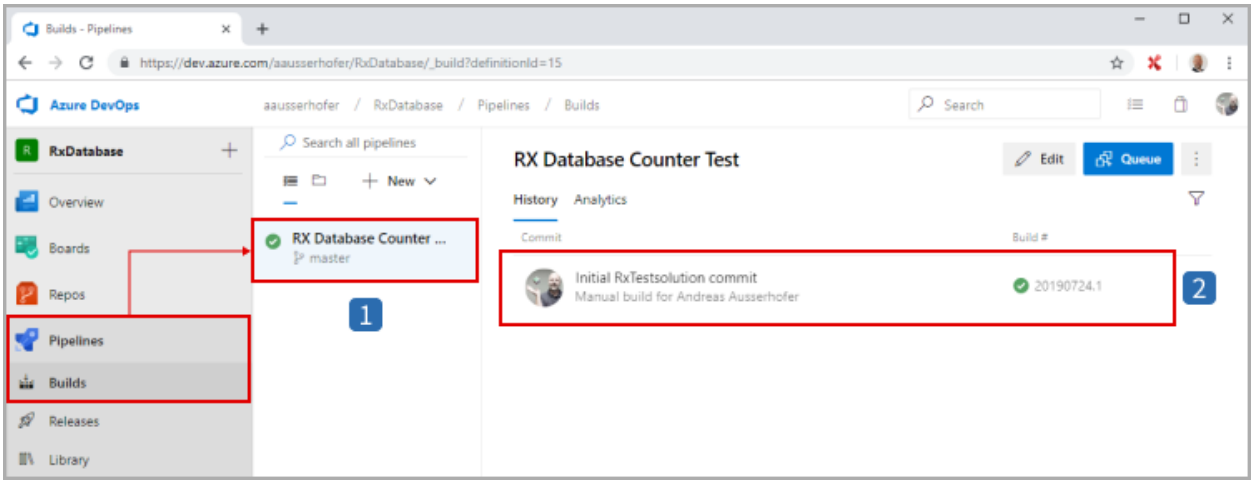
Ranorex Studio test run

The agent runs the Ranorex Studio test as if it were started from within Ranorex Studio.



Pipeline completion

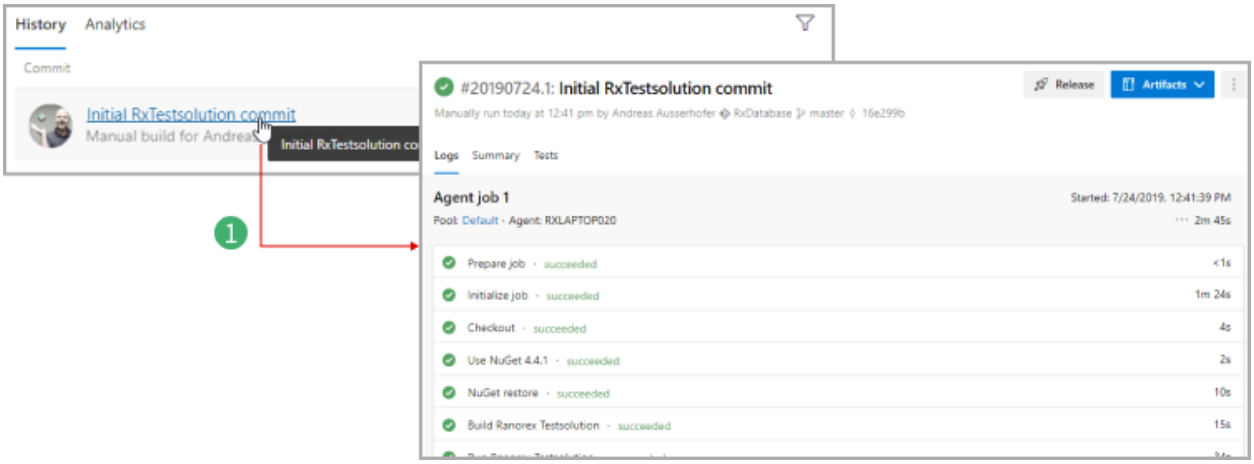
When a pipeline has completed, it will be indicated in the pipeline overview.



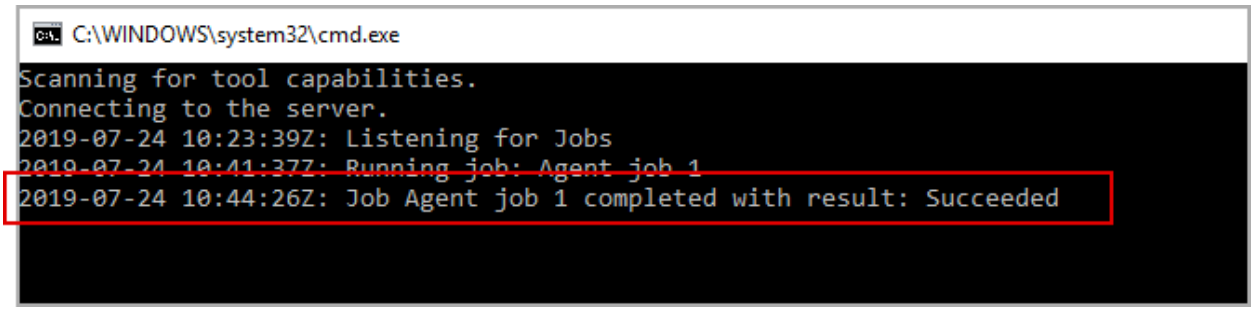
1 List of all created pipelines for the current project.

2 List of all builds in the selected pipeline, including execution stats (success/failure).

1 **Click** the build to display the results view for it.



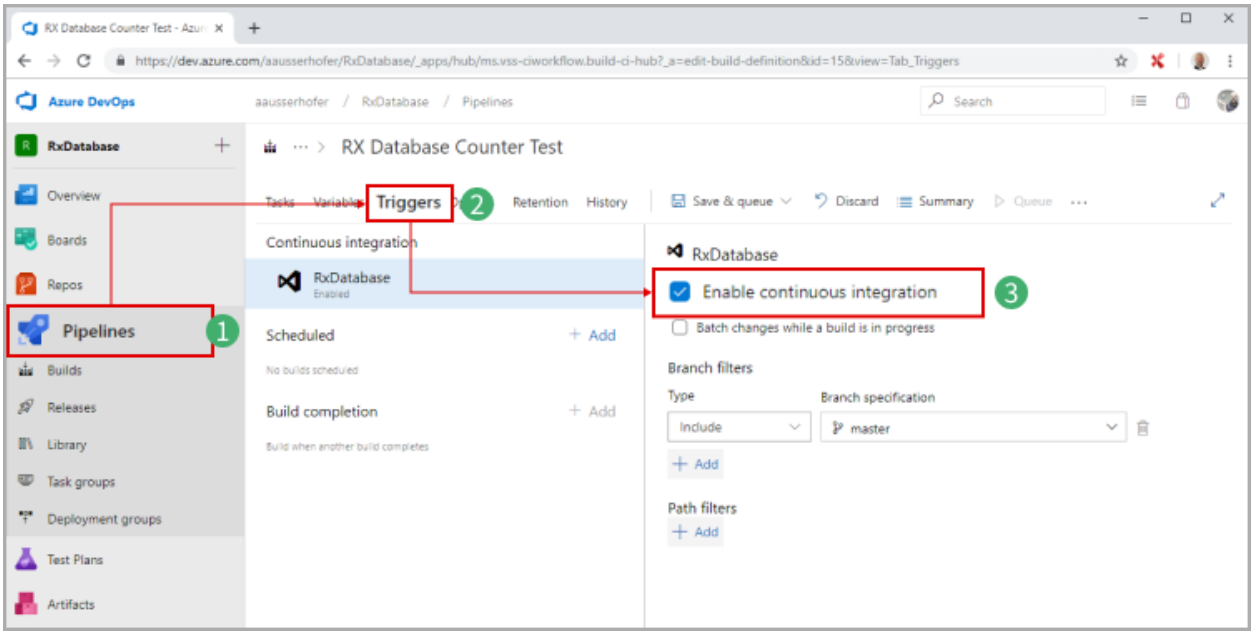
After completing a job, the agent shows the result and then returns to idling.



Trigger a pipeline automatically

You can also trigger pipelines automatically. To do so, you need to enable the option Enable continuous integration. Any changes to the Ranorex Studio solution committed and pushed to Git will then trigger the respective pipeline.

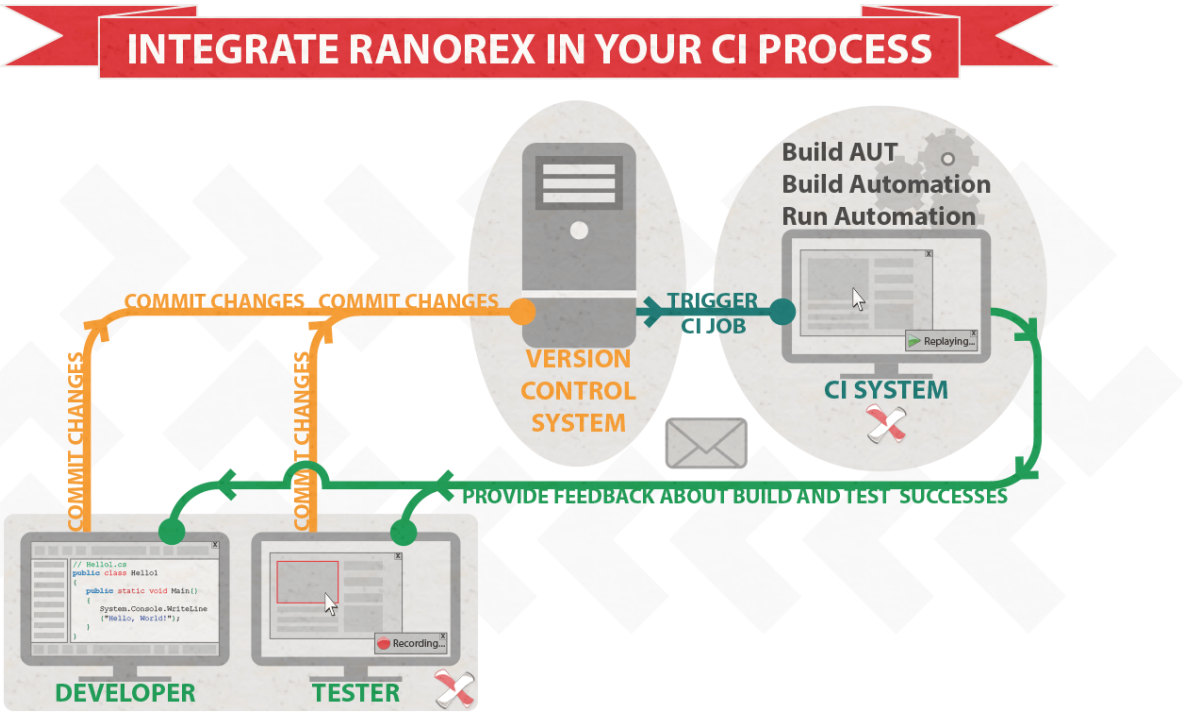
To enable this option:



1. **Click Pipelines**, select the pipeline, and **click Edit**.
2. **Click Triggers**.
3. **Check Enable continuous integration**.

Continuous integration & test management

Ranorex can perfectly be used as a standalone tool to launch automated tests manually on local and remote machines. However, it might be more powerful and comprehensive to integrate Ranorex with other tools. These tools could be continuous integration (CI) tools, test management (TM) tools or simply with task scheduling tools. This comes with the main benefit that automated tests get launched automatically. The launch could be triggered by a specific action (e.g., source code commits from a developer), a periodical build (e.g. “nightly build”), or a test case with underlying test automation.



Continuous integration tools

There are a lot of brilliant and useful Continuous Integration tools around. See how to integrate Ranorex into a typical Continuous Integration process which is explained in detail in our general blog post:

Integrate Ranorex into Any Continuous Integration Process

For several widely spread Continuous Integration tools please find the specific blog post here:

- Jenkins: Integrating Ranorex Automation in Jenkins CI Process
- Bamboo: Bamboo CI with Ranorex Test Automation
- TeamCity: Integrating Ranorex Automation in TeamCity CI Process

For a deeper integration of your Ranorex Report into a common CI tool (e.g. Jenkins) using xUnit/jUnit please see the blog post:

Fully integrate your Ranorex Report with CI tools like Jenkins using xUnit

This blog post also contains explanations and downloadable samples for the XSL transformation of the Ranorex report to xUnit/jUnit format.

Test management and application lifecycle tools

Nowadays Test Management (TM) tools and Application Lifecycle Management tools (ALM) are more and more important. They store the information how testing is organized, implemented, and executed as well as a clearly arranged history of executed tests from the past.

There is a lot of excellent tool support. For the integration of Ranorex with HP quality center, we have a blog post that explains the process in detail:

- Running Ranorex Automated Tests with HP Quality Center

Although Coded UI Tests as well as VAPI-XP tests might be specific for the particular tool, the common idea of integrating Ranorex with TM/ALM tools might be pretty much the same for all other tools around. It all breaks down to the usage of command line arguments for tailored automated test runs with Ranorex.

Team Foundation Server

There is a comprehensive article about the Microsoft Team Foundation Server, Visual Studio and Microsoft Testmanager and the Integration with Ranorex. Please find the documentation here:

TFS 2012 and Ranorex

Atlassian Jira

There is a very informative blog post on how to integrate Ranorex Test Cases into the issue and defect management tool Jira. Please find it here:

- Integrating Ranorex Test Cases into JIRA

Source Control / Revision Control Tools

Integration can also be accomplished with Ranorex Studio – namely integrating Source Control tools. First and foremost it should be mentioned that any common Source Control tool can be used to control relevant files of a Ranorex Studio Solution – which are text-based files only.

There is a specific integration of two of these tools, namely for SVN (Subversion) and TFS (Team Foundation Server). Please find detailed information on the source control integration and support here: ⇢ Source Control.

Jira integration

Jira is an issue tracking software by Atlassian. It is used for agile project management, which includes test management. The Jira integration allows you to connect your Ranorex Studio solution to a Jira project. You can then manually create issues for failed test cases directly from a test report or have Ranorex Studio automatically create and update issues on Jira.

On this page, you'll find out how to set up the integration with the Jira wizard, the difference between manual and automatic issue handling, and how you can further configure the integration.

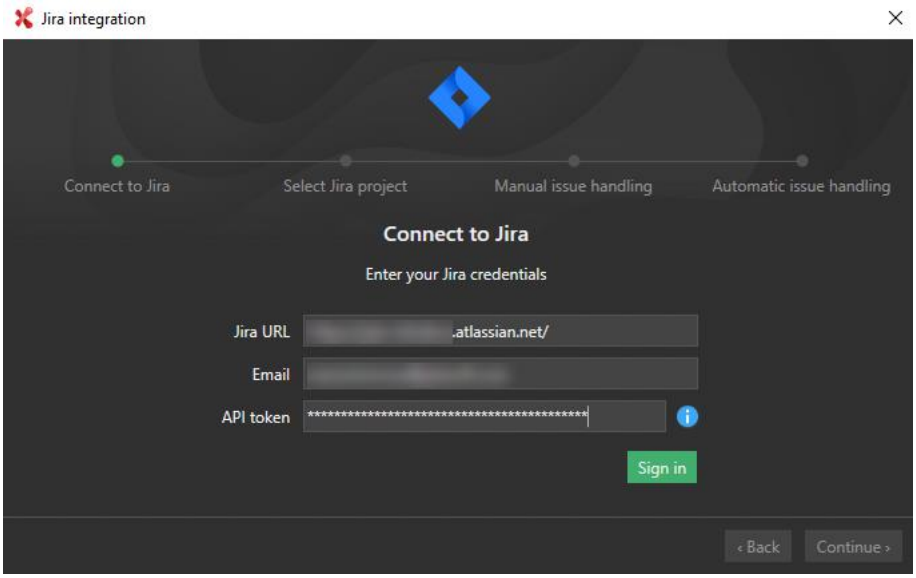
Setup with the Jira wizard

To set up the integration, you need to use the Jira wizard in Ranorex Studio.

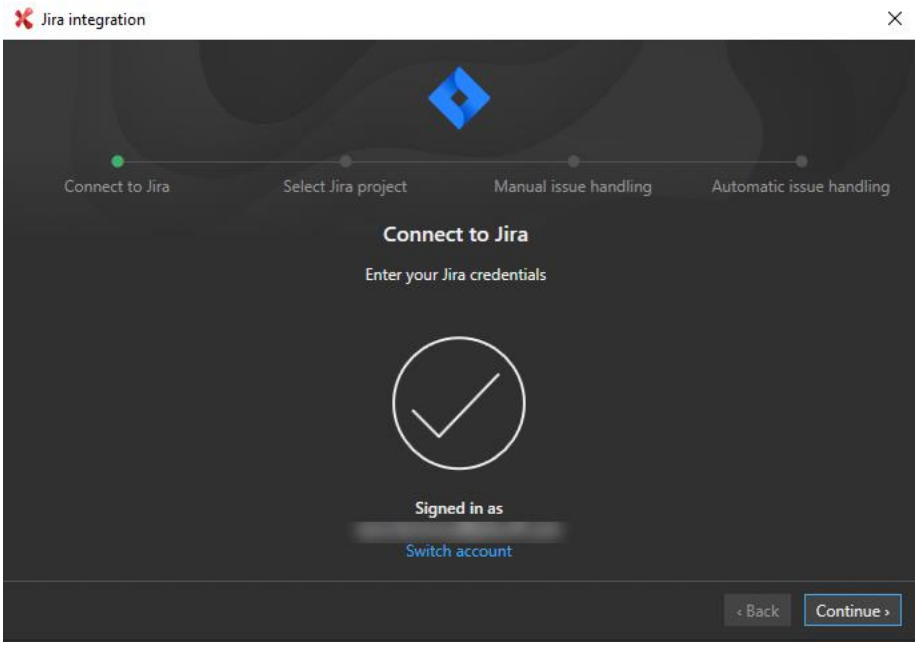
To start it:

1. With a solution opened in Ranorex Studio, **click Tools > Jira integration > Start Jira wizard**.
2. The wizard opens with the **Connect to Jira** screen. **Enter** your credentials and **click Sign in**.

> **Note**
>
> Your credentials will be saved in the Windows Credential Manager on this machine. Deleting them is explained further below.



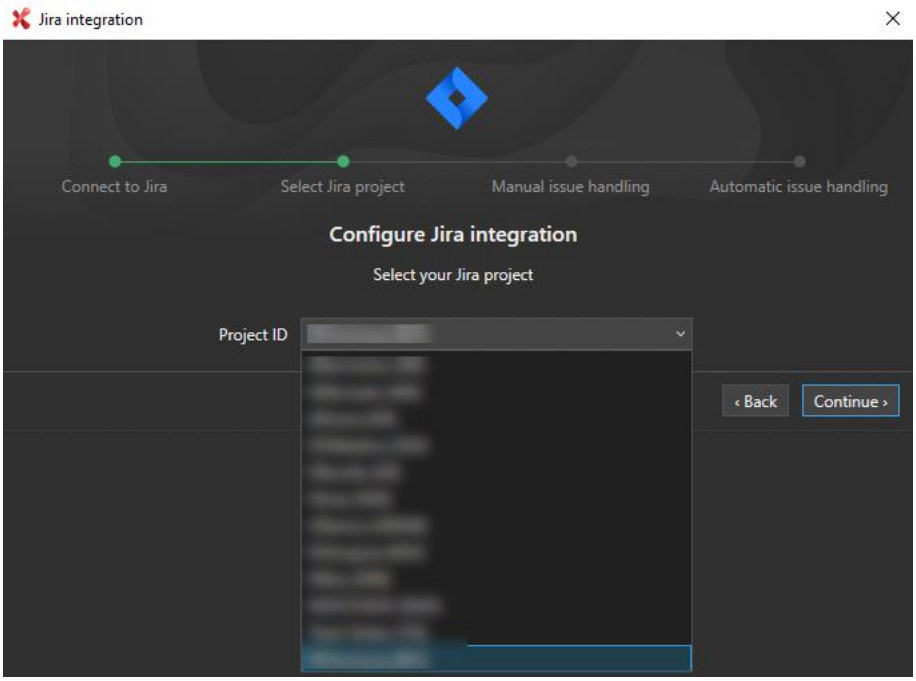
3 Once signed in successfully, **click Continue.**



Note

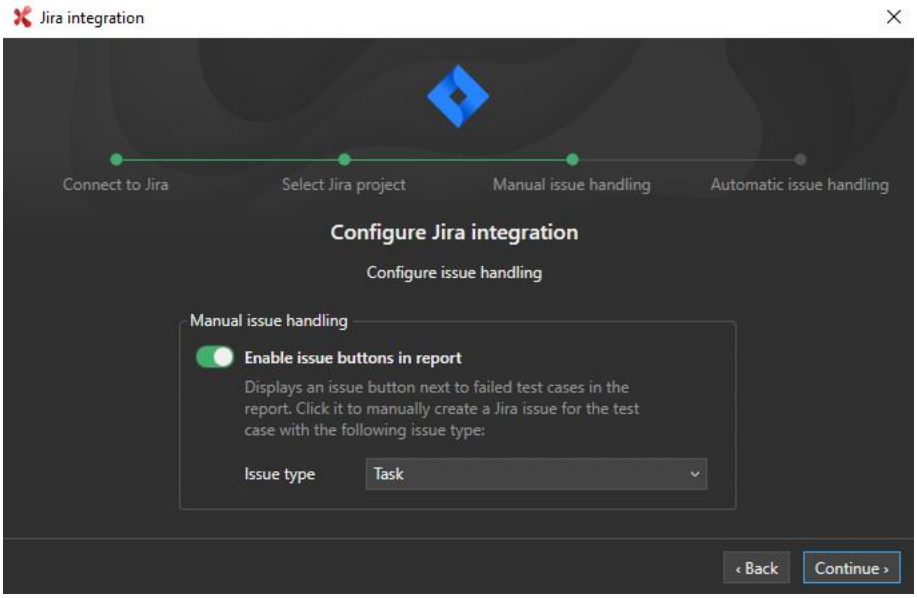
If you entered the URL to a **Jira cloud instance**, you need to enter your **email** and the cloud's **API token** instead. The UI will change to reflect this.

