SMARTBEAR

# Getting Started

with

# TestComplete 14

## Desktop, Web, and Mobile Testing Tutorials

# About the Tutorial

With TestComplete, you can test applications of three major types: desktop, web and mobile:

- **Desktop applications -** these applications are executed on desktop computers running the Windows operating system.
- **Web applications** - these applications are executed in web browsers (including those web browsers that are embedded into desktop applications).
- **Mobile applications -** these applications are executed on Android or iOS devices.

This document is for novice users. It provides a brief overview of automated testing and of the product, and includes tutorials that explain how to create tests for major application types. After you read these tutorials, you will be able to create, modify and execute tests for desktop, web, and mobile applications.

# Table of Contents

# Introducing Automated Testing and TestComplete

## Automated Testing

**Software testing** is the process of investigating an application and finding errors in it. The difference between testing and simply exploring is that testing involves comparing the application output to an expected standard and determining whether the application functions as expected. In other words, the tester may need not only to ensure that the application displays a list of values, but also to verify that the list contains the appropriate values.

So, the basic test sequence includes –

- Defining the expected output.
- Performing test actions (feeding the appropriate input).
- Gathering the application output and comparing it to expected result (baseline data).
- Notifying developers or managers if the comparison fails.

**Automated testing** is the automatic execution of software testing by a special program with little or no human interaction. Automated execution guarantees that no test action will be skipped; it relieves testers of having to repeat the same boring steps over and over.

**TestComplete** provides special features for automating test actions, creating tests, defining baseline data, running tests, and logging test results. For example, it includes a special “**recording tests”** feature that lets you create tests visually. You just need to start recording, perform all the needed actions against the tested application and TestComplete will automatically convert all the “recorded” actions to a test. TestComplete also includes special dialogs and wizards that help you automate comparison commands (or **checkpoints**) in your tests.

## Test Types

TestComplete supports various testing types and methodologies: unit testing, functional and GUI testing, regression testing, distributed testing and others (see *Different Ways of Testing* in TestComplete Help). In this tutorial, we will create functional tests - the kind that is used most often. Functional tests check the interface between the application on one side, and the rest of the system and users on the other side. They verify that the application functions as expected.

A typical functional test consists of test commands that perform various actions such as simulating clicks and keystrokes, running test commands in a loop and verifying object contents.

In TestComplete, functional tests can be created in the form of **keyword tests** and **scripts**. Tests of both kinds can be **recorded** or **created from scratch** with built-in editors. Creating keyword tests is visual, easy and does not require a programming background. Scripting requires understanding script commands, but gives you the ability to create more powerful and flexible tests. TestComplete supports scripting in:

- JavaScript,
- Python,

- VBScript,
- and other languages.

You can create scripts in the language you know best. To learn about supported languages and how to select a language to create scripts, see the *Selecting the Scripting Language* section in the TestComplete Help.

In this tutorial, we will use the keyword testing feature.

## TestComplete Projects and Project Items

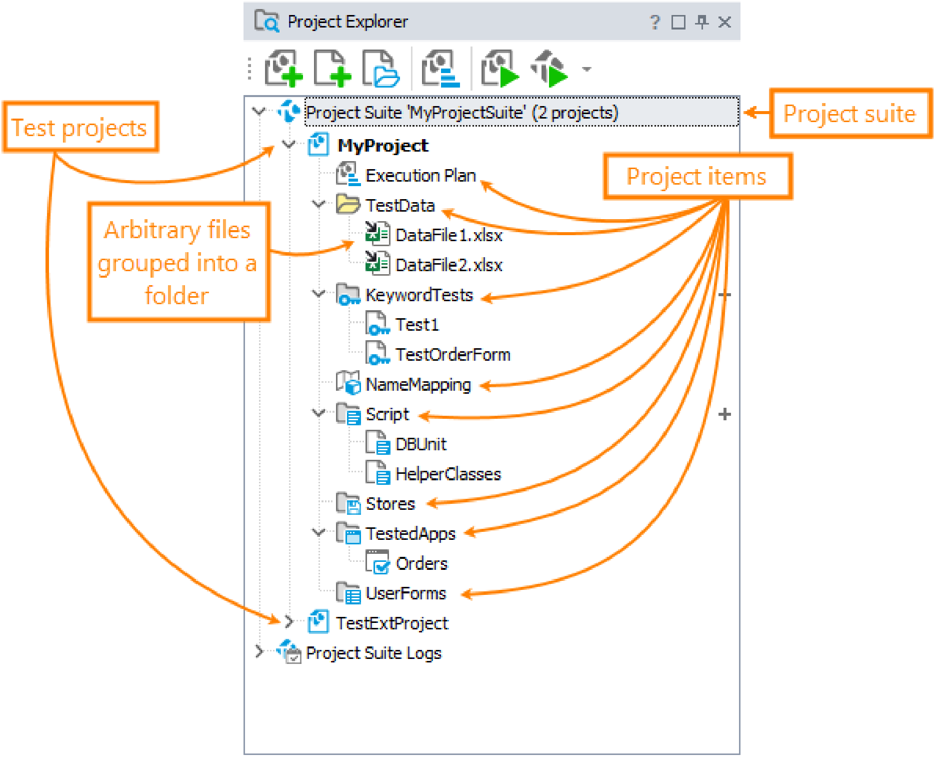
TestComplete operates with test projects and project suites. A **project** is a starting point for creating tests. It contains your tests, baseline data for checkpoints, information about tested applications, and other items needed to perform testing. The project also defines the execution sequence of tests.

A project can contain all the tests for your application. For complex applications, you may choose to devote a project to just one part of the application and other projects to other parts (normally, modules).

You can group related projects into a **project suite**. TestComplete automatically generates a project suite when you create a new project. You can also create an empty project suite and add projects to it.

**Project items** are project elements that perform or assist in performing various testing operations. Besides project items, your projects can also include *helper files* like Excel or .csv files with test data.

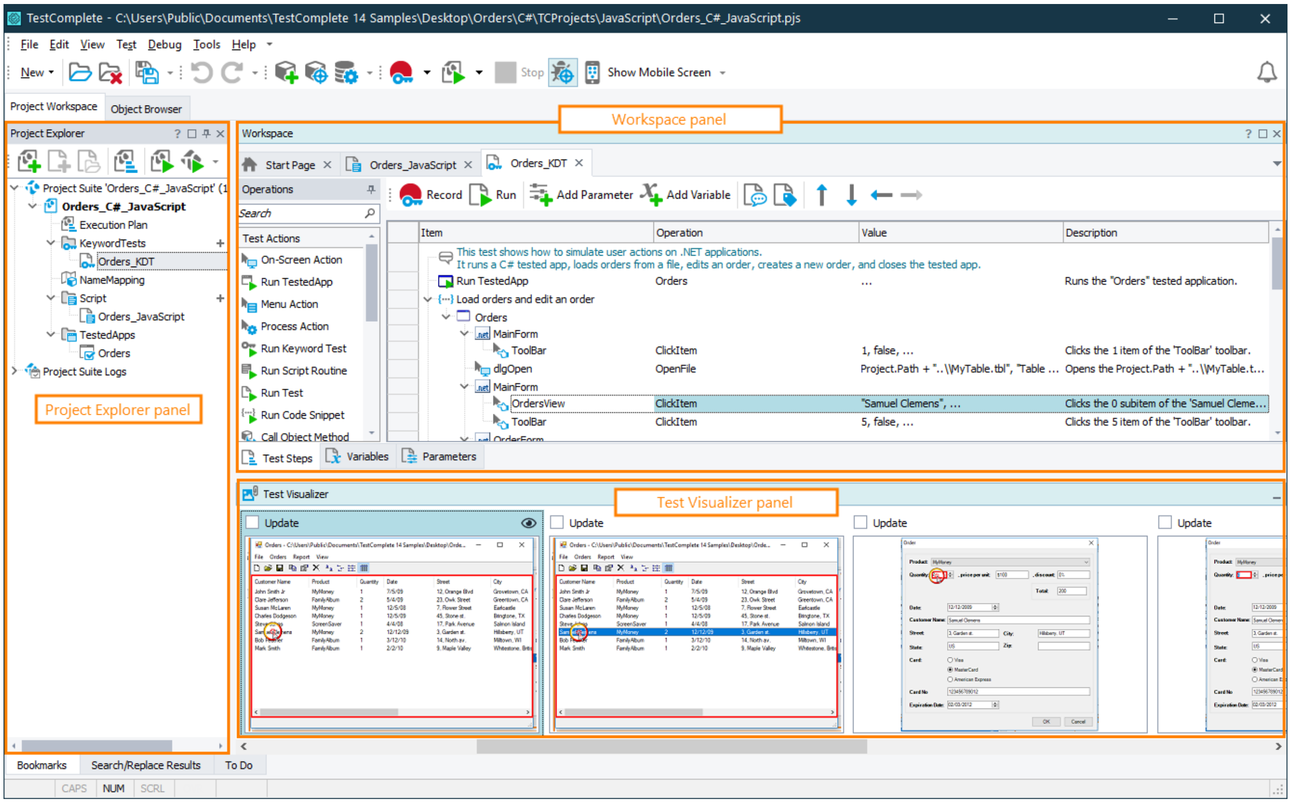
You can view and manage projects, project suites, and project items in the Project Explorer panel:



For complete information on project items available in TestComplete, see *About Project Items* in TestComplete Help.

# TestComplete User Interface

Here is a sample image of TestComplete main window:



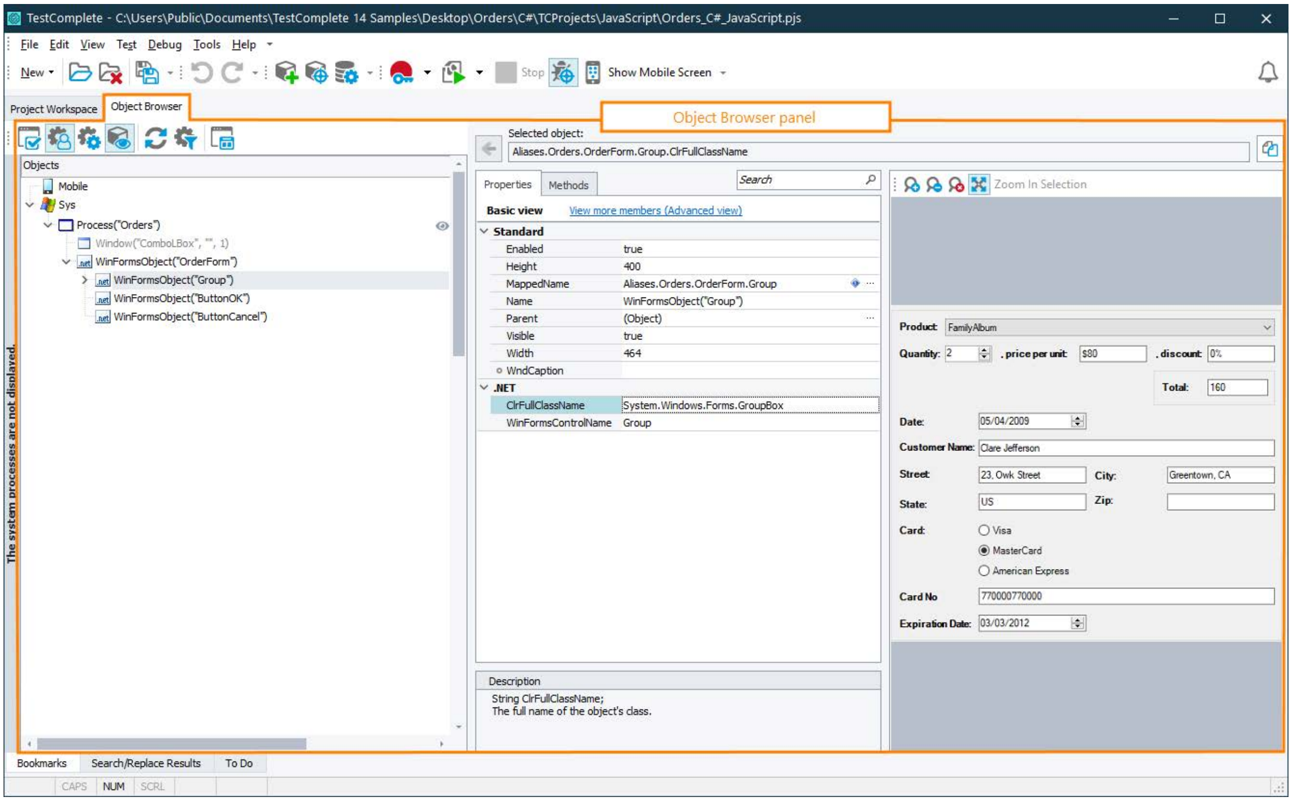
| Item | Operation | Value | Description |
|---|---|---|---|
| This test shows how to simulate user actions on .NET applications.<br>It runs a C# tested app, loads orders from a file, edits an order, creates a new order, and closes the tested app. | | | |
| Run TestedApp | Orders | ... | Runs the "Orders" tested application. |
| Load orders and edit an order | | | |
| Orders | | | |
| MainForm | | | |
| ToolBar | ClickItem | 1, false, ... | Clicks the 1 item of the 'ToolBar' toolbar. |
| dlgOpen | OpenFile | Project.Path + "..\\MyTable.tbl", "Table ... | Opens the Project.Path + "..\\MyTable.t... |
| MainForm | | | |
| OrdersView | ClickItem | "Samuel Clemens", ... | Clicks the 0 subitem of the 'Samuel Cleme... |
| ToolBar | ClickItem | 5, false, ... | Clicks the 5 item of the 'ToolBar' toolbar. |
| OrderForm | | | |

As you can see, TestComplete user interface is organized into a number of panels. The **Project Explorer** panel (on the left of the window) displays the contents of projects and the project suite. It also provides links to the test log nodes.

The **Workspace** panel is your working desktop: it displays the project and project item editors, where you create and modify tests and view test results. For instance, on the image above you can see the Keyword Test editor opened in the Workspace. Below the editor there is a **Test Visualizer** panel that displays images which the test engine captured during recording for test commands. These images help you understand the actions which test commands perform.

Besides the Project Explorer, Workspace, and Test Visualizer, TestComplete contains other panels. For example, the *Watch List*, *Locals*, *Breakpoints*, and *Call Stack* panels are used for test debugging. The *To Do* panel manages a list of tasks to be done.

The **Object Browser** panel holds one major TestComplete function that does not belong to a specific project: it shows the list of all processes and windows that exist on the machine. It also lists the processes of mobile applications, if the mobile device is connected, and the application is prepared in a special way. The Object Browser displays objects recognized by TestComplete and has a tree hierarchy. During test recording, TestComplete uses these objects and their methods and properties. In other words, the Object Browser tells you which objects, methods, and properties are available for testing, and how to get to them. *See Exploring Application Properties* in TestComplete Help.



TestComplete - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders\C#\TCProjects\JavaScript\Orders_C#_JavaScript.pjs

Selected object: Aliases.Orders.OrderForm.Group.ClrFullClassName

| | |
|---|---|
| **Standard** | |
| Enabled | true |
| Height | 400 |
| MappedName | Aliases.Orders.OrderForm.Group |
| Name | WinFormsObject("Group") |
| Parent | (Object) |
| Visible | true |
| Width | 464 |
| WndCaption | |
| **.NET** | |
| ClrFullClassName | System.Windows.Forms.GroupBox |
| WinFormsControlName | Group |

Description
String ClrFullClassName;
The full name of the object's class.

Product: FamilyAlbum
Quantity: 2 . price per unit: $80 . discount: 0%
Total: 160
Date: 05/04/2009
Customer Name: Clare Jefferson
Street: 23, Owk Street City: Greentown, CA
State: US Zip:
Card: ○ Visa ● MasterCard ○ American Express
Card No 770000770000
Expiration Date: 03/03/2012

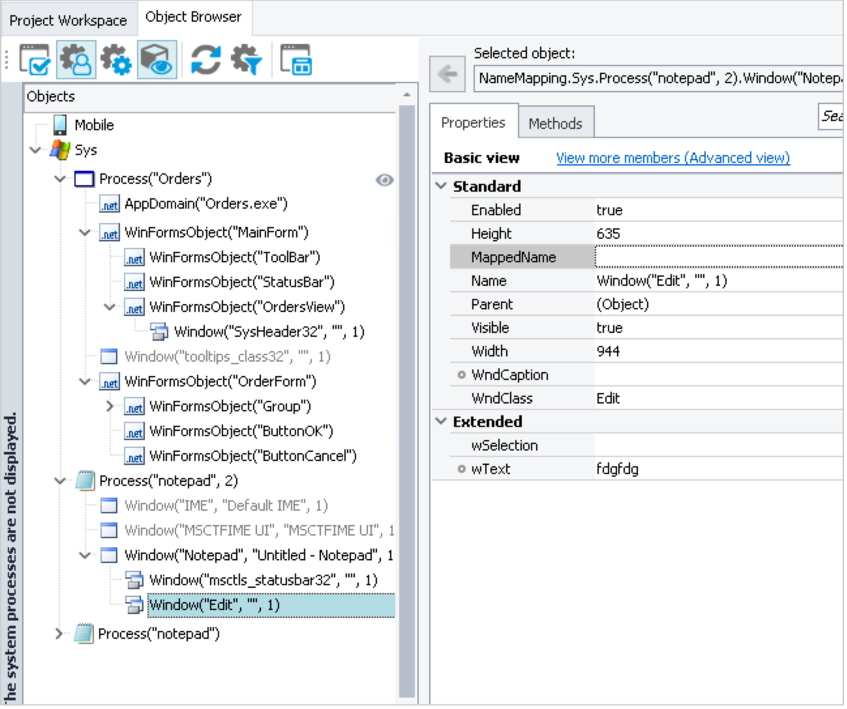
To learn about a panel, click within this panel, and then press F1. This will open the panel description.

You use menus and toolbars to command TestComplete to perform certain actions. Its menu subsystem is similar to the menus and toolbars of Microsoft Visual Studio and other popular Windows applications. You can change the toolbars location, move items from one menu or toolbar to another, hide items, add hidden items back and perform other tasks. For more information, see *Working With TestComplete Toolbars and Menus* in TestComplete Help.

## TestComplete Test Object Model

In automated tests, to simulate user actions on an object (a window, control, web page element) of a tested application, you first need to identify that object: instruct TestComplete how to locate the object. TestComplete is able to recognize individual windows and controls in applications and provide your tests with access to those objects.

Recognized objects are shown in the *Object Browser* panel of TestComplete:


Project Workspace
Object Browser
Objects
Mobile
Sys
Process("Orders")
AppDomain("Orders.exe")
WinFormsObject("MainForm")
WinFormsObject("ToolBar")
WinFormsObject("StatusBar")
WinFormsObject("OrdersView")
Window("SysHeader32", "", 1)
Window("tooltips_class32", "", 1)
WinFormsObject("OrderForm")
WinFormsObject("Group")
WinFormsObject("ButtonOK")
WinFormsObject("ButtonCancel")
Process("notepad", 2)
Window("IME", "Default IME", 1)
Window("MSCTFIME UI", "MSCTFIME UI", 1
Window("Notepad", "Untitled - Notepad", 1
Window("msctls_statusbar32", "", 1)
Window("Edit", "", 1)
Process("notepad")
he system processes are not displayed.
Properties
Methods
Sea

Selected object: NameMapping.Sys.Process("notepad", 2).Window("Notep

**Basic view** View more members (Advanced view)

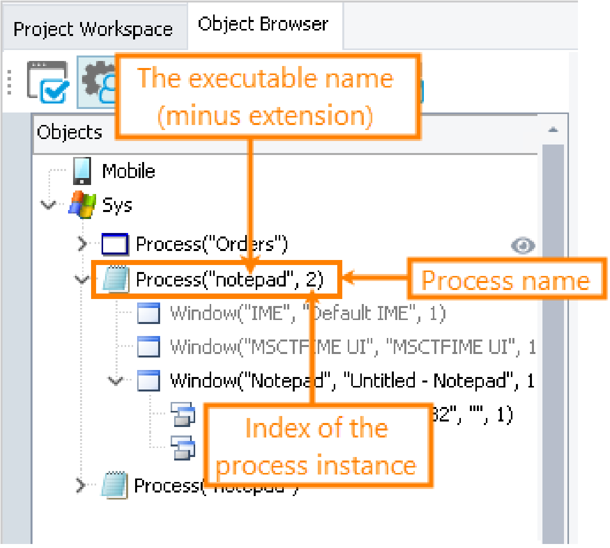
| | |
|---|---|
| **Standard** | |
| Enabled | true |
| Height | 635 |
| MappedName | |
| Name | Window("Edit", "", 1) |
| Parent | (Object) |
| Visible | true |
| Width | 944 |
| WndCaption | |
| WndClass | Edit |
| **Extended** | |
| wSelection | |
| wText | fdgfdg |

**Note:** The images in this topic demonstrate the object model of desktop applications. The object model of web and mobile applications is similar.

TestComplete uses a tree-like model for test objects. For desktop and web applications, the root node of the tree is **Sys**, while for mobile applications, the root node is **Mobile**.

**Process** objects correspond to applications running in the operating system. We use the term *process* rather than *application* because it corresponds to the concept of processes in Windows documentation.

A process object name includes the name of the process executable and its index (the index is used only if several application instances are running):

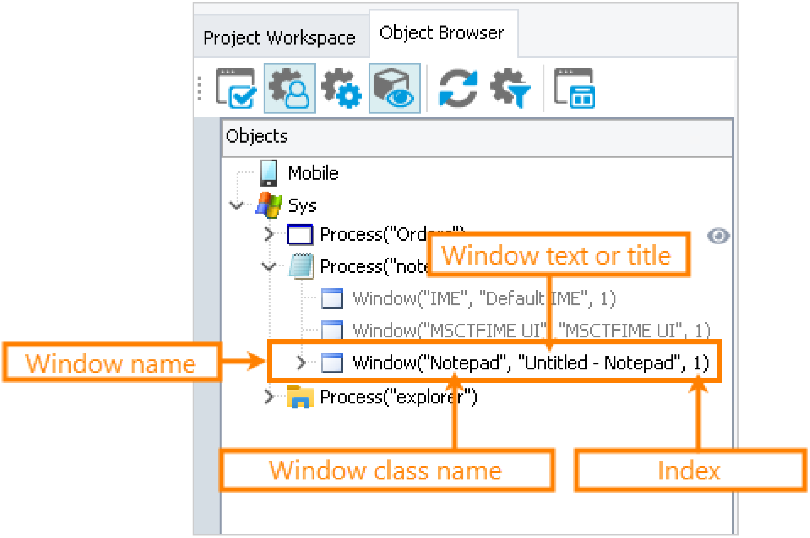

Project Workspace
Object Browser
The executable name (minus extension)
Objects
Mobile
Sys
Process("Orders")
Process("notepad", 2)
Process name
Window("IME", "Default IME", 1)
Window("MSCTFIME UI", "MSCTFIME UI", 1
Window("Notepad", "Untitled - Notepad", 1
32", "", 1)
Index of the process instance
Process("notepad")

The processes have child objects – windows – that correspond to top-level windows. These objects in their turn have other child window objects that correspond to controls. The window and control names depend on whether the test engine has access to internal methods and properties of the application under test. TestComplete works with applications of both types, but names their windows and controls in different ways.

The full name of an object in this structure can be long and hard to read, so you can assign shorter custom names to objects. This way your tests will be easier to read and maintain.

- **Black-box applications**

  Applications that do not provide access to their internal methods and properties are called **black-box applications**. The name of each window of such applications includes the window class name, the window text or title (caption) and its index. Controls are named in the same manner as windows, because in terms of the operating system, a control is just another type of a window:

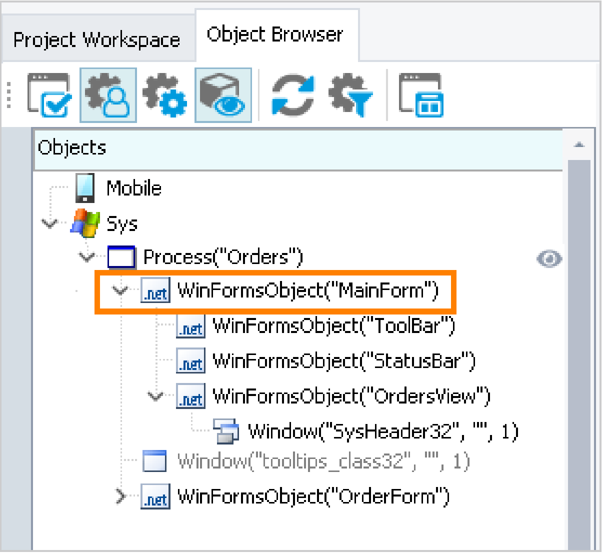


```
Mobile
Sys
  Process("Ord
  Process("note
    Window("IME", "Default IME", 1)
    Window("MSCTFIME UI", "MSCTFIME UI", 1)
    Window("Notepad", "Untitled - Notepad", 1)
  Process("explorer")
```

- **White-Box Applications**

  Applications that expose their internal objects, methods, and properties to TestComplete are called **white-box applications** or **Open Applications**. They are marked with the ◆ icon in the Object Browser (see the image below).

  To address windows and controls of Open Applications, TestComplete uses the names that reflect the window or control type and the name defined in the application sources. For instance, if you have a form named `MainForm` in a C# application created with the Microsoft WinForms library, then TestComplete will address this form as `WinFormsObject("MainForm")`:


Project Workspace
Object Browser
Objects
Mobile
Sys
Process("Orders")
WinFormsObject("MainForm")
WinFormsObject("ToolBar")
WinFormsObject("StatusBar")
WinFormsObject("OrdersView")
Window("SysHeader32", "", 1)
Window("tooltips_class32", "", 1)
WinFormsObject("OrderForm")

For detailed information on naming processes, windows, and controls, see *Naming Objects* in TestComplete Help.

**Note:** It is recommended that, whenever possible, your tests work with Open Applications rather than black-box applications. This enables the test engine to access the application internal methods and properties, allowing you to create more powerful and flexible tests.

Some applications like .NET, WPF, Visual Basic, Java or Web are always “open” to TestComplete. Others may need to be compiled in a special way. For more information on this, see *Open Applications* in TestComplete Help.

## Checkpoints and Stores

A typical test performs many comparisons. For instance, if a test simulates user actions for exporting some application data to a file, you will need to check whether the file contains valid data. To perform this check, you will compare the resulting file with a baseline copy. This is only one example of a comparison that you may need to perform. Real-life tests include hundreds if not thousands of comparisons. Every form of testing (regression, unit, functional, and so on) needs a validated reference during automation.

With TestComplete you can easily add comparison commands (or **checkpoints**) to your tests. You can create checkpoints both during test recording and at design time. TestComplete offers checkpoints for comparing different types of data: images, files, object text and properties, XML documents, database tables, etc. TestComplete includes the **Stores** project item that is used to store baseline data for these checkpoints. This project item is a container for images, files, and other elements that are stored along with the project for comparison purposes. The only exception is checkpoints that verify object properties: the baseline data for them is specified in tests.

For more information on creating checkpoints and verification code, see *About Checkpoints* in TestComplete Help.

# Testing Desktop Applications

TestComplete supports testing of 32- and 64-bit Windows desktop applications created with C++, C#, VB.NET, Java, Delphi, C++ Builder and with *many other development tools*.

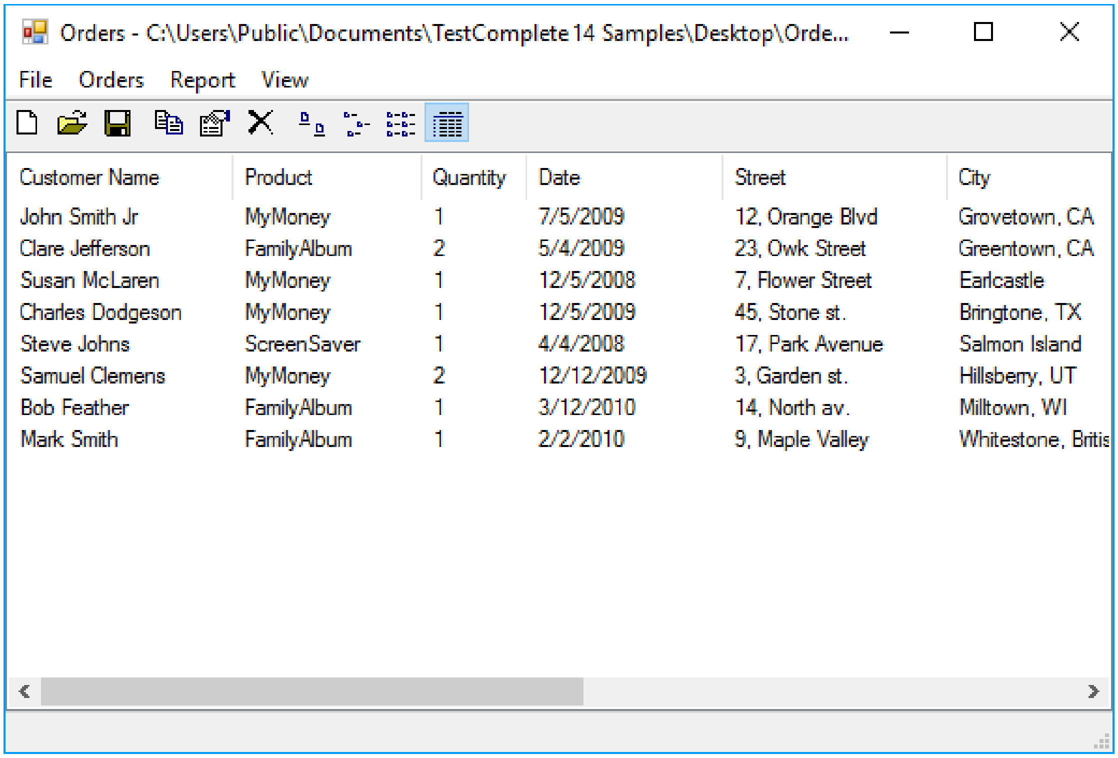
This tutorial explains the basics of testing desktop applications (that is, applications that run on desktop computers). It assumes that you are familiar with general principles of automated testing and have minimal knowledge of the TestComplete IDE.

! If you are a novice user, we recommend that you read an introduction to automated testing described above.

The sections of this tutorial contain a description of how to create a test project in TestComplete, record and play back a simple test, and analyze the results. The test emulates user actions over the tested application and verifies some data. The verification commands are created during test recording.

## About Tested Application

In our explanations, we will use the *Orders* application that is shipped along with TestComplete. The application displays a list of orders and contains special functions for adding, deleting, modifying, and exporting orders.


Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orde...
File
Orders
Report
View

| Customer Name | Product | Quantity | Date | Street | City |
|---|---|---|---|---|---|
| John Smith Jr | MyMoney | 1 | 7/5/2009 | 12, Orange Blvd | Grovetown, CA |
| Clare Jefferson | FamilyAlbum | 2 | 5/4/2009 | 23, Owk Street | Greentown, CA |
| Susan McLaren | MyMoney | 1 | 12/5/2008 | 7, Flower Street | Earlcastle |
| Charles Dodgeson | MyMoney | 1 | 12/5/2009 | 45, Stone st. | Bringtone, TX |
| Steve Johns | ScreenSaver | 1 | 4/4/2008 | 17, Park Avenue | Salmon Island |
| Samuel Clemens | MyMoney | 2 | 12/12/2009 | 3, Garden st. | Hillsberry, UT |
| Bob Feather | FamilyAlbum | 1 | 3/12/2010 | 14, North av. | Milltown, WI |
| Mark Smith | FamilyAlbum | 1 | 2/2/2010 | 9, Maple Valley | Whitestone, Britis |

The application is located in the following folder:

*C:\Users\Public\Public Documents\TestComplete 14 Samples\Desktop\Orders*

**Note:** Some file managers can display the *Public Documents* folder as *Documents*.

In this tutorial, we will use the Orders application created with Visual C#.

> ❗ For the Orders projects created with other compilers (Delphi, C++Builder, Swing, and so on), download the TestComplete Samples installation package from the https://support.smartbear.com/downloads/testcomplete/samples page of our website and run it.

## 1. Plan the Test

The sample Orders application maintains a list of orders. Suppose, we need to test whether the Edit Order application form functions correctly and modifies data in the order list. In this case, we should define the following:

- **Test purpose:** The test should check whether the Edit Order form saves the modified data, and the changes are visible in the order list.
- **Testing steps:** Our test should simulate modifying order details, and then verify the data in the order list. We will record a test simulating user actions over the application. For simplicity, our test will "change" only one property of one order.
- **Checking and logging the test result:** If the change made to the order has been saved correctly, it should be visible in the order list. To check this, our test will compare the data in the list with an expected value. We will add a special comparison command to the test for this. This command will post the comparison results to the test log, so we will see whether the verification failed or passed successfully.

For more information on planning tests with TestComplete, see *Planning Tests* in TestComplete Help.

## 2. About Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually -** You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.
- **Record tests -** Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them to script lines or keyword test operations. You create a test visually, and, in one sense, you **record** the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

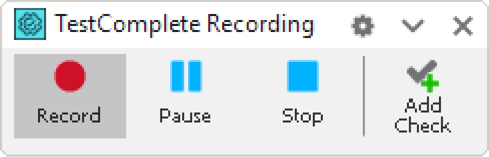
In this tutorial, we will demonstrate how to **record** tests with TestComplete.

The recording includes three steps:

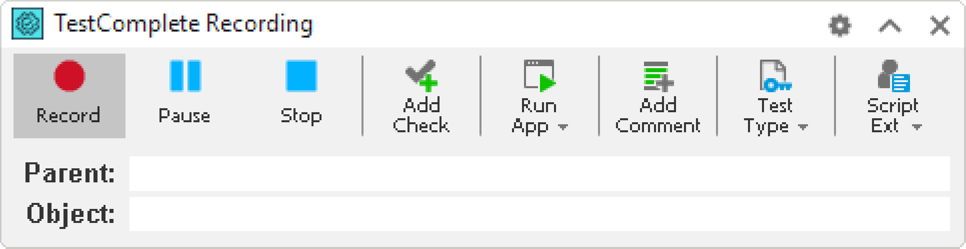
1. You start recording. You can do this by selecting **Test > Record > Record Keyword Test** or **Test > Record > Record Script** from the TestComplete main menu or from the Test Engine toolbar. You can also start recording by clicking ● **Record Test** on the Start Page.

You can record tests of various kinds: keyword tests, scripts, and low-level procedures. The menu item that you use to start the recording defines the main recorded test: keyword test or script code. Other tests will be recorded after the recording starts. The main recorded test will contain special commands that will run these tests.

TestComplete will switch to the recording mode and display the Recording toolbar on the screen. By default, the toolbar is collapsed and shows only the most commonly used commands you may need during the recording:



You can click the ⌄ arrow button to expand the Recording toolbar and view all its buttons:



You use the toolbar to perform additional actions during the recording, pause or stop recording, and change the type of the recorded test (keyword test, script code, or low-level procedure).

2. After starting the recording, perform the desired test actions: launch the tested application (if needed), work with it by clicking command buttons, selecting menu items, typing text, and so on.
3. After all the test actions are over, stop the recording by selecting ■**Stop** from the Recording toolbar.

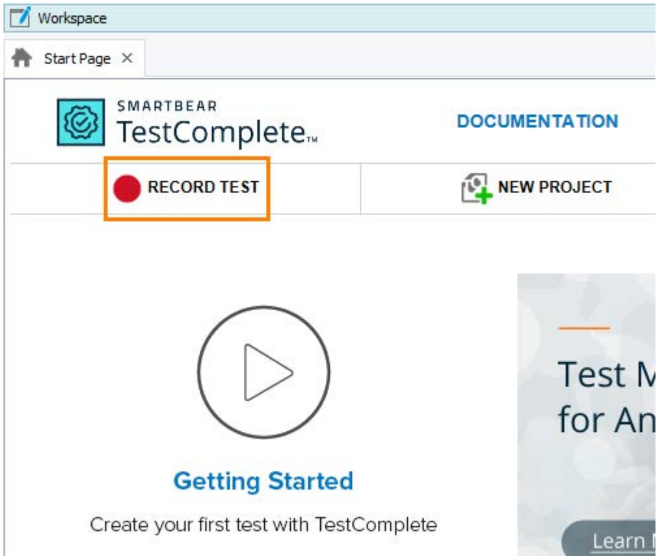
For complete information on test recording, see *Recording in TestComplete* in TestComplete Help.

For instructions on recording your first test, see *3. Start Test Recording* below.

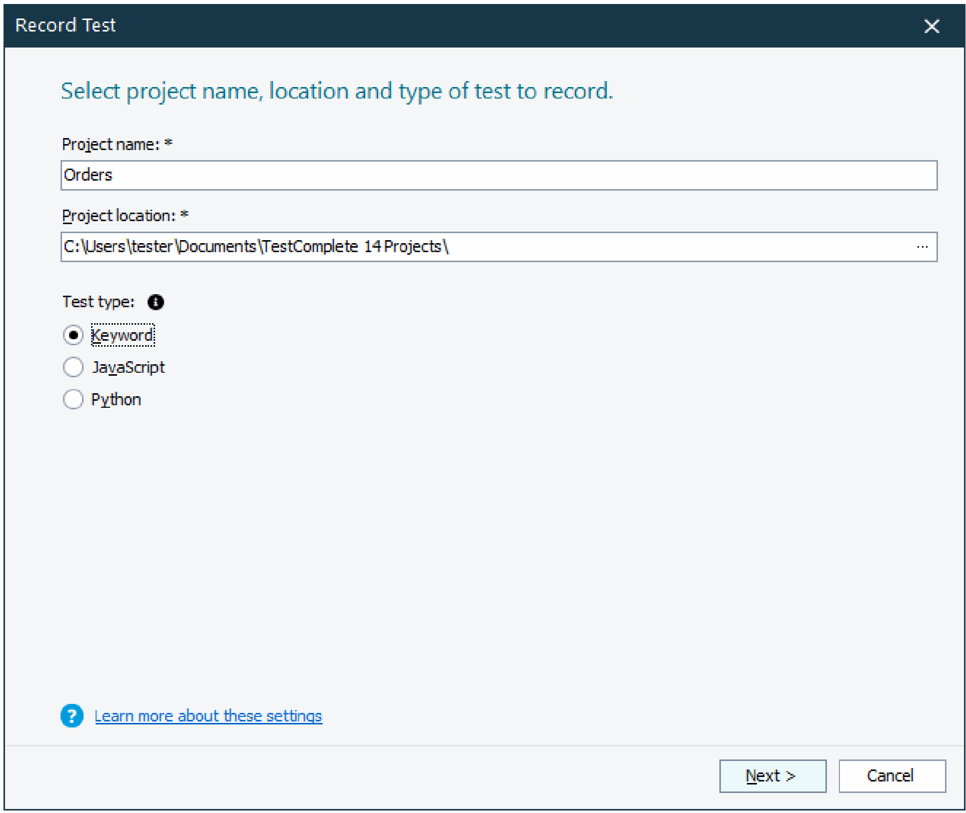
## 3. Start Test Recording

1. If a project or a project suite is open in TestComplete, close it. To do this, select **File** > **Close** from the TestComplete main menu.

2. Switch to the TestComplete **Start** page. If the page is hidden, select **Start Page** from the TestComplete **Help** menu.



3. On the Start page, click ● **Record Test**. TestComplete will show the **Record Test** wizard:



Select project name, location and type of test to record.

Project name: *

Orders

Project location: *

C:\Users\tester\Documents\TestComplete 14 Projects\

Test type:

● Keyword

○ JavaScript

○ Python

javascript:showImageEx(%22_images/tutorials/first-test/record-test-project.gif%22, %22Getting Started With TestComplete: Record Test wizard%22)

4. On the first page of the wizard, you can specify the project name, location and test type:

   In the **Project name** text box, enter Orders.

Leave the default value in the **Location** text box.

Select a test type. You can create either a keyword test, or a JavaScript or Python script test:

- A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.
- JavaScript and Python tests are script functions with instructions simulating user actions.

In this tutorial we will show how to create a keyword test. Click **Keyword**.

**5.** Click **Next** to continue.

We will continue working with the wizard to add our tested application to the project.

## 4. Define an Application to Test

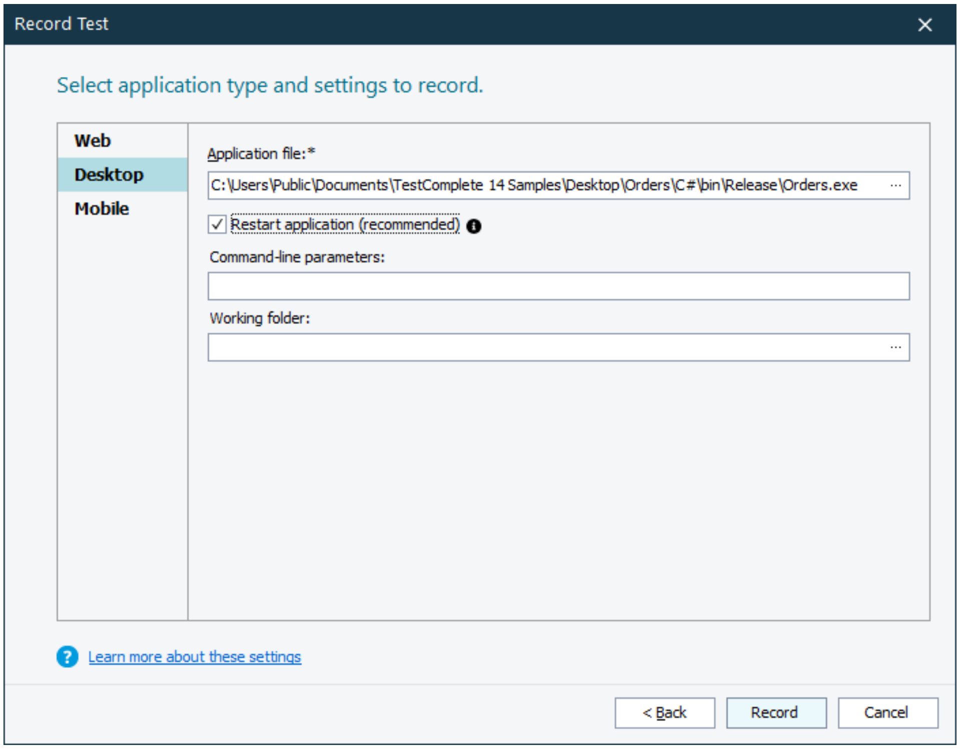
Each TestComplete project may have a list of tested applications. This way, you can keep track of which applications the project deals with, and how they are configured for testing.

There are several ways to add applications to your project:

- You can do this during project creation.
- You can do this at any time later, in the Project Explorer.
- TestComplete can add an application to a project automatically during test recording. The recorder is smart enough to detect the start of an application through the command line, Windows Explorer, or any other way.

Let’s add the tested Orders application to our project:

1. The wizard shows the second page where you can choose your tested application:



Select application type and settings to record.

Application file:* C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders\C#\bin\Release\Orders.exe

☑ Restart application (recommended)

Command-line parameters:

Working folder:

2. Since the tested Orders application is a .NET application that runs as a standalone executable, it falls under the Desktop application category.

   Click **Desktop**.

3. In the **Application file** box, click the ellipsis button. In the resulting **Select Tested Application** dialog, locate the Orders executable. The path is the following:

   *C:\Users\Public\Public Documents\TestComplete 14 Samples\Desktop\Orders\C#\bin\Release\Orders.exe*

   **Note:** Some file managers can display the *Public Documents* folders as *Documents*.

4. In order for the test run to be successful, the state of the tested application must be the same at the beginning of the test run and at the beginning of test recording. If your tested application is running when you start test recording, its state may differ from its state during subsequent test runs (for example, it may have some data loaded or dialogs may be open). That is why we recommend that you close all the existing instances of your tested application and start recording your tests from launching your tested application.

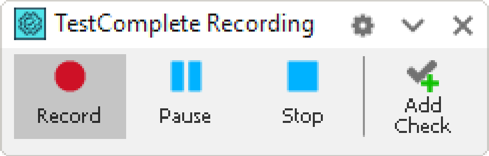
   Select the **Restart application** check box in order for TestComplete to restart all the existing instances of the Orders application automatically.

5. Click **Record** to complete the project creation and start recording.

## 5. Record a Test

! Do not switch to the TestComplete help during the recording. The recording engine traces and records all user actions, so the recorded test will contain commands that simulate “switching”. To see the instructions, you can print them before starting the record. Or, if you have two monitors, you can move the TestComplete help system window to the other monitor.

