SMARTBEAR

with

TestComplete 12

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's 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 objects' 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, JScript, Python, VBScript, DelphiScript, C#Script and C++Script, so you can create scripts in the language you know best.

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

TestComplete Projects and Project Items

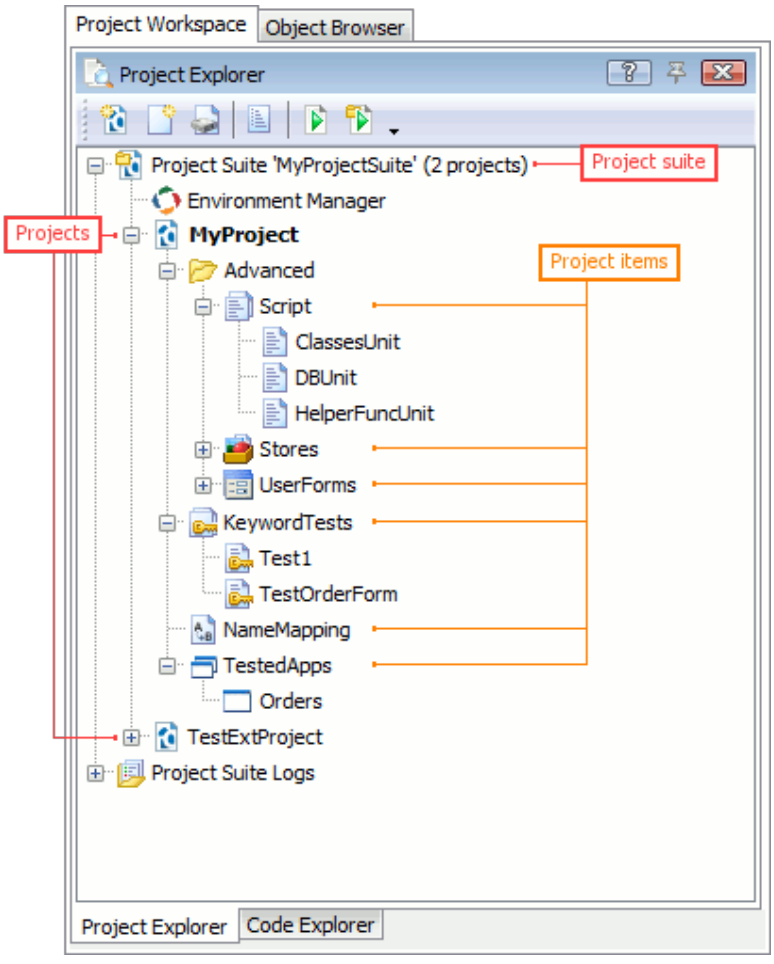
TestComplete operates with test projects and project suites. A **project** is a starting point for creating tests. It contains your tests, baseline data for checkpoints, information about tested applications and other items needed to perform testing. The project also defines the execution sequence of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

One project could 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).

Related projects can be united into a **project suite** that contains one or more projects. TestComplete automatically generates a project suite when you create a new project. You can also create empty project suites and then use TestComplete's dialogs to fill the suite with the desired project files.

Project items are project elements that perform or assist in performing various testing operations.

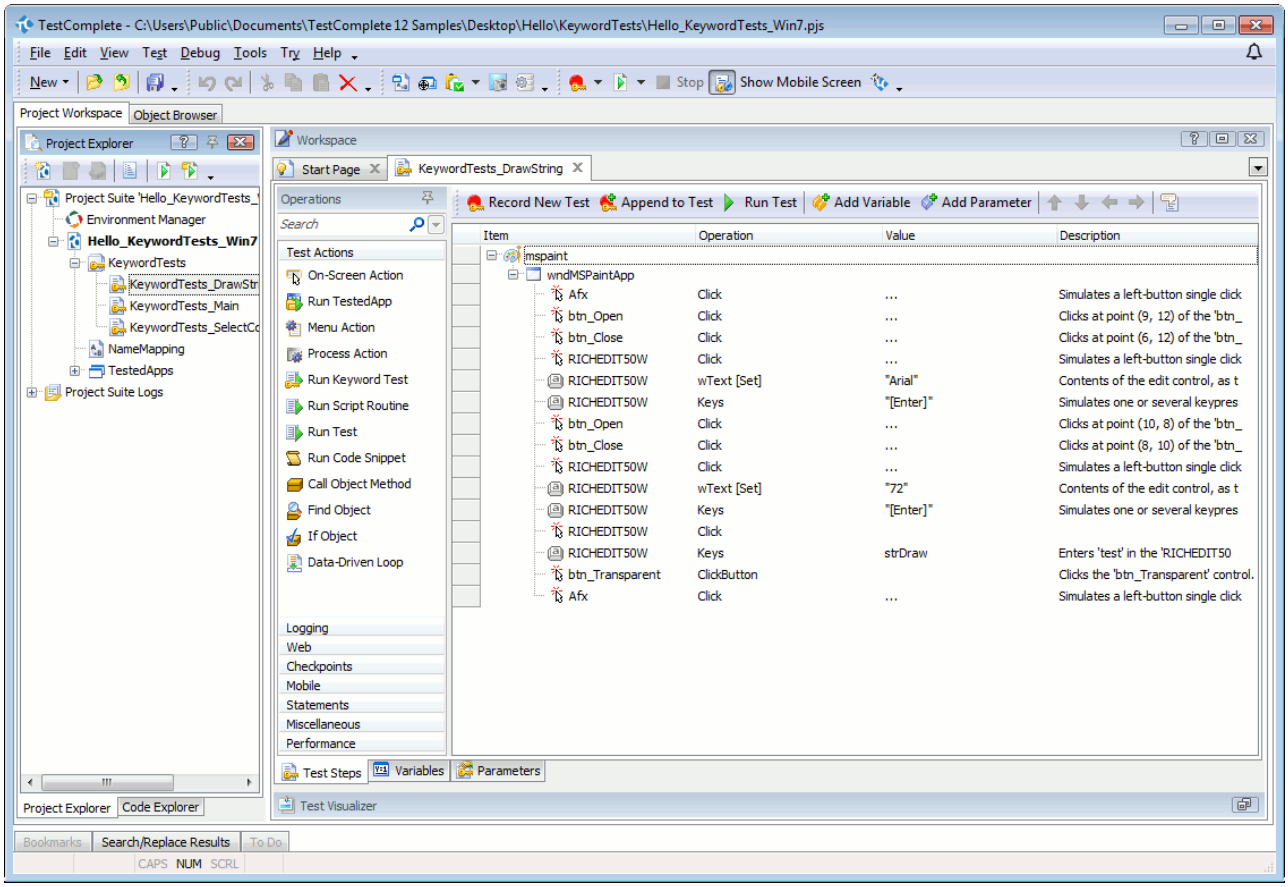
You can view and manage projects, project suites and project items in TestComplete's Project Explorer panel:



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's main window:



As you can see, TestComplete's 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's and project items' 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 and the Code Explorer panel provides a convenient way to explore script contents and navigate through script units.

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. For each process and window it shows methods and properties accessible externally through TestComplete facilities.

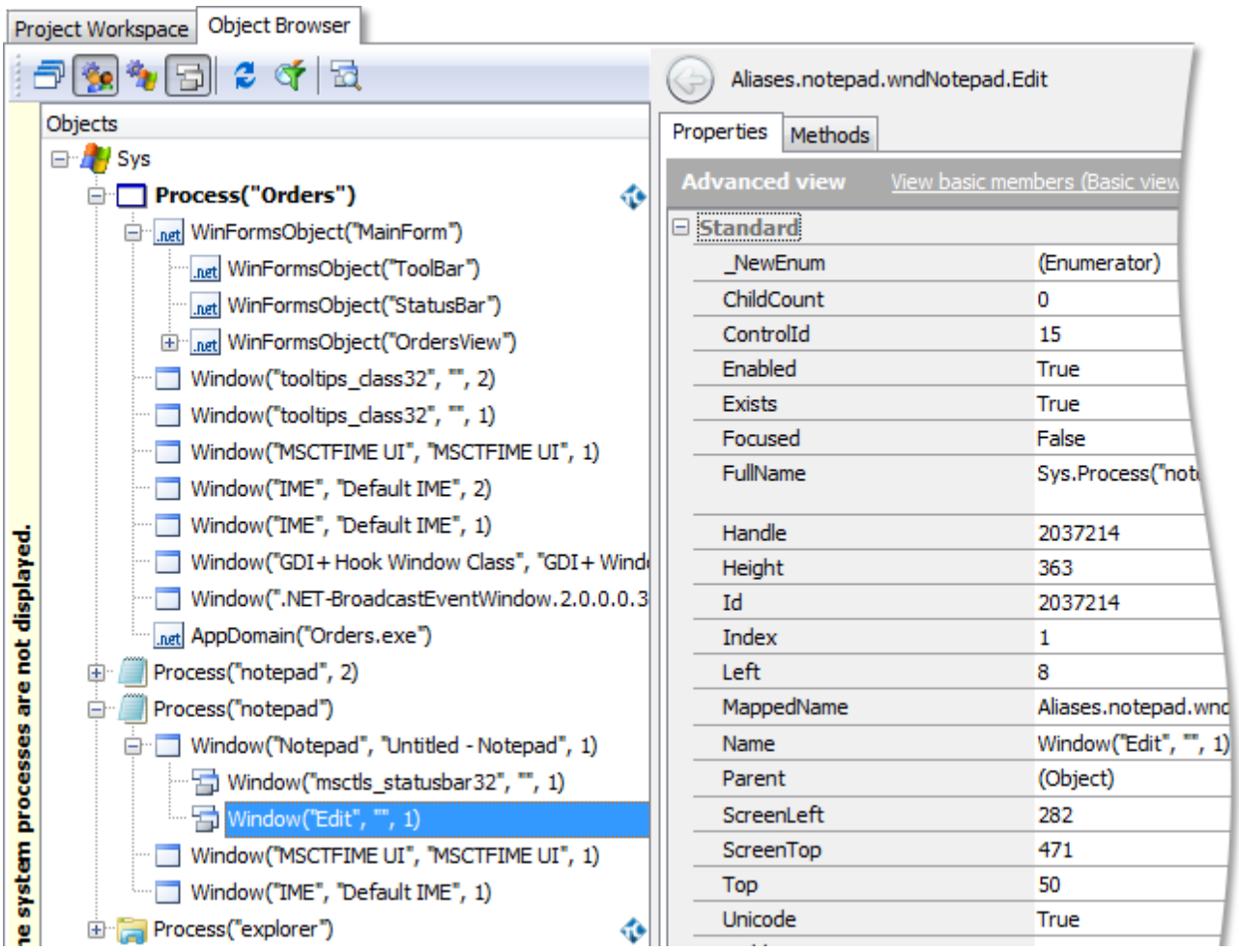
In other words, the Object Browser tells you which objects, methods and properties are available for testing, and how to get to them. See *Exploring Application Properties* in TestComplete Help.

