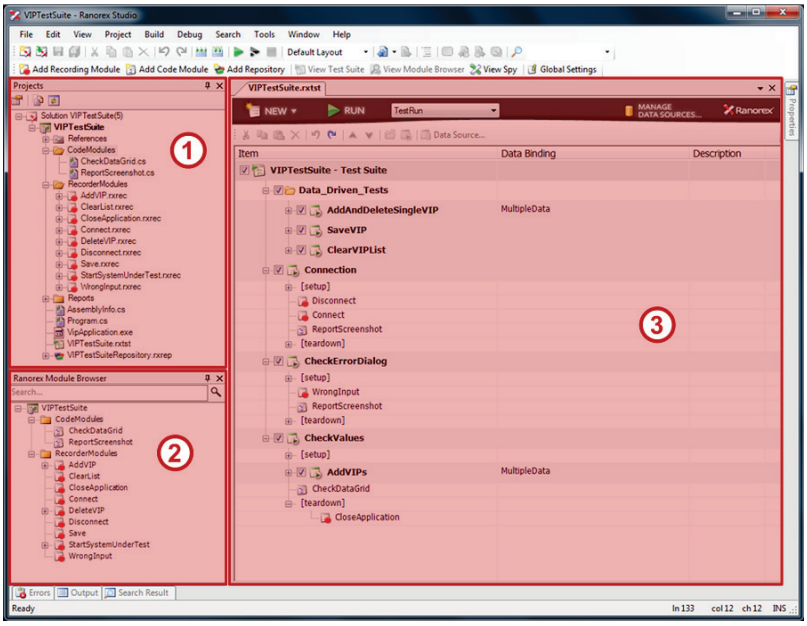
Ranorex®

Version 3

Tutorial

Ranorex Studio - The Layout

Before you start creating your first test case with Ranorex Studio you should know about the main views and layout of Ranorex Studio.



Ranorex Studio main views

#1 - Project View

A Ranorex Studio project is based on files and uses the same project file format as Microsoft Visual Studio 2008. The project view shows all files and references currently associated with the project. A Ranorex Studio project can have the following type of items:

Test Suite	Represents the projects test suite (*.rxtst)
Repository	Used to manage UI elements (*.rxrep)
Recording	Represents an automation module based on capture/replay (*.rxrec)
Code Files	Any type of C# or VB.NET code; typically used to create code based automation modules

The project view is mainly used to add new items like Recordings, Repositories or Code Modules.

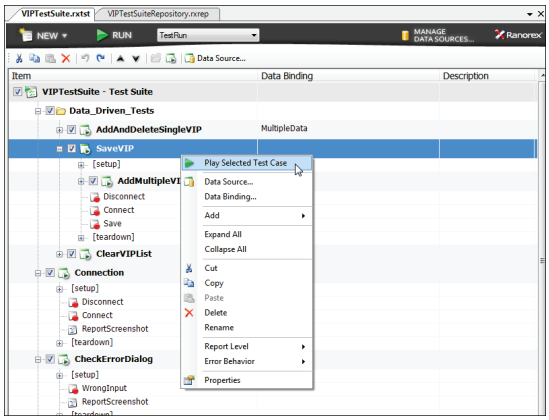
#2 - Module Browser

The 'Module Browser' view lists all available modules (Code Modules & Recording Modules) based on the project's code files. In addition it shows all the variables defined by a module. The view is mainly used to drag&drop and to reuse automation modules within the test suite view.

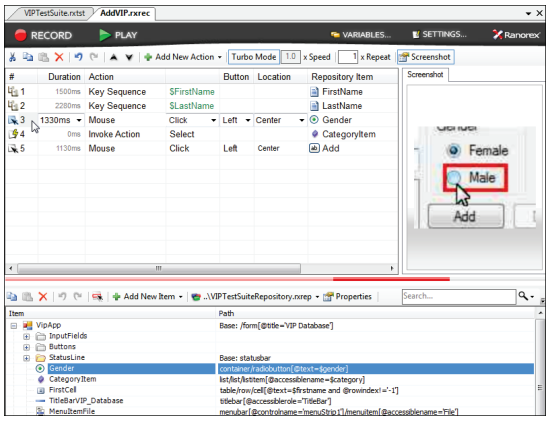
Specify folders (e.g. for recording files) within the project's view to group modules as shown in the Ranorex Studio project example 'VIPTestSuite'. In order to find already existing modules use the module browser's search field.

#3 - File View

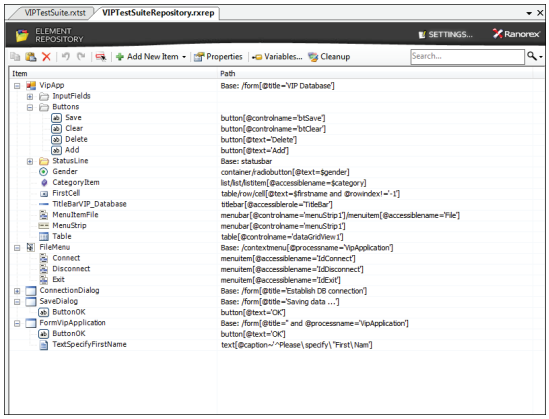
When double-clicking a file in the 'Project View' or a module in the 'Module Browser', the associated file will be opened in the Studio's file view. This view is mainly used for the following actions:



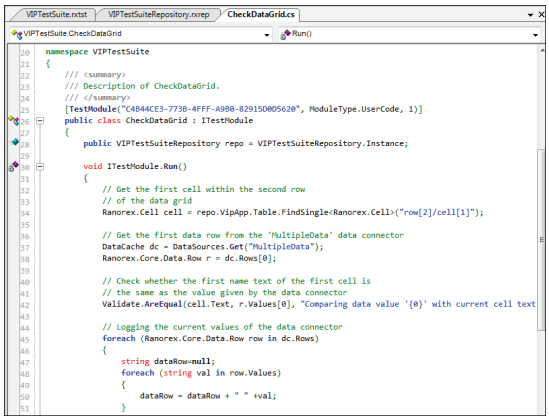
Working with the project's test suite



Creating or adapting a recording module



Working with the repository



Writing code modules

Lesson 1: Getting Started

The easiest way to create a first test with Ranorex is to record a manually executed test scenario. The recorded actions like mouse clicks or keyboard actions are the base for building a robust test case. Within this lesson you will learn how to:

- **Preparing to Record User Actions**
- **Recording a Test**
- **Analyzing Recorded Steps**
- **Executing the Test**

Preparing to Record User Actions

Before you start recording you need to ensure that your system under test (SUT) is ready to start with the manual test execution. In addition consider the following points in order to avoid too much effort in cleaning up the recording and the repository afterwards.

- **Do not run multiple instances of your application under test if that is not part of the test case itself.**
- **By default mouse movements are not recorded. For this reason please also perform a mouse click in situations like when navigating through menus.**

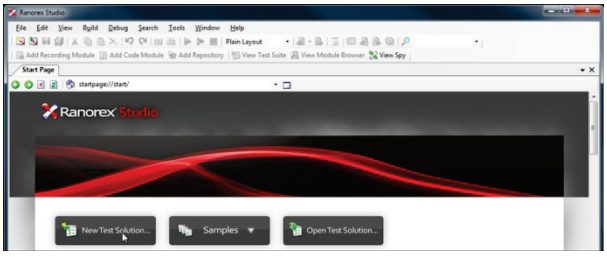
Note: Read more about how to activate mouse movement recording here.

Recording a Test

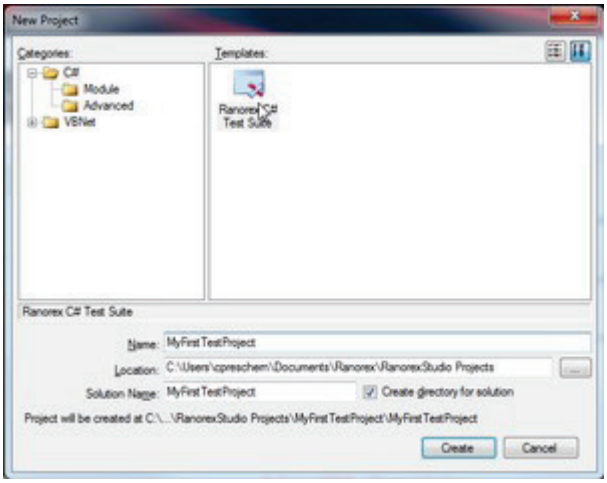
In the following section you will create a new Ranorex Test Solution and record the process of adding a new person within the VIP Database application.

1. Start Ranorex Studio and open the empty recording file.

Open Ranorex Studio by choosing Start > Programs > Ranorex > Ranorex Studio. Click the '**New Test Solution...**' Button to create a blank test suite project.



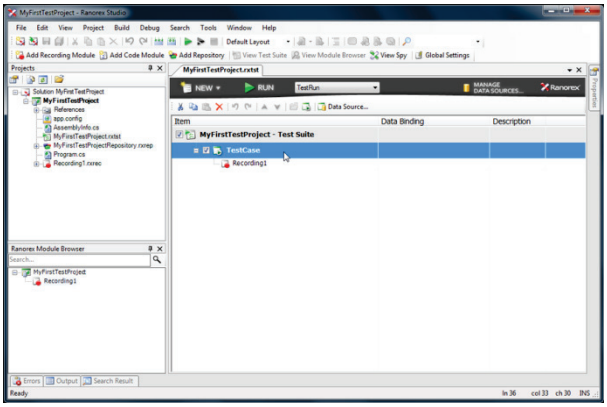
Ranorex Studio Start Page - Creating a new test solution



Specify programming language and project name

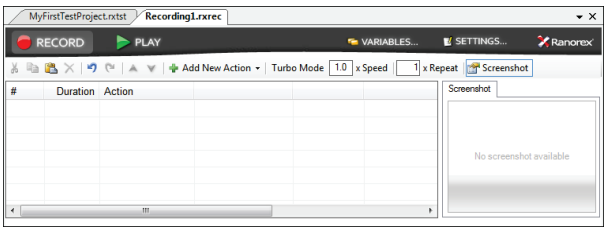
Within the categories box select **C#** and within the templates box select the **Ranorex C# Test Suite** item. In addition specify a name and a location for your new test suite project.

Click the '**Create**' button and a new test suite project opens.



New Ranorex test suite containing one test case using an empty recording

Within the test suite view the template already contains a single test case which uses an empty recording. In order to start recording simply open the 'Recording1' by double clicking the recording within the test suite view.

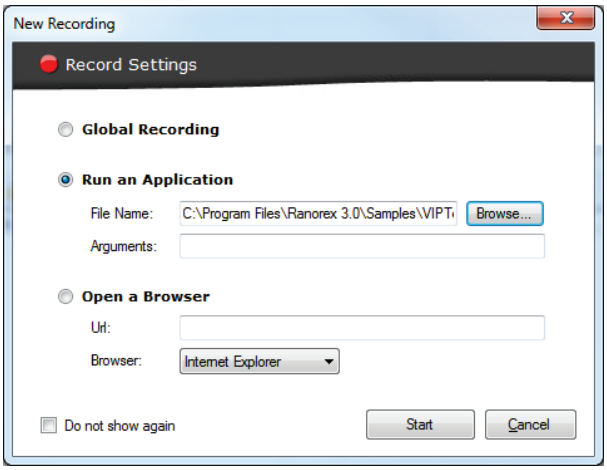


Empty Recording

2. Start recording a test performed on VIP Database application.

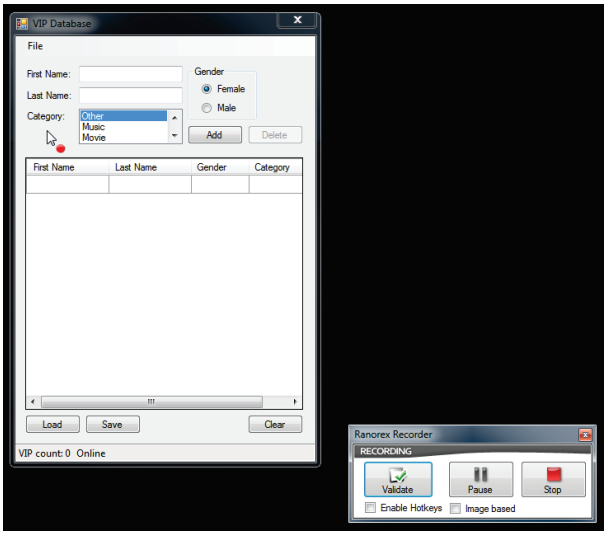
Click the 'Record' button in order to start. The Recorder assists in preparing the application under test. Simply select 'Run an Application' and specify the directory and file name of the VIP Database application

which is part of the sample directory within the Ranorex installation folder ('..\Ranorex 3.0\Samples\VIPTestSuite\C#\VipApplication.exe').



New Recording dialog - Ranorex automatically starts the specified application

Click on 'Start' in order to record the test. The VIP Database application opens and the Ranorex Recorder starts running.



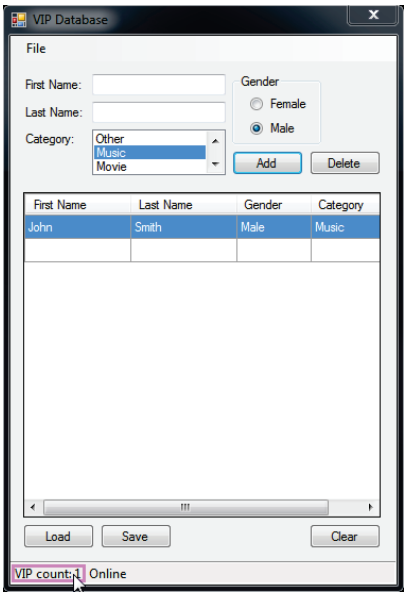
Recording actions for VIP Database application

3. Add a new VIP to the VIP Database.

- **Click on the 'First Name' text box and type in John**
- **Click on the 'Last Name' text box and type in Smith**
- **Click the radio button 'Male'**
- **Select the list item 'Music' within the category box**
- **Click the 'Add' button**

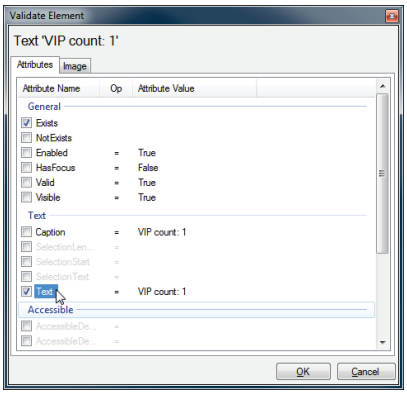
4. Validate the status text.

After adding a new person, the text value of the status bar should change from 'VIP count: 0' to 'VIP count: 1'. Click the 'Validate' button within the 'Recording' toolbar to check whether this is correct. Move the mouse pointer to the status field and wait a second until the underlying UI element is highlighted.



Identify element to validate

The dialog shows all available attributes for the given UI element to check. Just click the 'OK' button in order to accept the preselected 'Text' attribute.



Select attribute 'Text' to validate

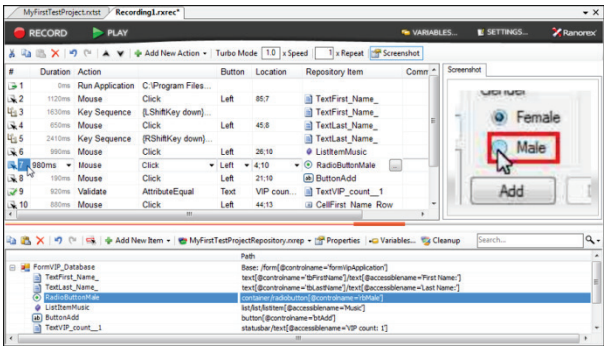
5. Delete added item and close application.

In order to finish the recording scenario select the added item within the data grid and delete it from the list by clicking the 'Delete' button. Also close the application by clicking the close button.

6. Stop the Recording by pressing the 'Stop' button in the recorder toolbar.

Analyzing Recorded Steps

The Recorder created single steps for each operation you performed on the application. These steps are represented within the actions table. In addition each action item is connected to a repository item which represents a UI element (text boxes, radio buttons, buttons, etc.) used during the recording.

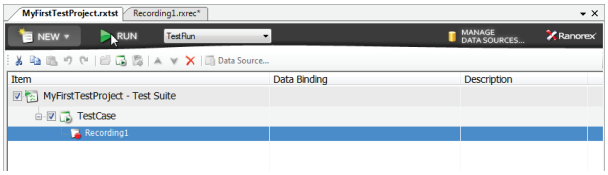


Recorded steps shown within the actions table

Read more about different action types within Ranorex Recorder here.

Executing the Test

In order to execute the recorded test you need to switch back to the Ranorex Test Suite file.

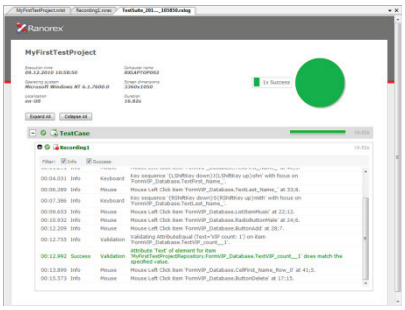


Start the test by clicking the 'Run' button

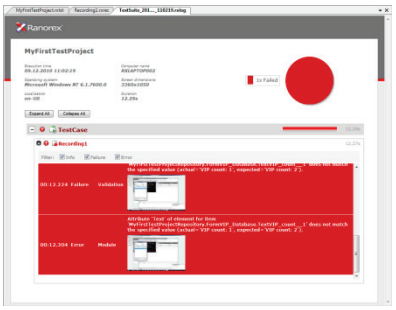
Just click on 'Run' to execute the test suite with your first recorded test. During the execution Ranorex simulates all user actions (mouse movements and keyboard events) in order to test the application the same way a tester would do it.

Reporting

After executing the test, Ranorex Studio automatically opens the generated test report file (*.rxlog) which shows whether the test run was successful or not.

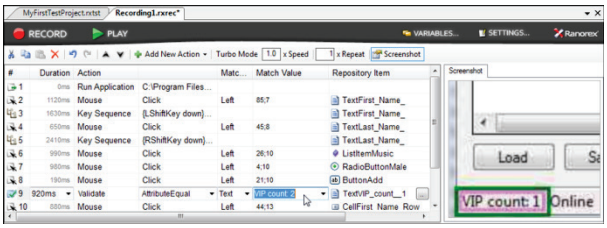


Test run succeeded



Test run failed

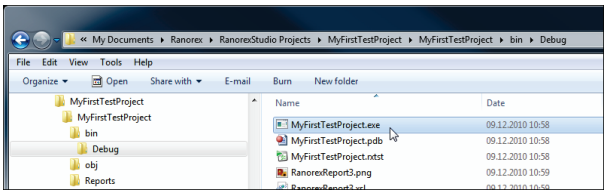
In order to force an error as shown by the right picture above, just modify the expected value used in the validation step of the recording.



Force an error by changing the expected value of the validation step

Now the test automation executable is also available within your project folder. To run the test suite without starting Ranorex Studio, simply double click the executable file.

Note: In order to run the test suite project on an external machine it is required to have the executable (*.EXE) as well as the test suite file (*.RXTST) in the same directory at the target machine. If your Ranorex Studio solution consists of more than one project you need to ensure that the library (*.DLL) files are part of the same directory too.



'MyFirstTestProject.exe' located within the build folder of the project

By default also the report files are generated within the same directory. Further information on changing the report folder can be found here. Read more about alternative ways to run the test suite from the command line or with the standalone test suite runner here.

Lesson 2: Ranorex Modules - Test Actions

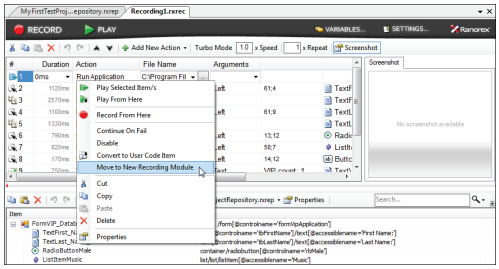
The ease of recording user actions encourages one to record all test cases without thinking about reusability. But in the long run, this could increase the effort in test automation maintenance. For this reason you should separate a recording into smaller reusable recording modules as suggested by the keyword driven methodology. Within the following section you learn how to separate the existing recording into smaller pieces, so that they can be used within other test cases too.

Identifying Modules

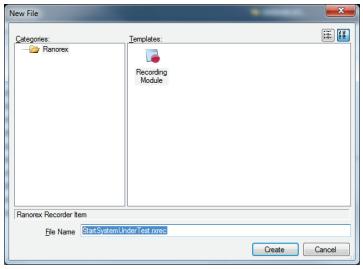
In your first recording you did the following from a keyword driven perspective:

- **Started the application VIP Database**
- **Added a new person**
- **Validated the current text of the status bar**
- **Deleted the added person**
- **Closed the application**

In order to split recordings into smaller automation modules use the context menu item 'New Recording From Selection'.

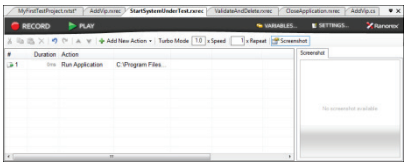


Move first action to a new recording

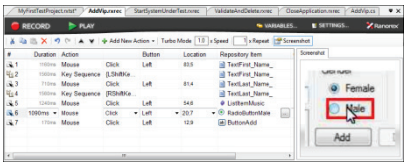


Create a new recording named 'StartSystemUnderTest'

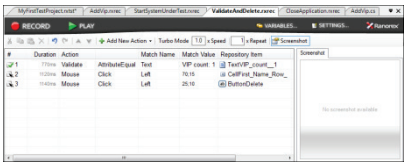
Also select and move all items within the recording used to do the validation and to delete the person to a new recording named ValidateAndDelete.rxrec the same way as before. Repeat the previous step to create the last new module for 'CloseApplication'. Also rename the initial recording file from 'Recording1.rxrec' to 'AddVip.rxrec'. Finally you should have four separated new recording modules.



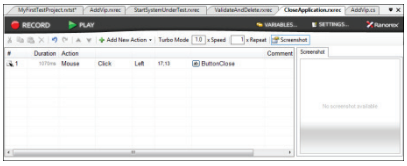
Recording module 'StartSystemUnderTest'



Recording module 'AddVip'

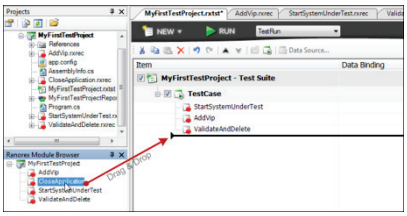


Recording module 'ValidateAndDelete'



Recording module 'CloseApplication'

After splitting the initial recording into smaller test actions the Ranorex Module Browser also contains the new modules. Now simply use the drag&drop feature to combine these modules within the test case as follows:



Use the drag and drop feature to specify your test case within the test suite view

Separating a recording into smaller modules is the first step to build robust and reusable automation modules. In addition replace constant values used within these modules with variables in order to enable parameterization. Read more about how to use variables in section Lesson 3: Data Driven Testing.

Lesson 3: Data-Driven Testing

When you test an application it might be necessary to run the same test with different input data. Within the following section you learn about:

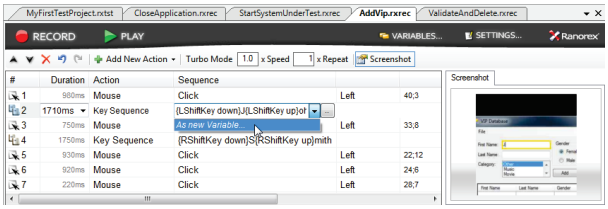
- **Using Variables within Recordings**
- **Using Variables within the Repository**
- **Creating Test Data**
- **Combining Variables with Test Data**
- **Combining Variables with Parameters**
- **Executing Data Driven Test**

Using Variables within Recordings

According to the VIP Database application we can identify four variable input actions:

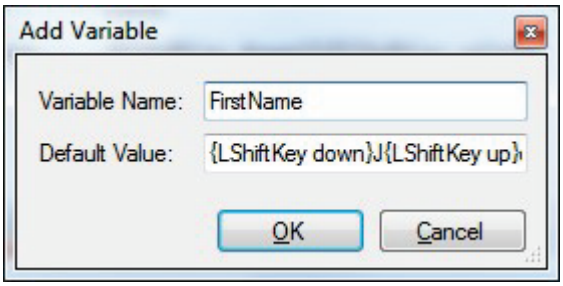
- **First Name (John)**
- **Last Name (Smith)**
- **Gender (Male)**
- **Category (Music)**

All these input actions are done within the AddVip recording. You need to open the recording file and identify the actions which have to be variable.



Create a new variable for a key sequence action item

Open the combo box as shown above and select 'As new Variable' to create a new variable.



Create a new variable

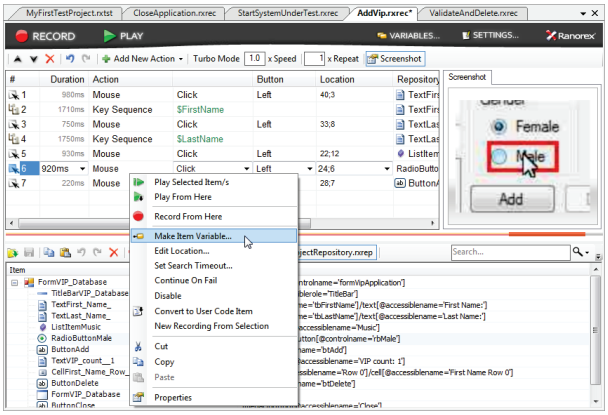
Specify the variable name and the default value, which is automatically set to the same value as it was recorded initially. If you want to alter the name or the default values simply open the variables dialog as described here.

Note: Please do not use variable names which are already in use by a recording or code module.

Repeat the previous step to create a variable 'LastName' for the key sequence action used to set the value of the last name field, too.

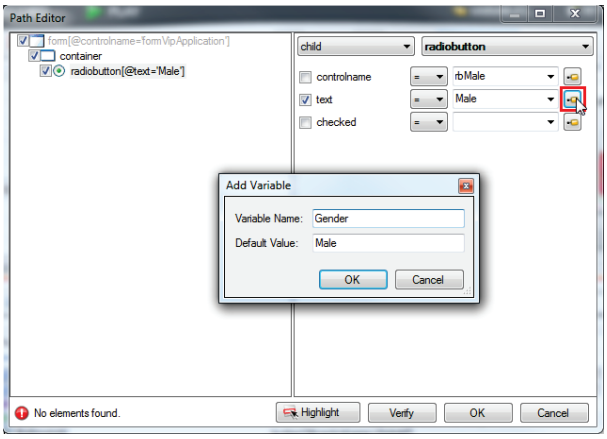
Using Variables within the Repository

Within the recording 'AddVip' the variables 'FirstName' and 'LastName' are used for key sequence actions. To make a click action data driven - for example selecting the radio button Male or Female - you need to define a variable used for identification within a RanoreXPath expression.



Make click action variable

Select that action item within the 'AddVip' recording which simulates the click on a radio button and open the context menu as shown above. Click on 'Make Item Variable...' to open the RanoreXPath editor.

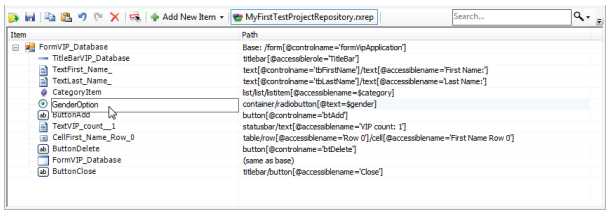


Create a new variable for selecting a radio button

Typically the radio button for the VIP Database application is identified by the controlname attribute. So the attribute controlname is checked by default. Because the test should use the text value as data driven input later on, you need to uncheck the control name attribute and check the text attribute as shown above. Now you can define a new variable for the attribute text by clicking the button on the right.

The same way you created a data driven click action on the radio button, you have to do it with the click action used to select a category item within the list box.

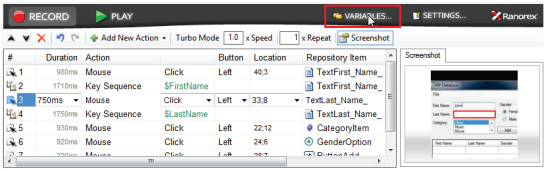
When looking at the repository you see that now two items are using the variables within the path expression specified before.



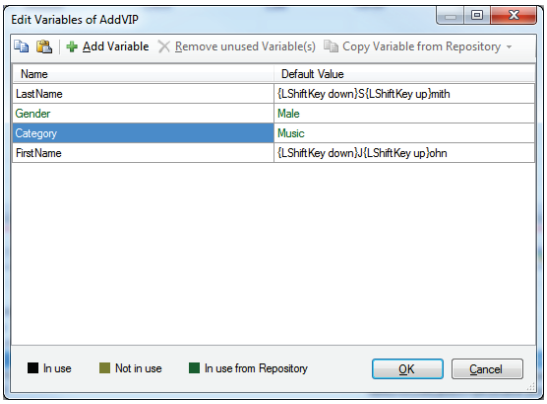
Renaming repository items

In addition rename the repository items to 'CategoryItem' and 'GenderOption' to clarify for what they can be used.

The recording AddVip now uses 4 variables. Two of them are used for key sequences directly within the recording, while the others are used within the repository. To get an overview about all variables used within the recording simply click the 'Variables' button as shown below.