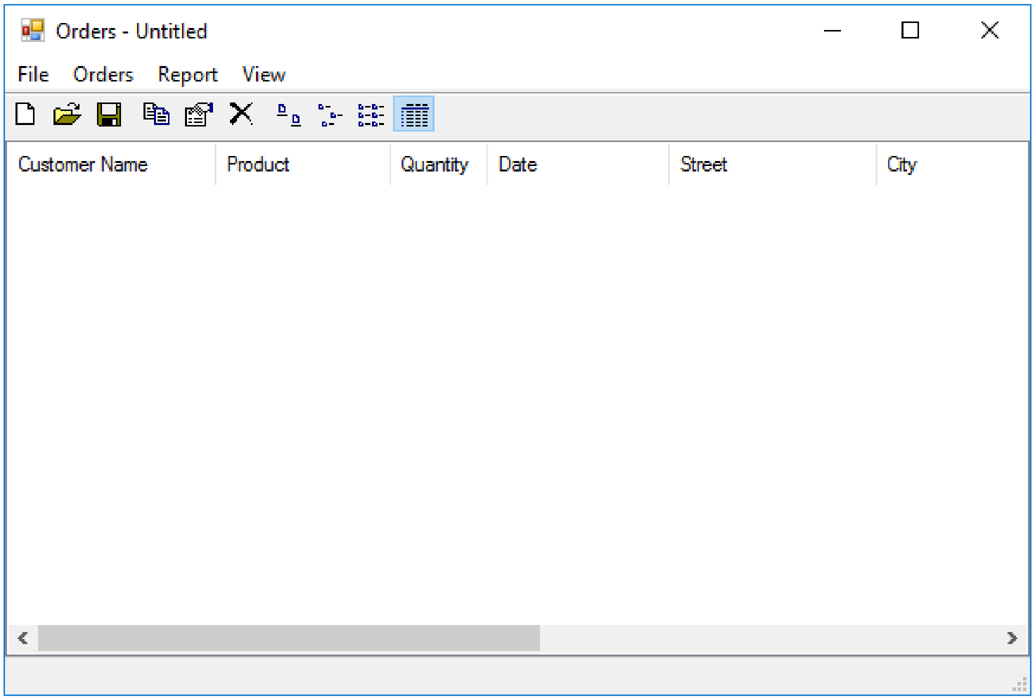
1. TestComplete will start the test recording, switch to the recording mode and show the Recording toolbar on screen:



2. When recording starts, TestComplete automatically launches the Orders tested application. If the application does not start, you can launch it manually by selecting it from the **Run App** menu of the Recording toolbar. You can also launch the application from Windows Explorer or any other file manager. If the application is not on the list of tested applications, TestComplete will add it there.

   TestComplete records the application start using a special application launch test command. You will see this command later, when we will analyze the recorded test.

3. Wait until the application starts, and the application main window appears:

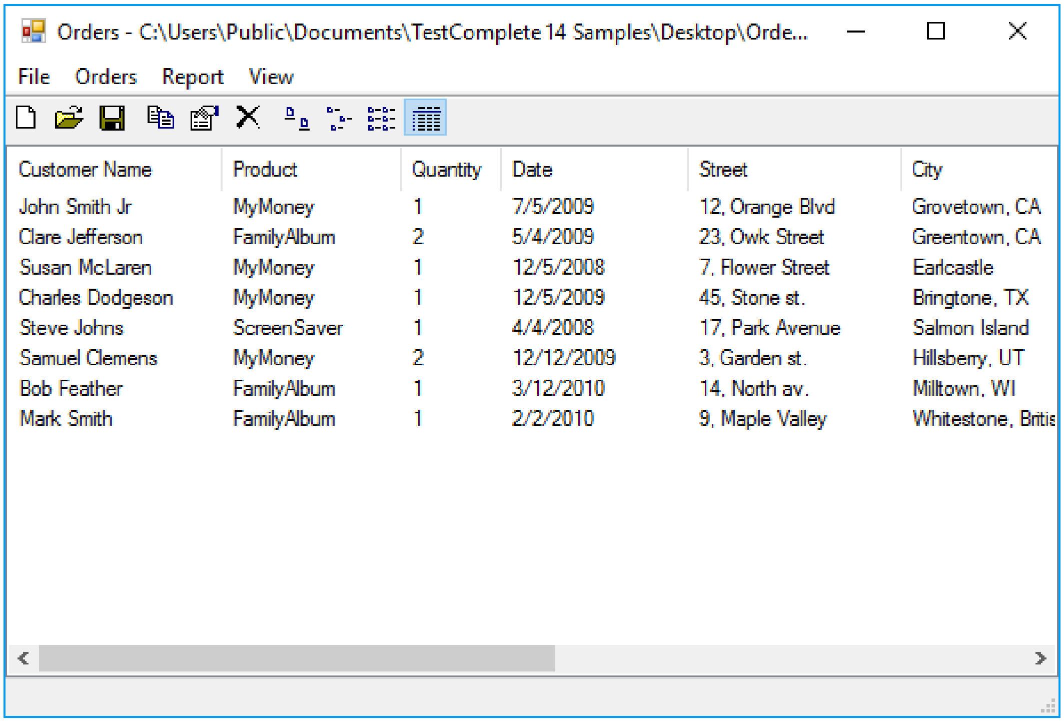


4. Switch to the Orders application and select **File** > **Open** from its main menu. This will bring up the standard Open File dialog.

5. In the dialog, open the *MyTable.tbl* file. It resides in the *C:\Users\Public\Public Documents\TestComplete 14 Samples\Desktop\Orders* folder. Some file managers can display the *Public Documents* folder as *Documents*.

> ! We recommend that you type the fully-qualified file name into the **File name** box of the Open File dialog. Typing instead of using the mouse will help you avoid problems if the test is played back on a different operating system, or if the Open File dialog displays a different initial folder when the test is played back later.

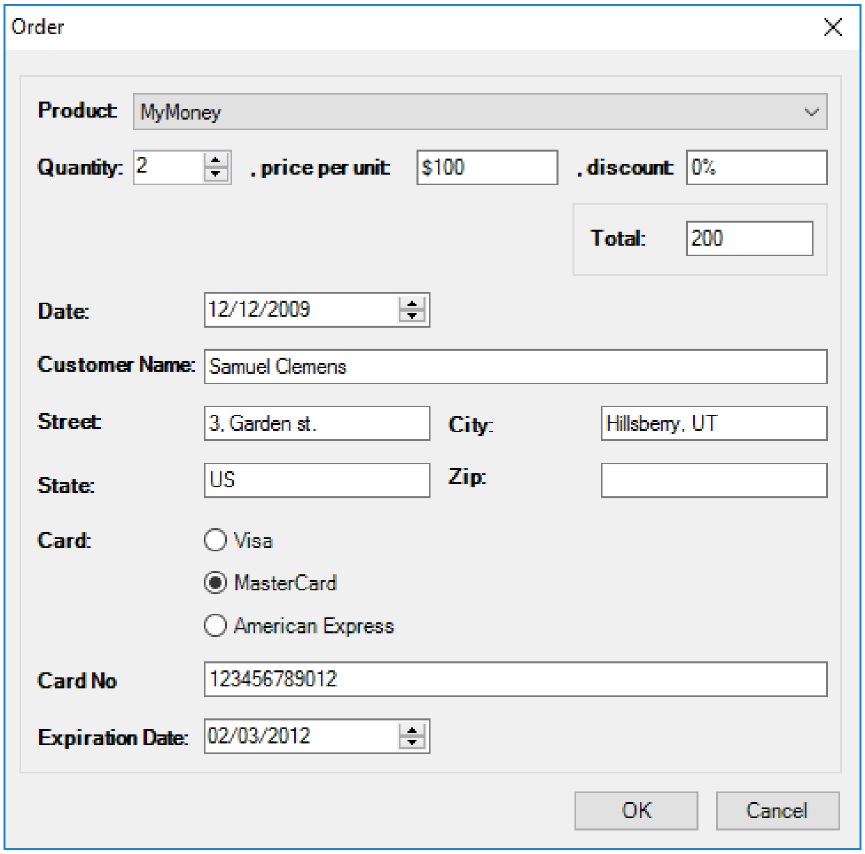
6. After specifying the file in the **File name** box, click **Open**. The Orders application will load data from the file and display this data in the application main window.


Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orde...
File
Orders
Report
View

| Customer Name | Product | Quantity | Date | Street | City |
|---|---|---|---|---|---|
| John Smith Jr | MyMoney | 1 | 7/5/2009 | 12, Orange Blvd | Grovetown, CA |
| Clare Jefferson | FamilyAlbum | 2 | 5/4/2009 | 23, Owk Street | Greentown, CA |
| Susan McLaren | MyMoney | 1 | 12/5/2008 | 7, Flower Street | Earlcastle |
| Charles Dodgeson | MyMoney | 1 | 12/5/2009 | 45, Stone st. | Bringtone, TX |
| Steve Johns | ScreenSaver | 1 | 4/4/2008 | 17, Park Avenue | Salmon Island |
| Samuel Clemens | MyMoney | 2 | 12/12/2009 | 3, Garden st. | Hillsberry, UT |
| Bob Feather | FamilyAlbum | 1 | 3/12/2010 | 14, North av. | Milltown, WI |
| Mark Smith | FamilyAlbum | 1 | 2/2/2010 | 9, Maple Valley | Whitestone, Britis |

7. Click the *Samuel Clemens* row in the list of orders.

8. On the Orders toolbar, click **Edit order**. This will call the **Order** dialog:



Product: MyMoney

Quantity: 2 , price per unit: $100 , discount: 0%

Total: 200

Date: 12/12/2009

Customer Name: Samuel Clemens

Street: 3, Garden st. City: Hillsberry, UT

State: US Zip:

Card: ○ Visa ● MasterCard ○ American Express

Card No 123456789012

Expiration Date: 02/03/2012

9. In the dialog, click within the **Customer Name** text box to move the insertion point there. Right-click within the Customer Name box and choose **Select All** from the context menu, and then enter *Mark Twain* as the customer name.

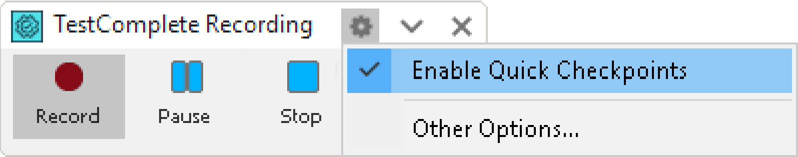
10. Click **OK** to close the dialog. TestComplete will update the customer list in the application main window.

11. Insert a comparison command into our test. It will verify that the application customer list displays the modified name - *Mark Twain*.

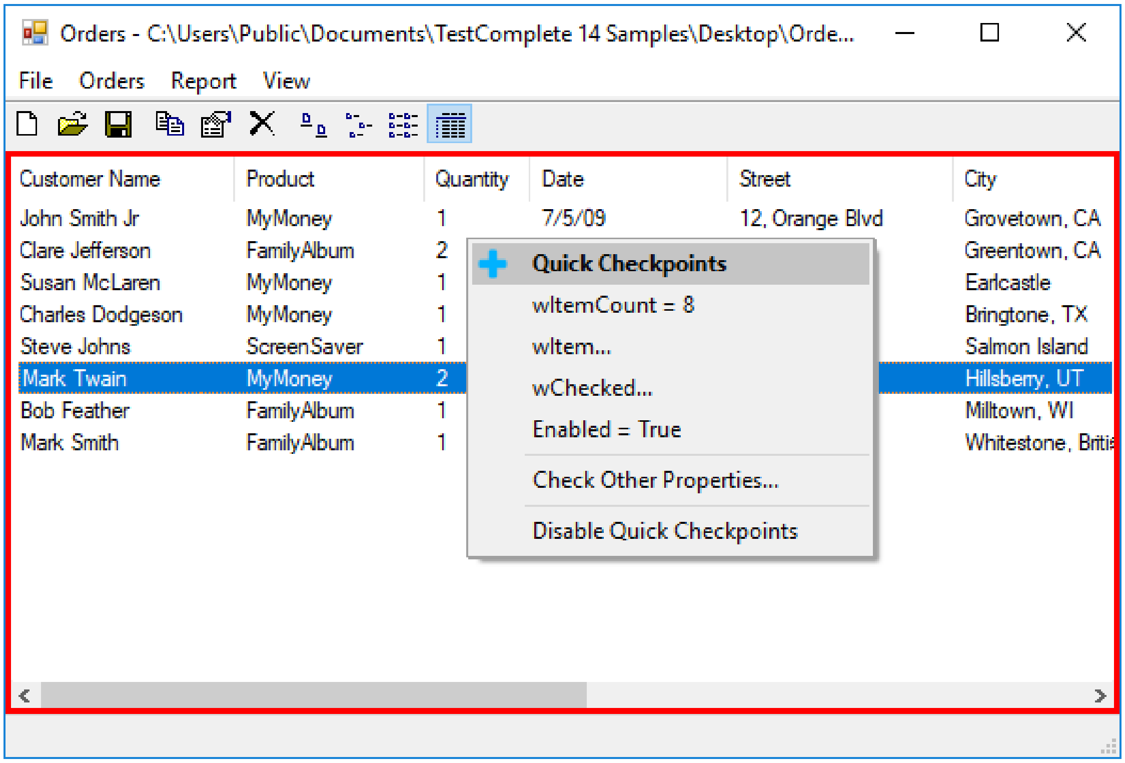
    We call the comparison commands **checkpoints**. TestComplete offers various types of checkpoints that are suitable for verifying different types of data (see *Checkpoints section* in TestComplete Help). One of the most frequently used checkpoints is a **Property checkpoint**. It is used to check data of applications controls. We will use this checkpoint in our tutorial.

    To create a property checkpoint, you can use the **Create Checkpoint** wizard, or you can create a **Quick Checkpoint**. In this tutorial, we will demonstrate how to create Quick Checkpoints:

- Click ⚙ on the **Recording** toolbar and make sure that Quick Checkpoints are enabled:

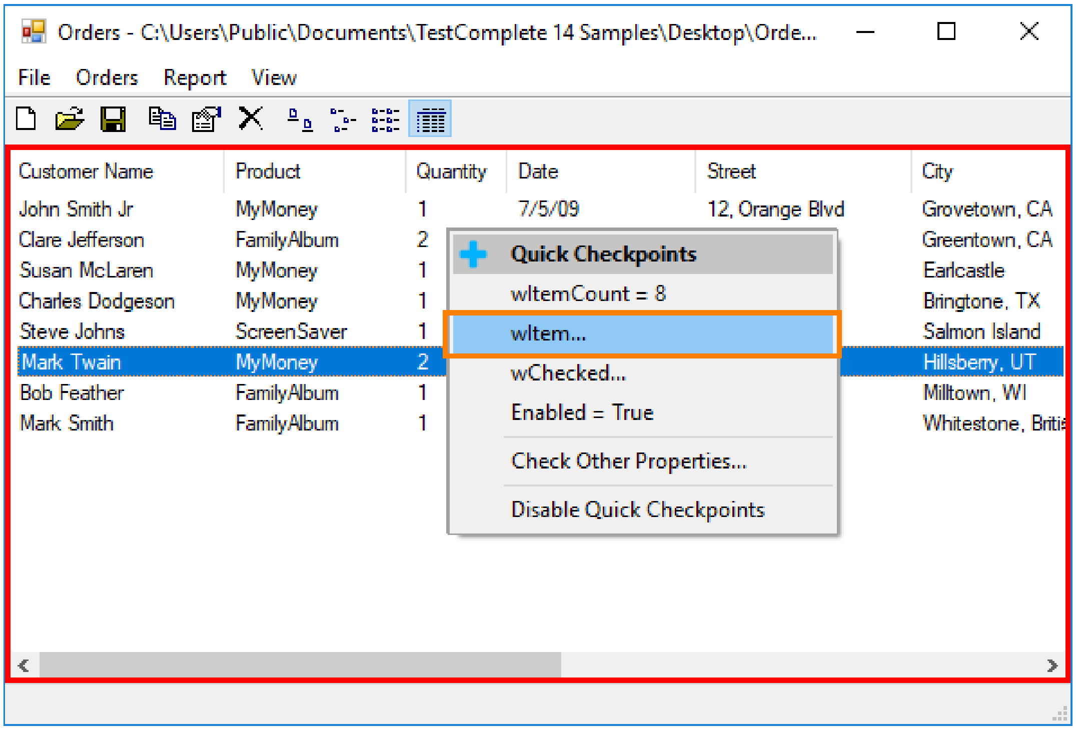


- Move the mouse pointer to the customer list. TestComplete will highlight it with a red frame. Wait until the ✚icon becomes opaque and move the mouse pointer to it. TestComplete will show a list of the most commonly used properties for which you can create Quick Checkpoints:


Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orde...
File
Orders
Report
View
Quick Checkpoints
wItemCount = 8
wItem...
wChecked...
Enabled = True
Check Other Properties...
Disable Quick Checkpoints

| Customer Name | Product | Quantity | Date | Street | City |
|---|---|---|---|---|---|
| John Smith Jr | MyMoney | 1 | 7/5/09 | 12, Orange Blvd | Grovetown, CA |
| Clare Jefferson | FamilyAlbum | 2 | | | Greentown, CA |
| Susan McLaren | MyMoney | 1 | | | Earlcastle |
| Charles Dodgeson | MyMoney | 1 | | | Bringtone, TX |
| Steve Johns | ScreenSaver | 1 | | | Salmon Island |
| Mark Twain | MyMoney | 2 | | | Hillsberry, UT |
| Bob Feather | FamilyAlbum | 1 | | | Milltown, WI |
| Mark Smith | FamilyAlbum | 1 | | | Whitestone, Briti |

- In the list, click **wItem**. This property provides access to individual items of tree view controls:

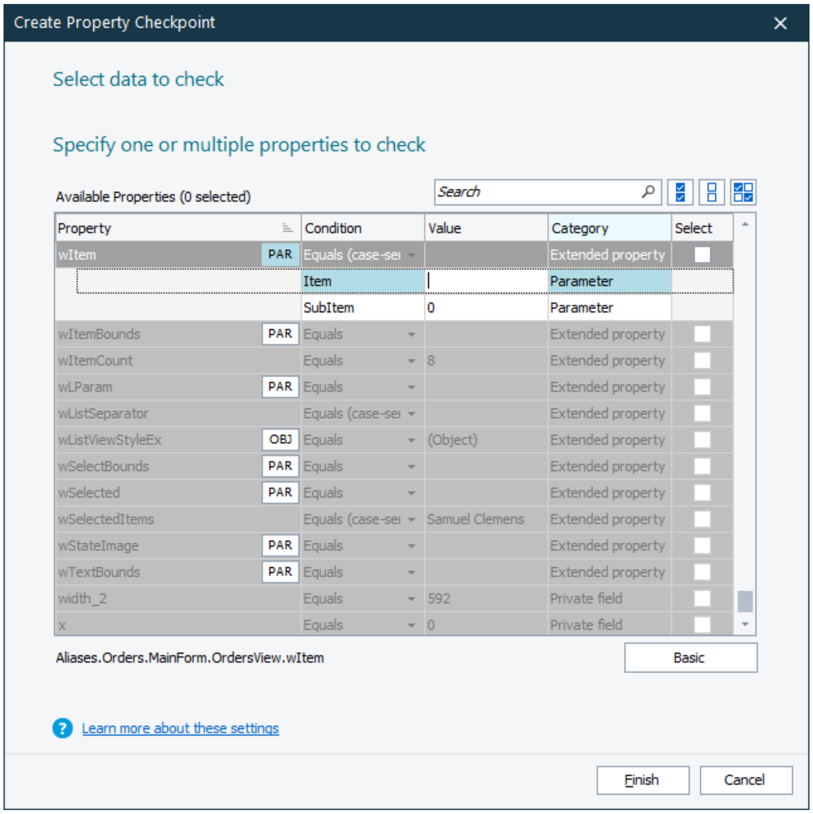


Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orde...
File
Orders
Report
View
Quick Checkpoints
wItemCount = 8
wItem...
wChecked...
Enabled = True
Check Other Properties...
Disable Quick Checkpoints

| Customer Name | Product | Quantity | Date | Street | City |
|---|---|---|---|---|---|
| John Smith Jr | MyMoney | 1 | 7/5/09 | 12, Orange Blvd | Grovetown, CA |
| Clare Jefferson | FamilyAlbum | 2 | | | Greentown, CA |
| Susan McLaren | MyMoney | 1 | | | Earlcastle |
| Charles Dodgeson | MyMoney | 1 | | | Bringtone, TX |
| Steve Johns | ScreenSaver | 1 | | | Salmon Island |
| Mark Twain | MyMoney | 2 | | | Hillsberry, UT |
| Bob Feather | FamilyAlbum | 1 | | | Milltown, WI |
| Mark Smith | FamilyAlbum | 1 | | | Whitestone, Britis |

- TestComplete will open the **Create Checkpoint** wizard that will show the list of object properties. This list includes properties provided by TestComplete as well as properties defined by the tested application. For instance, our tested application was created in C#, so the list includes properties of the appropriate .NET class. They are of the .Net category (see the **Available Properties** table).

  The `wItem` property we have clicked will be highlighted:



Create Property Checkpoint
×
Select data to check
Specify one or multiple properties to check
Search

Available Properties (0 selected)

| Property | | Condition | Value | Category | Select |
|---|---|---|---|---|---|
| wItem | PAR | Equals (case-sei | | Extended property | □ |
| | | Item | | Parameter | |
| | | SubItem | 0 | Parameter | |
| wItemBounds | PAR | Equals | | Extended property | □ |
| wItemCount | | Equals | 8 | Extended property | □ |
| wLParam | PAR | Equals | | Extended property | □ |
| wListSeparator | | Equals (case-sei | | Extended property | □ |
| wListViewStyleEx | OBJ | Equals | (Object) | Extended property | □ |
| wSelectBounds | PAR | Equals | | Extended property | □ |
| wSelected | PAR | Equals | | Extended property | □ |
| wSelectedItems | | Equals (case-sei | Samuel Clemens | Extended property | □ |
| wStateImage | PAR | Equals | | Extended property | □ |
| wTextBounds | PAR | Equals | | Extended property | □ |
| width_2 | | Equals | 592 | Private field | □ |
| x | | Equals | 0 | Private field | □ |

Aliases.Orders.MainForm.OrdersView.wItem

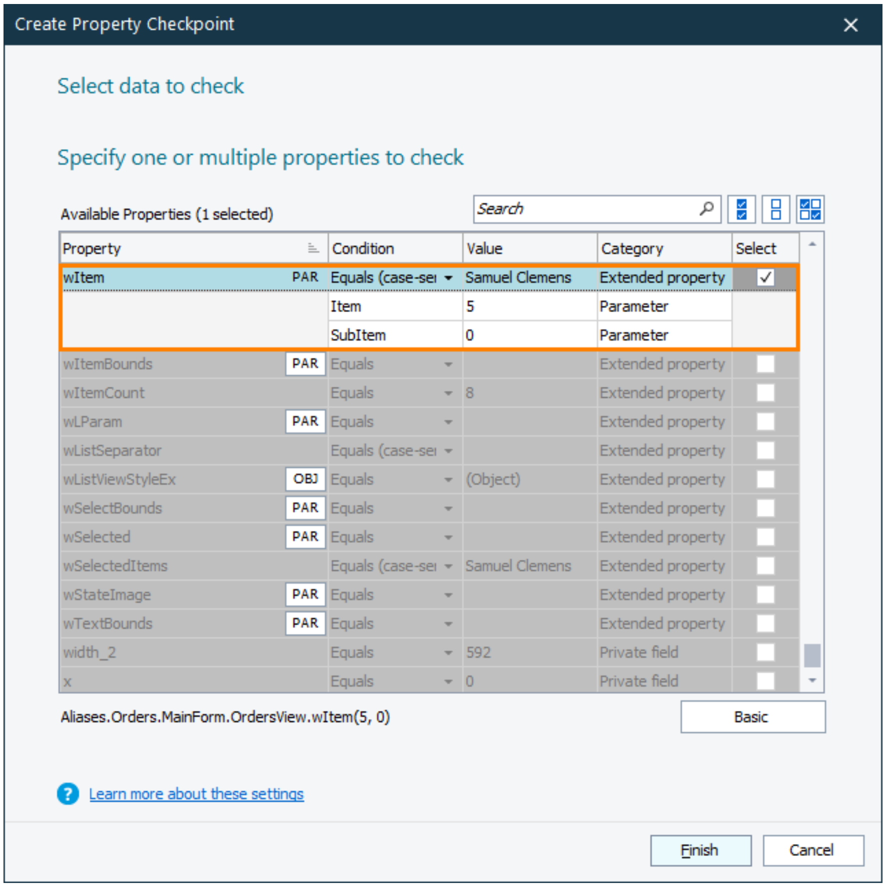
- To specify the cell holding the Mark Twain string, enter **5** into the **Item** box (5 is the index of the Mark Twain item in the tree view. Indexes are zero-based). Enter **0** in the **SubItem** box.

  The test engine will retrieve the item data and display it in the property list.

- In the **Condition** column, leave the default comparison condition, *Equals (case-sensitive)*.

- The **Value** column specifies the baseline data against which the checkpoint will compare the actual data during the test run. Leave the current value.

- Select the check box of the property:



Create Property Checkpoint ✕

Select data to check

Specify one or multiple properties to check

Available Properties (1 selected)

Search

| Property | | Condition | Value | Category | Select |
|---|---|---|---|---|---|
| wItem | PAR | Equals (case-sei | Samuel Clemens | Extended property | ☑ |
| | | Item | 5 | Parameter | |
| | | SubItem | 0 | Parameter | |
| wItemBounds | PAR | Equals | | Extended property | □ |
| wItemCount | | Equals | 8 | Extended property | □ |
| wLParam | PAR | Equals | | Extended property | □ |
| wListSeparator | | Equals (case-sei | | Extended property | □ |
| wListViewStyleEx | OBJ | Equals | (Object) | Extended property | □ |
| wSelectBounds | PAR | Equals | | Extended property | □ |
| wSelected | PAR | Equals | | Extended property | □ |
| wSelectedItems | | Equals (case-sei | Samuel Clemens | Extended property | □ |
| wStateImage | PAR | Equals | | Extended property | □ |
| wTextBounds | PAR | Equals | | Extended property | □ |
| width_2 | | Equals | 592 | Private field | □ |
| x | | Equals | 0 | Private field | □ |

Aliases.Orders.MainForm.OrdersView.wItem(5, 0)

- Click **Finish** to complete the checkpoint creation.

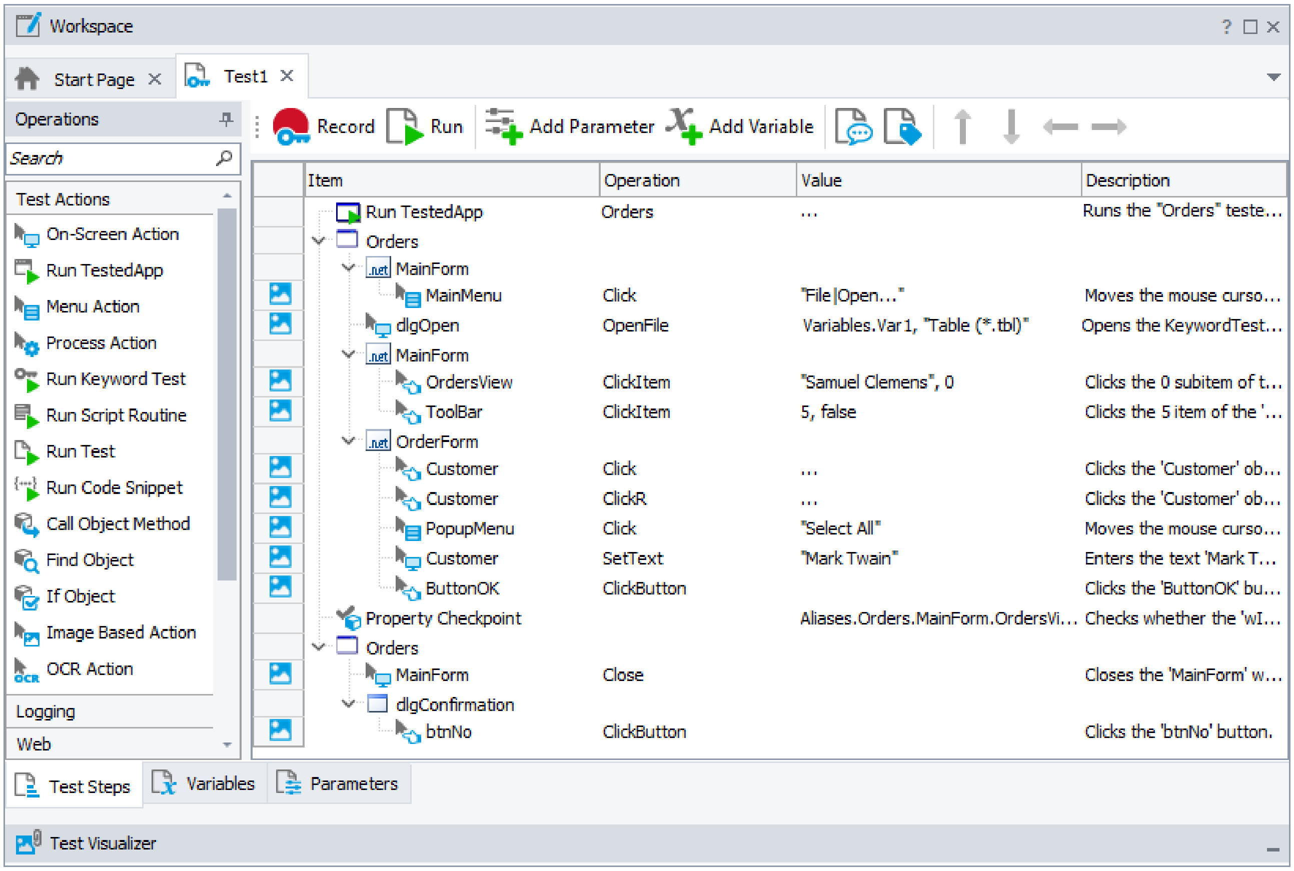
- TestComplete will append the checkpoint command to the recorded test and will show a notification about it. You can continue recording user actions.

**12.** Close the Orders window by clicking the **X** button on the window caption bar. This will display the dialog asking if you want to save changes. Click **No**.

**13.** Click 🟦 **Stop** on the Recording toolbar to stop the recording. TestComplete will process the recorded test commands and save them to a test.

## 6. Analyze the Recorded Test

After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test contents in the Keyword Test editor:

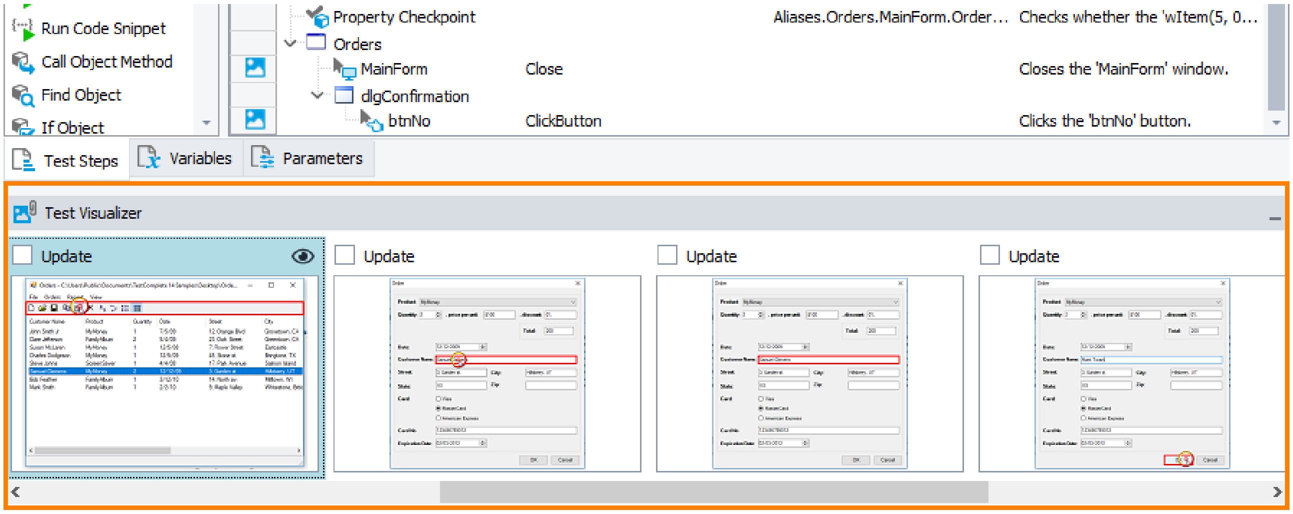


| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" teste... |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open..." | Moves the mouse curso... |
| dlgOpen | OpenFile | Variables.Var1, "Table (*.tbl)" | Opens the KeywordTest... |
| MainForm | | | |
| OrdersView | ClickItem | "Samuel Clemens", 0 | Clicks the 0 subitem of t... |
| ToolBar | ClickItem | 5, false | Clicks the 5 item of the '... |
| OrderForm | | | |
| Customer | Click | ... | Clicks the 'Customer' ob... |
| Customer | ClickR | ... | Clicks the 'Customer' ob... |
| PopupMenu | Click | "Select All" | Moves the mouse curso... |
| Customer | SetText | "Mark Twain" | Enters the text 'Mark T... |
| ButtonOK | ClickButton | | Clicks the 'ButtonOK' bu... |
| Property Checkpoint | | Aliases.Orders.MainForm.OrdersVi... | Checks whether the 'wI... |
| Orders | | | |
| MainForm | Close | | Closes the 'MainForm' w... |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

The recorded test is similar to the test shown in the image above. Your actual test may differ from this one. For example, it may have other object names or window indexes if you have recorded the test on a C++ Builder or Delphi application.

The test contains the commands that correspond to the actions you performed on the Orders application during the recording. We call the test commands **operations**.

Below the commands, there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:



Run Code Snippet
Call Object Method
Find Object
If Object

| Item | Operation | Value | Description |
|---|---|---|---|
| Property Checkpoint | | Aliases.Orders.MainForm.Order… | Checks whether the 'wItem(5, 0… |
| Orders | | | |
| MainForm | Close | | Closes the 'MainForm' window. |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

Test Steps | Variables | Parameters

Test Visualizer

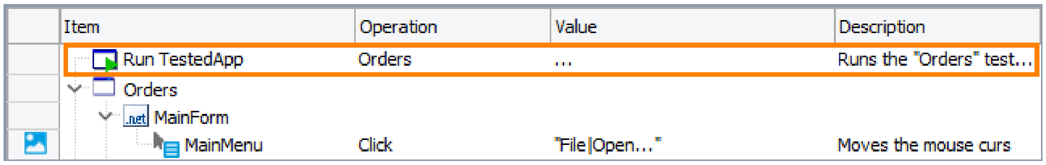
□ Update □ Update □ Update □ Update

These images illustrate the recoded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (mouse clicks, typing text and so on).

When you choose an operation in the editor, Test Visualizer automatically selects the appropriate image so you can easily explore the application state before the operation is executed. To view the needed image closely, double-click it in the Test Visualizer panel.

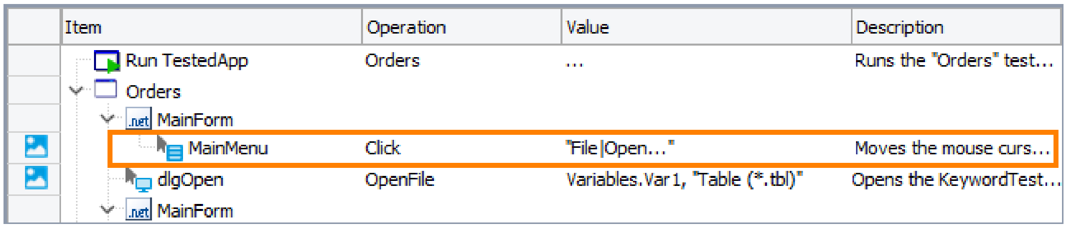
For more information on working with images, see the topics in the *Test Visualizer* section in TestComplete Help.

The first operation in the test is **Run TestedApp**. It is used to launch the tested application (in our case, it is the *Orders* application) from a keyword test. TestComplete automatically records this operation when it launches the application automatically or detects an application launch from the Recording toolbar or somewhere from the operating system UI.



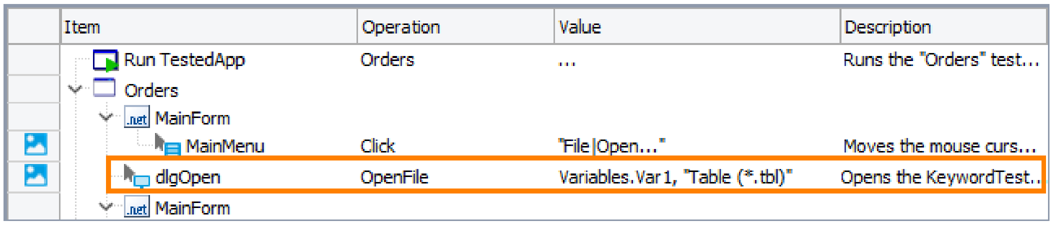
| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | … | Runs the "Orders" test… |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open…" | Moves the mouse curs |

The next operation corresponds to the selection of the **File** > **Open** menu item.



| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | … | Runs the "Orders" test… |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open…" | Moves the mouse curs… |
| dlgOpen | OpenFile | Variables.Var1, "Table (*.tbl)" | Opens the KeywordTest… |
| MainForm | | | |

The next operation simulates opening the file via the Open File dialog:



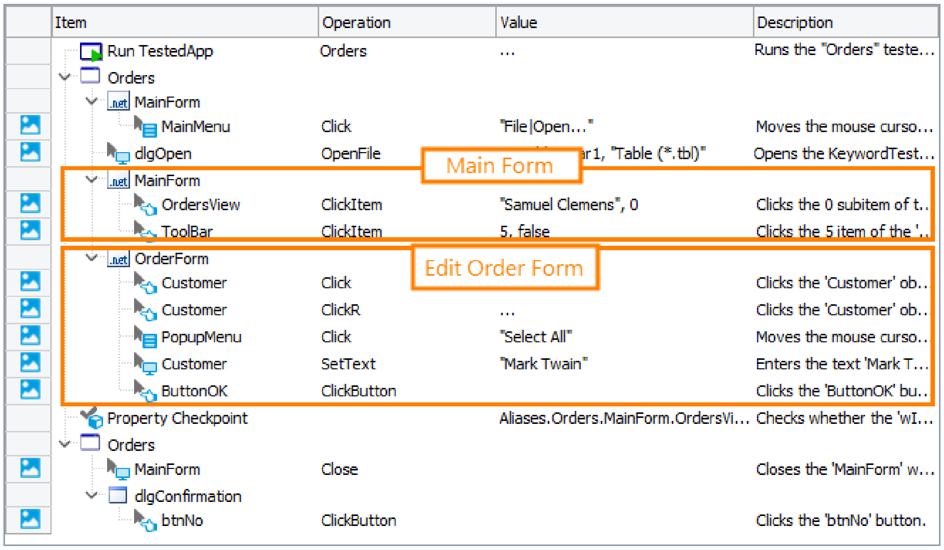
| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" test... |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open..." | Moves the mouse curs... |
| dlgOpen | OpenFile | Variables.Var1, "Table (*.tbl)" | Opens the KeywordTest.. |
| MainForm | | | |

In certain cases, TestComplete can record a sequence of operations that simulate actions you perform when working with the Open File dialog controls.

**Note:** It is recommended to type the full name of the file you want to open in the **File name** box of the Open file dialog instead of navigating to the file using the dialog controls. This approach lets you record a test that will be executed successfully regardless of the operating system, navigation bars and panels available in the dialog and of the path displayed in the dialog.

If your test contains a sequence of operations simulating actions over the Open File dialog, you can modify the test and manually replace those operations with the OpenFile method call.

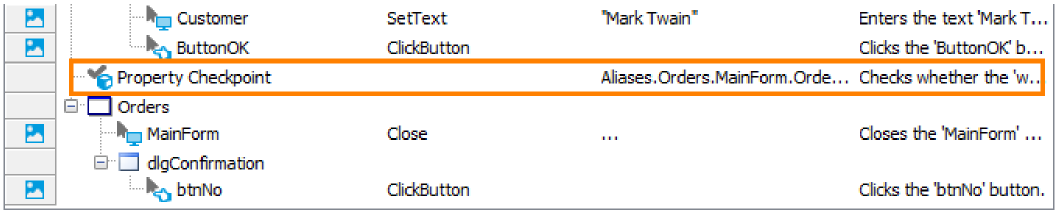
After that, there follow operations that simulate your actions with the application main window and the Order form:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" teste... |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open..." | Moves the mouse curso... |
| dlgOpen | OpenFile | r1, "Table (*.tbl)" | Opens the KeywordTest... |
| MainForm | | | |
| OrdersView | ClickItem | "Samuel Clemens", 0 | Clicks the 0 subitem of t.. |
| ToolBar | ClickItem | 5, false | Clicks the 5 item of the '.. |
| OrderForm | | | |
| Customer | Click | | Clicks the 'Customer' ob.. |
| Customer | ClickR | ... | Clicks the 'Customer' ob.. |
| PopupMenu | Click | "Select All" | Moves the mouse curso.. |
| Customer | SetText | "Mark Twain" | Enters the text 'Mark T... |
| ButtonOK | ClickButton | | Clicks the 'ButtonOK' bu.. |
| Property Checkpoint | | Aliases.Orders.MainForm.OrdersVi... | Checks whether the 'wI... |
| Orders | | | |
| MainForm | Close | | Closes the 'MainForm' w... |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

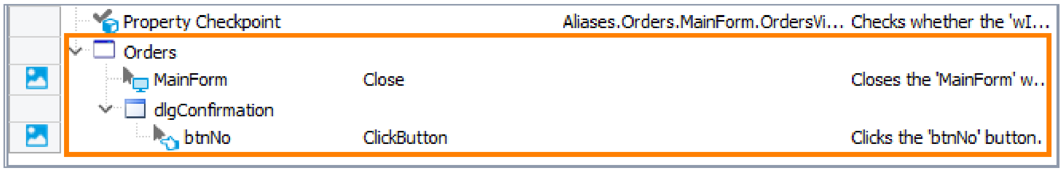
For more information on simulating mouse events, keyboard input and other actions from your scripts, see *Simulating User Actions* in TestComplete Help.

Then there is the comparison operation that we added during test recording:



| | | | |
|---|---|---|---|
| Customer | SetText | "Mark Twain" | Enters the text 'Mark T... |
| ButtonOK | ClickButton | | Clicks the 'ButtonOK' b... |
| Property Checkpoint | | Aliases.Orders.MainForm.Orde... | Checks whether the 'w.. |
| Orders | | | |
| MainForm | Close | ... | Closes the 'MainForm' ... |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

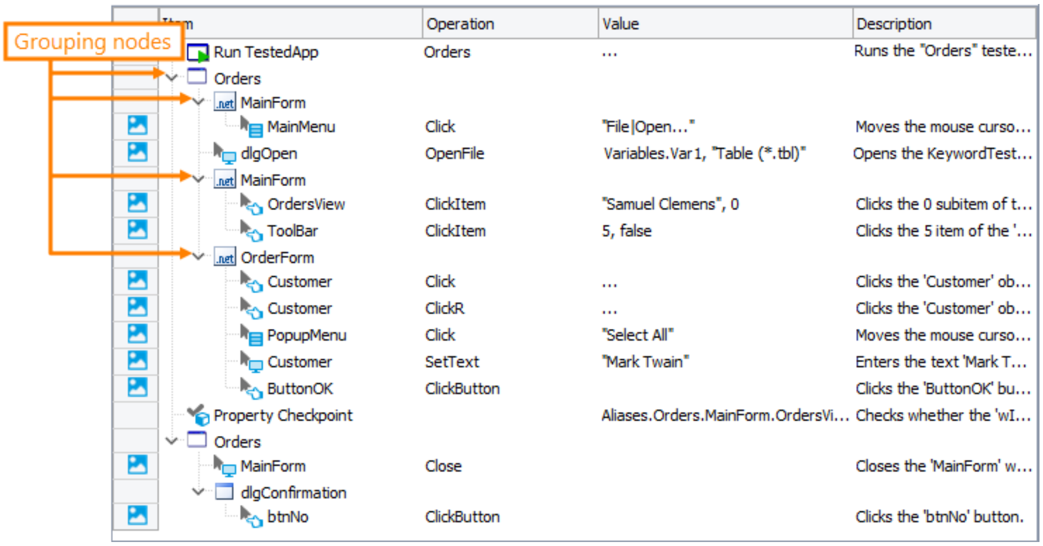
Finally, there is the operation that closes the Orders application and the operation that simulates the “No” button press in the message box.



| | | | |
|---|---|---|---|
| Property Checkpoint | | Aliases.Orders.MainForm.OrdersVi... | Checks whether the 'wI... |
| Orders | | | |
| MainForm | Close | | Closes the 'MainForm' w.. |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

As you can see, TestComplete automatically organizes the operations into groups that correspond to the processes and windows that you worked with. Grouping makes the test structure easier to understand and also provides some information on the object hierarchy that exists in the application under test.