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

The object structure is shown in the Object Browser panel:

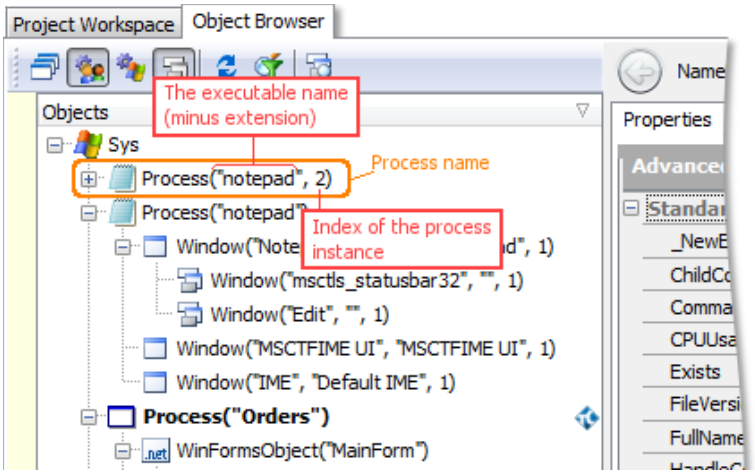


Note: The images in this topic demonstrate the object model of desktop applications. The object model of web and mobile applications is similar.

TestComplete uses a tree-like model for test objects. 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.

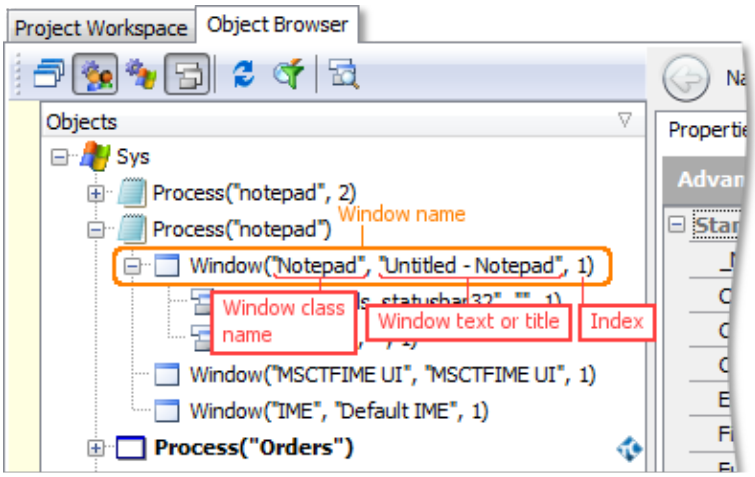
A process object's name includes the name of the process executable and its index (the index is used only if several application instances are running):



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 or not 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.

- **Black-box applications**

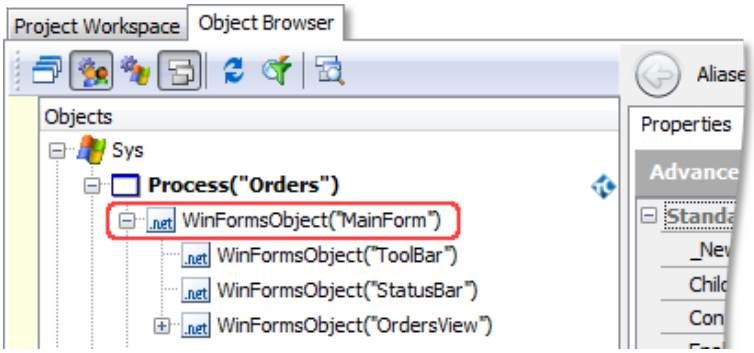
  Applications that do not provide access to their internal methods and properties are called **black-box applications**. The name of each window of such applications includes the window's class name, the window's text or title (caption) and its index. Controls are named in the same manner as windows,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



- **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's 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")`:



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's 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 an application's 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.

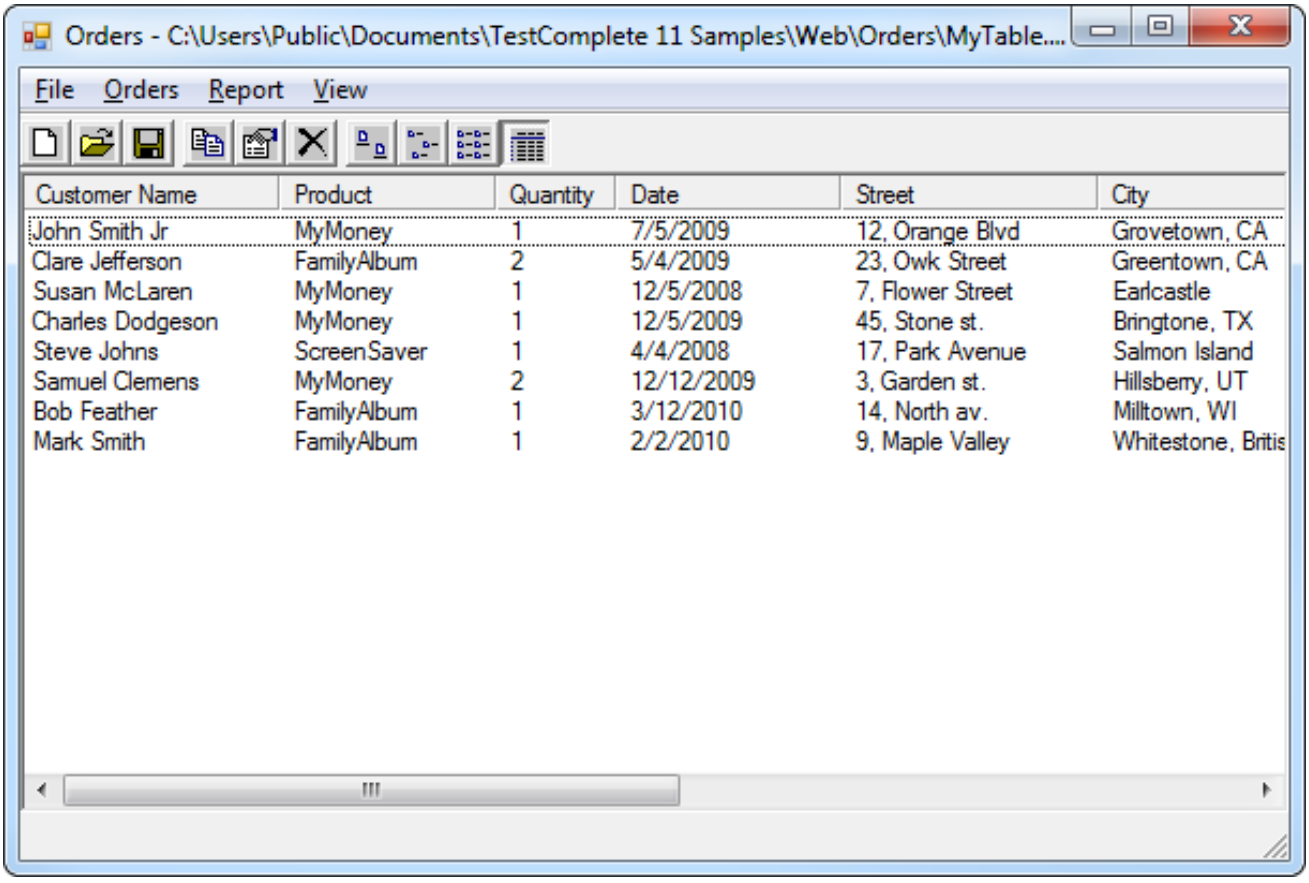
For more information on creating checkpoints and verification code, see *About Checkpoints* in TestComplete Help.

Testing Desktop Applications

This tutorial explains the basics of testing desktop applications (that is, applications that run on desktop computers). 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.