4 From the drop-down menu, **select** the Jira project you want to link the solution to and **click Continue**.

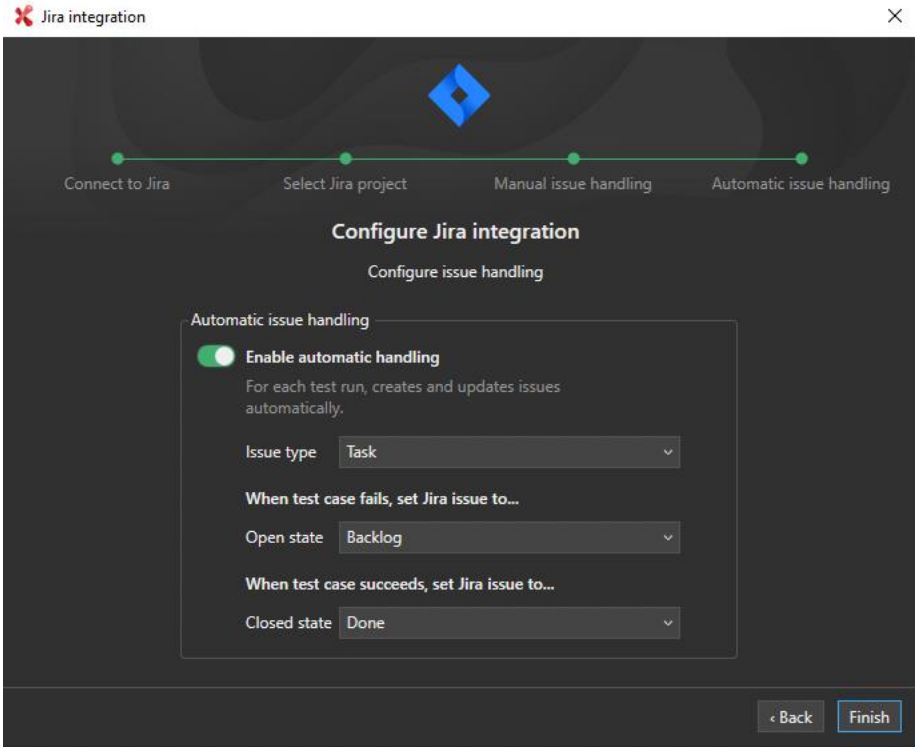


On the follow two screens, you'll configure **manual** and **automatic issue handling**, which are explained in detail further below.

5 If you want to activate manual issue handling, **switch on Enable issue buttons in report** and **select** the type of issue that should be created by default. **Click Continue.**



6 To activate automatic issue handling, **switch on Enable automatic handling** and **select** the default issue type and open/closed states. **Click Finish** to complete setup.



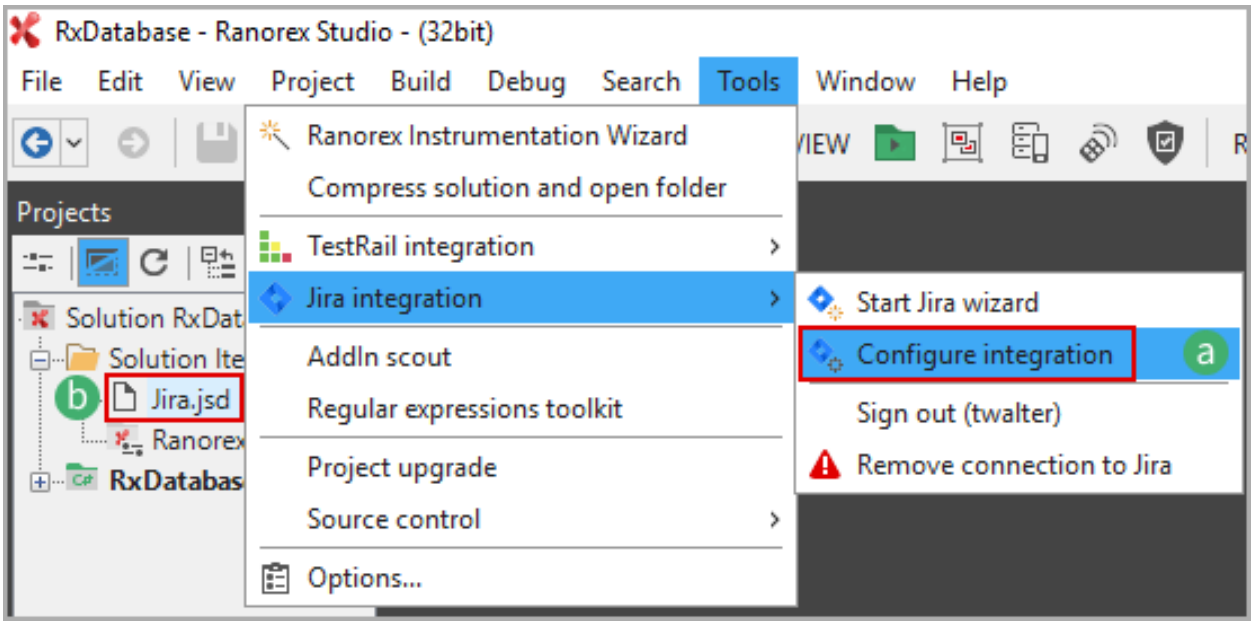
Jira Configuration File

After you've completed setup with the Jira wizard, you are ready to start running tests and creating issues on Jira. However, there are some more options you can configure in the configuration file.

The configuration file **Jira.jsd** is stored in the **Solution Items** folder and contains all information the Jira integration requires to work.

To open it:

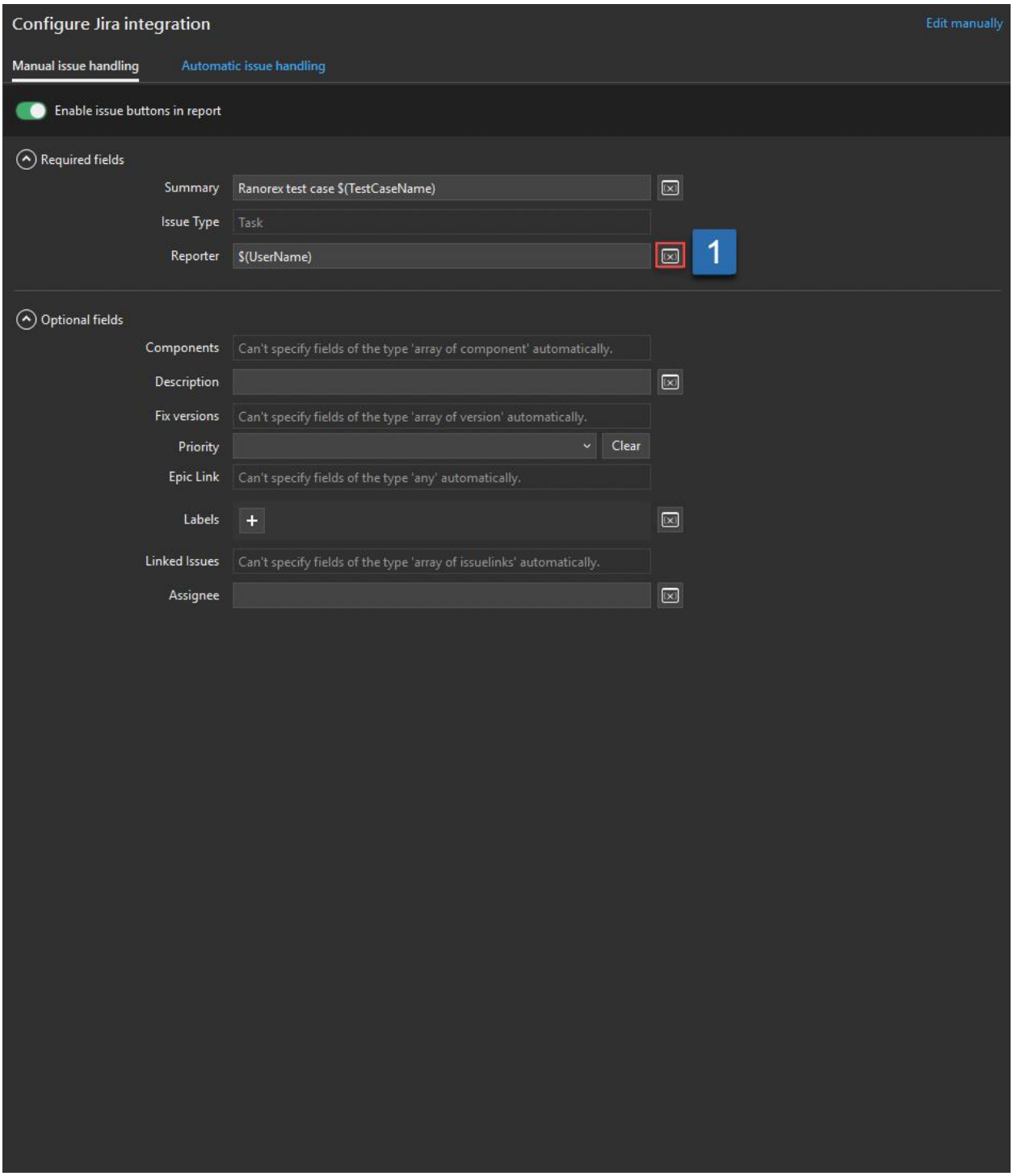
1. With the Jira integration set up in Ranorex Studio,
   a. **click Tools > Jira integration > Configure integration**.
   b. **double-click** Jira.jsd in the projects view.



The configuration screen opens, with tabs for manual and automatic issue handling. Here, you can specify default values for various fields. These will be used when you create an issue by clicking on the button in the report or when Ranorex Studio creates them automatically.

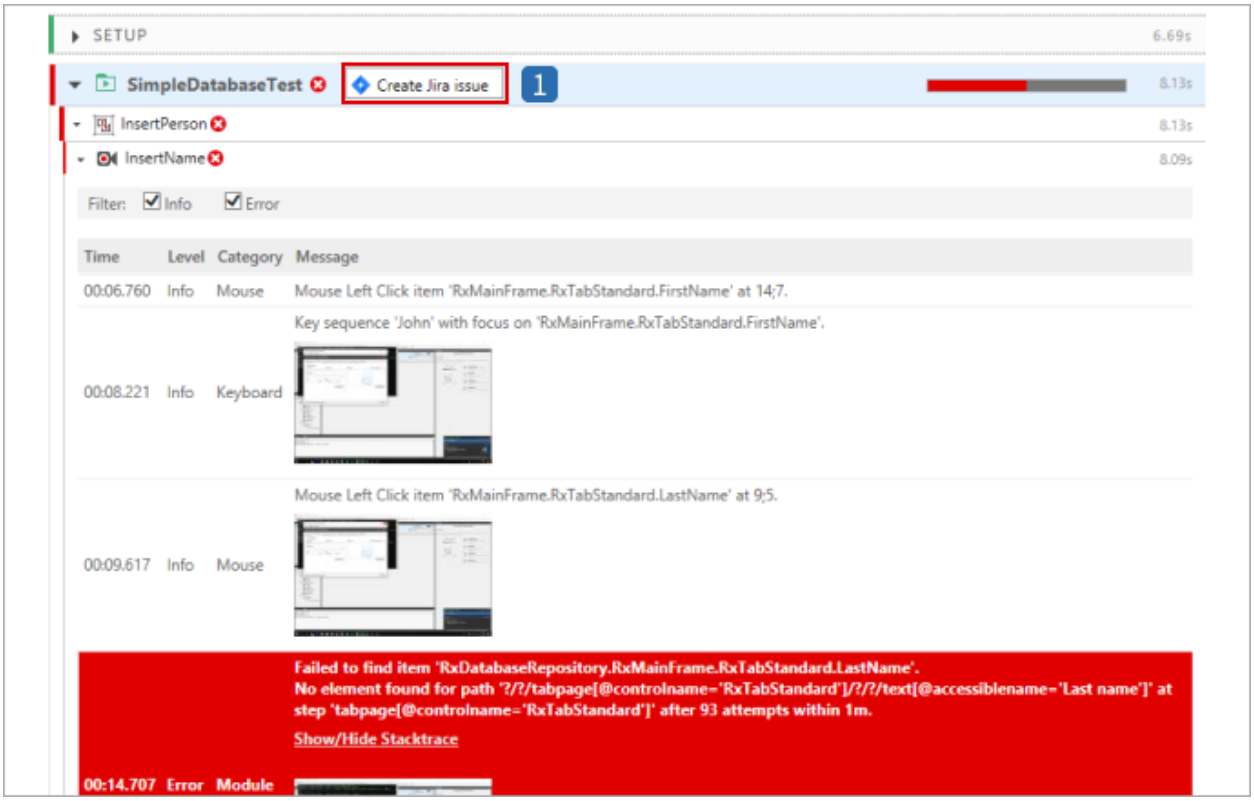
Manual issue handling

In the Manual issue handling tab you can create issues for failed test cases manually from the report.



1 **Variable editor**
Clicking this button brings up the variable editor. It lists the variables you can enter into the fields and a description for each of them.

For manual issue handling the test report will include a Create Jira issue button next to each failed test case:



1 **Create issue button next to a failed test case in the report.**
Clicking it opens the integrated Jira project and the dialog box for creating a new issue. The default issue type (task, story, etc.) was selected in the Jira wizard or in the configuration options of the integration (explained further above). You can also set values for other Jira issue fields in the new issue dialog box.

Automatic issue handling

Automatic issue creation/updating follows the behavior described below. For explanation purposes, suppose we set the issue type to **Bug**, the open state to **To do** and the closed state to **Done**. The behavior would be as follows:

If a test case fails one of the following actions will be taken:

- Ranorex creates a new bug issue in Jira and sets it to To do state. By default, the bug issue summary is the test case name and the description is the unique test case tag for identification by Ranorex Studio and automatically attaches the test report from

Ranorex Studio. This information allows you and your team to see exactly what went wrong.
- No action is taken if an open issue with the same test case tag already exists.
- If a closed issue with the same tag exists, the issue is reopened and set to To do state.

If a test case succeeds:

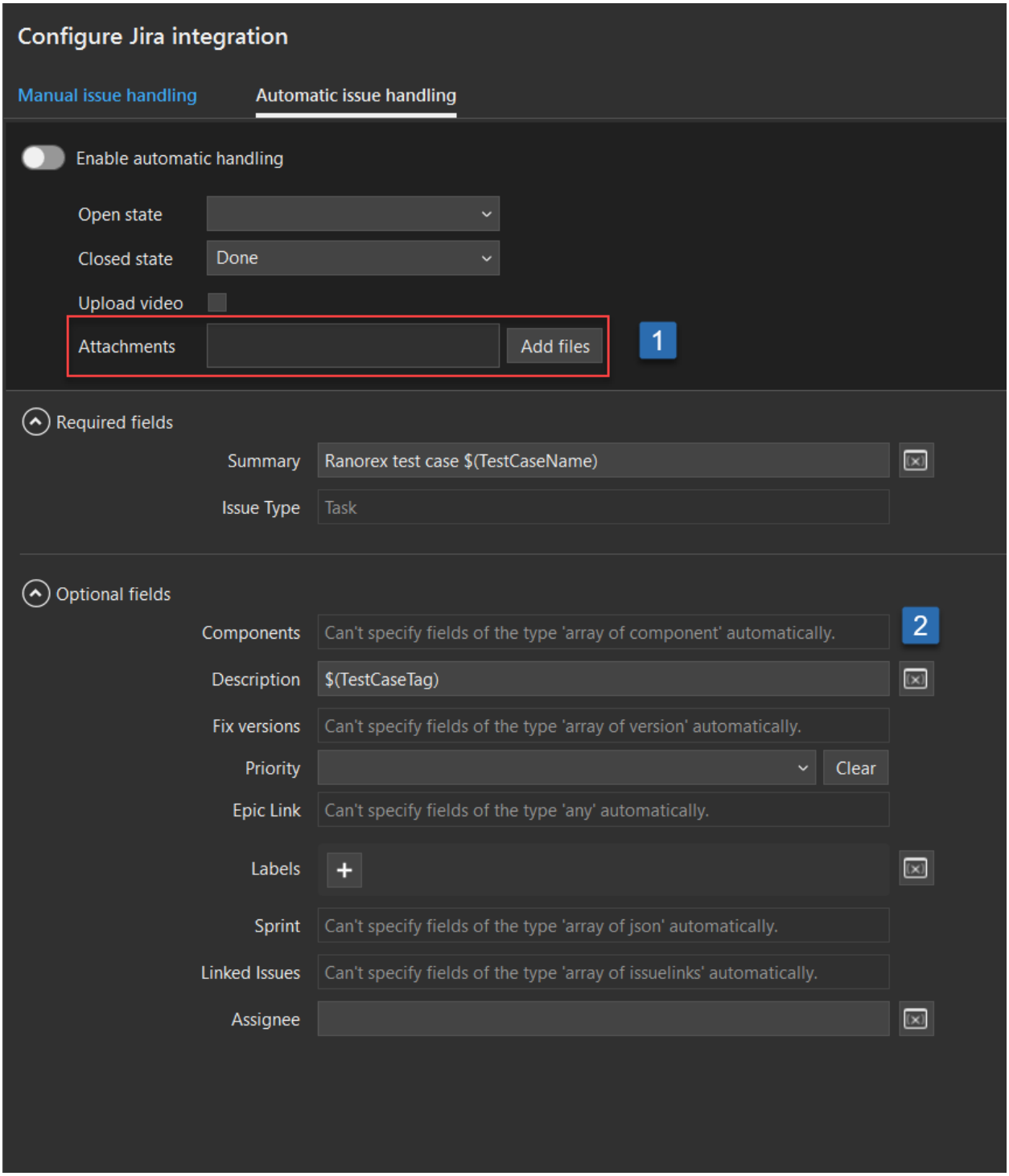
- No issue is needed.
- If an issue with the same test case tag exists, the state of this issue is set to Done, then the issue is closed.

> **Note**
>
> As you can see from the explanations above, automatic handling works by **identifying issues by the test case tag**. This also works if you include the tag in a manually created issue. If you then switch to automatic handling, Ranorex Studio will also update this issue automatically.

In the Automatic issue handling tab, you can configure that Ranorex Studio creates issues on Jira automatically for each test.

> **Attention**
>
> For automatic issue handling, at least one field of the type **string**, e.g. Description, **must contain the variable $TestCaseTag**. Otherwise, Ranorex Studio can't identify issues automatically. You can find out what type a field is by mousing over it.

Upload Attachments



1 To include attachments in the Jira issue, click Add Files. A window opens, where you need to select the files you want to include.

2. The selected files are listed in the Attachments section. You can delete the attached files by clicking the trash bin located on the right side of the attachment path.
2. Some field types cannot be configured automatically, which means you need to fill them in manually when you create the issues with the report button. For automatic issue handling, they will remain empty unless you update the issue manually after it's been created.

Jira Custom Fields

Ranorex supports Jira custom fields. When a Jira project contains custom fields, they show up in the Jira automated integration setup screen. If these custom fields contain dropdown values, Ranorex users can select the appropriate value.

Custom fields that are not dropdowns fields also display, but users will not be able to change these values within Ranorex.

Sign out

Signing out means **deleting your Jira credentials** from the current machine, i.e. from the Windows Credential Manager.

To do so:

1. In Ranorex Studio, **click Tools > Jira integration > Sign out (<user name>)** and **confirm** the dialog.

Remove connection to Jira

Removing the connection to Jira means **deleting the Jira.jsd file**. All configurations will be lost.

To do so:

1. In Ranorex Studio, **click Tools > Jira integration > Remove connection to Jira** and **confirm** the dialog.

Note

When a Jira connection is started, you can switch projects that are under the same connection. You need to sign out to remove the credentials.

If you select Remove connection to Jira, the Jira.jsd file along with the configurations are removed. In this scenario, the Jira credentials remain signed in.

TestRail integration

The TestRail integration allows you to **connect Ranorex Studio and TestRail**. You can import and export tests, synchronize between the two programs, and report results from test runs in Ranorex Studio to TestRail in real time.

Getting started

Before we can start connecting to TestRail, we first need to make sure it's configured correctly. Please refer to the respective sections of the TestRail User Guide for more information.

Configuring TestRail

> **Note**
>
> To make the following settings, you will need to have Admin privileges in TestRail.

Enable API

TestRail's API needs to be enabled so Ranorex Studio and TestRail can interact.

Go to Administration > Site Settings > API and check the Enable API option.

Permissions

Depending on how you use the TestRail integration, you will need certain permissions. To set them, go to Administration > Users & Roles and create a role with the appropriate permissions or change the permissions of an existing role. Then assign that role to the respective users.

Importing into Ranorex Studio: No permissions required.

Exporting to existing projects: Add/Edit Cases & Sections, Suites.

Exporting to new projects: You need to be an admin on TestRail.

Reporting results to an existing TestRail run: Add/Edit Test Results.

Reporting results to a new TestRail run: Add/Edit Runs & Plans.

Automation Type field

TestRail 5.5 and later:

The Automation Type drop-down field is already present by default. Simply set it to Ranorex.

TestRail 5.4 and earlier:

You will need to add a custom field to the user interface for importing and exporting to work correctly.