Open the recording's variable dialog

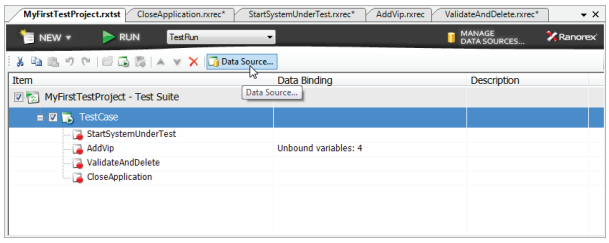


Currently used variables by the recording

Creating Test Data

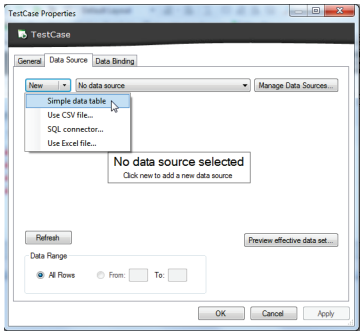
As you saw, the recording now uses four variables. In this section you will learn how to create a simple data table to provide the values for these variables. To create a new Ranorex data source you need to open the test suite view.

Select the test case which uses the 'AddVip' recording and open the Data Source dialog by clicking the 'Data Sources' button within the toolbar.

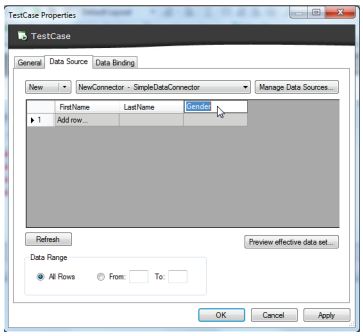


Open the data source dialog

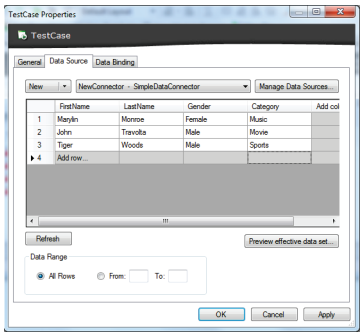
Now create a new data table of type 'Simple Data Table'. Click the column headers to specify the column names as shown below. Finish the creation of the data table by defining some data records.



Create a new 'Simple Data Table' connector



Specify column names

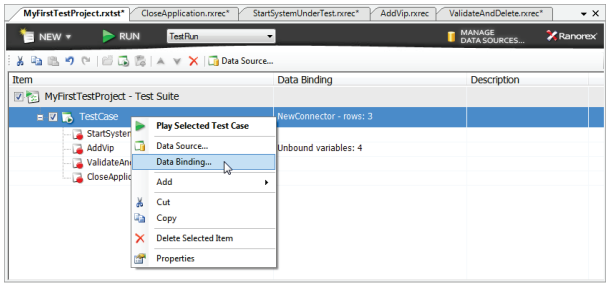


Define test data

For further information about different Ranorex data connector types click here: Data Connectors

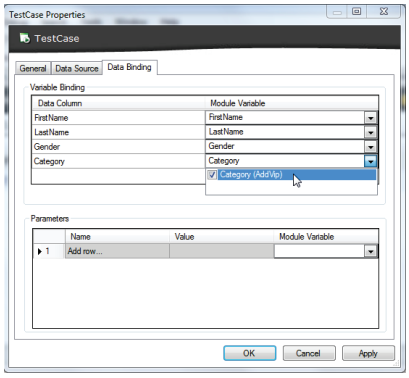
Combining Variables with Test Data

In order to combine your data table with the current test case - and finally with the AddVip recording which is used by the test case - you need to open the data binding dialog using the context menu as follows:



Open data binding dialog

Now you can connect each column specified within the data table with variables used by the test case. You can also map multiple variables to one data column. Right now you only have variables specified within the 'AddVip' module.

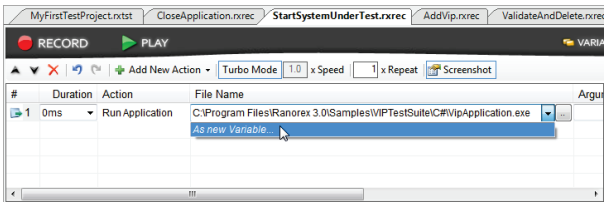


Connect data columns with module variables

Combining Variables with Parameters

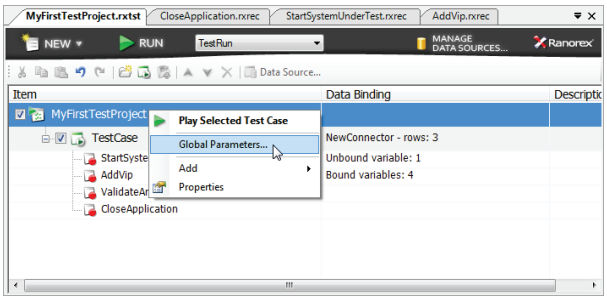
In addition to typical test data specified within data tables as shown before, a test suite and its test cases allow declaring global and local parameters. These parameters can be bound to module variables too. This type of data driven execution helps to specify for example environment related attributes as the following example describes.

The test case starts with a recording module called 'StartSystemUnderTest'. The recording only contains one single action item used to start the VIP Database application. Open the recording and create a new variable called 'ApplicationPath'.

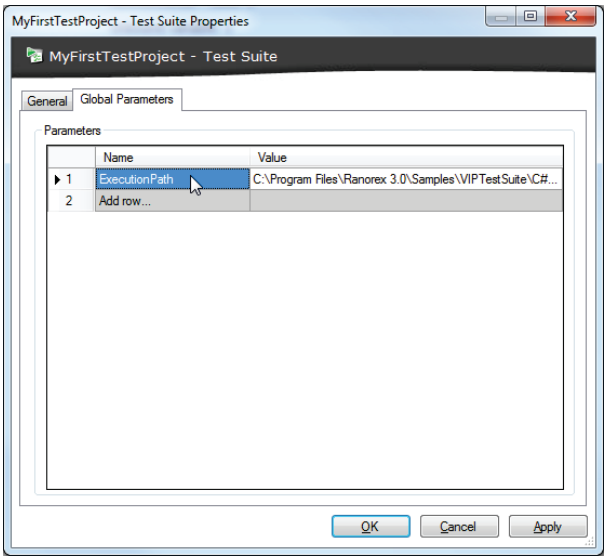


Create new variable for the execution path of the system under test

The execution path of the system under test has to be declared as a global parameter on the test suite level in order to combine it with multiple test cases of the test suite. Use the context menu within the test suite view and open the parameters dialog as follows.



Open global parameters dialog

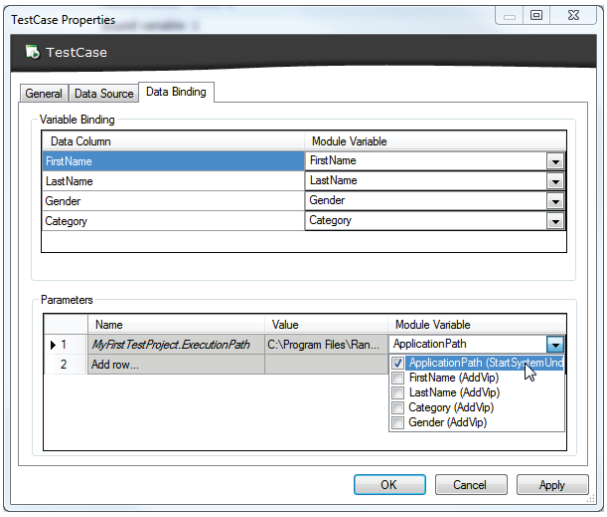


Define a new global parameter for the test suite

Specify a new parameter name by clicking in the first cell of the column 'Name'. Also specify a value for the new parameter. Just copy and paste the execution path from the default value shown within the variables dialog of the recording 'StartSystemUnderTest'. Read more about how to open a recording's variable dialog here: Recorder Variables

Click 'OK' to apply the changes and to close the dialog.

Now you are ready to combine the global parameter with the variable created within the recording before. Open the test case's data binding dialog the same way you did before for combining variables with the data table.

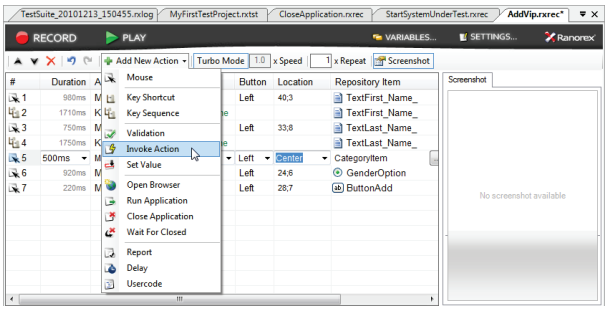


Combine global parameter with the variable used within recording 'StartSystemUnderTest'

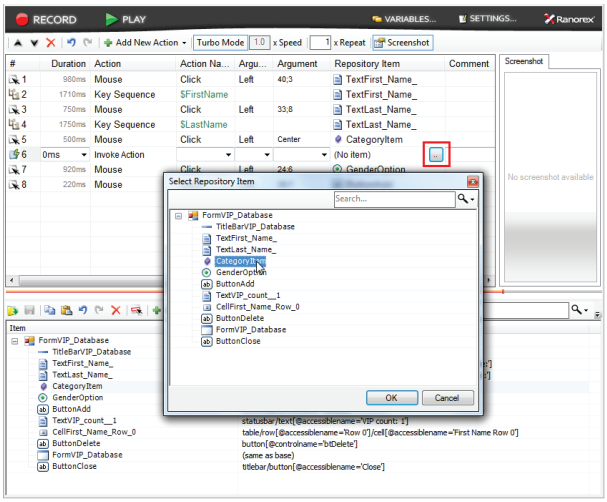
Executing Data Driven Test

Before you execute the data driven test case you have to make sure that the values specified within the data table can be used correctly within the recording 'AddVip'. The values for the first name, last name and gender will not cause any problems during automation, but the values used for the category (Music, Movie, Sport, etc.) will be a problem because of the current visibility state of each item. Remember, you recorded a click on a visible list item within the list box category. When starting the VIP Database application you see that for example the category 'Sports' is not visible. Therefore you need to ensure visibility - using some lines of user code before you simulate the click action - or use a so called 'Invoke Action' to select a list item as shown below.

First, open the recording 'AddVip' and add a new 'Invoke Action' after the existing mouse click action used to select one of the category items.



After adding the new item you need to specify for which repository element the invoke action should be used. Simply open the 'Select Repository Item' dialog as shown below and select the 'CategoryItem'.



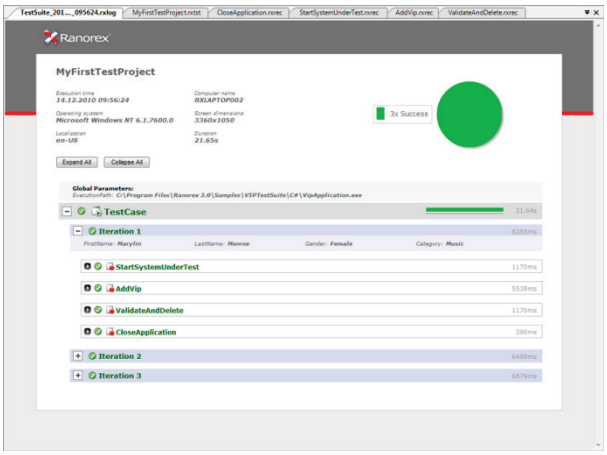
Select repository item 'CategoryItem'

After assigning the repository item to the invoke action the recorder provides the methods suitable for the role list item. The category list item only provides a 'Select' method which you can chose as follows.

Choose 'Select' method

Also delete the initial recorded mouse click action (action item #5 shown within picture above), because now the recording uses the new invoke action to select the list item.

Switch back to the test suite view and start your data driven test by clicking the 'Run' button.



Report file showing three iterations of the test case

After executing the test suite the report file shows the result for each iteration. The summary chart counts each iteration as a single test case run.

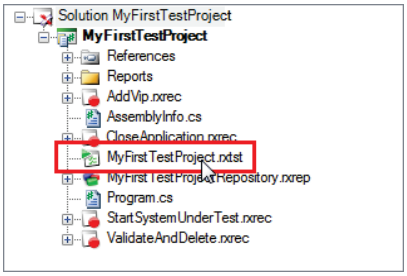
Lesson 4: Ranorex Test Suite

As already mentioned in Lesson 2 it is recommended to separate a test case into reusable and parametrizable test automation modules. Within the Ranorex test suite you can not only easily create new test cases by combining already existing automation modules but also specify global parameters and data connectors to set up data driven test cases. Within the following lesson you learn about:

- **Test Suite Editor**
- **General Structure of a Test Suite and its Test Cases**
- **Running a Test Suite**
- **Running Tests without Ranorex Studio**
- **Test Suite Settings**
- **Test Case Settings**

Test Suite Editor

After the creation of a new test suite project a Ranorex Studio project automatically contains a <Project-Name>.rxtst file. By default the test suite file is open automatically.



Ranorex test suite file

Looking on the project created during the lessons 1 - 3 you can now create a new test case based on the existing modules. To open up the test suite you can double click the *.rxtst file or use the 'View Test Suite' button shown in the toolbar panel.

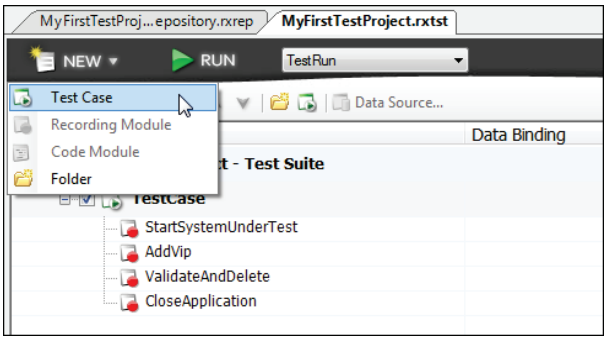
Ranorex Studio Toolbar

Add a New Test Case

You can add a new test case by clicking the 'NEW' button. Depending on the current item selection of the test suite you are allowed to add different types of test suite elements. A test suite can be made up of the following items:

- **Test Cases**
- **Record Module**
- **Code Module**
- **Folder**

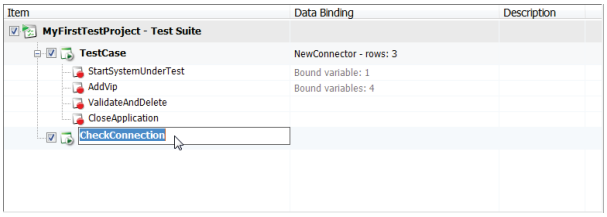
Select the 'Test Case' item to create a new test case at the current position.



Add new test case

Note: Within a test suite test cases can be added to different layers. A test case can also be added as a child to an existing test case or to a folder item.

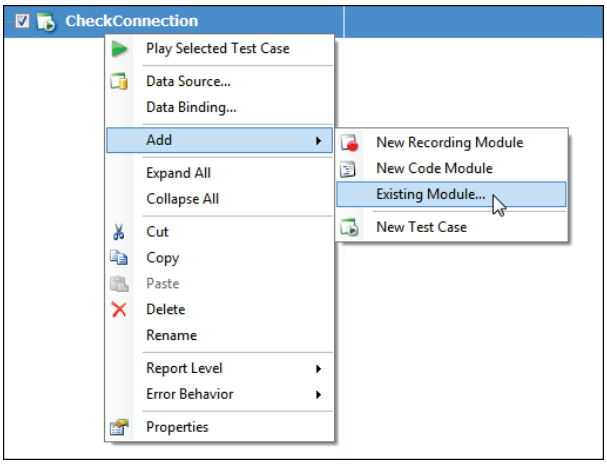
The second test case should check the connection state of the VIP Database application. First select and then click on the test case to rename it to 'CheckConnection'.



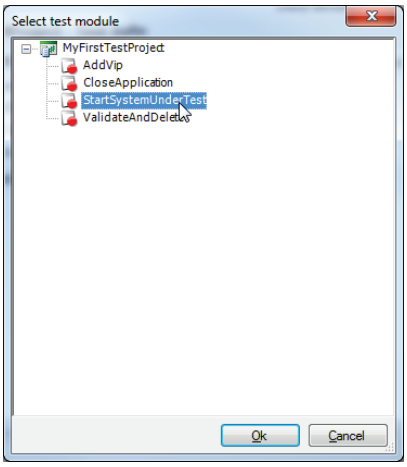
Rename the test case

Reuse of Existing Modules

Now, you are ready to reuse two existing record modules from the Ranorex Module Browser View. You can simply drag and drop items from the module browser into the test case or you can add new or existing items using the context menu as follows:



Add an existing recording module

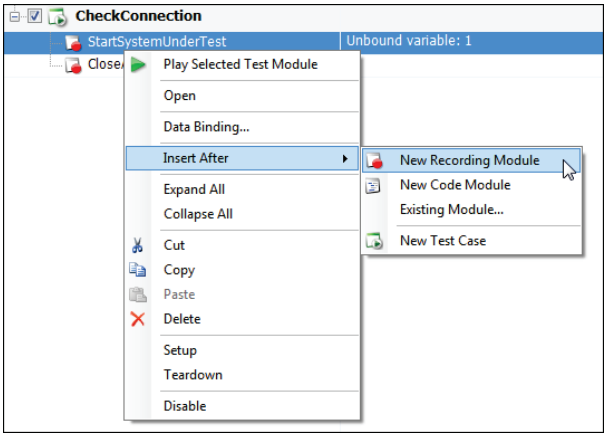


Select 'StartSystemUnderTest' module from the list

Press 'OK' to add the record module to the test case. Repeat the same to add the record item 'CloseApplication' to the test case.

Insert a New Recording Module

Now the test case only covers only starting and closing of the application under test using two existing modules. In order to check the connection state of the VIP Database application after starting, you can create a new recording for the test case. Use the context menu again but this time to insert a new recording into the test case.



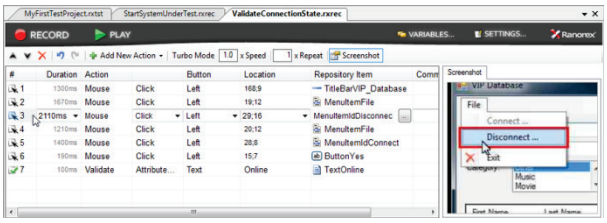
Insert a new record module

Before you start with a new recording to check the connection state of the VIP Database application, you can run the executable manually or simply open and run the 'StartSystemUnderTest' recording. The new recording module should cover the following steps:

- **Disconnect Application - Selection of menu item 'File' | 'Disconnect'**
- **Connect Application - Selection of 'File' | 'Connect'**
- **Confirm Dialog 'Establish DB Connection'**
- **Verify 'Online' Status Text**

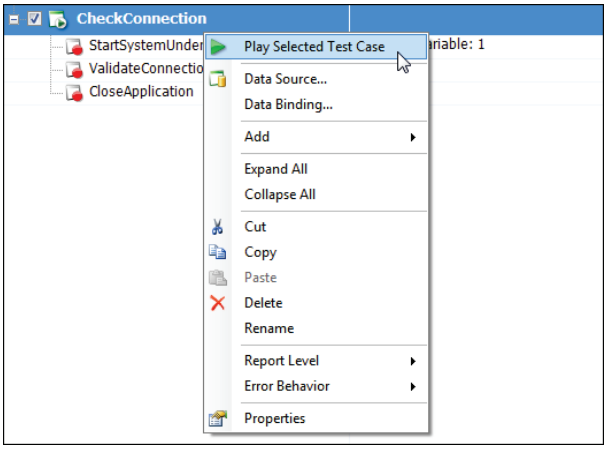
Keep in mind that you need to select the 'Global Recording' option because the system under test will be started by another recording module.

Note: In the event that the first action is carried out within a .NET based application is a click performed on a menu item, the click is not really executed. To avoid this situation you need to set the current focus first by clicking the title bar for example.



New created recording scenario

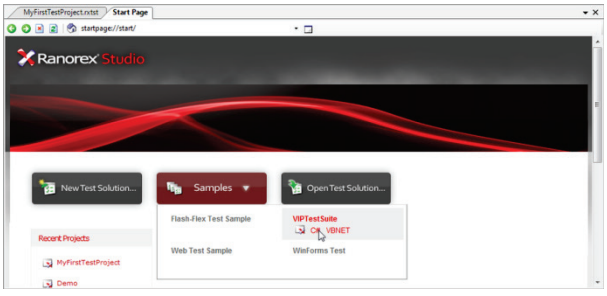
Now the test case 'CheckConnection' is ready to get executed. Use the context menu item 'Play Selected Test Case' to run it and see whether it works or not.



Play selected test case

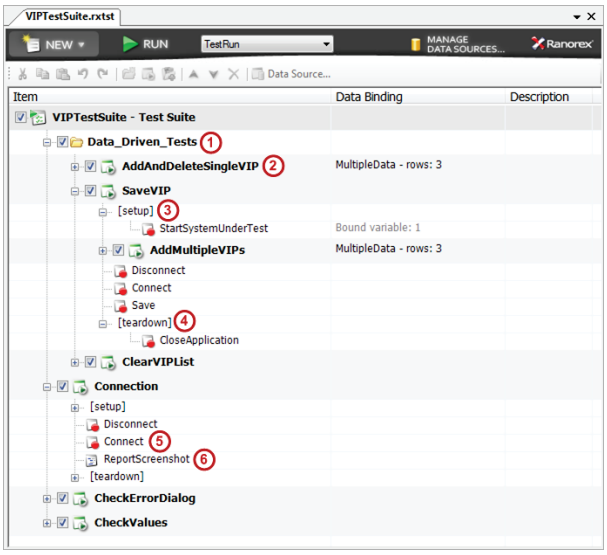
General Structure of a Test Suite and its Test Cases

During the last section of lesson 4 you learned how to add new test cases to the test suite by combining existing modules with a newly created recording. To see the different ways of organizing a more complex test suite project open the sample test suite project 'VIPTestSuite' from the Ranorex Studio start page:



Open sample test suite project 'VIPTestSuite'

The sample includes the different types of elements within a test suite and covers all possible combinations.

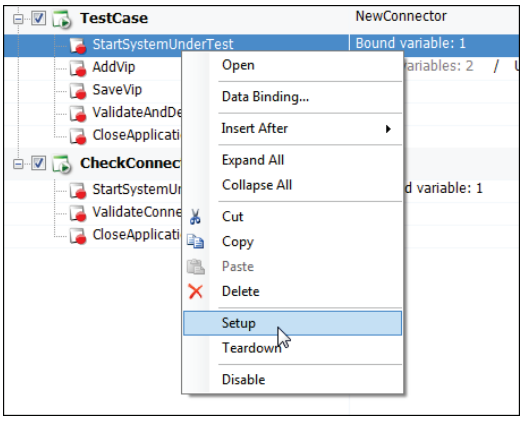


Ranorex test suite structure

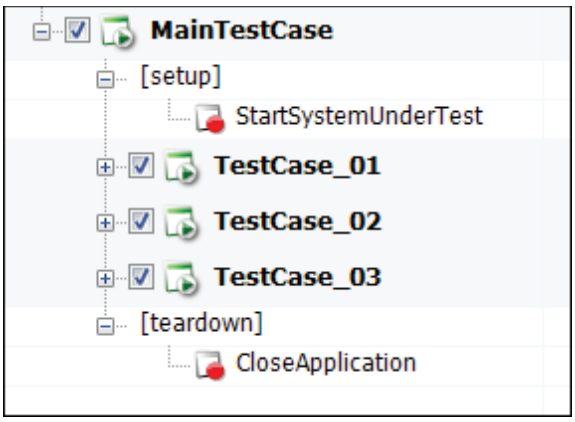
#1 Folder	Used to group multiple test cases
#2 Test Case	Represents a test case which can contain modules, a setup or teardown region or other test cases
#3 Setup Region	Groups modules used to prepare a test case (e.g. start system under test, initialize a database, etc.)
#4 Tear Down Region	Groups modules used to clean up a test case (e.g. deleting files generated during test execution, closing application, etc.)
#5 Recording Module	Automation module generated by recording
#6 Code Module	Automation module written in code

Setup and Teardown Regions

The Setup and Tear Down regions are used to prepare and clean up a single test case execution. Use the context menu to specify which modules of a test case should be part of the setup or teardown process.



Add module to setup region



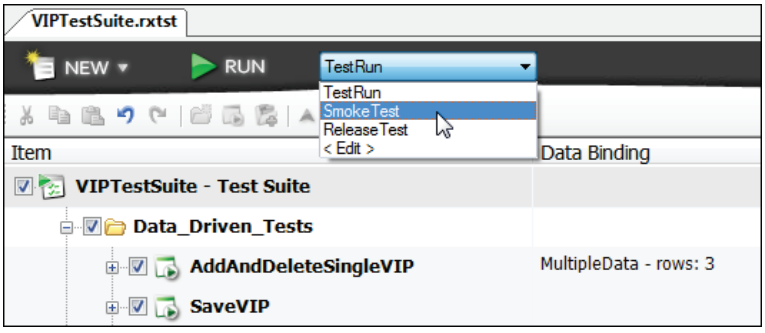
Nested test case for a general setup and teardown region

Note: If you want to define one setup and one teardown region for a set of test cases simply nest your test cases as shown in the picture on the right.

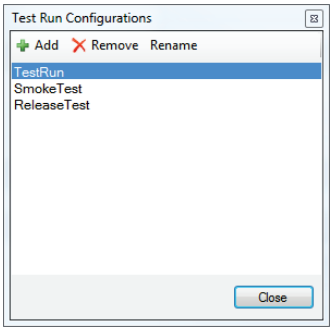
In order to quickly deactivate a certain module instead of deleting it from the test case use the context menu item 'Disable'.

Running a Test Suite

To run a test suite click the 'RUN' button shown within the test suite editor's toolbar. Use the check boxes to specify whether single test cases or group of test cases should be part of a test suite run. Create different test suite run configurations using the drop down box from the toolbar.



Activate different test suite configurations



Add or remove test suite configurations

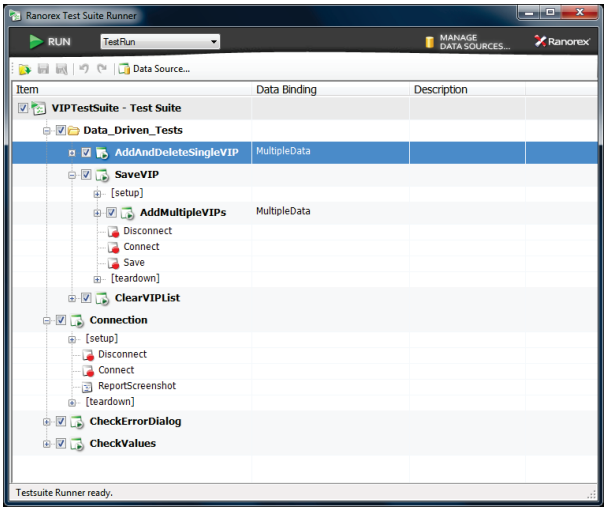
Running Tests without Ranorex Studio

As you already learned within Lesson 1 Ranorex Studio creates an executable file from your test suite project. In order to execute the test suite in a runtime environment it you have to have the **generated executable (*.EXE)** with the **test suite file (*.RXTST)** in the same directory. You can execute the test suite outside Ranorex Studio using

- **the Ranorex Test Suite Runner**
- **via command line**

Ranorex Test Suite Runner

Simply double-click the *.RXTST file from the project's output folder to open the Ranorex Test Suite Runner.



External Ranorex Test Suite Runner

You can use the Ranorex Test Suite Runner to run the test suite, execute certain test cases or just play a specific automation module. Additionally one can create new run configurations the same way as within a Ranorex Studio project.

Running Test via Command Line

You have the following options to execute the test suite from the command line:

GeneratedTestSuite.exe /<argument>

Allowed Arguments:

```
help|?
```

Prints this help text.

```
testcase|tc:<name of test case>
```

Runs this test case only. (e.g. GeneratedTestSuite.exe /testcase:TestCase1)

```
testsuite|ts:<path to testsuite file>
```

Runs the test cases defined by the test suite (*.rxtst) file. By default

- **the *.rxtst file with the same name as the <testsuiteexe> is used</testsuiteexe>**
- **<testsuiteexe>- or the first *.rxtst file in the same folder as <test-suiteexe>.</testsuiteexe></testsuiteexe>**

`runconfig|rc:<configuration name>`

Runs the test cases of the specified configuration defined by the *.rxtst file. Configurations can be edited using Ranorex Studio or TestSuiteRunner. By default, the currently selected run config is used. (e.g. *GeneratedTestSuite.exe /runconfig:TestRun)*

`reportfile|rf:<report file path>`

Sets the name (and path) of the report file. By default, the filename specified in the *.rxtst file is used.

`reportlevel|rl:Debug|Info|Warn|Error|Success|Failure|<any integer>`

Sets the minimum report level that log messages need to have in order to be included in the log file. These levels correspond to the following integer values: Debug=10, Info=20, Warn=30, Error=40, Success=110, Failure=120

`module|mo:<module name or guid>`

Runs the module with the specified name or guid. Assemblies loaded by <TestSuiteExe> and assemblies referenced in the *.rxtst file are searched. (e.g. *GeneratedTestSuite.exe /module:Recording1)*

`param|pa:<global parameter name>=<value>`

Creates or overrides values for global parameters specified in the test suite. (e.g. *GeneratedTestSuite.exe /param:Parameter1="New Value")*

`runlabel|rul:<custom value>`

Sets a custom runlabel for the test run.