We recorded user actions on one process (*Orders*). So, we have only one “process” group node. It contains all of the actions that you simulated on the process windows and controls. The actions that we performed on windows and controls of the *Orders* process are organized into a number of “window” grouping nodes:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" teste... |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open..." | Moves the mouse curso... |
| dlgOpen | OpenFile | Variables.Var1, "Table (*.tbl)" | Opens the KeywordTest... |
| MainForm | | | |
| OrdersView | ClickItem | "Samuel Clemens", 0 | Clicks the 0 subitem of t... |
| ToolBar | ClickItem | 5, false | Clicks the 5 item of the '... |
| OrderForm | | | |
| Customer | Click | ... | Clicks the 'Customer' ob... |
| Customer | ClickR | ... | Clicks the 'Customer' ob... |
| PopupMenu | Click | "Select All" | Moves the mouse curso... |
| Customer | SetText | "Mark Twain" | Enters the text 'Mark T... |
| ButtonOK | ClickButton | | Clicks the 'ButtonOK' bu... |
| Property Checkpoint | | Aliases.Orders.MainForm.OrdersVi... | Checks whether the 'wI... |
| Orders | | | |
| MainForm | Close | | Closes the 'MainForm' w... |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

You may notice that the names of the tested process and its windows and controls differ from the names that we saw in the Object Browser panel in one of the previous steps. For instance, in the Object Browser the tested process was named *Process("Orders")* while in the test it is called *Orders*; the main window was called *WinFormsObject("MainForm")* while in the test it is called *MainForm*, and so on.

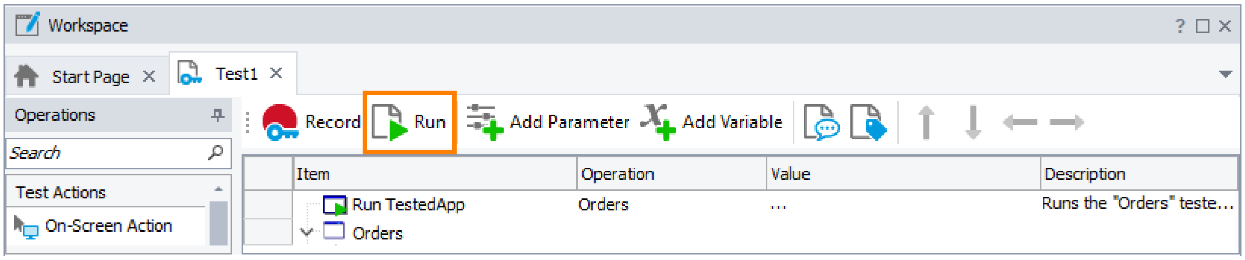
There is a logical reason for this: by default, TestComplete automatically generates and uses custom names for the objects that you worked with during the test recording. Generating and assigning custom names is called name mapping. TestComplete maps the names because the default names may be difficult to understand. It may be difficult to determine which window or control corresponds to a name. Using mapped names makes the test easier to understand and more stable. Along with mapping names, TestComplete also stores images of the mapped objects in the Name Mapping repository. This helps you understand which window or control one or another mapped object matches. For more information on mapping names, see *Name Mapping* in TestComplete Help.

# 7. Run the Recorded Test

Now we can run our simple test to see how TestComplete simulates user actions.

**Before running a recorded test, make sure it starts with the same initial conditions as the recording did.** For instance, the test almost always requires the tested application to be running. So, before simulating the user actions, you should launch the application. In our case, to launch our tested application, we use the *Run TestedApp* operation at the beginning of the test, so the test will launch it for us. Alternatively, you can run the tested application manually from TestComplete IDE.

To run the recorded test, simply click **Run** on the test editor toolbar:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" teste... |
| Orders | | | |

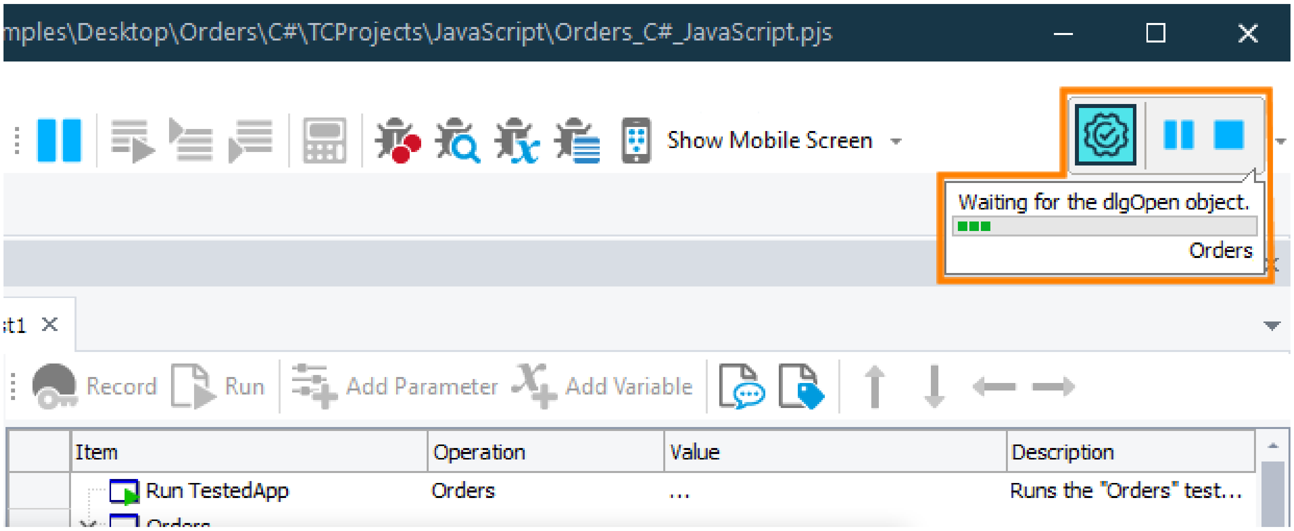
The test engine will minimize TestComplete window and start executing the test commands. In our case, the test will simply repeat your recorded actions.

> **Note:** Do not move the mouse or press keys during the test execution. Your actions may interfere with actions simulated by TestComplete and the test execution may go wrong.

After the test execution is over, TestComplete will restore its window and display the test results. In the next step, we will analyze them.

Some notes about the test run:

- The created tests are not compiled into an executable for test runs. You run the tests directly from TestComplete. To run tests on computers that do not have TestComplete installed, you can use a resource-friendly utility called TestExecute. You can also export script code (if you use it) to an external application and run it there. For more information on this, see *Connected and Self-Testing Applications* in TestComplete Help.
- During test execution, TestComplete displays an indicator in the top right corner of the screen:


mples\Desktop\Orders\C#\TCProjects\JavaScript\Orders_C#_JavaScript.pjs
Show Mobile Screen
t1 ✕
Record
Run
Add Parameter
Add Variable

Waiting for the dlgOpen object.

Orders

| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" test... |
| Orders | | | |

The indicator displays messages informing you about the simulated test actions.

- TestComplete executes the test commands until the test ends. You can stop the execution at any time by pressing ■ **Stop** on the Test Engine toolbar or select **Test > Stop** from TestComplete main menu.

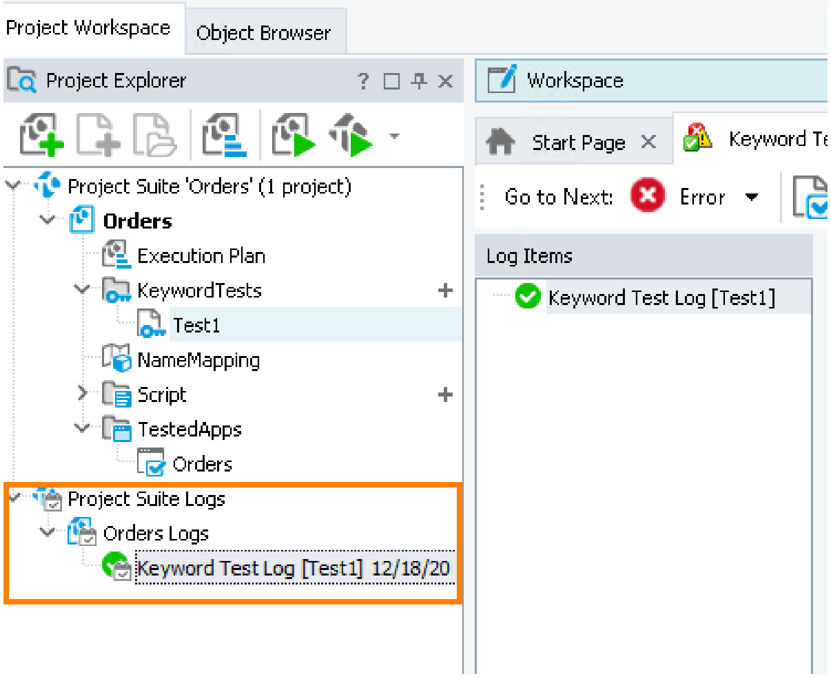
  You can pause the test execution by clicking ⏸ **Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check the test variables and objects using TestComplete **Watch List** or **Locals** panel or the **Evaluate** dialog (see *Debugging Tests* in TestComplete Help).

- To launch the test, we used the **Run Test** button on the test editor toolbar. This is only one of several possible ways to run the test. You can also run tests from the Project Explorer, or from another test. You can also use the Test Items page of the project editor to create a batch run.

For complete information on running tests in TestComplete, on project settings that affect the runs and on the test execution, see *Running Tests* in TestComplete Help.

## 8. Analyze Test Results

TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the **Project Explorer** panel under the **Project Suite Logs** > **Orders Log** node. This is the primary workspace for looking up the test history of the project and project suite. Each node corresponds to a test run. An image to the left of the node specifies whether the corresponding test run passed successfully:

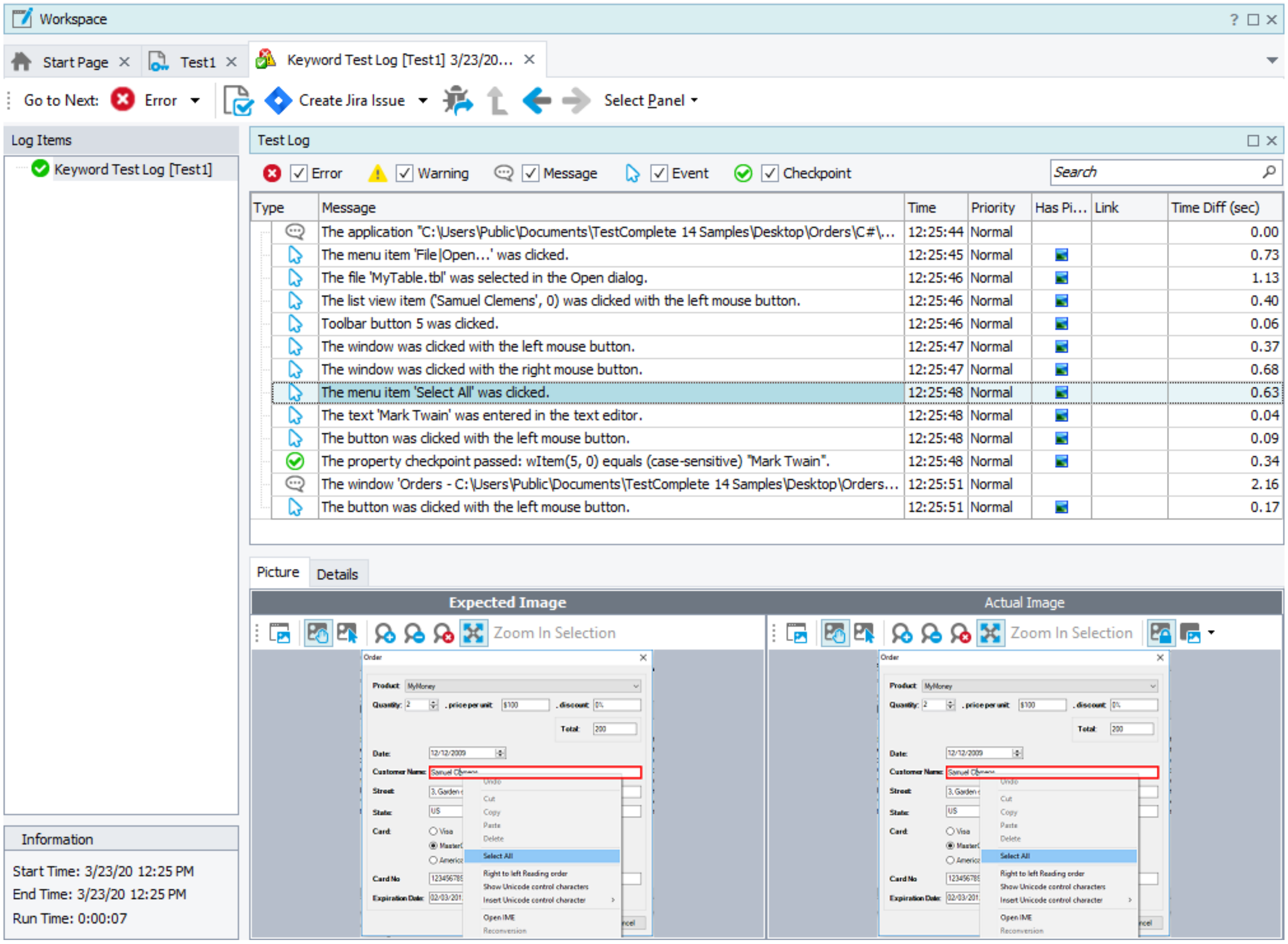


Note that TestComplete automatically adds nodes for the last results *after* the test execution is *over*. That is, the results are not displayed when the test is running (you can view intermediate results if you pause the test execution).

Since we have run only one test so far, we have only one log node in the Project Explorer. By default, TestComplete automatically opens the contents of this node in the **Workspace** panel. You can also view the log at any time. To do this, right-click the desired result in the Project Explorer panel and choose **Open** from the context menu.

> **Note:** By default, TestComplete stores all test results in log files. The number of the log files will grow with every test run, and this will result in the memory consumption increase. To reduce memory usage, you can delete files from the log manually or limit the number of log files to be kept.

In our example, the log is as follows –



| Type | Message | Time | Priority | Has Pi... | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
| 💬 | The application "C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders\C#\... | 12:25:44 | Normal | | | 0.00 |
| ↖ | The menu item 'File\|Open...' was clicked. | 12:25:45 | Normal | 🖼 | | 0.73 |
| ↖ | The file 'MyTable.tbl' was selected in the Open dialog. | 12:25:46 | Normal | 🖼 | | 1.13 |
| ↖ | The list view item ('Samuel Clemens', 0) was clicked with the left mouse button. | 12:25:46 | Normal | 🖼 | | 0.40 |
| ↖ | Toolbar button 5 was clicked. | 12:25:46 | Normal | 🖼 | | 0.06 |
| ↖ | The window was clicked with the left mouse button. | 12:25:47 | Normal | 🖼 | | 0.37 |
| ↖ | The window was clicked with the right mouse button. | 12:25:47 | Normal | 🖼 | | 0.68 |
| ↖ | The menu item 'Select All' was clicked. | 12:25:48 | Normal | 🖼 | | 0.63 |
| ↖ | The text 'Mark Twain' was entered in the text editor. | 12:25:48 | Normal | 🖼 | | 0.04 |
| ↖ | The button was clicked with the left mouse button. | 12:25:48 | Normal | 🖼 | | 0.09 |
| ✓ | The property checkpoint passed: wItem(5, 0) equals (case-sensitive) "Mark Twain". | 12:25:48 | Normal | 🖼 | | 0.34 |
| 💬 | The window 'Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders... | 12:25:51 | Normal | | | 2.16 |
| ↖ | The button was clicked with the left mouse button. | 12:25:51 | Normal | 🖼 | | 0.17 |

Information

Start Time: 3/23/20 12:25 PM
End Time: 3/23/20 12:25 PM
Run Time: 0:00:07

The log window shows the results of one test run at a time. On the left side of the window, there is a tree-like structure of the tests that were executed during the run; the node of each of these tests can be selected to view their results. For our example, we have run only one test, so in our case this tree only contains one node. The node's icon indicates whether the test passed successfully or failed.

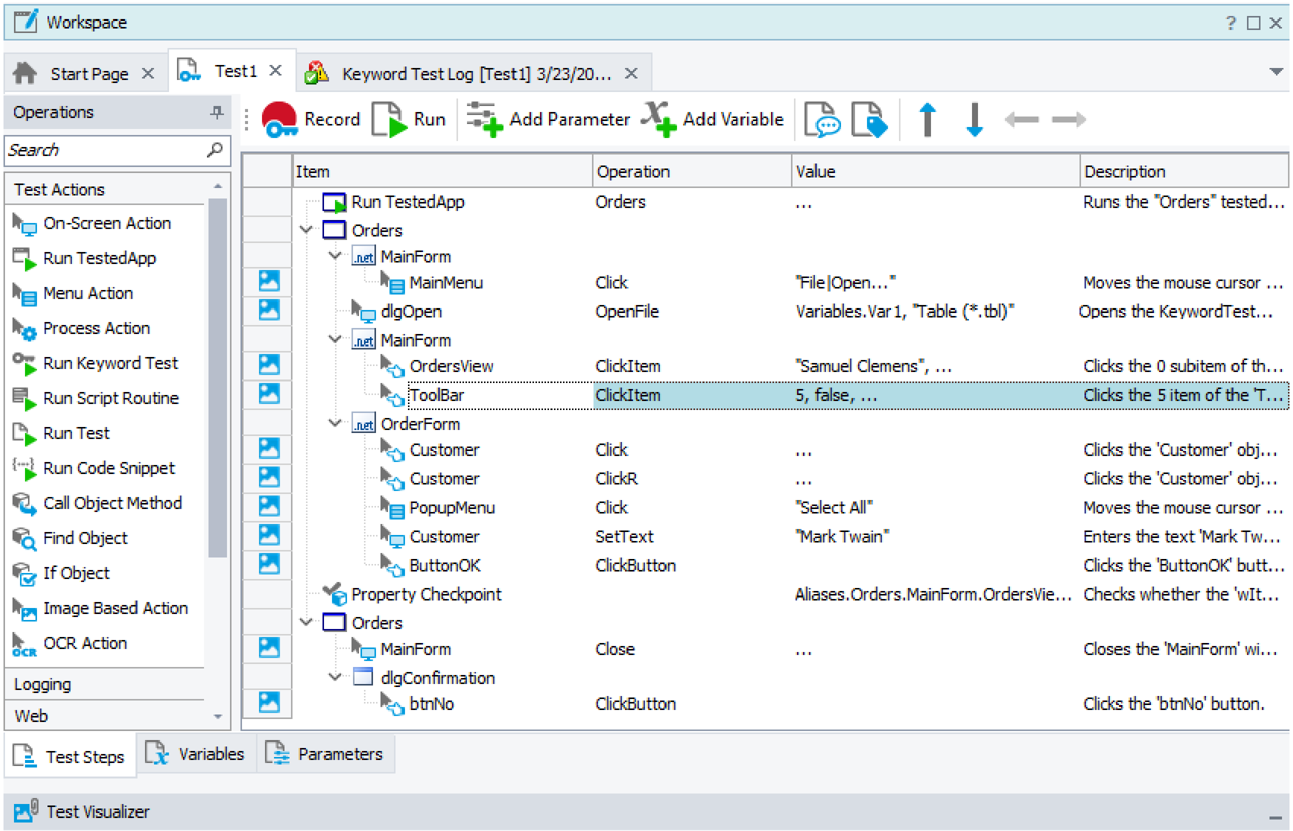
The test log contains error, warning, informative and other types of messages. The icon on the left indicates the message type. Using the check boxes at the top of the message list you can hide or view messages by type.

For each message, the log also shows the time that each action was performed. You can see it in the **Time** column.

TestComplete may post additional text and images along with the message. To view them, simply select the desired message in the log and look in the **Details** and **Picture** panes that are below the message list. For instance, on the image above the Picture pane displays the screenshots associated with "The menu item 'Orders|Edit order...' was clicked" message.

The **Picture** panel displays the images that show the expected and actual application state before executing the selected test command ("Expected" is the image that was captured for the command during test recording, "actual" means the image that was captured during test run). You can click **View Comparison Result** to see the difference between the images. This simplifies the search for errors that may occur in your test. For more information, see topics of the *Test Visualizer* section in TestComplete Help.

To view a test operation that posted a message to the log, double-click the desired message in the log. TestComplete will open the keyword test in the editor and highlight the appropriate operation. For instance, if you double-click the “Toolbar button 5 was clicked” message in the log, TestComplete will highlight the keyword test operation that performed this action:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run TestedApp | Orders | ... | Runs the "Orders" tested... |
| Orders | | | |
| MainForm | | | |
| MainMenu | Click | "File\|Open..." | Moves the mouse cursor ... |
| dlgOpen | OpenFile | Variables.Var1, "Table (*.tbl)" | Opens the KeywordTest... |
| MainForm | | | |
| OrdersView | ClickItem | "Samuel Clemens", ... | Clicks the 0 subitem of th... |
| ToolBar | ClickItem | 5, false, ... | Clicks the 5 item of the 'T... |
| OrderForm | | | |
| Customer | Click | ... | Clicks the 'Customer' obj... |
| Customer | ClickR | ... | Clicks the 'Customer' obj... |
| PopupMenu | Click | "Select All" | Moves the mouse cursor ... |
| Customer | SetText | "Mark Twain" | Enters the text 'Mark Tw... |
| ButtonOK | ClickButton | | Clicks the 'ButtonOK' butt... |
| Property Checkpoint | | Aliases.Orders.MainForm.OrdersVie... | Checks whether the 'wIt... |
| Orders | | | |
| MainForm | Close | ... | Closes the 'MainForm' wi... |
| dlgConfirmation | | | |
| btnNo | ClickButton | | Clicks the 'btnNo' button. |

For detailed information on the test log panels, on posting messages to the log and on working with the results, see *Test Results* section in TestComplete Help.

**Note:** The log that we described is typical for TestComplete keyword tests and scripts. Tests of other types may form a log of a different structure. For detailed information about these logs, see the description of the appropriate project item, or simply click within the log page and press F1.

## Resolving Errors

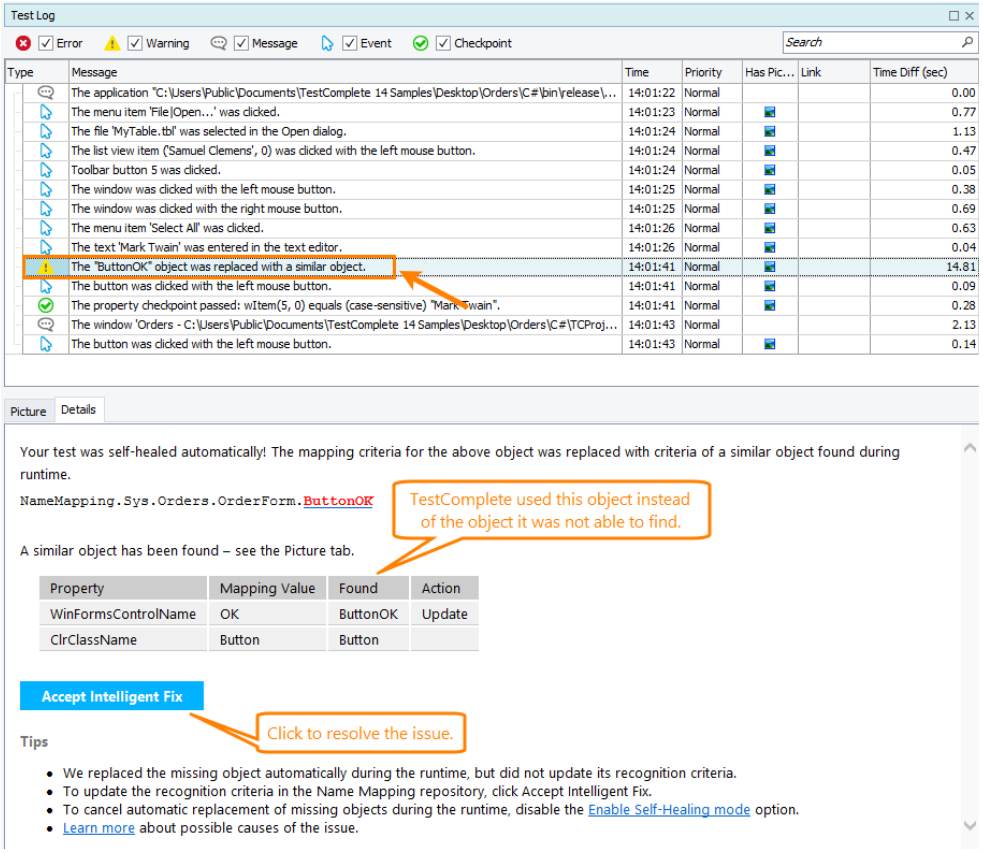
Your test may fail. There can be several possible reasons for this. For example, developers could change the application behavior, the recognition attributes of windows and controls could change and make the test engine fail to find the needed objects, a third-party application may overlap windows of your application and make the test engine fail to simulate actions on it, and so on.

One of the most typical reasons that novice users face is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance:

- If your tested application had been running before you recorded a test, it must also be running before you run the test.

- If test actions were performed on a particular window of the application, you should open the window when running the test.
- If you edited any data in the application, and then saved it, you need to revert the changes.

If the tested application has changed and the identification properties that your test uses to find tested objects are no longer valid, TestComplete will try to detect the changes to find the missing objects. If your test log reports the “*Object <Object_Name> was replaced with a similar object*” warning, this means that TestComplete was not able to find the needed tested object during the test run and used a similar object instead.



| Type | Message | Time | Priority | Has Pic... | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
| 🗨 | The application "C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders\C#\bin\release\... | 14:01:22 | Normal | | | 0.00 |
| ↖ | The menu item 'File\|Open...' was clicked. | 14:01:23 | Normal | 🖼 | | 0.77 |
| ↖ | The file 'MyTable.tbl' was selected in the Open dialog. | 14:01:24 | Normal | 🖼 | | 1.13 |
| ↖ | The list view item ('Samuel Clemens', 0) was clicked with the left mouse button. | 14:01:24 | Normal | 🖼 | | 0.47 |
| ↖ | Toolbar button 5 was clicked. | 14:01:24 | Normal | 🖼 | | 0.05 |
| ↖ | The window was clicked with the left mouse button. | 14:01:25 | Normal | 🖼 | | 0.38 |
| ↖ | The window was clicked with the right mouse button. | 14:01:25 | Normal | 🖼 | | 0.69 |
| ↖ | The menu item 'Select All' was clicked. | 14:01:26 | Normal | 🖼 | | 0.63 |
| ↖ | The text 'Mark Twain' was entered in the text editor. | 14:01:26 | Normal | 🖼 | | 0.04 |
| ⚠ | The "ButtonOK" object was replaced with a similar object. | 14:01:41 | Normal | 🖼 | | 14.81 |
| ↖ | The button was clicked with the left mouse button. | 14:01:41 | Normal | 🖼 | | 0.09 |
| ✓ | The property checkpoint passed: wItem(5, 0) equals (case-sensitive) "Mark Twain". | 14:01:41 | Normal | 🖼 | | 0.28 |
| 🗨 | The window 'Orders - C:\Users\Public\Documents\TestComplete 14 Samples\Desktop\Orders\C#\TCProj... | 14:01:43 | Normal | | | 2.13 |
| ↖ | The button was clicked with the left mouse button. | 14:01:43 | Normal | 🖼 | | 0.14 |

Your test was self-healed automatically! The mapping criteria for the above object was replaced with criteria of a similar object found during runtime.

`NameMapping.Sys.Orders.OrderForm.ButtonOK`

TestComplete used this object instead of the object it was not able to find.

A similar object has been found – see the Picture tab.

| Property | Mapping Value | Found | Action |
|---|---|---|---|
| WinFormsControlName | OK | ButtonOK | Update |
| ClrClassName | Button | Button | |

Click to resolve the issue.

Tips

- We replaced the missing object automatically during the runtime, but did not update its recognition criteria.
- To update the recognition criteria in the Name Mapping repository, click Accept Intelligent Fix.
- To cancel automatic replacement of missing objects during the runtime, disable the Enable Self-Healing mode option.
- Learn more about possible causes of the issue.

Do not ignore this warning. Examine it to learn about the missing and the replaced objects and update the object’s identification properties to resolve the issue.

For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* in TestComplete Help.

# Testing Web Applications

TestComplete supports functional testing of web applications working in Internet Explorer, Edge, Chrome, Firefox, or in a built-in WebBrowser or Chromium. TestComplete supports testing of any web application and provides enhanced support for HTML5, AJAX, ASP.NET, Flash, Flex, AIR, and Silverlight technologies.

This section explains how to create a test project in TestComplete, record and play back a simple web test, and analyze the results. It assumes that you are familiar with general principles of automated testing and have minimal knowledge of the TestComplete IDE.

> ❗ If you are a novice user, we recommend that you read an introduction to automated testing described above.

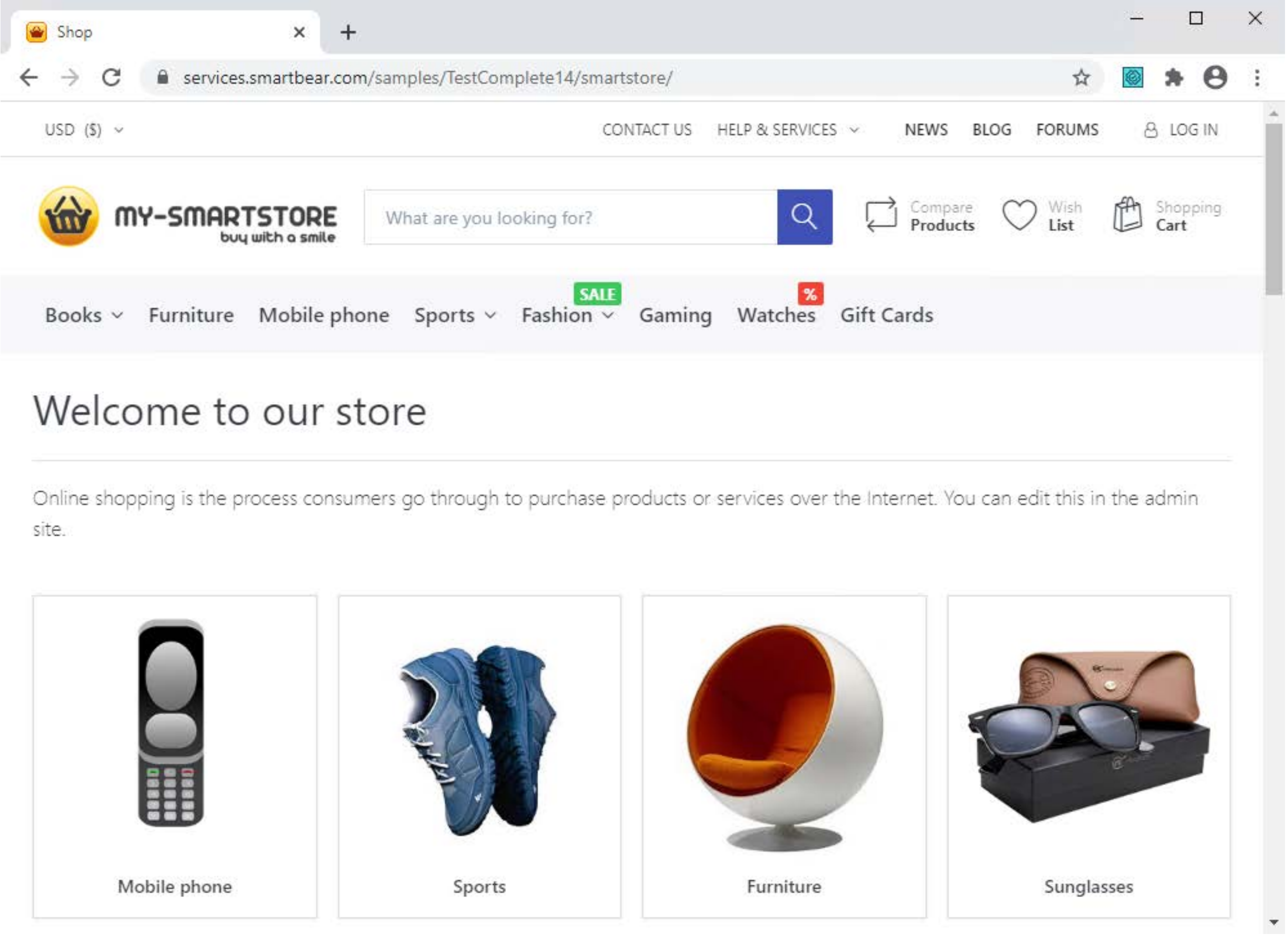
The test will emulate user actions over a web page and verify some data. Verification commands will be created during test recording.

## About the Tested Web Page

In our explanations we will use the sample *SmartStore* application that can be found on our web site:

https://services.smartbear.com/samples/TestComplete14/smartstore/

This application is a sample online store where you can browse items, add them to a wish list and to a shopping cart, compare items. You can create an account, login, and create mock-up orders:



Welcome to our store

Online shopping is the process consumers go through to purchase products or services over the Internet. You can edit this in the admin site.

**Note:** In this tutorial, we will use the Google Chrome web browser. All images in the tutorial are made with our tested application opened in Chrome. You can use any supported web browser you like.

## 1. Prepare Your Web Browser

To record and play back web tests in TestComplete, you need to configure the web browser in a special way. Also, it is recommended that you eliminate browser-specific behavior to make the cross-browser testing easier. For more information on configuring your browser, refer to the *Preparing Web Browsers* section in TestComplete Help.

After you have configured the browser settings as it is described in the section, you can create tests.

## 2. Plan the Test

The sample SmartStore application in an online store. Suppose, you need to test whether the application adds items to the Shopping Cart correctly. In this case, you need to define the following:

- **Test purpose**: The test must check whether the selected item is added to the Shopping Cart correctly. That is, the Shopping Cart shows the added item.
- **Testing steps**: The test must add an item to the Shopping Cart, and then verify that the Shopping Cart contains this item. We will record a test simulating user actions over the application.
- **Checking and logging the test result**: If the item has been added correctly, it must be in the Shopping Cart. To check this, the test will compare the data the Shopping Cart shows with the expected value. We will add a special comparison command to the test for this purpose. This command will post the comparison results to the test log, so we will see whether the verification has failed or passed.

For more information on planning tests with TestComplete, see *Planning Tests* in TestComplete Help.

## 3. About Test Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually -** You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.
- **Record tests** - Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them to script lines or keyword test operations. You create a test visually, and, in one sense, you **record** the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

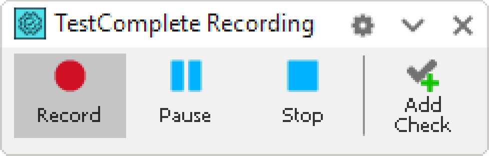
In this tutorial, we will demonstrate how to **record** tests with TestComplete.

The recording includes three steps:

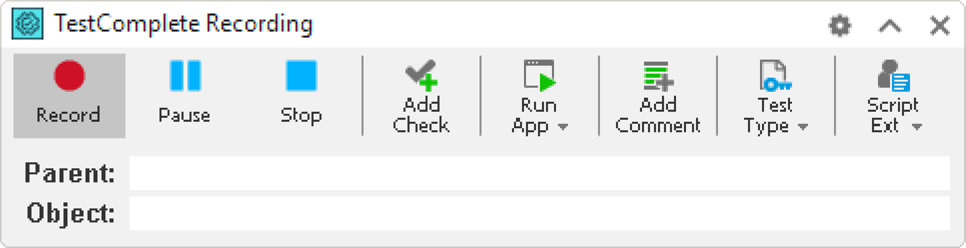
1. You start recording. You can do this by selecting **Test > Record > Record Keyword Test** or **Test > Record > Record Script** from the TestComplete main menu or from the Test Engine toolbar. You can also start recording by clicking ● **Record Test** on the Start Page.

   You can record tests of various kinds: keyword tests, scripts, and low-level procedures. The menu item that you use to start the recording defines the main recorded test: keyword test or script code. Other tests will be recorded after the recording starts. The main recorded test will contain special commands that will run these tests.

   TestComplete will switch to the recording mode and display the Recording toolbar on the screen. By default, the toolbar is collapsed and shows only the most commonly used commands you may need during the recording:



You can click the ⌄arrow button to expand the Recording toolbar and view all its buttons:



You use the toolbar to perform additional actions during the recording, pause or stop recording, and change the type of the recorded test (keyword test, script code, or low-level procedure).

2. After starting the recording, perform the desired test actions: launch the tested application (if needed), work with it by clicking command buttons, selecting menu items, typing text, and so on.

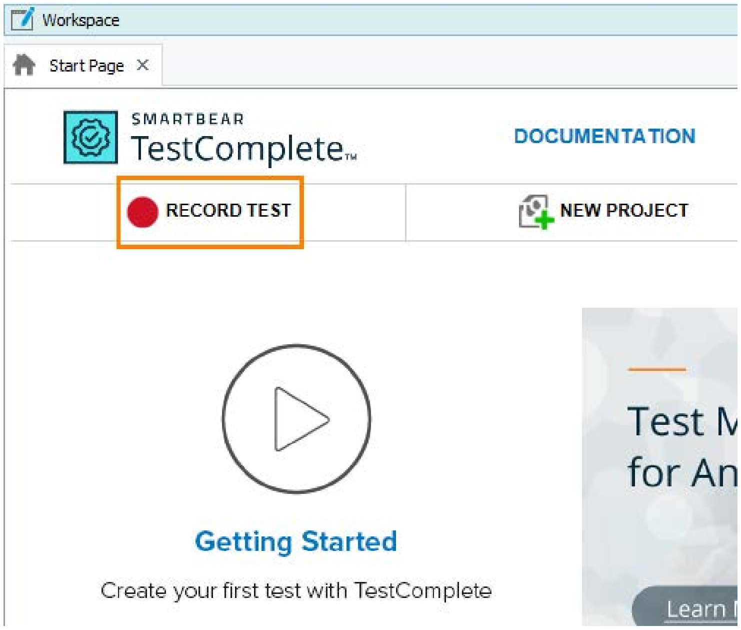
3. After all the test actions are over, stop the recording by selecting 🟦 **Stop** from the Recording toolbar.

For complete information on test recording, see *Recording in TestComplete* in TestComplete Help.

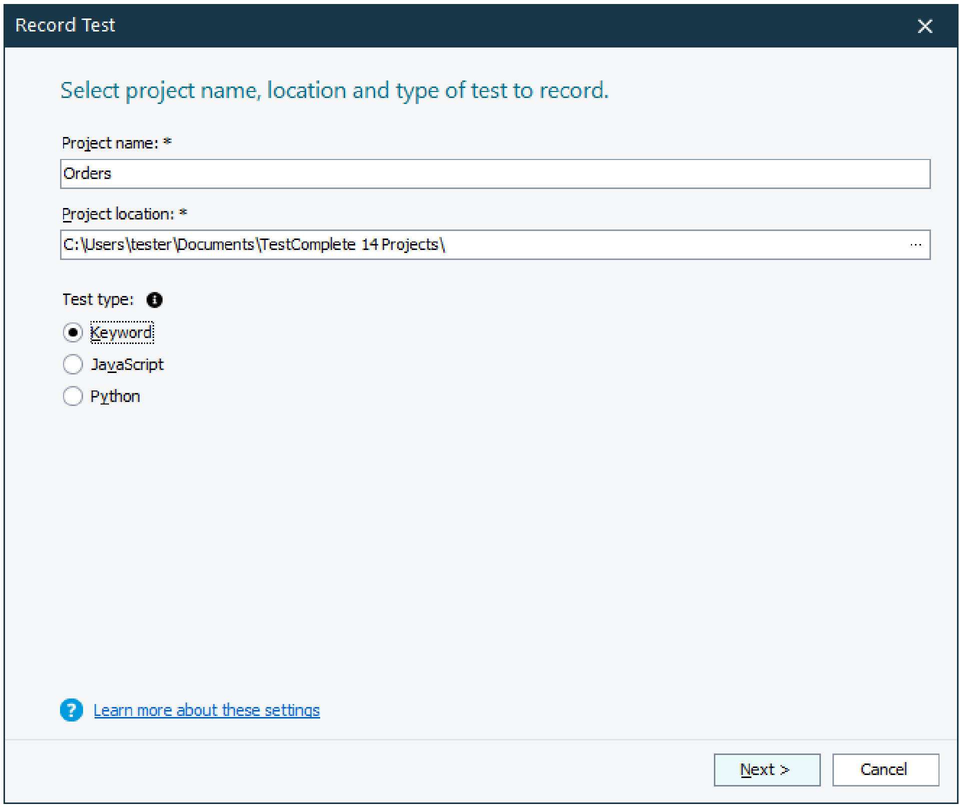
## 4. Start Test Recording

1. If a project or a project suite is open in TestComplete, close it. To do this, select **File > Close** from the TestComplete main menu.

2. Switch to the TestComplete **Start** page. If the page is hidden, select **Start Page** from the TestComplete **Help** menu.



3. On the Start page, click ● **Record Test**. TestComplete will show the **Record Test** wizard:



Record Test

Select project name, location and type of test to record.

Project name: * Orders

Project location: * C:\Users\tester\Documents\TestComplete 14 Projects\

Test type:
- ● Keyword
- ○ JavaScript
- ○ Python

javascript:showImageEx(%22_images/tutorials/first-test/record-test-project.gif%22, %22Getting Started With TestComplete: Record Test wizard%22)

4. On the first page of the wizard, you can specify the project name, location and test type:

   In the **Project name** text box, enter Orders.

   Leave the **Location** to its default value.

   Select a test type. You can create either a keyword test, or a JavaScript or Python script test:

   - A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.
   - JavaScript and Python tests are script functions with instructions simulating user actions.

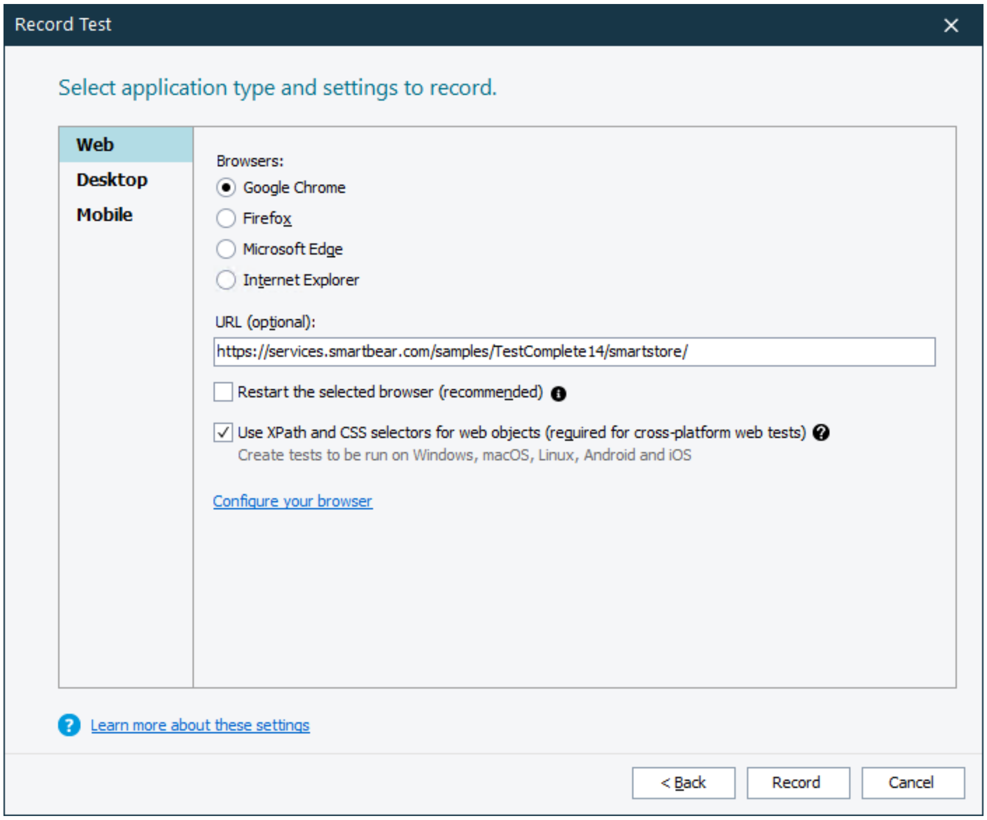
   In this tutorial we will show how to create a keyword test. Click **Keyword**.