Go to Administration > Customizations > Case Fields and add a field with the following values:

Label: Automation Type

System Name: automation_type

Type: Dropdown

Click Add Projects & Options and specify Ranorex as Option 1. Click the Selected Projects tab and apply the field to your projects.

Required custom fields

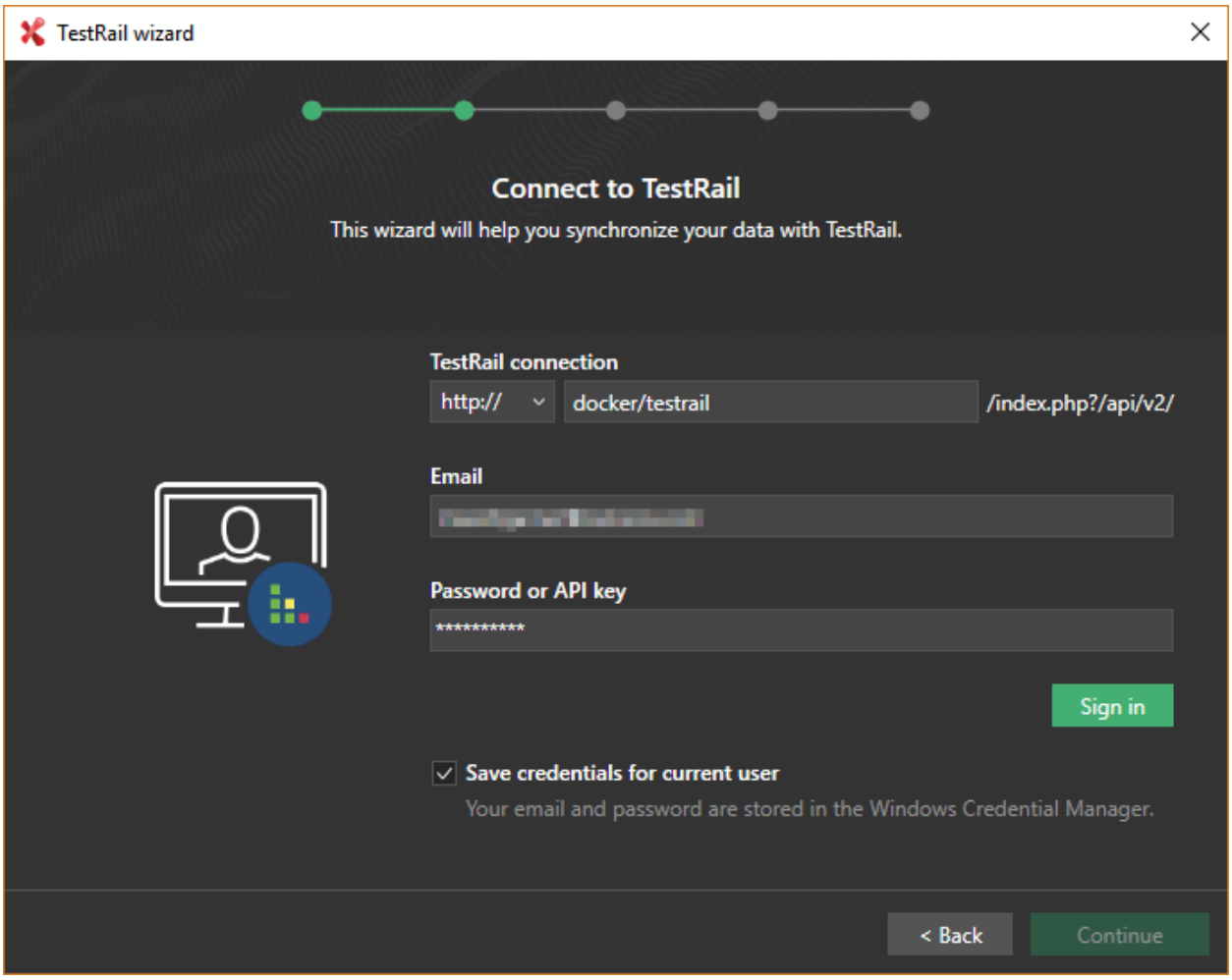
If you have required custom fields in TestRail, **all of them should have default values set.** Otherwise, an error will occur during exporting and synchronizing because Ranorex Studio won't know what value to populate the required custom field with.

Fixing this error is explained in ⇢ Importing, exporting, synchronizing.

Starting the TestRail wizard and signing in

In Ranorex Studio, open the solution you want to work with. Then go to Tools > TestRail integration > Start TestRail wizard. The wizard's welcome screen will appear. Before you continue, make sure you've ⇢ configured TestRail correctly. Then click Continue.

The Connect to TestRail screen will appear. Enter the server address and your credentials.



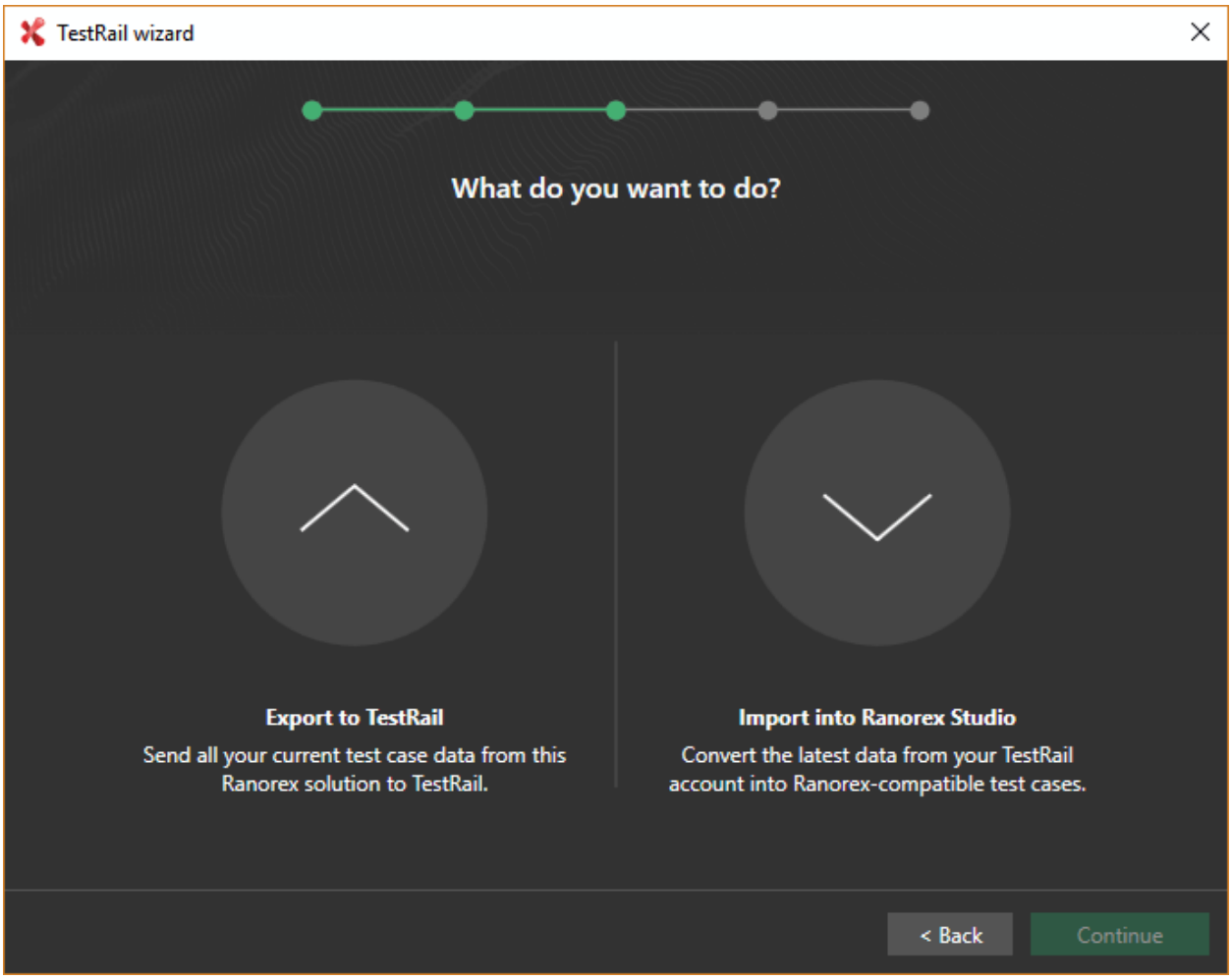
Please note

The server address applies to both the TestRail on-premises and the cloud solution. You can find it in the address box of your browser.

You can save your credentials, so you won't have to sign in again.

Click Sign in to connect to TestRail. When you start the TestRail wizard again already signed in, you can just click Continue on this screen.

The Import/Export screen will appear. Go to ⇢ Importing, exporting, synchronizing to find out how this feature works.



Importing, exporting, synchronizing

This subchapter explains the importing, exporting and synchronizing processes and related topics.

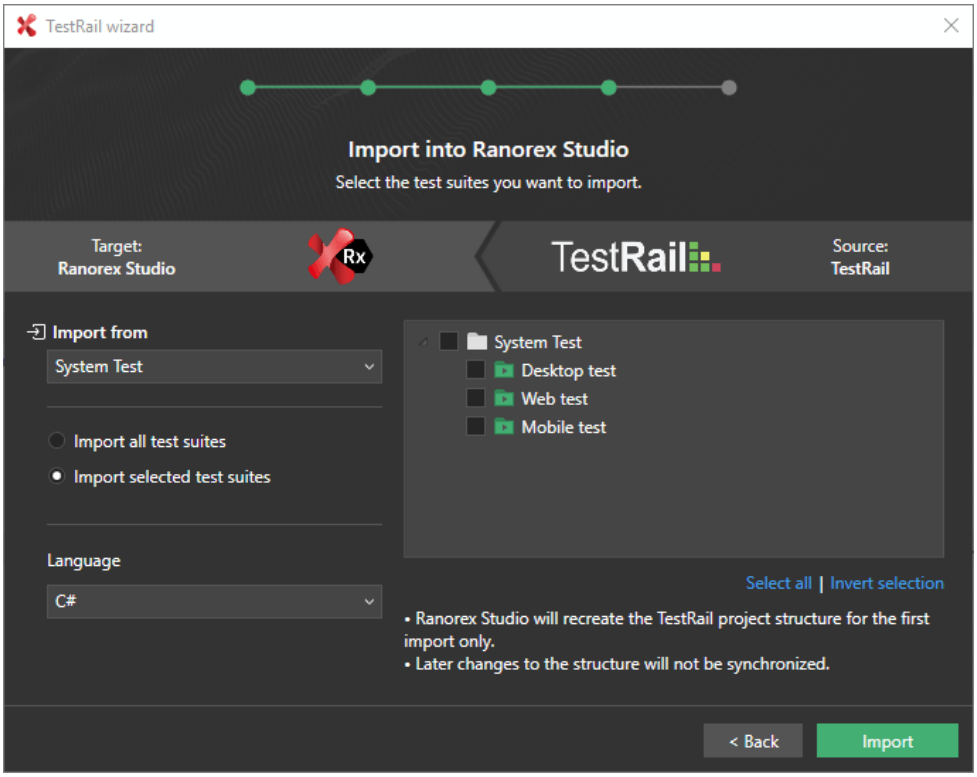
Before you get started, take a look at the following table to get an overview of **how elements in the two programs are mapped to each other** in the importing and exporting processes.

Ranorex Studio	Direction	TestRail
Solution	↔	Project
Project with test suite	↔	Test suite
Test case	↔	Case
Smart folder outside a test case	↔	Section
Smart folder inside a test case	→	Step
Module	→	Step

Importing

Importing allows you to import test suites from a TestRail project into a solution in Ranorex Studio.

Select Import into Ranorex Studio in the TestRail wizard and click Continue. The Import screen will appear.



To import one or more test suites:

1. Select the TestRail project you want to import from.
2. Select whether you want to import all test suites in the project or only the selected ones. If the latter, select your test suites.
3. Select the programming language you want your project to use in Ranorex Studio.
4. Click Import.

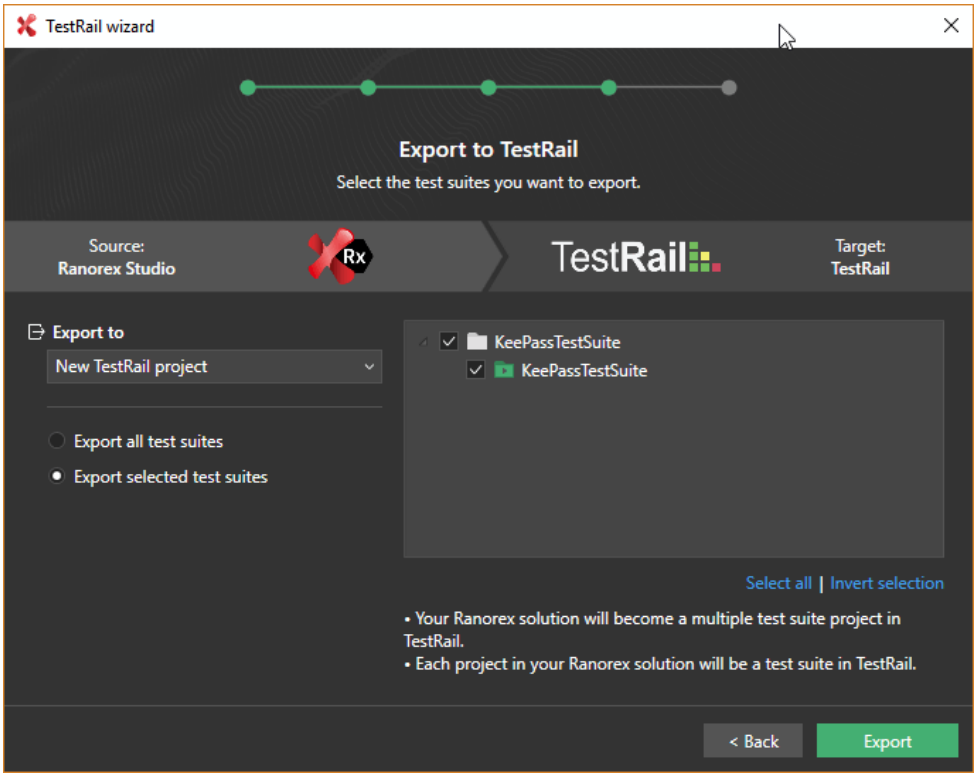
Note

- You can import from all types of TestRail projects with no limitations.
- Ranorex Studio will only import TestRail cases that have the field **Automation Type set to Ranorex**. All other cases will be ignored.
- Ranorex Studio will recreate the structure of the TestRail project and test suite(s) for the first import only. After that, ⇢ only certain changes will be synchronized.
- Steps will not be imported.

Exporting

Exporting allows you to export test suites from a Ranorex Solution to a TestRail project.

Select Export to TestRail in the TestRail wizard and click Continue. The Export screen will appear.



To export one or more test suites:

1. Select the TestRail project you want to export to. It can be a new or existing project. If the former, Ranorex Studio will automatically create a new project on TestRail with the name of the Ranorex solution.
2. Select whether you want to export all test suites in the solution or only the selected ones. If the latter, select your test suites.
3. Click Export.

Note

- It's not possible to export multiple Ranorex test suites to single test suite projects in TestRail (baseline included). You will need to switch the TestRail project to multiple test suites or create a new Ranorex solution with only one test suite.
- Your Ranorex solution will be created as a multiple test suite project in TestRail, with each Ranorex test suite project being recreated as a TestRail test suite.
- Ranorex Studio will recreate the structure of the Ranorex solution and test suite(s) for the first export only. After that, ⇢ only certain changes will be synchronized.
- Modules will be exported as steps.

After importing/exporting

Once you have imported from or exported to a TestRail project, the Ranorex solution is connected to it. This has the following effects:

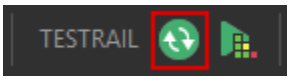
- Connected elements (test suite, test cases/smart folders in test suites) in the solution display a TestRail icon. Test cases and smart folders will also have a link icon in their description. Double click it to go to the corresponding case in TestRail.
- The TestRail sync file is now present in the Ranorex solution.
- Synchronizing is now possible.
- Imports from/exports to any other TestRail projects are not possible. To import from/export to a different TestRail project, create a new Ranorex solution or ⇢ remove your current solution's connection to TestRail.

The TestRail sync file – TestRail.trsd

The TestRail sync file contains all information required for the connection between the Ranorex solution and the TestRail project. It's created after importing/exporting and stored as TestRail.trsd in the Solution Items folder of your solution. You can access it from the projects view.

Synchronizing

Once you've imported or exported a test, the Synchronize button will appear in the TestRail section of the Ranorex Studio menu bar.



Click it to synchronize. Synchronizing carries out an import and an export for the currently connected test suites to update them with changes made in either program.

The following changes are synchronized

- Adding new items.
- Converting smartfoldersto test cases. A new case will be added in TestRail.

The following changes are NOT synchronized

- Deleting items. However, when you delete a test case on either side, synchronizing will remove the connection between them and the Automation Type will be set to empty in TestRail.
- Moving items.
- Renaming items.

Not synchronizing these changes makes it easier to work and collaborate between the two programs, as they serve different purposes and therefore may require different structures.

When should I use importing/exporting, when synchronizing?

Use importing/exporting when you want to

- transfer your data from one program to the other for the first time.
- add additional test suites from the same TestRail project/Ranorex solution.

Use synchronizing when you

- want already imported/exported test suites to be updated with changes made in either program.

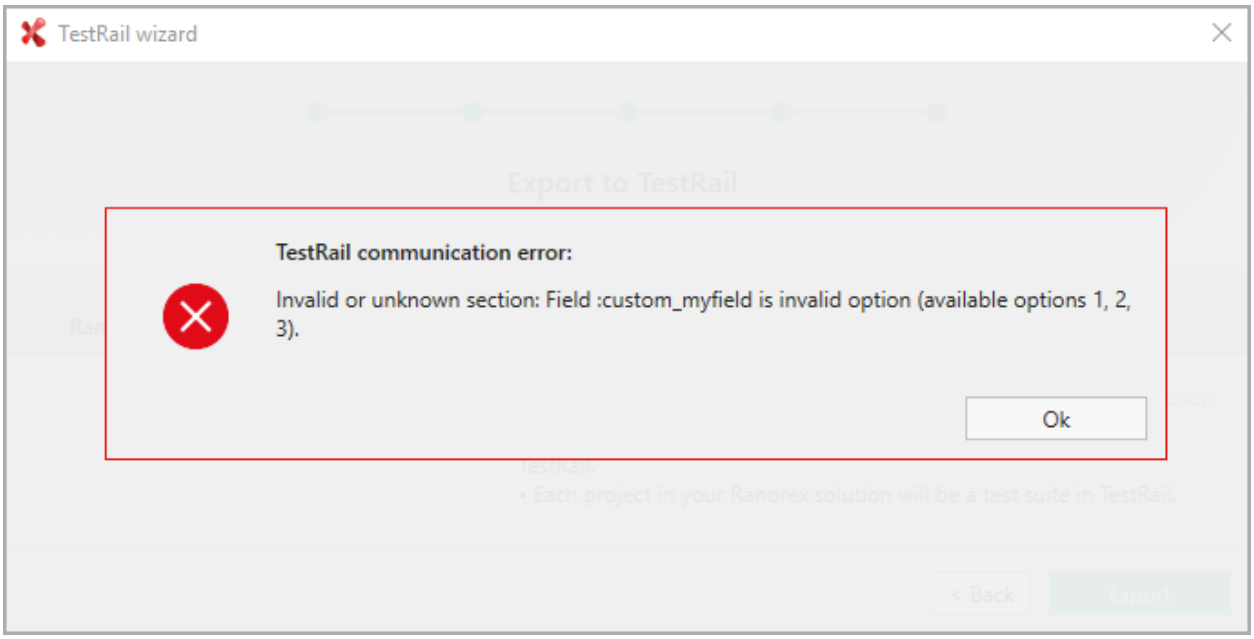
Removing the connection to TestRail

There are several ways to remove the connection to TestRail.

- Go to Tools > TestRail integration > Remove connection to TestRail.
- Delete TestRail.trsd from the projects view.
- Delete TestRail.trsd in the solution folder in Windows Explorer.

TestRail communication error – invalid option

This error occurs when exporting or syncing if a required custom field in TestRail doesn't have a default value set.



There are two ways to fix this.

Fix 1: Edit the custom field in TestRail

This fixes the issue for everyone using the integration with your TestRail instance. However, you will need to have rights to edit custom fields.

1. In TestRail, **find** the field and **assign** it a valid default value.

2. **Repeat** the export/sync in Ranorex Studio.

Fix 2: Change the value in the TestRail sync file

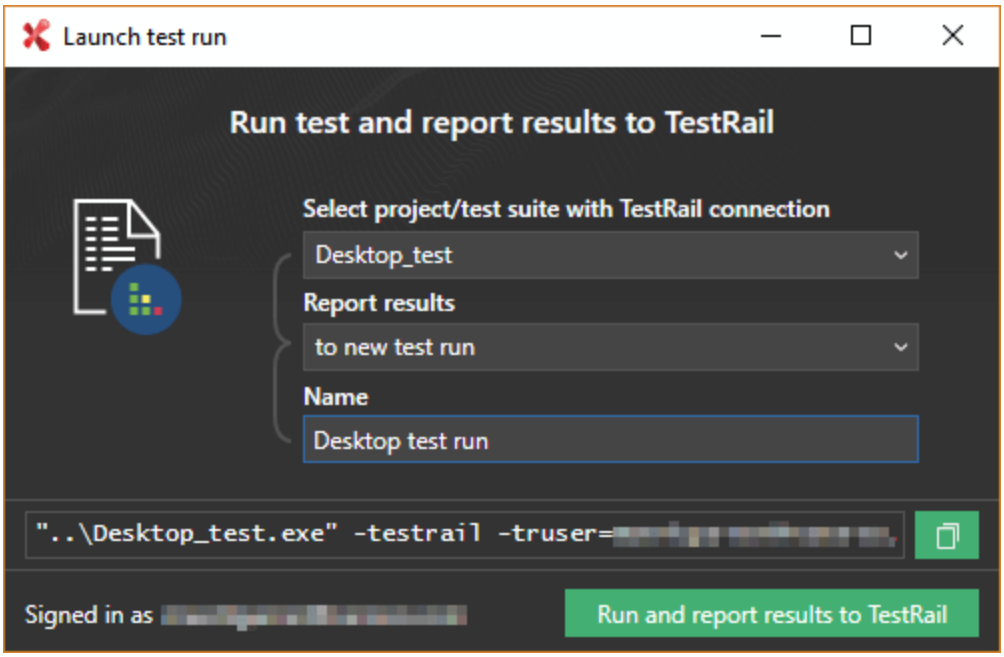
Alternatively, if you can't or don't want to edit custom fields, you can also change the value of the field in the TestRail.trsd sync file. This only fixes the problem for your machine. Others will still get the error.