`listconfigparams|lcp:<custom value>`

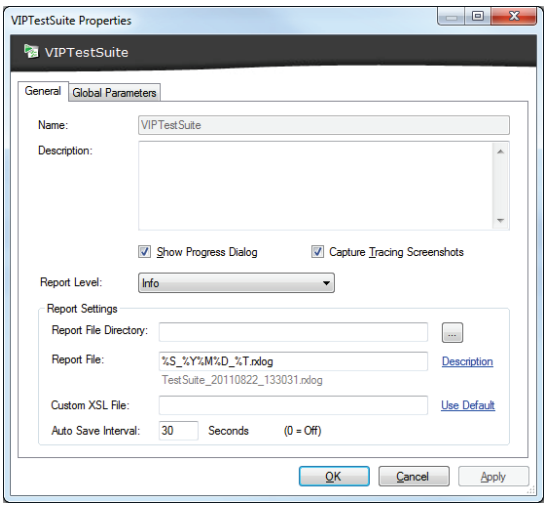
Lists all settable configuration parameters and their values.

`listglobalparams|lp:<custom value>`

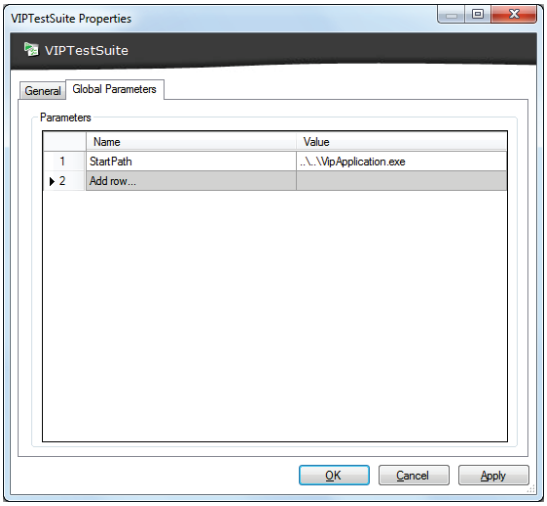
Lists all global parameters and their values.

Test Suite Settings

Open the test suite's settings dialog by selecting the context menu item 'Properties' on the test suite root node.



General settings of a test suite



Global parameters table

General Test Suite Settings

Name	Specifies the name of the test suite (same as shown within the test suite editor)
Description	Description of the test suite (same as shown within the description column of the test suite editor)
Show progress dialog	Specifies whether a progress dialog should be shown during test suite execution or not
Report Level	Specifies the level of report shown with the report file

Additional Report Settings

Report File Directory	Specifies the directory for the generated report files
Reporting Filename	Specifies the filename generated by the report; move the mouse over the 'Description' label to see the variable values
Custom XSL Filename	Specifies a customized style sheet file used instead of the Ranorex style sheet to present the reports
Auto Save Interval	Specifies how often the report file is saved during an execution

Global Parameters

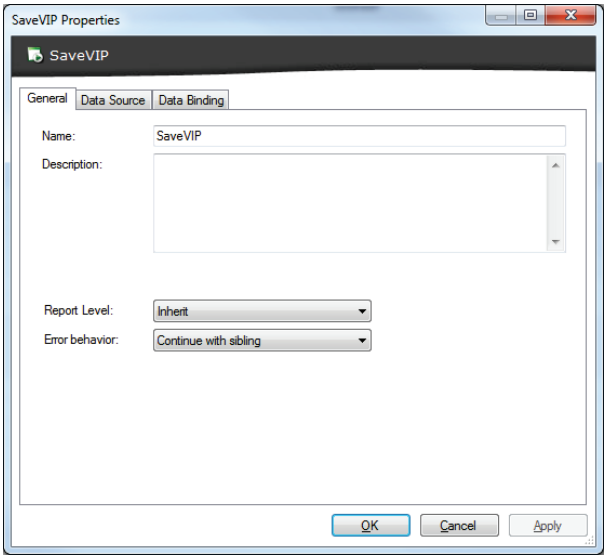
Globally specified parameters can be accessed and used by every test case within the test suite. The VIPTestSuite example project uses a global parameter to specify the directory path for the application under test. You can easily connect global parameters with variables as shown in the recording module 'StartSystemUnderTest'. In addition you can use the global parameter to transfer values or states between test cases.

Test Case Settings

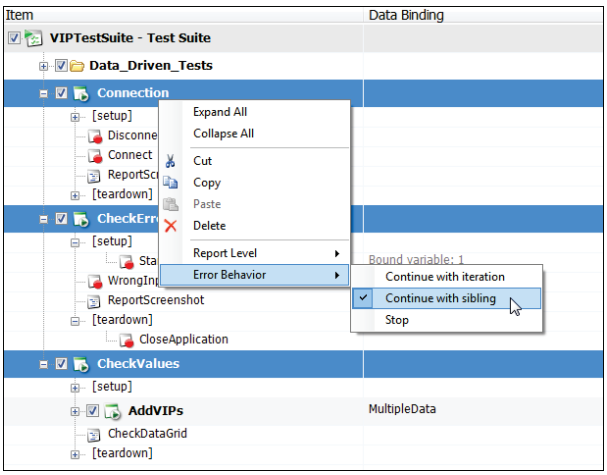
The 'General' tab of the test case's properties dialog is mainly used to setup how a broken test case impacts other test cases within the test suite.

Name	Name of the test case

Description	Description of the test case (same as shown within the description column of the test suite editor)
Report Level	Specifies the level of report shown with the report file; by default the level of report is set to 'Inherit' which means that the test case inherits the setting from its parent (test case or test suite)
Error Behavior	Specifies the behavior of the test case and the test suite in case of on error Set to 'Continue with iteration' if the case should not break in case of an error caused by a particular data set Set to 'Continue with sibling' if the case should not break and continue with the next test case Set to 'Stop' in order to break the test suite run in case of an error



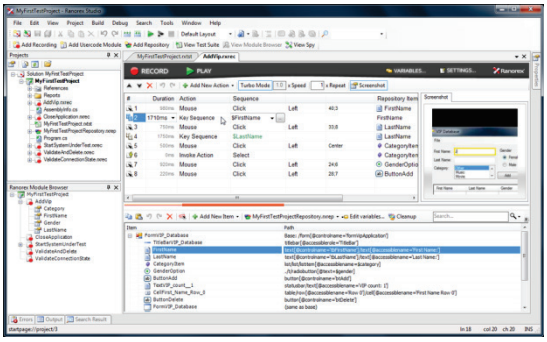
Test case properties dialog



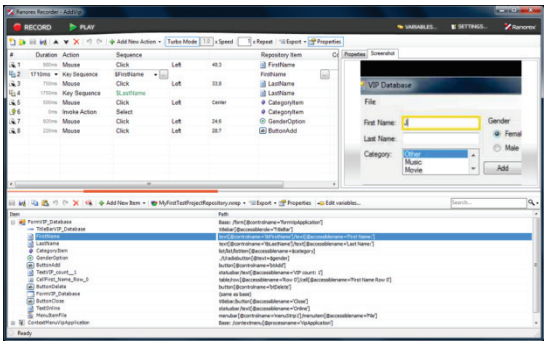
Specify error behavior for multiple test cases

Lesson 5: Ranorex Recorder

The Ranorex Recorder is used to record and replay the user's keyboard and mouse actions during a manually executed test of the user interface. In addition the Recorder can validate current states, properties like 'Text' or 'Visibility' and compare images of UI elements. The Recorder is a capture replay editor which is available both as stand-alone tool and as an integrated editor within Ranorex Studio.



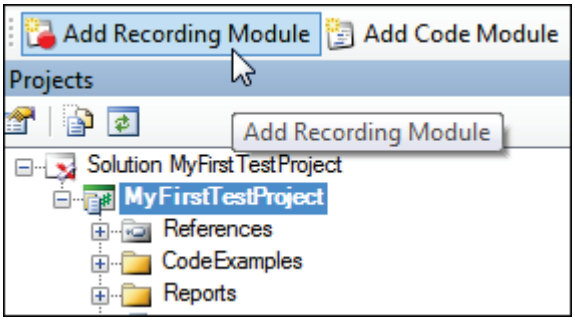
Integrated into Ranorex Studio



Ranorex Recorder as standalone tool

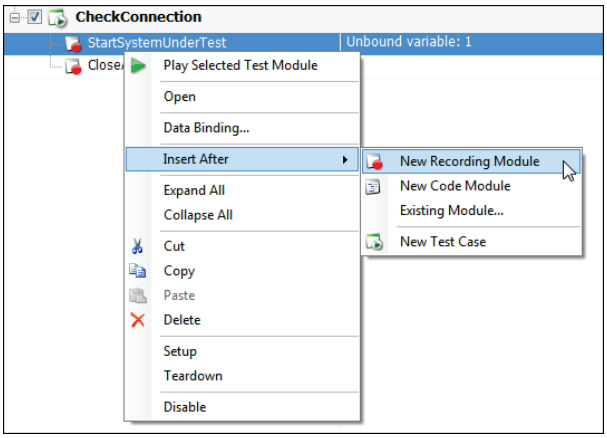
Within Ranorex Studio you can add a new recording in different ways.

Use the project view to add a new recording by clicking the 'Add Recording Module' button on the toolbar.



Adds a new recording to the project

Alternatively you can also add a recording directly to a test case within the test suite editor.



Adds a new recording to an existing test case

During the next chapters you learn about the following:

- **Before Start Recording**
- **During Recording**
- **After Recording**
- **Replay and Debug Actions**
- **Recorder Variables**
- **User Code Actions**
- **Additional Editing Options**
- **Image-Based Automation**

Before Start Recording

Regardless of whether you use Ranorex Recorder within Ranorex Studio or as a stand-alone tool, not only the Recorder but also the system under test has to be prepared.

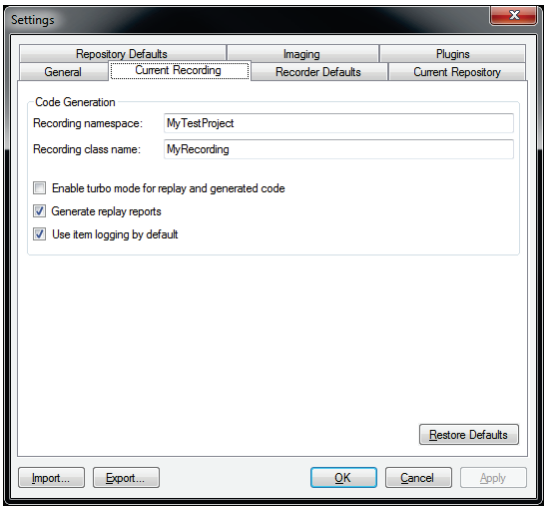
Prepare to Record

Before you start recording, you need to ensure that your system under test is ready to start with the manual test execution. In addition, consider the following points in order to avoid too much effort in cleaning up the recording and the repository afterwards.

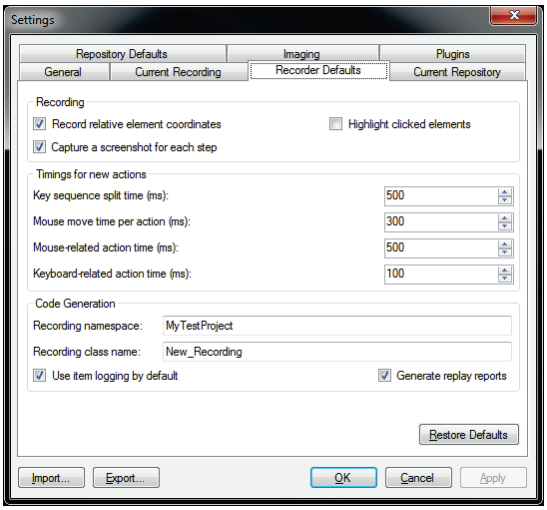
- **Do not run multiple instances of your application under test if that is not part of the test case itself**
- **By default, mouse movements are not recorded; for this reason please also perform mouse clicks in situations like when navigating through menus (or use Recorder Hotkeys to record mouse movements)**
- **Think about which steps should be part of the final test**
- **Try to keep recordings small in order to make them modular**

Recorder Settings Dialog

Before running a new recording session, you can also configure Ranorex Recorder using the settings dialog. Click the 'Settings' button to open the settings dialog.



Settings for current recording file



Default values used for every recording

Current Recording

This tab primarily contains configuration parameters for code generation specific to the current recording. All settings on this tab page are stored within each recording.

Recording namespace:

Specifies the namespace used for the generated code.

Recording class name:

Specifies the class name used for the generated code.

Enable turbo mode for replay and generated code:

Is used to specify whether recorded delays between actions should be part of the generated code.

Generate replay reports:

This setting is used to turn the generation of reports during replay on or off.

Use item logging by default:

Activate this setting to turn on a default logging message for each action item.

Recorder Defaults

Within this tab you can specify global default values for every newly created recording. The settings are divided into three sections:

Recording

Record relative element coordinates:

If checked, relative coordinates within a recognized element are recorded. Otherwise, actions will always be invoked on the "Center" of the recognized elements.

Capture a screenshot for each step:

Specifies whether a screenshot of the current action should be made during the recording.

Highlight clicked elements:

Specifies whether objects clicked during a recording should be highlighted.

Timings for new actions

Please note, that the following settings only affect newly created actions, not existing actions.

Key sequence split time (ms):

Use this setting to specify the maximum time for a key sequence during recording. If a key sequence exceeds the time set, it is split into several key sequences.

Mouse move time per action (ms):

Specifies the time in milliseconds used to move the mouse to the UI object related with the action.

Mouse related action time (ms):

Specifies the overall time in milliseconds used for a manually created mouse action. The value set for this setting is only taken into account for manually created actions or when the time of an action cannot be determined during recording.

Keyboard related action time (ms):

Specifies the default overall time in milliseconds used for a manually keyboard action. The value set for this setting is only taken into account for manually created actions or when the time of an action cannot be determined during recording.

Code Generation

Recording namespace:

Specifies the namespace used for the generated code.

Recording class name:

Specifies the class name used for the generated code.

Generate replay reports:

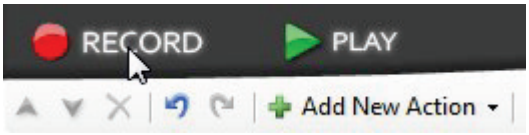
This setting is used to turn the generation of reports during replay on or off.

Use item logging by default:

Activate this setting to turn on a default logging message for each action item.

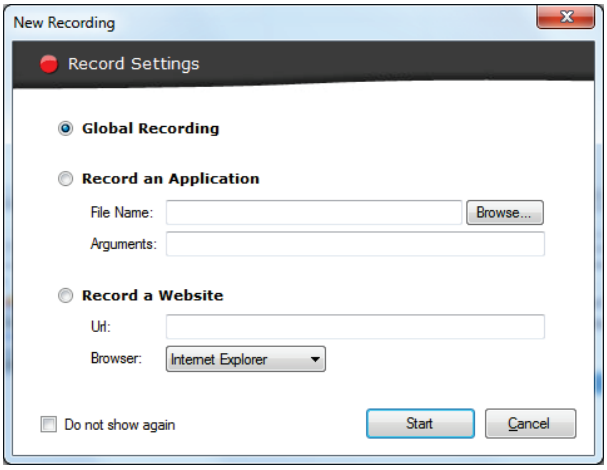
During Recording

Click the 'Record' button to trigger a new recording.



Click 'Record' to start recording

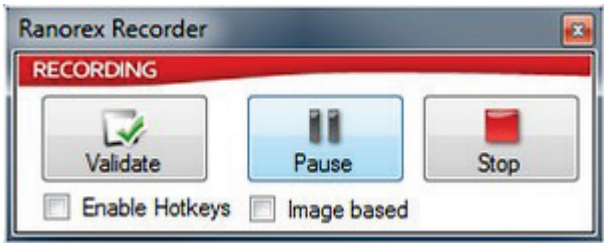
After clicking the record button you get assistance in running an application or opening a browser to navigate to a particular URL before recording any user performed actions. Therefore it's not necessary to record actions like double clicking a shortcut on the system's desktop.



'New Recording' dialog

Note: If you select 'Record a Website' or 'Record an Application' the recording is not limited to the application or browser.

After clicking the 'Start' button the Recorder or Ranorex Studio will be minimized. A running recording session is indicated by the Ranorex Recorder tool bar.



Ranorex Recorder toolbar

Now perform all the actions you want to be part of the automated test. You can also pause the recording session at any time by clicking the 'Pause' button.

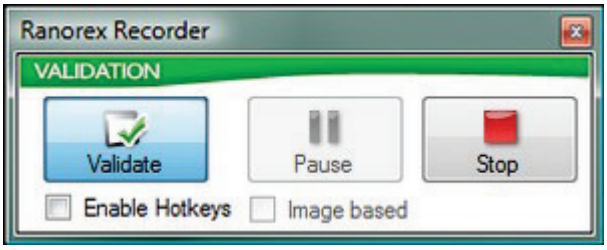
Validation

The Ranorex Recorder provides two different types of validation during a running recording session.

- **Validation based on property values of a UI element**
- **Image based validation**

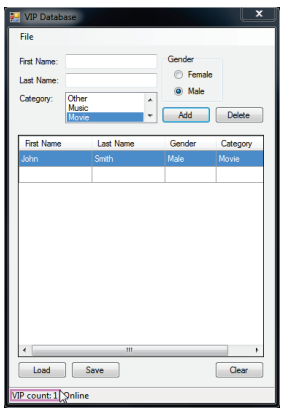
Validating States or Properties

The Ranorex Recorder tool bar can be used to validate states or properties of GUI elements during a running recording session. Click the 'Validate' button to pause the recording.

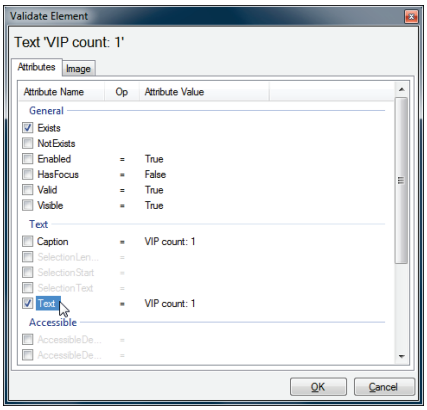


Validation mode is active

Recording is now paused. Move your mouse pointer over a specific UI element. Wait a moment to ensure that the text box is tracked and ready to be validated. Click it to open the validation dialog.

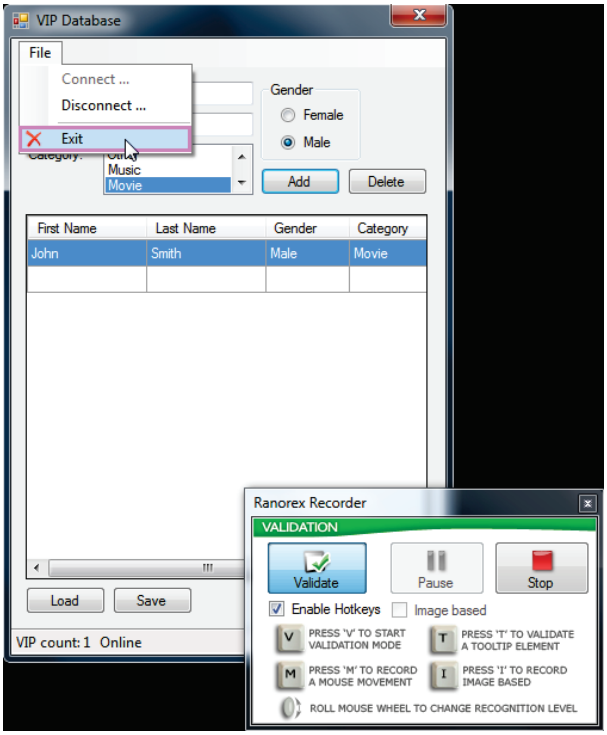


The highlight frame specifies the element to validate



The dialog shows the available attributes to check

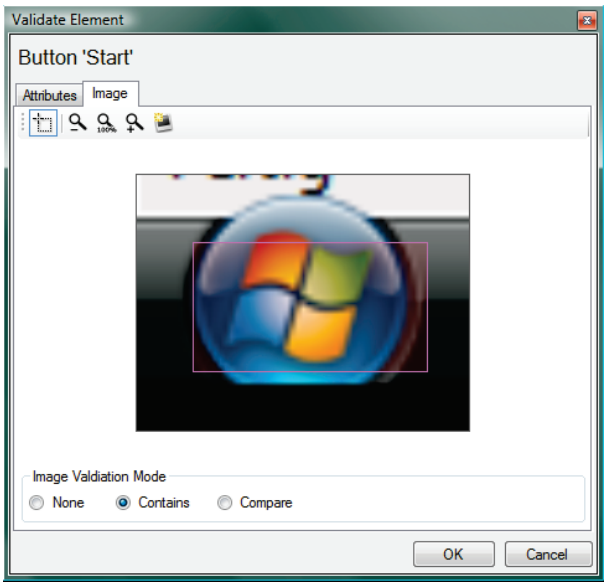
To validate UI elements of pop-up windows like menus, use the shortcut key 'V' to turn on validation mode without clicking the 'Validate' button. Read more about how to enable the Recorder's hot keys here.



Validation of menu items

Validating Images

In addition to validating attribute values of a UI element, it is also possible to validate images or screen regions of an application. Simply switch to validation mode as described above and select a UI element. To validate the element's image, open the 'Image' tab in the validation dialog.

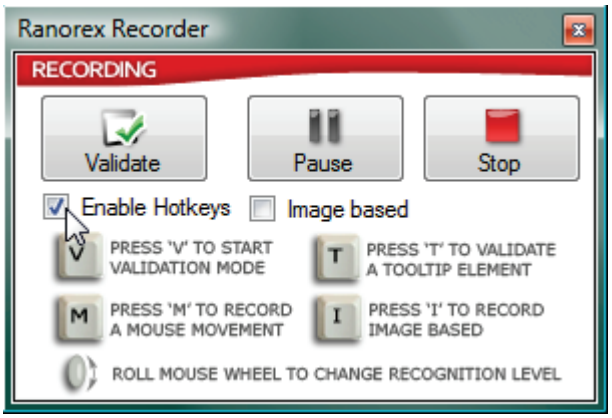


Validate the existence of an image region within the screenshot of Windows start button

Recorder Hotkeys

Ranorex Recorder provides useful hot keys for triggering special features like validation or the recording of simple mouse movements. When you start a new recording session, the hot key functionality is turned off. To activate the Recorder's hot keys simply click on the 'Enable Hotkeys' checkbox or press the master hot key <SCROLL>.

The recorder's toolbar will then show what type of shortcut keys are available.



Available hot keys during recording session

'V':
Press the hot key 'V' to turn on validation mode - especially useful for validating elements of a pop-up window like menu items and drop-down lists.

'M':

Use the hot key 'M' to record simple mouse movements. Simply move the mouse pointer to a certain position and press the hot key to record the mouse movement in relation to the underlying UI element.

'T':

It's easy to validate the content of a tooltip by pressing the hot key 'T' after the tooltip box appears. Ranorex Recorder automatically captures a mouse move action to the underlying item and validates the current content of the tooltip.

'I'

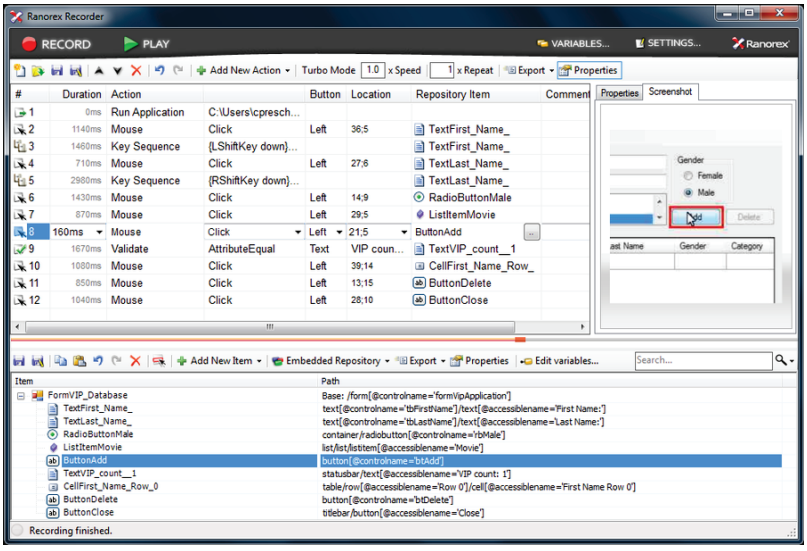
The hot key 'I' is used to turn on/off image based recording - used for pop-up windows, menu items or drop down lists.

'Roll Mouse Wheel'

Roll the mouse wheel during validation mode to change the level of UI element selection.

After Recording

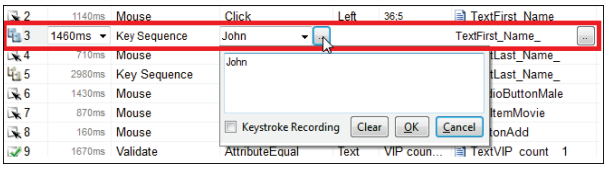
After pressing the 'Stop' button of the Recorder's toolbar the recorder represents all low level actions like clicking on a button or typing a text within the action's table.



Recorded actions listed in the recorder's actions table

By default each recorded action has an additional screenshot shown on the right. Depending on the type of action each row shows additional action-related information. Whereas a mouse related action provides more information about the type of mouse action (Click, Double-Click, Move, etc.) or which button was used (Left, Right, etc.) a key sequence action only stores the keystrokes hit during a recording.

Recorded mouse click action

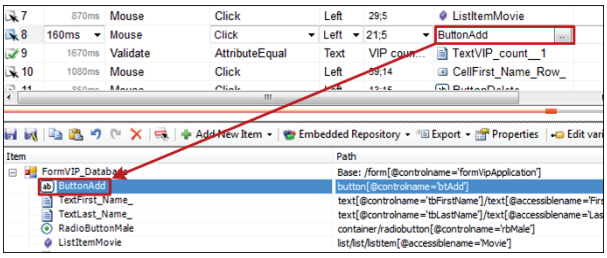


Recorded key sequence action

In addition each action can be connected to a UI element represented by the repository view below. For each UI element used during a recording session the recorder generates an item in the given repository.

Note: In case an item already exists in the repository or if the UI element is used twice during a recording session the recorder reuses this element.

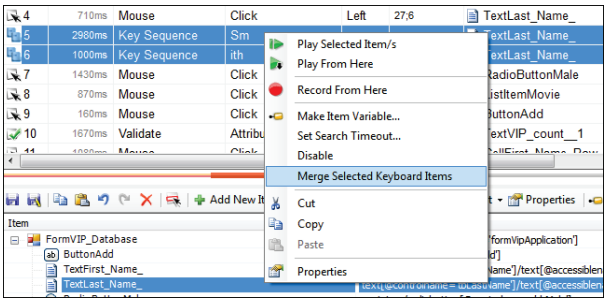
By default the repository is not part of the recording file and only refers to the main repository used within a project. In comparison to the integrated Recorder the standalone Recorder tool embeds the repository by default. You can also unlink and refer to another file or embed the repository into the recording file. Read more about how to do that here:



Button click action refers to repository item 'Button Add'

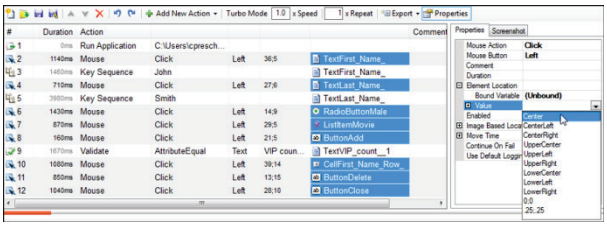
Cleaning up Actions

After finishing the recording, it is advisable to look at each recorded action item in detail. The recorder offers some editing features to clean up the sequence, for example, merging key sequence actions.



Merge selected keyboard items

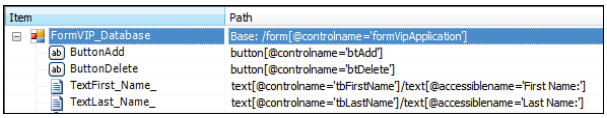
By default the recorder records the mouse click location performed by the user relative to a button or a text field. It's possible to turn that off in advance within the Recorder's settings dialog. Alternatively, you can also change the click location afterwards within the properties dialog.



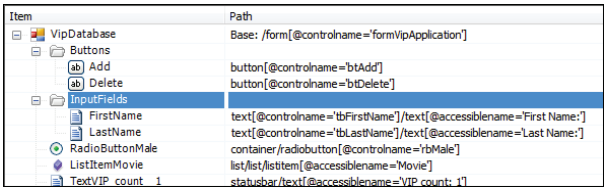
Changing click location for all mouse related action items to 'Center'

Cleaning up Repository

Each item within the repository refers to one RanoreXPath expression necessary to identify the object. The names used for the items and folders of the repository are generated automatically. It is recommended to rename them if necessary. In addition to that, the structure of the repository is generated automatically as well and depends on the internal UI hierarchy given by the application under test. To rename items of the repository simply click into the cell or press <F2> at the selected item.