5. Click **Next** to continue.

We will continue working with the wizard to add our tested application to the project.

## 5. Define a Tested Web Page

1. The wizard shows the second page where you can choose your tested application:



Select application type and settings to record.

Browsers:

● Google Chrome
○ Firefox
○ Microsoft Edge
○ Internet Explorer

URL (optional): https://services.smartbear.com/samples/TestComplete14/smartstore/

□ Restart the selected browser (recommended)

☑ Use XPath and CSS selectors for web objects (required for cross-platform web tests)
Create tests to be run on Windows, macOS, Linux, Android and iOS

Configure your browser

Learn more about these settings

2. As we are going to test the SmartStore application that is located on the web page, click **Web**.

3. Select one of the supported web browsers:

   - Internet Explorer
   - Firefox
   - Google Chrome
   - Microsoft Edge

   **Note:** TestComplete supports only Edge Chromium. If you have an earlier version of Edge installed, recording in it will be not available.

   In this tutorial, we will use **Google Chrome**. You can use any other supported web browser.

4. In the **URL** text box, enter the following URL:

   *https://services.smartbear.com/samples/TestComplete14/smartstore/*

5. In order for the test to run successfully, the tested application state at the beginning of the test run must be the same as in the beginning of the test recording. If your tested application is running when you start test recording, its state may differ from its state during subsequent test runs (for example, some data may be loaded or dialogs may be open). That is why we recommend that you close all the running instances of your web browsers before starting test recording.

If any of the supported web browsers is running, the wizard will show the **Restart the selected browser** check box. Select the check box to command the wizard to close the web browser you have selected and to restart it automatically when recording starts.

(If any other web browser is running in your system, it will continue running.)

6. The **Use XPath and CSS selectors for web objects** check box enables identifying web objects by using XPath expressions and CSS selectors.

   This is compulsory for **cross-platform web tests**. These are tests that can be run in remote environments in various web browsers and platforms, for example, Safari and MacOS. Creating such tests is beyond the scope of this tutorial.

   However, we recommend that you **leave the check box selected**.

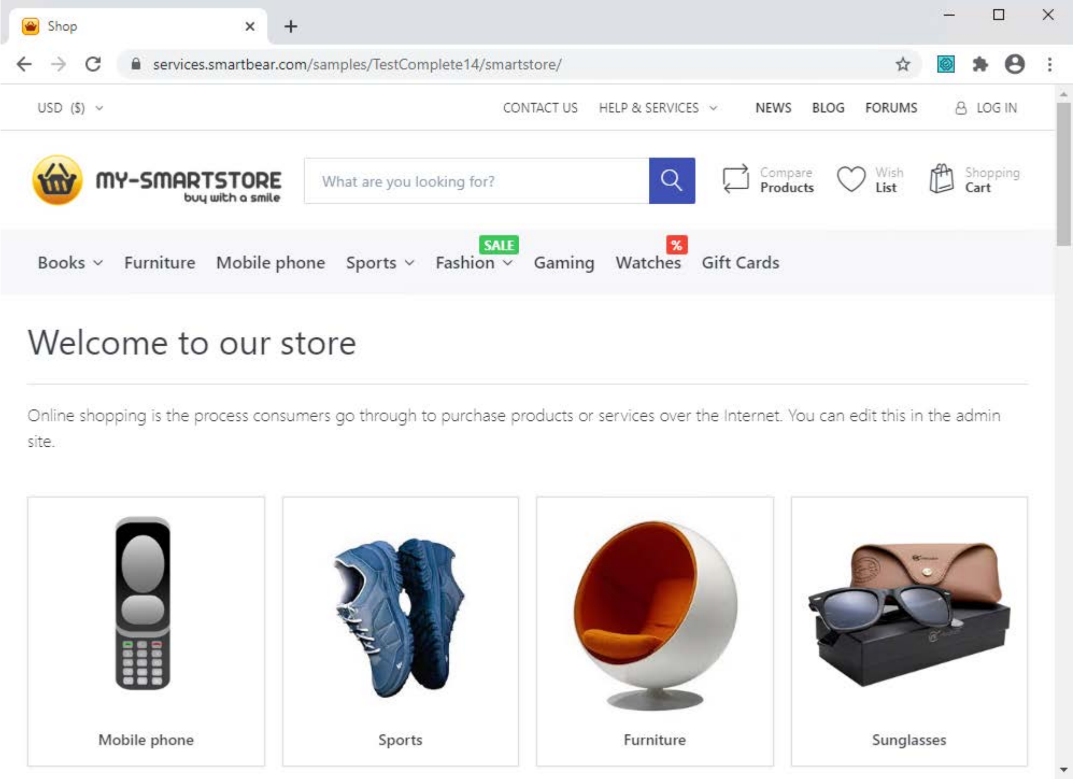
7. Click **Record** to complete the project creation and start recording.

# 6. Record a Test

**Note:** Do not switch to the TestComplete help during the recording. The recording engine traces and records all user actions, so the recorded test will contain commands that simulate “switching”.

To see the instructions, you can print them before starting the record. Or, if you have two monitors, you can move the TestComplete help system window to the other monitor.

1. Wait until the selected web browser starts and navigates to the main page of the SmartStore application:



services.smartbear.com/samples/TestComplete14/smartstore/

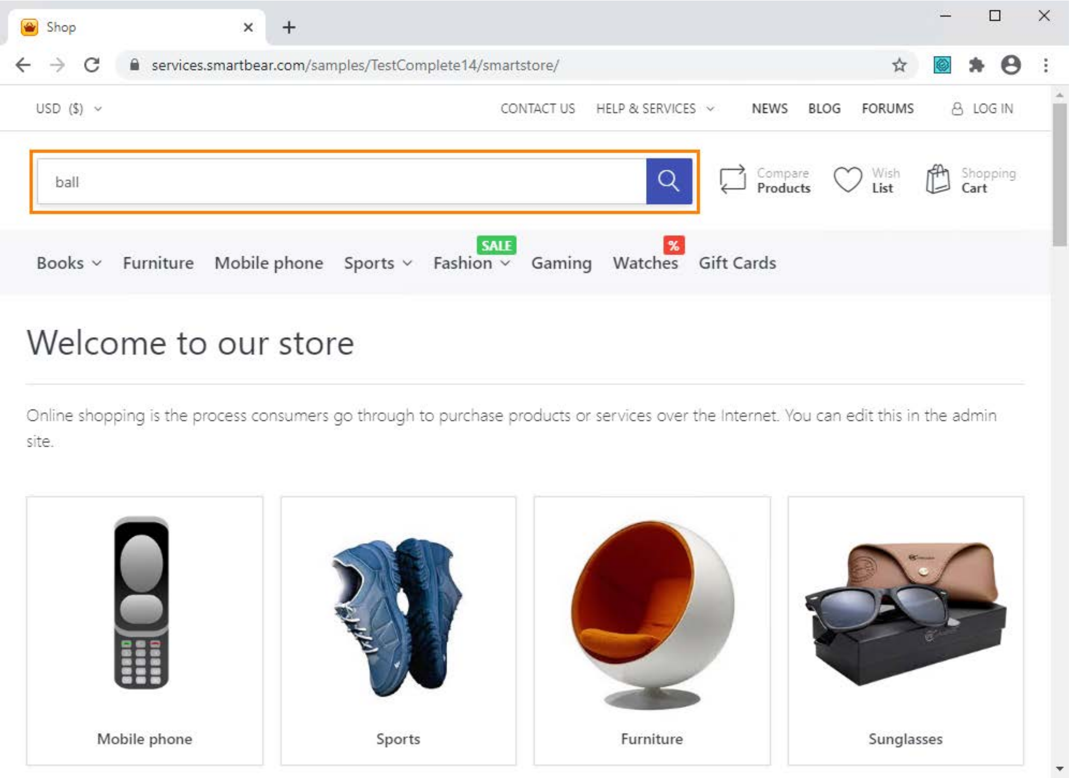
Welcome to our store

Online shopping is the process consumers go through to purchase products or services over the Internet. You can edit this in the admin site.

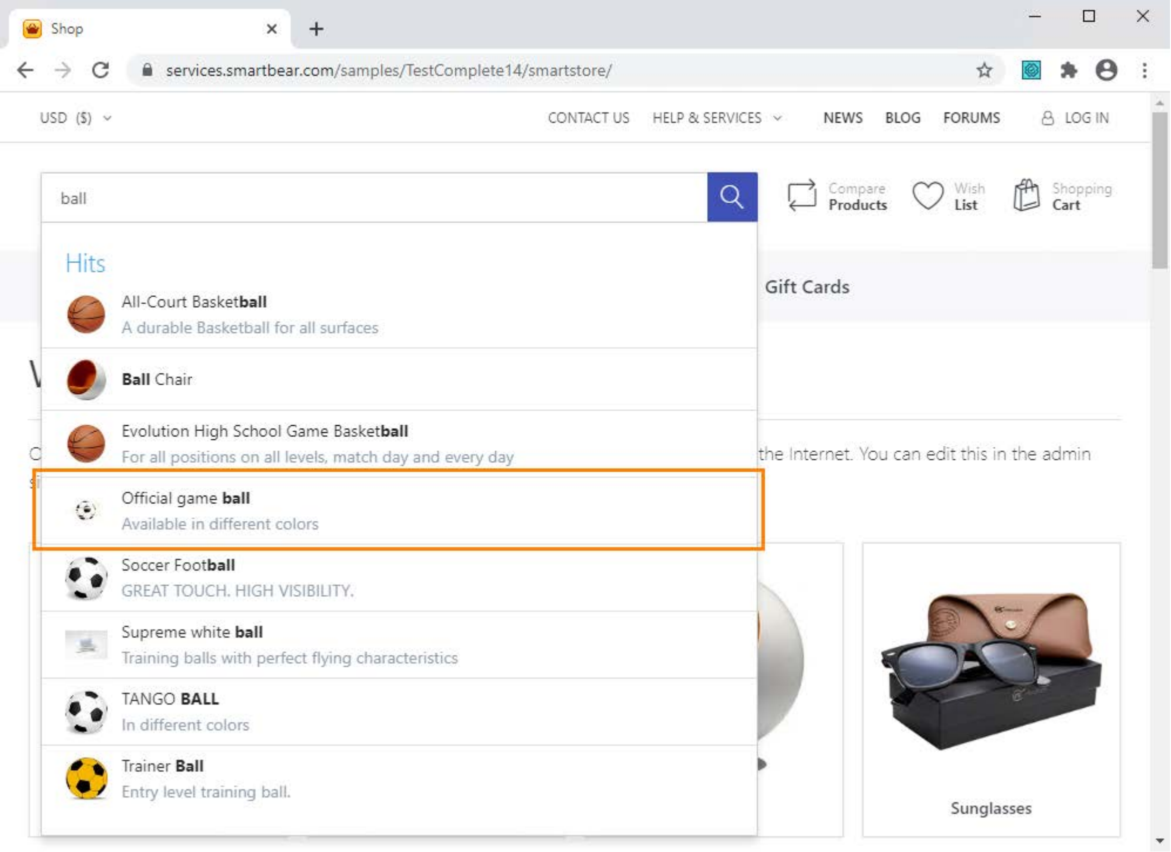
If the browser does not, you can launch it and open the needed web page manually.

TestComplete records the start of the web browser using a special test command. You will see this command later when we analyze the recorded test.

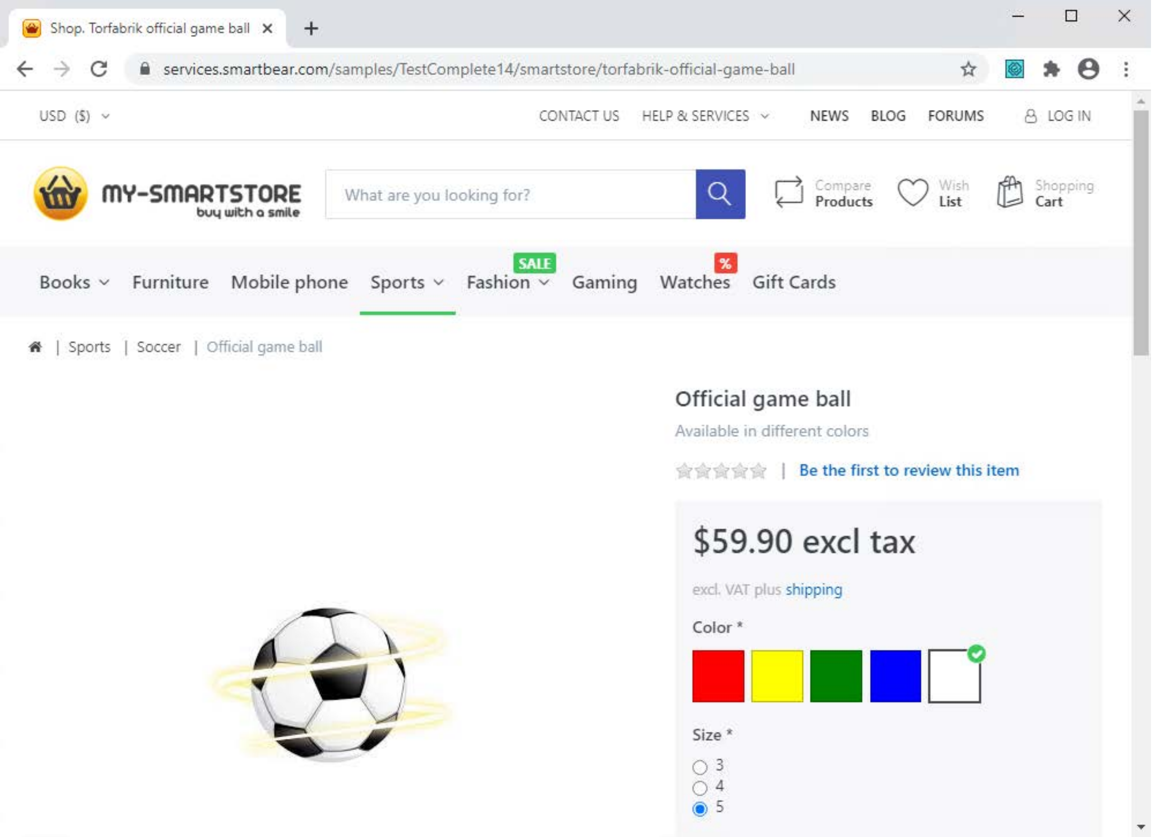
**2.** Click in the **Search** text box and type *ball*:



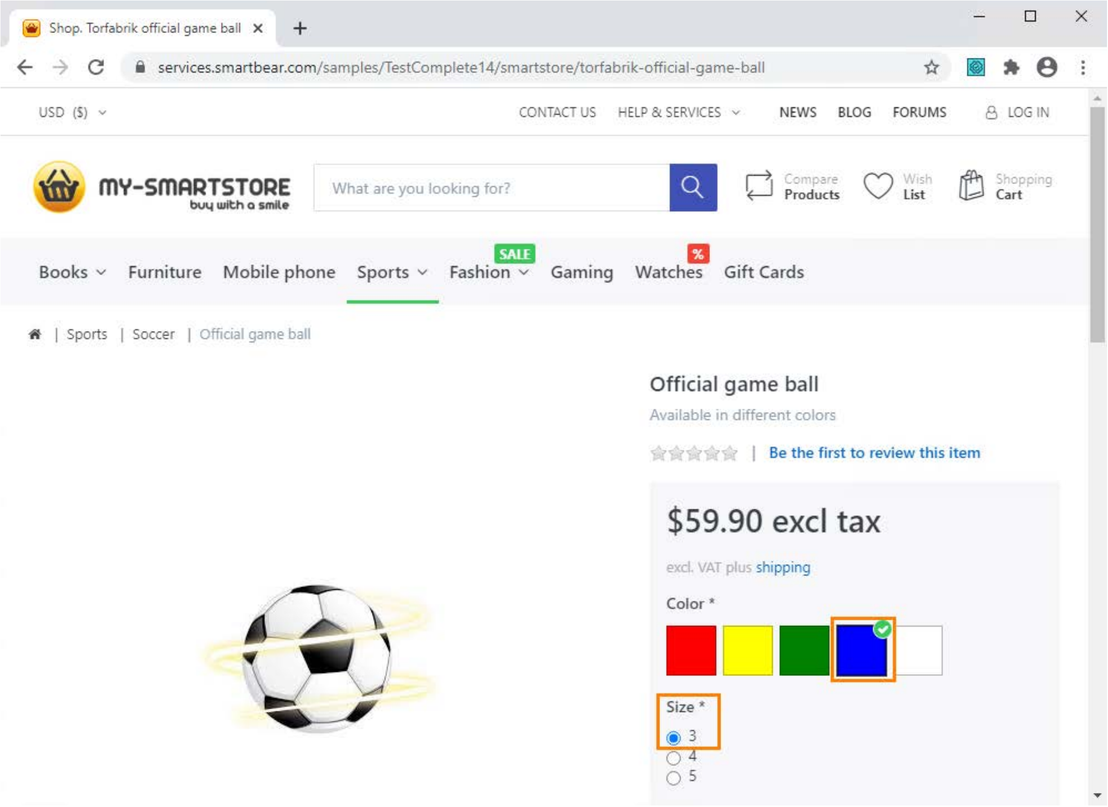
3. In the search results, find the **Official game ball** item and click it:


Shop
services.smartbear.com/samples/TestComplete14/smartstore/
USD ($)
CONTACT US
HELP & SERVICES
NEWS
BLOG
FORUMS
LOG IN
ball
Compare Products
Wish List
Shopping Cart
Gift Cards
Hits
All-Court Basketball
A durable Basketball for all surfaces
Ball Chair
Evolution High School Game Basketball
For all positions on all levels, match day and every day
Official game ball
Available in different colors
Soccer Football
GREAT TOUCH. HIGH VISIBILITY.
Supreme white ball
Training balls with perfect flying characteristics
TANGO BALL
In different colors
Trainer Ball
Entry level training ball.
the Internet. You can edit this in the admin
Sunglasses

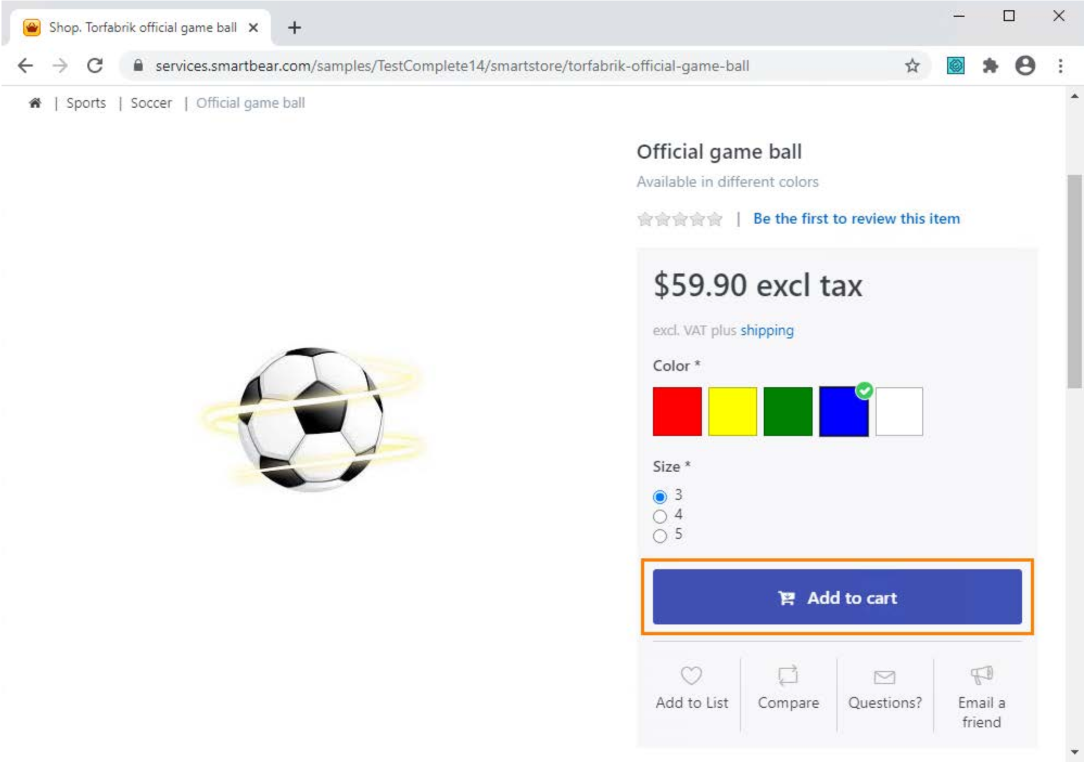
4. The application will open the selected item page:



5. On the page, select the item color: **blue** and the item size: **3**:

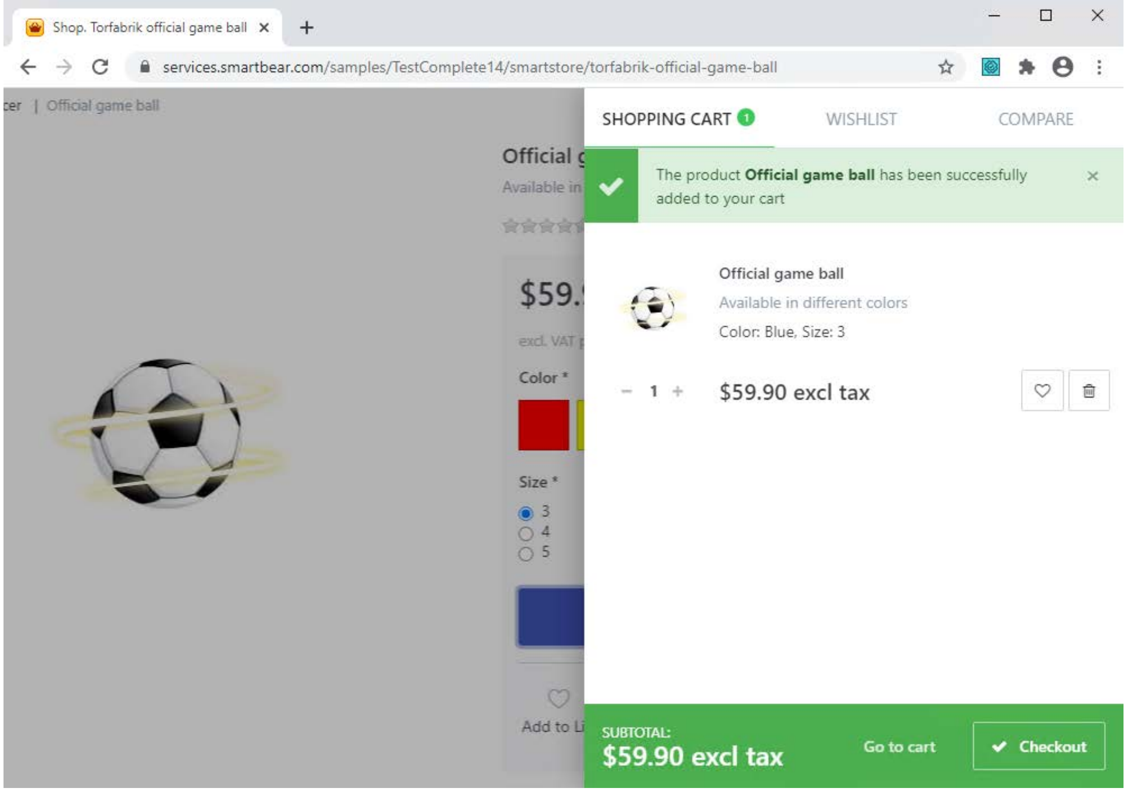


**6.** Click **Add to cart**:



Shop. Torfabrik official game ball
services.smartbear.com/samples/TestComplete14/smartstore/torfabrik-official-game-ball
Sports | Soccer | Official game ball
Official game ball
Available in different colors
Be the first to review this item
$59.90 excl tax
excl. VAT plus shipping
Color *
Size *
● 3
○ 4
○ 5
Add to cart
Add to List
Compare
Questions?
Email a friend

7. The application will show you your **Shopping Cart** with the item added to it:


Shop. Torfabrik official game ball
services.smartbear.com/samples/TestComplete14/smartstore/torfabrik-official-game-ball
cer | Official game ball
Official g
Available in
$59.
excl. VAT
Color *
Size *
3
4
5
Add to Li
SHOPPING CART 1
WISHLIST
COMPARE

The product **Official game ball** has been successfully added to your cart

Official game ball
Available in different colors
Color: Blue, Size: 3

1 $59.90 excl tax

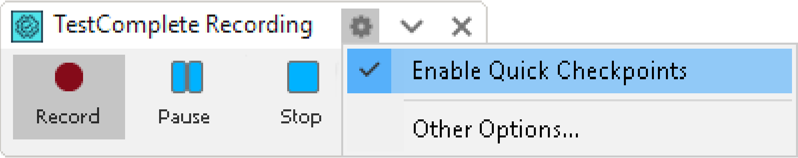
SUBTOTAL:
$59.90 excl tax

8. Insert a comparison command into our test. It will verify that the Shopping Cart contains an item with the selected features - *color: blue and size: 3*.

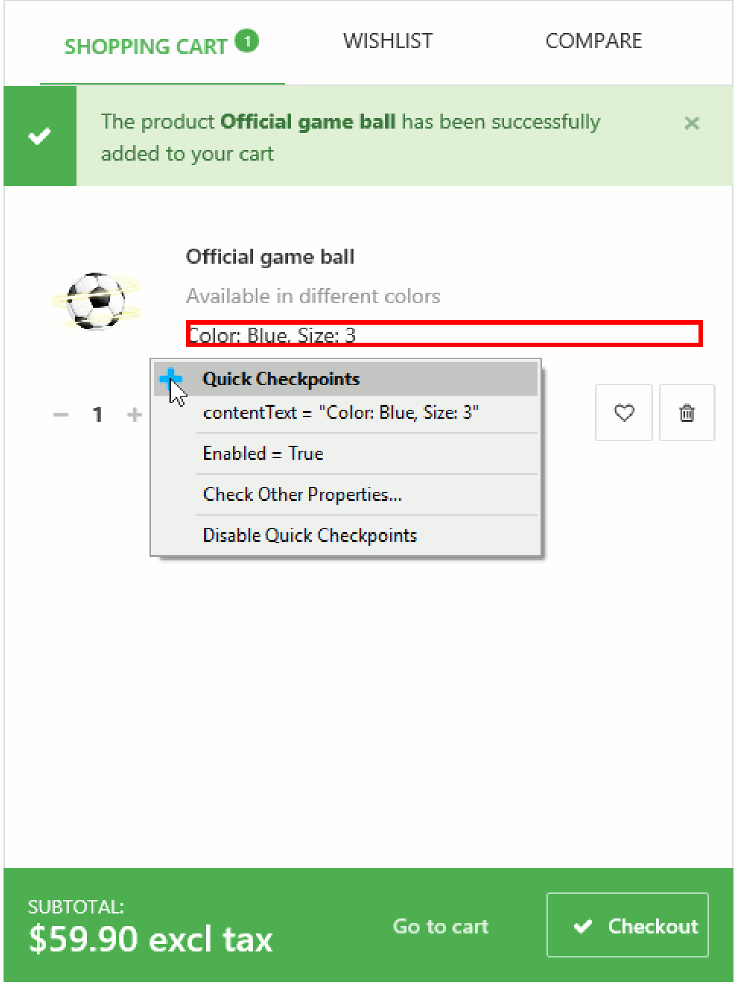
We call the comparison commands **checkpoints**. TestComplete offers various types of checkpoints that are suitable for verifying different types of data (see *Checkpoints* in TestComplete Help.). One of the most frequently used checkpoints is a **Property checkpoint**. It is used to check data of applications controls. We will use this checkpoint in our tutorial.

To create a property checkpoint, you can use the **Create Checkpoint** wizard, or you can create a **Quick Checkpoint**. In this tutorial, we will demonstrate how to create Quick Checkpoints:

- Click ⚙ on the **Recording** toolbar and make sure that Quick Checkpoints are enabled:



- Move the mouse pointer to the **Color: Blue, Size:3** line. TestComplete will highlight it with a red frame. Wait until the ✚ icon becomes and move the mouse pointer to the icon. TestComplete will show the most commonly used properties for which you can create Quick Checkpoints:



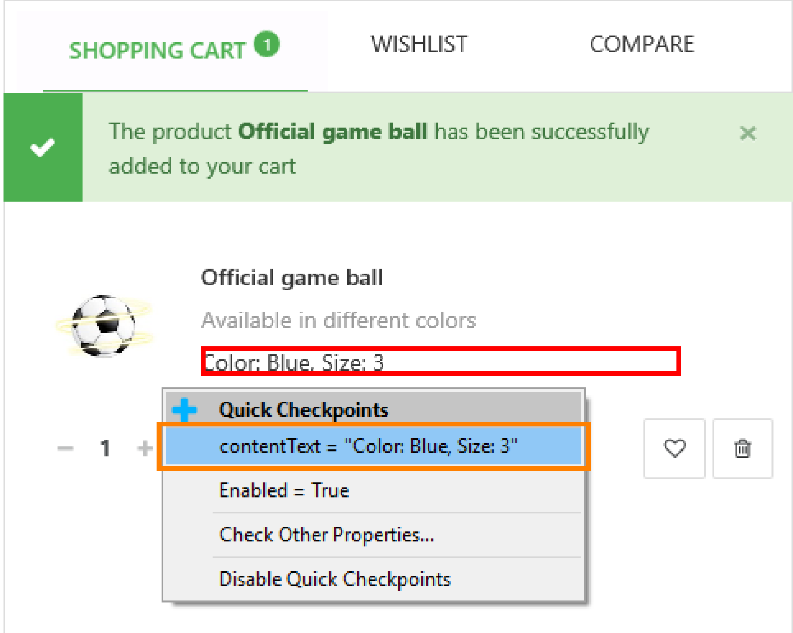
The product **Official game ball** has been successfully added to your cart

Official game ball

Available in different colors

Color: Blue, Size: 3

- In the list, select the **contentText** property:



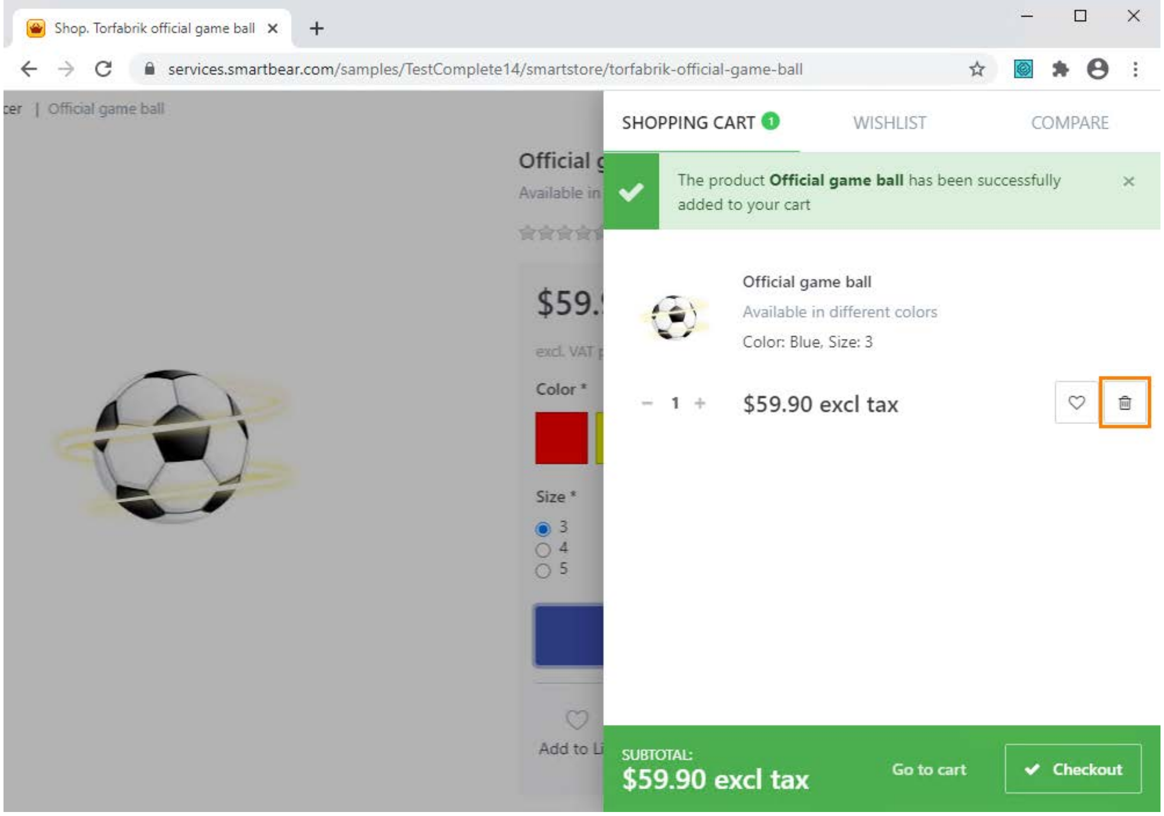
SHOPPING CART 1
WISHLIST
COMPARE
The product **Official game ball** has been successfully added to your cart
Official game ball
Available in different colors
Color: Blue, Size: 3
Quick Checkpoints
contentText = "Color: Blue, Size: 3"
Enabled = True
Check Other Properties...
Disable Quick Checkpoints
1

The property provides access to the text that the selected web element shows. The property is browser-independent. That is, it works in the same way in all the supported web browsers, which helps you avoid problems when playing back your tests in various browsers.

- TestComplete will append the checkpoint command to the recorded test and show a notification about it.

- You can continue recording user actions.

9. Click the **Delete** button next to the item to clear the Shopping Cart:



The product **Official game ball** has been successfully added to your cart

Official game ball
Available in different colors
Color: Blue, Size: 3

1 $59.90 excl tax

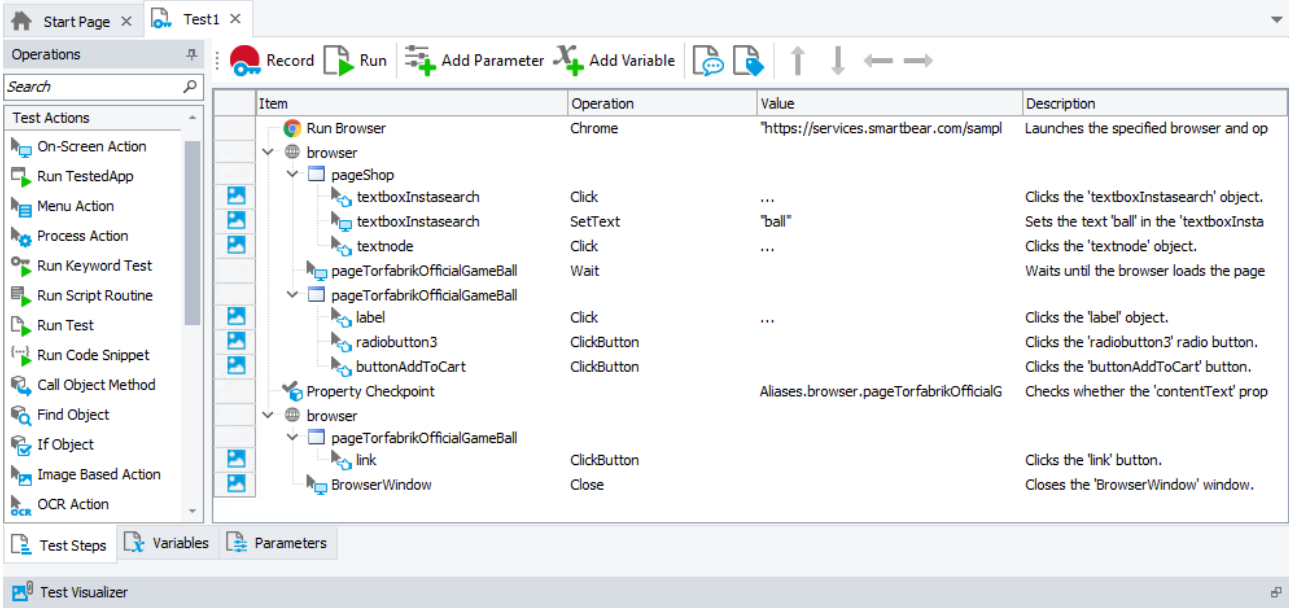
SUBTOTAL:
$59.90 excl tax

10. Close the web browser by clicking the X button on its caption bar.

11. Press 🟦 **Stop** on the Recording toolbar to stop the recording. TestComplete will process the recorded test commands and save them to a test.

# 7. Analyze the Recorded Test

After you have finished recording, TestComplete opens the recorded keyword in the Keyword Test editor:

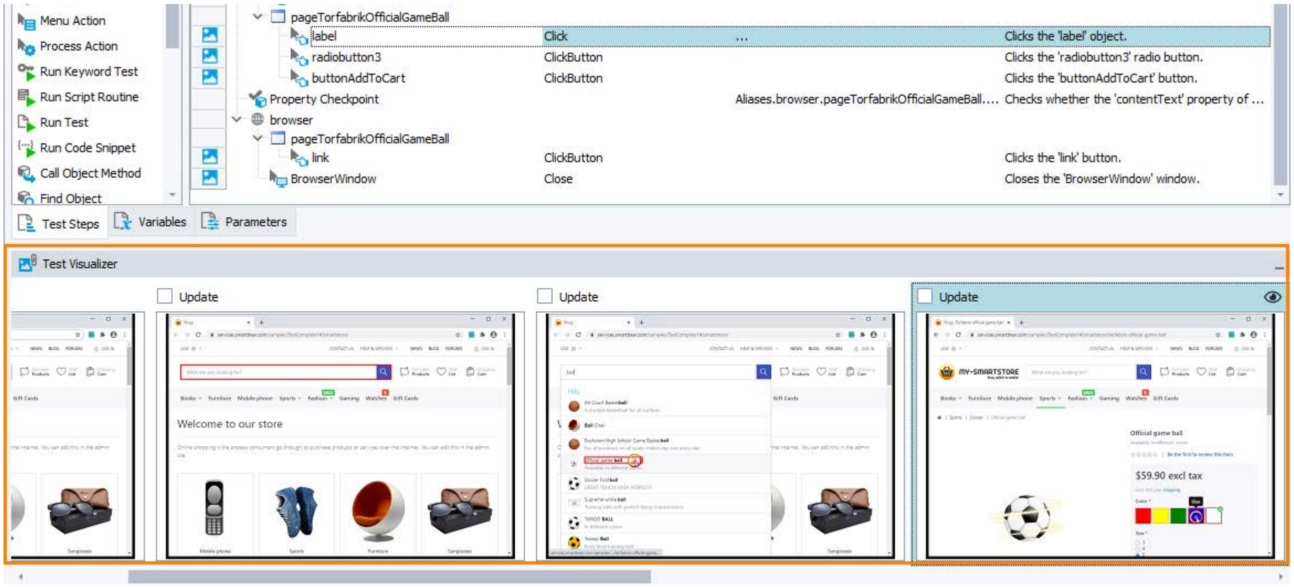


| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/sampl | Launches the specified browser and op |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInsta |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | ClickButton | | Clicks the 'buttonAddToCart' button. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whether the 'contentText' prop |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

The recorded test is similar to the test shown in the image above. Your actual test may differ from this one. For example, it may contain some unnecessary clicks.

The test contains the commands that correspond to the actions you performed on the SmartStore application during the recording. We call the test commands **operations**.

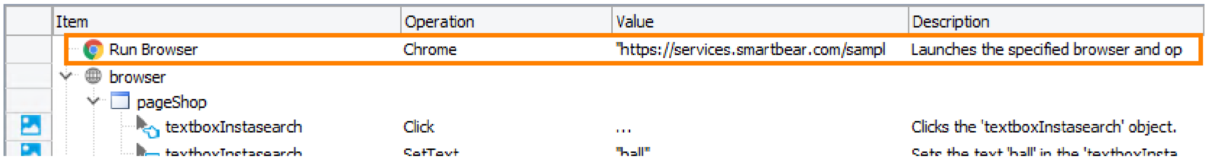
Below the commands, there is the **Test Visualizer** panel that displays images, which TestComplete captured for operations during test recording:



| | | | |
|---|---|---|---|
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | ClickButton | | Clicks the 'buttonAddToCart' button. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialGameBall.... | Checks whether the 'contentText' property of ... |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

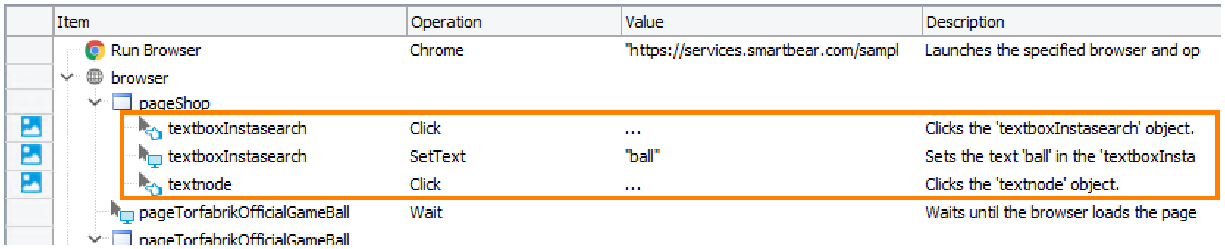
These images illustrate the recorded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (mouse clicks, typing text, and so on). When you choose an operation in the editor, Test Visualizer automatically selects the appropriate image so you can easily explore the application state before the operation is executed. For more information on working with images, see the topics in the *Test Visualizer* section.

The first operation in the test is **Run Browser**. It launches the web browser and navigates to the specified page from a keyword test. TestComplete automatically records this operation when it detects a browser launch.