1. In the projects view in the Solution Items folder of your solution, **double-click TestRail.trsd**.
2. **Find** the custom field mentioned in the error message, i.e. custom_myfield in our case.
3. **Change** its **Value** parameter to a valid value of the custom field.
4. **Save** your solution and **repeat** the export/sync option.

> **Note**
>
> When repeating the export/sync, you may get a message telling you that changes were detected and to repeat the operation. You can safely ignore it.

Running tests and reporting to TestRail

After you've completed an import or export, you can **run your test suite and report the results to TestRail in real time**. You can run your test from Ranorex Studio, or you can use Continuous Integration tools. Either way, the results on TestRail will be the same. The only difference is in how the run is started.



Running from Ranorex Studio

1. Click the TestRail run button. The Launch test run screen will appear.
2. Select which project connected to TestRail you want to run.
3. Select whether you want to report the results ⇢ to a new or ⇢ an existing test run on TestRail.
4. If to a new run, enter a name, if to an existing run, select to which one.
5. Click on Run and report results to TestRail to launch the test run.

Running from a CI tool

1. Click the TestRail run button. The Launch test run screen will appear.
2. Select which project connected to TestRail you want to run.
3. Select whether you want to report the results ⇢ to a new or ⇢ an existing test run on TestRail.
4. If to a new run, enter a name, if to an existing run, select to which one.
5. Click the **green copy button** to the right of the generated string to copy it to the clipboard. **Do not use manual copy + paste**, or the password will not be copied correctly.

6. Paste the string into the appropriate field in your CI tool. You can add ⇢ more arguments if desired.

Whichever method you choose for running, the test will always execute according to the currently selected run configuration. Whether a test case is connected to TestRail or not is irrelevant for execution.

> **Tip**
>
> As the test is running, you can open the run in TestRail to check its status. This is especially useful for long test runs.

Reporting to TestRail

Once Ranorex Studio has started running the test, results will be reported to the TestRail test run in real time. The Ranorex report will always show the full results for the selected run configuration.

Reporting to a new test run

Ranorex Studio will report results only for test cases that are linked to TestRail AND part of the current run configuration. Other test cases will not be reported to TestRail.

Reporting to an existing test run

Ranorex Studio will report results only for test cases that have a counterpart in the existing TestRail run AND are part of the current run configuration. Other test cases will not be reported to TestRail. The Ranorex report will contain a message detailing which test cases didn't have a counterpart in TestRail.

Iterations

In principle, the results for test cases with iterations are displayed just like in the Ranorex report. In other words, if all iterations pass, the case will display as passed, if one fails, the case will display as failed in TestRail. However, the pass/fail information for the individual iterations is only displayed in the Results & Comments section of a case in TestRail.

Blocked test cases

In a Ranorex report, blocked always means not executed. This is not the case in TestRail. Therefore, blocked test cases are reported as “untested” in TestRail.

Collaboration

When collaborating between TestRail and Ranorex Studio, there are a couple of things you should consider.

Basic best practice

TestRail is a test management tool, while Ranorex Studio is a test automation tool. They serve similar, but ultimately different purposes. Based on this we recommend the following approach to leverage the advantages of both programs.

One team is responsible for planning and managing tests on TestRail. They designate what needs to be tested, which of the tests are manual/automated, and they plan test runs.

The other team works in Ranorex Studio. They import the tests designated as automated and automate them in Ranorex Studio. They adapt the structure of the TestRail suites to what makes sense in Ranorex Studio.

Test runs can be initiated by either team or by using Continuous Integration tools. What’s **most important is communicating it to each other**. This applies in general: Keep each other up to date. Let others know what you’re doing. Most **collaboration pitfalls are rooted in a lack of communication**.

Source control and TestRail integration in Ranorex Studio

When using source control and the TestRail integration, it’s important that you designate **one person to coordinate and do the synchronizing**. Otherwise, you will end up with duplicated test cases in TestRail or worse.

Follow this workflow:

1. Connect your solution to TestRail, i.e. carry out an import or export, depending on your starting point.
2. Check in TestRail.trsd into your version control tool.
3. People make changes to the solution and check them in.
4. Before synchronizing with TestRail, make sure everyone has the latest version.
5. Synchronize and check everything in again.

6. Repeat from step 3.

Applitools Eyes integration

Applitools Eyes is a tool for automated visual validation. It is used for visual regression tests. Normally, Applitools Eyes needs to be integrated with an SDK. With the Ranorex Applitools Eyes integration, you can use the tool with your Ranorex tests. No coding required.

Throughout the chapters of this guide, we assume you're familiar with Applitools Eyes. Please refer to the Applitools documentation if you're unsure how to use Applitools Eyes.

Adding the integration to your solution

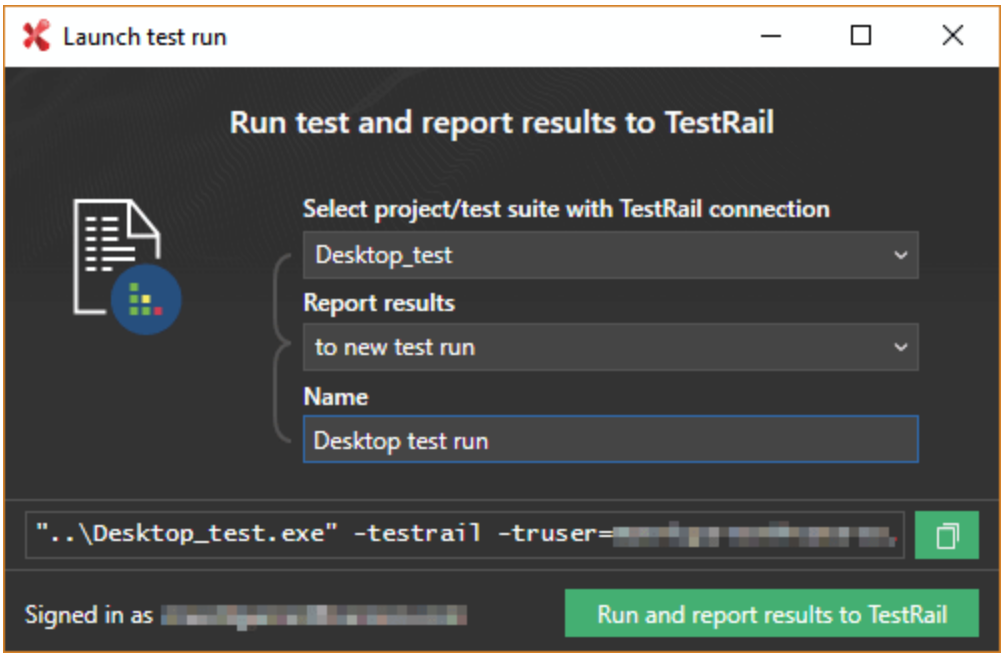
1. In the projects browser, **right-click** the desired solution.
2. **Click Manage packages...**
3. In the search field of the window that appears, **enter Ranorex Applitools**.
4. From the results on the left, **select Ranorex with Applitools Eyes** and **click Add**.
5. **Click Close**.
6. In the module browser, **verify** that the two modules **InitializeEyes** and **FinalizeEyes** are present.

Create tests

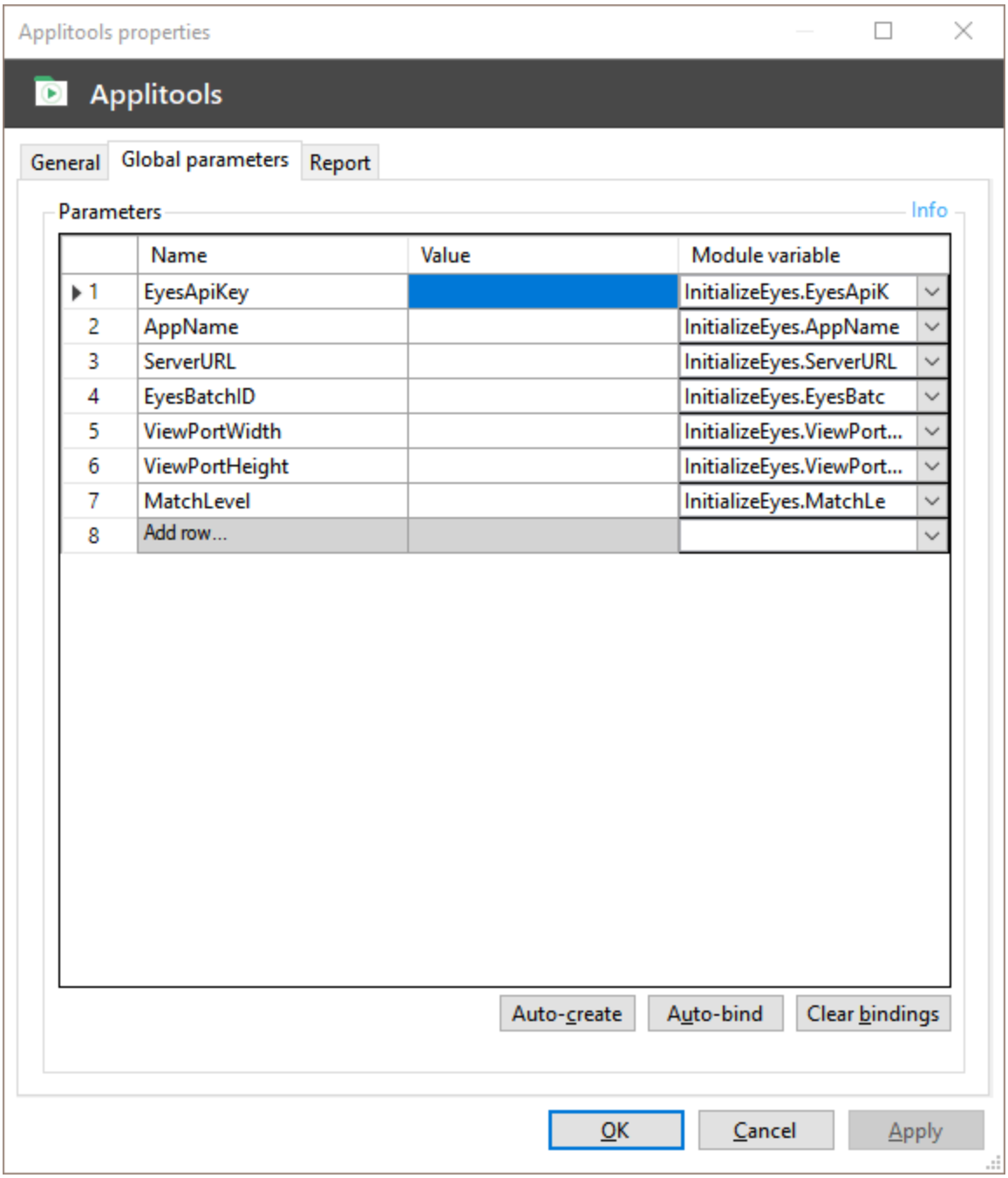
In this chapter, you'll learn how to create tests using the Applitools Eyes integration in Ranorex Studio.

Prepare the test suite

1. In the test suite view, **right-click** the test suite and **click Add setup**. Repeat for **Add teardown**.
2. From the module browser, **drag and drop** the **InitializeEyes** code module to the setup region and the **FinalizeEyes** module to the teardown region.



3. **Right-click** the test suite and **click Global parameters...**
4. In the dialog that appears, **click Auto-create** and then **Auto-bind**.
5. Next to **EyesApiKey**, **enter** your Applitools Eyes API key. If you're using an on-premise installation, enter the URL to the server in **ServerURL**. You can also define other parameters here, but they are optional. See below for an explanation of what they do.



AppName: Enter the name you want the test to be saved under in the Apps & tests category in Eyes.

EyesBatchId: Enter the ID of a test batch to always pass results to this batch.

ViewPortWidth: Define a custom width in pixels for the Applitools Eyes viewport .

ViewPortHeight: Define a custom height in pixels for the Applitools Eyes viewport.

MatchLevel: Define the match level that will be used. Possible values: Exact, Strict, Layout, Content. Default is Strict.

Build your test

You can now build your test as usual and add visual checks by Applitools Eyes to it. This is done in recording modules through **user code methods**.

1. **Open** the recording you want to add a visual check to.
2. **Drag and drop** the repository item you want to check from the repository to the desired position in the action table.
3. In the context menu that appears, **click User code > Select from library**.
4. **Double-click** the **VisualCheckpoint** method.

Note

- You can perform visual checks on **any repository item**, from complete websites to single buttons.
- You can add **as many visual checks as you like**, but each check will take **additional time** and slow down your tests because image data needs to be passed to Applitools Eyes.

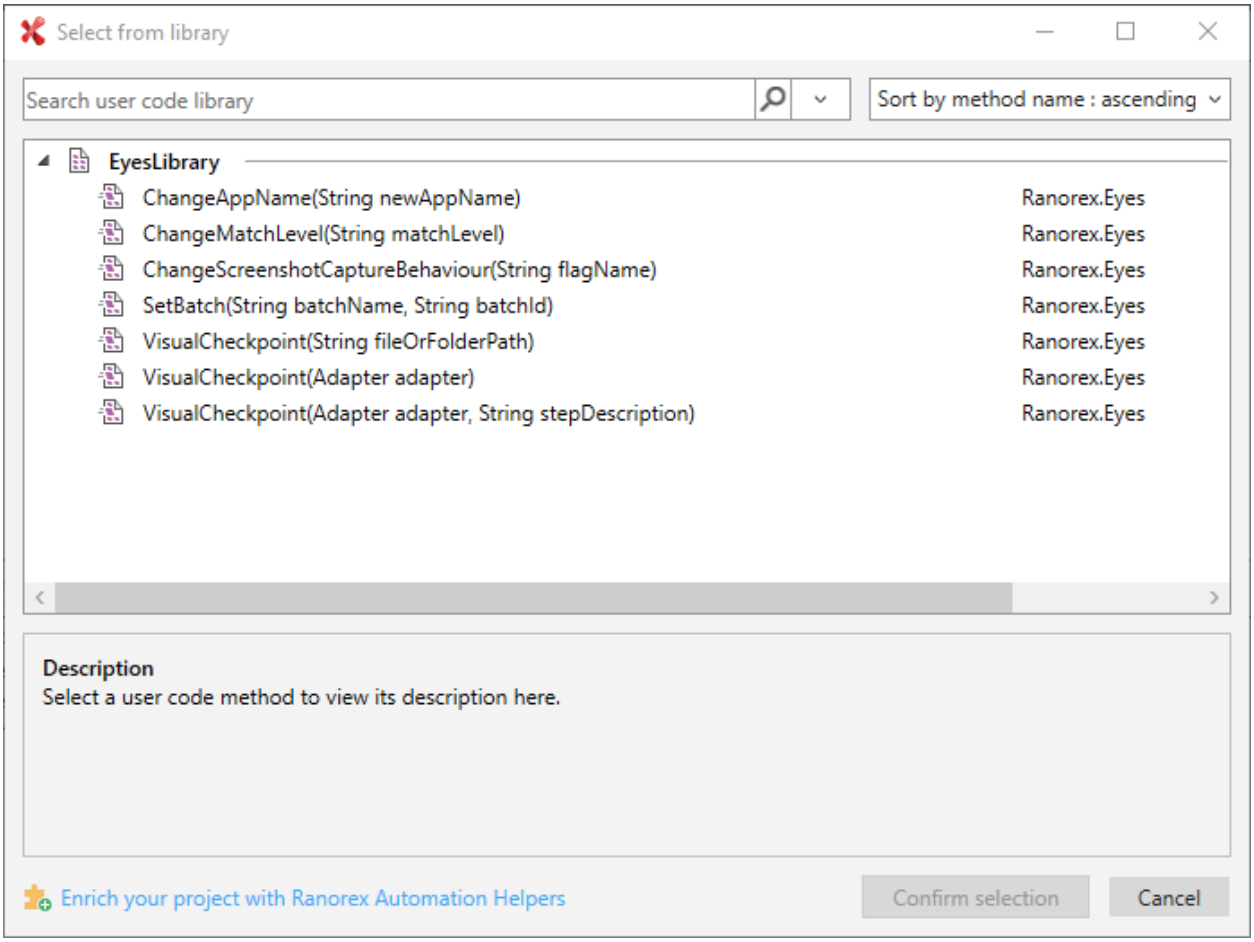
Reference

Find out more about user code methods and the user code library in Ranorex Studio expert > User code library > ⇢ Introduction

Other Applitools user code methods

Aside from VisualCheckpoint, there are several **more user code methods** you can add to a recording module. Use them to pass **custom Applitools Eyes parameters** at runtime. These will overrule global parameters you've set. Refer to the descriptions in Ranorex to see what they do. To add them:

1. **Open** the recording you want to add the method to.
2. In the action table, click **Add new action > User code > Select from library**.
3. **Double-click** the desired user code method.
4. In the action table, **enter** your value for the required argument.



PDF and image file checks

You can also use the Applitools Eyes integration to perform visual regression tests on PDF and image files. There is a special user code method for this purpose.

1. Open the recording you want to add a file check to.
2. In the action table, click **Add new action > User code > Select from library**.
3. **Double-click VisualCheckpoint(String fileORFolderPath)**.
4. In the action table, **enter** the value that points to the file or folder you want to check. The following formats are possible:

- Absolute, e.g. `C:\users\user\documents\MyPDF.pdf` for a single file or `C:\users\user\documents\` for an entire folder.
- Relative, e.g. `MyPDF.pdf` for a single file or `documents\` for an entire folder. Relative paths only work for files and folders in the same location as the test executable.

Run tests

In this chapter, you'll learn about running Applitools Eyes tests in Ranorex Studio.

Run your test

You can run your tests with Applitools Eyes visual checks just like any other test. Simply click **RUN** in the test suite view. During the test run, Ranorex will upload the required image for each visual check to Applitools Eyes. Eyes then passes the results of the check back to Ranorex.

Command line

You can also run your test from the command line. You can pass all global parameters that you've defined as explained in ⇢ Create tests, e.g. `ApplitoolsIntegration.exe -param:viewPortHeight=1080 -param:viewPortWidth=1920`

Reference

Running Ranorex tests from the command line is explained in Ranorex Studio fundamentals > Ranorex test suite > ⇢ Run tests without Ranorex Studio

Attention

The Ranorex option **Use UiaLauncher to elevate privileges for processes started by tools** needs to be disabled for tests with visual checks to run. To do so, first **close** your solution, **go** to **Settings > Advanced**, **uncheck** the option, and **click OK**.

Note

Tests with visual checks may take longer to run depending on the number of visual checks. This is because for each visual check, image data must be passed to Applitools Eyes.

Jenkins Integration

Jenkins is an automation server for continuous integration in software development.

It's available for free under an open-source license at www.jenkins.io. For information on how to set up Jenkins, please follow the official Jenkins documentation.

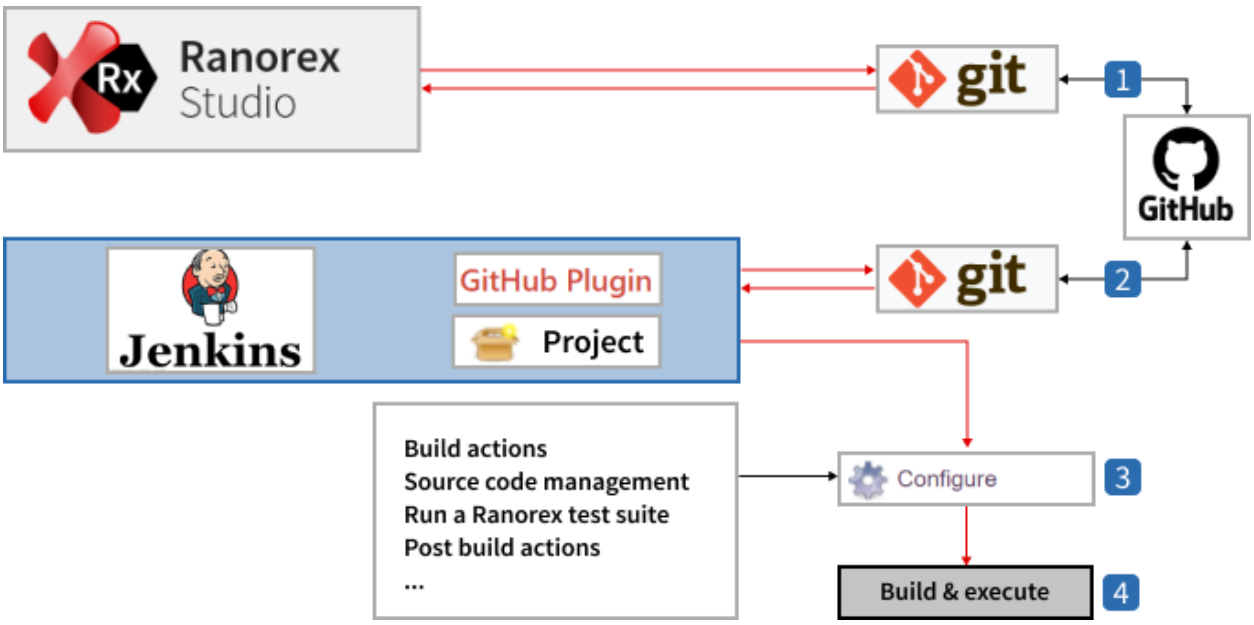
In this chapter, you'll find out how to integrate your Ranorex Studio tests as a build step in Jenkins.