Automatically generated repository

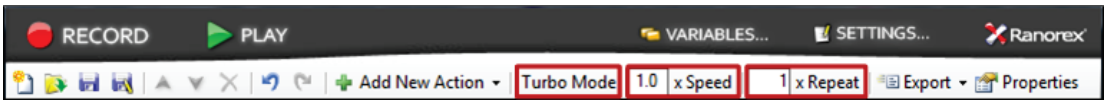


New structured repository

Use logical folders to create additional structure for example to group input fields, buttons or a log on form with user name and password text boxes. Also, already existing rooted folders can be used for grouping as well. Learn more about how to work with Ranorex repositories here: UI Mapping with Ranorex Repository

Replay and Debug Actions

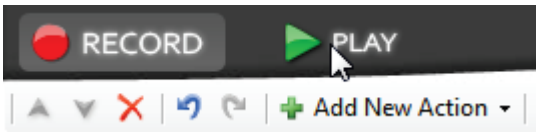
The playback speed of each recording can be defined individually. Use the speed factor text box to increase replay speed by a factor value.



Replay settings within Ranorex Recorder

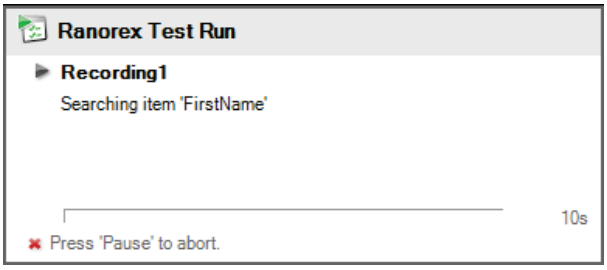
In order to replay all actions as fast as possible activate 'Turbo Mode' by clicking the button within on the tool bar. By activating 'Turbo Mode' the value for speed factor is ignored. Use the repeat count text box to specify the number of iterations.

Simply click the 'Play' button in the recording to replay all actions.



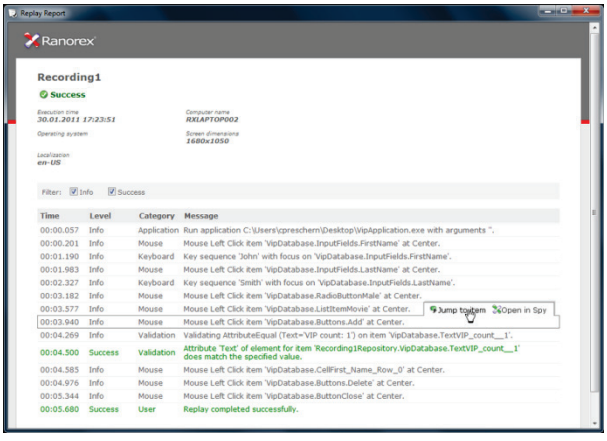
Replay recorded actions

During the execution, the progress box shown at the right lower corner gives information about the current automation state. Press the 'Pause' key to stop replaying.



Replay status box

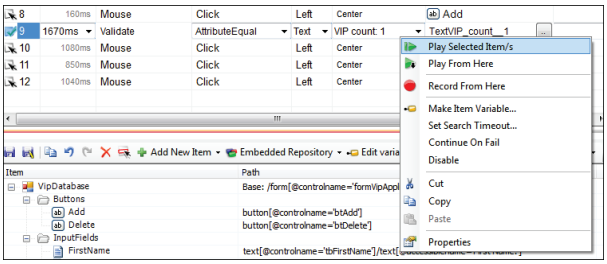
If the option 'Generate replay reports' is set to true a report is shown after the replay has finished. By default each action item logs one message to the report.



Report after replay

Within Ranorex Studio the report is shown as a new file view whereas the standalone Recorder opens the report viewer to show the result. Learn more about how to use Ranorex reporting here.

As shown in the picture above each log entry allows it to quickly jump to the source action item. This helps to quickly debug single actions in case of an error. Then within the Recorder's actions table you can fix an error causing step and quickly re-run single or selected steps using the context menu.



Play a selected item

Recorder Variables

As described in lesson 3 a recording can have variables instead of static string values generated during the recording session. Especially key sequence actions used to type in for example log in data into a form are commonly used action items which have to be data driven in many cases. At any position within the actions table where you can change or set values in a cell it is possible to have variables instead.

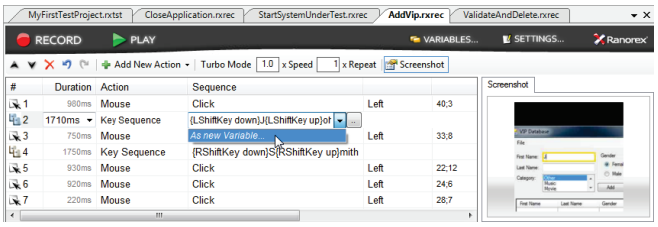
#	Duration	Action					Comment
1	1000ms	Key Sequence	$varText			FirstName	Simulates a key sequ...
2	0ms	Open Browser	$varBrowserUrl	IE			Opens a browser with...
3	0ms	Run Application	$varApplicationPath				Runs an application
4	1000ms	Key Shortcut	Press	$varKeyShort...		GenderOption	
5	0ms	Validate	AttributeEqual	Text	$varExpectedCount	TextVIPCount	Validates the value of ...
6	0ms	Set Value	Text	$varTextValue		LastName	Sets the value of a te...
7	0ms	Wait For Clo...	$varTimeToWait			MyApp	Waits for disappearan...

Variables used for different types of action items

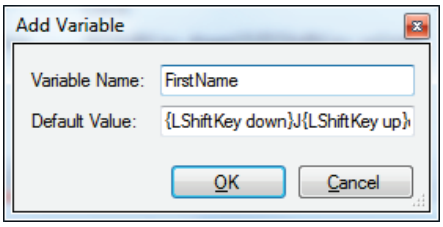
The variables shown in the table above are written in green font. All variables used within the recording can be connected to a Ranorex data connector or to simple parameters provided by the entire test suite or a single test case.

Creating Variables

There are different ways to create new variables for a recording. During the lesson 'Data Driven Testing' you learned how to create variables directly within the actions table.

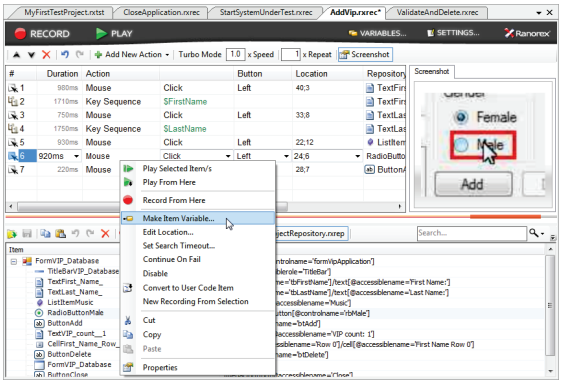


Create a new variable for a key sequence action item

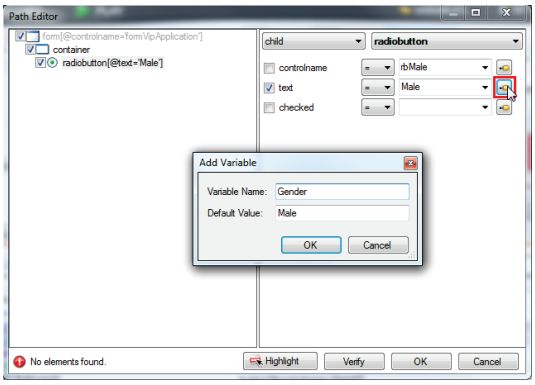


Create a new variable

Though there are not only key sequences which might use a variable. To make a click action data driven - for example selecting the radio button Male or Female - you need to define a repository variable used for identification within a RanoreXPath expression. Select the context menu item 'Make Item Variable ...' to create a new repository variable to have a data driven mouse click action.



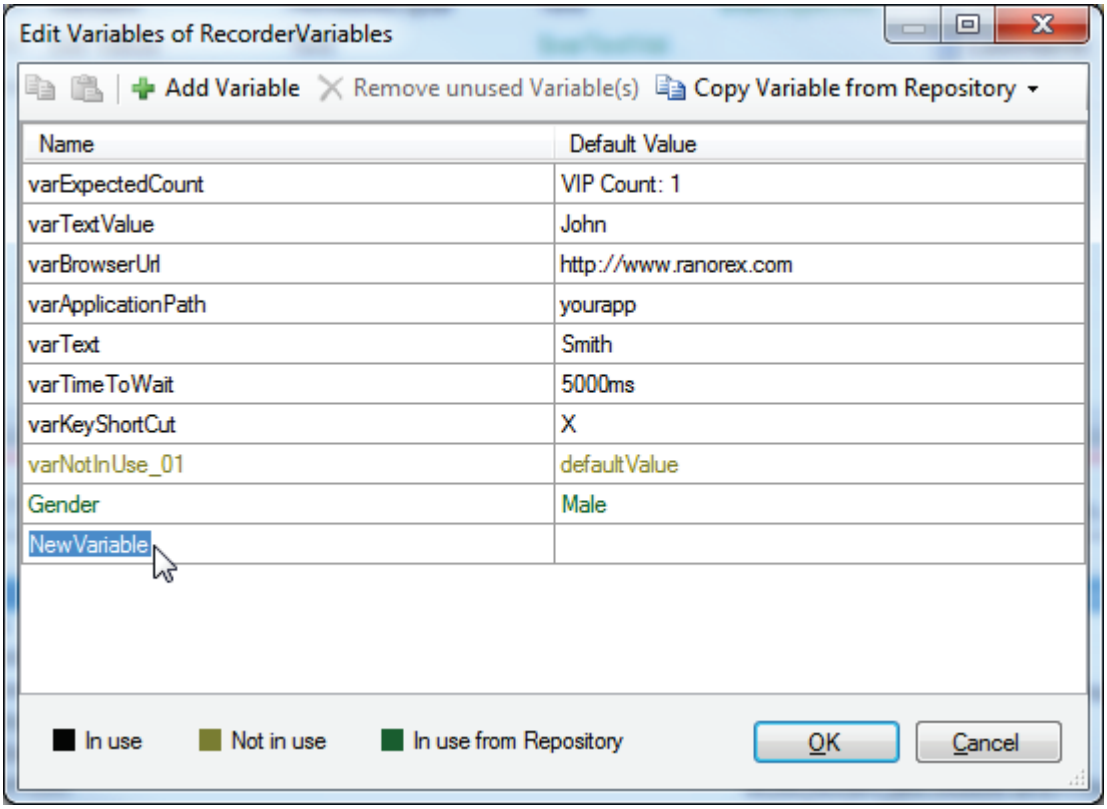
Make click action variable



Create a new variable for selecting a radio button

Another option is to open the variables dialog where you can add new or change existing variables already used by the recording. Moreover, you can see which variables are 'In use', 'Not in use' or 'In use from Repository'.

Opens the 'Variables' dialog



Variable dialog of a recording

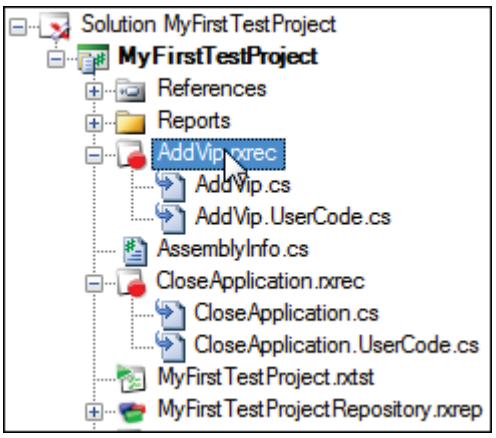
Use the toolbar to add, remove or copy a variable from the repository. Change the name of the variable and set a default value by clicking into the cells of the table. Press the 'Copy Variable from Repository' button at the tool bar to make variables defined at the given repository available for the recording. Read more about variables specified in repositories here.

User Code Actions

A user code action is used in situations where standard out-of-the-box features provided by the recorder are not enough. Here are some examples where a user code action could be a great convenience.

- **User Specific Validation**
- **Accessing Test Case Related Data and Parameters**
- **Extended Reporting**

Looking at a recording file in the project's view you will see that each recording has two code items.



Recording 'AddVIP' has two code file items

Note: User Code actions are not available within the standalone Recorder.

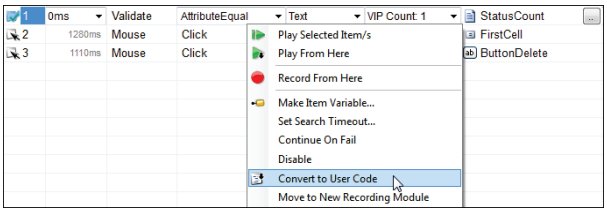
Within Ranorex Studio each recording manages two types of source code files.

- **Automatically generated main recording source code file <RecordingName>.<ScriptFileExtension>**
- **User specific source code file<RecordingName>.UserCode.<ScriptFileExtension>**

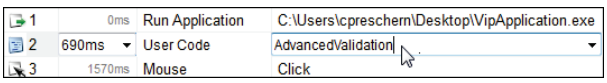
Every time you change and save a recording the main code file 'AddVip.cs' is generated newly. Any change of code should always be made within the recording's UserCode.cs file.

Create User Code Actions

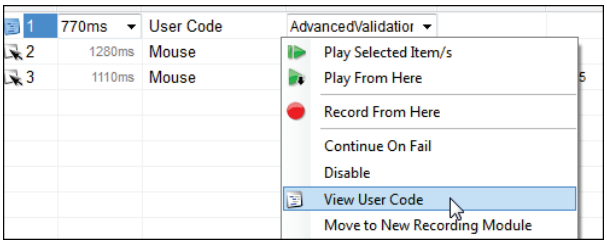
You can create user specific code actions by converting existing items or by adding a new 'UserCode' action item from the toolbar button 'Add New Action'.



Convert an existing action item to a code action



Specify method name used for user code item



Use the context menu item 'View UserCode' to jump into the code

After creation of a new user code action within the actions table a new method is added to the partial class of the recording. In case of converting an existing action item the code generated for the main recording file is transferred to the new usercode method.

C#

```
namespace VIPTestSuite.RecorderModules
{
    public partial class AddVIP
    {
        /// <summary>
        /// This method gets called right after the recording has been started.
        /// It can be used to execute recording specific initialization code.
        /// </summary>
        private void Init()
        {
            // Your recording specific initialization code goes here.
        }

                public void AdvancedValidation()
                {
                        // Your code goes here. Code inside this method will
                        // not be changed by the code generator.
                        Report.Log(ReportLevel.Info, "Validation", "Validating At-
tributeEqual (Text='yourtext') on item 'VipApp.StatusLine.Count'.", re-
po.VipApp.StatusLine.CountInfo, new RecordItemIndex(-1));
                        Validate.Attribute(repo.VipApp.StatusLine.CountInfo, "Text",
"yourtext");
                }
    }
}
```

VB.NET

```
Namespace MyFirstTestProject

        Public Partial Class AddVip

                ''' <summary>
                ''' This method gets called right after the recording has been started.
                ''' It can be used to execute recording specific initialization code.
                ''' </summary>
                Private Sub Init()
                        ' Your recording specific initialization code goes here.
                End Sub

                Public Sub AdvancedValidation()
                        ' Your code goes here. Code inside this method will not be
changed by the code generator.
```

```
                        Validate.Attribute(repo.VipApp.StatusLine.CountInfo, "Text",
"yourtext")

                End Sub

        End Class
End Namespace
```

In this example the validation of the text shown at the status bar depends on the number of added persons. In case of a data driven execution you can use the test case context in code to ask for the current iteration index.

C#

```
public void AdvancedValidation()
{
        // In case of a data driven context the exptected text value
        // shown in the status bar depends on the current iteration
        string validationText = "VIP Count: ";
        // User the test case context to access the data context
        if ( TestCase.Current.DataContext.CurrentRowIndex > 0 )
                validationText = validationText + Test-
Case.Current.DataContext.CurrentRowIndex.ToString();
        else
                validationText = validationText + "1";

        Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual
(Text='"+validationText+"') on item 'VipApp.StatusLine.Count'.", re-
po.VipApp.StatusLine.CountInfo, new RecordItemIndex(-1));
        Validate.Attribute(repo.VipApp.StatusLine.CountInfo, "Text", validationText);
}
```

VB.NET

```
Public Sub AdvancedValidation()
        ' In case of a data driven context the exptected text value
        ' shown in the status bar depends on the current iteration
        Dim validationText As String = "VIP Count: "
        ' User the test case context to access the data context
        If TestCase.Current.DataContext.CurrentRowIndex > 0 Then
                validationText = validationText & Test-
Case.Current.DataContext.CurrentRowIndex.ToString()
        Else
                validationText = validationText & "1"
        End If

        Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual (Text='" &
validationText & "') on item 'VipApp.StatusLine.Count'.", repo.VipApp.StatusLine.CountInfo,
New RecordItemIndex(-1))
        Validate.Attribute(repo.VipApp.StatusLine.CountInfo, "Text", validationText)
End Sub
```

Conditions in Code

Another reason for writing user code is to read text values from UI elements like text boxes and to reuse them for conditional automation steps.

C#

```
public void CheckConnectItemState()
{
	// Your code goes here. Code inside this method
	// will not be changed by the code generator.
	Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual (Ena-
bled='False') on item 'FileMenu.Connect'.", repo.FileMenu.ConnectInfo, new RecordItemIn-
dex(-1));
	if ( repo.VipApp.StatusLine.Connection.TextValue.Equals("Online") )
		Validate.Attribute(repo.FileMenu.ConnectInfo, "Enabled", "False");
	else
		Validate.Attribute(repo.FileMenu.ConnectInfo, "Enabled", "True");
}

public void CheckDisconnectItemState()
{
	// Your code goes here. Code inside this
	// method will not be changed by the code generator.
	Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual (Ena-
bled='True') on item 'FileMenu.Disconnect'.", repo.FileMenu.DisconnectInfo, new Record-
ItemIndex(-1));
	if ( repo.VipApp.StatusLine.Connection.TextValue.Equals("Offline") )
		Validate.Attribute(repo.FileMenu.DisconnectInfo, "Enabled", "False");
	else
		Validate.Attribute(repo.FileMenu.DisconnectInfo, "Enabled", "True");
}
```

VB.NET

```
Public Sub CheckConnectItemState()
	' Your code goes here. Code inside this method
	' will not be changed by the code generator.
	Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual (Ena-
bled='False') on item 'FileMenu.Connect'.", repo.FileMenu.ConnectInfo, New RecordItemIn-
dex(-1))
	If repo.VipApp.StatusLine.Connection.TextValue.Equals("Online") Then
		Validate.Attribute(repo.FileMenu.ConnectInfo, "Enabled", "False")
	Else
		Validate.Attribute(repo.FileMenu.ConnectInfo, "Enabled", "True")
	End If
End Sub

Public Sub CheckDisconnectItemState()
	' Your code goes here. Code inside this
	' method will not be changed by the code generator.
	Report.Log(ReportLevel.Info, "Validation", "Validating AttributeEqual (Ena-
bled='True') on item 'FileMenu.Disconnect'.", repo.FileMenu.DisconnectInfo, New Record-
ItemIndex(-1))
	If repo.VipApp.StatusLine.Connection.TextValue.Equals("Offline") Then
		Validate.Attribute(repo.FileMenu.DisconnectInfo, "Enabled", "False")
	Else
		Validate.Attribute(repo.FileMenu.DisconnectInfo, "Enabled", "True")
	End If
End Sub
```

Note: Only create small and easy to maintain user code actions for a recording. If an implemented method should be available for other test cases too, create a Code Module instead of it.

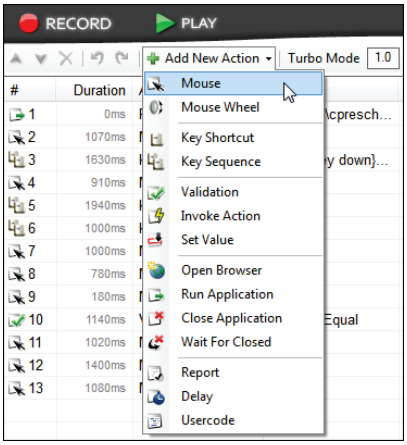
Additional Editing Options

As you have learned during the lessons before, the Recorder is typically used to record user actions. After recording a session it might be necessary to edit the recorded data for example to merge split key sequences or to delete single actions recorded by mistake. You can also add new actions like a new validation step which was not considered during a recording. In the following section you learn about:

- **Adding New Actions**
- **Recorder Variables**
- **Splitting Recordings**

Adding New Actions

To add a new action item open the 'Add New Action' drop down box as shown below:

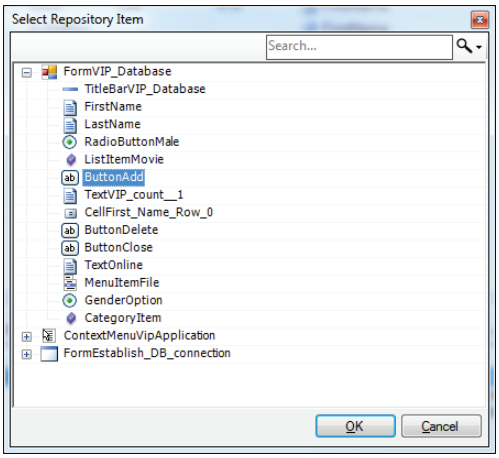


Add a new action item to the actions table

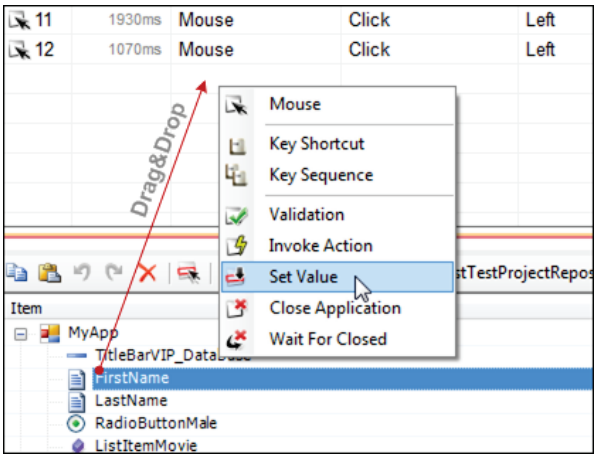
By selecting one of the items a new action is added after the current selection within the actions table.

New mouse move action added

Depending on the type of action item you can configure each action individually. Use the drop down buttons within each cell, for example to change the action from 'Click' to 'Move' as it is shown in the picture above. In order to specify on which UI element the action should be performed click the small button framed in red in the graphic above.



Select a repository item to be used by the action



Use drag&drop feature to create a new action for a particular repository item

The items listed within the 'Select Repository Item' dialog are the same as shown within the recording's repository. You can use the 'Search' text box to filter for elements. Read more about how to use repository search and how to add new UI elements here. Alternatively, you can also drag and drop repository items into the actions table of a recording as shown in the graphic on the right.

Types of Action Items

	Mouse	Adds a new mouse action item at the current position
	Mouse Wheel	Adds a new mouse wheel action item
	Key Shortcut	Adds a new key shortcut action item (e.g. {F1}, {ENTER})
	Key Sequence	Adds a new key sequence action item (e.g. "Hello")
	Validation	Adds a new validation action item
	Invoke Action	Adds a new invoke action item (e.g. call method 'Select' for a UI element of type list item)
	Set Value	Adds a new set value action to set an attribute value (e.g. 'Text' for a UI element of type text)
	Open Browser	Opens a browser and navigates to the given URL
	Run Application	Runs an application at the given directory and file path

	Close Application	Closes an application or web site containing the given repository item, respectively
	Wait For Not Exists	Waits for disappearance of the given repository item **Note:** This action can be used for any type of UI element.
	Report	Adds a new report action item
	Delay	Adds a new delay action item
	User Code	Adds a new user code action item which is used to implement a user specific validation in code for example

Validating Attributes

Use the action item 'Validate' to check the attributes, images or different states of a repository item. Open the property grid to configure the behavior of a failed or successful validation individually.

Action	Specifies the type of validation (e.g. Exists, NotExists, AttributeEqual, etc.)
Continue on Fail	Specifies whether to stop replaying in case of a failed validation or not
Create Screenshot	Specifies when to create a screenshot of the validated element (Never, OnFail, OnSucess or Always)
Match Name	Specifies the name of the item's attribute to check
Match Value	Specifies the expected value used for validation
Report Level On Failure	Specifies the report level in case of a failed validation **Note:** This setting has no effect on test case error behavior
Report Level On Success	Specifies the report level in case of a succeeded validation

Continue On Fail and Disable

Each action item listed in the table can be disabled or set to 'Continue On Fail'. Set an action item to 'Continue On Fail' if, in case of an error, the replay should not stop at that position. You can set both options via the context menu or the property grid.

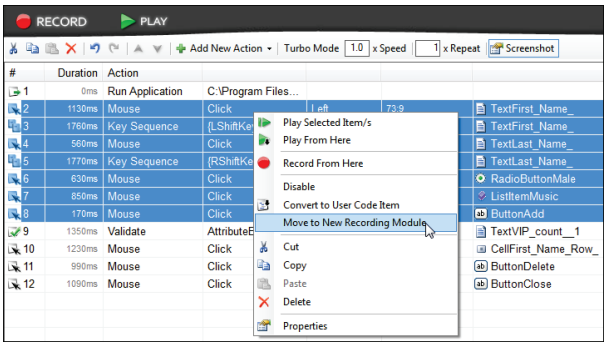
#	Duration	Action				
[1]	*770ms*	*Validate*	*AttributeEqual*	*Text*	*VIP count: 1*	*TextVIPCount*
2	1280ms	Mouse	Click	Left	41;5	CellFirst_Name_Row_0
3	1110ms	Mouse	Click	Left	17;15	ButtonDelete

Disabled action in gray font color - Continue-On-Fail action using italic font style

Note: In case of an error the particular action logs a warning to the report.

Splitting of Recordings

The more recorded actions you have after finishing a new recording, the less clear each single action becomes. As already mentioned within Lesson 2 Ranorex Modules - A Test in Parts, it is recommended to identify reusable steps within a newly created recording. Use the context menu item 'New Recording From Selection' to create a new recording module.

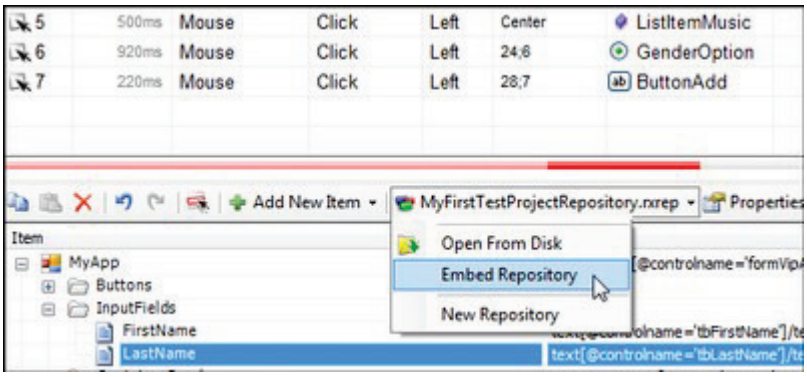


Creates a new Recording from the selected items

Use the test suite editor as described during Lesson 2 or Lesson 4 to combine multiple recording modules within a test case.

Change Repository of Recording Module

By default each newly created recording refers to the main repository file. To create a new repository or to refer to another repository simply open the drop down menu from the repositories toolbar as shown below.

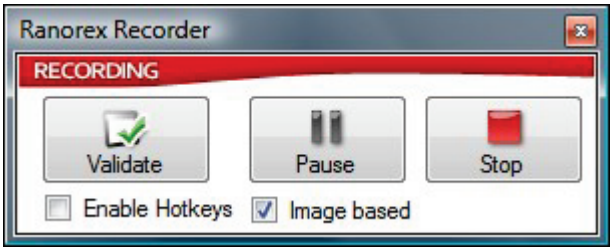


Change referring repository

Select 'Open From Disk' if you want to open and refer to a repository which is currently not part of your Ranorex Studio projects. If you want to use a repository exclusively for a single recording, simply embed it into the recording file. In this case all the repository items are saved to the recording file. Creating a new repository for your recording automatically adds a new repository file to the current project.

Image-Based Automation

Sometimes you need to automate a click action based on image information. For this reason, Ranorex Recorder provides an optional way to capture image related mouse actions. To activate image based recording, simply check the 'Image-Based' checkbox in the Recorder's toolbar during a recording session.



'Image-Based' recording activated

Now, move the mouse pointer over a certain GUI element. Ranorex highlights the last accessible UI element, and also highlights a recognized image within the GUI element.

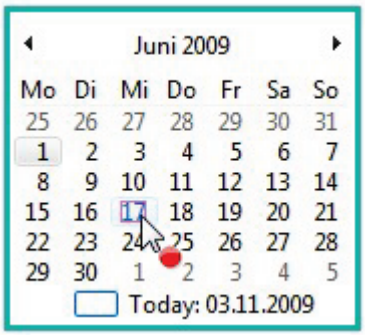


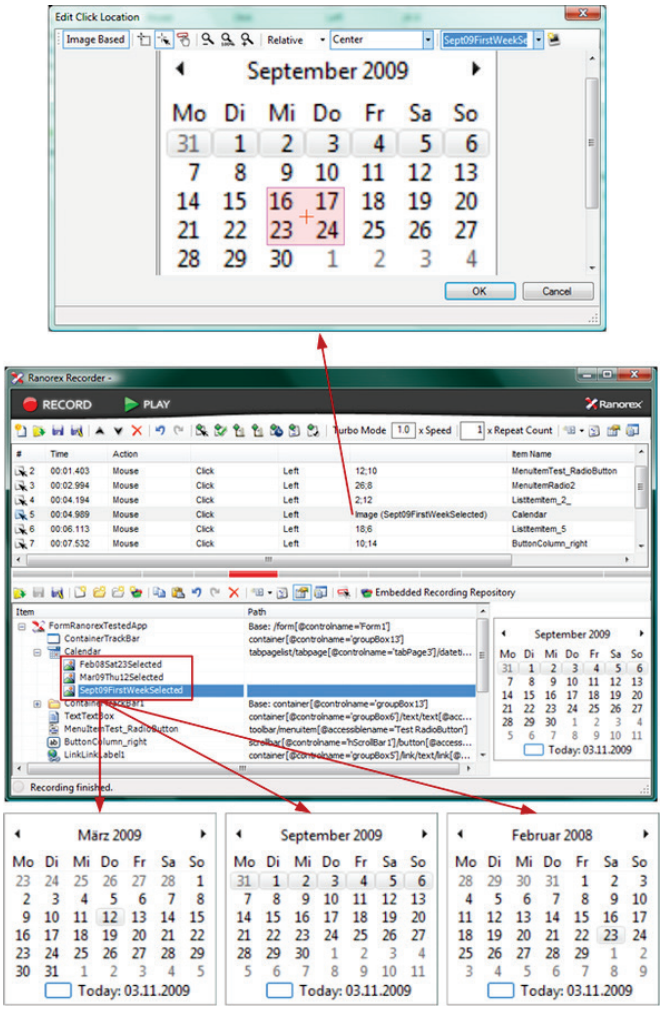
Image-based recording within calendar control

To turn off image-based recording, uncheck the checkbox or press the shortcut key 'I'.