The application is located in the following folder:

- On Windows 7, Windows Vista, Windows Server 2008 or later operating systems:

  C:\Users\Public\Public Documents\TestComplete 12 Samples\Open Applications

- On Windows XP or Windows Server 2003:

C:\Documents and Settings\All Users\Shared Documents\TestComplete 12 Samples\Open Applications

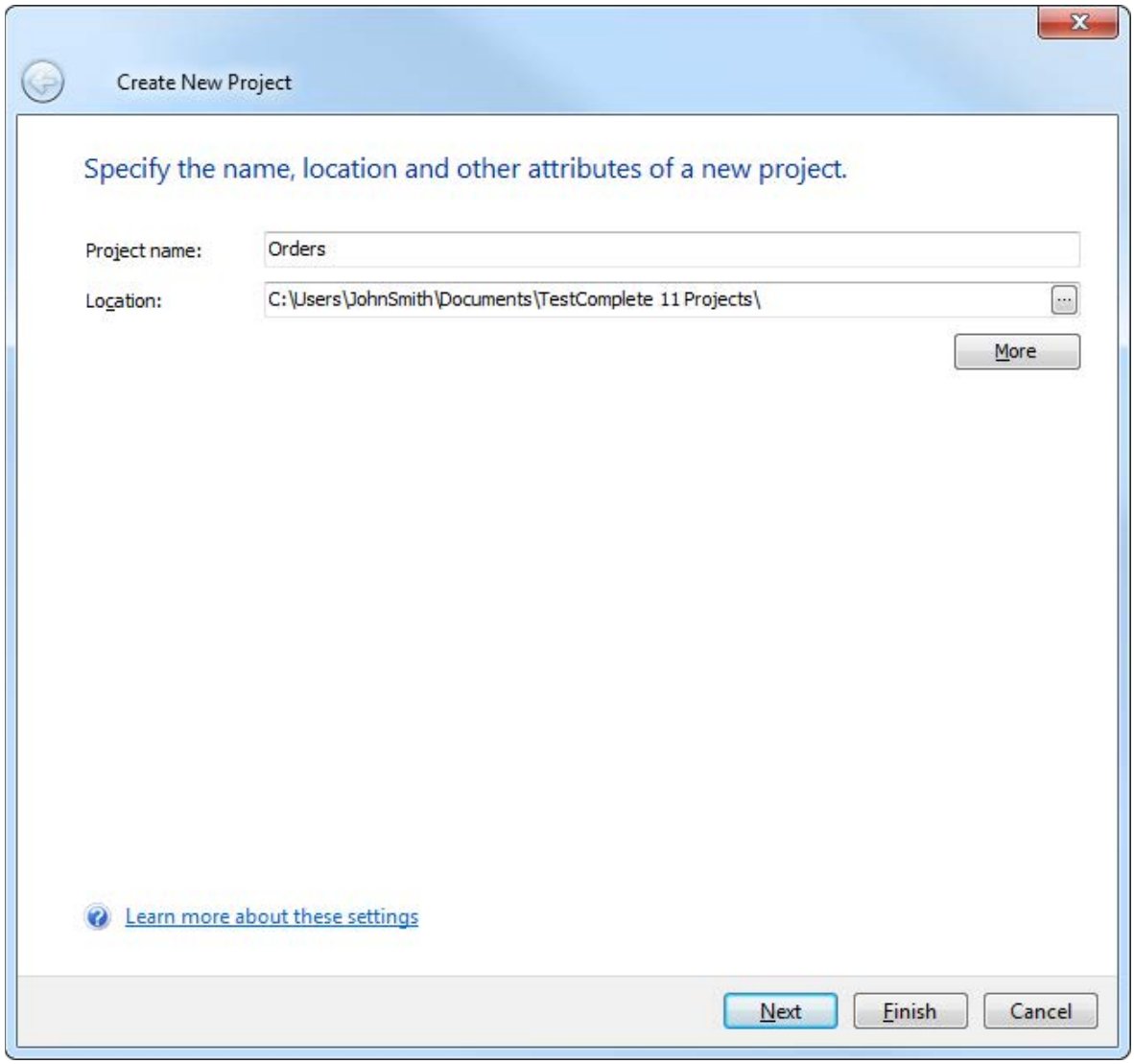
Note: Some file managers can display the *Shared Documents* and *Public Documents* folders as the *Documents* folder.

The folder stores several *Orders* projects created with different compilers: C#, Visual C++, Visual Basic, Delphi, C++Builder, Swing and so on. We will use the *Orders* application created with Visual C#.

1. Creating a Test Project

Let's create a new test project:

1. If you have a project or project suite opened in TestComplete, close it. To do this, choose **File | Close** from TestComplete's main menu.
2. Select **File | New | New Project** from TestComplete's main menu. This will call up the **Create New Project** wizard:



3. On the first page of the wizard, you can specify the project name and location. Enter *Orders* to the **Project name** edit box. TestComplete will automatically generate the project path and display it in the **Location** field. The project folder is used to store all information generated for or by the project: keyword tests, scripts, test logs, stores, and so on. You can change the project's folder in the Location box. In our example we will keep the folder name unchanged.

   You can also specify the project suite name and its actual location by clicking the **More** button and filling in the corresponding edit fields. In our example, we will keep the project suite name and location unchanged.

4. After you specify the project name and location, click **Next to continue**.

We will continue working with the wizard and use its pages to add tested applications to the project and specify some other project settings.

2. Defining Applications to Test

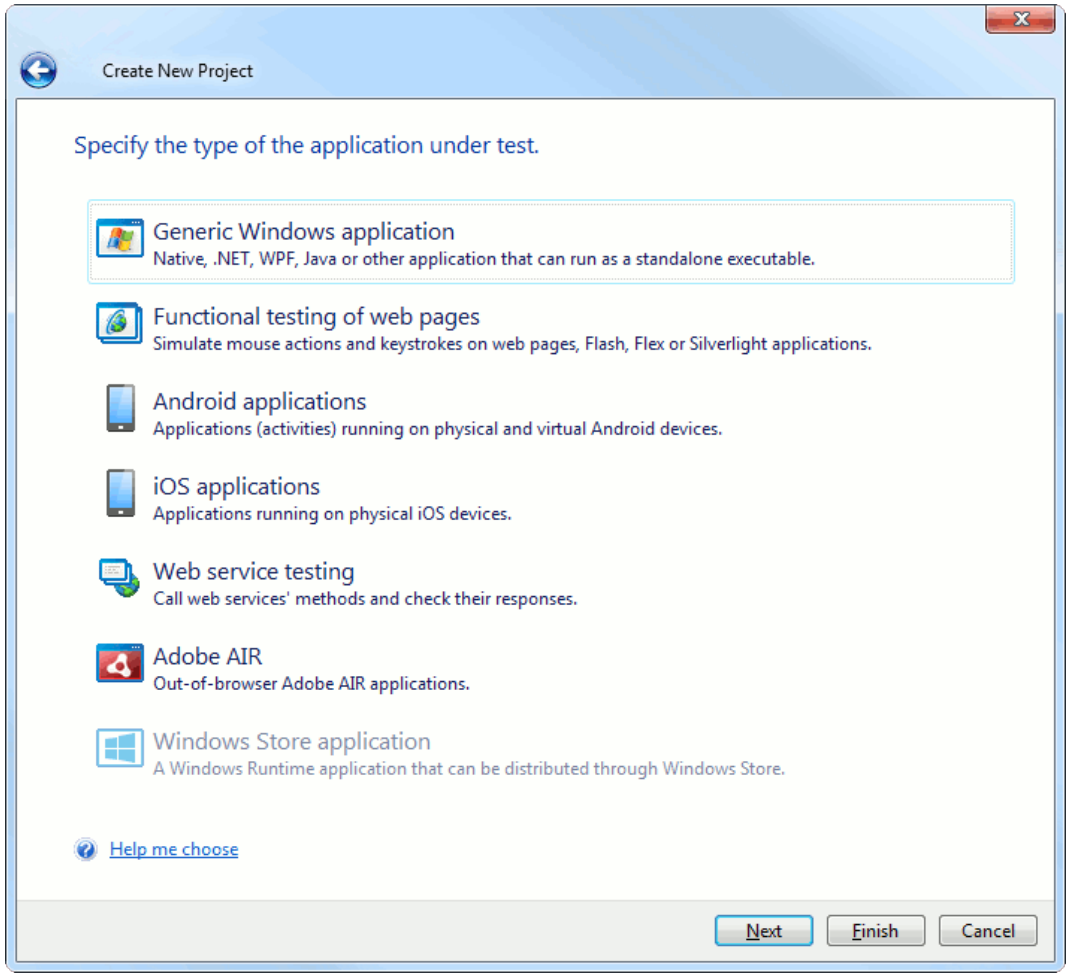
Each TestComplete project may have a list of tested applications. This is a way for you to keep track of which applications the project deals with and how they are configured for testing. It also allows TestComplete to launch all applications specified in the list or only those applications that are enabled to be launched manually via the context menu, or from a test. Of course, since projects are independent from each other, any application may be in the list of more than one project.

There are several ways to add applications to the list of tested applications:

- You can do this with the **Create New Project** wizard during project creation.
- You can do this at any time later by using the context menu of the **Project Explorer** panel.
- TestComplete can also 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. After the recording is over, TestComplete will add the tested application to the list and insert the "Run Tested Application" command into the recorded test.

In this tutorial, we will add the tested application to the project by using the Create New Project wizard.