On this introduction page, we explain the basic structure of this integration and which tools you need to set it up. The other pages of this chapter then show the required steps in detail.

Overview

Integrations are always based on making different systems work with each other. The following diagram illustrates a typical structure of Ranorex Studio tests integrated in Jenkins.

> **Note**
>
> While typical, this structure is only an example. You can also use other version control and repository providers that are compatible with Jenkins.



1. The Ranorex Studio solution is under version control with Git and stored and managed in a repository on GitHub
2. The Jenkins server connects to the GitHub repository through a plugin and in this way has access to the Ranorex Studio solution
3. In Jenkins, the build process is set up as a project. It contains all the required build and post-build steps and their configurations. Some of the steps are implemented through Jenkins plugins
4. Once the build process is started, Jenkins carries out the build steps, which means it builds and executes the Ranorex Studio solution, thereby running the test

Software requirements

To set up the process illustrated above, you need the following software.

Ranorex Studio

For creating and maintaining tests, you will need a full installation of Ranorex Studio and a Premium license. For only running a test on a machine with a Jenkins server, you will need a Runtime license.

> **Reference**
>
> Licensing in Ranorex Studio explained in Ranorex Studio system details > ⇢ Licensing

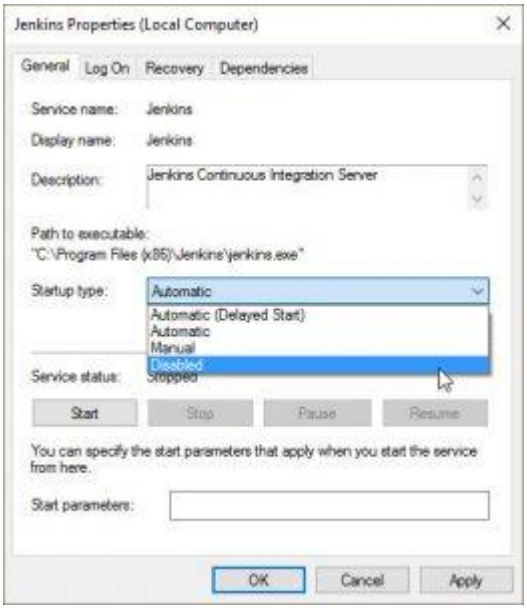
Jenkins

You need a Jenkins server set up and running. For more information about downloading and setting up Jenkins, please refer to the official Jenkins documentation at www.jenkins.io.

Note

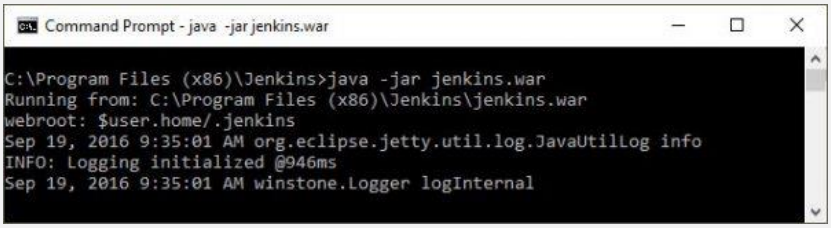
Considering that Jenkins or the nodes executing the CI jobs do not have sufficient rights to start UI-applications, they initialize as a windows service. Ensure Jenkins is disabled as a service on the master or on the slave nodes where Ranorex automation is triggered.

To disable the Jenkins service, **open** the **Services** tool, **right-click** the **Jenkins service**, **select Properties**, and set the **Startup type** to **Disabled**, as shown below:



Use the following command to start Jenkins manually from the installation folder:

```
java -jar jenkins.war
```



After starting Jenkins, use this address to access the web interface: http://localhost:8080/

Git (version control)

The Ranorex Studio solution(s) you want to integrate as a Jenkins build step must be under version control. For the example process described in this chapter, we use Git. It is free under the GNU GPL and you can get it from https://git-scm.com/. You can also use other version control providers that are compatible with Jenkins.

Reference

Putting your Ranorex Studio solution under version control with Git is explained in Interfaces and connectivity > Source control & revision control > ⇢ Git

GitHub (repository provider)

Typically, a free GitHub repository is used to store the files under Git version control online. This is optional, but in the explanations in this chapter, we use GitHub. You can create a GitHub repository for free on www.github.com.

Microsoft Visual Studio

Jenkins builds integrated Ranorex Studio solutions with a Microsoft Visual Studio compiler. You therefore need to have Microsoft Visual Studio installed. The free version is enough to use the integration, but depending on your requirements, you may need to upgrade to a paid version.

Go to https://visualstudio.microsoft.com/ for more information about different versions.

Install and configure plugins

Jenkins uses plugins to integrate functionality of other applications. To integrate Ranorex Studio tests in Jenkins, a series of plugins is required. Other optional plugins add additional functionality. Some of these plugins normally come preinstalled and activated on a Jenkins server, others have to be downloaded and activated.

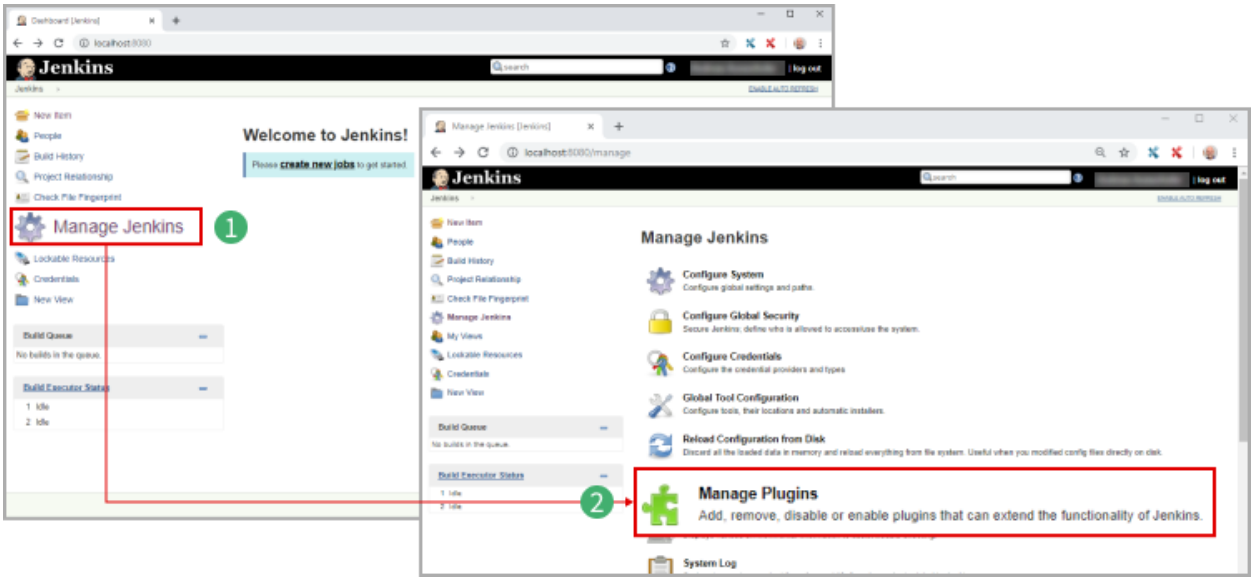
On this page, we list the the required and optional plugins and what configurations you need to make in Jenkins for them to work correctly for the Ranorex Studio integration.

Access plugins

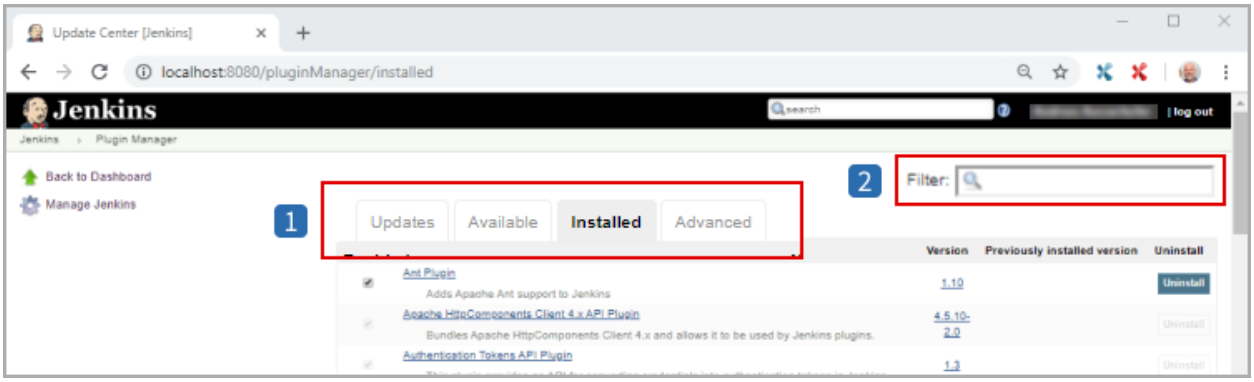
For detailed information on downloading and installing plugins, please refer to this article in the official Jenkins documentation.

To access plugins in Jenkins:

1. **Click Manage Jenkins**.
2. **Click Manage Plugins**.



The following overview appears:



1. Installed plugins are listed under Installed; plugins that can be downloaded are listed under Available
2. The filter allows you to search for plugins

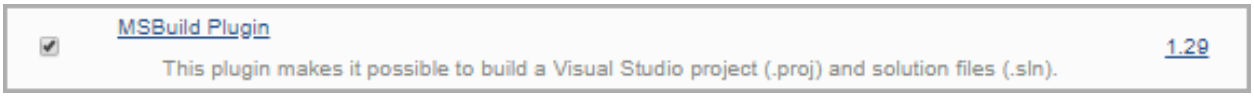
Required and optional plugins

Ranorex Test Execution Plugin

Must be downloaded and installed so Jenkins can process Ranorex Studio tests.

MSBuild Plugin

Must be downloaded and installed so Jenkins can compile and build the Ranorex Studio solution.

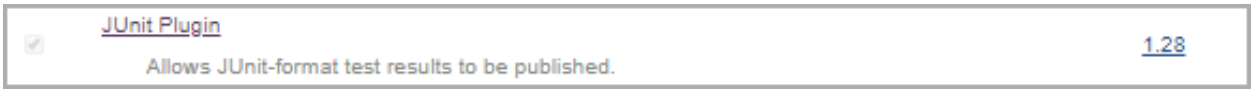


GitHub Plugin

Preinstalled. Optional. Allows Jenkins to access GitHub repositories. We need it because we use GitHub; if you use a different provider, a different plugin may be required.

JUnit Plugin

Preinstalled. Optional. Allows Jenkins to output test results in a report in JUnit format. Required if you want to use the JUnit format for Ranorex Studio reports.

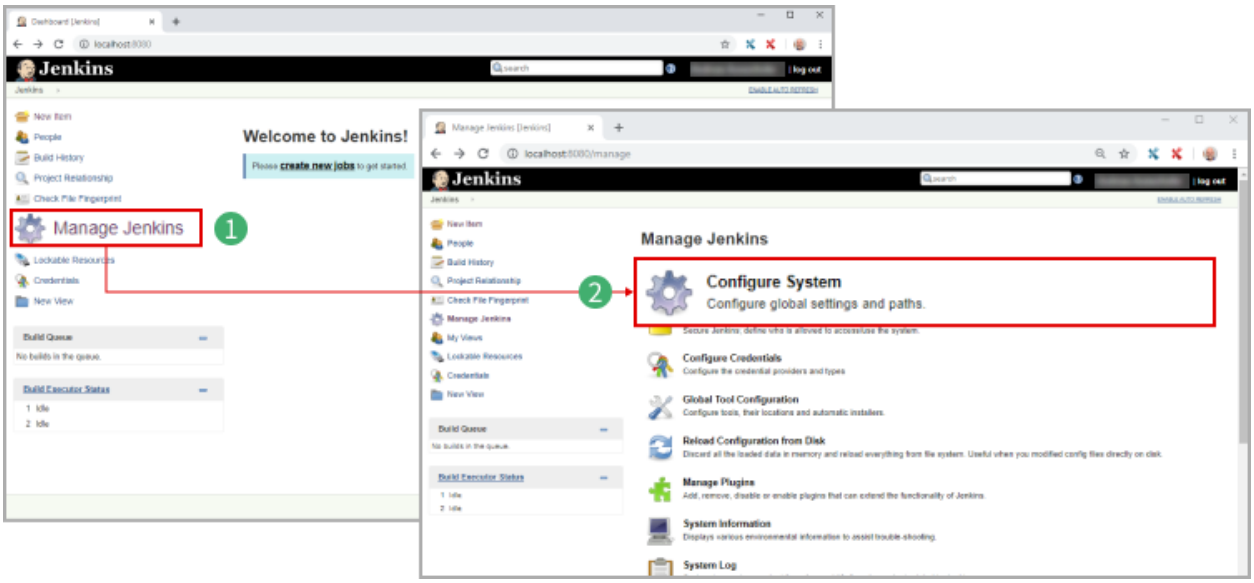


Configure plugins

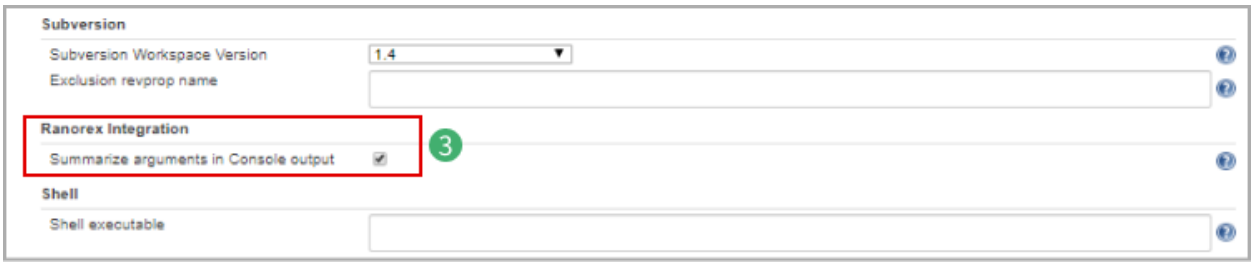
Some of the plugins require that you make some configurations before they will work correctly for the Ranorex Studio integration.

Configure Ranorex Test Execution Plugin

1. **Click Manage Jenkins**.
2. **Click Configure System**.

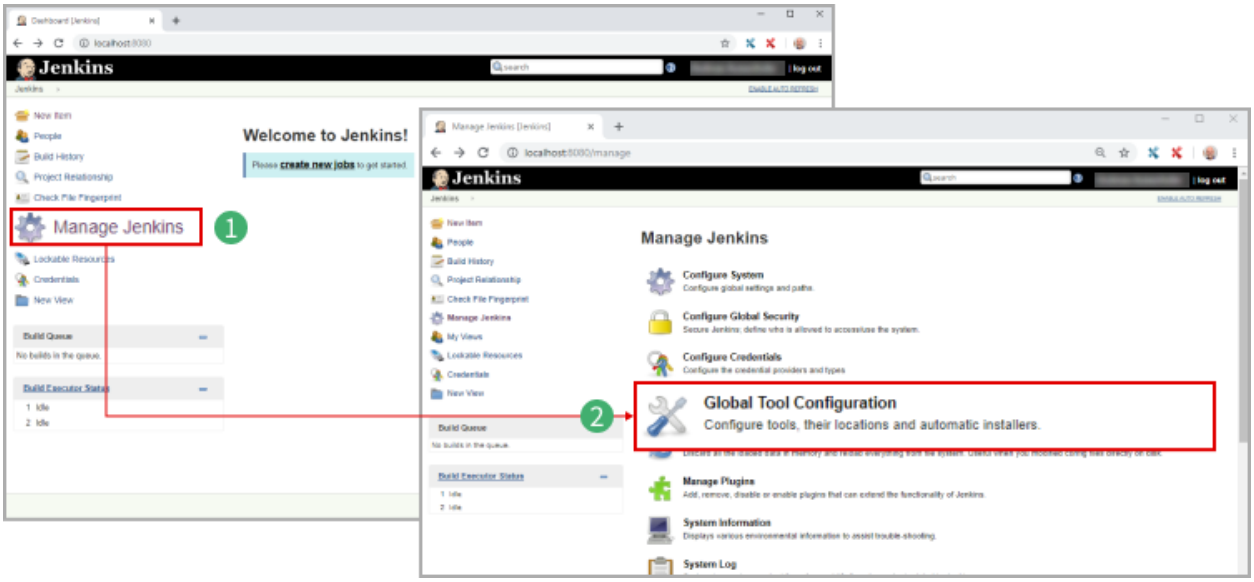


3. Under Ranorex Integration, **check** the option **Summarize arguments in Console output**.
4. **Click Save**.

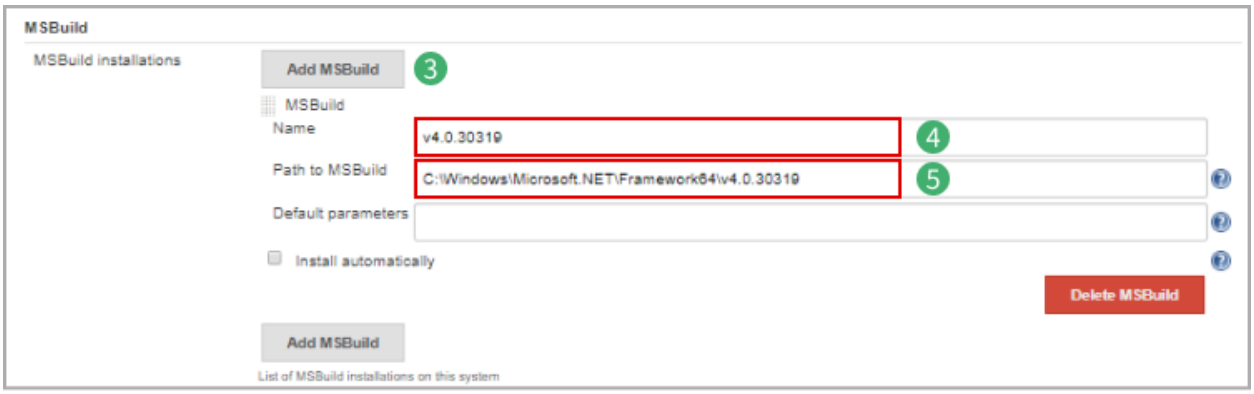


Configure MSBuild Plugin

1. **Click Manage Jenkins**.
2. **Click Global Tool Configuration**.



3. **Click Add MSBuild**.
4. **Enter** a sensible name for the build.
5. **Enter** the path to your local .NET installation.
6. **Click Save**.



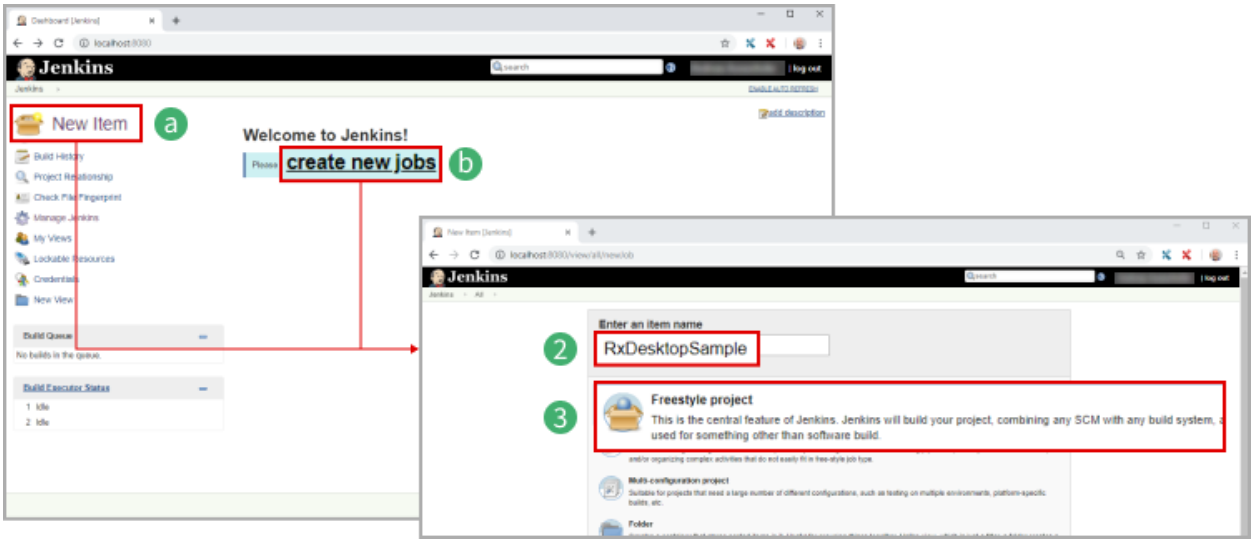
Create and set up a Jenkins project

To run a Ranorex Studio test as a build step in Jenkins, you need to create a project in Jenkins. On this page, you'll find out how to create and configure a Jenkins project, including the settings for the Ranorex Test Execution Plugin.

Create Jenkins project

To create a project in Jenkins:

1. **Click**…
   a. **…New Item.**
   b. **…Create new jobs.**
2. Enter **a project name.**
3. **Select Freestyle project** as template and **click OK**.