Note: Hot key functionality has to be activated by first using the master hot key 'SCROLL'. Read more about hot key use during Ranorex recording.

After stopping the recording session the actions table contains an image based mouse click action which is indicated by the 'Image' keyword. In addition the related repository item stores a snapshot at the time of recording.

Generally, the repository item saves an image to specify where to search for the image region which is specified by the image based mouse action in the recorder's actions table.



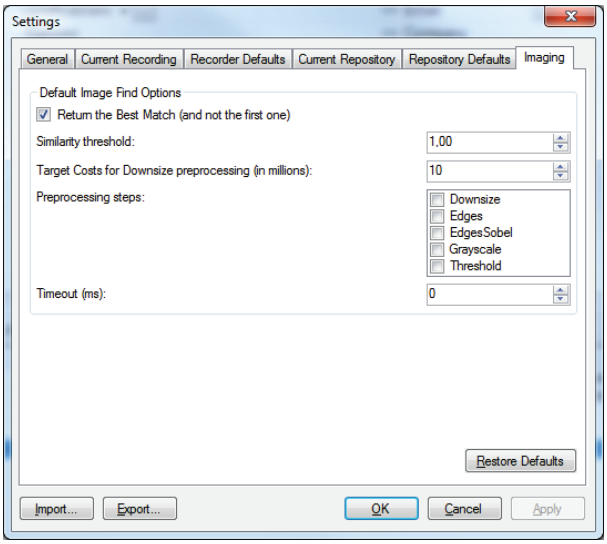
Separation of 'where' and 'what' to search for

To alter the settings of an image-based automation action, open the property group 'Image-Based Location' within property grid.

Image-Based Location Settings

Advanced Options	Use the property group to set advanced image search options like 'Clipping' or 'Best Match' If 'Best Match' is set to true, the result position with the highest similarity is used for validation. If is set to false, the first available result position will be used of the validation. The first one is more accurate and the second one is faster.
Preprocessing	Defines preprocessing steps that can be performed on an image before search (e.g. GrayScale or DownSize)
Screenshot Name	Specifies the name of the screenshot where to search for.
Selection Rectangle	Defines the image selection region and what to search for
Similarity	Specifies the minimum similarity that the image region to search for needs to have in common with the image in order to be considered a match

To set up the default values for image based recording open the 'Settings' dialog and the 'Imaging' tab.

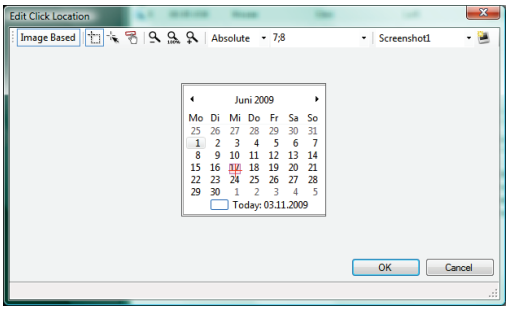


Settings dialog for image based recording

Image Editor

Use the context menu item 'Edit Location ...' to open the image editor and to change the click location options.

The editor provides a more detailed view of the captured image information. In addition it offers some useful features to change the image rectangle to be searched for.



Click location dialog to change the settings of image based mouse actions

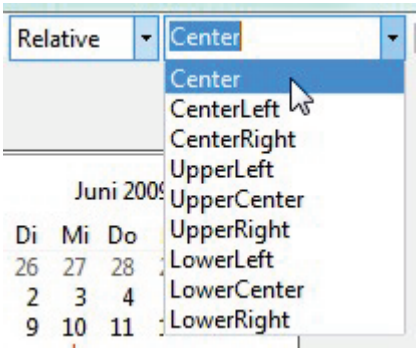
Generally the edit mode is indicated via the yellow highlighted editor box and it allows using the RanoreX-Path expressions as a query. After indicating the completion of editing RanoreXPath by pressing the <Enter> key, the result is shown within the spy tree as follows:

Toolbar Shortcuts

	Image Based	Specifies whether the click is based on image information or not
	Select Image Find Region	Specifies the region to search for within the image
	Select Click Location	Defines the click position related to the searched region
	Autoselect Image Find Region	Helps to specify the search region within the image
	Zoom Out	Zooms out
	Zoom 100%	Switches to 1:1 view

	Zoom In	Zooms in
	Capture New Screenshot	Captures a new screenshot from the related repository item

Use the drop-down combo boxes to specify whether a mouse action should be performed relative to the image region found or not.



Set relative or absolute mouse action location

If the captured screenshot has to be updated click the 'Capture New Screenshot' toolbar button. Also, every newly created screenshot is automatically added to the image list of the related repository item so it's available for other image-based validation or automation actions.

Add a new image or select an existing image to specify where to search

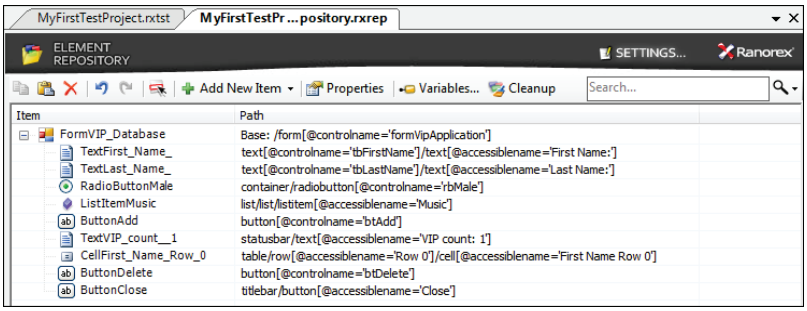
Lesson 6: UI Mapping with Ranorex Repository

The Ranorex Element Repository is used to separate identification information (RanoreXPath) from Ranorex test automation modules. With regard to test maintenance it is recommended that you also use Ranorex repositories within code modules to reduce the effort in adaptions necessary when the UI under test changes. In the following lesson you learn:

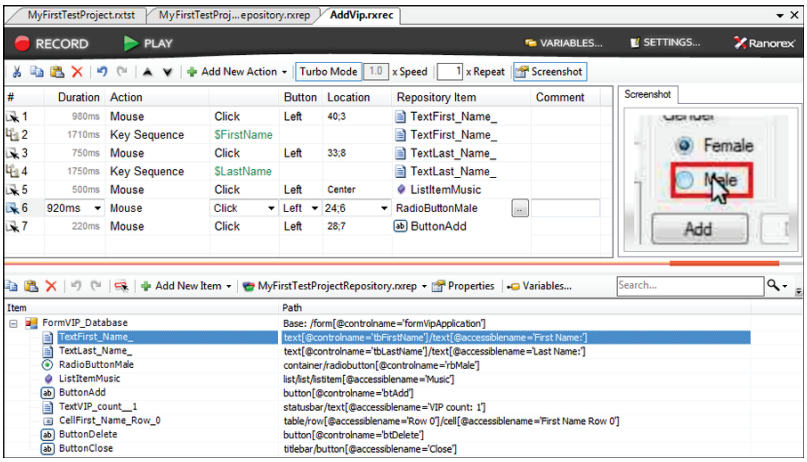
- **Adapting an Existing Repository**
- **Adding Repository Items**
- **Waiting for UI Elements - Repository Timeouts**
- **Editing RanoreXPath**
- **Repository Separation**
- **Repository Settings Dialog**

Adapting an Existing Repository

During lessons 1-3 you already created a repository by recording a manual test scenario. For each UI element used during the recording, a new item was created within the repository. By default a new Ranorex Studio project contains one repository file (*.rxrep) which can be used by multiple recording or code modules.



File view of a repository

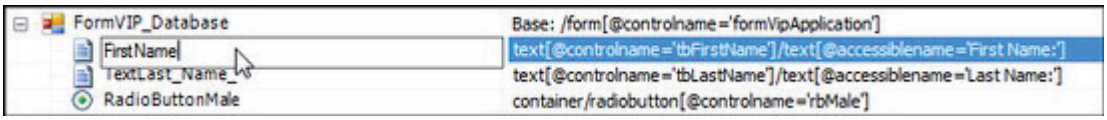


Integrated repository view within the Recorder

You can access and edit the repository within a recording or by double-clicking the file ('MyFirstTestProjectRepository.rxrep') in the projects view as shown in the figures above.

Renaming Repository Items

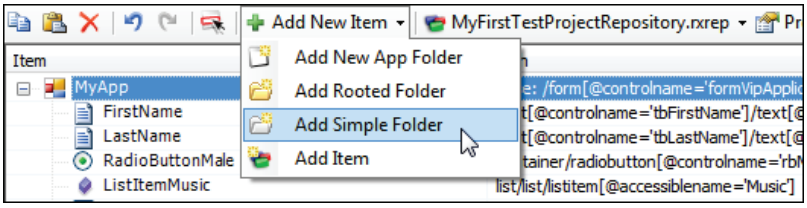
Each UI element within the repository can have a logical user-defined name. The more logical names you use the easier it is to understand test automation code and report files. In order to rename an item in the repository fist select it and then click the item to enter edit mode. Optionally you can use the keyboard shortcut <F2> to edit the names.



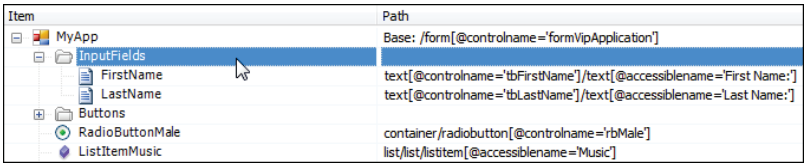
Renaming of repository items

Creating Logical Folders

The more objects you have in a repository the more structured and organized it should be. For this reason you can structure and group UI elements which logically belong together. Add a new 'Simple Folder' using the drop-down button at the repository toolbar.



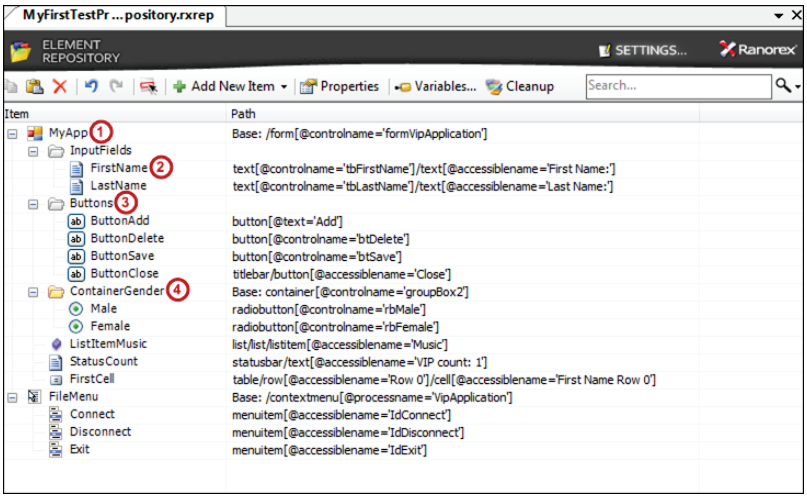
Adding a 'Simple Folder' to the repository



Repository using two logical folders to group input fields and buttons

Repository Structure - Types of Elements

A repository can have the following types of items:



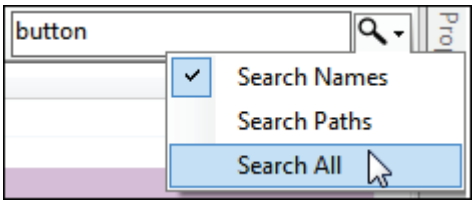
Ranorex repository structure

	Application Folder #1	Represents a top level application, a dialog or a context menu
ab	Adapter Item #2	Represents a Ranorex adapter (Button, ListItem, TextBox, ComboBox, etc.)
	Simple Folder #3	Used to group items
	Rooted Folder #4	Contains GUI elements having the same parent or RanoreXPath substring

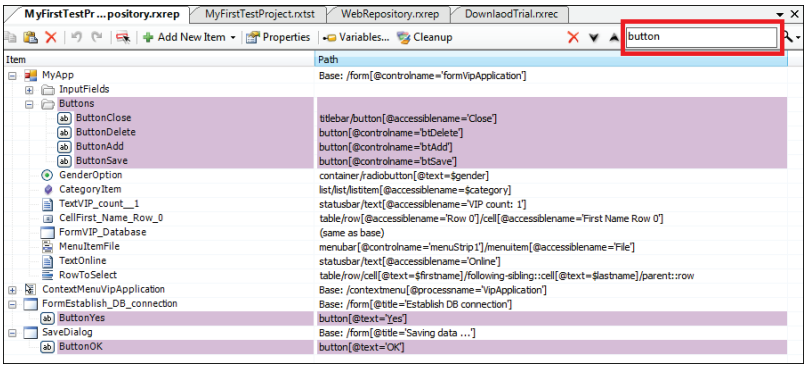
Use the 'Add New Item' button at the toolbar to add new items manually.

Searching For Elements

Use the 'Search' text box to find elements in the repository. Specify where to search for the given text value using the drop down menu.



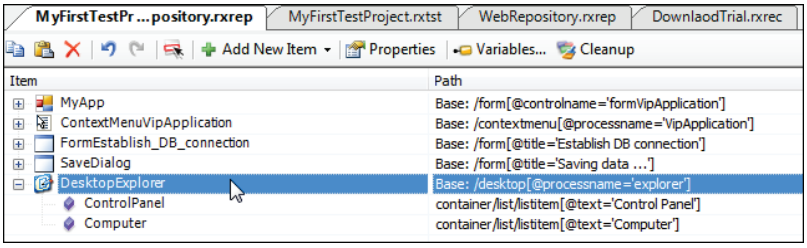
Specify whether to search in 'Names', 'Paths' or to 'Search All' elements



Search result for text 'button' used in item names

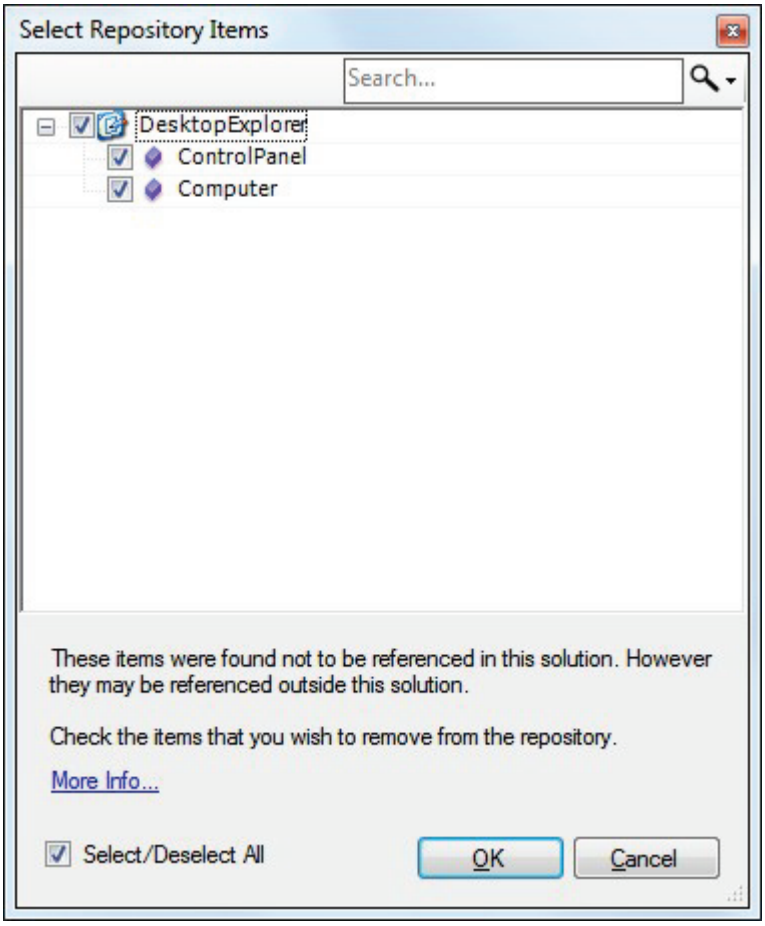
Repository Cleanup

The repository's 'Cleanup' button is used to search for currently unused items and then to delete items which are not used by a recording or code module. Use the Ranorex Spy to add some new elements to your repository like two desktop items shown in the figure below.



New items from the system desktop added to the repository

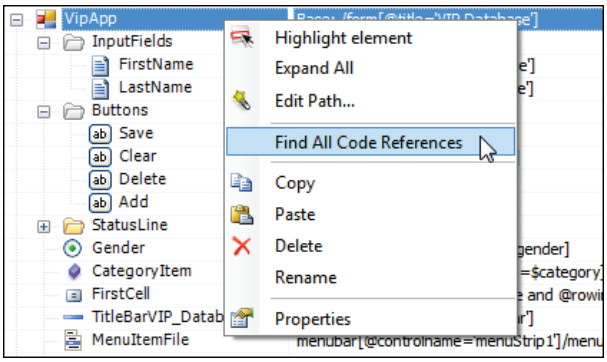
After clicking the 'Cleanup' button a dialog shows which elements of the repository are currently not in use by any of recording or code module within the project.



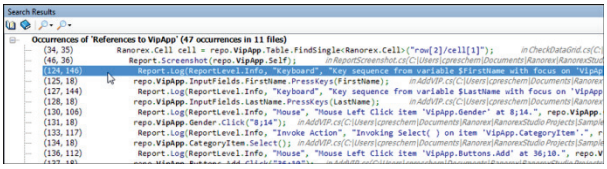
Select items to delete from the repository

Find Repository References

In lesson 7 you will learn about how to use repository items in code modules. Before you modify an existing repository item (e.g. the item's name or the item's RanoreXPath) it might be necessary to check in which code files the item is used. Use the context menu item 'Find All Code References' to list all code files using a certain repository item.



Search for references of repository item VipApp

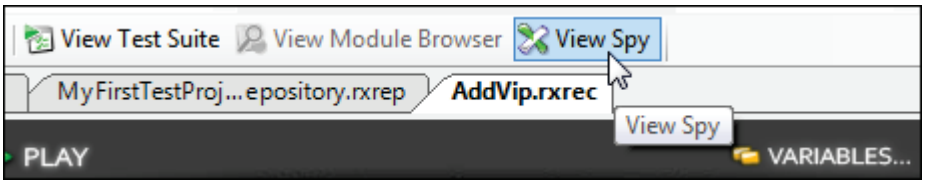


Search result - Simply double-click a result item to open the code file

Adding Repository Items

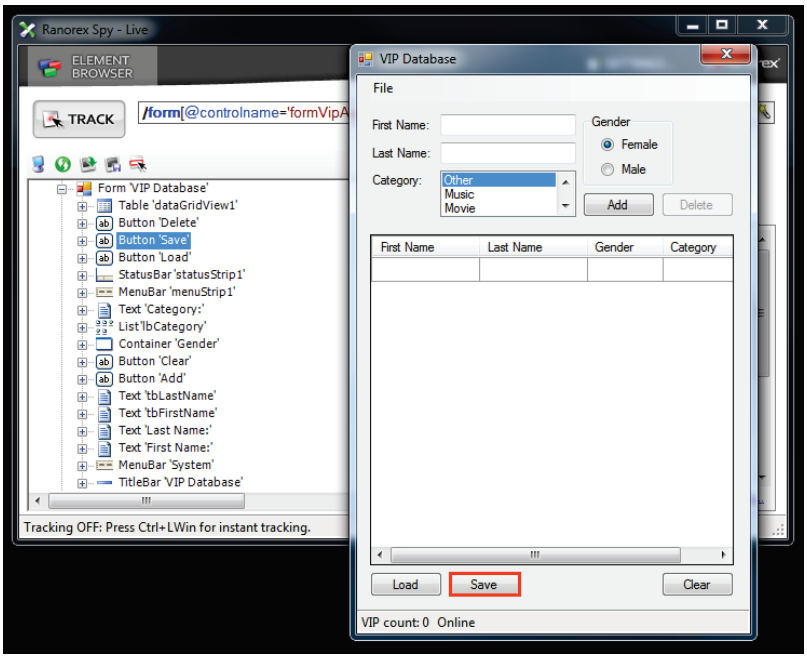
You can also add new items to your repository using Ranorex Spy tool. The following section describes how to add the 'Save' button which was not clicked during the recording session in order to automate a click on the button later on.

Firstly click the 'View Spy' button within the Ranorex Studio toolbar to open Ranorex Spy.



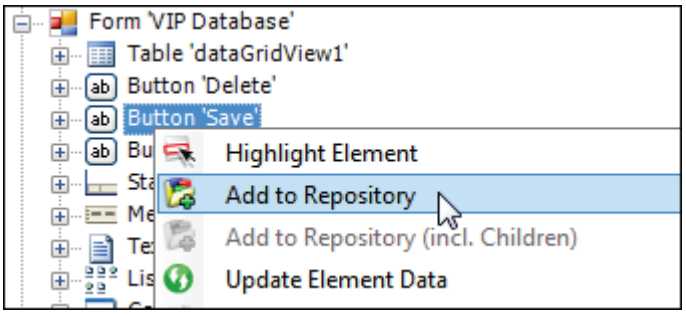
Open Ranorex Spy to track and to add items to the repository

Secondly track the 'Save' button using the 'Track' button of Ranorex Spy.

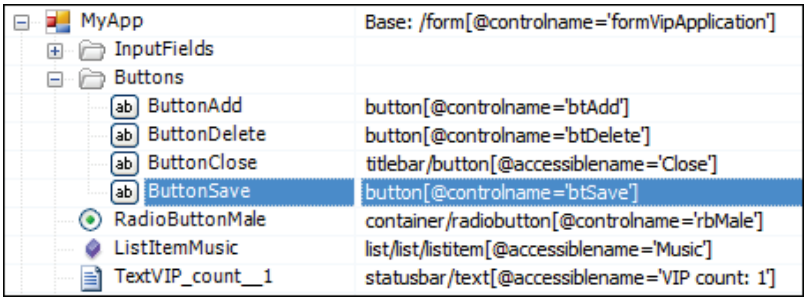


Track and identify the 'Save' button with Ranorex Spy

Third add it to the repository as follows:



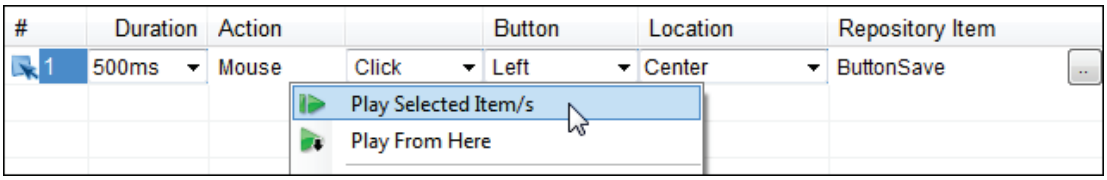
Add the item to the repository



Newly created item in the repository

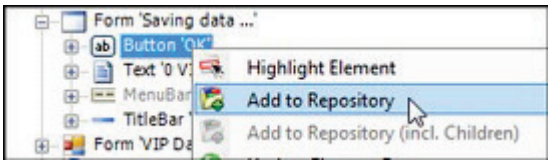
Now the 'Save' button is part of the repository and ready to be used by code or recording modules.

Create a new recording module to handle the 'Save' scenario. Use the 'Add New Action' drop down button at the toolbar and add a new 'Click' action. Finally connect the new action with the 'Save' button from the repository.

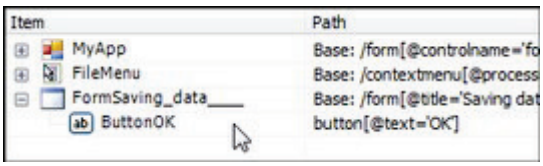


Execute the single action to simulate a click on the 'Save' button

If the newly added click action was simulated correctly, a dialog is shown by the VIP application. You have to finish the save recording scenario with a click action on 'OK'. Therefore you need to track the dialog's 'OK' button, add it to the repository and finally add a new click action to the recording the same way you previously did for the 'Save' button.

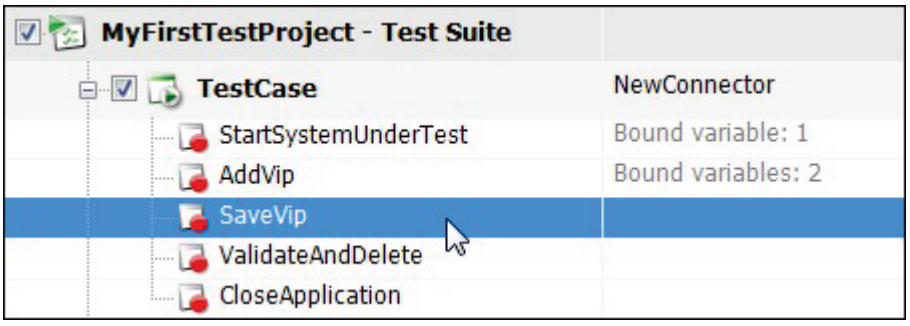


Add tracked 'OK' button to the repository



'OK' button with application folder

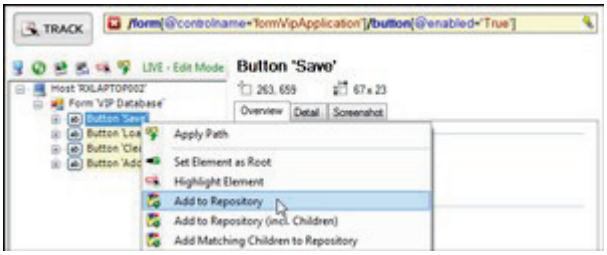
By adding the 'OK' button to the repository a new application folder for the dialog with title 'Saving data ...' was created. Now extend the test case you created at the beginning of this tutorial with the 'SaveVip' module and run the test case.



Extend test case with 'SaveVip' module

One Repository Item For Multiple Elements

Repository items can not only be assigned to RanoreXPath expressions reflecting a single UI element. You can create a repository item using a RanoreXPath which delivers multiple UI elements too. Use the Ranorex Spy to prepare a RanoreXPath for a multiple result and add it to the repository as described in the previous section.

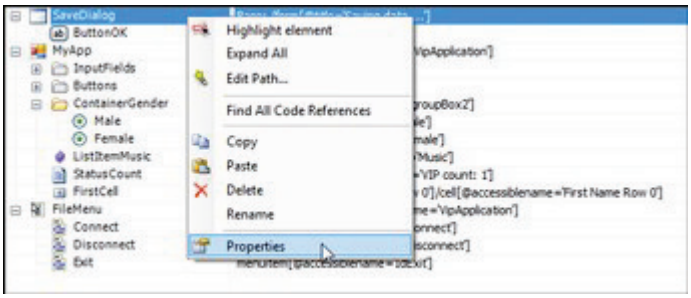


RanoreXPath delivering multiple results

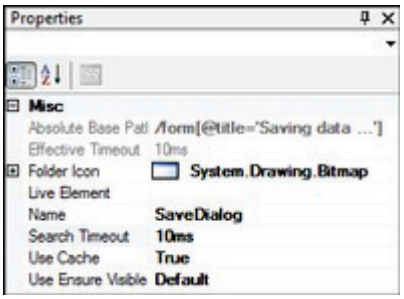
Repository items delivering a list of elements are mainly used in code modules as described in the chapter Code Examples.

Waiting for UI Elements - Repository Timeouts

You might have recognized that the dialog showing up after clicking the 'Save' button takes a few seconds to appear. This depends on the number of VIPs listed in the table. To handle a scenario like that, you can set a timeout value for each item and folder of a repository. Use the context menu item 'Properties' to open the properties for the application folder which represents the save dialog.

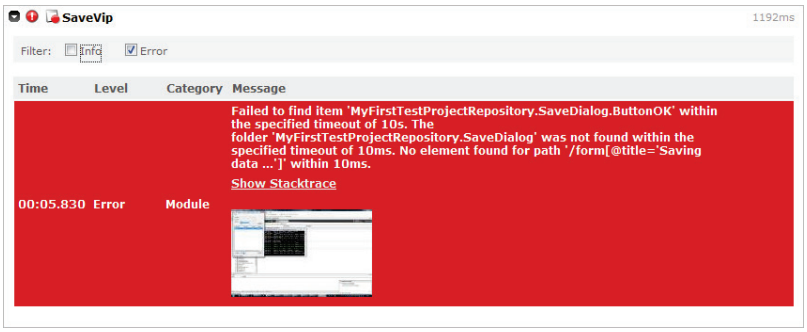


Open properties of 'SaveDialog' application folder



Set 'Search Timeout' to 10ms in order to force a timeout error

Set the dialog's search timeout to 10 milliseconds in order to force an error when executing the test case again.