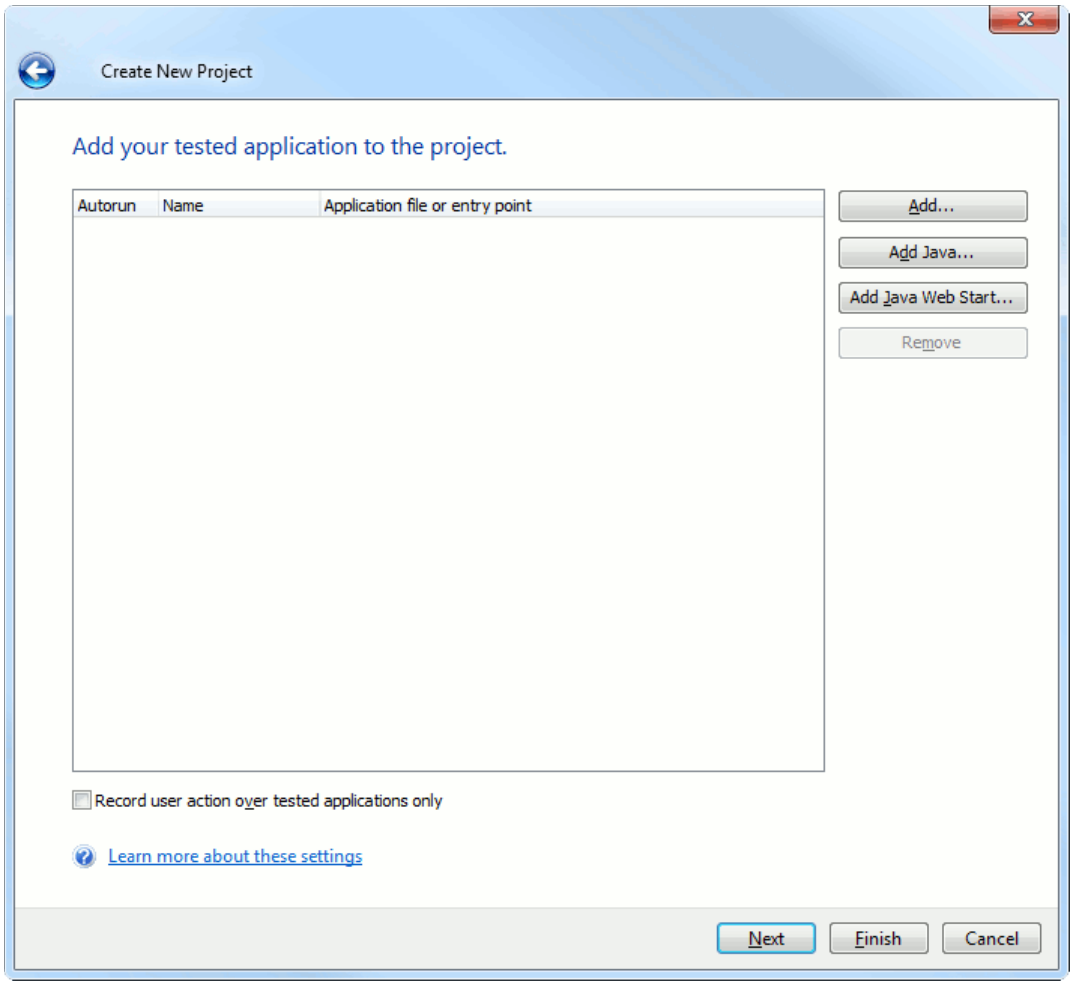
1. After you specify the project name and location on the first page of the wizard, the wizard shows the second page where you can choose the type of your tested application:



This will help TestComplete choose the appropriate run mode for your application.

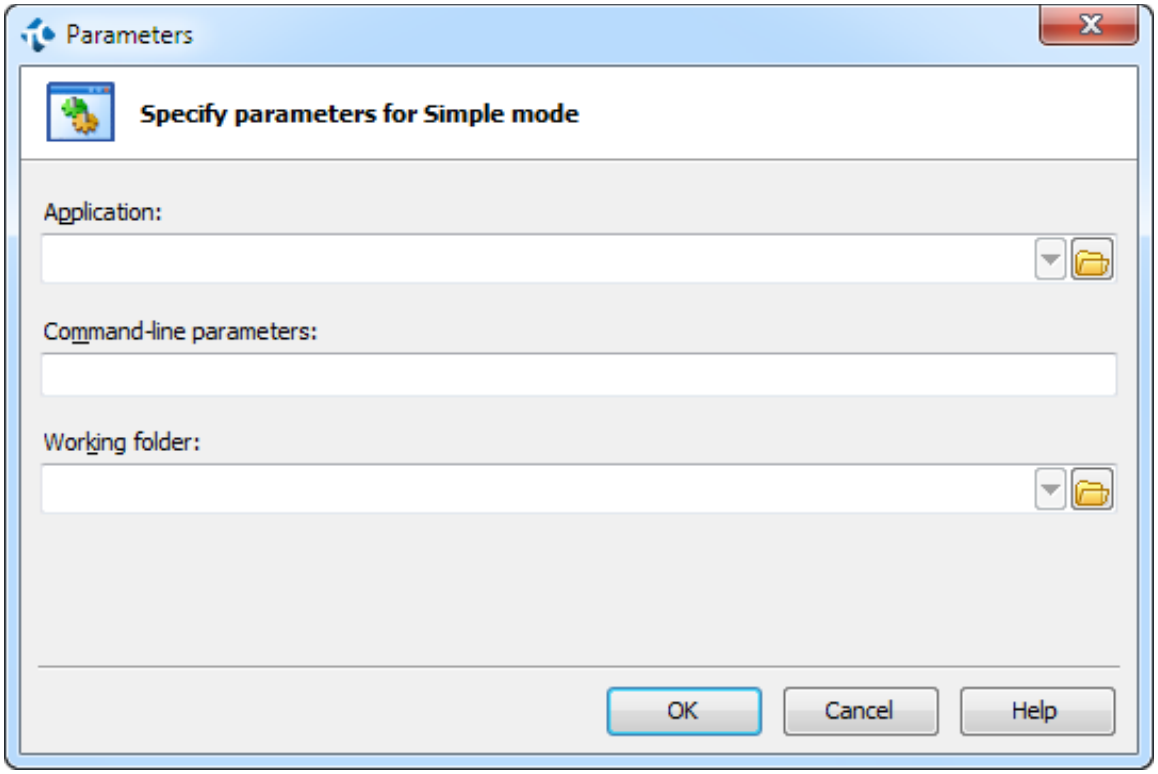
As you may remember, we are going to test the *Orders* application written in C# and shipped with TestComplete. This is an ordinary .NET application that runs as a stand-alone executable. In the wizard, it falls under the Generic Windows Application category. So, click **Generic Windows Application** and, if you use Windows XP, click **Next** to continue. On Windows Vista and later versions of the operating system, the wizard will switch to the next page automatically after you click the category name.

2. On the next page of the wizard, you can add the tested application to your test project:



To do this:

- Click **Add**. This will invoke the **Parameters** dialog in which you specify the launch parameters of the application under test:



- In the **Application** text box of the dialog, click the button to invoke the standard dialog for opening files
- In this dialog, locate *Orders.exe* and then click **Open**.

  The path to the C# version of the Orders.exe file looks like this:

  - If you are working under Windows Vista, Windows 7 or later operating systems:

    C:\Users\Public\Public Documents\TestComplete 12 Samples\Desktop\Orders\C#\bin\Release\Orders.exe

  - If you are working under Windows XP or Windows Server 2003:

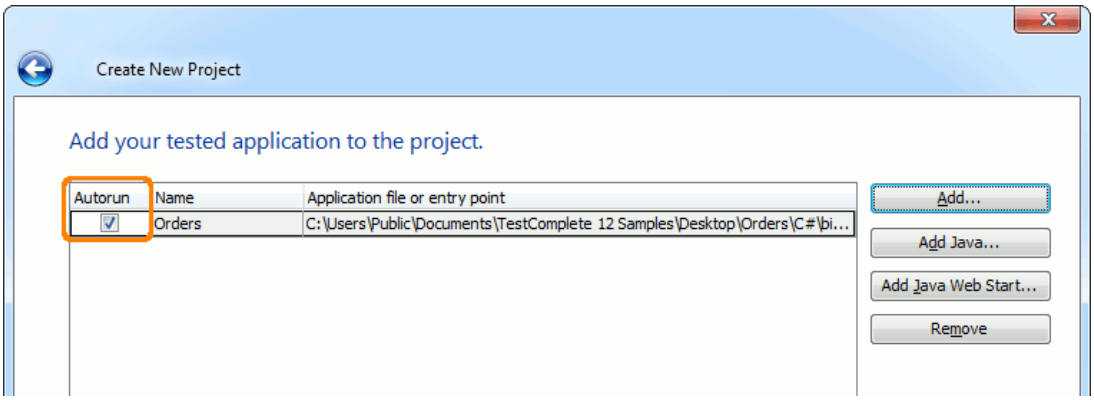
    C:\Documents and Settings\All Users\Shared Documents\TestComplete 12 Samples\Desktop\Orders\C#\bin\Release\Orders.exe

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

- After you choose Orders.exe, the Parameters dialog will display the full path to the application. In the dialog, click OK.

The wizard will display the Orders application's name and path in the list of tested applications.

3. Make sure that the **Autorun** check box in the list is selected. If it is selected, TestComplete will automatically launch the Orders tested application when you start recording tests. If the check box is clear, then to record user actions over your application you will have to launch the application manually.