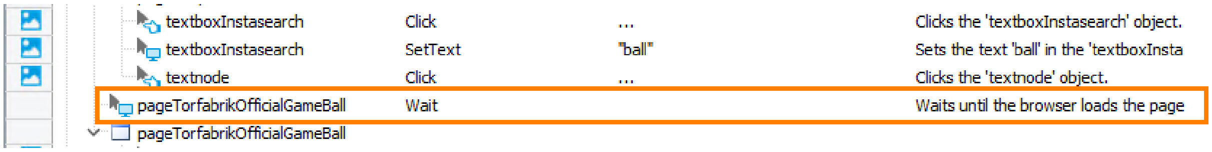
| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/sampl | Launches the specified browser and op |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInsta |

The following operations work with the Search box and search results:



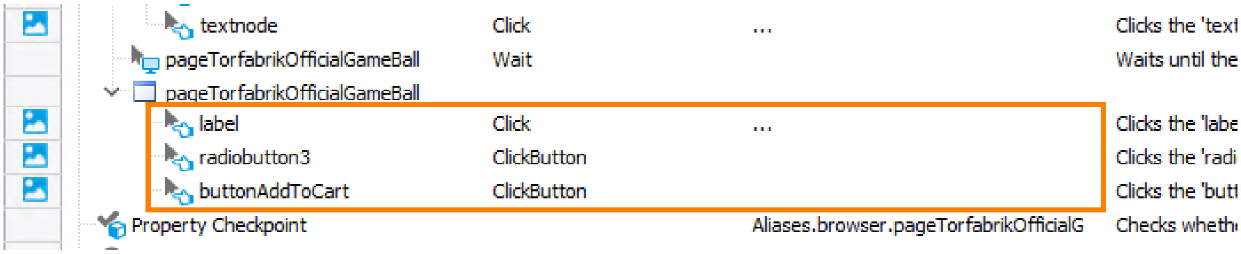
| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/sampl | Launches the specified browser and op |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInsta |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page |
| pageTorfabrikOfficialGameBall | | | |

Then, there is an operation that waits for the page with the selected item description:



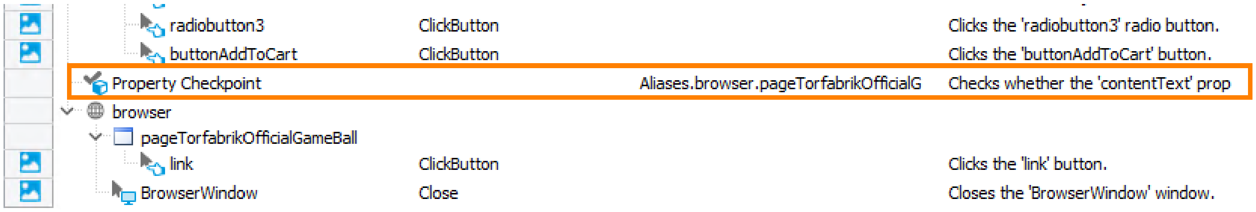
| | | | |
|---|---|---|---|
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInsta |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page |
| pageTorfabrikOfficialGameBall | | | |

It is followed by operations that simulate selecting the item color and size and adding the item to the Cart:



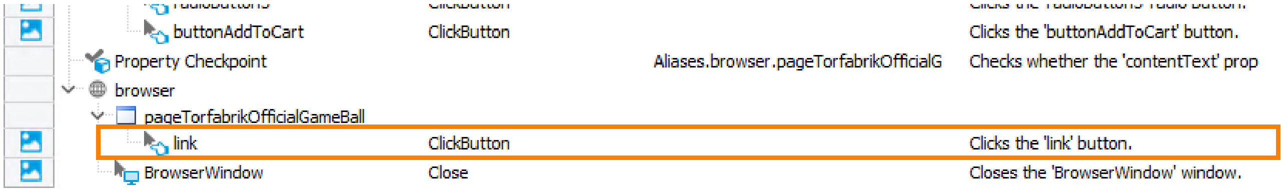
| | | | |
|---|---|---|---|
| textnode | Click | ... | Clicks the 'text |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'labe |
| radiobutton3 | ClickButton | | Clicks the 'radi |
| buttonAddToCart | ClickButton | | Clicks the 'butt |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks wheth |

Next is the comparison operation that we added during the test recording:



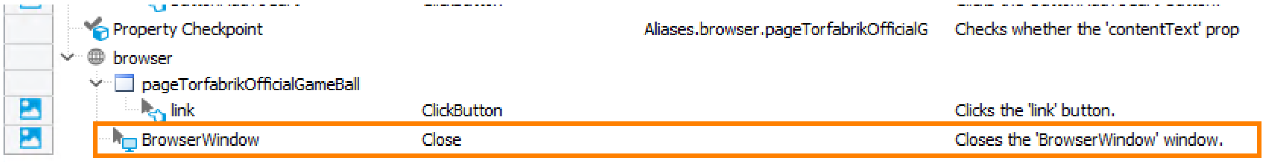
| | | | |
|---|---|---|---|
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | ClickButton | | Clicks the 'buttonAddToCart' button. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whether the 'contentText' prop |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

After that, there is an operation that clears the Shopping Cart:



| | | | |
|---|---|---|---|
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | ClickButton | | Clicks the 'buttonAddToCart' button. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whether the 'contentText' prop |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

And, an operation that closes the web browser:

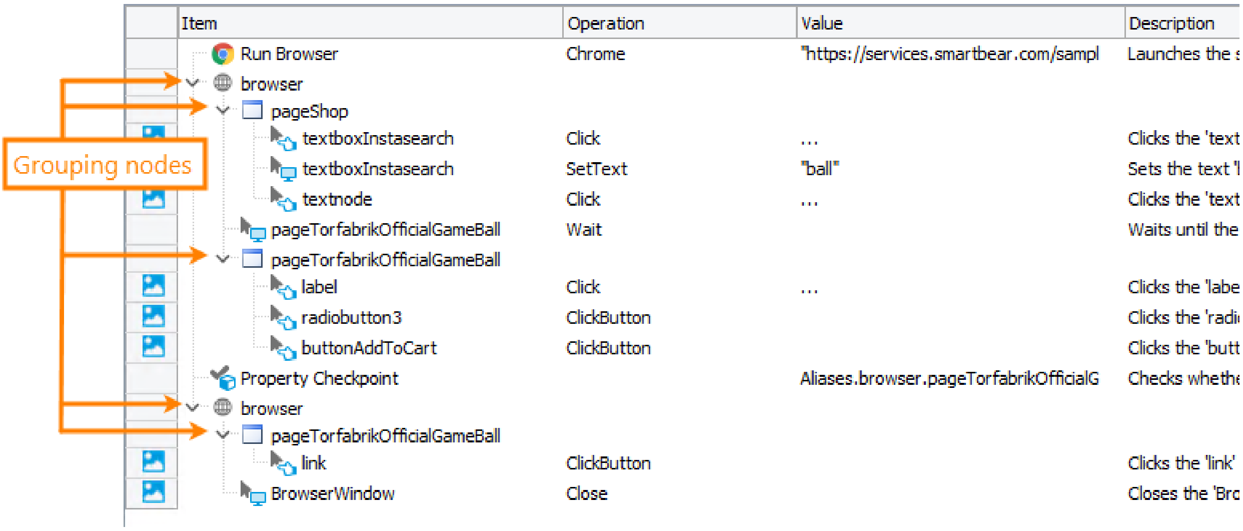


| | | | |
|---|---|---|---|
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whether the 'contentText' prop |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

For more information on simulating mouse events, keyboard input and other actions from your scripts, see *Simulating User Actions* in TestComplete Help.

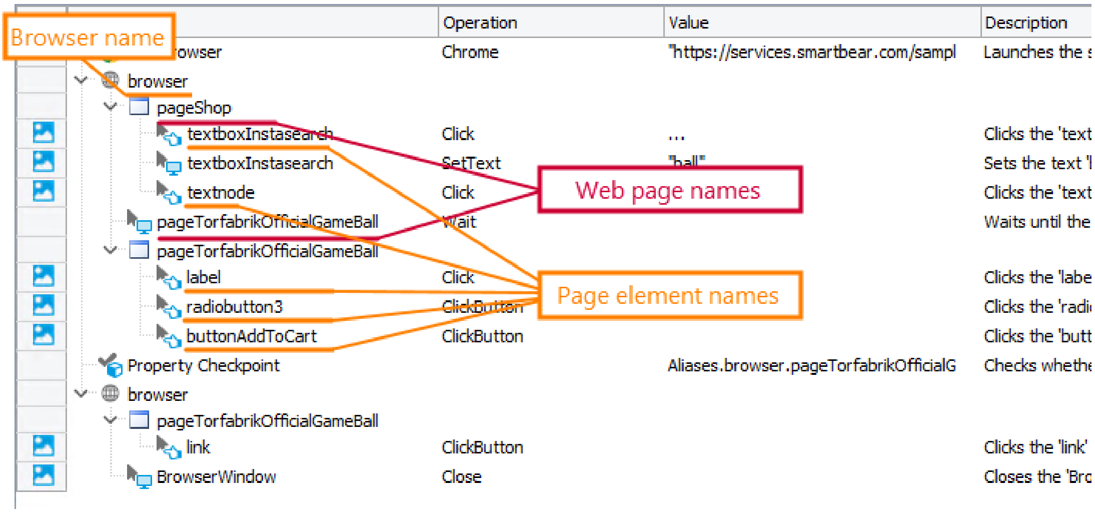
As you can see, TestComplete automatically organizes the operations into groups that correspond to the processes and windows that you worked with. Grouping makes the test structure easier to understand. It also provides some information on the object hierarchy that exists in the application under test.

We have recorded user actions for a single browser. Therefore, we have group nodes for the browser. They contain all the actions that you performed on the browser window and controls. The actions that you performed on various web pages are organized into "page" group nodes:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/sampl | Launches the s |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'text |
| textboxInstasearch | SetText | "ball" | Sets the text 'l |
| textnode | Click | ... | Clicks the 'text |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'labe |
| radiobutton3 | ClickButton | | Clicks the 'radi |
| buttonAddToCart | ClickButton | | Clicks the 'butt |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whethe |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' |
| BrowserWindow | Close | | Closes the 'Bro |

You may want to pay attention to the name of the web browser, its web pages and page elements in the recorded tests:

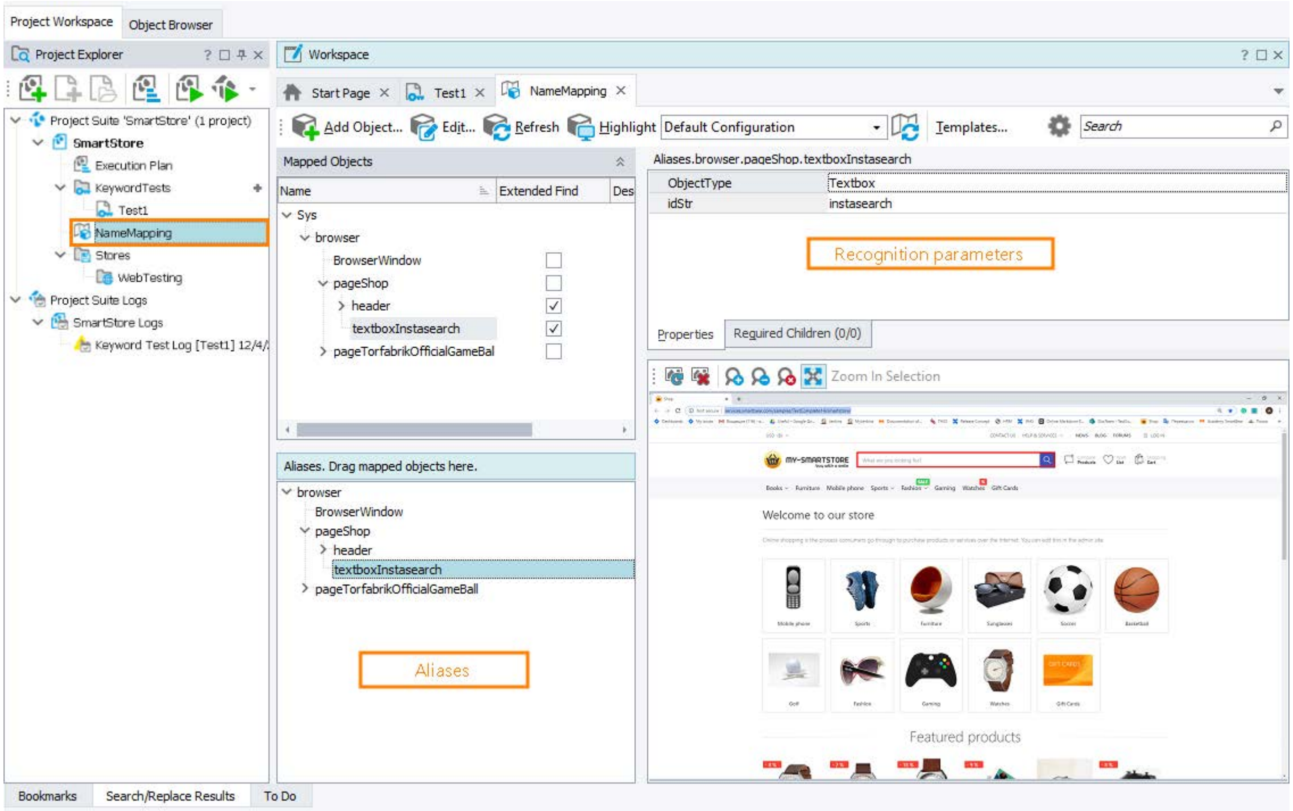


| | Operation | Value | Description |
|---|---|---|---|
| owser | Chrome | "https://services.smartbear.com/sampl | Launches the s |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'text |
| textboxInstasearch | SetText | "ball" | Sets the text 'l |
| textnode | Click | | Clicks the 'text |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | | Clicks the 'labe |
| radiobutton3 | ClickButton | | Clicks the 'radi |
| buttonAddToCart | ClickButton | | Clicks the 'butt |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialG | Checks whethe |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' |
| BrowserWindow | Close | | Closes the 'Bro |

They differ from the names you can see in the **Object Browser** panel. For instance, in the Object Browser, the name of the web browser is `Browser("iexplore")`, `Browser("edge")`, `Browser("firefox")` or `Browser("chrome")` (depending on which browser you use), while in the test, the web browser is simply called a `browser`. Another example - the main page of the SmartStore application. In the Object Browser it is called `Page("https://services.smartbear.com/samples/TestComplete14/smartstore/")`, while in the test its name is much shorter: `pageShop`.

That is why this happens: by default, while you are recording a test, TestComplete automatically adds all items with which you are interacting to the **Name Mapping** repository. For each item, TestComplete stores the recognition parameters it will later use to find the object in the application. Also, for each item, it stores an **alias -** a short name it uses to address objects in tests.

You can double-click the **Name Mapping** item in the Project Explorer to open the Name Mapping repository and view the recognition parameters and aliases that TestComplete created for objects during the test recording:



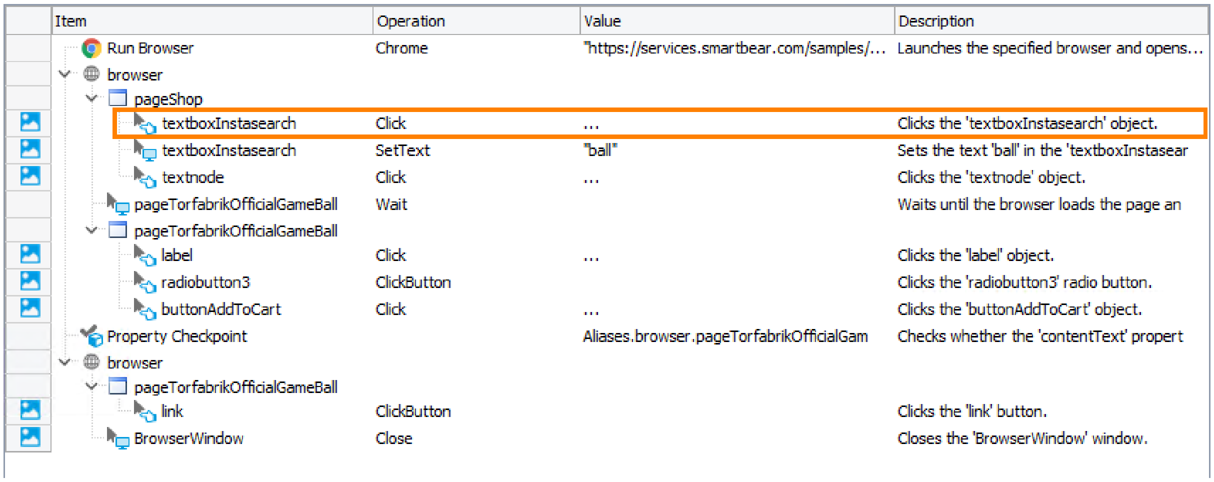
| | |
|---|---|
| ObjectType | Textbox |
| idStr | instasearch |

Using Name Mapping and aliases makes tests easier to understand and more stable.

## 8. Refine the Recorded Test

You can run the recorded test as is, but we suggest that you adjust it to make it more stable.

When you record user actions, for example, mouse clicks, TestComplete records coordinates where the click is performed. For example, the image below shows the **Click** operation that simulates a click at the specific point of the **Search** text box of the tested application:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/samples/... | Launches the specified browser and opens... |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInstasear |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page an |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | Click | ... | Clicks the 'buttonAddToCart' object. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialGam | Checks whether the 'contentText' propert |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

When you run the recorded test, TestComplete will simulate a click at the exact same point where it was performed during the test recording. This can be useful when you are testing a *black-box application* – an application to the controls of which TestComplete does not have access, and it has to rely on coordinates to simulate user actions. It also can be useful for testing complex controls, for which clicking on various parts of the control brings various results.

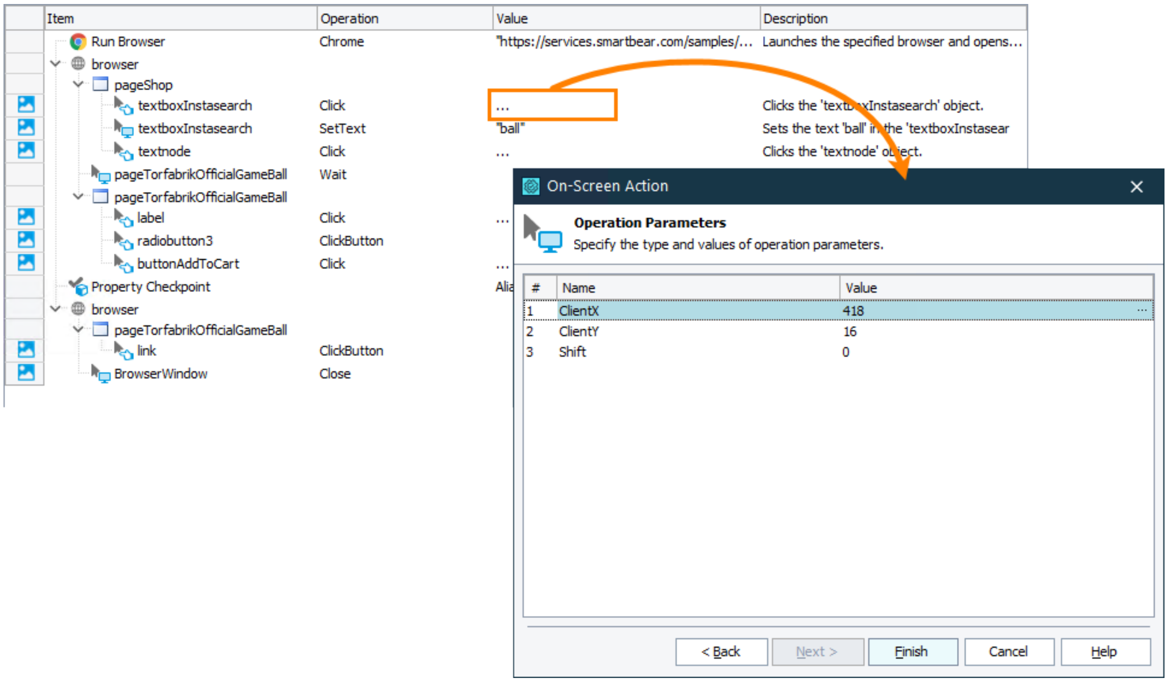
However, in this tutorial we are testing an **Open Application**: TestComplete can recognize all its controls and has access to their properties and methods. Therefore, we do not need to rely on coordinates to simulate user actions.

In addition, the application we are testing has a responsive design: it adjusts the size and position of its controls according to the browser window size. Coordinates that TestComplete records for one browser window size will become incorrect when you run your test for another browser window size.

Let’s modify the recorded test so that it simulates all click actions in the top left corner of the appropriate controls. This way, the test will simulate clicks correctly regardless of the browser window size and the application’s current control layout:

1. In your test, find the operation that simulates a click in the **Search** text box and click the ellipsis button in the **Value** column.

   TestComplete will open the **Operation Parameters** dialog for this operation:



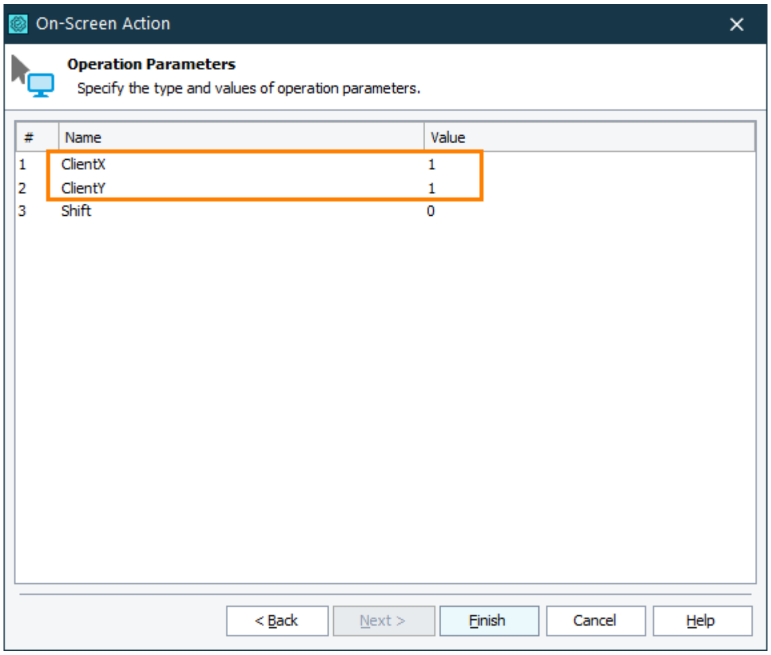
| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/samples/… | Launches the specified browser and opens… |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | … | Clicks the 'textb xInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInstasear |
| textnode | Click | … | Clicks the 'textnode' o ject. |
| pageTorfabrikOfficialGameBall | Wait | | |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | … | |
| radiobutton3 | ClickButton | | |
| buttonAddToCart | Click | … | |
| Property Checkpoint | | Alia | |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | |
| BrowserWindow | Close | | |

**Operation Parameters**

Specify the type and values of operation parameters.

| # | Name | Value |
|---|---|---|
| 1 | ClientX | 418 |
| 2 | ClientY | 16 |
| 3 | Shift | 0 |

2. In the dialog, replace the recorded *ClientX* and *ClientY* parameters with **1**. For TestComplete, this corresponds to clicking at the top left corner of the control:



**Operation Parameters**

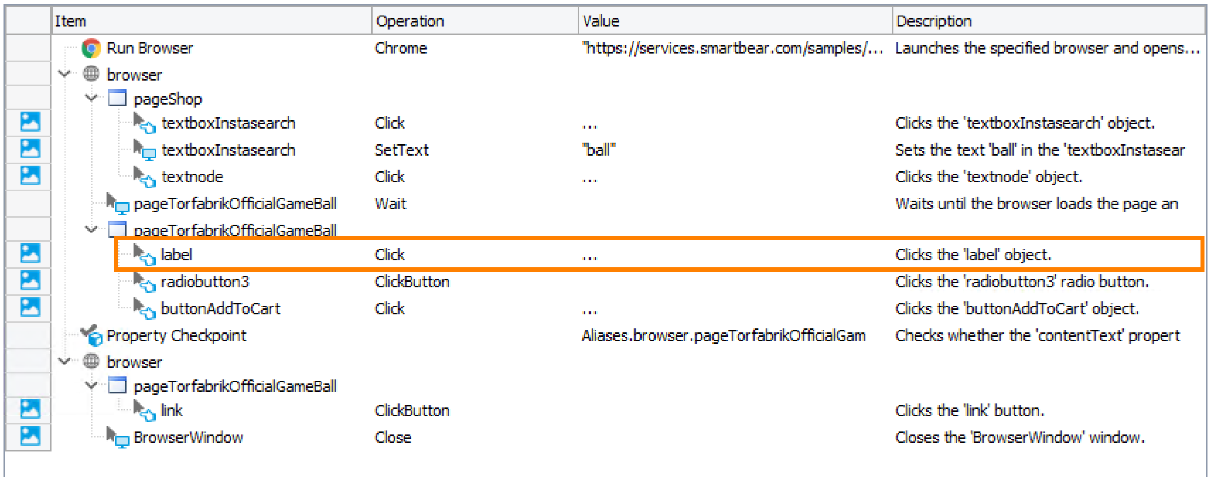
Specify the type and values of operation parameters.

| # | Name | Value |
|---|---|---|
| 1 | ClientX | 1 |
| 2 | ClientY | 1 |
| 3 | Shift | 0 |

   Click **Finish** to apply the changes and close the dialog.

**Note:** As an alternative, you can set the click coordinates to *-1*, *-1*. For TestComplete this will correspond to clicking at the center of the control.

3. The same way, modify the coordinates of the operation that simulates a click in the search result list and on the item color check box:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/samples/... | Launches the specified browser and opens... |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInstasear |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page an |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | ... | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | Click | ... | Clicks the 'buttonAddToCart' object. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialGam | Checks whether the 'contentText' propert |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

4. If in your test you have any other operations that simulate clicks at hard-coded coordinates, adjust the coordinates if you wish.

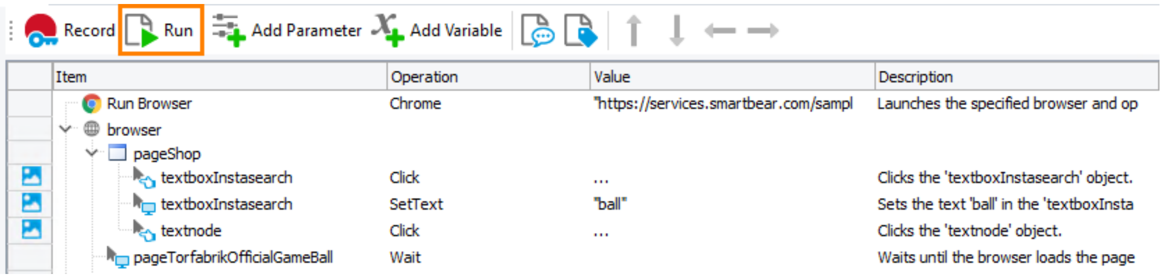
5. From the TestComplete main menu, select **File** > **Save All** to save all the changes.

## 9. Run the Recorded Test

Now we can run our simple test to see how TestComplete simulates user actions.

**Before running a recorded test, make sure the initial conditions are the same as those when you started recording.** For instance, a web test almost always requires that a web browser is running and the needed page is open. In our case, to launch the browser and open the tested web page, we use the Run Browser operation at the beginning of the test.

To run the recorded test, click **Run** on the test editor toolbar:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/sampl | Launches the specified browser and op |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | ... | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInsta |
| textnode | Click | ... | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page |

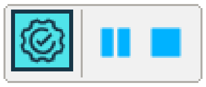
TestComplete will start executing test commands. In our case, the test will simply repeat your recorded actions.

> **Note:** Do not move the mouse or press keys during the test execution. Your actions may interfere with actions simulated by TestComplete and the test execution may go wrong.

After the test execution is over, TestComplete will restore its window and display the test results. In the next step, we will analyze them.

Some notes about the test run:

- During test execution, TestComplete displays an indicator in the top right corner of the screen:

  The indicator displays messages informing you about the simulated test actions.

- TestComplete executes the test commands until the test ends. You can stop the execution at any time by clicking ■ **Stop** on the Test Engine toolbar or in the indicator, or by selecting **Test > Stop** from TestComplete main menu.

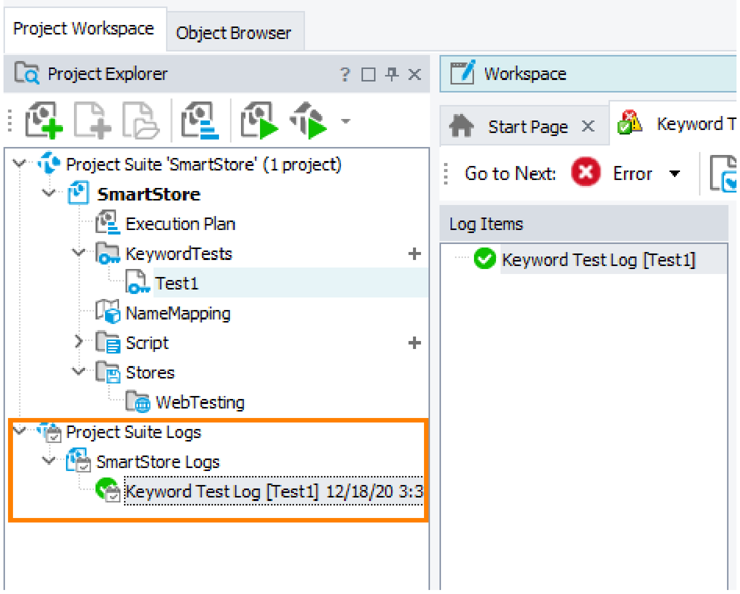
  You can pause the test execution by clicking ▮▮ **Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check the test variables and objects using TestComplete **Watch List** or **Locals** panel or the **Evaluate** dialog (see *Debugging Tests* in TestComplete Help).

- To run tests on computers that do not have TestComplete installed, you can use a resource-friendly utility called TestExecute.

For complete information on running tests in TestComplete, on project settings that affect the runs and on the test execution, see *Running Tests* in TestComplete Help.

## 10. Analyze Test Results

TestComplete keeps a complete log of all operations performed during testing. The Project Explorer keeps the test results under the **Project Suite Logs > SmartStore Log** node. This is where you are viewing the test history of the project and project suite. Each node corresponds to a test run. An icon to the left of the node shows whether the corresponding test run passed successfully:

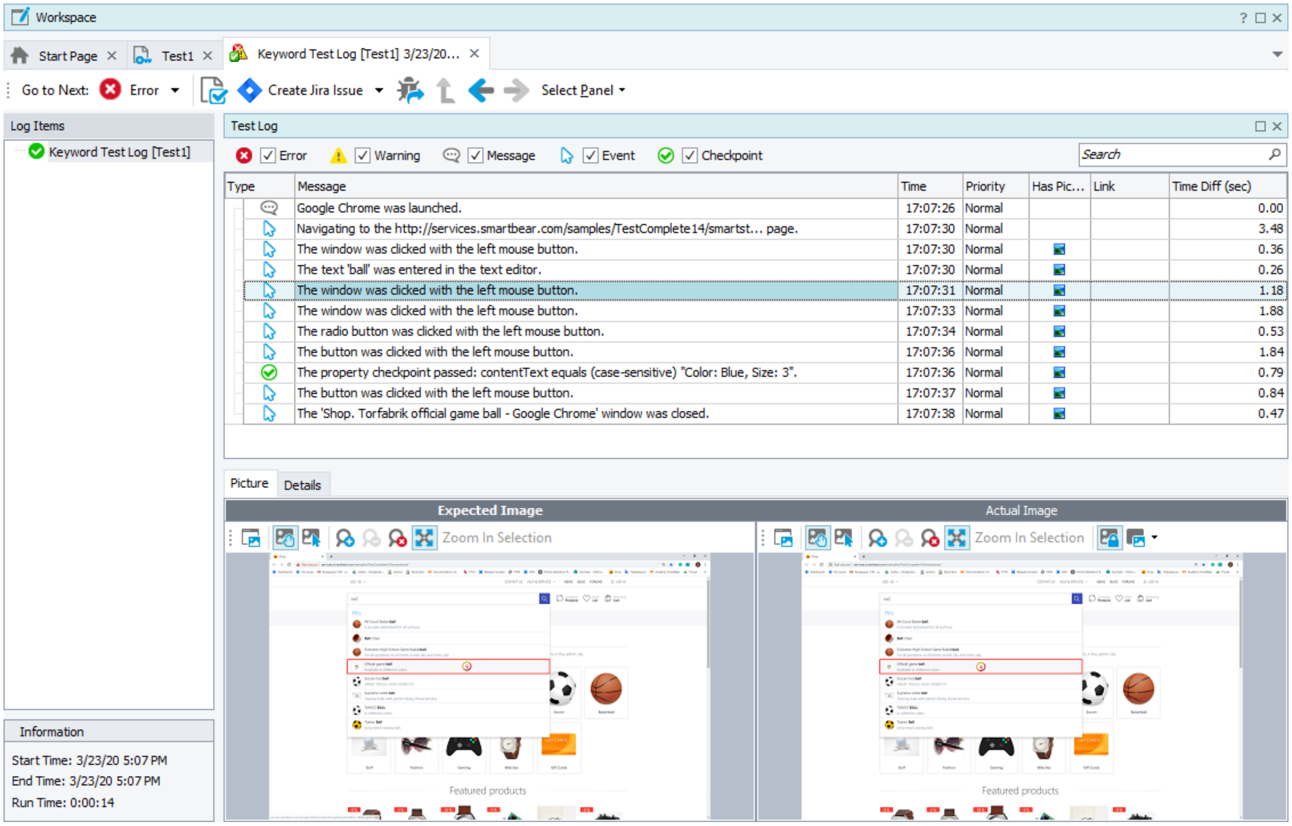


TestComplete adds nodes for the last results automatically *after* the test execution is *over*. It does not display the results when the test is running (you can view intermediate results if you pause the test execution).

Because we have run only one test so far, we have only one log node in the Project Explorer. By default, TestComplete automatically opens that log. To open a test log manually, double-click the needed log node in the Project Explorer panel.

**Note:** By default, TestComplete stores all test results in log files. The number of the log files will grow with every test run, and this will result in the memory consumption increase. To reduce memory usage, you can delete files from the log manually or limit the number of log files to be kept.

In our example, the log is as follows –



| Type | Message | Time | Priority | Has Pic... | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
|  | Google Chrome was launched. | 17:07:26 | Normal |  |  | 0.00 |
|  | Navigating to the http://services.smartbear.com/samples/TestComplete14/smartst... page. | 17:07:30 | Normal |  |  | 3.48 |
|  | The window was clicked with the left mouse button. | 17:07:30 | Normal | ▨ |  | 0.36 |
|  | The text 'ball' was entered in the text editor. | 17:07:30 | Normal | ▨ |  | 0.26 |
|  | The window was clicked with the left mouse button. | 17:07:31 | Normal | ▨ |  | 1.18 |
|  | The window was clicked with the left mouse button. | 17:07:33 | Normal | ▨ |  | 1.88 |
|  | The radio button was clicked with the left mouse button. | 17:07:34 | Normal | ▨ |  | 0.53 |
|  | The button was clicked with the left mouse button. | 17:07:36 | Normal | ▨ |  | 1.84 |
| ✓ | The property checkpoint passed: contentText equals (case-sensitive) "Color: Blue, Size: 3". | 17:07:36 | Normal | ▨ |  | 0.79 |
|  | The button was clicked with the left mouse button. | 17:07:37 | Normal | ▨ |  | 0.84 |
|  | The 'Shop. Torfabrik official game ball - Google Chrome' window was closed. | 17:07:38 | Normal | ▨ |  | 0.47 |

Start Time: 3/23/20 5:07 PM
End Time: 3/23/20 5:07 PM
Run Time: 0:00:14

The log window shows the results of one test run at a time. On the left side of the window, there is a tree-like structure of the executed tests; you can select each of them to view their results. We have run only one test, so in our case this tree only contains one node. The node icon indicates whether the test passed successfully or failed.

The test log contains error, warning, informative, and other types of messages. The icon on the left indicates the message type. With the check boxes at the top of the message list, you can hide or view messages by type.

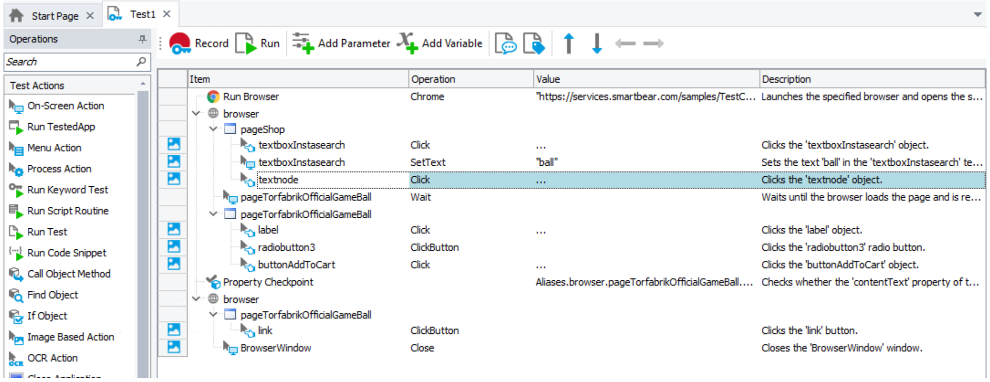
For each message, the log also shows the time that each action was performed. You can see it in the Time column.

TestComplete may post additional text and images along with the message. To view them, select the desired message in the log and look in the **Details** and **Picture** panes that are below the message list.

The Picture panel shows the images of the expected and the actual application state when executing the selected test command: the “Expected” image was captured for the command during the test recording, the

"Actual" image was captured during the test run. You can click **View Comparison Results** to compare the images and easily see the difference. This simplifies the search for errors that may occur in your test. For more information, see the topics of the **Test Visualizer** section.

To view a test operation that posted a message to the log, double-click the desired message in the log. TestComplete will open the keyword test in the editor and highlight the appropriate operation. For instance, if you double-click the "The button was clicked with the left mouse button." message in the log, TestComplete will highlight the keyword test operation that performed this action:



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Browser | Chrome | "https://services.smartbear.com/samples/TestC… | Launches the specified browser and opens the s… |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | … | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInstasearch' te… |
| textnode | Click | … | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page and is re… |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | … | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | Click | … | Clicks the 'buttonAddToCart' object. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialGameBall.… | Checks whether the 'contentText' property of t… |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

For detailed information on the test log panels, on posting messages to the log and on working with the results, see the *About Test Log* section in TestComplete Help.

> **Note:** The log that we described is typical for TestComplete keyword tests and scripts. Tests of other types may form a log of a different structure. For detailed information about these logs, see the description of the appropriate project item, or simply click within the log page and press **F1**.

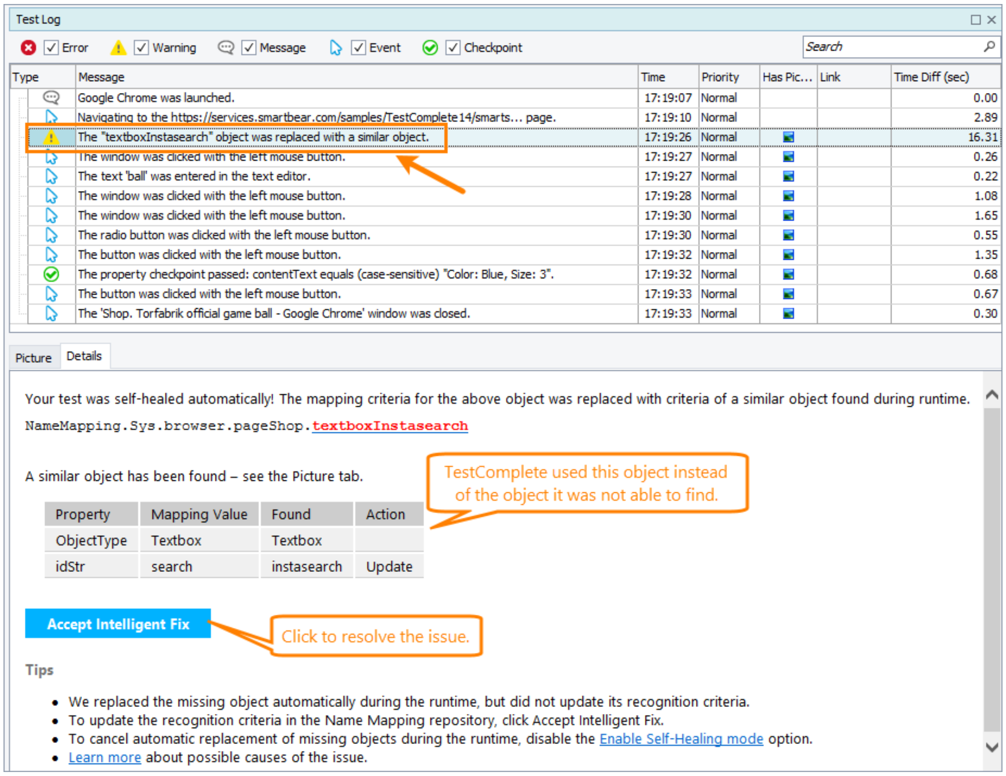
## Resolving Errors

Your test may fail. There can be several possible reasons for this. For example, developers could change the application behavior, the recognition attributes of pages and web elements could change and make the test engine fail to find the needed objects, a third-party application may overlap your web browser and make the test engine fail to simulate actions on it, and so on.

One of the most typical reasons that novice users face is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance:

- If your web browser had been running before you recorded a test, it must also be running before you run the test.
- If your tested web page was opened in your web browser when you recorded your test, it should also be opened when you run the test.
- If you edited any data in the application, and then saved it, you need to revert the changes.

If the tested application has changed and the identification properties that your test uses to find tested objects are no longer valid, TestComplete will try to detect the changes to find the missing objects. If your test log reports the "*Object <Object_Name> was replaced with a similar object*" warning, this means that TestComplete was not able to find the needed tested object during the test run and used a similar object instead.



| Type | Message | Time | Priority | Has Pic... | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
| 💬 | Google Chrome was launched. | 17:19:07 | Normal | | | 0.00 |
| | Navigating to the https://services.smartbear.com/samples/TestComplete14/smarts... page. | 17:19:10 | Normal | | | 2.89 |
| ⚠ | The "textboxInstasearch" object was replaced with a similar object. | 17:19:26 | Normal | ▣ | | 16.31 |
| | The window was clicked with the left mouse button. | 17:19:27 | Normal | ▣ | | 0.26 |
| | The text 'ball' was entered in the text editor. | 17:19:27 | Normal | ▣ | | 0.22 |
| | The window was clicked with the left mouse button. | 17:19:28 | Normal | ▣ | | 1.08 |
| | The window was clicked with the left mouse button. | 17:19:30 | Normal | ▣ | | 1.65 |
| | The radio button was clicked with the left mouse button. | 17:19:30 | Normal | ▣ | | 0.55 |
| | The button was clicked with the left mouse button. | 17:19:32 | Normal | ▣ | | 1.35 |
| ✓ | The property checkpoint passed: contentText equals (case-sensitive) "Color: Blue, Size: 3". | 17:19:32 | Normal | ▣ | | 0.68 |
| | The button was clicked with the left mouse button. | 17:19:33 | Normal | ▣ | | 0.67 |
| | The 'Shop. Torfabrik official game ball - Google Chrome' window was closed. | 17:19:33 | Normal | ▣ | | 0.30 |

Your test was self-healed automatically! The mapping criteria for the above object was replaced with criteria of a similar object found during runtime.

`NameMapping.Sys.browser.pageShop.textboxInstasearch`

A similar object has been found – see the Picture tab.

| Property | Mapping Value | Found | Action |
|---|---|---|---|
| ObjectType | Textbox | Textbox | |
| idStr | search | instasearch | Update |

TestComplete used this object instead of the object it was not able to find.

Click to resolve the issue.

Tips

- We replaced the missing object automatically during the runtime, but did not update its recognition criteria.
- To update the recognition criteria in the Name Mapping repository, click Accept Intelligent Fix.
- To cancel automatic replacement of missing objects during the runtime, disable the Enable Self-Healing mode option.
- Learn more about possible causes of the issue.

Do not ignore this warning. Examine it to learn about the missing and the replaced objects and update the object's identification properties to resolve the issue.

For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* in TestComplete Help.

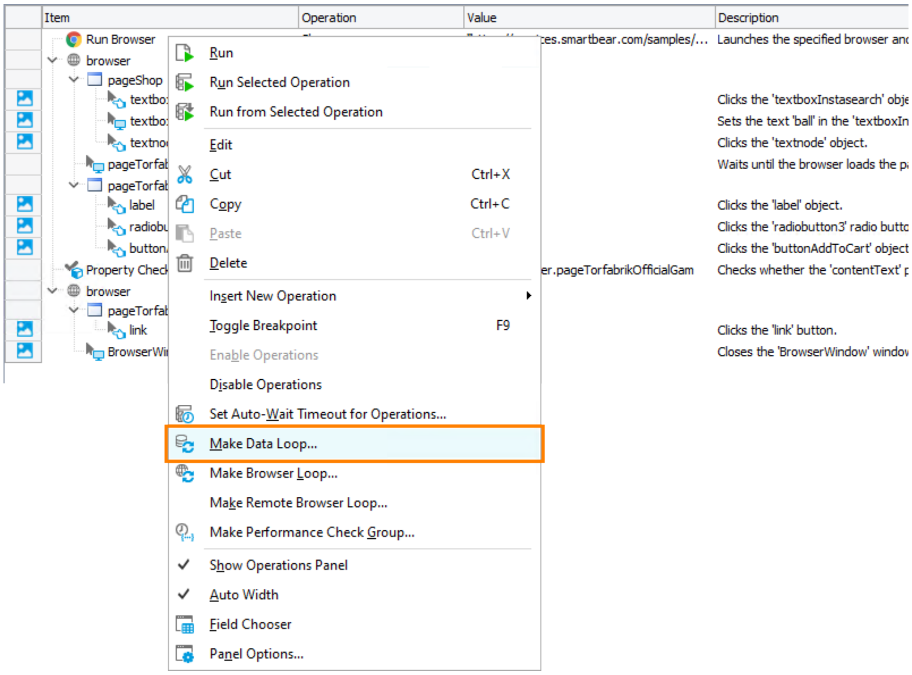
## 11. Run the Test in Multiple Browsers

After you make sure that the test is executed successfully in the browser you used for recording, you can easily modify the test so that it can be executed in different browsers (the so called **cross-browser testing**). This helps you ensure that the web application works correctly in various browsers.