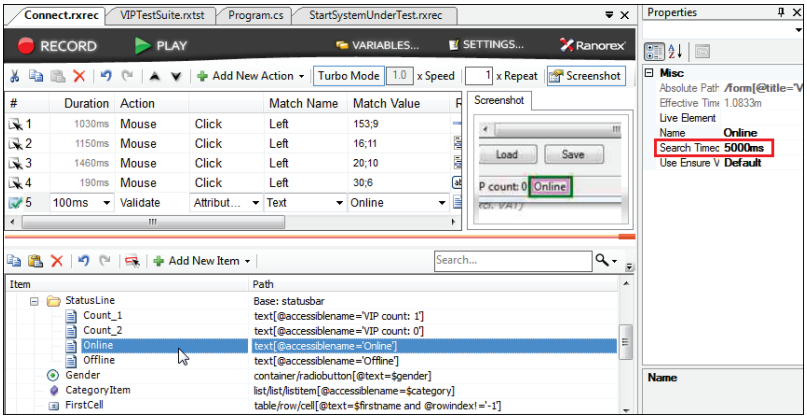


Test case execution failed - 'SaveDialog' was not found within the specified timeout of 10ms

After reducing the timeout the test case execution fails. The error message shown in the picture above logs that the 'SaveDialog' repository item was not found within the specified timeout.

Waiting for a Particular UI Element State - Event Handling

You can also use the timeout approach to wait for text states e.g. changing text values. Open the 'Connection' test case part of the VIPTestSuite project example. During the test the application first disconnects and then connects again. The last action of the recording module 'Connect' used within the test case validates the status text shown in the application's status bar. It takes some time to connect the application and finally to show the 'Online' text value in the status text. For this reason the validation action refers to a repository item which uses a timeout to wait for the RanoreXPath



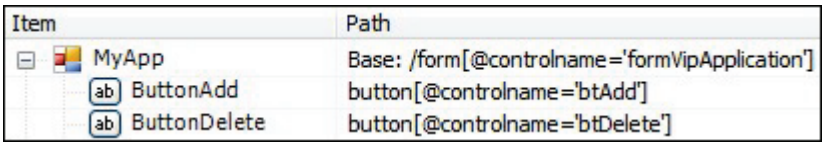
Waiting 5 seconds for status text showing 'Online'

Note: To wait for element states it is necessary to use the attributes within RanoreXPath. For example to wait for a disabled button the RanoreXPath has to include [... @Enabled='False' ...].

Learn more about how to check for existence or to wait for UI element states in code here: Using Repository in Code

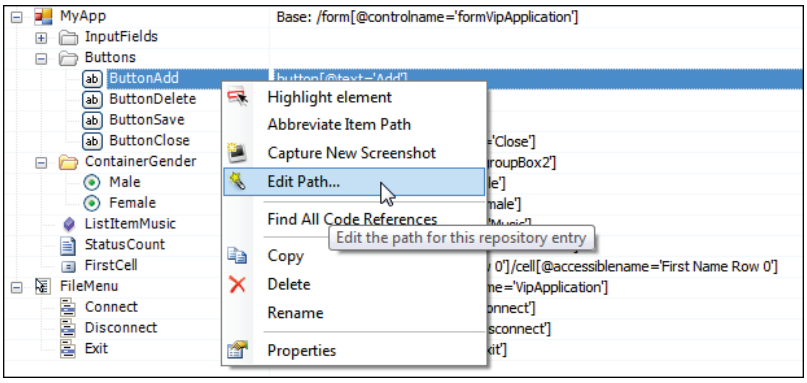
Editing RanoreXPath

Each repository item refers to a RanoreXPath used to identify the corresponding UI element. In order to reduce effort in maintaining path expressions, repository items like buttons, text fields or list items are automatically grouped within application folders holding the base path for each child item.

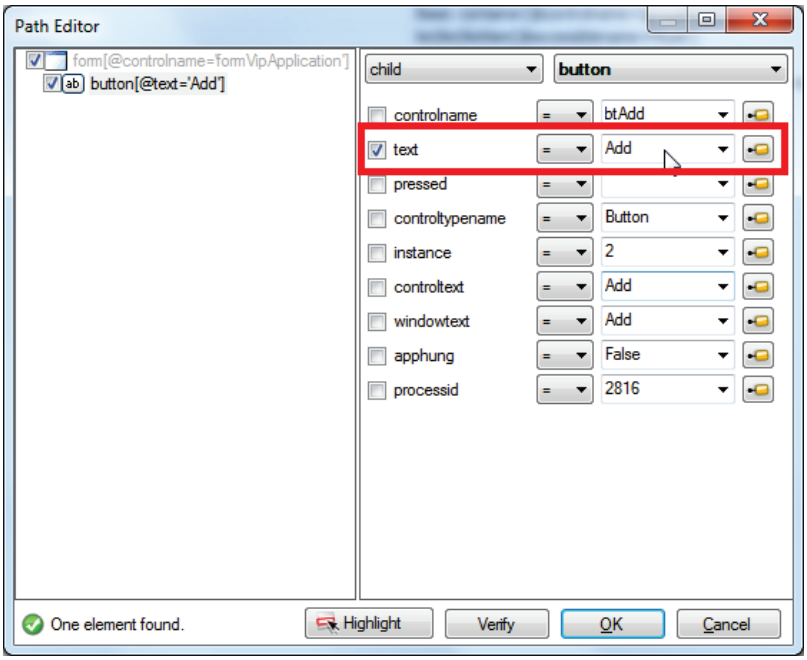


RanoreXPath separation

Press <ENTER> to change the RanoreXPath expression of an item using the edit mode or open the Path Editor using the context menu.



Open Path Editor ...

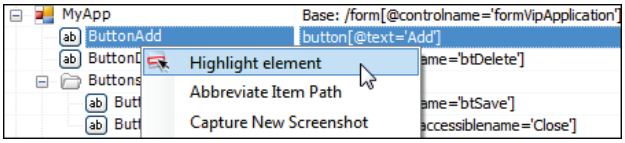


... to change attributes used for identification

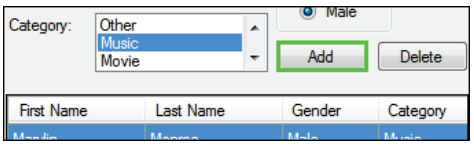
Use the Path Editor to select alternative attributes or to combine multiple attributes for unique UI element identification. Additionally, here you can create variables to be used instead of static string values for a data driven identification.

Learn more about how to work with the Path Editor here: RanoreXPath Editor

After changing the identification attribute from 'ControlName' to 'Text' you can quickly verify the new RanoreXPath using 'Highlight element' as shown below:



Highlight repository item to check the modified path



Button 'Add' as highlighted element

Repository Separation

By default each Ranorex Studio project contains one repository file which is automatically used for each newly created recording. You can manage all UI elements of a test suite project in a single repository file, but there are several reasons for having multiple repositories within a test automation project:

- **Testing Different User Interfaces**
- **Common Repositories for Common Modules**
- **Advanced RanoreXPath Repositories for Complex Identification**
- **Multiple Testers Working on the Same Test Automation Project**

Testing Different User Interfaces

If your test suite contains, for example test cases for a web application and tests for a user interface of a client application, it might be useful to have two repositories. One is used to manage the UI elements of the web application while the other one exclusively stores the elements from the client hosted application. Although you can separate it within one repository using simple folders for grouping, it makes sense to split it especially when working in teams.

Common Repositories for Common Modules

The same way you think about modularization and reusability of small action recordings, the same goes for repositories. For example when you think about a rich client application using main menus, ribbons or toolbars you would create small reusable recordings like to click on the main menu 'File' | 'Open' | 'Handle the Open File Dialog' | etc. All these reusable modules working with a main menu, a main tool bar or similar common available controls, should be based on a repository which exclusively represents commonly used main controls on the user interface.

Advanced RanoreXPath Expressions

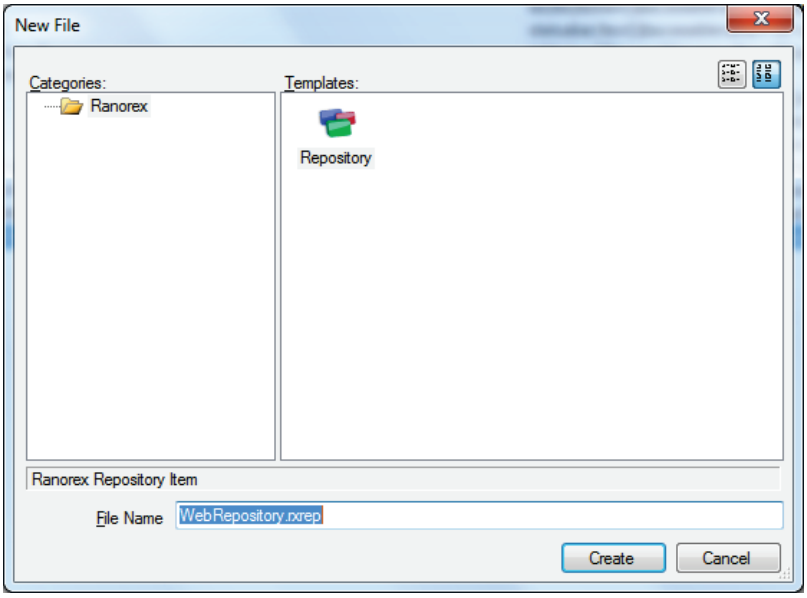
Another reason to build a separate repository could be to store advanced RanoreXPath expressions which should exclusively be used to create new actions manually instead of recording them.

Multiple Testers on the Same Project

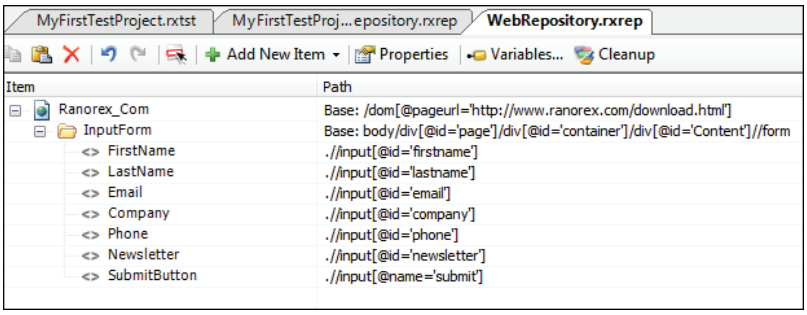
As long as you're working alone on a test automation project it's not a problem to work with one single repository. When working in teams and everyone in the team only clicks the 'Record' button to create test automation modules, it is recommended to keep in mind who is responsible for the main repository. Who is allowed to rename objects and to re-structure the repository? The main reason to separate repositories is to avoid accidental damage of repository items which are used by other team members.

Adding a New Repository

Add a new repository to your project by clicking the 'Add Repository' button in the Ranorex Studio toolbar.



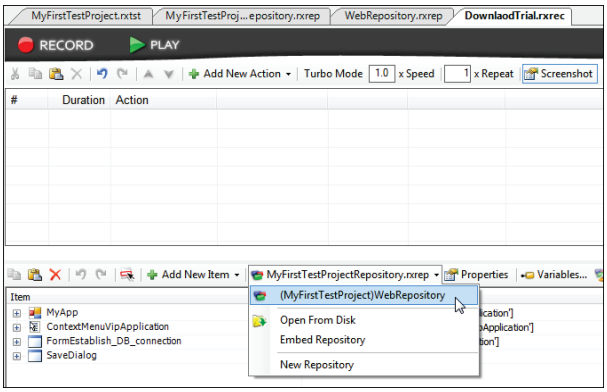
Create a new repository file



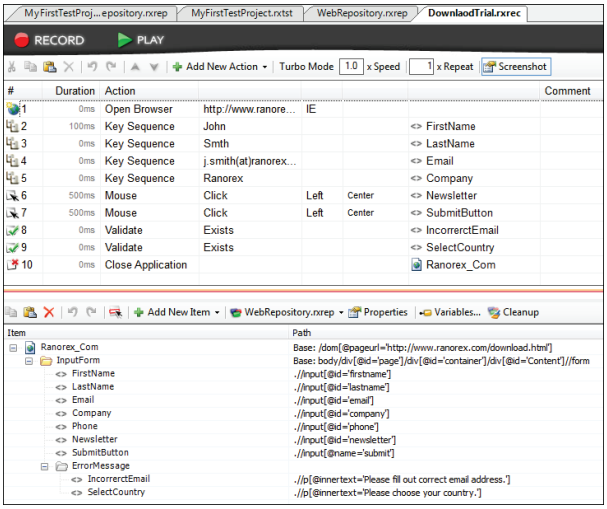
Repository representing user interface of Ranorex download page

Using Repository in Recording

Now create a new recording file and switch from the default repository to the repository which contains the web elements.



Use the web repository instead of the default repository



Manually created actions table used to fill in the download form at Ranorex.com

Using Multiple Repositories in Code Modules

Now your test project contains two repository files. While a recording module can only have one repository assigned, a code module can use multiple repositories:

C#

```
[TestModule("41F66FF2-66D3-42F5-8EC1-F577F8D2D6BC")]
public class CodeModule: ITestModule
{
    // Repository object to access UI elements of VIP Database
    MyFirstTestProjectRepository repo = MyFirstTestProjectRepository.Instance;
    // Repository object to access UI elements of Ranorex web page
    WebRepository webRepo = WebRepository.Instance;

    void ITestModule.Run()
    {
        Mouse.DefaultMoveTime = 300;
        Keyboard.DefaultKeyPressTime = 100;
        Delay.SpeedFactor = 1.0;

        // Set text value in VIP appliction
        repo.MyApp.FirstName.TextValue = "John";
```

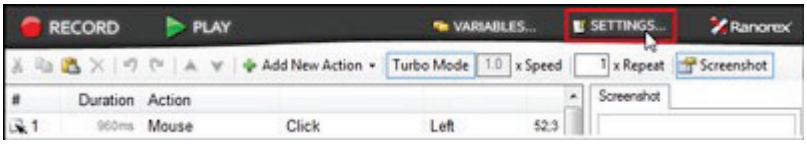
```
        // Set text value in Ranorex.com download form
        webRepo.Ranorex_Com.InputForm.FirstName.Value = "John";
    }
}
```

Note: Even though it's possible to use multiple repositories within a code module you should not ...

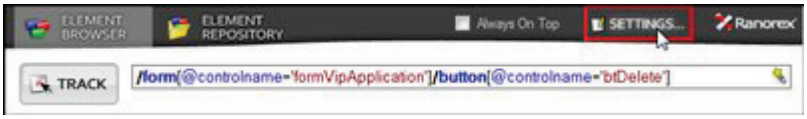
Repository Settings Dialog

There are two ways to open the 'Settings' dialog for the repository:

- **Clicking the 'Settings' button within Ranorex Spy**
- **Clicking the 'Settings' button within Ranorex Recorder**



Open 'Settings' from Recorder

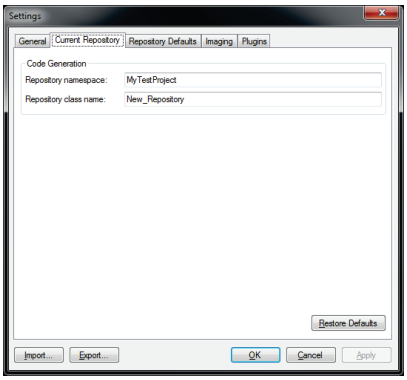


Open 'Settings' from Spy

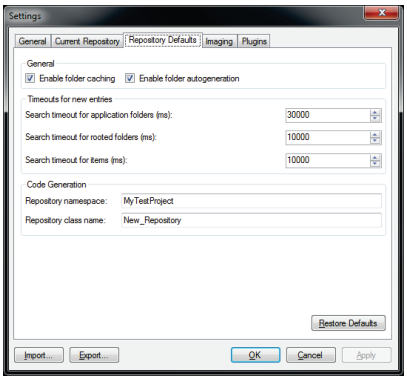
Within the 'Current Repository' tab you can specify the class name and the namespace of the automatically generated source code for the current repository.

On the 'Repository Default' tab page you can specify defaults for the same for newly created repositories. Additionally, the timeouts used for newly created repository folders and items can by simply adapted in the 'Timings for new entries' group box.

The settings 'Enable folder caching' can be unchecked to turn off folder caching for all items by default. Uncheck the checkbox 'Enable folder autogeneration' to prevent the repository from creating rooted folders automatically.



Current settings of a repository



Default repository configuration

Repository Troubleshooting - Folder Caching

In some situations, repository items cannot be found because the caching information of the item's parent folder is incorrect. In such cases, replaying the steps involving these items may only work part of the time or only with long delays.

This is caused by a fallback mechanism which is used to search for an item without using the cache if the first attempt fails. If this occurs, it is recommended to disable the folder cache for the item's parent folder by setting the 'Use Cache' property to 'False'.

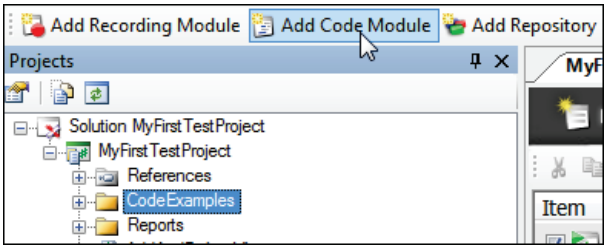
Lesson 7: Code Modules

Though a Ranorex recording with all smart actions, variables and user code capabilities is good enough to create robust test automation modules, it might be useful or preferred to write pure Ranorex automation code. Within the following section you learn how to create a new code module which automates the process of adding and deleting a new person to the VIP Database application.

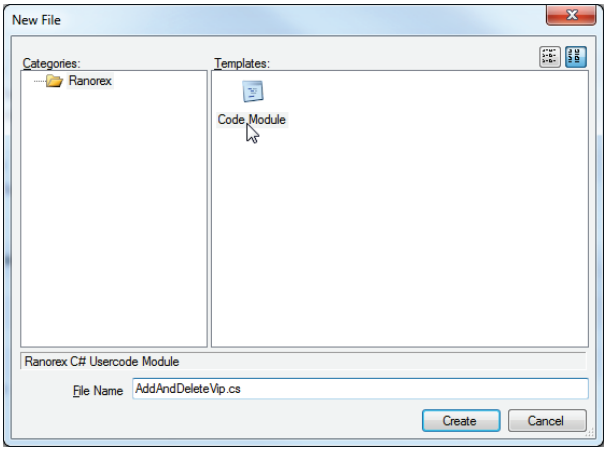
- **Creating Code Module**
- **Using Repository within Code Module**
- **Using Variables with Code Modules**
- **Using Code Modules within Test Cases**

Creating Code Module

Create a new code module by clicking the 'Add Code Module' button at the toolbar.



Add a new code module



Specify file name used for the code module

After clicking the 'Create' button a new file is added to the project and automatically opened in the file view. Ranorex Studio creates a new test module class which contains a 'Run' method ready to be extended with test automation code.

C#

```
namespace MyFirstTestProject
{
    /// <summary>
    /// Description of AddAndDeleteVip.
    /// </summary>
```

```
    [TestModule("A730AB98-9CFE-49D5-BDA7-0CAE6C614866")]
    public class AddAndDeleteVip : ITestModule
    {
        public AddAndDeleteVip()
        {
        }

        void ITestModule.Run()
        {
            Mouse.DefaultMoveTime = 300;
            Keyboard.DefaultKeyPressTime = 100;
            Delay.SpeedFactor = 1.0;
        }
    }
}
```

VB.NET

```
Public Class AddAndDeleteVip
         Implements ITestModule

         Public Sub New()
         End Sub

         Private Sub ITestModule_Run() Implements ITestModule.Run
                  Mouse.DefaultMoveTime = 300
                  Keyboard.DefaultKeyPressTime = 100
                  Delay.SpeedFactor = 1.0
         End Sub
End Class
```

Using Repository within Code Module

The same way you use a repository in the recording to identify UI elements for automation, you can use it in code. Simply add a new private member which represents the repository to your code module class as follows:

C#

```
namespace MyFirstTestProject
{
    /// <summary>
    /// Description of AddAndDeleteVip.
    /// </summary>
    [TestModule("A730AB98-9CFE-49D5-BDA7-0CAE6C614866")]
    public class AddAndDeleteVip : ITestModule
    {
        // Repository object to access UI elements
         MyFirstTestProjectRepository repo = MyFirstTestProjectRepository.Instance;

        public AddAndDeleteVip()
        {}
        void ITestModule.Run()
        {
            Mouse.DefaultMoveTime = 300;
            Keyboard.DefaultKeyPressTime = 100;
            Delay.SpeedFactor = 1.0;

            repo.MyApp.FirstName.TextValue = "John";
            repo.MyApp.LastName.TextValue = "Smith";
```

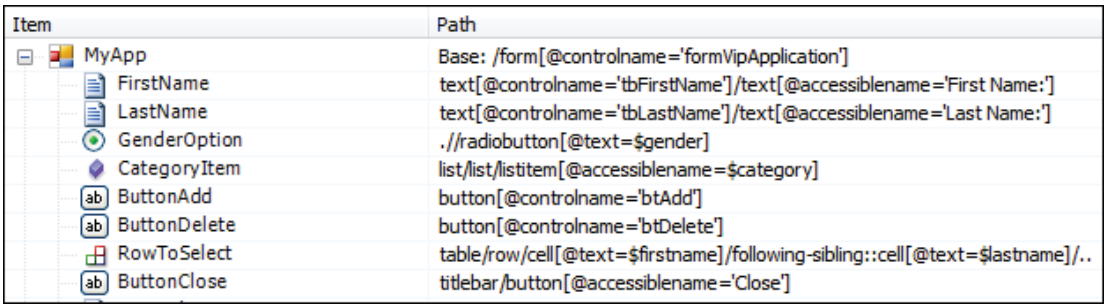
```
            repo.MyApp.ButtonAdd.Click();
        }
    }
}
```

VB.NET

```
''' <summary>
''' Description of UserCodeModule1.
''' </summary>
[TestModule("A730AB98-9CFE-49D5-BDA7-0CAE6C614866")]
Public Class AddAndDeleteVip
Implements ITestModule
    ' Repository object to access UI elements
    Private repo As MyFirstTestProject.MyFirstTestProjectRepository = MyFirstTestPro-
ject.MyFirstTestProjectRepository.Instance

    Public Sub New()
    End Sub
    Private Sub ITestModule_Run() Implements ITestModule.Run
                Mouse.DefaultMoveTime = 300
                Keyboard.DefaultKeyPressTime = 100
                Delay.SpeedFactor = 1.0

                ' Set text values using the Text Value
                repo.MyApp.InputFields.FirstName.TextValue = FirstName
                repo.MyApp.InputFields.LastName.TextValue = LastName
            repo.MyApp.Buttons.ButtonAdd.Click()
        End Sub
End Class
```

Note: By default the class name of a repository is the same as it is used for the repository file name (*.rxrep) shown in the project's view.

Now the class uses a private member referring to repository in order to reuse some of the objects ('First-Name', 'LastName', 'ButtonAdd') within the 'Run' method.



Repository used within code example

Depending on the structure of your repository accessing items in code might get complex like 'repo.MyApp.Buttons.ButtonSave.Click();' To reduce this - especially when 'ButtonSave' is used more than once - you should use local variables instead.

C#

```
namespace MyFirstTestProject
{
    /// <summary>
    /// Description of AddAndDeleteVip.
    /// </summary>
    [TestModule("A730AB98-9CFE-49D5-BDA7-0CAE6C614866")]
    public class AddAndDeleteVip : ITestModule
    {

        private MyFirstTestProjectRepository repo =
                          MyFirstTestProjectRepository.Instance;
        public AddAndDeleteVip()
        {
        }

        void ITestModule.Run()
        {
            Mouse.DefaultMoveTime = 300;
            Keyboard.DefaultKeyPressTime = 100;
            Delay.SpeedFactor = 1.0;

            // Click on 'ButtonSave' using repository directly
            repo.MyApp.Buttons.ButtonSave.Click();

            // Click on 'ButtonSave' and log text value
            // after assignment to local variable 'buttonSave'
            var buttonSave = repo.MyApp.Buttons.ButtonSave;
            buttonSave.Click();
            Report.Info(buttonSave.Text);
        }
    }
}
```

VB.NET

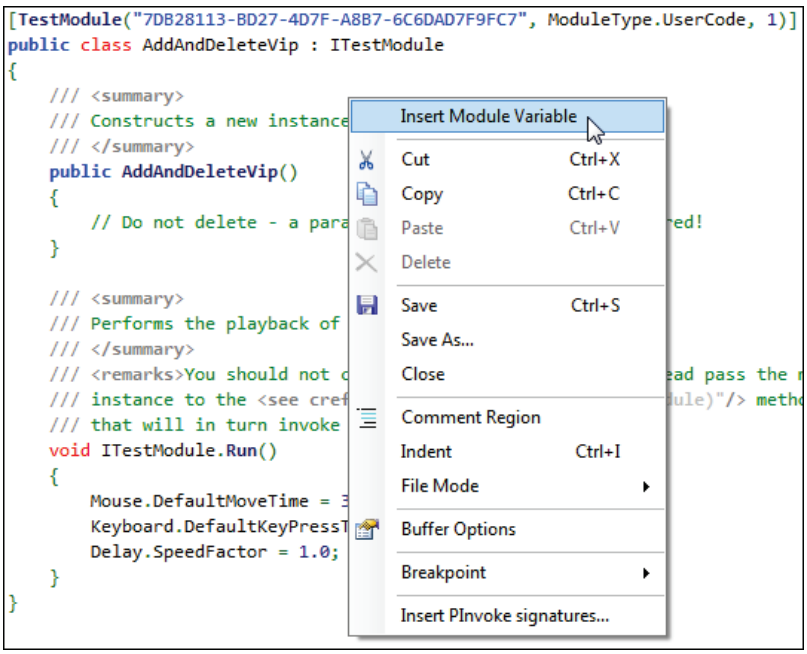
```
Public Class AddAndDeleteVip
         Implements ITestModule

         Private repo As MyFirstTestProject.MyFirstTestProjectRepository = MyFirstTestPro-
ject.MyFirstTestProjectRepository.Instance

         Public Sub New()
         End Sub

         Private Sub ITestModule_Run() Implements ITestModule.Run
                  Mouse.DefaultMoveTime = 300
                  Keyboard.DefaultKeyPressTime = 100
                  Delay.SpeedFactor = 1.0


                  ' Click on 'ButtonSave' using repository directly
                  repo.MyApp.Buttons.ButtonSave.Click()

                  ' Click on 'ButtonSave' and log text value
                  ' after assignment to local variable 'buttonSave'
                  Dim buttonSave As Button = repo.MyApp.Buttons.ButtonSave
                  buttonSave.Click()
                  Report.Info(buttonSave.Text)
         End Sub
End Class
```

To create local variables like shown in the code before simply drag and drop elements from the repository browser directly into the code.

Note: If the repository itself is not already part of the class (e.g. in case of newly created code modules) a local variable for the repository is generated too.

Using Variables with Code Modules

In order to use values provided by a data connector within your code modules you need to add variables to the code. Use the context menu item 'Insert Module Variable'.



Add a new variable to your code module

By adding a new variable Ranorex Studio inserts a new code fragment at the current cursor position. A variable implementation consists of a public property '<VaribleName>' and a private member '_<VariableName>'. By default the name used for the new variable is 'MyTestVariable'.

C#

```
public class AddAndDeleteVip : ITestModule
{
	string _MyTestVariable  = "default_value";

	[TestVariable("FDF6BC6C-AFFC-40F3-8899-5F1BC59B8DC8")]
	public string MyTestVariable
	{
		get
		{
			return _MyTestVariable;
		}
		set
		{
```

```
                        _MyTestVariable = value;
                }
        }

        public AddAndDeleteVip()
        // ...
    }
}
```

VB.NET

```
Public Class AddAndDeleteVip
        Implements ITestModule
        Dim _MyTestVariable As String = "Default Value"
        <TestVariable("8DC1549E-8DC2-434B-BA96-32E2CFDA84D4")> _
         Public Property MyTestVariable As String
         Get
                 Return _MyTestVariable
         End Get
         Set(ByVal value As String)
                 _MyTestVariable = value
         End Set
        End Property

        ' ....

        Sub Run() Implements ITestModule.Run
            Mouse.DefaultMoveTime = 300
            Keyboard.DefaultKeyPressTime = 100
            Delay.SpeedFactor = 1.0
        End Sub
    End Class
```

Now rename the newly added variable to 'FirstName' and create additional variables for the 'LastName', 'Gender' and 'Category'.

Note: Variables used by the repository are not automatically connected with variables defined by code modules. If it is necessary, use the 'set' method as explained below for the variable 'First-Name' to assign values to the repository variable.