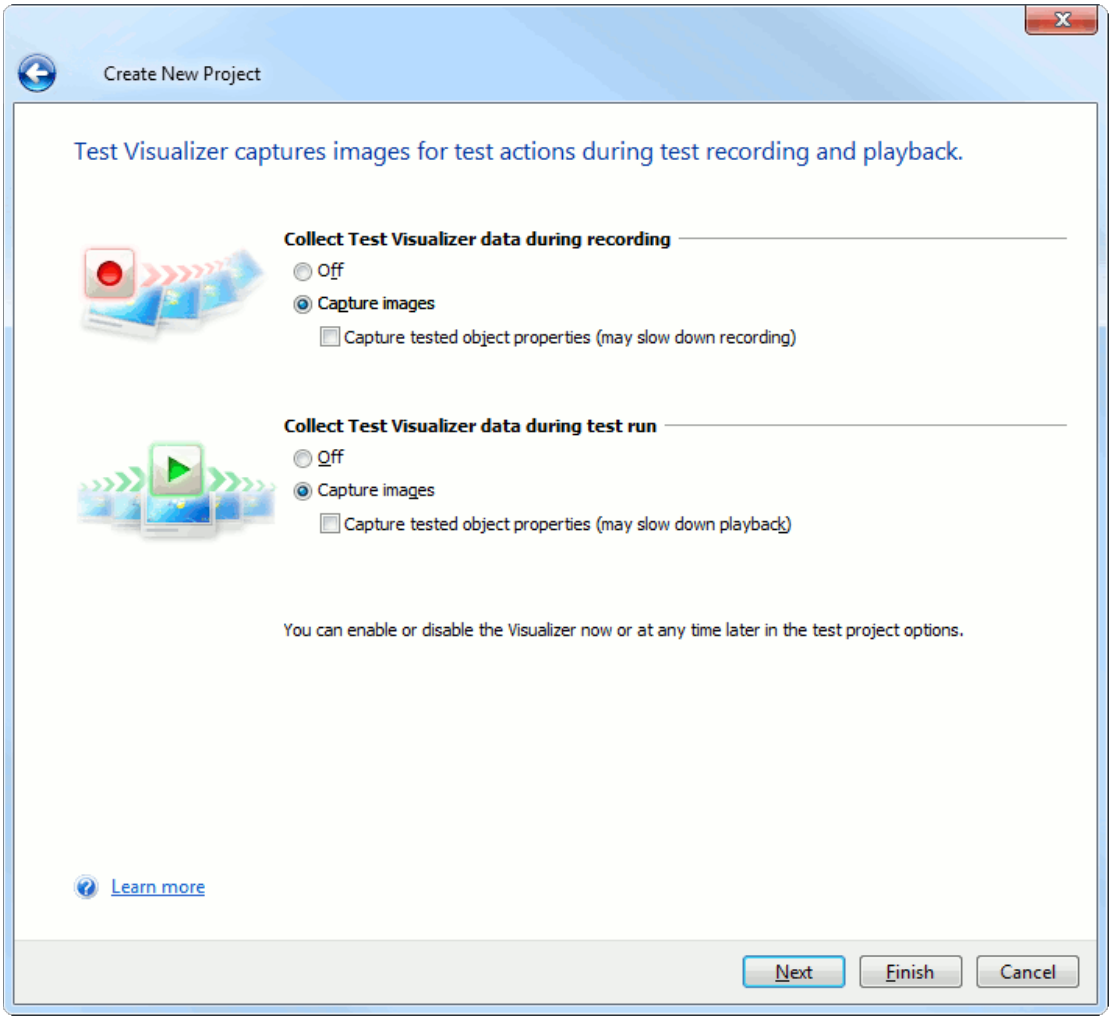
4. After you add the application to the list and verify that the Autorun check box is selected, click **Next** to continue.

In the next section, we will go through the rest pages of the wizard and complete the project creation.

3. Completing the Project Creation

On the previous step, we added our sample application, Orders, to the project's list of tested applications. Let's quickly go through the rest pages of the wizard and complete the project creation:

1. After you added tested applications to your project in the Create New Project wizard, the wizard displays the page where you can enable or disable TestComplete's Test Visualizer functionality:



Test Visualizer captures information for test actions during test recording and playback. Depending on the selected options, Test Visualizer frames can contain screenshots only or screenshots along with information about the objects they contain.

The Visualizer frames that were captured during recording help you better understand what the recorded test commands do, what is important when you have just started learning the product.

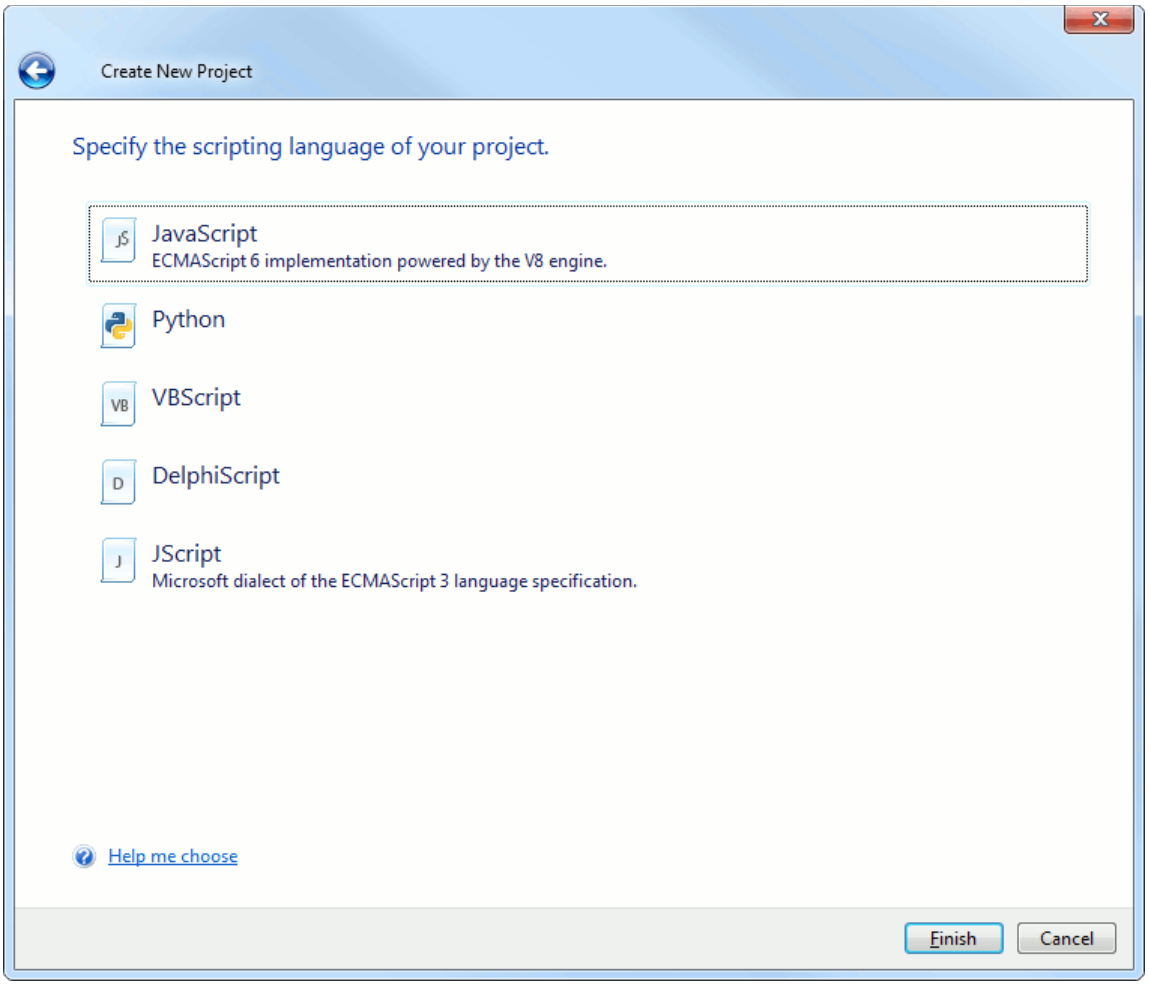
The Visualizer frames captured during test execution let you easily determine what happens to the tested application or system at that time. This information is helpful when you are debugging errors.

However, the images and test object data occupy hard disk space and in large projects they may be the reason of significant increase of the size of test result files. So, you can limit the amount of

collected data by capturing images only, or, if Visualizer is not needed, you may disable it and enable it at any time later using your project's settings.

In our tutorial, we select **Capture images** both for test recording and playback and leave the **Capture tested object properties** unselected. Then, click **Next** to continue.

2. On the next page of the wizard, you can choose the scripting language to be used in your project.



Every TestComplete project uses one of the supported scripting languages: JavaScript, JScript, Python, VBScript, DelphiScript, C#Script or C++Script. The scripting language is important even if you are not going to use script units in your project. Even if you are going to use only keyword tests, you may need to call code snippets or use script statements to specify operation parameters.

The scripting language is also important because it defines the format of object names with which your tests will work, regardless of whether you are using scripts or keyword tests. The name format depends on the language syntax. For instance, in JavaScript, JScript, Python and VBScript, the name of the Notepad process looks like `Process("Notepad")`. In DelphiScript you should replace double quotes with single quotes, that is, `Process('Notepad');` and in C#Script and C++Script, the word Process should be enclosed in brackets: `["Process"]("Notepad").`

For more information on choosing the scripting language, see *Selecting the Scripting Language* topic in TestComplete Help.

In this tutorial, we will use VBScript. So, select **VBScript** on the page. On Windows Vista and later versions of the operating system, this will close the wizard. If you are using Windows XP, click **Finish**.