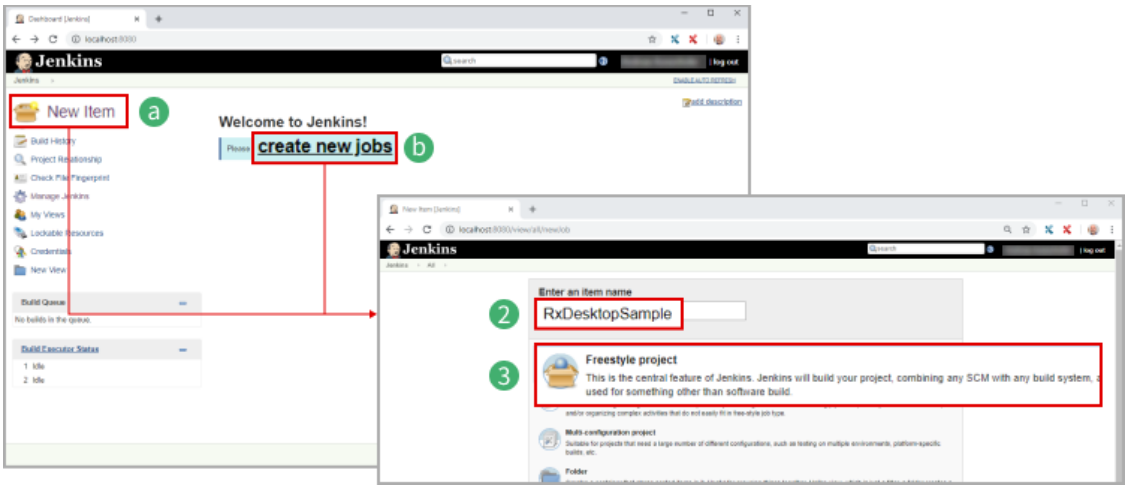
Once Jenkins has created the project, it displays the settings for it.

Configure Jenkins project

There are many ways in which you can configure a Jenkins project. On this page, we focus on the settings relevant to the integration of Ranorex Studio tests.

The project settings are displayed automatically after creating a project. Alternatively, you can also bring them up at any time as follows:

1. With the project open, **click Configure**.

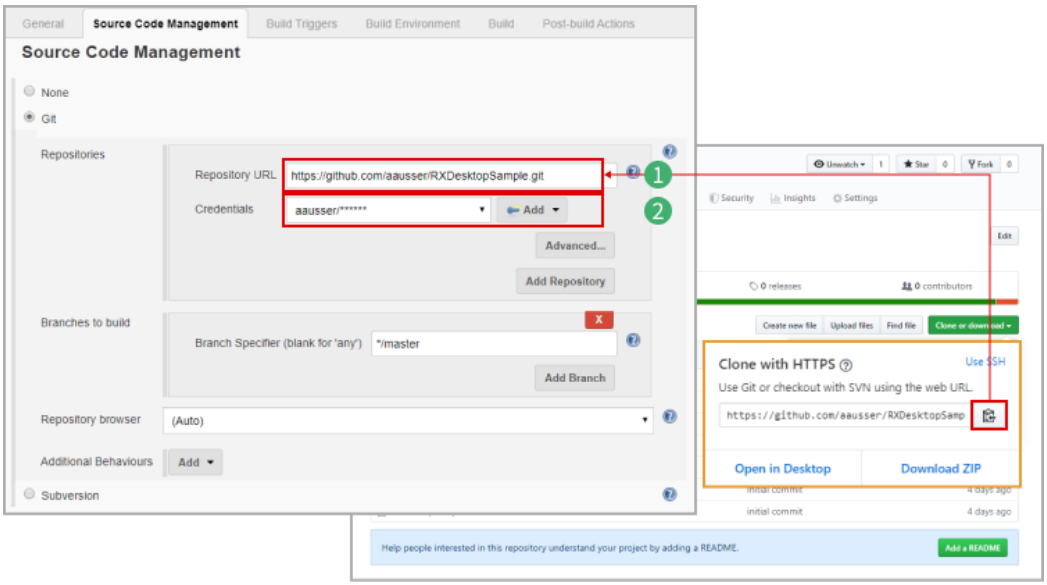


Source Code Management

This is where you configure how Jenkins accesses the repository where your files are under version control.

In our case, we use GitHub, so we make the following configurations:

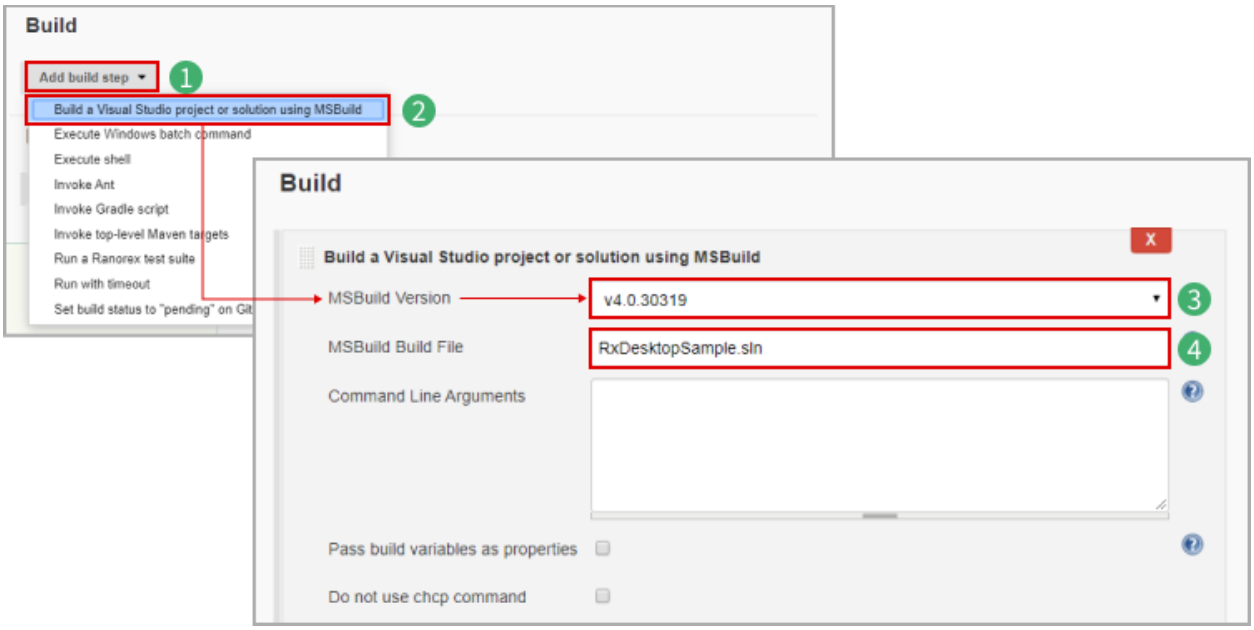
1. Into **Repository URL**, **paste** your GitHub repository's URL.
2. If credentials are necessary to access the repository, **enter** them under **Credentials**.



MSBuild step

You need to add and configure an MSBuild step so that Jenkins will build the Ranorex Studio solution correctly. To do so:

1. **Click Add build step**.
2. **Click Build a Visual Studio project or solution using MSBuild**.
3. **Select** an MSBuild version (defined as part of ⇢ plugin configuration).
4. **Enter** the path to the Visual Studio version (.sln) of your Ranorex Studio solution (.rxsln). The path must be relative to your version control repository's root folder. In our example, the .sln file is located directly in the root folder, so the file name is enough.



Run a Ranorex test suite build step

Finally, we need to add a build step to run the Ranorex Studio test. The following basic configurations are required. Additional settings are explained further below.

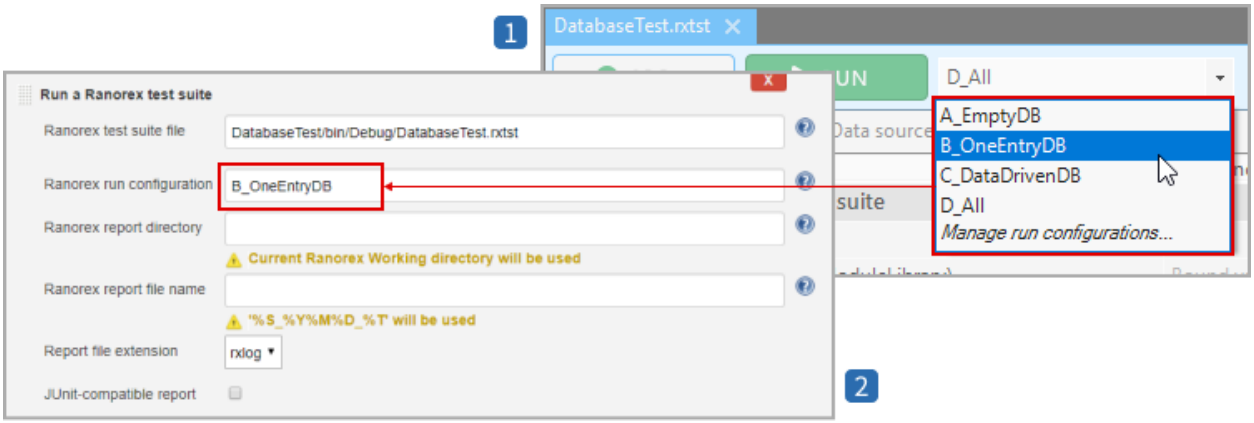
1. **Click Add build step**.
2. **Click Run a Ranorex test suite**.
3. **Enter** the path to the test suite file (.rxtst) relative to the solution folder. By default, the test suite file is located in the output folder, i.e. <solution folder>/bin/Debug/<test suite file>.rxtst

Advanced settings

Aside from the required settings above, the Ranorex test suite build step also contains a range of advanced settings. They can be used to customize the test run. These settings are available from the Advanced… button in the build step.

Ranorex run configuration

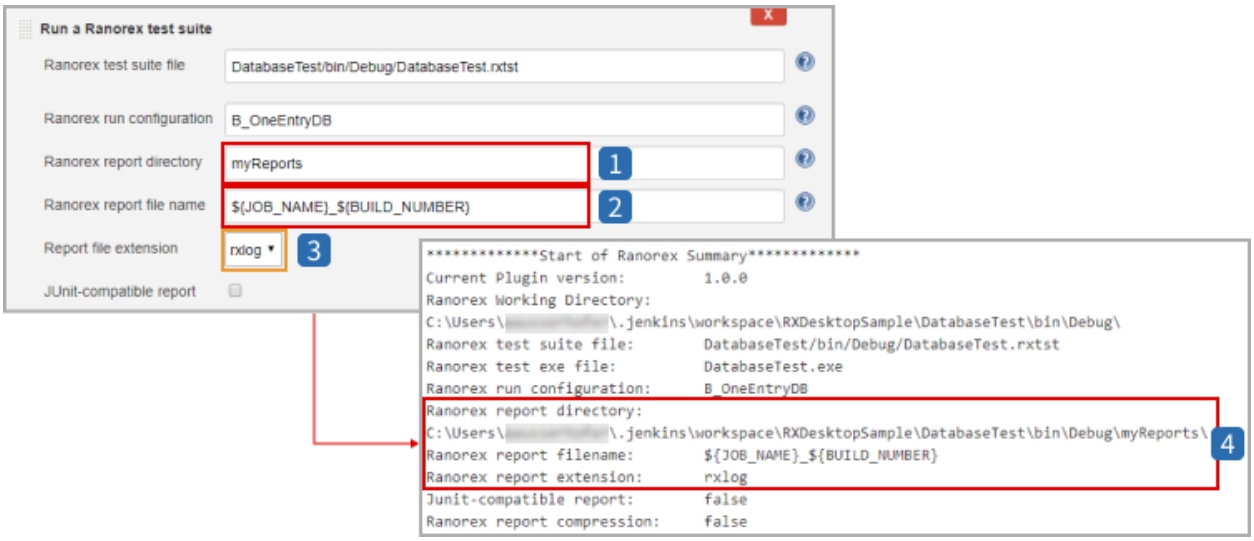
Here you can specify a run configuration. If you don't specify one, the test uses the run configuration that was active when the Ranorex Studio solution was last saved.



1. The available run configurations in the test suite view in Ranorex Studio
2. The desired run configuration needs to be spelled exactly the same way in the Jenkins field

Change report directory, name, and file extension

You can change the default report name and directory in Jenkins. The default name and directory are indicated in yellow below the respective fields if they are left empty. Here's an example:



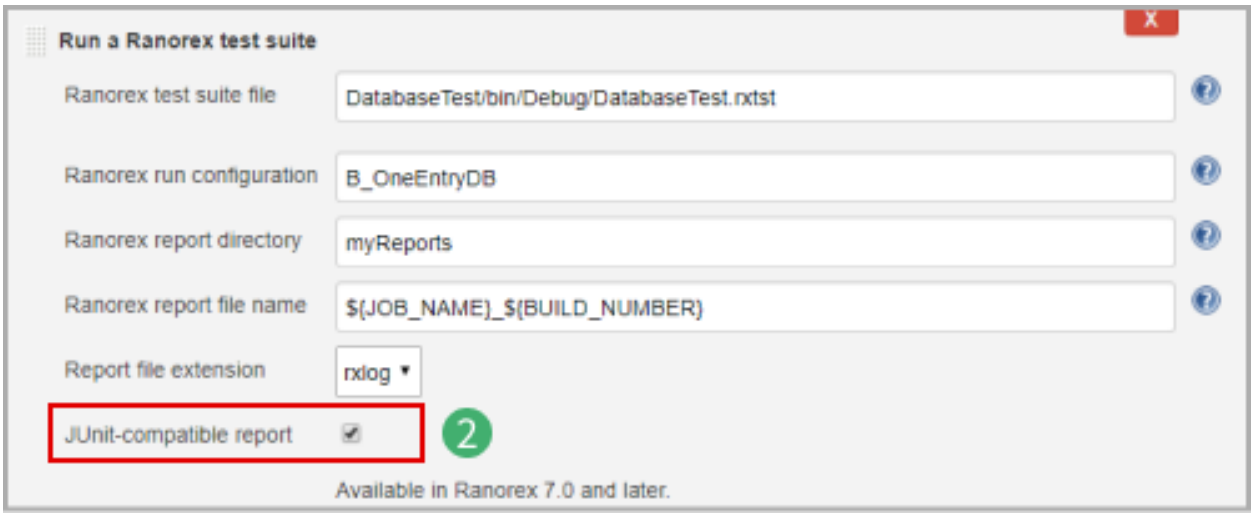
1. Custom report directory. The path must be relative to the output folder (bin/debug) of the Ranorex Studio project
2. Custom report name with Jenkins variables. You can also use the available ⇢ Ranorex Studio report name placeholders
3. File extension selector: rxlog and xml are available
4. The console output when building a project with the above settings

> **Note**
>
> There is a range of variables available in Jenkins that serve as placeholders for various Jenkins parameters. You can use these in file names, for example. For an overview of the available variables, please refer to this page in the official Jenkins documentation.

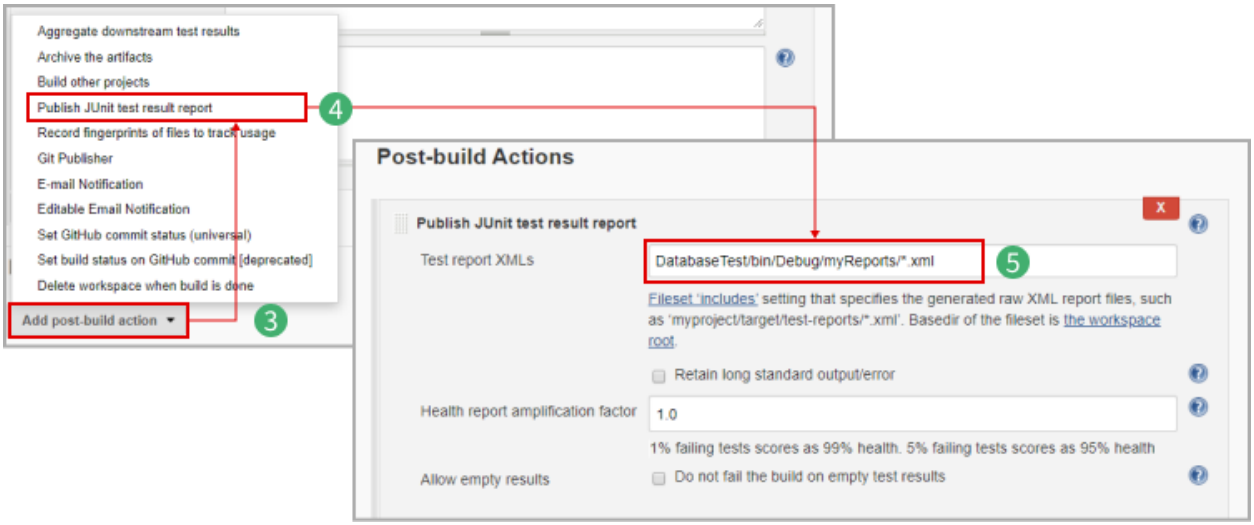
JUnit-compatible report

The JUnit data format is used by many CI systems to interpret test result data. When activated, this setting adds a copy of the report in JUnit format.

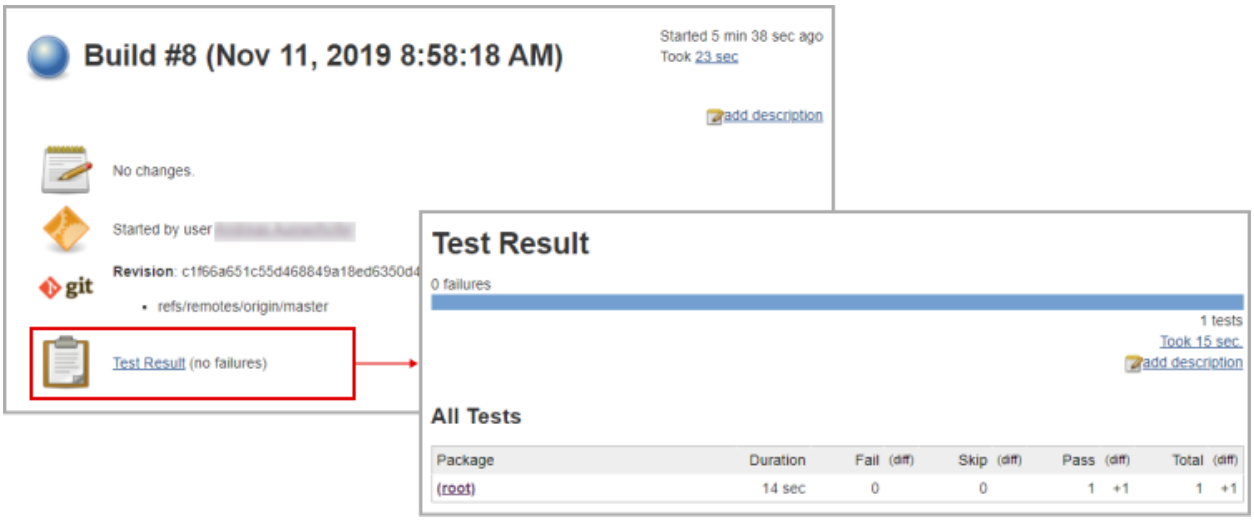
1. **Ensure** you have the Jenkins JUnit plugin installed and actived, as explained in ⇢ Install and configure plugins.
2. **Activate** the JUnit-compatible report.



3. In the Jenkins project, **click Add post build action**.
4. **Click Publish JUnit test result report**.
5. Under **Test report XMLs**, **enter** the Ranorex Studio report directory.



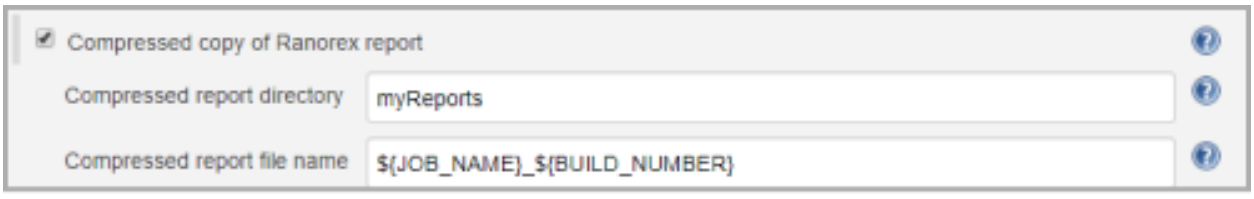
When you've run a build, the build summary will contain a link to the JUnit report.



Compressed copy of Ranorex report

To create a compressed copy of the report:

1. **Activate** the compressed report copy.
2. **Enter** a custom directory if desired. The path must be relative to the output folder (bin/debug) of the Ranorex Studio project. If you leave the field empty, the default report directory is used.
3. **Enter** a custom name for the compressed report if desired. You can use Jenkins and Ranorex Studio variables.



Note

There is a range of variables available in Jenkins that serve as placeholders for various Jenkins parameters. You can use these in file names, for example. For an overview of the available variables, please refer to this page in the official Jenkins documentation.

Ranorex TestRail integration

These settings allow you to pass test results to TestRail. They correspond to the command line parameters for the TestRail integration.