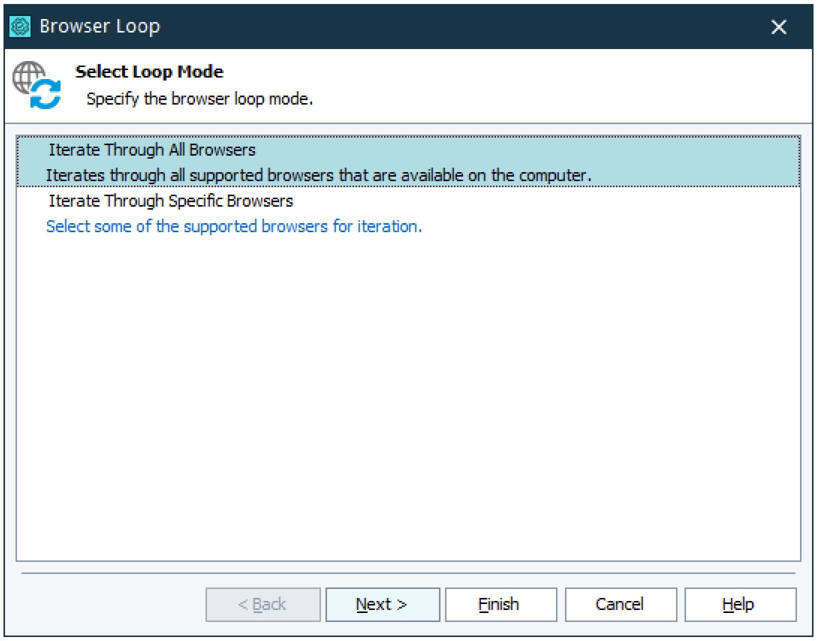
Now, we will modify the test so that it runs in all supported browsers installed in our system:

1. Open the test in the Keyword Test editor.

2. Right-click the Run Browser operation and select **Make Browser Loop** from the context menu.


Item
Operation
Value
Description
Run Browser
browser
pageShop
textbo:
textbo:
textno
pageTorfal
pageTorfal
label
radiobu
button
Property Check
browser
pageTorfal
link
BrowserWi
...ces.smartbear.com/samples/...
Launches the specified browser and
Clicks the 'textboxInstasearch' obje
Sets the text 'ball' in the 'textboxIn
Clicks the 'textnode' object.
Waits until the browser loads the p
Clicks the 'label' object.
Clicks the 'radiobutton3' radio butto
Clicks the 'buttonAddToCart' object
...er.pageTorfabrikOfficialGam
Checks whether the 'contentText' p
Clicks the 'link' button.
Closes the 'BrowserWindow' window
Run
Run Selected Operation
Run from Selected Operation
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Insert New Operation
Toggle Breakpoint F9
Enable Operations
Disable Operations
Set Auto-Wait Timeout for Operations...
Make Data Loop...
Make Browser Loop...
Make Remote Browser Loop...
Make Performance Check Group...
Show Operations Panel
Auto Width
Field Chooser
Panel Options...

3. In the ensuing operation parameters dialog, select **Iterate Through All Browsers** and click **Finish**.



**Select Loop Mode**

Specify the browser loop mode.

Iterate Through All Browsers

Iterates through all supported browsers that are available on the computer.

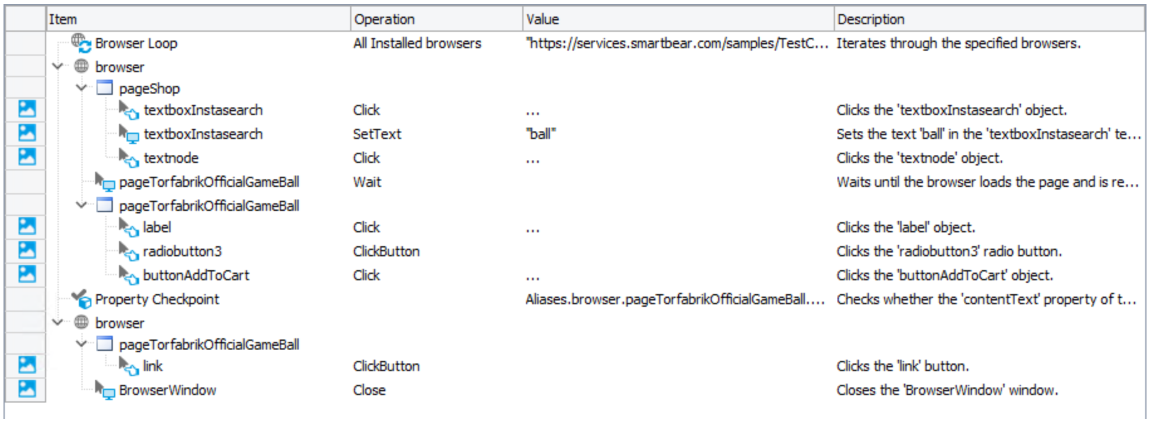
Iterate Through Specific Browsers

Select some of the supported browsers for iteration.

This will convert the **Run Browser** operation into the **Browser Loop** operation.

4. Select all the test operations that go after the Browser Loop operation and click ➡ to move them inside the loop. Now these operations will be executed on each loop iteration.

Here is how the resulting test should look like:



| Item | Operation | Value | Description |
|---|---|---|---|
| Browser Loop | All Installed browsers | "https://services.smartbear.com/samples/TestC… | Iterates through the specified browsers. |
| browser | | | |
| pageShop | | | |
| textboxInstasearch | Click | … | Clicks the 'textboxInstasearch' object. |
| textboxInstasearch | SetText | "ball" | Sets the text 'ball' in the 'textboxInstasearch' te… |
| textnode | Click | … | Clicks the 'textnode' object. |
| pageTorfabrikOfficialGameBall | Wait | | Waits until the browser loads the page and is re… |
| pageTorfabrikOfficialGameBall | | | |
| label | Click | … | Clicks the 'label' object. |
| radiobutton3 | ClickButton | | Clicks the 'radiobutton3' radio button. |
| buttonAddToCart | Click | … | Clicks the 'buttonAddToCart' object. |
| Property Checkpoint | | Aliases.browser.pageTorfabrikOfficialGameBall.… | Checks whether the 'contentText' property of t… |
| browser | | | |
| pageTorfabrikOfficialGameBall | | | |
| link | ClickButton | | Clicks the 'link' button. |
| BrowserWindow | Close | | Closes the 'BrowserWindow' window. |

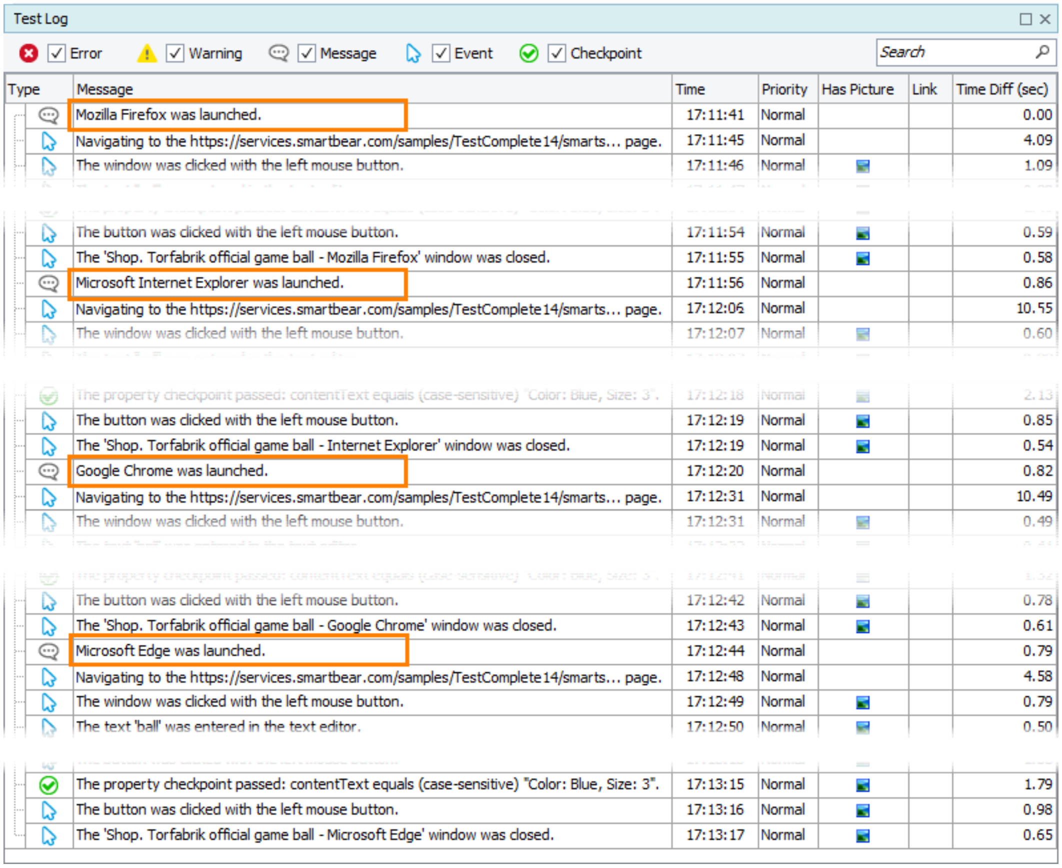
5. Save the test by selecting **File** > **Save** from TestComplete main menu.

Prepare and configure another browser as described in the *Preparing Web Browsers* section in TestComplete help.

Now run the resulting test.

TestComplete will repeat the test operations several times. Each time, the test actions will be performed in a different browser.

The test log contains information about which browser was used and the results of the test operations performed in each browser:



| Type | Message | Time | Priority | Has Picture | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
| 💬 | Mozilla Firefox was launched. | 17:11:41 | Normal | | | 0.00 |
| ↖ | Navigating to the https://services.smartbear.com/samples/TestComplete14/smarts… page. | 17:11:45 | Normal | | | 4.09 |
| ↖ | The window was clicked with the left mouse button. | 17:11:46 | Normal | ▣ | | 1.09 |
| ↖ | The button was clicked with the left mouse button. | 17:11:54 | Normal | ▣ | | 0.59 |
| ↖ | The 'Shop. Torfabrik official game ball - Mozilla Firefox' window was closed. | 17:11:55 | Normal | ▣ | | 0.58 |
| 💬 | Microsoft Internet Explorer was launched. | 17:11:56 | Normal | | | 0.86 |
| ↖ | Navigating to the https://services.smartbear.com/samples/TestComplete14/smarts… page. | 17:12:06 | Normal | | | 10.55 |
| ↖ | The window was clicked with the left mouse button. | 17:12:07 | Normal | ▣ | | 0.60 |
| ✓ | The property checkpoint passed: contentText equals (case-sensitive) "Color: Blue, Size: 3". | 17:12:18 | Normal | ▣ | | 2.13 |
| ↖ | The button was clicked with the left mouse button. | 17:12:19 | Normal | ▣ | | 0.85 |
| ↖ | The 'Shop. Torfabrik official game ball - Internet Explorer' window was closed. | 17:12:19 | Normal | ▣ | | 0.54 |
| 💬 | Google Chrome was launched. | 17:12:20 | Normal | | | 0.82 |
| ↖ | Navigating to the https://services.smartbear.com/samples/TestComplete14/smarts… page. | 17:12:31 | Normal | | | 10.49 |
| ↖ | The window was clicked with the left mouse button. | 17:12:31 | Normal | ▣ | | 0.49 |
| ↖ | The button was clicked with the left mouse button. | 17:12:42 | Normal | ▣ | | 0.78 |
| ↖ | The 'Shop. Torfabrik official game ball - Google Chrome' window was closed. | 17:12:43 | Normal | ▣ | | 0.61 |
| 💬 | Microsoft Edge was launched. | 17:12:44 | Normal | | | 0.79 |
| ↖ | Navigating to the https://services.smartbear.com/samples/TestComplete14/smarts… page. | 17:12:48 | Normal | | | 4.58 |
| ↖ | The window was clicked with the left mouse button. | 17:12:49 | Normal | ▣ | | 0.79 |
| ↖ | The text 'ball' was entered in the text editor. | 17:12:50 | Normal | ▣ | | 0.50 |
| ✓ | The property checkpoint passed: contentText equals (case-sensitive) "Color: Blue, Size: 3". | 17:13:15 | Normal | ▣ | | 1.79 |
| ↖ | The button was clicked with the left mouse button. | 17:13:16 | Normal | ▣ | | 0.98 |
| ↖ | The 'Shop. Torfabrik official game ball - Microsoft Edge' window was closed. | 17:13:17 | Normal | ▣ | | 0.65 |

For more information on cross-browser testing with TestComplete, see *Cross-Browser Testing – Overview* in TestComplete help.

# Testing Android Applications

This tutorial explains how to test Android applications with TestComplete. It assumes that you are familiar with general principles of automated testing and have minimal knowledge of the TestComplete IDE.

> ! If you are a novice user, we recommend that you read an introduction to automated testing described above.

## Requirements

You can follow this tutorial and create a test for an Android application only if you have the following:

- A Windows computer with TestComplete installed and an active license for the TestComplete Mobile module.
- Access to an Android device controlled by an Appium server. It can be:
    - The mobile device cloud provided by BitBar. We will use it in this tutorial. We will show how to create a free trial BitBar account and access the mobile devices it provides.

    – or –

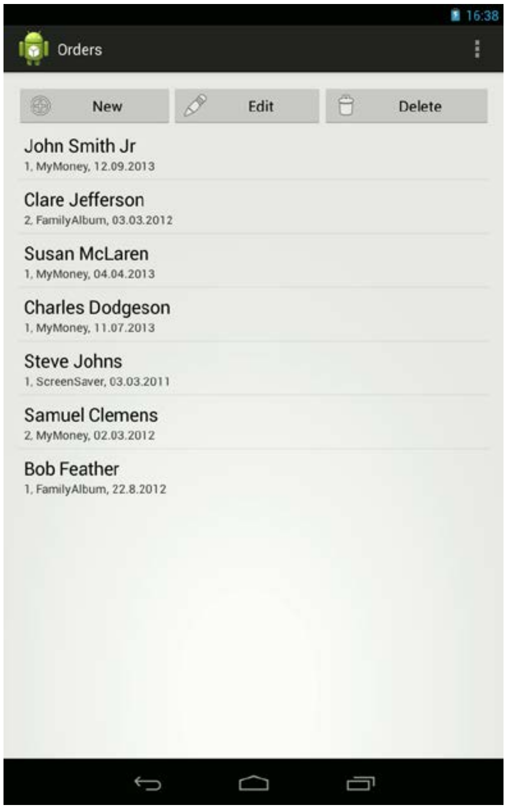
    - A private Appium server running on your local computer or on a remote computer in your local network. Using a private Appium server is beyond this tutorial scope.

## In this tutorial

We are going to create a trial BitBar account, connect to an Android mobile device in the BitBar device cloud, deploy a tested application to it, create and run a simple test, and analyze the results.

## About Tested Application

In our explanations, we will use the Android version of the Orders application. This application lets you manage a table of purchase orders: you can view the list of existing orders, modify, or remove them, as well as add new orders to the list:



John Smith Jr
1, MyMoney, 12.09.2013

Clare Jefferson
2, FamilyAlbum, 03.03.2012

Susan McLaren
1, MyMoney, 04.04.2013

Charles Dodgeson
1, MyMoney, 11.07.2013

Steve Johns
1, ScreenSaver, 03.03.2011

Samuel Clemens
2, MyMoney, 02.03.2012

Bob Feather
1, FamilyAlbum, 22.8.2012

### To get the application

1. Download the TestComplete Samples installation package from our website:

   ➩ https://support.smartbear.com/testcomplete/downloads/samples/

2. Run the installation.
3. The sample will be installed to *the <TestComplete Samples>\Mobile\Android\Orders\* folder.

## 1. Sign Up for a Free BitBar Account

In TestComplete, you can connect to iOS devices controlled by an Appium server and create and run automated tests on these devices. In this tutorial, we will use a ready-to-use solution provided by BitBar, a cloud-based mobile testing service by SmartBear.

We will show how to create a free trial account and use it in the tutorial. Requesting a trial takes only several moments and requires only an active email.

You can sign up for the trial before you start the tutorial, or you can skip it and do it later when you start the actual test recording. Both ways are acceptable.

### 1. Sign up for a trial BitBar account

1. Click the following link to sign up for a free trial on the BitBar website:

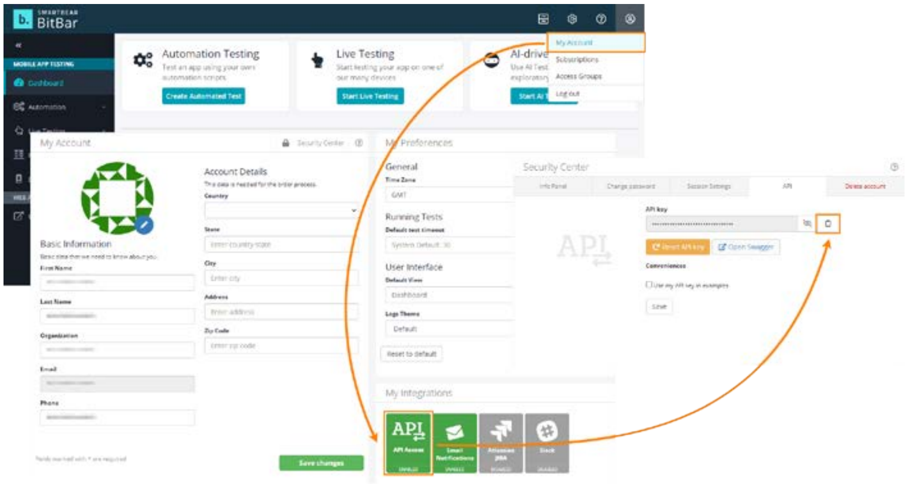
   ➩ bitbar.com/signup

2. Follow the web form instructions.

### 2. Get the BitBar API Key

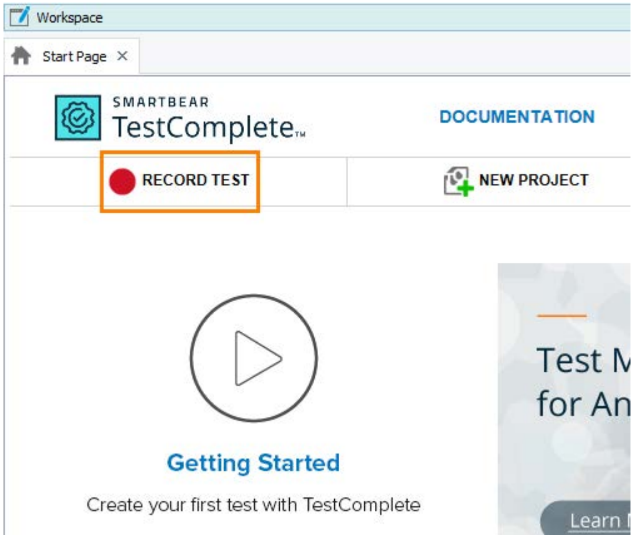
To access BitBar resources from TestComplete tests, the **BitBar API Key** is used. To get the key:

1. Log in to BitBar with your BitBar account.
2. Click 👤 > **My Account** at the top right of the page.
3. In the My Integrations section, click **API**.
4. On the resulting page, click 📋 to copy your API key to the clipboard.

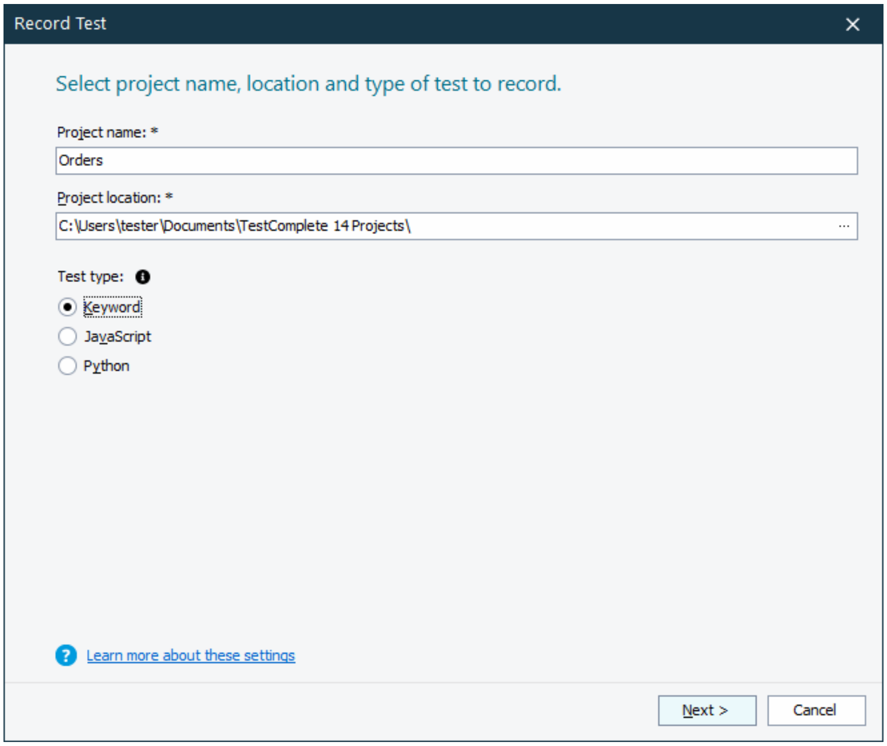


## 2. Start Test Recording

1. If a project or a project suite is open in TestComplete, close it. To do this, select **File > Close** from the TestComplete main menu.
2. Switch to the TestComplete **Start** page. If the page is hidden, select **Start Page** from the TestComplete **Help** menu.



3. On the Start page, click **Record Test**. TestComplete will show the **Record Test** wizard:



Select project name, location and type of test to record.

Project name: *
Orders

Project location: *
C:\Users\tester\Documents\TestComplete 14 Projects\

Test type:
● Keyword
○ JavaScript
○ Python

4. On the first page of the wizard, you can specify the project name, location and test type:

   In the **Project name** text box, enter *Orders*.

   Leave the default value in the **Location** text box.

   Select a test type. You can create either a keyword test, or a JavaScript or Python script test:

   - A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.

   - JavaScript and Python tests are script functions with instructions simulating user actions.

   In this tutorial we will show how to create a keyword test. Click **Keyword**.

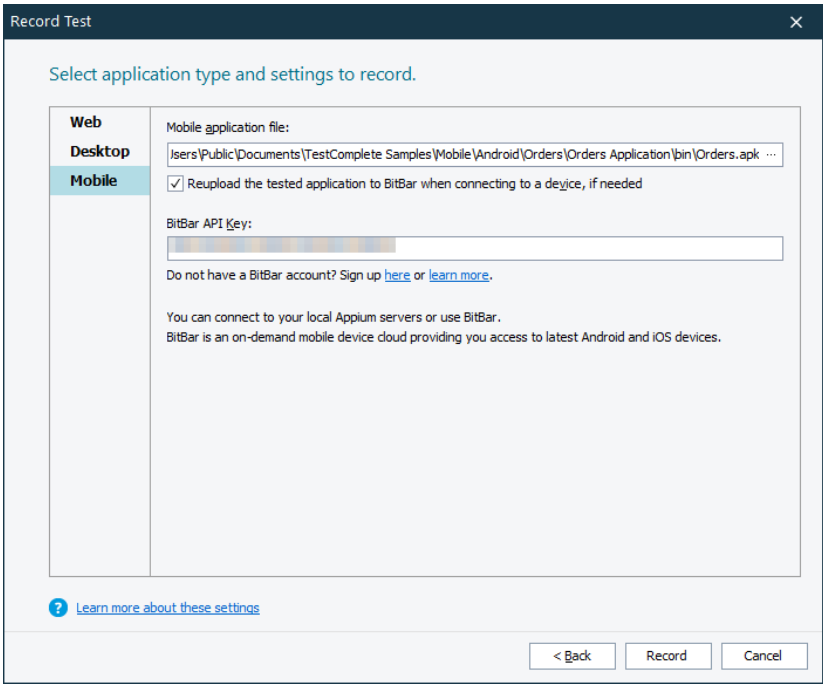
5. Click **Next** to continue.

   We will continue working with the wizard to add our tested application to the project.

## 3. Define a Tested Application

To create a test for an application running on a mobile device, you need to deliver the application to the device. If you use BitBar as your mobile device cloud provider, upload the application file (*Orders.apk* in our case) to your account file library. The easiest way to upload the file from TestComplete is to add it to your TestComplete project, for example, when you start test recording:

1. The wizard shows the second page where you can choose your tested application:



Select application type and settings to record.

Mobile application file:
Jsers\Public\Documents\TestComplete Samples\Mobile\Android\Orders\Orders Application\bin\Orders.apk

☑ Reupload the tested application to BitBar when connecting to a device, if needed

BitBar API Key: [redacted]

Do not have a BitBar account? Sign up here or learn more.

You can connect to your local Appium servers or use BitBar.
BitBar is an on-demand mobile device cloud providing you access to latest Android and iOS devices.

2. The tested Orders application is an Android application shipped as a *.apk* file, it falls under the Mobile application category.

   Click **Mobile**.

3. In the **Mobile application fil**e text box, click the ellipsis button. In the resulting **Select Tested Application** dialog, locate the *Orders.apk* file.

   Leave the **Reupload the tested application to BitBar when connecting to a device, if needed** check box selected by default.

4. If you already have a BitBar account, enter the API Key assigned to your account in the **BitBar API Key** text box.

   For the instructions on how to get the API Key, see above.

   If you have skipped step 1 of this tutorial, you can sign up for a free trial directly from the wizard by clicking the **Sign up** link in it.

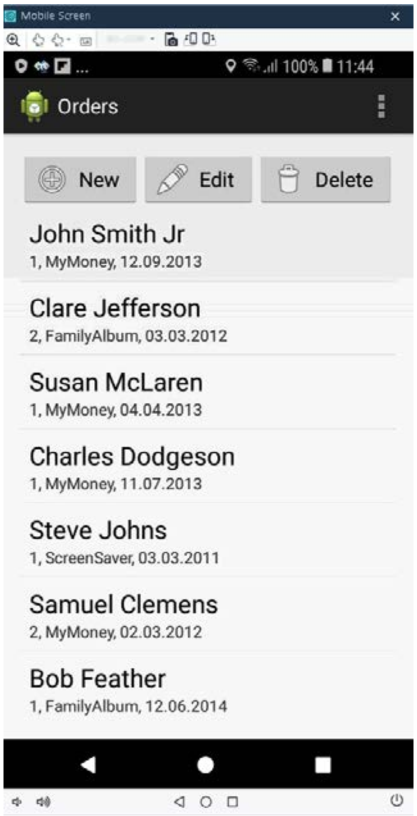
5. Click **Record** to complete the project creation and start recording.

# 4. Record a Test

**Note:** Do not switch to the TestComplete help during the recording. The recording engine traces and records all user actions, so the recorded test will contain commands that simulate “switching”.

To see the instructions, you can print them before starting the record. Or, if you have two monitors, you can move the TestComplete help system window to the other monitor.

1. Wait until the Mobile Screen window will display the initial window of the application:



John Smith Jr
1, MyMoney, 12.09.2013

Clare Jefferson
2, FamilyAlbum, 03.03.2012

Susan McLaren
1, MyMoney, 04.04.2013

Charles Dodgeson
1, MyMoney, 11.07.2013

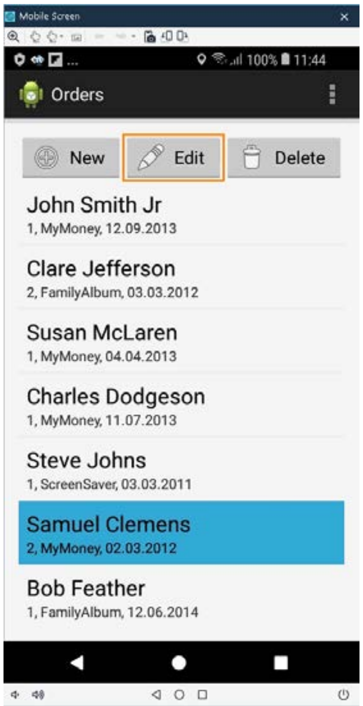
Steve Johns
1, ScreenSaver, 03.03.2011

Samuel Clemens
2, MyMoney, 02.03.2012

Bob Feather
1, FamilyAlbum, 12.06.2014

If the application won't start, launch it manually.

2. In the **Mobile Screen** window, click Samuel Clemens's order and then click the **Edit** button:



John Smith Jr
1, MyMoney, 12.09.2013

Clare Jefferson
2, FamilyAlbum, 03.03.2012

Susan McLaren
1, MyMoney, 04.04.2013

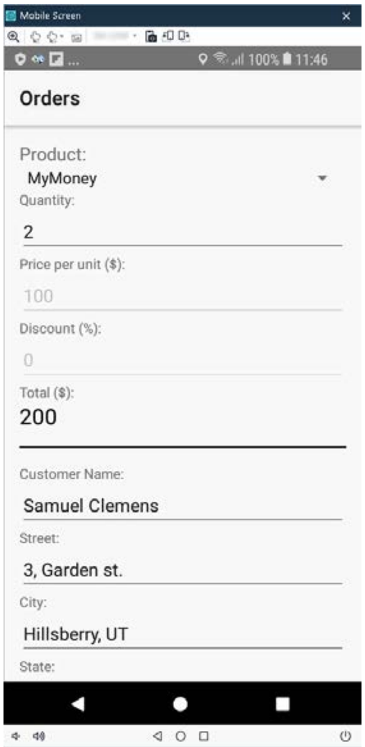
Charles Dodgeson
1, MyMoney, 11.07.2013

Steve Johns
1, ScreenSaver, 03.03.2011

Samuel Clemens
2, MyMoney, 02.03.2012

Bob Feather
1, FamilyAlbum, 12.06.2014

This will switch the Orders application to the edit mode.



Orders

Product: MyMoney

Quantity: 2

Price per unit ($): 100

Discount (%): 0

Total ($): 200

Customer Name: Samuel Clemens

Street: 3, Garden st.

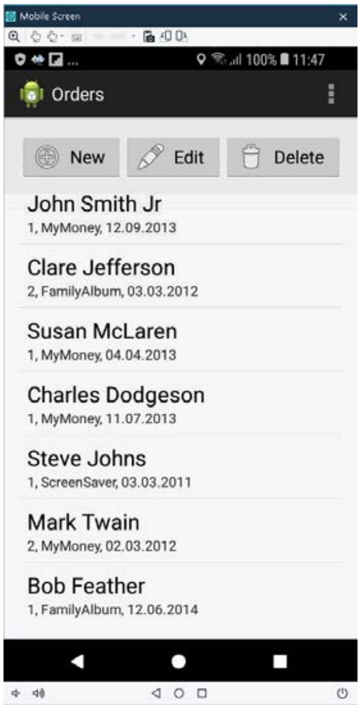
City: Hillsberry, UT

State:

**3.** Change the customer name in the order details.

Clear the Samuel Clemens text, type Mark Twain, and press Enter. Use your desktop keyboard to input text in the Mobile Screen window.

Scroll down the Edit Order page and click **OK**. This will save the order changes and return back to the orders list.



John Smith Jr
1, MyMoney, 12.09.2013

Clare Jefferson
2, FamilyAlbum, 03.03.2012

Susan McLaren
1, MyMoney, 04.04.2013

Charles Dodgeson
1, MyMoney, 11.07.2013

Steve Johns
1, ScreenSaver, 03.03.2011

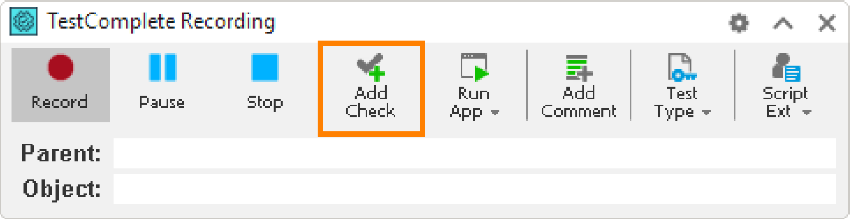
Mark Twain
2, MyMoney, 02.03.2012

Bob Feather
1, FamilyAlbum, 12.06.2014

**4.** Insert a comparison command into our test. It will verify that the application customer list contains the modified name - Mark Twain.

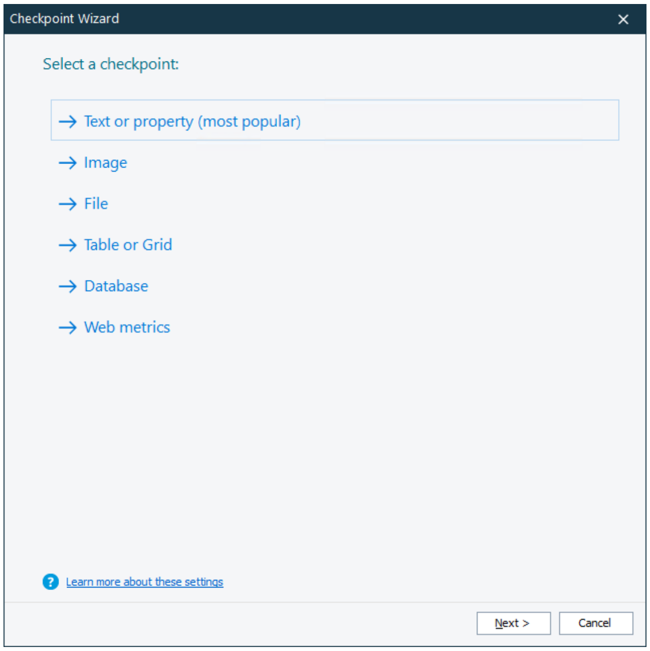
We call comparison commands checkpoints. TestComplete offers various types of checkpoints to verify various types of data. One of the most frequently used checkpoints is the Property checkpoint. You use it to check data of application controls. We will use this checkpoint in our tutorial to verify the text in the Customer Name edit box.

- Click Mark Twain’s order and then click the Edit button.
- Click ✓ Add Check on the Recording toolbar:



TestComplete will open the Checkpoint wizard. It will guide you through the process of checkpoint creation.

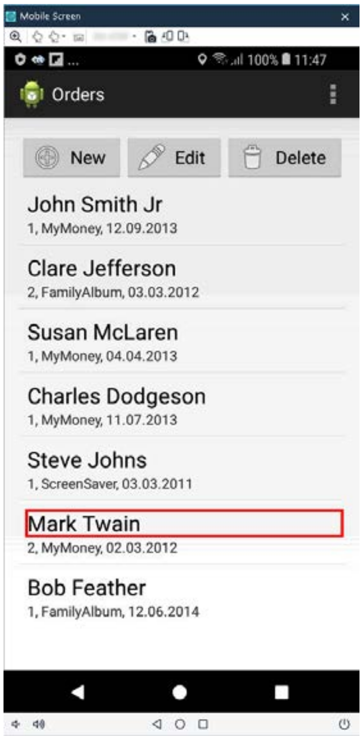
- On the first page of the wizard, click Text or property.



- Click the target glyph (⌖) with the left mouse button and keep the button pressed.

  Wait until the wizard minimizes, and then drag the icon to the Edit Order page of the Orders application. While you are dragging, TestComplete will highlight the controls and windows under the mouse cursor with a red frame.

  Release the mouse button when the target glyph is over the desired edit box, and when the button is highlighted with the red frame:



John Smith Jr
1, MyMoney, 12.09.2013

Clare Jefferson
2, FamilyAlbum, 03.03.2012

Susan McLaren
1, MyMoney, 04.04.2013

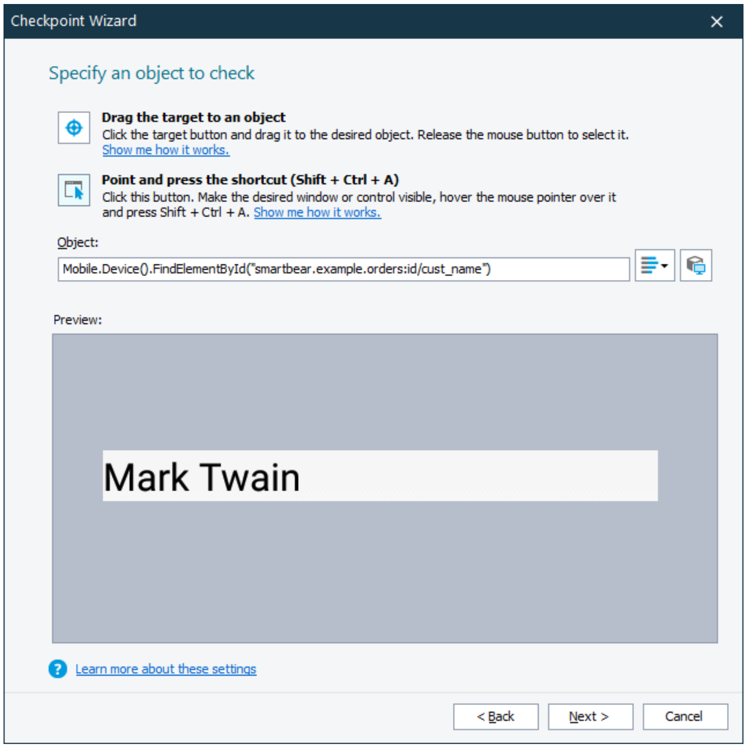
Charles Dodgeson
1, MyMoney, 11.07.2013

Steve Johns
1, ScreenSaver, 03.03.2011

Mark Twain
2, MyMoney, 02.03.2012

Bob Feather
1, FamilyAlbum, 12.06.2014

- After you release the mouse button, TestComplete will restore the wizard and show the name of the selected object in the Object box and the image of the object below it:



Specify an object to check

**Drag the target to an object**
Click the target button and drag it to the desired object. Release the mouse button to select it.
Show me how it works.

**Point and press the shortcut (Shift + Ctrl + A)**
Click this button. Make the desired window or control visible, hover the mouse pointer over it and press Shift + Ctrl + A. Show me how it works.

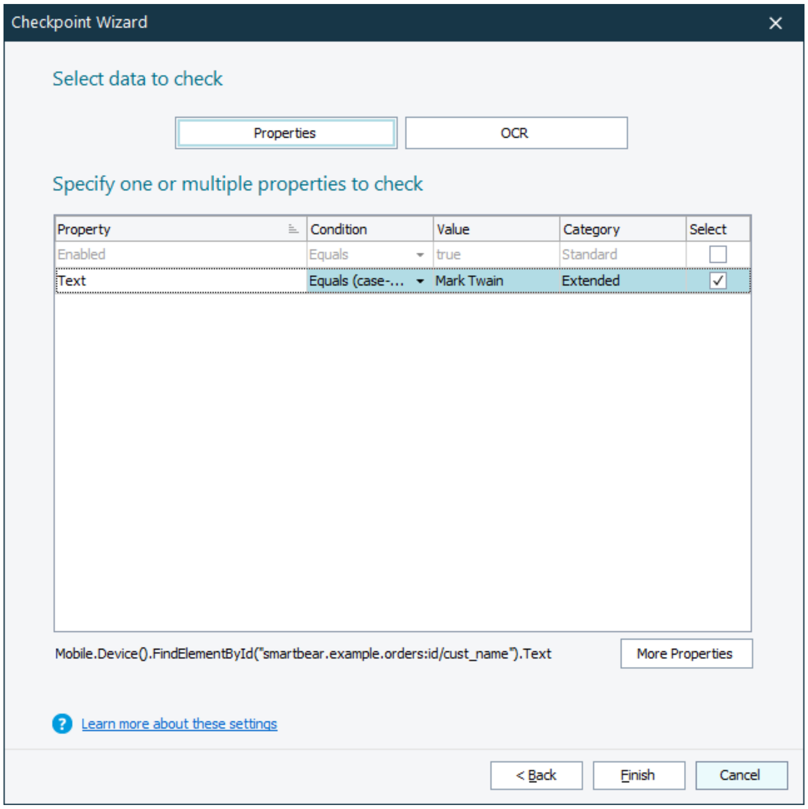
Object: Mobile.Device().FindElementById("smartbear.example.orders:id/cust_name")

Mark Twain

Learn more about these settings

Click **Next** to continue.

- The next page of the wizard shows a list of properties suggested for checking:



Select data to check

Specify one or multiple properties to check

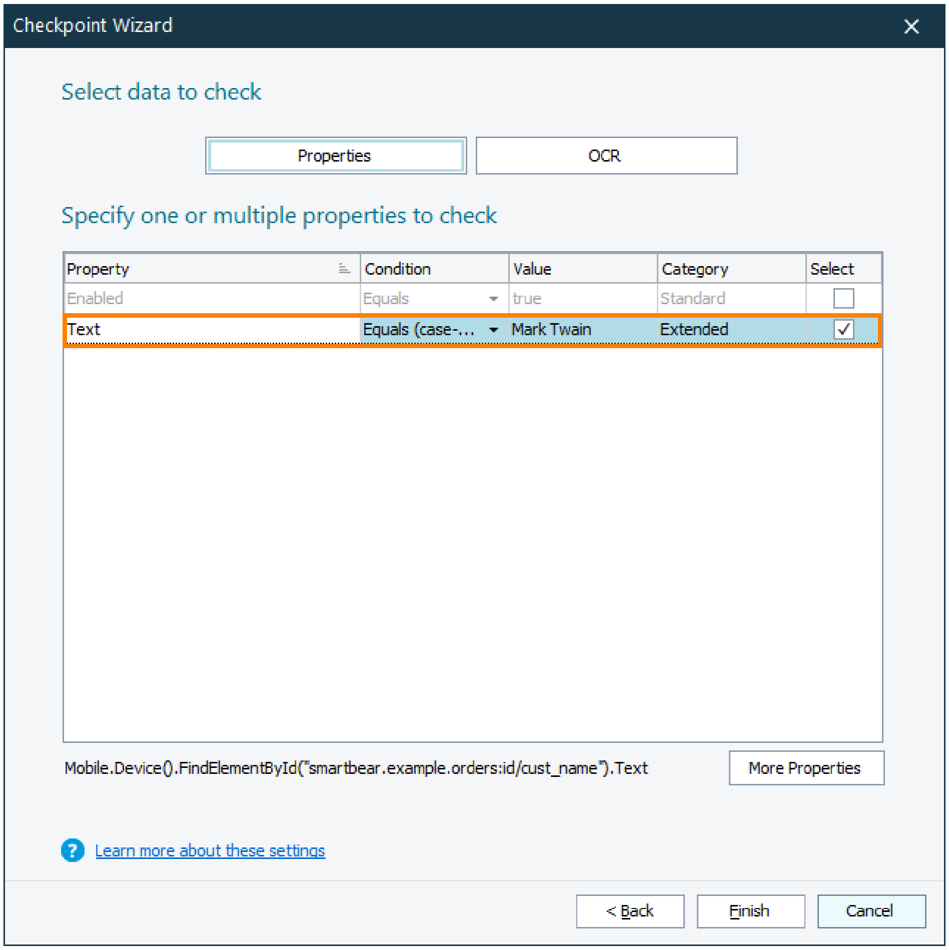
| Property | Condition | Value | Category | Select |
|---|---|---|---|---|
| Enabled | Equals | true | Standard | ☐ |
| Text | Equals (case-… | Mark Twain | Extended | ☑ |

Mobile.Device().FindElementById("smartbear.example.orders:id/cust_name").Text

Learn more about these settings

This list includes the properties provided by TestComplete and the properties defined by the tested application. To view all the available properties, you can click **More Properties**.