TestComplete will create a new project, *Orders.mds*, and a project suite for it. It will then display the project suite's and the project's contents in the **Project Explorer** panel.

Now we can create tests.

4. Creating a Test

Planning a Test for the Orders Application

The sample Orders application maintains a list of orders. Suppose we need to test whether the application's Edit Order form functions correctly and modifies data in the order list. In this case –

- **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 the order's 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.

Creating Tests in TestComplete

TestComplete allows you to create tests in two ways. You can:

- Create tests manually
- Record tests

When you create a test manually, you enter all the needed commands and actions that your test must perform via appropriate script objects or keyword test commands. This approach is very 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 different problems. For example, while creating a test manually you must know the classes and names of your application's objects you want to work with. To solve such problems, TestComplete includes a special feature that lets you easily create tests. 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. We call this feature

"**recording a test"**, because you create a test visually and in one sense you record the performed actions to a script or keyword test. It is a very useful approach and it does not require much experience in creating tests. So, in this tutorial we will demonstrate how to record tests with TestComplete. For more information, see the section below.

Recording Tests in TestComplete

The recording includes three steps:

1. You start recording by selecting **Test | Record | Record Keyword Test** or **Test | Record | Record Script** from TestComplete's main menu or from the Test Engine toolbar. You can also start recording by clicking **Record a New Test** on the Start Page.

   With TestComplete you can record tests of various kinds: keyword tests, scripts, low-level procedures and HTTP load testing tasks. 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 is started. The main recorded test will contain special commands that will run these tests.

   After you command TestComplete to start the recording, it will switch to the recording mode and display the **Recording toolbar** on screen:

   The toolbar contains items that let you 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 the *Recording in TestComplete* section in TestComplete Help.

Recording Test for the Orders Application

Let's now record a keyword test for the sample Orders application. The test will launch the application, load data in it, simulate clicks and keystrokes within the application's window and verify the application's data.

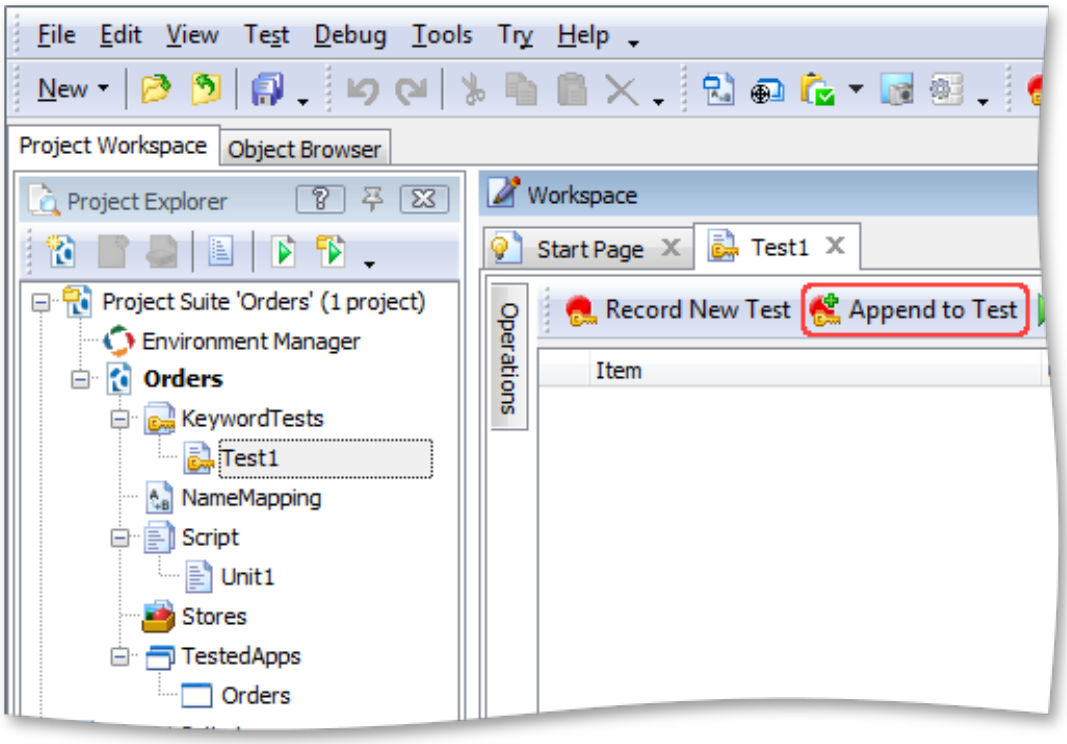
Note: Do not switch to the TestComplete help during the recording. The recording engine traces and records all user actions, so the recorded test will contain commands that simulate "switching".

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.

We would like to mention that after you start the recording, TestComplete's main window is automatically minimized, and it cannot be activated until you stop the recording. If you try to switch to the TestComplete window when the recording is in process, TestComplete displays a "TestComplete is in recording mode and cannot be activated." message. To continue creating a test, click **Continue** in this message, and TestComplete will resume the recording. Note that when the message is shown, TestComplete automatically pauses the recording, and all your actions against the tested application are not recorded.

Let's start recording:

1. When creating a new project, TestComplete automatically creates an empty keyword test in this project. Let's record test commands into this test. To start recording, select the **Append to Test** item on the test editor's toolbar:



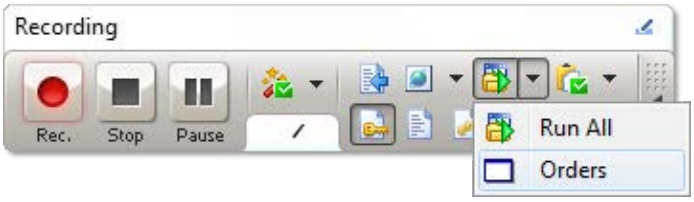
TestComplete will display the Recording toolbar on screen. If the **Interactive Help** panel is visible, TestComplete will also show information about the recording in it.

By default, the **Recording** toolbar is collapsed:

Click the ▸ arrow button to expand the Recording toolbar and view all its buttons:

2. After you start the recording, TestComplete automatically launches the Orders tested application that we added to the project's list of tested applications. This happens, because we enabled the application's Autorun setting when we were adding the application to the project (see *Defining Applications to Test* in TestComplete Help).

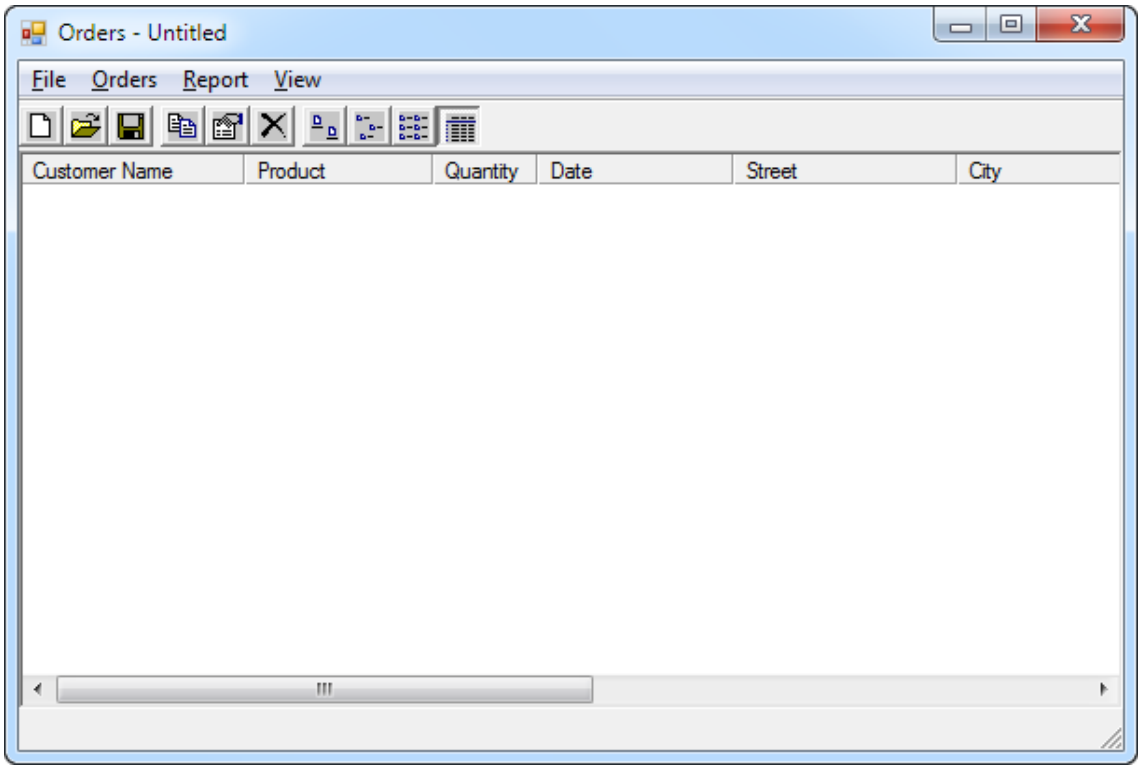
If we had disabled this property, we would have had to launch the application manually. You can do this by selecting the Run command from the Recording toolbar:



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's main window is shown:



If the Interactive Help panel is visible, resize or move it so that it does not overlap the application's window. Your actions on this panel are not recorded.