Reference

The command line parameters for the TestRail integration are explained in

Ranorex Studio expert > Runtime and remote execution > ⇢ Command line execution

The TestRail integration itself is explained in

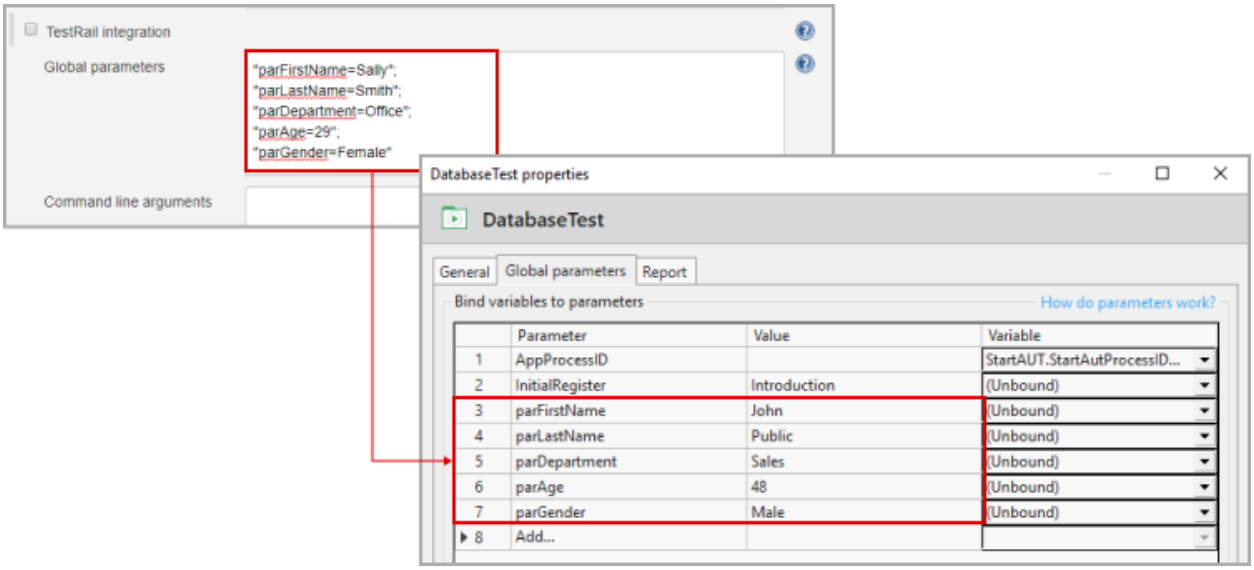
Interfaces and connectivity > ⇢ TestRail integration

Global parameters

If the test suite contains global parameters, you can define values for them here. They will then override any existing values for these parameters when the build runs.

To define values:

1. **Enter** parameters and values according to the following pattern: “ParameterName=Value”. **Separate** parameters with semicolons.



Command line arguments

In this field, you can pass any command line arguments available in Ranorex Studio.

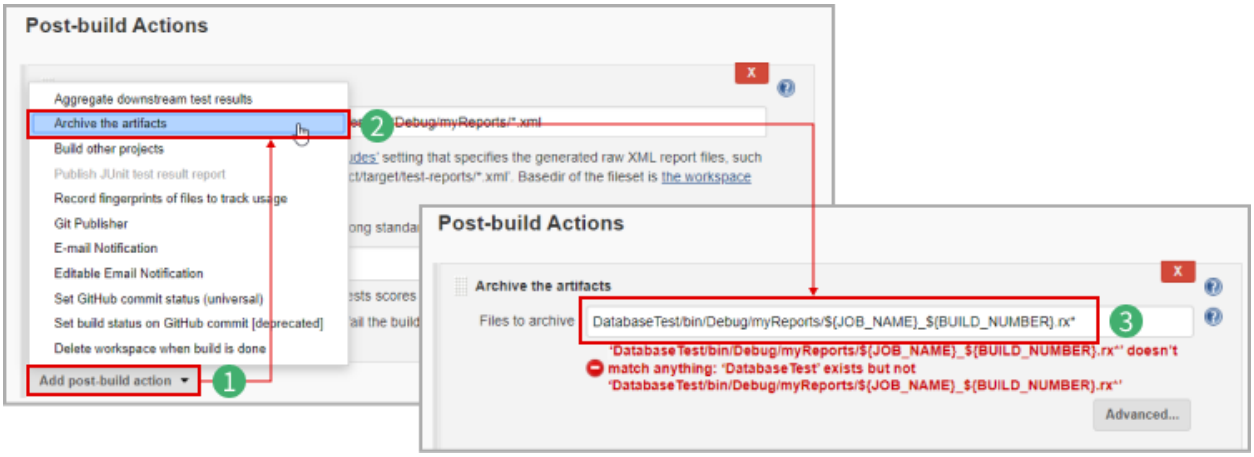
> **Reference**
>
> All command line arguments are explained in
>
> Ranorex Studio expert > Runtime and remote execution > ⇢ Command line execution

Archive build artifacts

Build artifacts are files that are created as a result of the build, i.e. the executable build of the test and the report files. You can make these artifacts available for download from Jenkins, which is more convenient than hunting for the files in the respective directories.

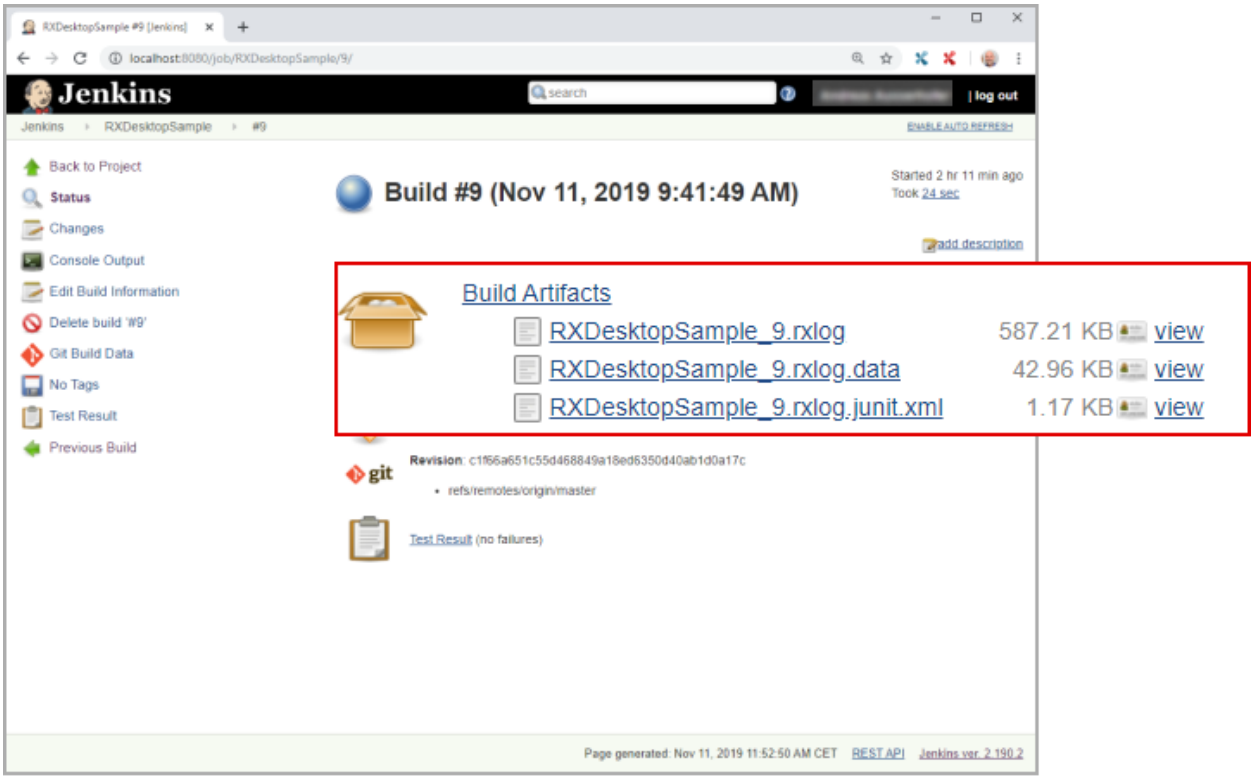
To do so:

1. In the Jenkins project, **click Add post build action**.
2. **Click Archive the artifacts**.
3. **Specify** the files you want to archive.



In the image, the post-build step archives all files that are in the directory /myReports/, contain the Jenkins build name and number, and whose file ending contains “rx”.

After the build has executed, the artifacts are available for download as shown below:

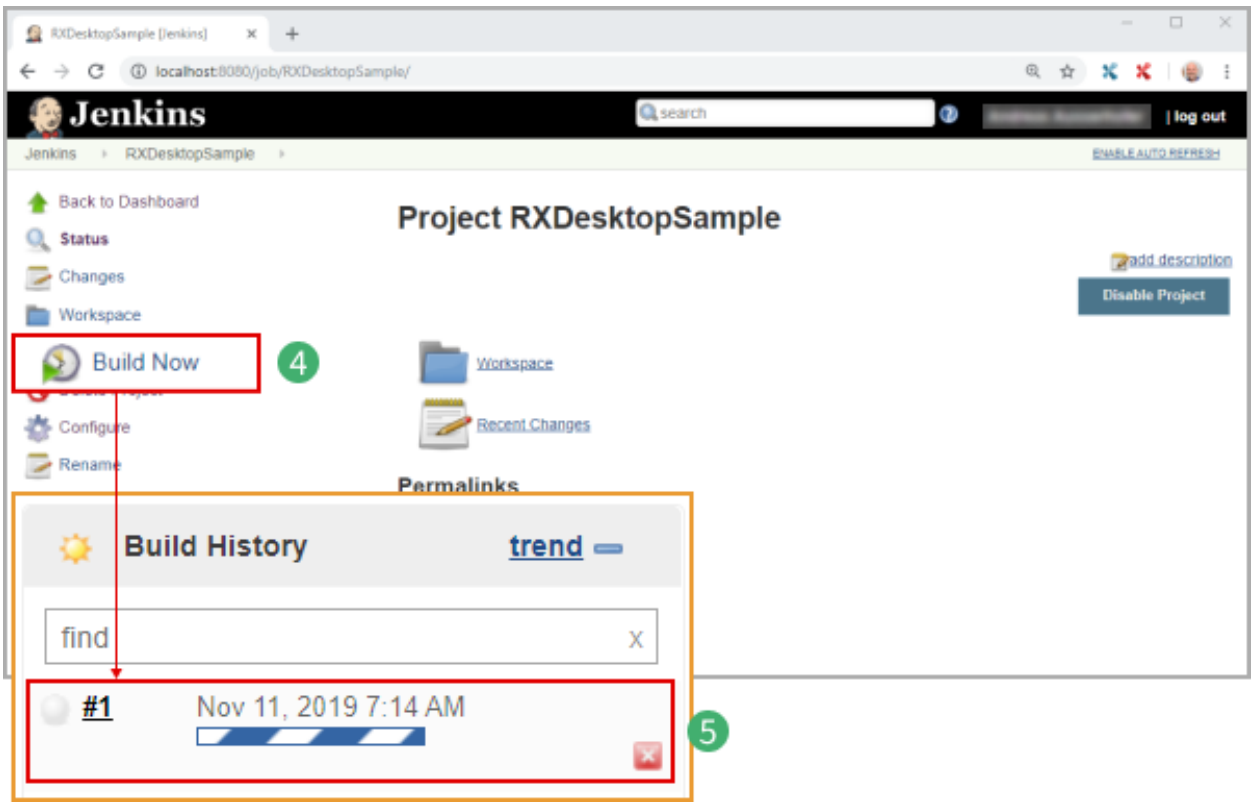


Build a Jenkins project

To execute a Ranorex Studio test as part of a Jenkins process, you need to build the Jenkins project that contains the test. Find out how to do so on this page.

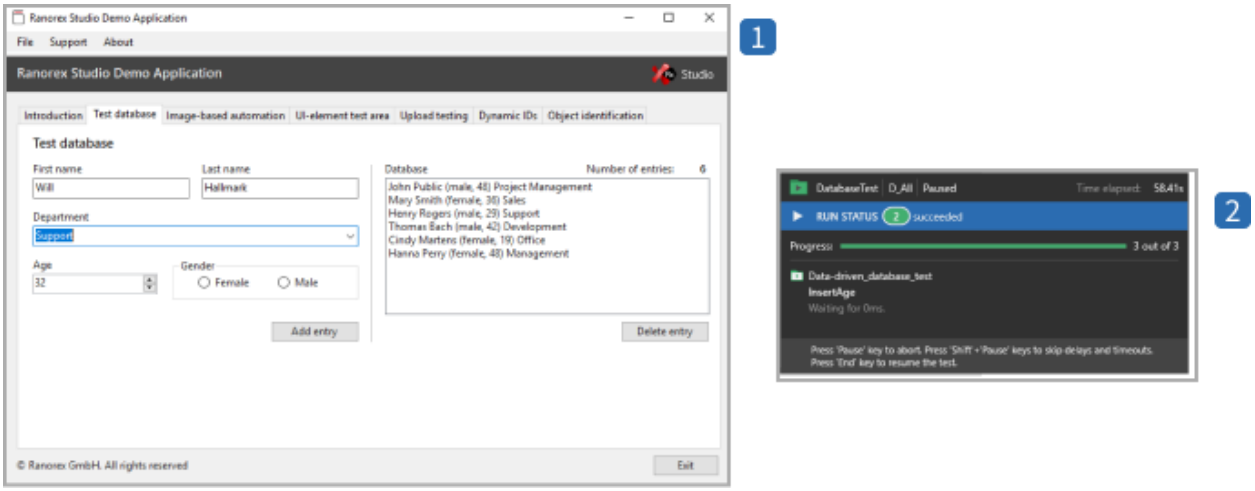
Build the project

1. ⇢ **Install** and **configure** plugins.
2. ⇢ **Create** and **configure** a Jenkins project.
3. With the project open, **click Build now**.
4. **Wait** as Jenkins builds the project.



Test execution

While the project is building, the Ranorex Studio test executes and the progress dialog appears.



1 Ranorex Studio test being executed

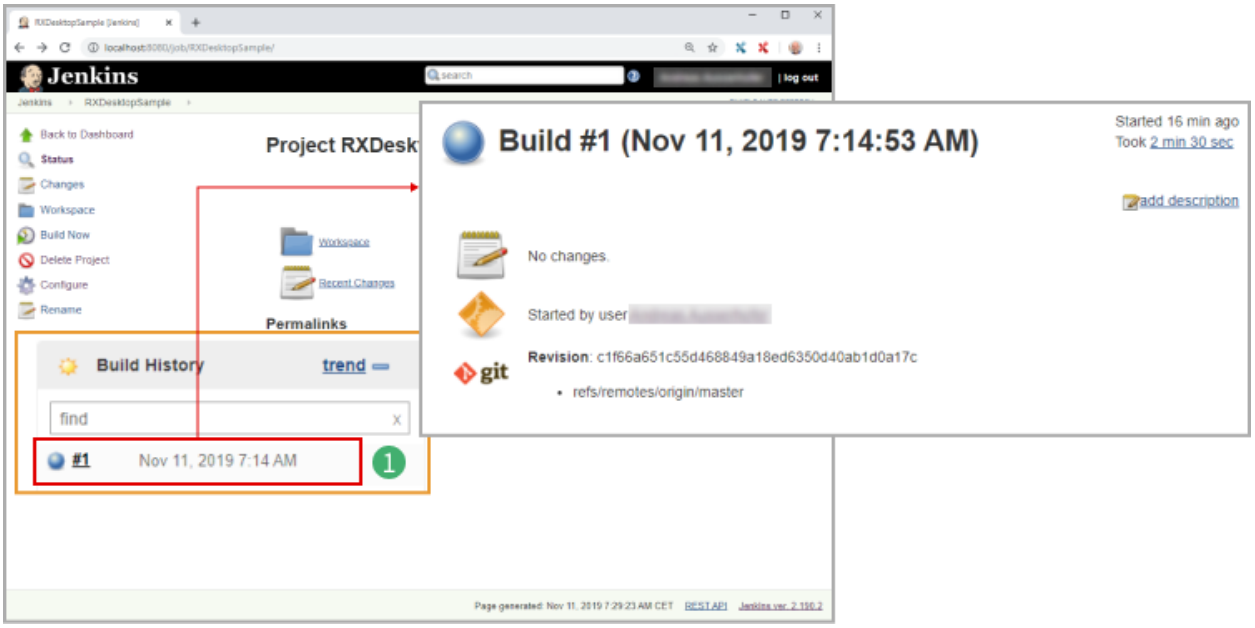
2 Progress dialog

Jenkins build details

Once the build has finished and the test executed, Jenkins returns to the project view and the build appears in the project's build history.

To display build details:

1. Under **Build History**, **click** the respective build.

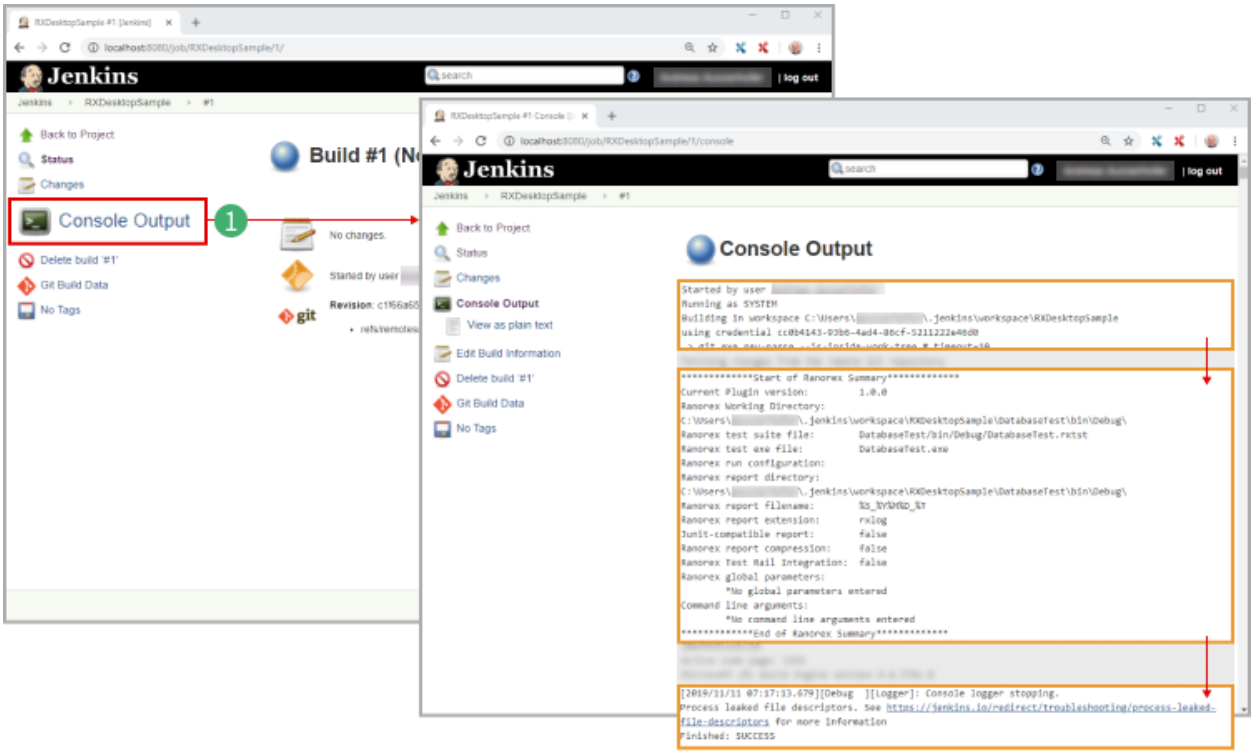


Console output

Jenkins also saves build details in a console output file.

To display it:

1. While viewing a build's details, **click Console Output**.



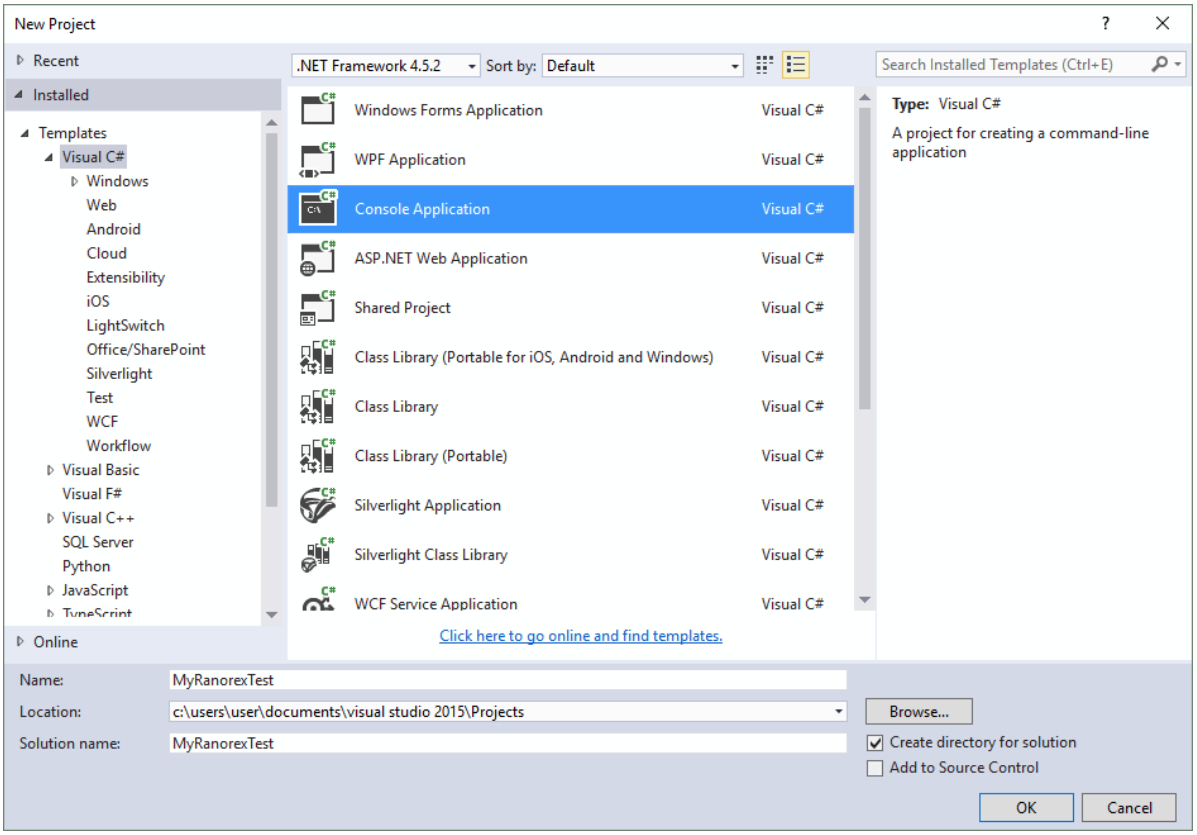
Visual Studio integration

Microsoft Visual Studio is an integrated development environment (IDE) by Microsoft. It is used to develop computer programs for Microsoft Windows, as well as websites, web apps, web services and mobile apps. This chapter introduces and explains how to use Ranorex Studio in a simple Visual Studio C# console application. It shows how to create a new Visual Studio C# console application and how to start and automate the Windows Calculator.