- In the table, make sure the **Text** property is selected. Make sure that the **Equals (case-sensitive)** condition is selected in the **Condition** column:



| Property | Condition | Value | Category | Select |
|---|---|---|---|---|
| Enabled | Equals | true | Standard | □ |
| Text | Equals (case-… | Mark Twain | Extended | ☑ |

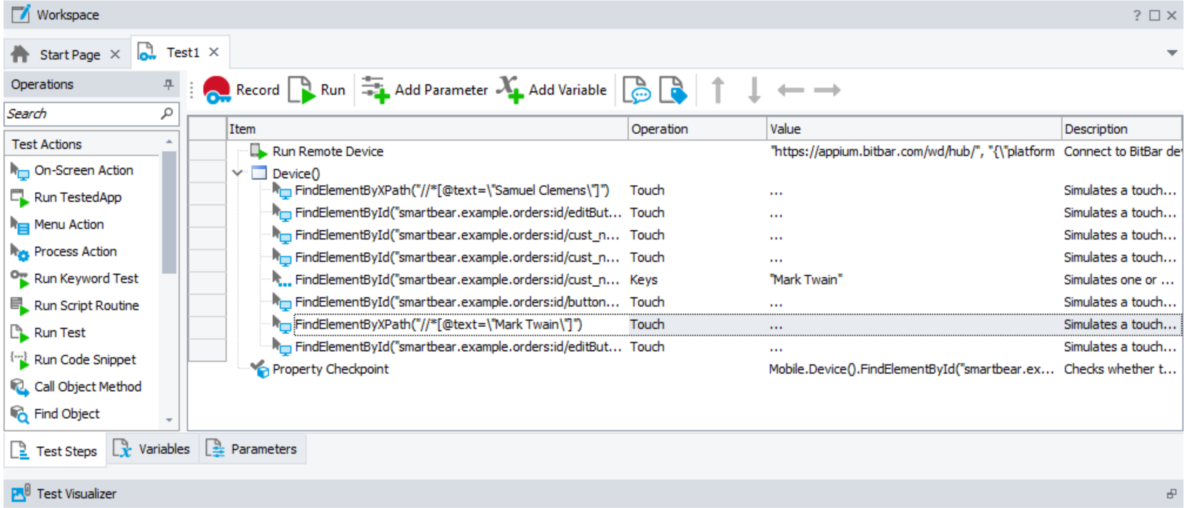
Mobile.Device().FindElementById("smartbear.example.orders:id/cust_name").Text

- Click **Finish** to complete the checkpoint creation. TestComplete will append the checkpoint command to the recorded test.

5. Click ■ **Stop** on the Recording toolbar to stop recording. TestComplete will process the recorded test commands and save them to the keyword test.

## 5. Analyze the Recorded Test

After you finish recording, TestComplete opens the recorded keyword test in the Keyword Test editor:

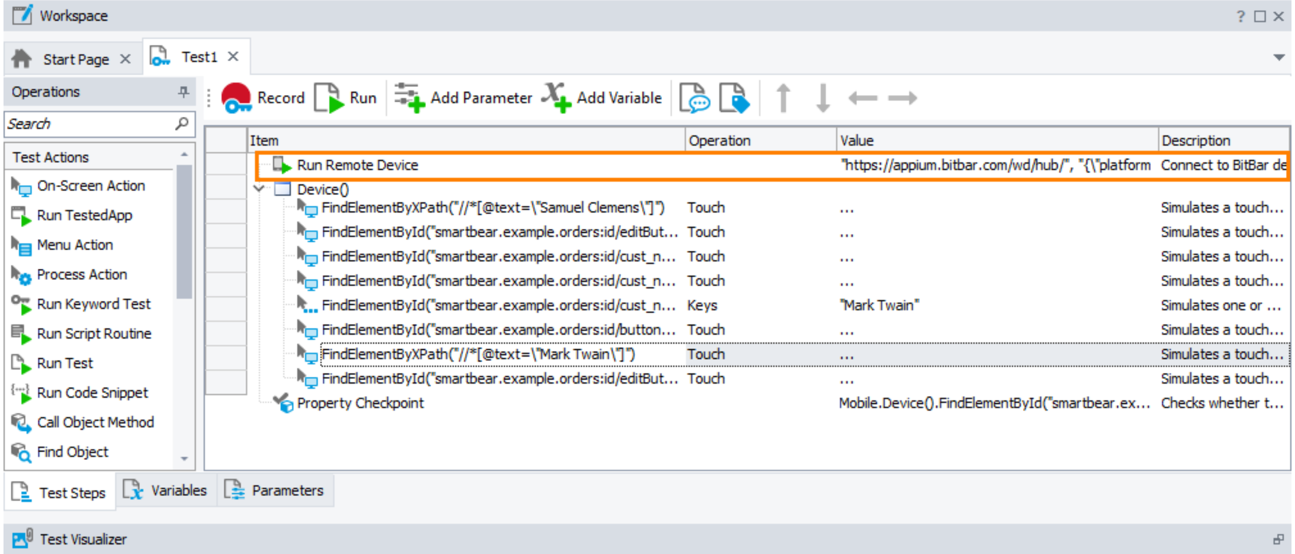


| Item | Operation | Value | Description |
|---|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platform | Connect to BitBar de |
| Device() | | | |
| FindElementByXPath("//*[@text=\"Samuel Clemens\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Keys | "Mark Twain" | Simulates one or ... |
| FindElementById("smartbear.example.orders:id/button... | Touch | ... | Simulates a touch... |
| FindElementByXPath("//*[@text=\"Mark Twain\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| Property Checkpoint | | Mobile.Device().FindElementById("smartbear.ex... | Checks whether t... |

The recorded test is similar to the test shown in the image above. Your actual test may differ from this one. For example, it may contain some unnecessary touch actions.

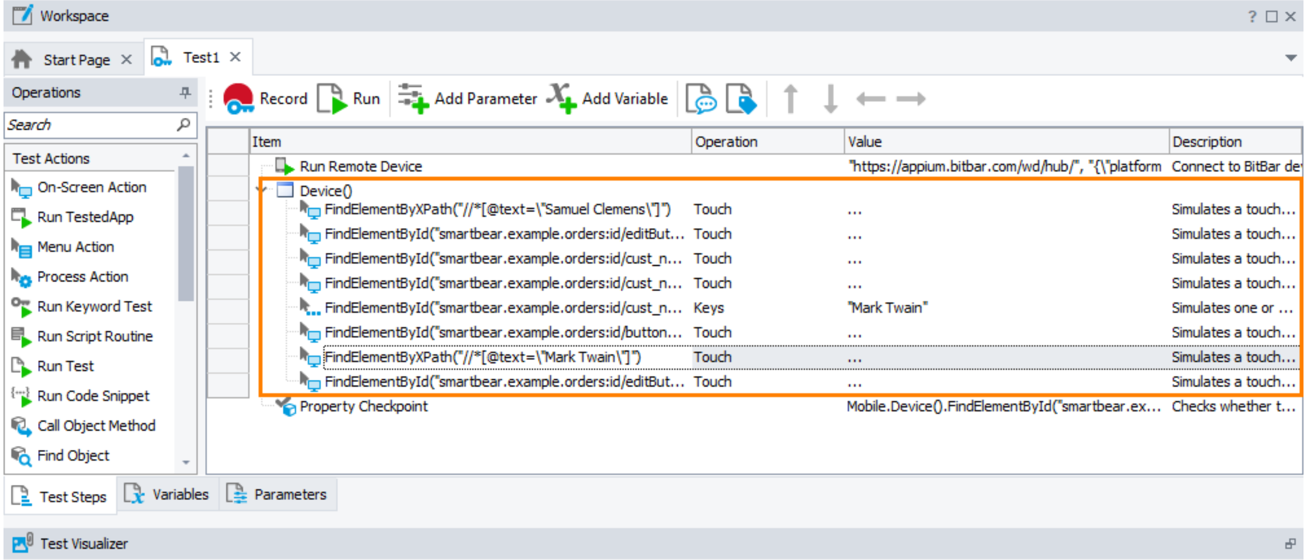
The test contains the commands that correspond to the actions you performed on the Orders application during test recording. We call these test commands operations.

The first operation in the test is Run Remote Device. It connects to a mobile device in the BitBar mobile device cloud, deploys the Orders application to the specified device, and opens a testing session for the application. The other test operations refer to this device.



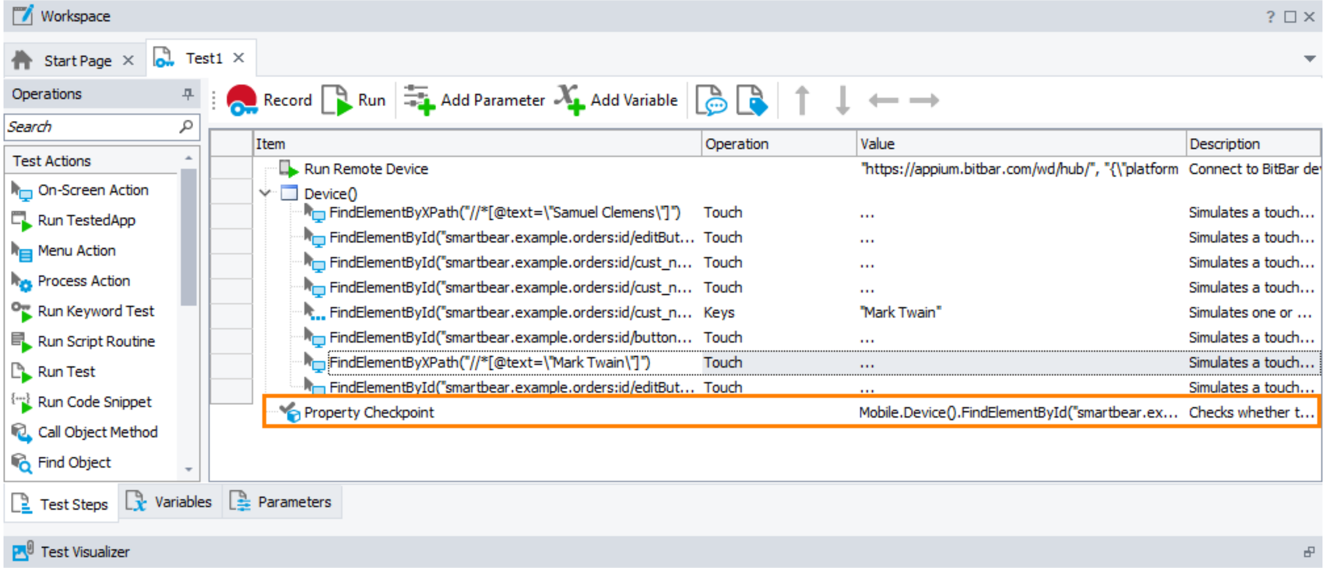
| Item | Operation | Value | Description |
|---|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platform | Connect to BitBar de |
| Device() | | | |
| FindElementByXPath("//*[@text=\"Samuel Clemens\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Keys | "Mark Twain" | Simulates one or ... |
| FindElementById("smartbear.example.orders:id/button... | Touch | ... | Simulates a touch... |
| FindElementByXPath("//*[@text=\"Mark Twain\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| Property Checkpoint | | Mobile.Device().FindElementById("smartbear.ex... | Checks whether t... |

Then there are the operations that simulate your actions with the application. These operations click the Edit button, select an item in the orders list, change the value of the text field, save the changes, and click the Done button.



| Item | Operation | Value | Description |
|---|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platform | Connect to BitBar de |
| Device() | | | |
| FindElementByXPath("//*[@text=\"Samuel Clemens\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Keys | "Mark Twain" | Simulates one or ... |
| FindElementById("smartbear.example.orders:id/button... | Touch | ... | Simulates a touch... |
| FindElementByXPath("//*[@text=\"Mark Twain\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| Property Checkpoint | | Mobile.Device().FindElementById("smartbear.ex... | Checks whether t... |

Finally, there is the comparison operation that you added during test recording:



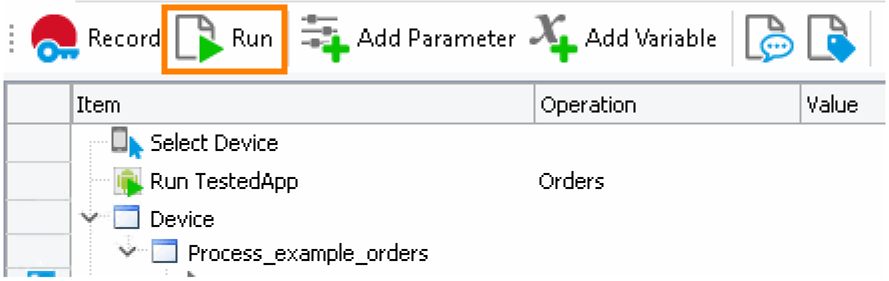
| Item | Operation | Value | Description |
|---|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platform | Connect to BitBar de |
| Device() | | | |
| FindElementByXPath("//*[@text=\"Samuel Clemens\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/cust_n... | Keys | "Mark Twain" | Simulates one or ... |
| FindElementById("smartbear.example.orders:id/button... | Touch | ... | Simulates a touch... |
| FindElementByXPath("//*[@text=\"Mark Twain\"]") | Touch | ... | Simulates a touch... |
| FindElementById("smartbear.example.orders:id/editBut... | Touch | ... | Simulates a touch... |
| Property Checkpoint | | Mobile.Device().FindElementById("smartbear.ex... | Checks whether t... |

Each operation specifies the simulated action and the object over which the action was recorded. To locate the objects, the operations use various `FindElement` methods and search expressions based on the accessibility information provided by the application.

# 6. Run the Test

## Run the Test

To run the test, click Run on the test editor toolbar:



| Item | Operation | Value |
|---|---|---|
| Select Device | | |
| Run TestedApp | Orders | |
| Device | | |
| Process_example_orders | | |

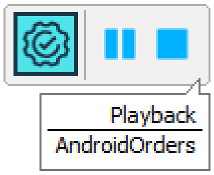
TestComplete will connect to the specified device in the BitBar device cloud, launch the Orders application on the device, and play back the recorded actions over it: open Samuel Clemens's order and change the customer name to Mark Twain.

> **Note:** Do not move the mouse or press any keys during the test execution. Your actions may interfere with the actions simulated by TestComplete and the test execution may go wrong.

After the test execution is over, TestComplete will restore its windows and display test results. In the next step, we will analyze them.

Some notes about the test run:

- During the test execution, TestComplete displays an indicator in the top right corner of the screen:

  Playback
  AndroidOrders

  The indicator displays messages informing you about simulated test actions.

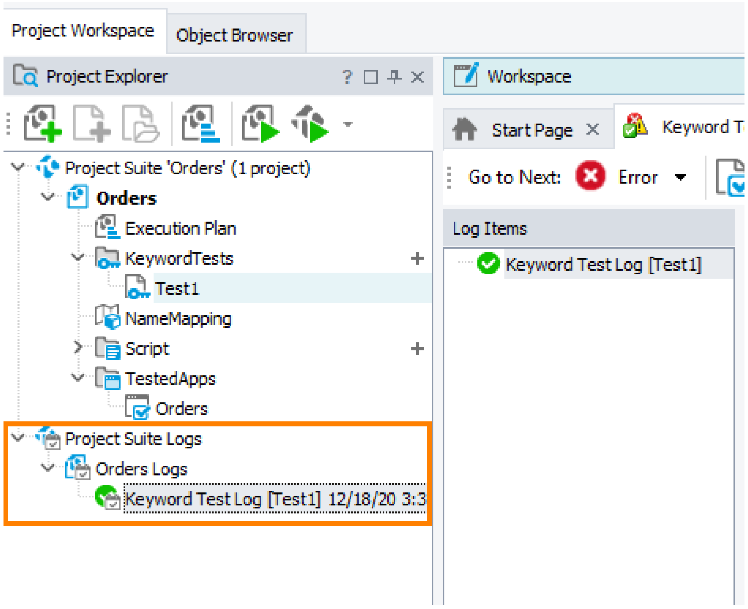
- TestComplete executes test commands until the test ends. You can stop the execution at any time by pressing ■ **Stop** on the Test Engine toolbar or in the indicator, or by selecting **Test | Stop** from TestComplete main menu.

  You can pause the test execution by clicking ❚❚ **Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check test variables and objects using TestComplete Watch List or Locals panel or the **Evaluate** dialog (see *Debugging Tests* in TestComplete Help).

For complete information on running tests in TestComplete, on project settings that affect the runs and on the test execution, see *Running Tests* section in TestComplete Help.

## 7. Analyze Test Results

After the test finishes, TestComplete shows the test log with the results of all test operations. The results of our test look as follows:

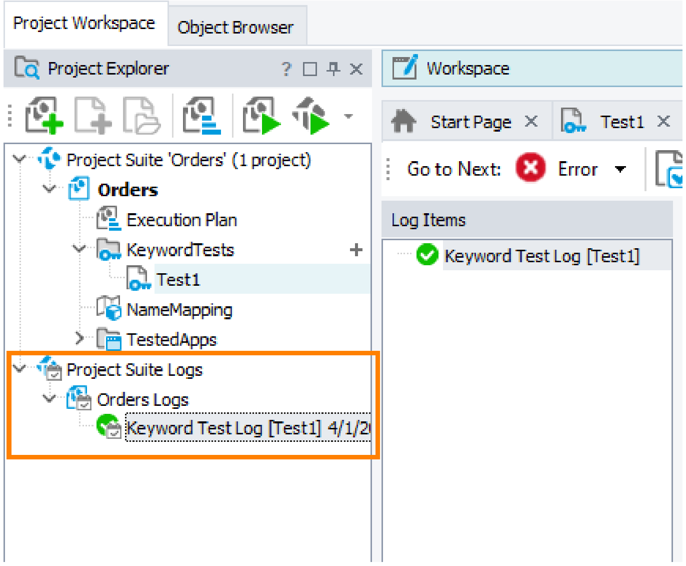


The log contains messages of different types: actions, events, checkpoints, and so on. You can filter the messages using the toolbar above the message list. If you double-click a log message, TestComplete will navigate to the test operation that posted this message. This is useful when you need to know which operation caused an error.

The **Picture** panel contains images that show the application state when the selected test command was executed. The images help you understand what happened in the application during the run, and find errors faster.

The **Details** panel contains details on the performed operation. For example, Details for property checkpoints includes the check type (equals, contains, matches a regular expression, and so on) and other details.

All logs are kept under **Project Suite Logs** > **ProjectName Logs** in the **Project Explorer**, so that you can review the previous logs.



## Resolving Errors

Your test may fail. There can be several possible reasons for this. For example, developers could change the application behavior, the recognition attributes of application controls could change and make the test engine fail to find the needed objects, and so on.

One of the most typical reasons that novice users face is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance:

- If the tested application had been running before you recorded a test, it must also be running before you run the test.
- If test actions were performed on a particular screen of the application, you should open the screen when running the test.
- If you edited any data in the application, and then saved it, you need to revert the changes.

Sometimes, the test engine fails to find an application object because it needs to wait for a specific application response. To solve this problem, insert a delay command and specify for how long the text execution should be paused:

- In keyword tests, use the **Delay** operation with the *Delay Time* parameter specified.
- In scripts, use the `aqUtils.Delay` method with the loop body.

For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* and *Debugging Tests* sections in TestComplete Help.

# Testing iOS Applications

This tutorial explains how to test *iOS applications* with TestComplete. It assumes that you are familiar with general principles of automated testing and have minimal knowledge of the TestComplete IDE.

> ❗ If you are a novice user, we recommend that you read an introduction to automated testing described above.

## Requirements

You can follow this tutorial and create a test for an iOS application only if you have the following:

- A Windows computer with TestComplete installed and an active license for the TestComplete Mobile module.
- Access to an iOS device controlled by an Appium server. It can be:
    - The mobile device cloud provided by BitBar. We will use it in this tutorial. We will show how to create a free trial BitBar account and access the mobile devices it provides.

    – or –

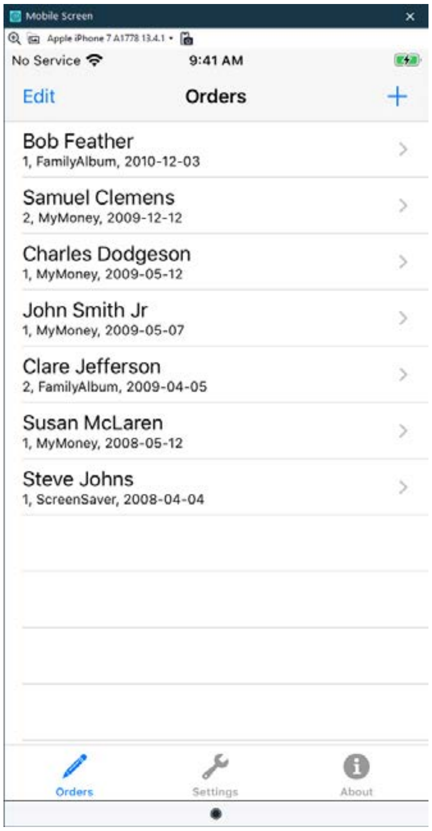
    - A private Appium server running on your local computer or on a remote computer in your local network. Using a private Appium server is beyond this tutorial scope.
- A Mac computer with Xcode, iOS SDK, and an iOS development license to compile the sample Orders application.

## In this tutorial

We are going to create a trial BitBar account, connect to an iOS mobile device in the BitBar device cloud, deploy a tested application to it, create and run a simple test, and analyze the results.

### About Tested Application

In our explanations, we will use the iOS version of the Orders application. This application lets you manage a table of purchase orders: you can view the list of existing orders, modify, or remove them, as well as add new orders to the list:



Bob Feather
1, FamilyAlbum, 2010-12-03

Samuel Clemens
2, MyMoney, 2009-12-12

Charles Dodgeson
1, MyMoney, 2009-05-12

John Smith Jr
1, MyMoney, 2009-05-07

Clare Jefferson
2, FamilyAlbum, 2009-04-05

Susan McLaren
1, MyMoney, 2008-05-12

Steve Johns
1, ScreenSaver, 2008-04-04

### To get the application

1. Download the TestComplete Samples installation package from our website:

   ➩ https://support.smartbear.com/testcomplete/downloads/samples/

2. Run the installation.

3. The sample will be installed to the *<TestComplete Samples>\Mobile\iOS\Orders\* folder.

The application should be compiled on your Mac.

## 1. Sign Up for a Free BitBar Account

In TestComplete, you can connect to iOS devices controlled by an Appium server and create and run automated tests on these devices. In this tutorial, we will use a ready-to-use solution provided by BitBar, a cloud-based mobile testing service by SmartBear.

We will show how to create a free trial account and use it in the tutorial. Requesting a trial takes only several moments and requires only an active email.

You can sign up for the trial before you start the tutorial, or you can skip it and do it later when you start the actual test recording. Both ways are acceptable.

### 1. Sign up for a trial BitBar account

3. Click the following link to sign up for a free trial on the BitBar website:

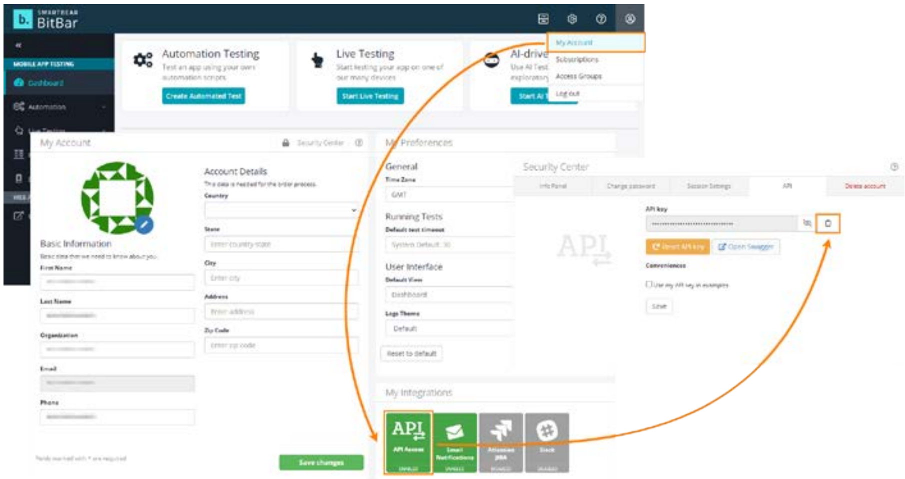
   ➩ bitbar.com/signup

4. Follow the web form instructions.

### 2. Get the BitBar API Key

To access BitBar resources from TestComplete tests, the **BitBar API Key** is used. To get the key:

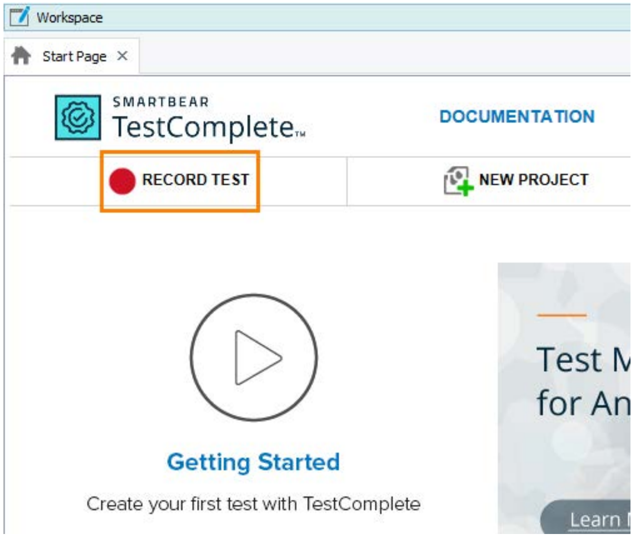
5. Log in to BitBar with your BitBar account.
6. Click 👤 > **My Account** at the top right of the page.
7. In the My Integrations section, click **API**.
8. On the resulting page, click 📋 to copy your API key to the clipboard.



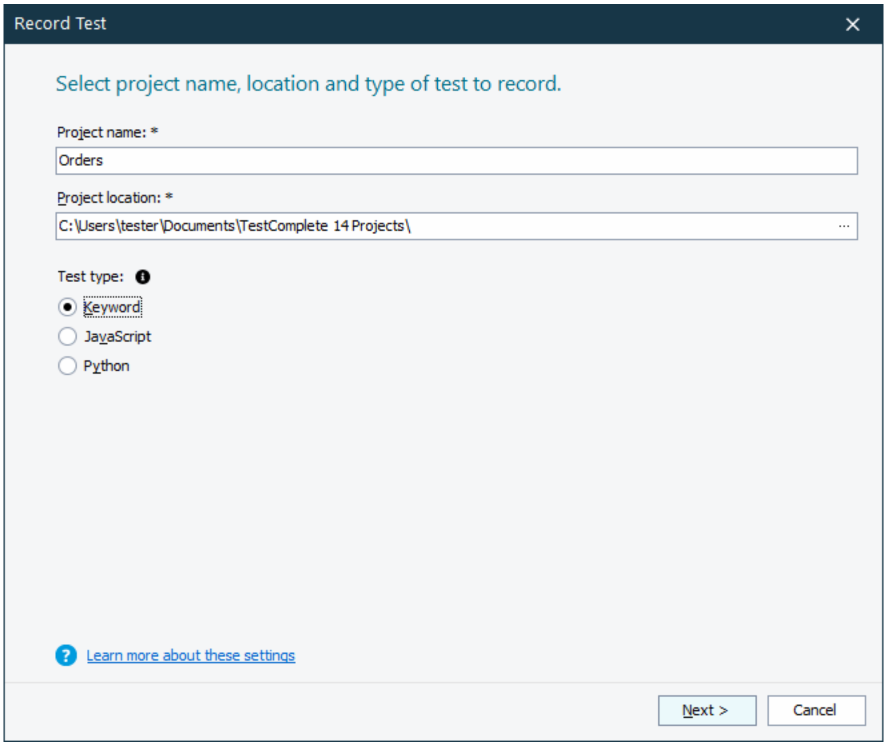
## 2. Start Test Recording

6. If a project or a project suite is open in TestComplete, close it. To do this, select **File > Close** from the TestComplete main menu.

7. Switch to the TestComplete **Start** page. If the page is hidden, select **Start Page** from the TestComplete **Help** menu.



8. On the Start page, click 🔴 **Record Test**. TestComplete will show the **Record Test** wizard:



Record Test

Select project name, location and type of test to record.

Project name: * Orders

Project location: * C:\Users\tester\Documents\TestComplete 14 Projects\

Test type:

● Keyword
○ JavaScript
○ Python

9. On the first page of the wizard, you can specify the project name, location and test type:

   In the **Project name** text box, enter *Orders*.

   Leave the default value in the **Location** text box.

   Select a test type. You can create either a keyword test, or a JavaScript or Python script test:

   - A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.

- JavaScript and Python tests are script functions with instructions simulating user actions.

  In this tutorial we will show how to create a keyword test. Click **Keyword**.

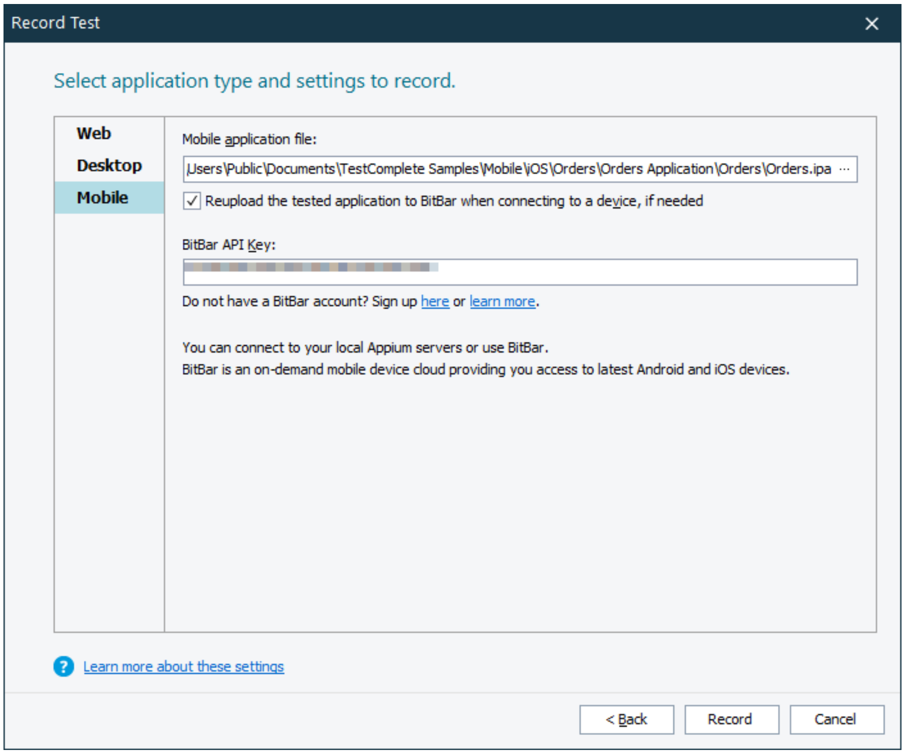
10. Click **Next** to continue.

    We will continue working with the wizard to add our tested application to the project.

## 3. Define a Tested Application

To create a test for an application running on a mobile device, you need to deliver the application to the device. If you use BitBar as your mobile device cloud provider, upload the application file (*Orders.ipa* in our case) to your account file library. The easiest way to upload the file from TestComplete is to add it to your TestComplete project, for example, when you start test recording:

1. The wizard shows the second page where you can choose your tested application:



Record Test

Select application type and settings to record.

Mobile application file: |Users\Public\Documents\TestComplete Samples\Mobile\iOS\Orders\Orders Application\Orders\Orders.ipa

☑ Reupload the tested application to BitBar when connecting to a device, if needed

BitBar API Key: [redacted]

Do not have a BitBar account? Sign up here or learn more.

You can connect to your local Appium servers or use BitBar.
BitBar is an on-demand mobile device cloud providing you access to latest Android and iOS devices.

Learn more about these settings

2. The tested Orders application is an iOS application shipped as a *.ipa* file, it falls under the Mobile application category.

   Click **Mobile**.

3. In the **Mobile application fil**e text box, click the ellipsis button. In the resulting **Select Tested Application** dialog, locate the *Orders.ipa* file.

   Leave the **Reupload the tested application to BitBar when connecting to a device, if needed** check box selected by default.

4. If you already have a BitBar account, enter the API Key assigned to your account in the **BitBar API Key** text box.

For the instructions on how to get the API Key, see above.

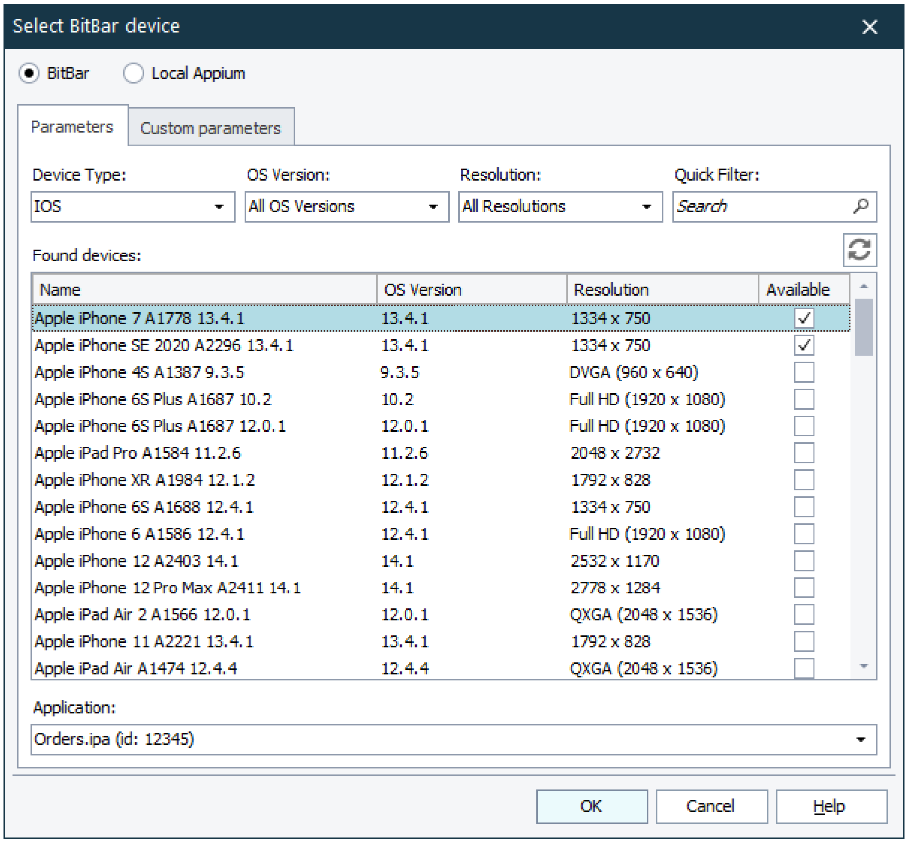
If you have skipped step 1 of this tutorial, you can sign up for a free trial directly from the wizard by clicking the **Sign up** link in it.

5. Click **Record** to complete the project creation and start recording.

## 4. Connect to a Device and Open a Testing Session

Before the recording starts, select a device, on which you want to record a test, and open a testing session for the Orders application:

1. TestComplete shows the **Select BitBar Device** dialog. It lists all mobile devices that BitBar provides:



Select BitBar device

● BitBar ○ Local Appium

Device Type: IOS

OS Version: All OS Versions

Resolution: All Resolutions

Quick Filter: Search

Found devices:

| Name | OS Version | Resolution | Available |
|---|---|---|---|
| Apple iPhone 7 A1778 13.4.1 | 13.4.1 | 1334 x 750 | ☑ |
| Apple iPhone SE 2020 A2296 13.4.1 | 13.4.1 | 1334 x 750 | ☑ |
| Apple iPhone 4S A1387 9.3.5 | 9.3.5 | DVGA (960 x 640) | □ |
| Apple iPhone 6S Plus A1687 10.2 | 10.2 | Full HD (1920 x 1080) | □ |
| Apple iPhone 6S Plus A1687 12.0.1 | 12.0.1 | Full HD (1920 x 1080) | □ |
| Apple iPad Pro A1584 11.2.6 | 11.2.6 | 2048 x 2732 | □ |
| Apple iPhone XR A1984 12.1.2 | 12.1.2 | 1792 x 828 | □ |
| Apple iPhone 6S A1688 12.4.1 | 12.4.1 | 1334 x 750 | □ |
| Apple iPhone 6 A1586 12.4.1 | 12.4.1 | Full HD (1920 x 1080) | □ |
| Apple iPhone 12 A2403 14.1 | 14.1 | 2532 x 1170 | □ |
| Apple iPhone 12 Pro Max A2411 14.1 | 14.1 | 2778 x 1284 | □ |
| Apple iPad Air 2 A1566 12.0.1 | 12.0.1 | QXGA (2048 x 1536) | □ |
| Apple iPhone 11 A2221 13.4.1 | 13.4.1 | 1792 x 828 | □ |
| Apple iPad Air A1474 12.4.4 | 12.4.4 | QXGA (2048 x 1536) | □ |

Application: Orders.ipa (id: 12345)

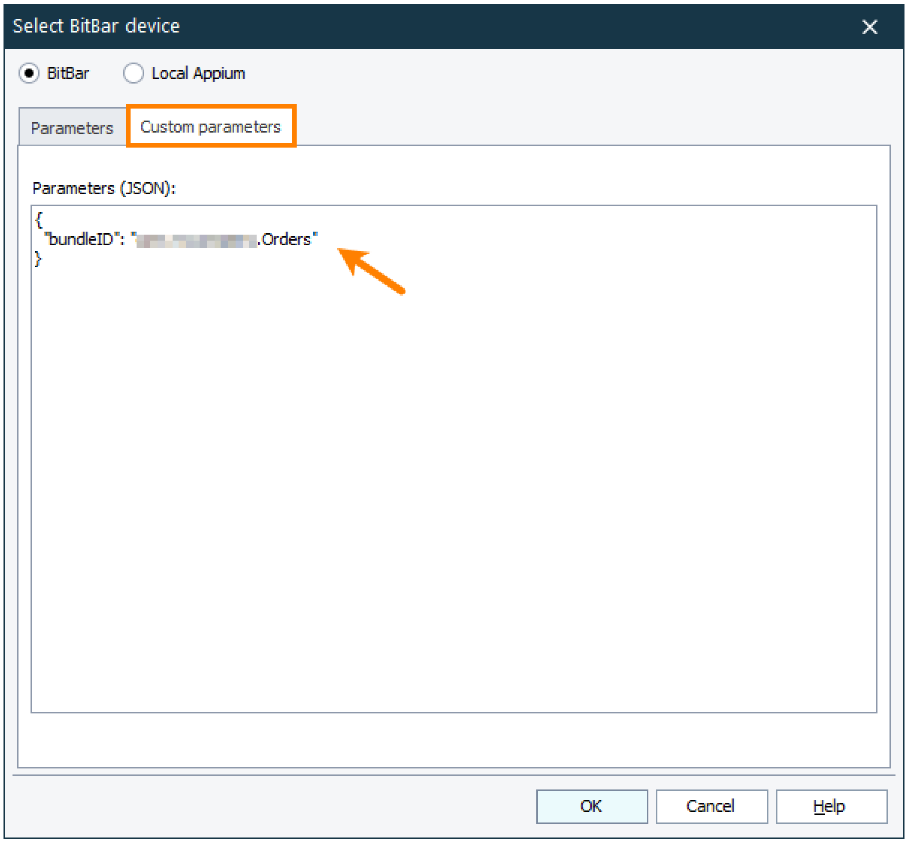
2. Select any of the available iOS devices — check the **Available** check box next to each device. There are few devices available for trial users. We will use iPhone 7 in this tutorial.

   To find a suitable device faster, you can filter out the list of devices by their platform, version, or name.

3. To open a testing session for the *Orders* application, it must be uploaded to the target device. If you have specified a valid *.ipa* file on the previous step, TestComplete will upload the file automatically.

4. For the tested application to start on the selected device automatically when the recording begins, specify the bundle ID of the application. It is the bundle ID that was used to compile the specified .ipa file.

Switch to the **Custom Parameters** tab and add the `bundleID` capability to the list of desired capabilities:



● BitBar ○ Local Appium

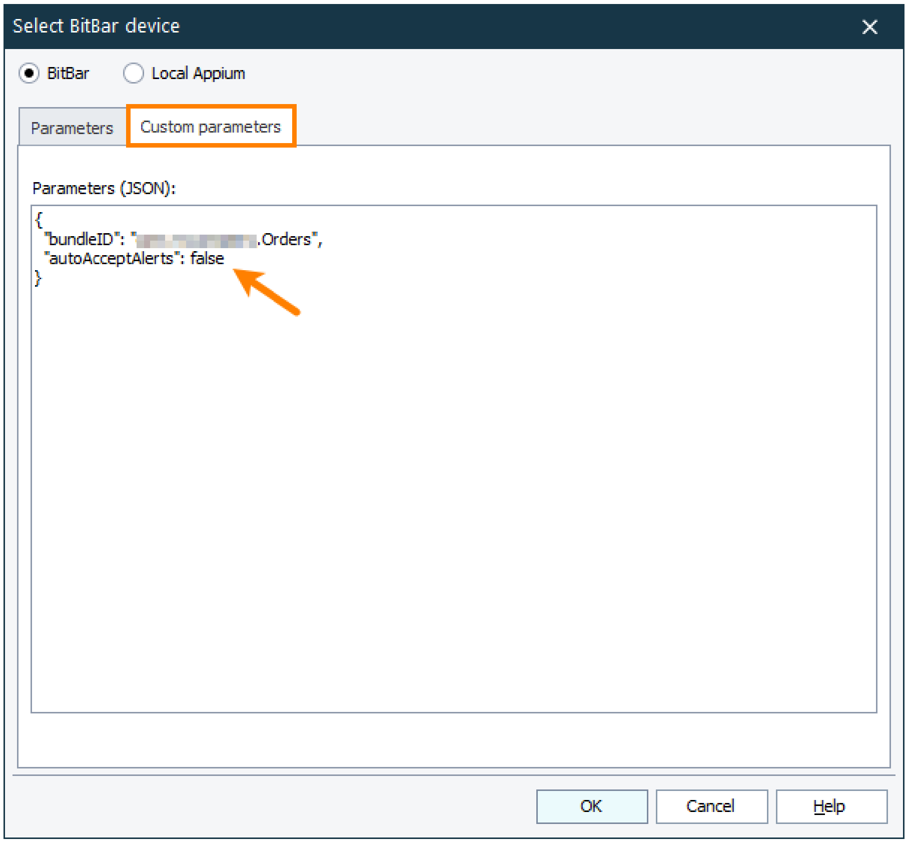
Parameters (JSON):

```
{
 "bundleID": "[redacted].Orders"
}
```

If you skip this step, you will have to launch the application manually.

5. By default, mobile devices running in the BitBar cloud are configured to automatically close all notifications, messages, and alerts. In this tutorial, we are going to record the user interactions with the alert that the Orders application shows, therefore, let's configure the device not to handle alerts automatically.

On the **Custom Parameters** tab, add the `autoAcceptAlerts` capability set to false to the list of desired capabilities:



```
{
 "bundleID": "[redacted].Orders",
 "autoAcceptAlerts": false
}
```

6. Click **OK** to proceed.

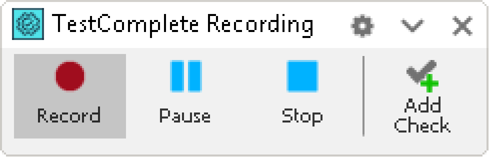
TestComplete will connect to the selected device, install the Orders application to it and open a testing session for the application. The Mobile Screen window of TestComplete will show the screen of the connected device. To record user actions over a tested application, interact with the application in that window.

## 5. Record a Test

Do not switch to TestComplete Help during test recording. The recording engine traces and records all user actions, so the recorded test will contain the commands that simulate this action.

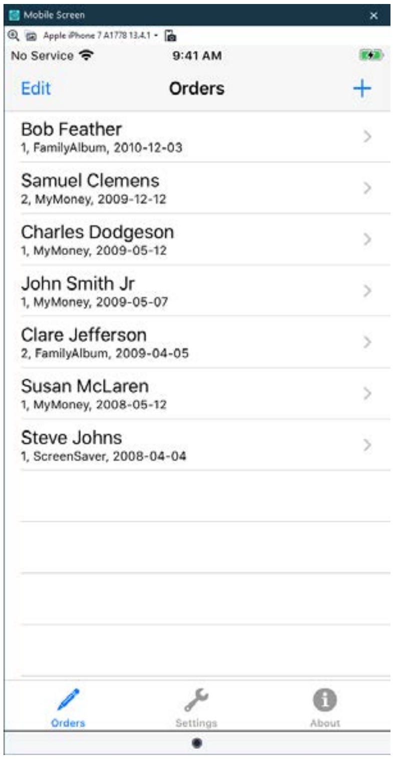
To see the instructions, you can print them before starting the record. Or, if you have two monitors, you can move the window of TestComplete help system to the other monitor.