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

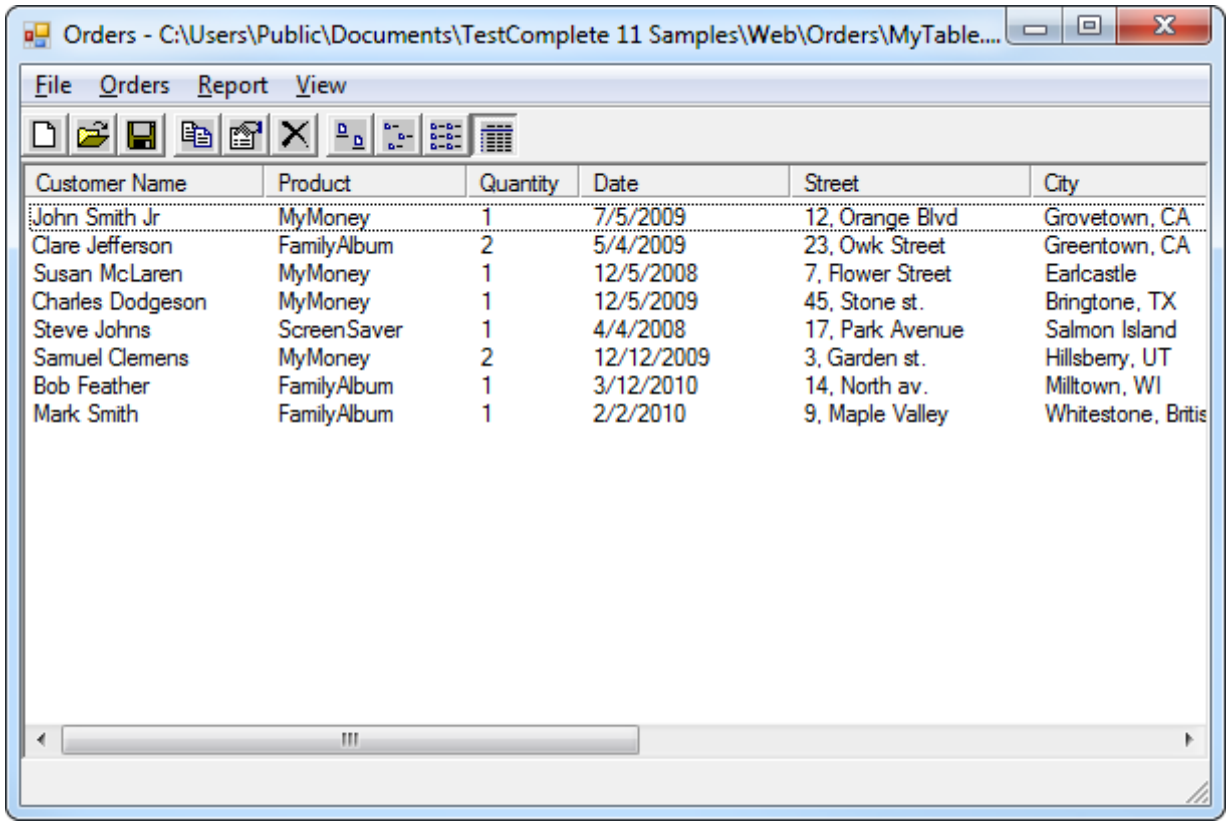
5. In the dialog, open the *MyTable.tbl* file. The location of this file depends on the operating system you use.

   On Windows Vista, Windows 7 and later operating systems it resides in the *C:\Users\Public\Public Documents\TestComplete 12 Samples\Open Applications* folder. On other operating systems, the file is located in the *C:\Documents and Settings\All Users\Shared Documents\TestComplete 12 Samples\Open Applications* folder.

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

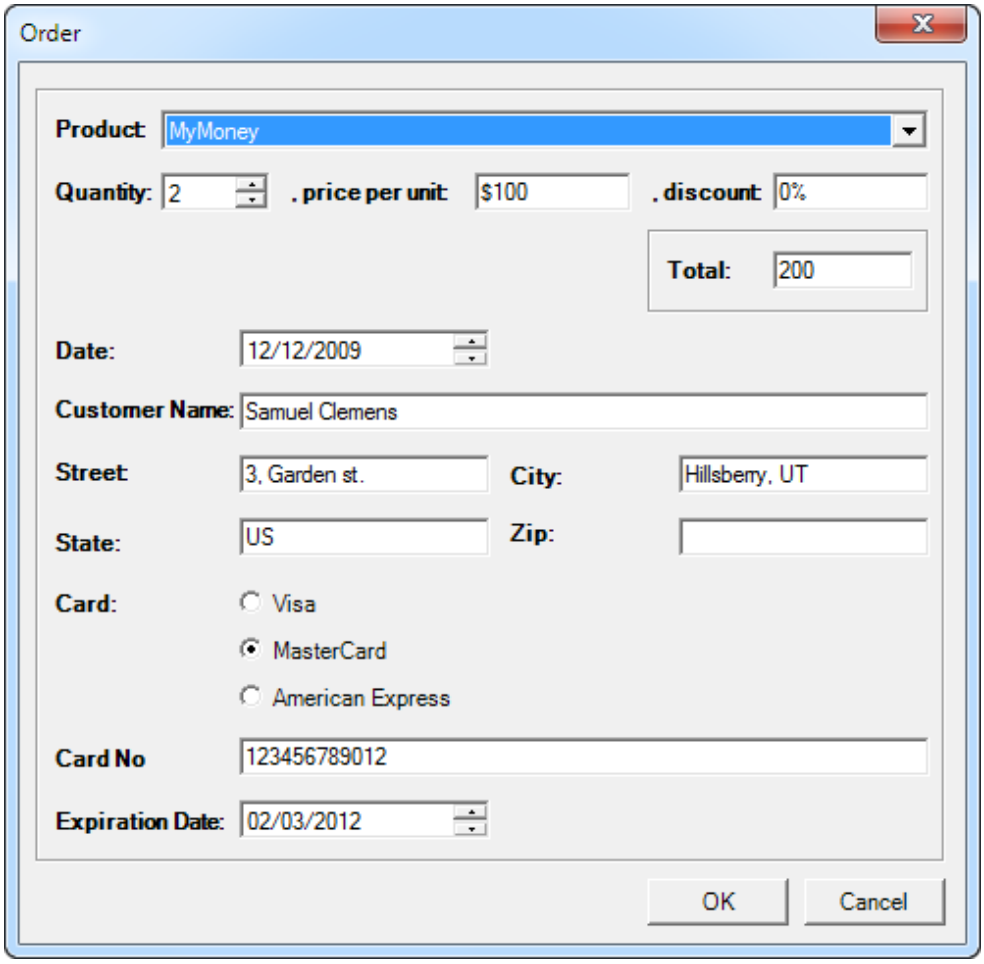
   It is recommended to type the fully-qualified file name into the **File name** box of the Open File dialog. Typing instead of using the mouse will help you avoid problems if the test is played back on a different operating system or if the Open File dialog displays a different initial folder when the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

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



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

8. Move the mouse cursor to the Orders toolbar and press **Edit order**. This will call the **Order** dialog:



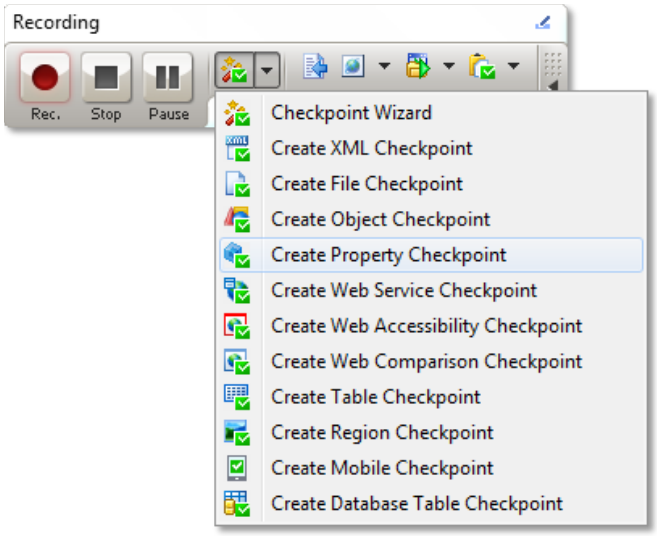
9. In the dialog, click within the **Customer Name** text box to move the insertion point there. Right-click within the Customer Name box and choose **Select All** from the context menu and then enter *Mark Twain* as the customer name.