C#

```
[TestModule("41F66FF2-66D3-42F5-8EC1-F577F8D2D6BC")]
public class AddAndDeleteVip : ITestModule
{
    // Repository object to access UI elements
    MyFirstTestProjectRepository repo = MyFirstTestProjectRepository.Instance;

    // Local variables with default values
    string _FirstName = "John";
    string _LastName = "Smith;
    string _Gender = "Male";
    string _Category = "Music";

    [TestVariable("A6605D21-BD14-4F3E-9111-E28A9923C66A")]
    public string FirstName
    {
```

```
            get
            {
                    return _FirstName;
            }
            set
            {
                    _FirstName = value;
                    // Setting value for FirstName variable
                    // used by the repository
                    repo.FirstName = FirstName;
            }
    }

    [TestVariable("28BA2FD2-4926-45C5-B07E-4B6DA1ACD708")]
    public string LastName
    {    ..   }
    [TestVariable("6D84257F-12BC-4E2A-B42B-3B119015F646")]
    public string Gender
    {    ..   }
    [TestVariable("279F312F-E2D5-4F9C-A077-1BE5BF830367")]
    public string Category
    {  ..  }

    public AddAndDeleteVip()
    {
    }

    void ITestModule.Run()
    {
        Mouse.DefaultMoveTime = 300;
        Keyboard.DefaultKeyPressTime = 100;
        Delay.SpeedFactor = 1.0;

        // Set text values using the Text Value
        repo.MyApp.FirstName.TextValue = FirstName;
        repo.MyApp.LastName.TextValue = LastName;

        // Select 'Male' or 'Female' depending on
        // the given test data
        // Note: The value is set by the variable's
        // property implementation
        repo.MyApp.GenderOption.Select();

        // Select the category list item depending
        // on the the given test data
        repo.MyApp.CategoryItem.Select();

        // Press button 'Add'
        repo.MyApp.ButtonAdd.Press();

        // Report a screenshot of the application
        Report.Screenshot(repo.MyApp.Self);

        // Click the correct row within the
        // datagrid.
        repo.MyApp.RowToSelect.Click();
        // Press button 'Delete'
        repo.MyApp.ButtonDelete.Press();
    }
}
```

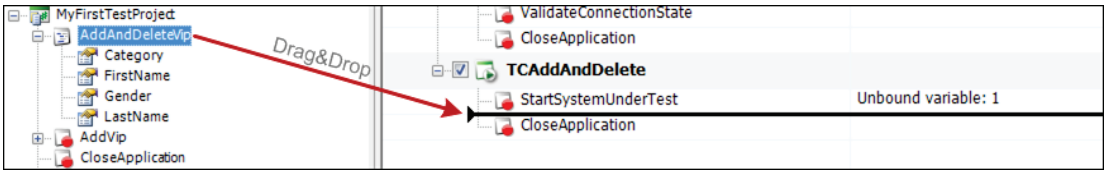
VB.NET

```
<TestModule("56849E42-5A21-4EDB-BD57-94F2594EEF79", ModuleType.UserCode, 1)> _
Public Class AddAndDeleteVip
        Implements ITestModule
        ' Repository object to access UI elements
        Private repo As MyFirstTestProjectRepository = MyFirstTestProjectReposito-
ry.Instance

        ' Local variables with default values
        Dim _FirstName As String = "John"
        Dim _LastName As String = "Smith"
        Dim _Gender As String = "Male"
        Dim _Category As String = "Music"

        <TestVariable("A6605D21-BD14-4F3E-9111-E28A9923C66A")> _
        Public Property FirstName() As String
                Get
                        Return _FirstName
                End Get
                Set
                        _FirstName = value
                        ' Setting value for FirstName variable
                        ' used by the repository
                        repo.FirstName = FirstName
                End Set
        End Property

        <TestVariable("28BA2FD2-4926-45C5-B07E-4B6DA1ACD708")> _
        Public Property LastName() As String
                Get
                        Return m_LastName
                End Get
                Set
                        m_LastName = Value
                End Set
        End Property
        Private m_LastName As String
        <TestVariable("6D84257F-12BC-4E2A-B42B-3B119015F646")> _
        Public Property Gender() As String
                Get
                        Return m_Gender
                End Get
                Set
                        m_Gender = Value
                End Set
        End Property
        Private m_Gender As String
        <TestVariable("279F312F-E2D5-4F9C-A077-1BE5BF830367")> _
        Public Property Category() As String
                Get
                        Return m_Category
                End Get
                Set
                        m_Category = Value
                End Set
        End Property
        Private m_Category As String

        Public Sub New()
        End Sub

        Private Sub ITestModule_Run() Implements ITestModule.Run
                Mouse.DefaultMoveTime = 300
                Keyboard.DefaultKeyPressTime = 100
                Delay.SpeedFactor = 1.0
```

```
                ' Set text values using the Text Value
                repo.MyApp.FirstName.TextValue = FirstName
                repo.MyApp.LastName.TextValue = LastName

                ' Select 'Male' or 'Female' depending on
                ' the given test data
                ' Note: The value is set by the variable's
                ' property implementation
                repo.MyApp.GenderOption.[Select]()

                ' Select the category list item depending
                ' on th the given test data
                repo.MyApp.CategoryItem.[Select]()

                ' Press button 'Add'
                repo.MyApp.ButtonAdd.Press()

                ' Report a screenshot of the application
                Report.Screenshot(repo.MyApp.Self)

                ' Click the correct row within the
                ' datagrid.
                repo.MyApp.RowToSelect.Click()
                ' Press button 'Delete'
                repo.MyApp.ButtonDelete.Press()
        End Sub
End Class
```

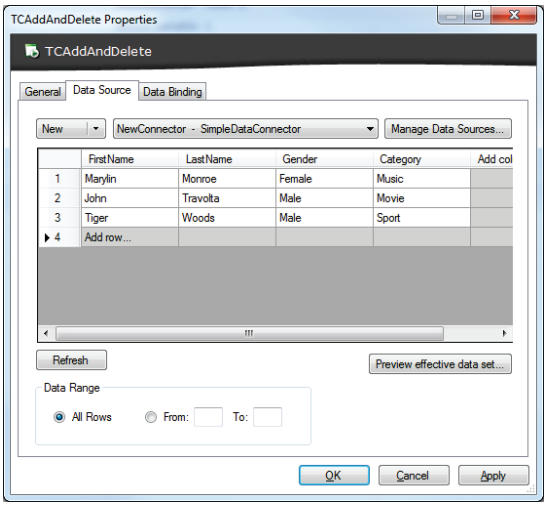
Using Code Modules within Test Cases

The code module implemented above is now ready to be started by a test case. Add a new test case ('TCAddAndDeleteVip') to your test suite and reuse two already existing modules to start the VIP Application at the beginning and to close it at the end of the test case. You can use the drag&drop feature to quickly insert the newly created code module into the test case.

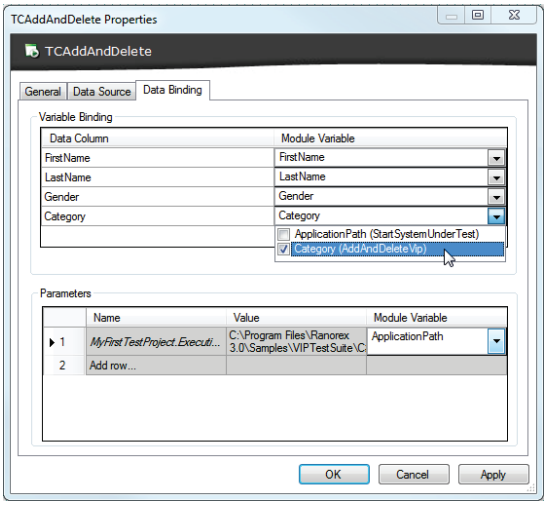


Drag&Drop code module into a test case

You can reuse the existing data connector created during Lesson 3 as the same data provider for your new test case.



Reuse the already existing data connector



Bind variables to the data connector's columns

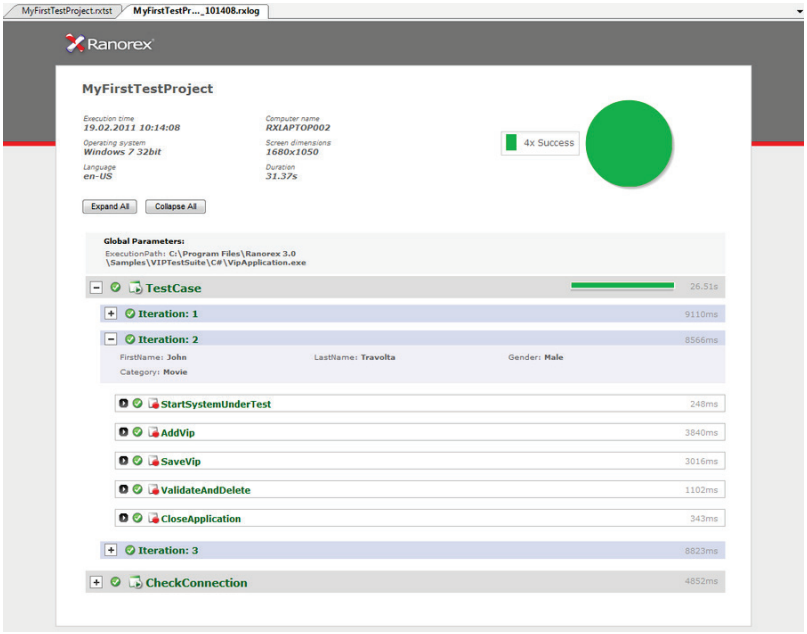
Lesson 8: Reporting

By default each run of a test suite, a test case or a single recording generates a report file (*.rxlog) which informs you about whether the run was successful or not. Within the following lesson you will learn about the following:

- **Reading Ranorex Reports**
- **Logging Individual Information**

Reading Ranorex Reports

After executing a test suite in Ranorex Studio the generated report file is shown in the file view as follows.



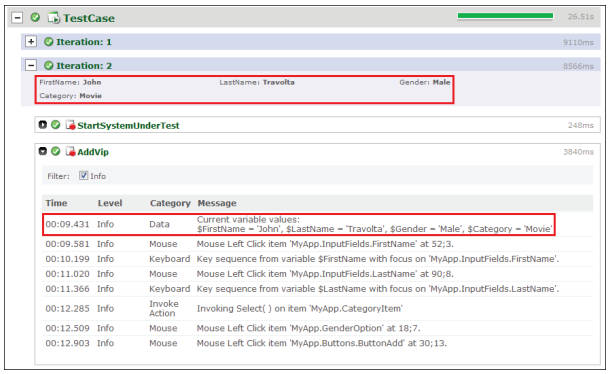
Ranorex report after running a test suite

The report shown after running a test suite provides a general overview about how many of the test cases have been executed successfully, failed or blocked. Each executed test case and all its child modules can be analyzed more in depth by expanding a particular test case.

In addition a test suite report also informs you about the following:

- **System related information like execution time, machine name, operating system, screen dimensions, language, duration**
- **Global parameter values**

If a test case uses local parameter values or external test data, the report shows the values used during automation as follows:



Detailed view of a test case report

The low-level log messages generated by a recording or by a code module shown in the figure above consist of a time stamp, a report level, a category and message text. By default logging is turned on for recordings. Use the Recorder's settings dialog to change the default value and to turn off logging for new recordings. In order to turn logging off or on for a particular action item, open the properties and set the 'Use Default Logging' attribute to false.

Jump to Item and Analyze with Spy

Expand the report as shown above and move the mouse pointer over the first log message.

Debug a test run with quick links 'Jump to Item' and 'Open in Spy'

Click at 'Jump to Item' to open the module and to focus on the source action item. Use the quick link 'Open in Spy' to analyze the RanoreXPath expression used for the particular action item. This is especially useful in situations when a test run fails with the message 'Failed to find item...' error message.

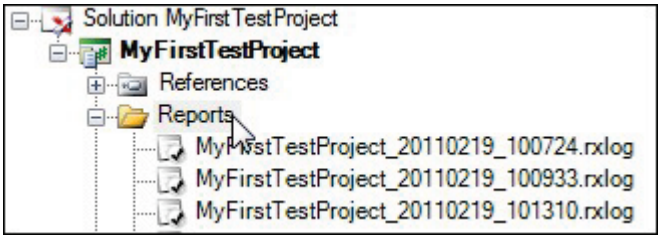
Filter Log Messages

Use the checkboxes shown at the top of each module to filter log messages.

Filter for different log levels

Report Folder

For each run of a test suite, a single test case or a recording a new log file (*.rxlog) is generated and saved in the project's 'Report' folder. You can open older report files by double-clicking the file in the project view. Specify the file names used for the report files within the test suite's settings dialog.



Log file history within Ranorex Studio project

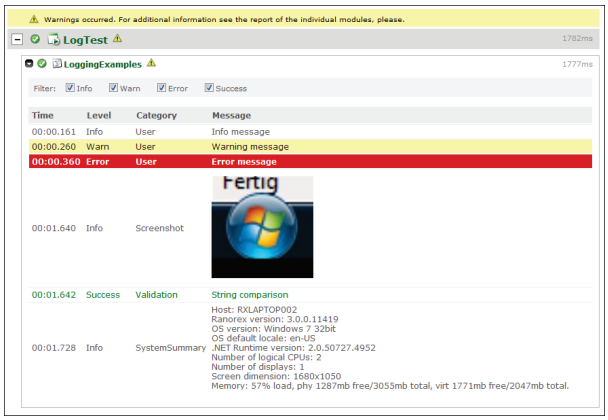
Note: It is not required to run Ranorex Studio to open a report file. You can also open the report file from the Windows Explorer. If you're going to copy or send the file by email always ensure that the stylesheet file 'RanorexReport3.xsl' is located in the same target directory.

Logging Individual Information

Use code actions or code modules to log individual messages to the report file as shown below:

C#

```
Report.Debug("Debug message");
Report.Info("Info message");
Report.Warn("Warning message");
Report.Error("Error message");

// Get windows start button ...
Ranorex.Button windowsStartButton= Host.Local.FindSingle("button[@text='Start']");
// ... and create a screenshot for the report
Report.Screenshot(windowsStartButton);

// Simple 'AreEqual' validation
Ranorex.Validate.AreEqual("compareText", "compareText","String comparison",true);
// Log current system environment
Report.SystemSummary();
```

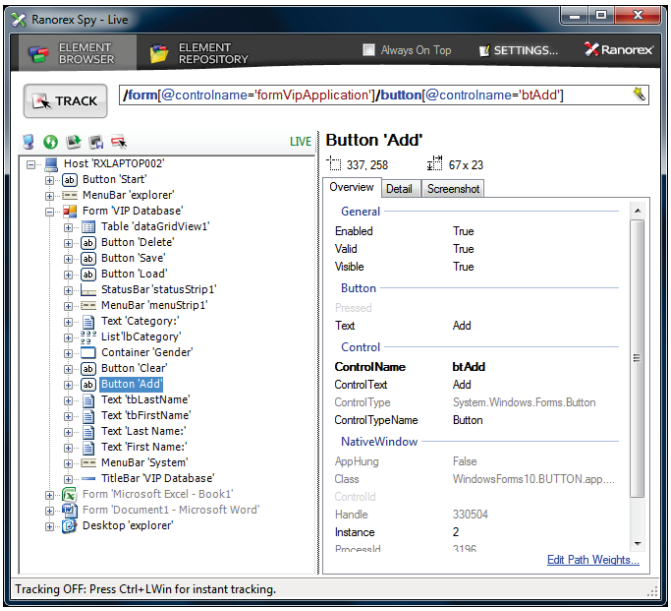


Report file with individual log messages

Note: A test case's failure depends on whether a Ranorex exception has been thrown or not. This could automatically happen if a validation step fails or a new Ranorex exception is thrown.

Lesson 9: Ranorex Spy

As a standalone tool the Ranorex Spy provides all the functionality needed to explore and analyze applications or websites under test - including their controls and UI elements. After starting Ranorex Spy the element browser contains all currently running applications from your Windows desktop.



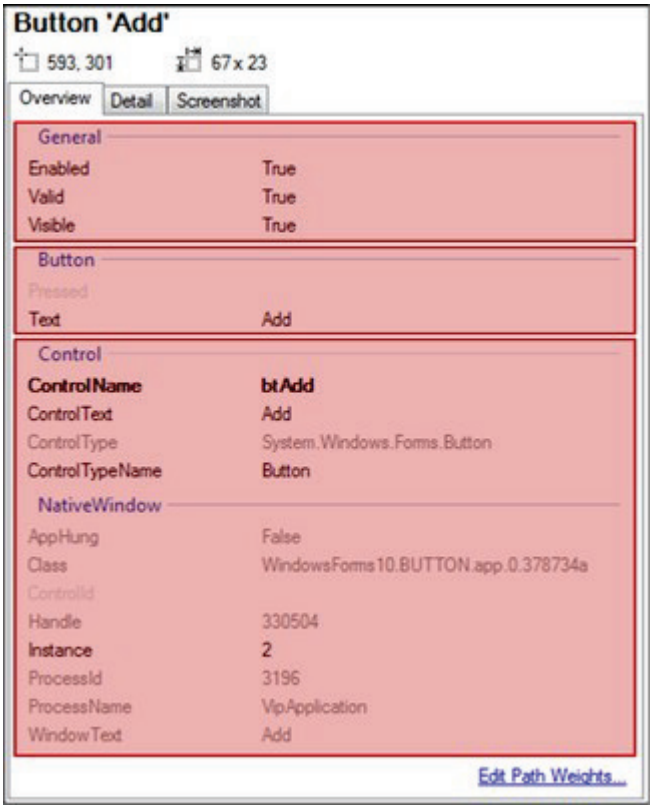
Ranorex Spy - applications and their UI elements

- **Tracking UI Elements**
- **RanoreXPath Edit Mode**
- **Creating Ranorex Snapshot Files**
- **General Ranorex Settings**

The element browser tree shown on the left side represents the computer's desktop with all currently opened applications. The name of the root node is set to the machine's host name. By selecting one of the elements from the browser tree, the Spy provides more detailed information about the selected item in the tabs ‘Overview’, ‘Detail’ and ‘Screenshot’ shown on the right. Ranorex recognizes over 30 of the most commonly known types of UI elements. In Ranorex vocabulary they are called 'adapters'. Every element shown within the element tree of Ranorex Spy is represented by a Ranorex UI Adapter. If Ranorex is not able to specify the type of adapter of a UI element it will be displayed as Unknown.

The 'Overview' tab is the most important view into the details of a UI element. The available attributes and their values are divided into the following sections:

- **'General'**
- **Logical adapters like 'Form', 'Button' or 'Text'**
- **Technology related adapters like 'Control', 'Accessible' or 'NativeWindow'**



Ranorex Spy 'Overview' tab divided into three attribute sections

General

Regardless of what type of UI element is currently selected the 'General' section provides information about whether the element is enabled, valid or visible.

Logical adapters like 'Form', 'Button' or 'Text'

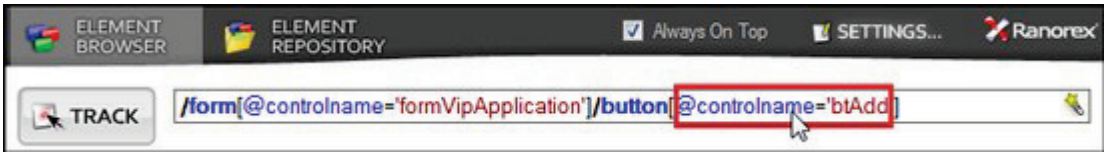
Regardless of what technology is used from the application under test, Ranorex tries to abstract it with logical adapters. These adapters are also used within repositories and provide class specific attribute values like the text value or the state of a button.

Technology related adapters like 'Control', 'Accessible' or 'NativeWindow'

This type of adapter is used to group technology related information like the 'ControlName' of a .NET WinForms button or the process name of an application.

All attributes shown within this tab are accessible during an automated test for validation. Depending on the type of a UI element, they may also be used to set values, too. Learn more about how to use different adapters to read and set attribute values here.

In addition all attributes can be used within the RanoreXPath. By default the attribute written in bold is automatically used to identify the UI element.



Attribute 'ControlName' is used to identify the button

The RanoreXPath expression shown in the figure above can be seen as a navigation path through the element browser tree used to identify a UI element. You can use the RanoreXPath Editor to change element identification or edit the path expression directly within the text box's RanoreXPath Edit Mode.

Toolbar short cuts

	Browse Local Host	Shows currently running host applications
	Refresh	Updates the current state of applications
	Load from Snapshot	Loads an existing Ranorex snapshot file
	Save as Snapshot	Saves all UI elements from the tree view to a Ranorex snapshot file
	Highlight Elements	When switched on, highlights selected UI elements on the user's desktop

Context menu

	Add to Repository	Adds the current element to a Ranorex Repository
	Set Element as Root	Sets the currently selected element as root element
	Refresh	Updates all attributes of the specified UI element
	Save as Snapshot	Saves the current UI element including the child structure to a Ranorex snapshot file
	Add Class or Process name to GDI capture list	Specifies whether the element's class or process name should use the GDI Plug-In to turn on text based object recognition
	Highlight Element	Highlights the current element
	Capture Screenshot...	Captures a screenshot from the current element to be used for further image based validation

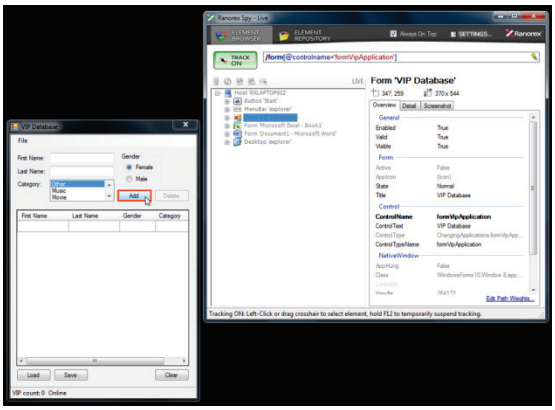
Tracking UI Elements

You can navigate manually through the Ranorex Spy's element tree or use element tracking to identify UI elements in your application under test. The Ranorex Spy supports two ways of tracking UI elements:

- **Using the TRACK Button**
- **How to Track Menu Items - 'Instant Tracking'**

Using the TRACK Button

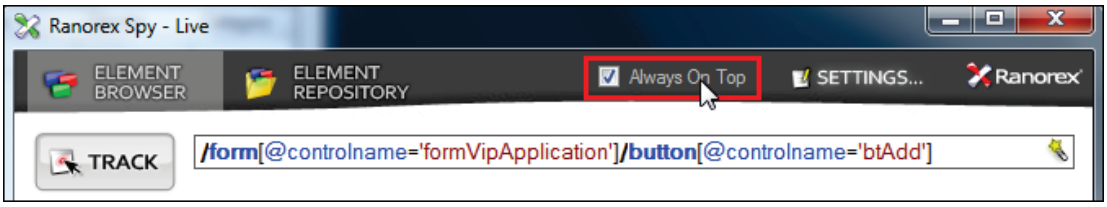
Click the 'TRACK' button to start tracking a UI element. Move your mouse pointer over a specific control (e.g. button, text box) so that the currently identified object is highlighted. By clicking the control, the tracking mode will be stopped and all information of the selected UI element will be shown within Ranorex Spy.



Click 'Track' button to start element tracking.

Note: To abort the tracking session press <ESC> or click the 'TRACK' button again.

In some cases it might be necessary to track UI elements which are currently behind the Ranorex Spy window. Simply uncheck the 'Always On Top' checkbox to cause Ranorex Spy to minimize during the tracking process.

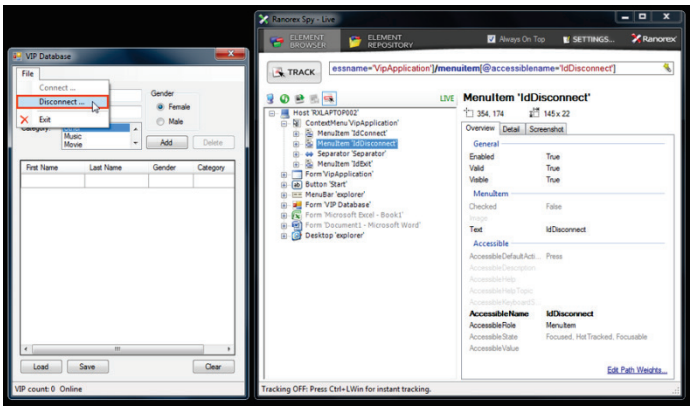


'Always On Top' checkbox within Ranorex Spy

How to Track Menu Items - 'Instant Tracking'

When tracking menu or popup items you use the 'Instant Tracking' method by using a short cut to select a UI element. Simply move your mouse pointer over a menu item and press the <CTRL> + <LWin>* key combination to instantly track the UI element. 'Instant Tracking' is not only available for popup items; this tracking mode can also be used for all other UI elements.

*The <LWin> key is typically between the <CTRL> and <ALT> keys.

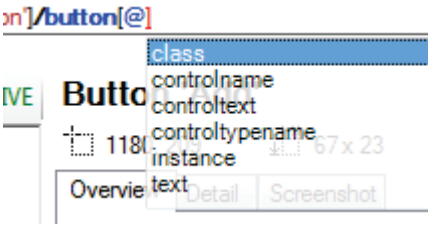


Instant tracking of menu items

RanoreXPath Edit Mode

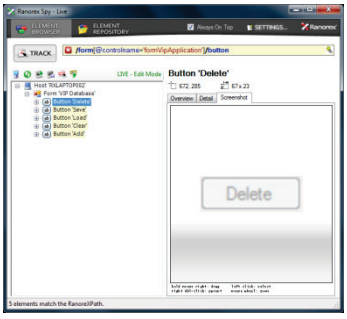
There are two ways to change the RanoreXPath expression. One way is to work with the Path Editor which you can open with the magic wand in the textbox. The second option is to edit the path directly within the textbox of Ranorex Spy which is described in the following section.

To support the editing of RanoreXPath expressions, the editor textbox provides an auto completion feature which helps to avoid syntax errors and to speed typing. By pressing the key combination <CTRL>+<SPACE> at any time, the RanoreXPath editor suggests a list of suitable keywords, attributes or syntax codes.



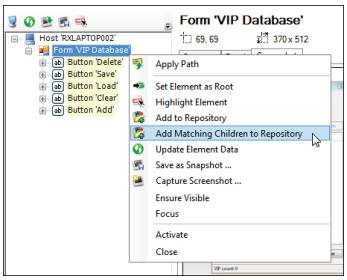
RanoreXPath auto completion

The RanoreXPath editor mode is indicated with a yellow highlighted text box. In edit mode, RanoreXPath is used as a query to filter UI elements as follows:



Use of RanoreXPath as a filter to search for buttons

The result is presented directly within the Ranorex Spy tree. All matching items are highlighted in yellow. In addition the result can easily be added to a repository by selecting the menu item 'Add Matching Children to Repository'.



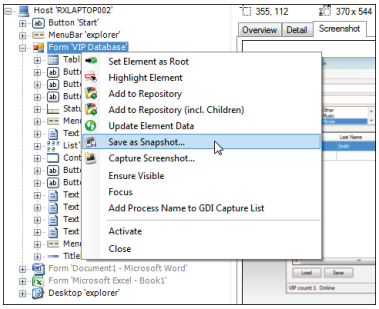
Add matching items to the repository

To stop the edit mode click the red cancel button which is located in the RanoreXPath edit box.

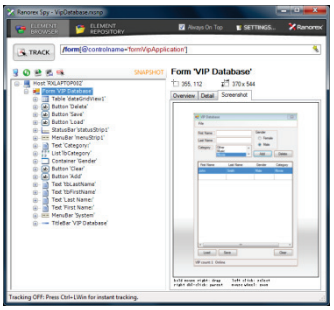
Stop RanoreXPath edit mode

Creating Ranorex Snapshot Files

With the Ranorex Snapshot feature, you can not only save graphic but also UI structure information into an XML file (*.rxsnp). Create and send a snapshot by email to support@ranorex.com to request application specific assistance from the Ranorex team. Based on the information stored within the snapshot, the support team is able to analyze your application under test offline and to give advice on how to optimize unique object identification with RanoreXPath.



Saving UI structure of VIP Database as Snapshot file



Ranorex Snapshot file for VIP Database application

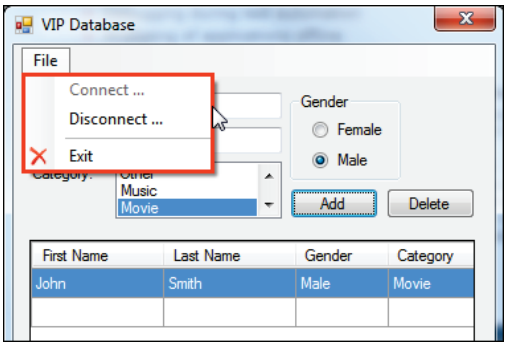
Use the context menu item 'Save as Snapshot' within Ranorex Spy to store all elements, dialogs or applications including their sub items into a single Ranorex snapshot file.

Note: In order to save screenshot information correctly, activate the desired application for which you wish to create a snapshot before saving.