1. TestComplete will start test recording, switch to the recording mode and show the **Recording** toolbar on screen:



2. Wait until the **Mobile Screen** window will display the initial window of the application:

**Note**: If you’ve skipped specifying the `bundleID` capability, the application won’t start automatically. Launch it manually.



Bob Feather
1, FamilyAlbum, 2010-12-03

Samuel Clemens
2, MyMoney, 2009-12-12

Charles Dodgeson
1, MyMoney, 2009-05-12

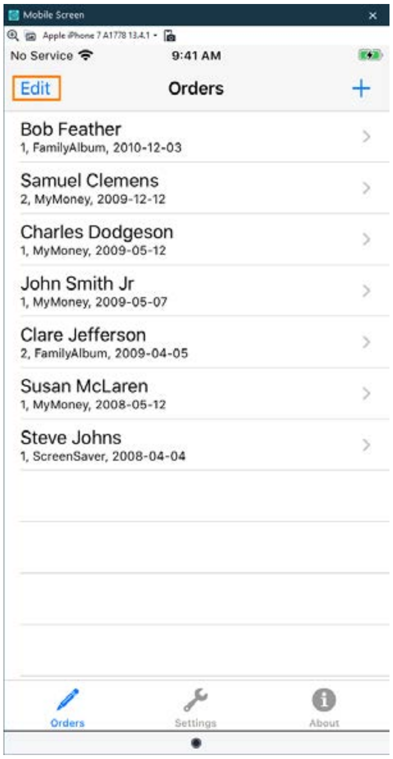
John Smith Jr
1, MyMoney, 2009-05-07

Clare Jefferson
2, FamilyAlbum, 2009-04-05

Susan McLaren
1, MyMoney, 2008-05-12

Steve Johns
1, ScreenSaver, 2008-04-04

3. In the Mobile Screen window, click the **Edit** button:



Bob Feather
1, FamilyAlbum, 2010-12-03

Samuel Clemens
2, MyMoney, 2009-12-12

Charles Dodgeson
1, MyMoney, 2009-05-12

John Smith Jr
1, MyMoney, 2009-05-07

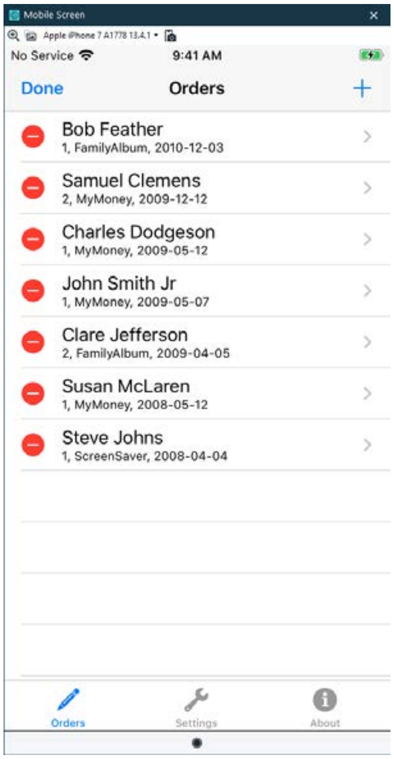
Clare Jefferson
2, FamilyAlbum, 2009-04-05

Susan McLaren
1, MyMoney, 2008-05-12

Steve Johns
1, ScreenSaver, 2008-04-04

This will switch the Orders application to the edit mode.

4. Click the second order in the list made by Samuel Clemens. This will display the Edit Order panel:



Bob Feather
1, FamilyAlbum, 2010-12-03

Samuel Clemens
2, MyMoney, 2009-12-12

Charles Dodgeson
1, MyMoney, 2009-05-12

John Smith Jr
1, MyMoney, 2009-05-07

Clare Jefferson
2, FamilyAlbum, 2009-04-05

Susan McLaren
1, MyMoney, 2008-05-12

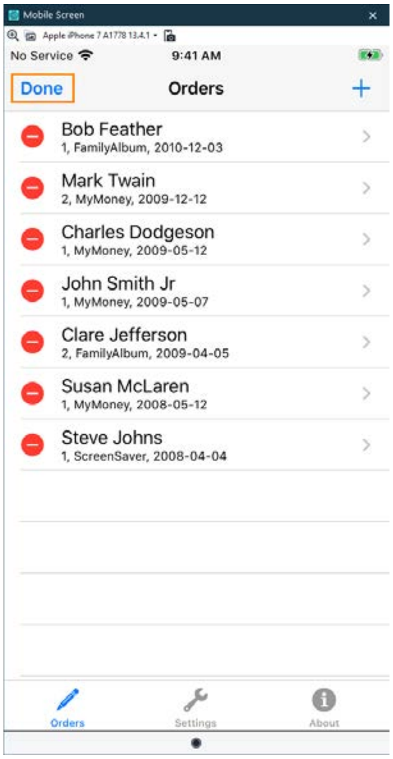
Steve Johns
1, ScreenSaver, 2008-04-04

5. Change the customer name in the order details.

Clear the *Samuel Clemens* text, type *Mark Twain*, and press **Enter**. Use your desktop keyboard to input text in the Mobile Screen window.

6. Click the **Save** button in the Edit Order panel and confirm the changes.
7. If you are recording the test on an iPhone device (whose display shows only one panel at a time), you need to touch the Orders button in the navigation bar. This will take you back to the Orders List panel.

   iPad devices display both panels simultaneously, so, there is no need to perform such an action.
8. Click **Done** to switch off the edit mode:



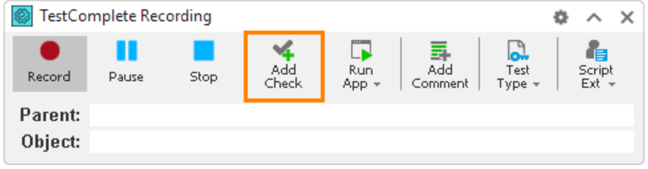
Bob Feather
1, FamilyAlbum, 2010-12-03
Mark Twain
2, MyMoney, 2009-12-12
Charles Dodgeson
1, MyMoney, 2009-05-12
John Smith Jr
1, MyMoney, 2009-05-07
Clare Jefferson
2, FamilyAlbum, 2009-04-05
Susan McLaren
1, MyMoney, 2008-05-12
Steve Johns
1, ScreenSaver, 2008-04-04

The orders list will return to normal mode.

9. Let’s insert a comparison command into our test. It will verify that the Orders button at the bottom of the application screen is selected.

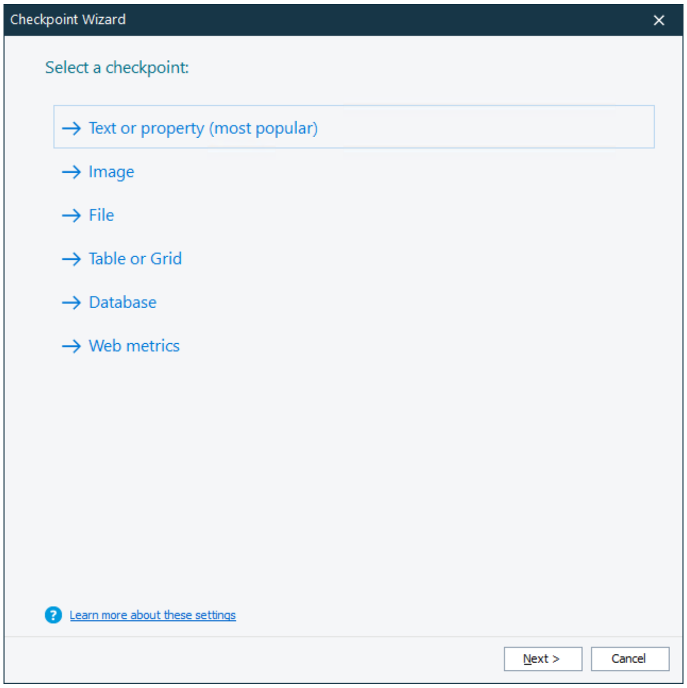
   We call comparison commands **checkpoints**. TestComplete offers various types of checkpoints to verify various types of data. One of the most frequently used checkpoints is the **Property** checkpoint. You use it to check data of application controls. We will use this checkpoint in our tutorial.

   - Click **Add Check** on the Recording toolbar:



TestComplete will open the **Checkpoint** wizard. It will guide you through the process of checkpoint creation.

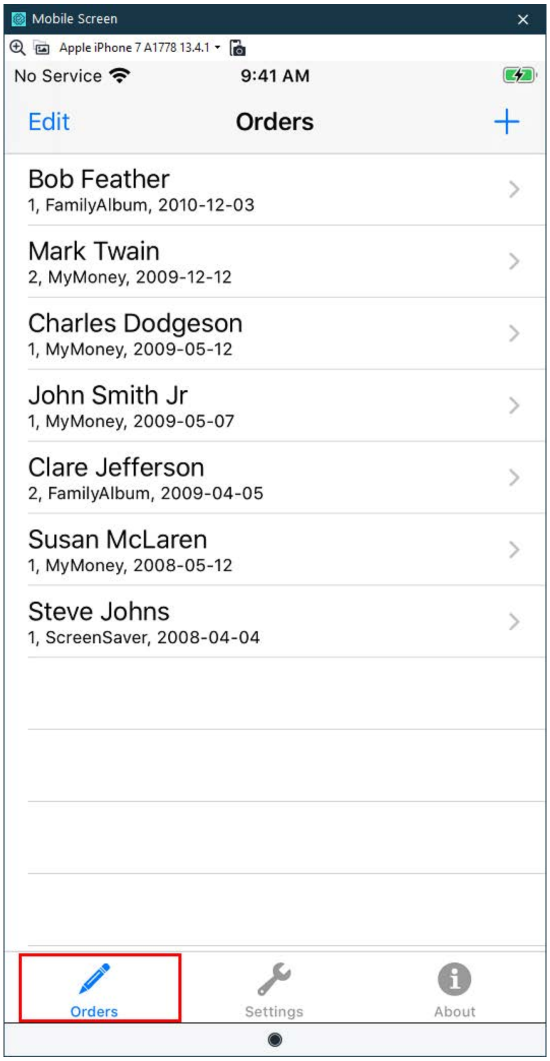
- On the first page of the wizard, click **Text or property**.



Click the target glyph (⊕) with the left mouse button and keep the button pressed.

Wait until the wizard minimizes, and then drag the icon to the orders list of the Orders application. While you are dragging, TestComplete will highlight the controls and windows under the mouse cursor with a red frame.

Release the mouse button when the target glyph is over the button, and when the button is highlighted with the red frame:



Bob Feather
1, FamilyAlbum, 2010-12-03

Mark Twain
2, MyMoney, 2009-12-12

Charles Dodgeson
1, MyMoney, 2009-05-12

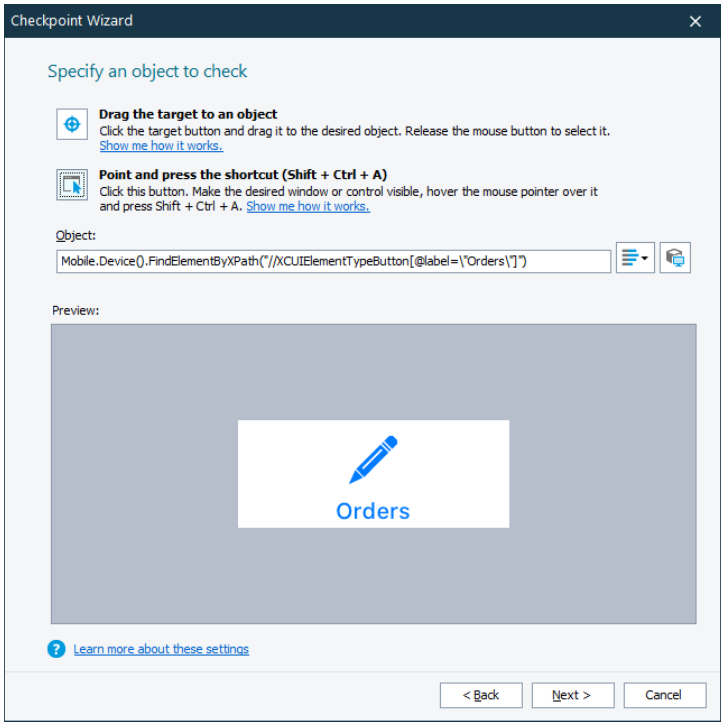
John Smith Jr
1, MyMoney, 2009-05-07

Clare Jefferson
2, FamilyAlbum, 2009-04-05

Susan McLaren
1, MyMoney, 2008-05-12

Steve Johns
1, ScreenSaver, 2008-04-04

- After you release the mouse button, TestComplete will restore the wizard and show the name of the selected object in the **Object** text box and the image of the object below it:



Specify an object to check

**Drag the target to an object**
Click the target button and drag it to the desired object. Release the mouse button to select it.
Show me how it works.

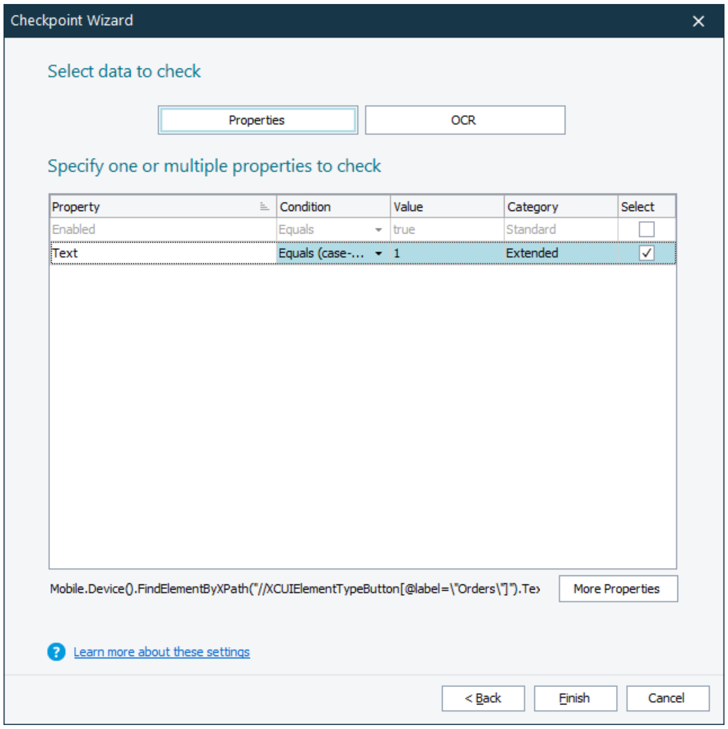
**Point and press the shortcut (Shift + Ctrl + A)**
Click this button. Make the desired window or control visible, hover the mouse pointer over it and press Shift + Ctrl + A. Show me how it works.

Object: Mobile.Device().FindElementByXPath("//XCUIElementTypeButton[@label=\"Orders\"]")

Preview:

Click **Next** to continue.

- The next page of the wizard shows a list of properties suggested for checking:



Select data to check

Specify one or multiple properties to check

| Property | Condition | Value | Category | Select |
|---|---|---|---|---|
| Enabled | Equals | true | Standard | ☐ |
| Text | Equals (case-… | 1 | Extended | ☑ |

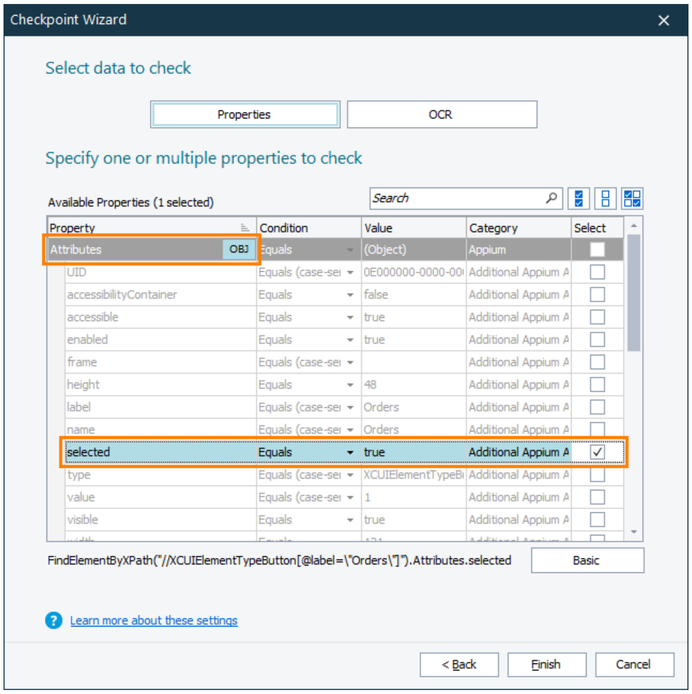
Mobile.Device().FindElementByXPath("//XCUIElementTypeButton[@label=\"Orders\"]").Te›

This list includes the properties provided by TestComplete and the properties defined by the tested application. To view all the available properties, you can click **More Properties**.

- Clear the property selected by default.

  To verify that the button is selected, we will use the selected attribute provided by the application accessibility information. Find the attributes property in the table and click its ellipsis button.

  Find the selected attribute and select its check box.



Checkpoint Wizard
Select data to check
Properties
OCR
Specify one or multiple properties to check
Available Properties (1 selected)
Search

| Property | Condition | Value | Category | Select |
|---|---|---|---|---|
| Attributes OBJ | Equals | (Object) | Appium | □ |
| UID | Equals (case-sei | 0E000000-0000-00 | Additional Appium A | □ |
| accessibilityContainer | Equals | false | Additional Appium A | □ |
| accessible | Equals | true | Additional Appium A | □ |
| enabled | Equals | true | Additional Appium A | □ |
| frame | Equals (case-sei |  | Additional Appium A | □ |
| height | Equals | 48 | Additional Appium A | □ |
| label | Equals (case-sei | Orders | Additional Appium A | □ |
| name | Equals (case-sei | Orders | Additional Appium A | □ |
| selected | Equals | true | Additional Appium A | ☑ |
| type | Equals (case-sei | XCUIElementTypeB | Additional Appium A | □ |
| value | Equals (case-sei | 1 | Additional Appium A | □ |
| visible | Equals | true | Additional Appium A | □ |

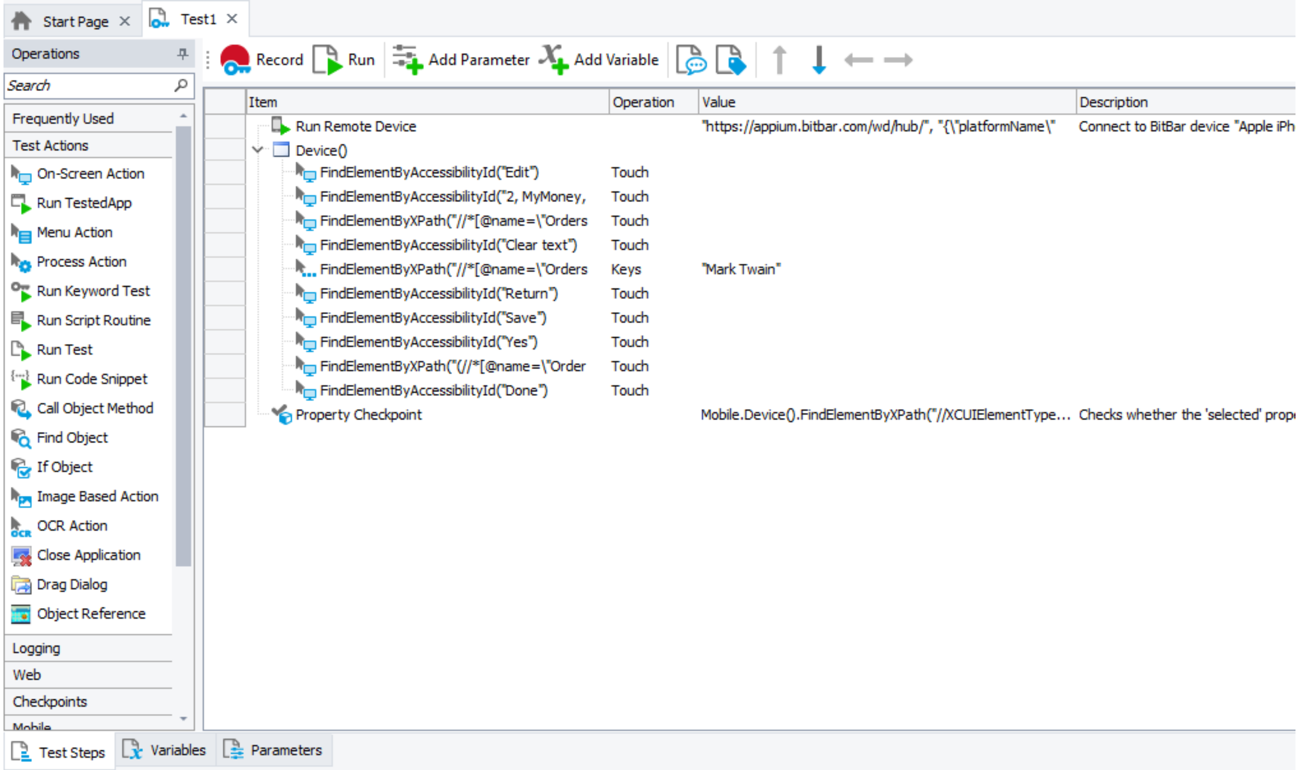
FindElementByXPath("//XCUIElementTypeButton[@label=\"Orders\"]").Attributes.selected

- Click **Finish** to complete the checkpoint creation. TestComplete will append the checkpoint command to the recorded test.

10. Click 🟦 **Stop** on the Recording toolbar to stop recording. TestComplete will process the recorded test commands and save them to the keyword test.

## 6. Analyze the Recorded Test

After you finish recording, TestComplete opens the recorded keyword test in the Keyword Test editor:

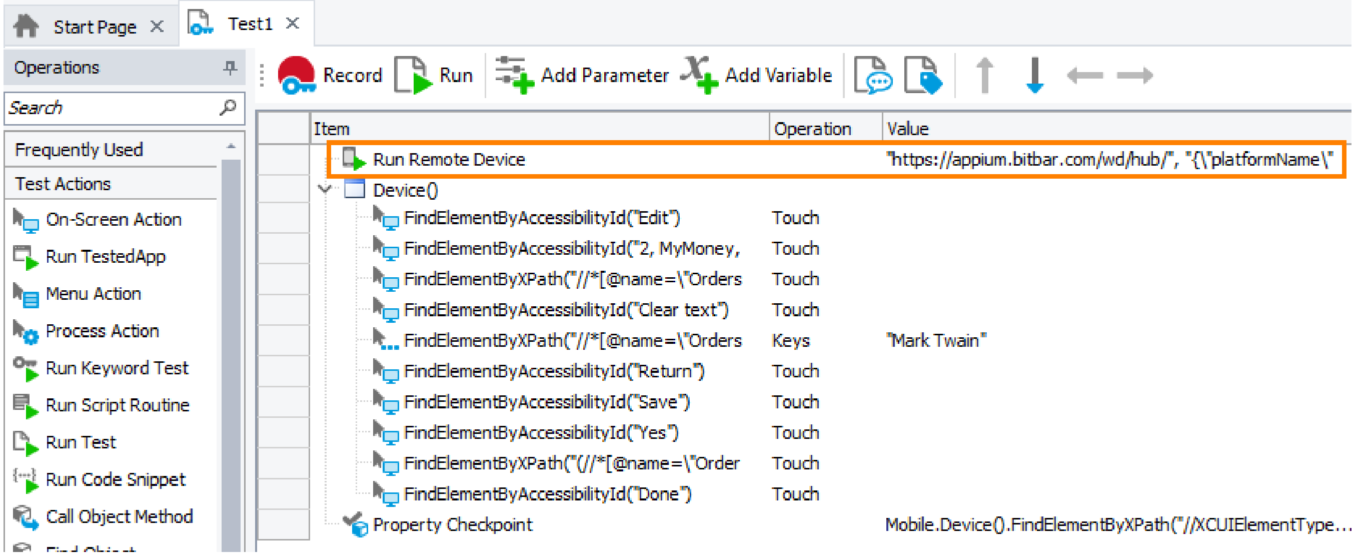


| Item | Operation | Value | Description |
|---|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platformName\" | Connect to BitBar device "Apple iPh |
| Device() | | | |
| FindElementByAccessibilityId("Edit") | Touch | | |
| FindElementByAccessibilityId("2, MyMoney, | Touch | | |
| FindElementByXPath("//*[@name=\"Orders | Touch | | |
| FindElementByAccessibilityId("Clear text") | Touch | | |
| FindElementByXPath("//*[@name=\"Orders | Keys | "Mark Twain" | |
| FindElementByAccessibilityId("Return") | Touch | | |
| FindElementByAccessibilityId("Save") | Touch | | |
| FindElementByAccessibilityId("Yes") | Touch | | |
| FindElementByXPath("(//*[@name=\"Order | Touch | | |
| FindElementByAccessibilityId("Done") | Touch | | |
| Property Checkpoint | | Mobile.Device().FindElementByXPath("//XCUIElementType… | Checks whether the 'selected' prop |

The recorded test is similar to the test shown in the image above. Your actual test may differ from this one. For example, it may contain some unnecessary touch actions.

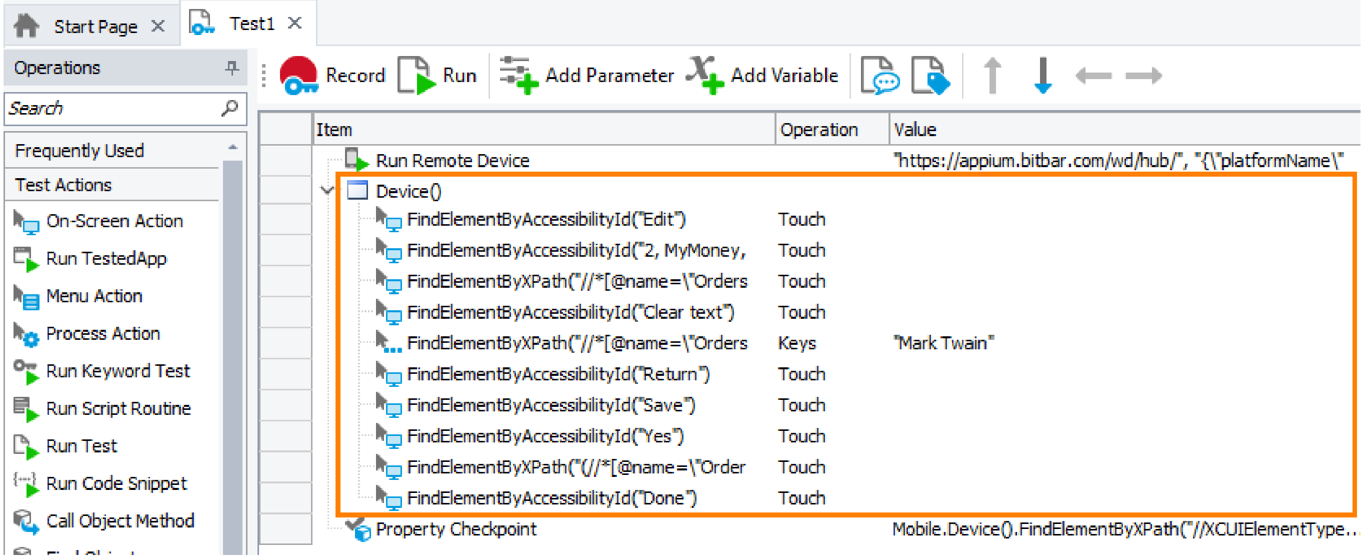
The test contains the commands that correspond to the actions you performed on the Orders application during test recording. We call these test commands operations.

The first operation in the test is **Run Remote Device**. It connects to a mobile device in the BitBar mobile device cloud, deploys the Orders application to the specified device, and opens a testing session for the application. The other test operations refer to this device.



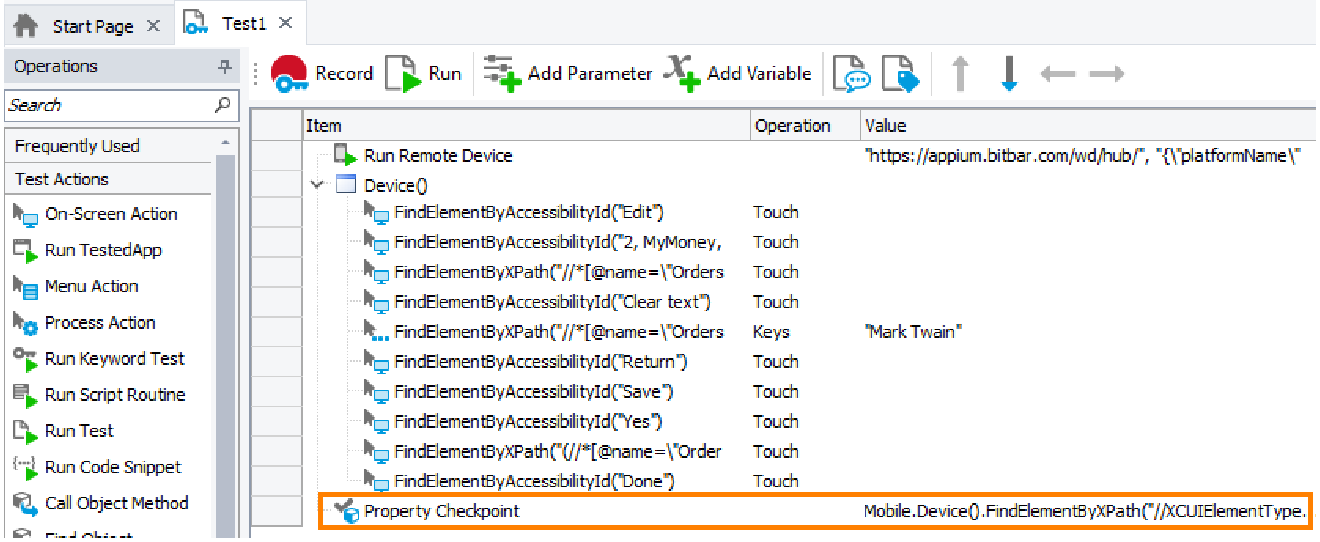
| Item | Operation | Value |
|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platformName\" |
| Device() | | |
| FindElementByAccessibilityId("Edit") | Touch | |
| FindElementByAccessibilityId("2, MyMoney, | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Touch | |
| FindElementByAccessibilityId("Clear text") | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Keys | "Mark Twain" |
| FindElementByAccessibilityId("Return") | Touch | |
| FindElementByAccessibilityId("Save") | Touch | |
| FindElementByAccessibilityId("Yes") | Touch | |
| FindElementByXPath("(//*[@name=\"Order | Touch | |
| FindElementByAccessibilityId("Done") | Touch | |
| Property Checkpoint | | Mobile.Device().FindElementByXPath("//XCUIElementType... |

Then there are the operations that simulate your actions with the application. These operations click the Edit button, select an item in the orders list, change the value of the text field, save the changes, and click the Done button.



| Item | Operation | Value |
|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platformName\" |
| Device() | | |
| FindElementByAccessibilityId("Edit") | Touch | |
| FindElementByAccessibilityId("2, MyMoney, | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Touch | |
| FindElementByAccessibilityId("Clear text") | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Keys | "Mark Twain" |
| FindElementByAccessibilityId("Return") | Touch | |
| FindElementByAccessibilityId("Save") | Touch | |
| FindElementByAccessibilityId("Yes") | Touch | |
| FindElementByXPath("(//*[@name=\"Order | Touch | |
| FindElementByAccessibilityId("Done") | Touch | |
| Property Checkpoint | | Mobile.Device().FindElementByXPath("//XCUIElementType.. |

Finally, there is the comparison operation that you added during test recording:



| Item | Operation | Value |
|---|---|---|
| Run Remote Device | | "https://appium.bitbar.com/wd/hub/", "{\"platformName\" |
| Device() | | |
| FindElementByAccessibilityId("Edit") | Touch | |
| FindElementByAccessibilityId("2, MyMoney, | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Touch | |
| FindElementByAccessibilityId("Clear text") | Touch | |
| FindElementByXPath("//*[@name=\"Orders | Keys | "Mark Twain" |
| FindElementByAccessibilityId("Return") | Touch | |
| FindElementByAccessibilityId("Save") | Touch | |
| FindElementByAccessibilityId("Yes") | Touch | |
| FindElementByXPath("(//*[@name=\"Order | Touch | |
| FindElementByAccessibilityId("Done") | Touch | |
| Property Checkpoint | | Mobile.Device().FindElementByXPath("//XCUIElementType. |

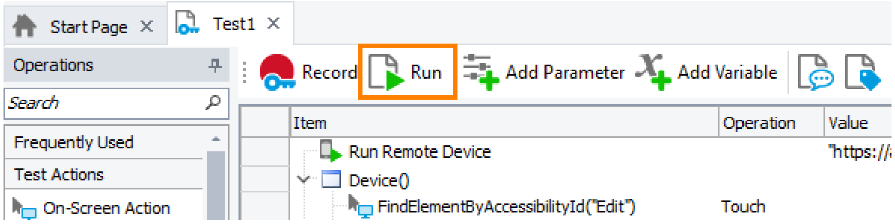
Each operation specifies the simulated action and the object over which the action was recorded. To locate the objects, the operations use various `FindElement` methods and search expressions based on the accessibility information provided by the application.

# 7. Run the Recorded Test

Now let's run our test to see how it works.

## Run the Test

To run the test, click **Run** on the test editor toolbar:



| Item | Operation | Value |
|---|---|---|
| Run Remote Device | | "https://i |
| Device() | | |
| FindElementByAccessibilityId("Edit") | Touch | |

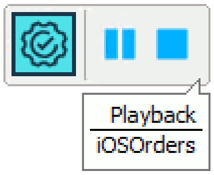
TestComplete will connect to the specified device in the BitBar device cloud, launch the Orders application on the device, and play back the recorded actions over it: open Samuel Clemens's order and change the customer name to Mark Twain.

## Test Results

After the test finishes, TestComplete shows the test log. You can look at it to see if the test has passed. We will tell you more about the test results in the next step of this tutorial.

## Notes on Running Tests

- **Important:** Do not touch the device screen during the test run to avoid interference with the test actions.
- During the test run, the TestComplete indicator in the top right corner of the computer screen displays information about the test operations performed.



The indicator displays messages informing you about simulated test actions.

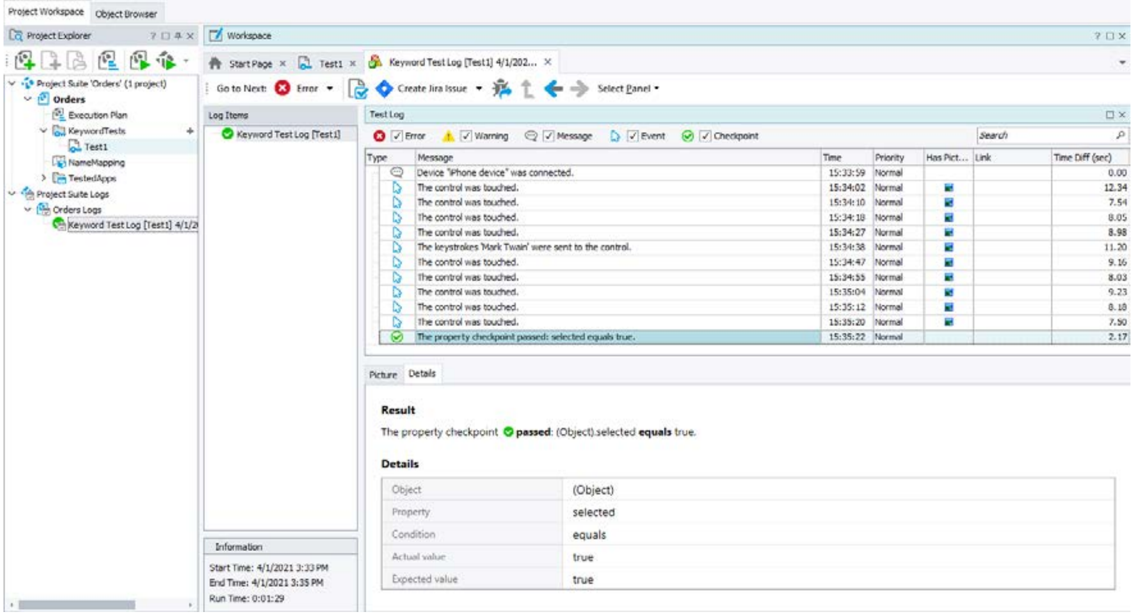
- You can stop the execution at any time by pressing 🟦 **Stop** on the Test Engine toolbar or in the indicator, or by selecting **Test > Stop** from TestComplete main menu.

  You can pause the test execution by clicking ⏸ **Pause** in the indicator. During the pause, you can perform any actions needed. For instance, you can explore the test log or check test variables and objects using TestComplete Watch List or Locals panel or the **Evaluate** dialog (see *Debugging Tests* in TestComplete Help).

For complete information and tips, see *Running Tests* section in TestComplete Help.

# 8. Analyze Test Results

After the test finishes, TestComplete shows the test log with the results of all test operations. The results of our test look as follows:



| Type | Message | Time | Priority | Has Pict... | Link | Time Diff (sec) |
|---|---|---|---|---|---|---|
| | Device "iPhone device" was connected. | 15:33:59 | Normal | | | 0.00 |
| | The control was touched. | 15:34:02 | Normal | ▪ | | 12.34 |
| | The control was touched. | 15:34:10 | Normal | ▪ | | 7.54 |
| | The control was touched. | 15:34:18 | Normal | ▪ | | 8.05 |
| | The control was touched. | 15:34:27 | Normal | ▪ | | 8.98 |
| | The keystrokes 'Mark Twain' were sent to the control. | 15:34:38 | Normal | ▪ | | 11.20 |
| | The control was touched. | 15:34:47 | Normal | ▪ | | 9.16 |
| | The control was touched. | 15:34:55 | Normal | ▪ | | 8.03 |
| | The control was touched. | 15:35:04 | Normal | ▪ | | 9.23 |
| | The control was touched. | 15:35:12 | Normal | ▪ | | 8.18 |
| | The control was touched. | 15:35:20 | Normal | ▪ | | 7.50 |
| ✓ | The property checkpoint passed: selected equals true. | 15:35:22 | Normal | | | 2.17 |

**Result**

The property checkpoint ✓ **passed**: (Object).selected **equals** true.

**Details**

| | |
|---|---|
| Object | (Object) |
| Property | selected |
| Condition | equals |
| Actual value | true |
| Expected value | true |

Information

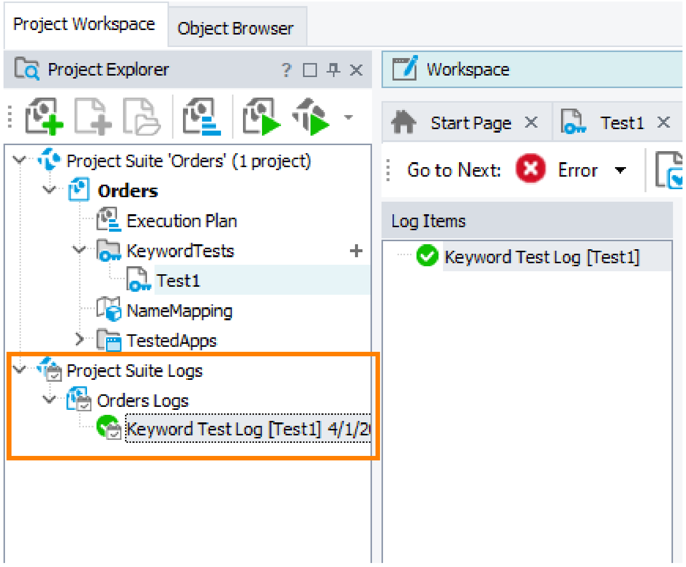
Start Time: 4/1/2021 3:33 PM
End Time: 4/1/2021 3:35 PM
Run Time: 0:01:29

The log contains messages of different types: actions, events, checkpoints, and so on. You can filter the messages using the toolbar above the message list. If you double-click a log message, TestComplete will navigate to the test operation that posted this message. This is useful when you need to know which operation caused an error.

The **Picture** panel contains images that show the application state when the selected test command was executed. The images help you understand what happened in the application during the run, and find errors faster.

The **Details** panel contains details on the performed operation. For example, Details for property checkpoints includes the check type (equals, contains, matches a regular expression, and so on) and other details.

All logs are kept under **Project Suite Logs** > **ProjectName Logs** in the **Project Explorer**, so that you can review the previous logs.



## Resolving Errors

Your test may fail. There can be several possible reasons for this. For example, developers could change the application behavior, the recognition attributes of application controls could change and make the test engine fail to find the needed objects, and so on.

One of the most typical reasons that novice users face is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance:

- If the tested application had been running before you recorded a test, it must also be running before you run the test.
- If test actions were performed on a particular screen of the application, you should open the screen when running the test.
- If you edited any data in the application, and then saved it, you need to revert the changes.

Sometimes, the test engine fails to find an application object because it needs to wait for a specific application response. To solve this problem, insert a delay command and specify for how long the text execution should be paused:

- In keyword tests, use the **Delay** operation with the *Delay Time* parameter specified.
- In scripts, use the `aqUtils.Delay` method with the loop body.

For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* and *Debugging Tests* sections in TestComplete Help.

# Where to Go Next

This concludes the Getting Started tutorial. We hope it helped you to get acquainted with TestComplete. Now you can learn how to create tests for other types of applications, or you can learn about more advanced features and even start creating your own tests. To get more information about TestComplete and its features, please refer to TestComplete Help. Below are some help topics that may interest you:

## Common

- *Recording in TestComplete*

  This section contains information on recording tests in TestComplete.

- *Checkpoints*

  This section describes various checkpoint types offered by the test engine and explains how to create checkpoints during test recording or test design.

- *Running Tests*

  This section contains information on how to run tests, how to organize batch runs (run a group of tests), how to schedule test runs and so on.

- *Test Log*

  Explains how TestComplete logs test results and describes the test log panels. This section also describes how to post messages, images, and files to the log.

- *Handling Playback Errors*

  Explains how to handle errors that occur during the test run.

- *Working With TestComplete*

  Explains how to share TestComplete projects with teammates and how to integrate your TestComplete tests into the build, development and quality assurance processes adopted in your organization.

## Specific for Testing Desktop Applications

- *Naming Objects*

  This section provides information on how TestComplete names processes, windows, and controls.

- *Simulating User Actions*

  This section describes how to simulate mouse clicks, keystrokes and selecting menu items with TestComplete.

- *Working With Application Objects and Controls*

  This section contains topics that explain how to perform specific actions over test objects and retrieve data from them.

- *Advanced Tasks*

Provides information about various TestComplete features that help you enhance your tests (how to handle events, how to work with ActiveX objects, files and databases, and so on).

## Specific for Testing Web Applications

- *Default Web Testing*

  Explains how to create web tests that identify web objects via properties induced by TestComplete.

- *Cross-Browser Testing in TestComplete*

  Explains how to create web tests that identify web objects via XPath expressions and CSS selectors and how to run those tests in device clouds.

- *Running Cross-Platform Web Tests in Parallel*

  Explains how to run cross-platform web tests in parallel in device clouds.

- *Testing Mobile Web Applications Using Emulator - Tutorial*

  A step-by-step tutorial for creating a sample cross-browser mobile web test.

## Specific for Testing Mobile Applications

- *About Mobile Tests*

  Explains how to create tests for mobile applications with TestComplete.

- *Connecting to Mobile Devices and Opening Testing Sessions*

  Describes how to connect to mobile devices from TestComplete.

- *Creating and Running Mobile Tests*

  Provides a step-by-step instruction on creating and running automated tests on mobile devices.

# Technical Support and Resources

If you have questions, problems or need help with TestComplete, contact our **support team** at:

- **https://support.smartbear.com/message/?prod=TestComplete**

The support team will answer you via e-mail and all further communication will be made via e-mail. However, to start the conversation, please use the Contact Support Form.

You can also ask questions, search for answers, exchange comments and suggestions on our **forums**, find answers to your question in the list of the **frequently asked questions**, watch **video tutorials** and **screencasts**, read **blogs** and **technical papers,** and take part in TestComplete **training seminars** offered by SmartBear.

For information on our support offerings, please visit our web site at **https://support.smartbear.com/**.

# Index