> **Note**
>
> The sample in this chapter works with with Microsoft Visual Studio 2005 and later.

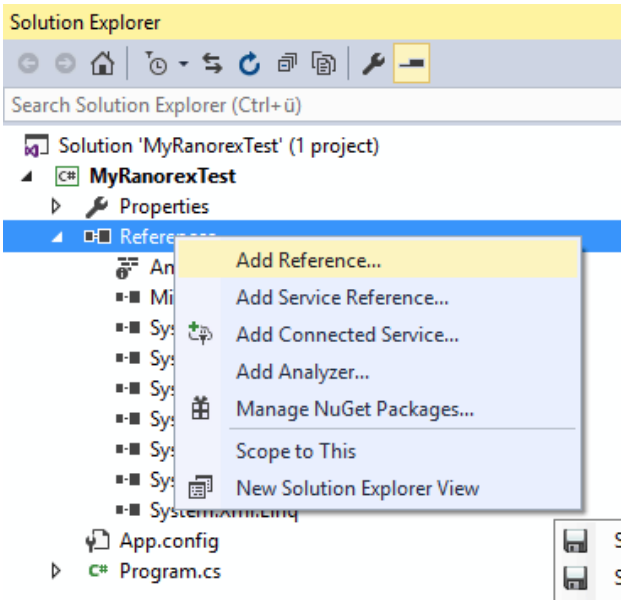
Create a new Visual Studio project

1. Start Microsoft Visual Studio.
2. Go to **File > New Project**.
3. Select .NET Framework 4.5.2 or higher (excluding .NET Core) and choose your programming language. We will be using Visual C# in this example.
4. Select **Console Application** from the list.
5. Enter a name for the project and click **OK**.

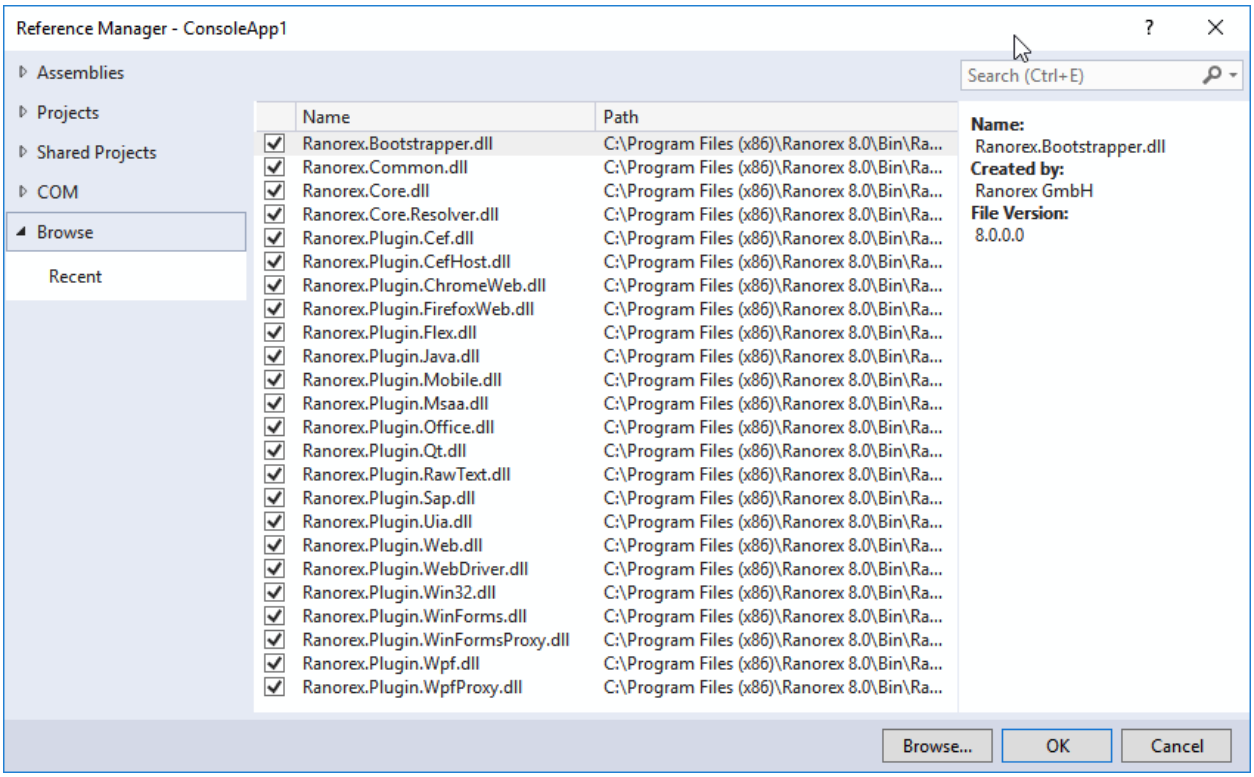


Add Ranorex core assemblies as references

1. In the project's **Solution Explorer**, right click the **References** folder and select **Add Reference…**



2. Select **Browse** in the menu on the left.
3. Browse to the Bin folder of your Ranorex installation (default: C:Program Files (x86)Ranorex Bin).
4. Add Ranorex.Bootstrapper, Ranorex.Common, Ranorex.Core, Ranorex.Core.Resolver, and all Ranorex.Plugin assemblies.



Write some Ranorex automation code

Before you start writing code, we recommend you set the **Copy Local** option to **False** for all Ranorex assemblies **except for Ranorex.Core.Resolver**. This assembly **must always be set to *True***. You can leave **Copy Local** enabled for other assemblies if your solution requires it.

Now, to prepare the example in this section, open the file 'Program.cs' and add the following 'using' statement to your existing using section:

C#

```
using Ranorex;
using System;
using System.Runtime.CompilerServices;
using System.Threading;
```

VB.NET

```
Imports Ranorex
Imports System;
Imports System.Runtime.CompilerServices;
Imports System.Threading;
```

It is crucial that the **Ranorex.Core.Resolver assembly is initialized before any other Ranorex core functionalities** are used. It is the assembly that finds all other Ranorex assemblies at runtime.

To do so, add the following code to the *Main* routine of the class *Program*. Your test automation code goes in the *Run* section. In our example, we perform a simple calculation in the Windows calculator. Once you've added the code, press F5 to build and start your project.

C#

```
static void Main(string[] args)
{
    InitResolver();
    RanorexInit();
    run();
}
[MethodImpl(MethodImplOptions.NoInlining)]
private static void InitResolver()
{
    Ranorex.Core.Resolver.AssemblyLoader.Initialize();
}

[MethodImpl(MethodImplOptions.NoInlining)]
private static void RanorexInit()
{
    TestingBootstrapper.SetupCore();
}

[MethodImpl(MethodImplOptions.NoInlining)]
private static int run()
{
    int error = 0;
    //Start calculator and wait for UI to be loaded
    try
    {
        System.Diagnostics.Process pr = System.Diagnostics.Process.Start("calc.ex
e");
        Thread.Sleep(2000);
```

```
        //Get process name
        string processName = GetActualCalculatorProcessName();

        //Find Calculator | Windows 10
        if (IsWindows10())
        {
            WindowsApp calculator = Host.Local.FindSingle<WindowsApp>("winapp[@pr
ocessname='" + processName + "']");

            Button button = calculator.FindSingle<Ranorex.Button>(".//button[@aut
omationid='num2Button']");
            button.Click();

            button = calculator.FindSingle<Ranorex.Button>(".//button[@automation
id='plusButton']");
            button.Click();

            button = calculator.FindSingle<Ranorex.Button>(".//button[@automation
id='num3Button']");
            button.Click();

            button = calculator.FindSingle<Ranorex.Button>(".//button[@automation
id='equalButton']");
            button.Click();

            //Close calculator
            calculator.As<Form>().Close();
        }
        //Find Calculator | Windows 8.X or older
        else
        {
            Form calculator = Host.Local.FindSingle<Form>("form[@processname='" +
 processName + "']");

            calculator.EnsureVisible();

            Button button = calculator.FindSingle<Ranorex.Button>(".//button[@con
trolid='132']");
            button.Click();

            button = calculator.FindSingle<Ranorex.Button>(".//button[@controlid=
'92']");
            button.Click();
```

```
            button = calculator.FindSingle<Ranorex.Button>(".//button[@controlid='133']");
            button.Click();

            button = calculator.FindSingle<Ranorex.Button>(".//button[@controlid='121']");
            button.Click();

            //Close calculator
            calculator.Close();
        }
    }
    catch (RanorexException e)
    {
        Console.WriteLine(e.ToString());
        error = -1;
    }

    return error;
}
private static string GetActualCalculatorProcessName()
{
    string processName = String.Empty;
    var processes = System.Diagnostics.Process.GetProcesses();

    foreach (var item in processes)
    {
        if (item.ProcessName.ToLowerInvariant().Contains("calc"))
        {
            processName = item.ProcessName;
            break;
        }
    }

    return processName;
}

private static bool IsWindows10()
{
    var reg = Microsoft.Win32.Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\Windows NT\CurrentVersion");

    string productName = (string)reg.GetValue("ProductName");

    return productName.StartsWith("Windows 10");
```

```
}
```

For reference purposes, this is what the code looked like for Ranorex versions before 8.0.

C#

```
 [STAThread]
static int Main(string[] args)

int error = 0;

try
{
    System.Diagnostics.Process pr = System.Diagnostics.Process.Start("calc.exe");

    Form form = Host.Local.FindSingle <ranorex.form>("form[@processname='"+pr.ProcessName+"']");
    form.Activate();

    Button button = form.FindSingle<ranorex.button>(".//button[@controlid='132']");
    button.Click();

    button = form.FindSingle<ranorex.button>(".//button[@controlid='92']");
    button.Click();

    button = form.FindSingle<ranorex.button>(".//button[@controlid='133']");
    button.Click();

    button = form.FindSingle<ranorex.button>(".//button[@controlid='121']");
    button.Click();
}
catch (RanorexException e)
{
    Console.WriteLine(e.ToString());
    error = -1;
}

return error;

}
```

VB.NET

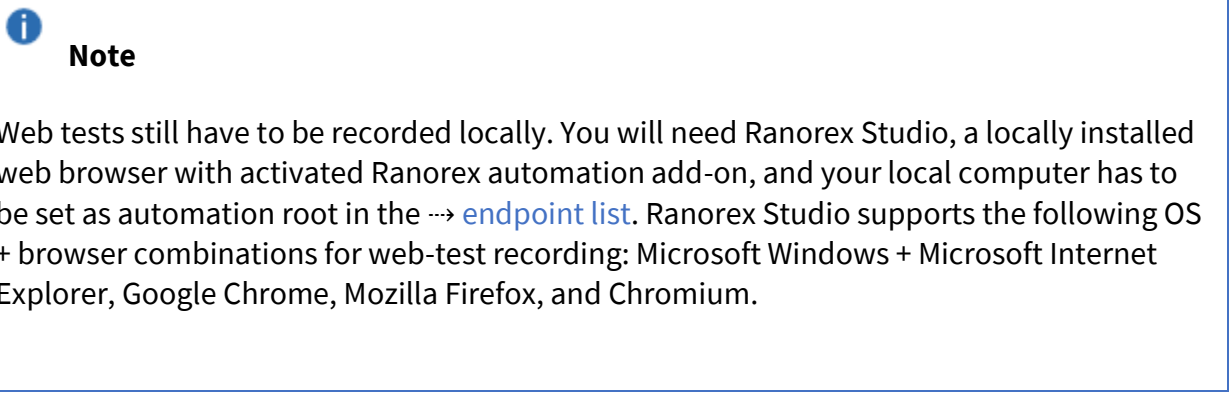
```
<stathread> _
Public Shared Function Main(args As String()) As Integer

Dim returnError As Integer = 0

Try
  Dim pr As System.Diagnostics.Process = System.Diagnostics.Process.Start("calc.e
xe")
  Dim form As Form = Host.Local.FindSingle(Of Ranorex.Form)("form[@processname='"
 & pr.ProcessName & "']")
  form.Activate()

  Dim button As Button = form.FindSingle(Of Ranorex.Button)(".//button[@controlid
='132']")
  button.Click()

  button = form.FindSingle(Of Ranorex.Button)(".//button[@controlid='92']")
  button.Click()

  button = form.FindSingle(Of Ranorex.Button)(".//button[@controlid='133']")
  button.Click()

  button = form.FindSingle(Of Ranorex.Button)(".//button[@controlid='121']")
  button.Click()
Catch e As RanorexException
  Console.WriteLine(e.ToString())
  returnError = -1
End Try
Return returnError
```

Selenium WebDriver integration

With the Selenium WebDriver integration, you can run web tests created with Ranorex on different browsers, operating systems, and machines without any plug-ins or add-ons.

Ranorex uses the existing Selenium WebDriver infrastructure to **run web tests** on:

- Microsoft Internet Explorer, Microsoft Edge, Google Chrome, Mozilla Firefox, and Chromium on Microsoft Windows
- Apple Safari, Google Chrome, and Mozilla Firefox on Apple macOS
- Google Chrome and Mozilla Firefox on Linux



Note

Web tests still have to be recorded locally. You will need Ranorex Studio, a locally installed web browser with activated Ranorex automation add-on, and your local computer has to be set as automation root in the ⇢ endpoint list. Ranorex Studio supports the following OS + browser combinations for web-test recording: Microsoft Windows + Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, and Chromium.

Quick start guide for Selenium WebDriver integration in Ranorex

1. ⇢ Set up a Selenium WebDriver infrastructure.
2. ⇢ Start Selenium Standalone Server.
3. ⇢ Add a WebDriver endpoint to Ranorex Studio.
4. Record your web test on your local machine or use an existing one.
5. Review the web test following these ⇢ guidelines.
6. Set the WebDriver endpoint as ⇢ automation root.
7. Run the test suite.

Setting up a Selenium WebDriver infrastructure

This chapter only provides a basic introduction. For complete documentation of the Selenium WebDriver infrastructure, please visit the official Selenium web site.

For a basic setup, you will need:

- The Java Runtime Environment
- A Selenium Server Download the latest version of Selenium Standalone Server and place it into a folder.
- Drivers for the web browser you want Selenium Server to automate.

All related links and setup steps can be found here.

Download the correct driver for your platform, unpack if needed, and place into the same folder as Selenium Standalone Server.

> **Note**
>
> The InternetExplorerDriver requires several settings to be made before it will work correctly.

Running Selenium Server

Before adding an endpoint or running a test you need to start the server.

Open a command line console and switch to the folder containing Selenium Standalone Server. Execute the following command:

```
java -jar selenium-server-standalone-.jar
```

Replace “” with the version number of the downloaded Selenium Standalone Server file. The window has to remain open.

Web test guidelines

Selenium WebDriver differs from the web testing capabilities that Ranorex offers for desktop-based web tests. After recording a web test on your local web browser, follow these steps to make it compatible for execution with Selenium WebDriver:

- Remove all actions and repository items which interact with the web browser application itself.
- Use the ‘Close Application’ action for closing the web browser.

- Please also refer to the Ranorex 7.0 release notes. They contain important information about the Selenium WebDriver integration.

Plugings

Improved WPF plugin

With Ranorex 7.0, we've introduced a new and improved WPF plug-in. It produces less complex element trees and more logical RanoreXPaths, making your tests more reliable and robust. This is why we recommend you **use the improved WPF plug-in for all your new projects**. It is the default setting for new solutions.

Your existing projects that use the WPF plug-in from previous Ranorex versions **will still work in Ranorex 7.0**. Ranorex automatically activates the correct plug-in for you based on whether:

- the solution contains any WPF elements,
- any solution settings are present,
- you performed a fresh installation of Ranorex 7.0 or upgraded from an existing installation.

This following table illustrates this. You can find a more detailed explanation of all the cases further below.

	Existing solution		New solution
	with solution settings	without solution settings	
Fresh installation			
contains WPF elements	use solution settings, no changes	create solution settings, use legacy plug-in with default settings	n/a
doesn't contain WPF elements	use solution settings and improved plug-in	use current local settings and improved plug-in	use improved plug-in
Upgrade			
contains WPF elements	use solution settings, no changes	create solution settings, use legacy plug-in with previous local settings	n/a
doesn't contain WPF elements	use solution settings and improved plug-in	use current local settings and improved plug-in	use improved plug-in

Local settings: ⇢ User settings and solution settings stored on the local machine.

Solution settings: Solution settings stored in a solution.

Legacy plug-in: The WPF plug-in for previous versions of Ranorex.

How to read the table:

You performed a fresh installation of Ranorex 7.0. You are opening an existing solution that contains WPF elements and solution settings: Ranorex will use the existing solution settings and make no changes.

Fresh installation

When you create a new solution, the improved plug-in is activated by default.

When you open an existing solution created in a previous version of Ranorex, its repositories are first scanned for WPF elements:

WPF elements are found

- There are solution settings: No changes are made. The existing solution settings apply.
- There are no solution settings: Ranorex automatically generates the solution settings. The legacy WPF plug-in for previous versions is activated with the default settings.

No WPF elements are found

- There are solution settings: The improved WPF plug-in is activated, all other existing solution settings apply.
- There are no solution settings: The improved WPF plug-in is activated and the current local settings are used.

Upgrade from an existing installation

If you upgraded from an existing installation, Ranorex 7.0 will use two separate local-settings files when determining the correct WPF settings for you. The first file is the old local-settings file of your previous installation. For Ranorex 6, this file would be called RanorexConfig6.xml. The second file is created when you start Ranorex 7.0 for the first time. It is a copy of the old local-settings file, renamed to RanorexConfig7.xml and configured to use the improved WPF plug-in by default.

When you create a new solution, RanorexConfig7.xml is used and the improved WPF plug-in is activated by default.

When you open an existing solution created in a previous version of Ranorex, its repositories are first scanned for WPF elements:

WPF elements are found

- There are solution settings: No changes are made. The existing solution settings apply.
- There are no solution settings: Ranorex automatically generates the solution settings. The legacy WPF plug-in is activated and its settings from the old local-settings file (e.g. RanorexConfig6.xml) are applied to the solution settings.

No WPF elements are found

- There are solution settings: The improved WPF plug-in is activated, all other existing solution settings apply.
- There are no solution settings: The improved WPF plugin is activated and the current local settings are used.

Attention – Ranorex standalone tools

In the standalone tools, (Ranorex Spy, Ranorex Recorder, etc.), the improved WPF plug-in is active by default. If you are working with a solution that doesn't use the improved WPF plug-in, you must activate the correct WPF plug-in in the standalone tool you want to use.

> **Attention**
>
> Do not mix repository items created in different WPF-plug-in operation modes. It will lead to conflicts.

Follow these instructions to activate the correct WPF plug-in:

First, find out what WPF settings your solution is using:

1. With a solution open, click on 'Settings' in the toolbar.
2. Click on the 'Plugins' tab, select Solution Settings at the top, and then scroll down to the WPF section.
3. Find the option 'WPF Legacy/UIA Interaction' and note what it is set to.
4. Keep the window open for reference.

Now activate the correct plug-in in the standalone tool you want to use:

1. Open the standalone tool you want to use.
2. Click on ‘Settings’, select Solution Settings at the top, and then scroll down to the WPF section.
3. Find the option ‘WPF Legacy/UIA Interaction’ and select the same option from the drop-down menu as in the window you kept open.
4. Click on OK in the bottom right of the window.

Selecting the WPF plug-in manually

You can also manually select the WPF plug-in you want to be activated for your solution in the solution settings. To do so, follow these steps:

1. With a solution open, click on ‘Settings’ in the toolbar.
2. Click on the ‘Plugins’ tab, select Solution Settings at the top, and then scroll down to the WPF section.
3. Find the option ‘WPF Legacy/UIA Interaction’ and select your desired option from the drop-down menu.
4. The other WPF settings will change accordingly. Please refer to the WPF section of the ⇢ Settings overview in the User Guide for more information on what these settings do.
5. Click OK in the bottom right of the window.

Delphi plugin

The dedicated Delphi plugin provides automation capabilities for all standard and a range of third-party controls and makes Delphi tests faster and more stable. Delphi tests created before the introduction of the Delphi plugin are not affected but will continue to work.

Supported Delphi versions

XE2 (x86)

XE3 and later (x86 and x64)

Supported controls

All VCL controls

DevExpress:

TdxDockControl

TcxButton

TcxCheckBox

TcxCheckListBox

TcxColorComboBox

TcxComboBox

TdxNavBar

TdxBars

TdxRibbon

TcxVerticalGrid

TcxImageComboBox

TcxGrid (including all different View kinds, i.e. Table, Layout, Master-Detail, …)

TcxCustomTreeList

TcxSpreadSheet

TcxLayoutControl

TMS:

TAdvCustomOfficeTabSet

TAdvStringGrid