Snapshots from popup windows, dropdown combo boxes or dropdown menus

For example you can create a snapshot from the 'File' menu popup of the VIP Database application using the shortcut key <SCROLL> in combination with the instant tracking feature of Ranorex Spy:

- **Open Ranorex Spy and activate 'Highlight Elements'.**
- **Start VIP Database application and open the 'File' menu.**
- **Track the parent menu container window of the menu items. Track the popup frame by mousing over the edge of the window and pressing <CTRL>+<LWin>.**
- **Press the <SCROLL> shortcut to cache the current element including their underlying items for Ranorex Spy.**
- **Finally save the current view as Snapshot.**

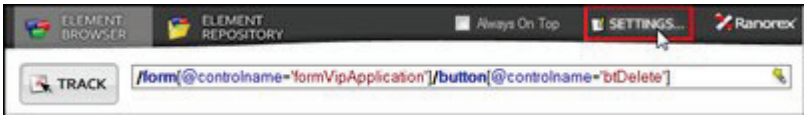


Track menu container window using instant tracking (<CTRL>+<LWin>)

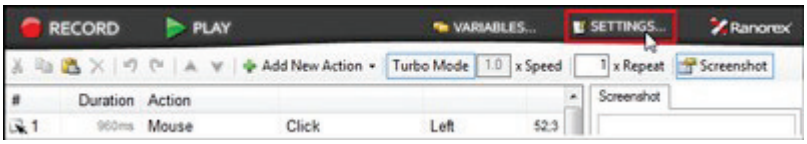
General Ranorex Settings

There are three ways to open the 'Settings' dialog for adapting general Ranorex automation and identification settings:

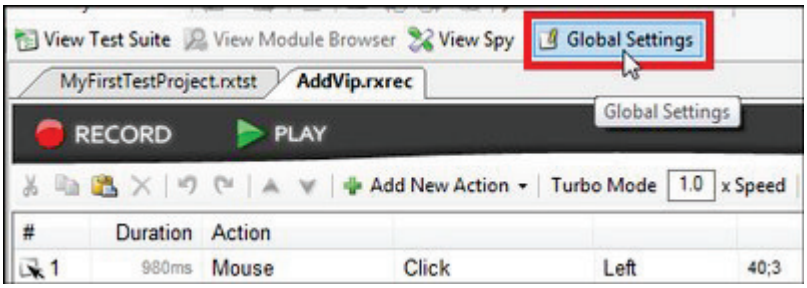
- **Clicking 'Settings' button within Ranorex Spy**
- **Clicking 'Settings' button within Ranorex Recorder**
- **Clicking 'Global Settings' button within Ranorex Studio**



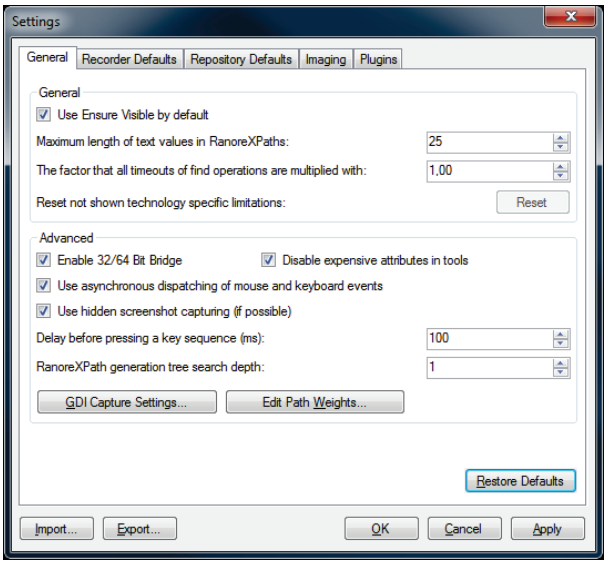
Open 'Settings' from Recorder



Open 'Settings' from Spy



General settings



General Ranorex settings

'Use Ensure Visible by default'

Specifies whether a UI element used as Ranorex Adapter should be forced to become visible before automation or not. By default this value is automatically used for each repository item.

'Maximum length for text values in RanoreXPaths'

Specifies the maximum length of attribute values used within RanoreXPath.

'The factor that all timeouts of find operations are multiplied with'

Multiplies all repository timeouts with the specified value to prevent test cases from failing in case of different timing behavior. This is especially useful when executing tests on different system configurations.

Note: The value can also be set at the time of executing a test suite using the test suite's command line arguments or by directly using the **'Ranorex.Core.Configuration.Current.Adapter.TimeoutFactor'** property in code.

'Reset not shown technology specific limitations'

Resets 'Do not show again' checkbox used to suppress assistance dialog for technology limitations. Click 'Reset' to show the dialog again for not instrumented technologies or applications.

Advanced Settings

Most of the settings shown in the 'Advanced' group box are used to configure Ranorex object recognition and RanoreXPath generation. Please be careful when changing these settings.

'Enable 32/64 Bit Bridge'

Use the checkbox 'Enable 32/64 Bit Bridge' to turn off the bit bridge required to handle 32/64 bit based applications on 64 bit operating systems automatically.

'Disable expensive attribute in tools'

This setting instructs plug-ins not to evaluate computationally intensive attributes for Ranorex Spy, Recorder, and Ranorex Studio. If checked, attributes like Row.Index do not have a value for certain technologies when shown in Ranorex Spy, that way speeding up performance.

'Use asynchronous dispatching of mouse and keyboard events'

This setting is used to turn on or off dispatching of mouse and keyboard events during recording.

'Use hidden screenshot capturing (if possible)'

Tries to create screenshots used for recordings, repositories or Ranorex snapshot files also for hidden applications.

'Delay before pressing a key sequence (ms)'

Specifies the time to wait in milliseconds before performing a key sequence simulation.

'RanoreXPath generation tree search depth'

Defines the number of unique RanoreXPath levels required for a unique identification.

Note: The higher the value, the shorter the RanoreXPath strings are generated. But keep in mind that the shorter the path, the longer it takes to find the expected UI element.

'GDI Capture Settings...'

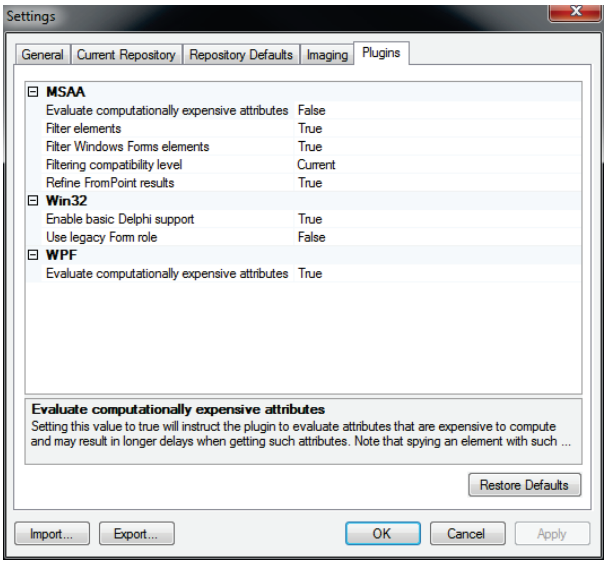
Opens the dialog to change the current GDI capture list. More information about the Ranorex GDI Plug-In can be found here.

'Edit Path Weights...'

In order to get assistance with editing RanoreXPath Weights, contact the Ranorex support team.

Click the button **'Restore Defaults'** to reset all values to their default values. Use the **'Import...'** and **'Export...'** buttons to save and load user specific configurations.

Plug-in specific settings



Plug-In specific settings

Plug-in specific settings can be used to alter the behavior of individual Ranorex plug-ins, for example, to achieve backwards-compatibility with older versions. For more information on each setting, read the description shown when you click on a particular setting in the property grid.

Lesson 10: RanoreXPath Editor

In this section you will learn about

- **Layout of Advanced RanoreXPath Editor**
- **The Tree View Section**
- **The Attribute Equation Section**
- **Relationship and Attribute Example**
- **Defining Variables**
- **Live and Offline View**

The advantages of Advanced RanoreXPath Editor

As you might know, the RanoreXPath expression is a powerful identifier of UI elements for desktop and Web applications. To learn more about the RanoreXPath click here: RanoreXPath Syntax

The Advanced RanoreXPath Editor assists you in setting up the RanoreXPath of your components in an easy and well-arranged way. So if you track some UI component with the Ranorex Spy and you want to make the RanoreXPath of this component more readable or you'd like to simply modify the RanoreXPath you tracked, before the Advanced RanoreXPath Editor is the tool to use.

How to access the Advanced RanoreXPath Editor

You can simply follow this: *Everywhere a RanoreXpath is visible in a Ranorex Tool you can edit this RanoreXPath with the Advanced RanoreXPath Editor.*

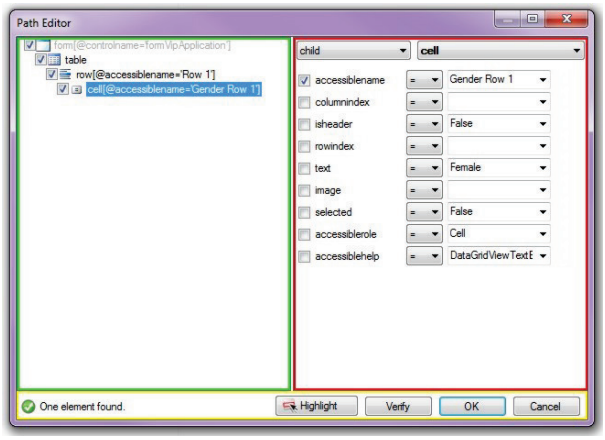
More precisely, this will be applicable in Ranorex Repositories and in the Ranorex Spy. You can access the Advanced RanoreXPath Editor by choosing any UI Element in Repository or in Spy and switch into edit mode of the corresponding RanoreXPath by simply right-clicking on the path.

Click magic wand to open RanoreXPath Editor

Layout of Advanced RanoreXPath Editor

So as you know how to access the Advanced RanoreXPath Editor, you will get an overview of the structure of this editor.

As you can see you can break down the editor in three areas, the RanoreXPath shown in a tree view on the left (marked in green), the attribute equation list of the selected tree item on the right (marked in red) and a status and button section at the bottom (marked yellow).



RanoreXPath Editor

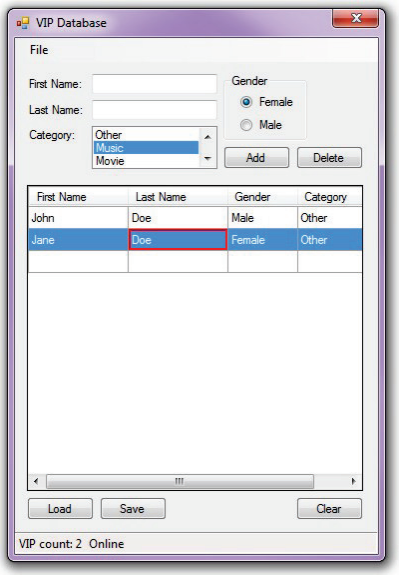
- **The Tree View Section**
- **The Attribute Equation Section**
- **Relationship and Attribute Example**
- **Defining Variables**
- **Live and Offline View**

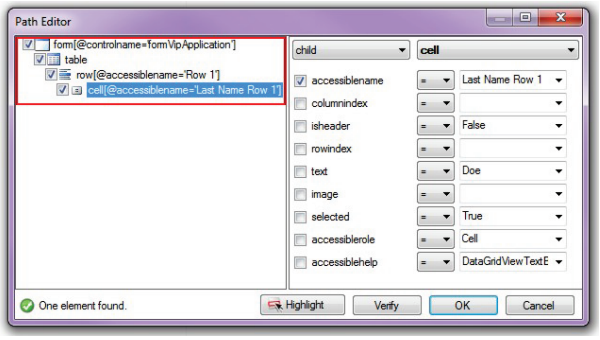
As you can see the whole path is shown in this tree structure with each path element as one node of the tree. In the left section of the editor you can access each of these path elements to alter the attributes equations in the right section on which the path is based. In the bottom section you can on the one hand verify and accordingly highlight the adapters which correspond to the path and on the other hand apply or reject the changes on the path you made. Furthermore you can inspect how many adapters correspond to the currently selected tree item shown at the bottom left of the editor.

To understand the behavior of the tree view section and the attribute equation section you should have a closer look on the sections noted above.

The Tree View Section

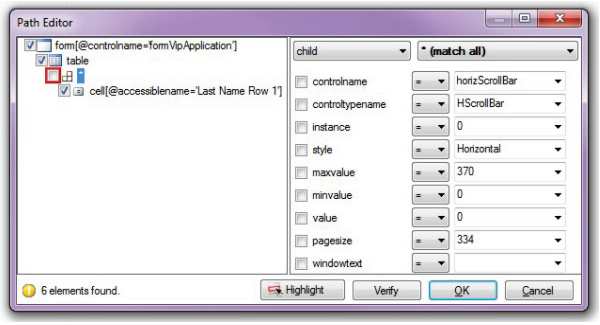
As described above you can see the whole RanoreXPath you want to edit mapped as a tree structure. If you for example start the VIP Database example application and add two entries, one called John Doe and the other called Jane Doe and you track the cell holding the last name of Jane Doe you will get following tree structure in the Advanced RanoreXPath Editor:





RanoreXPath structure in Path Editor for cell in second row

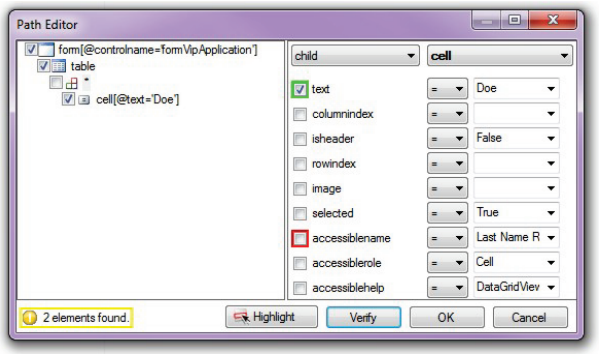
As you can see there are several check boxes before each of the tree items in your tree structure. They are used to enable and disable the attribute equations specified on the attribute equation section on the right. So by unchecking one of these check boxes the level of your path will no longer be requested and it will match all adapters:



Unchecking levels in tree view

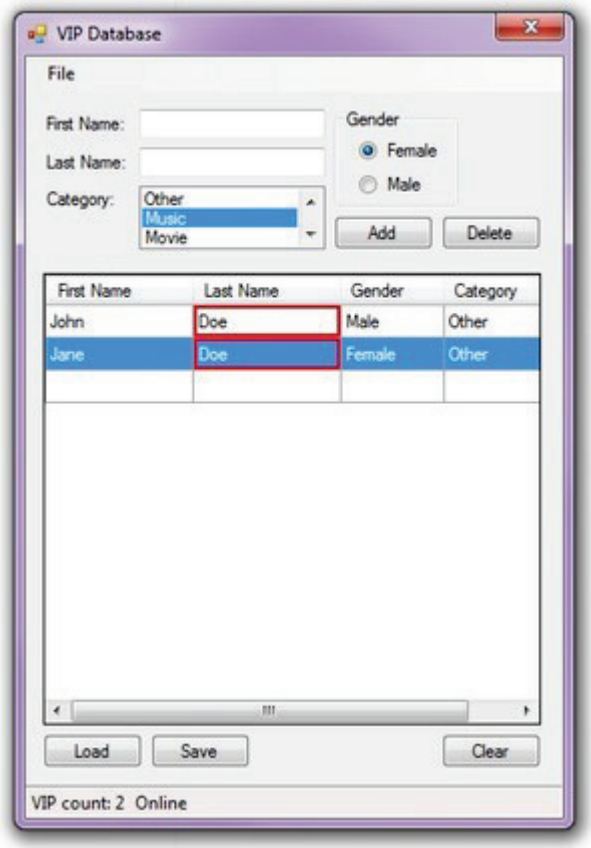
The Attribute Equation Section

In the attribute equation section of the Advanced RanoreXPath Editor you can define the attribute value comparison which the RanoreXPath is based on. So if you want to define your adapter not by the accessible name attribute as selected by default but by the text value, just uncheck the accessible name check box and check the text check box. If you now verify your path with the Verify button in the bottom section of the editor you will see that the path now corresponds to two elements, the last name cells of both Jane and John Doe, because you first have made the level in your tree structure which held the equation row[@accessiblename='Row 1'] match all adapters and second you don't compare the accessible name of the cell but the text value.

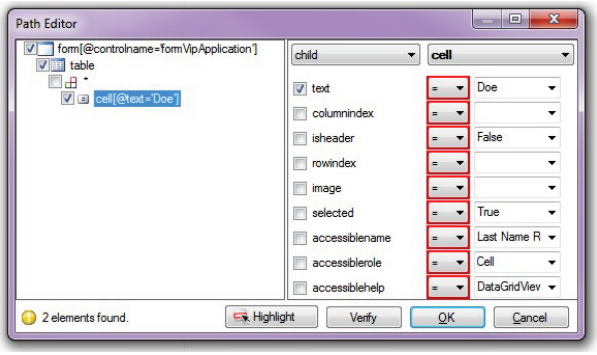


Two elements found in grid

If you now highlight the UI elements corresponding to your altered RanoreXPath with the Highlight button in the bottom section of the editor you will get the result verified before visualized on your application under test:



As you can see in the attribute equation section there is a combo box for every available attribute to select the type of equation:



The following types of equations are defined:

- **= The attributes value must be equal to the given value**
- **~ The attributes value must match the regular expression**
- **!= The attributes value must not be equal to the given value**
- **> The attributes value must start with the given string (case sensitive)**

Wright now you have just used the equal operator. You can try out the other operators by yourself:

This equation will ...

First Name	Last Name	Gender	Category
John	Doe	Male	Other
Jane	Doe	Female	Other

... return these UI elements.

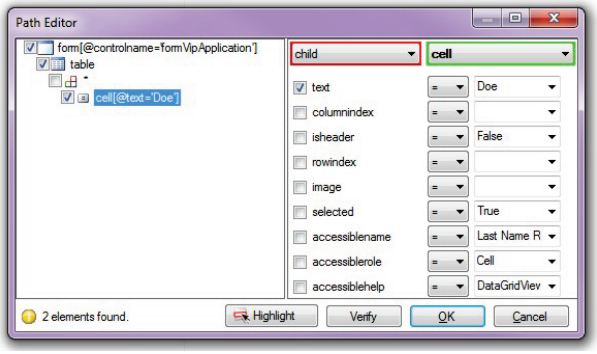
For more details about regular expression see the RanoreXPath documentation.

'Not Equal' will deliver all cells which don't contain the value 'Doe'.

This equation will deliver you both cells holding text values starting with "J", which are the cells holding the text value John and the text value Jane.

Relationship and Attribute Example

After exploring the equation operators you are going to learn more about the combo box holding the relationship operators (marked red) and the combo box holding the attribute types (marked green).



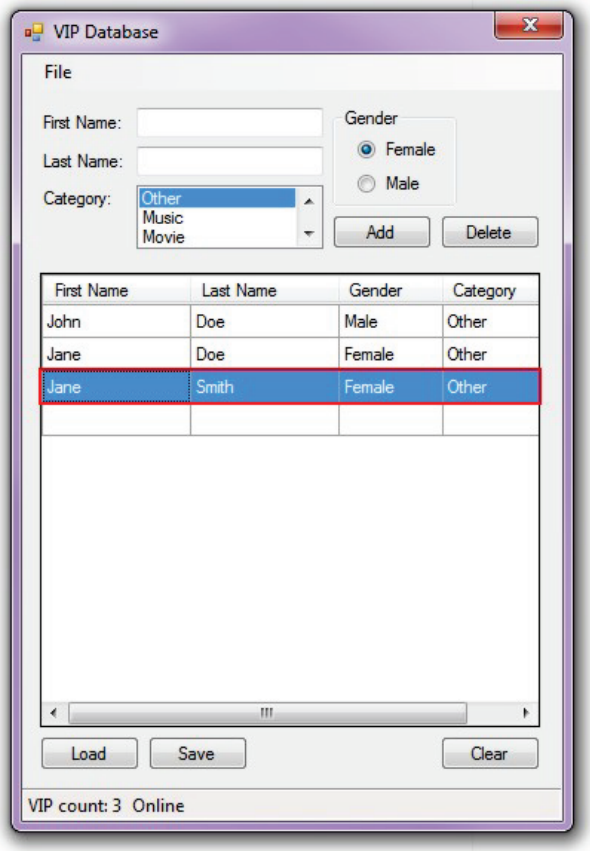
Relationship operators and attribute types

In the combo box, holding the relationship operators, the following types of relationship are defined:

child	Refers to all children of the current nod
descendant-or-self	Refers to all descendants (children, grandchildren, etc.) of the current node and the current node itself
ancestor	Refers to all ancestors (parent, grandparent, etc.) of the current node
self	Refers to the current node
descendant	Refers to all descendants (children, grandchildren, etc.) of the current node
parent	Refers to the parent of the current node

ancestor-or-self	Refers to all ancestors (parent, grandparent, etc.) of the current node and the current node itself
preceding-sibling	Refers to all siblings before the current node
following-sibling	Refers to all siblings after the current node

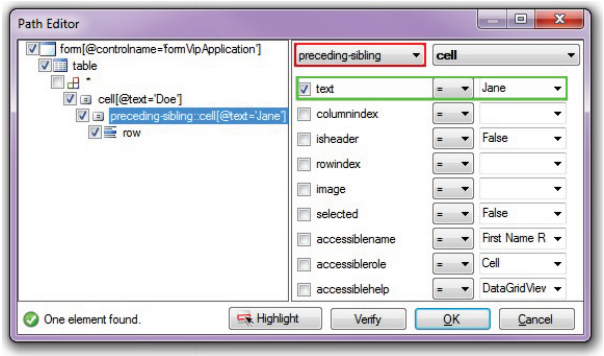
You will now see how the relationship operators work in practice, by doing a simple example with the preceding-sibling operator and the parent operator. In this simple example you modify your path to make it refer to the whole row holding the entry Jane Doe. But before that, you first have to add a new VIP called Jane Smith in the sample application, so that we have not only one Jane in our Database on which we can refer to:

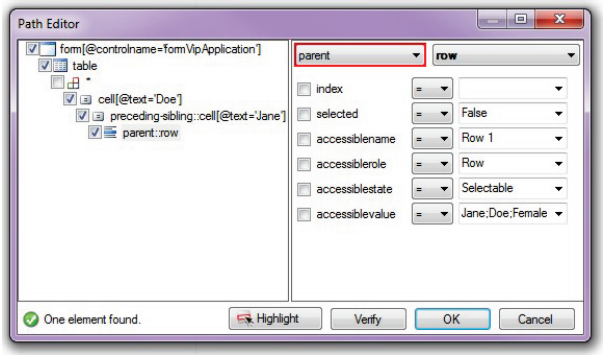


And after that you have to add two new layers in your path, one holding a cell and the other holding a row:

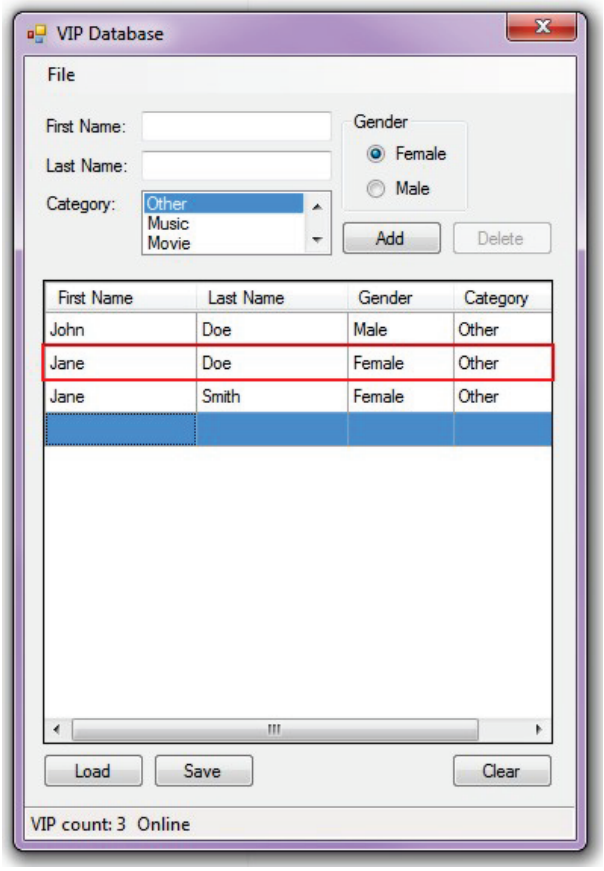
Now we have a cell which refers to the cells holding "Doe" as text value, a cell which will refer to the cell holding Jane as text value coming from one of the cells one layer above, and the row which is the parent of the cell one layer above again. This path won't refer to any UI element for now. To make the path refer to the row holding Jane Doe you have to

- **open the Advanced RanoreXPath Editor again,**
- **select the tree element representing the cell we added before,**
- **choose preceding sibling as relationship operator,**
- **add the equation, that the text value of the cell equals to Jane,**
- **select the tree element representing the row we added before,**
- **and choose the parent operator.**



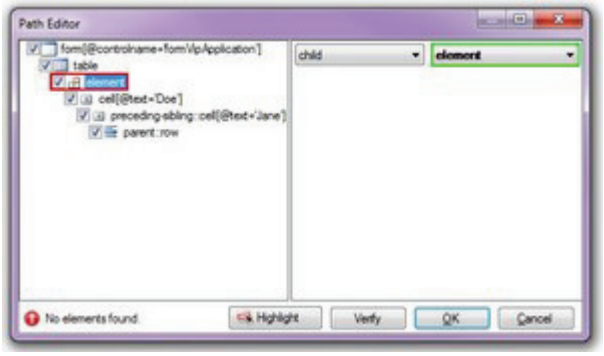


If you now verify your Path you should get one element matching your path. By highlighting the matching element you should see the row containing the entry Jane Doe.

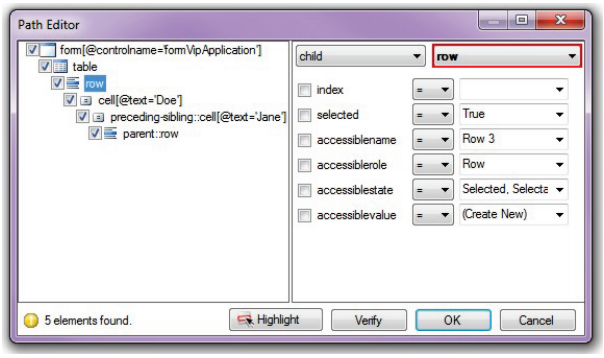


The attribute types can be used to define the type of an attribute if it is not defined right now. So in your example if you activate the equations, of the tree item where you have deactivated them in the tree view

before, again, the attribute type will be set to element. This is because the information about the attribute type of this layer is lost, thru making the layer match all types.

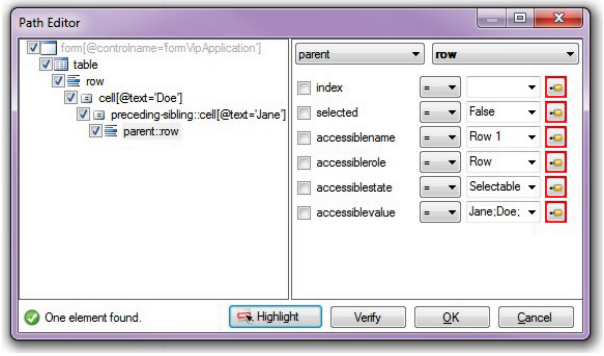


You can change the attribute type to the type row again, as we know that this element is a row control.

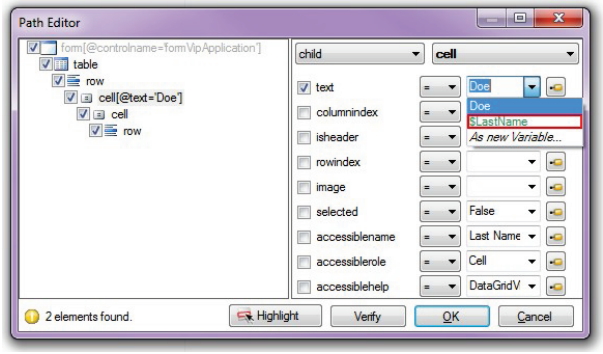


Defining Variables

If you start the Advanced Ranorex Path Editor from an existing Repository you have an additional button for each attribute equation.



With these buttons you can define variables which can be used for dynamically alter the RanoreXPath. The variables you have added are then inherited from the Repository you are working on. Each of the variables held by the Repository can be taken for attribute equations.



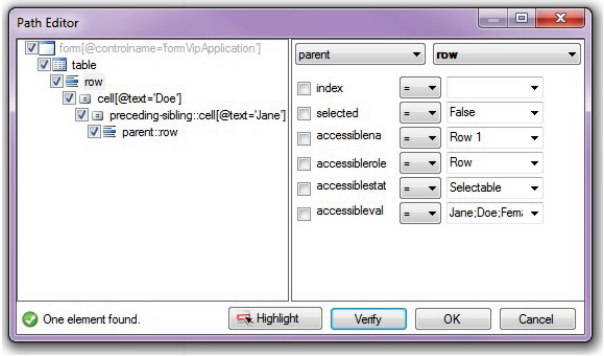
Creating variables for repository items

For more details about variables used within the repository click here.

Live and Offline View

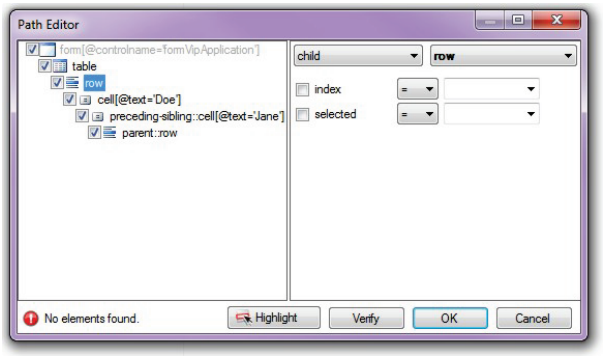
The last thing to mention is that if you are in Live View you basically get delivered more information about the adapter you are working on as not being in Live View and if you are not in Live View you cannot verify or highlight the elements matching your path.

If you open the editor with opened application under test you see following attributes:



Live view of VIP Database application

By closing the application you will only see following attributes in editor for the same adapter:



Offline view of VIP Database application