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

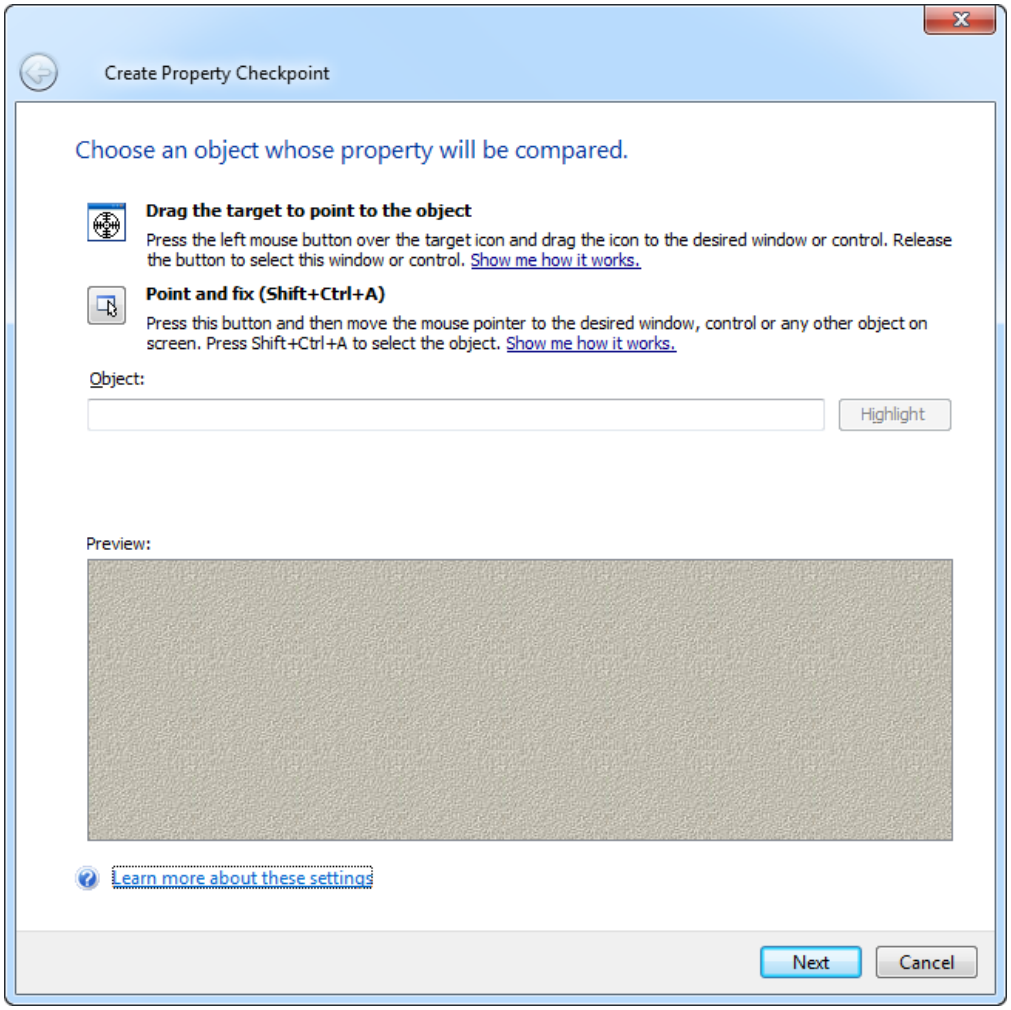
11. Now let's insert a comparison command into our test. It will verify that the application's customer list displays the modified name - *Mark Twain*.

    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.

- Select **Create Property Checkpoint** from the **Checkpoint** drop-down list of the Recording toolbar:



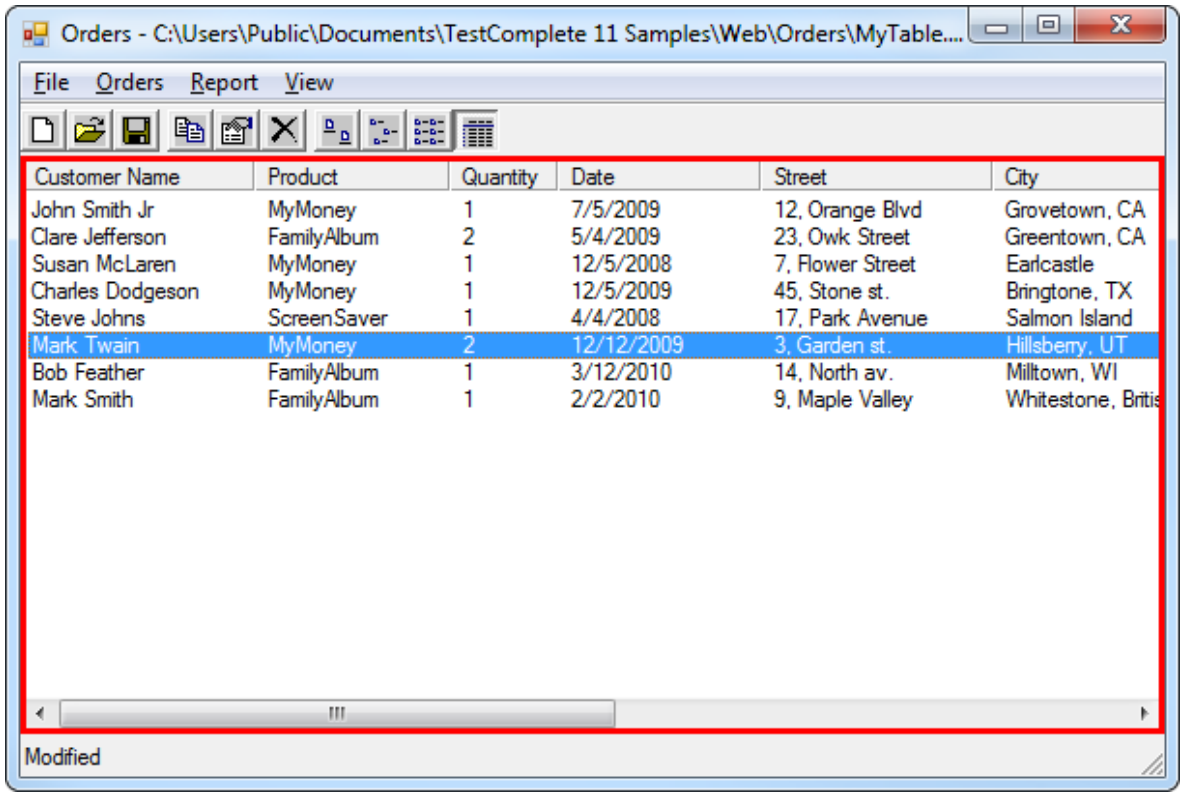
This will invoke the **Property Checkpoint** wizard. It will guide you through the process of checkpoint creation:



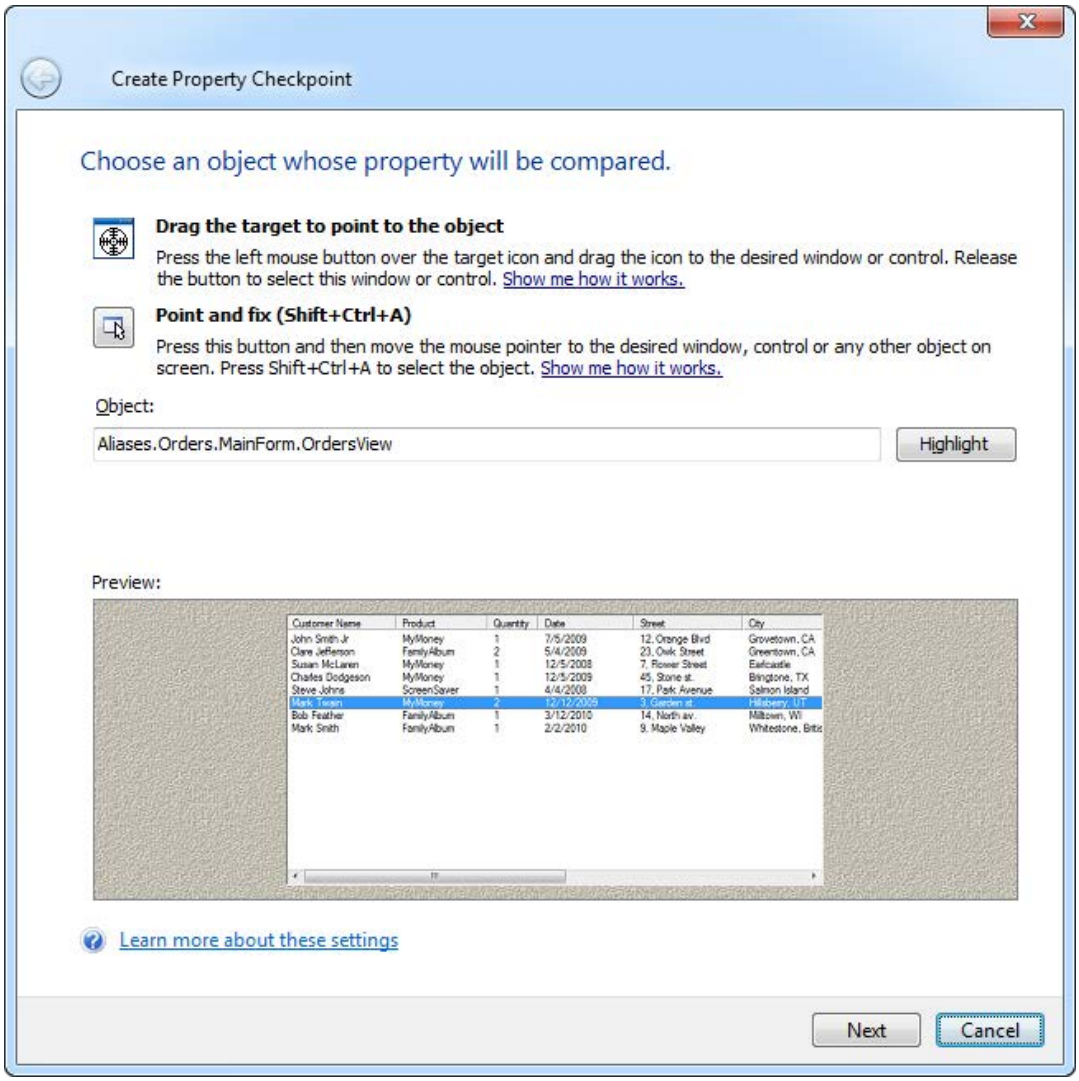
- On the first page of the wizard, click the target glyph ( ) with the left mouse button and keep the button depressed.

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

  Release the mouse button when the target glyph is over the customer list and it is highlighted with the red frame:



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

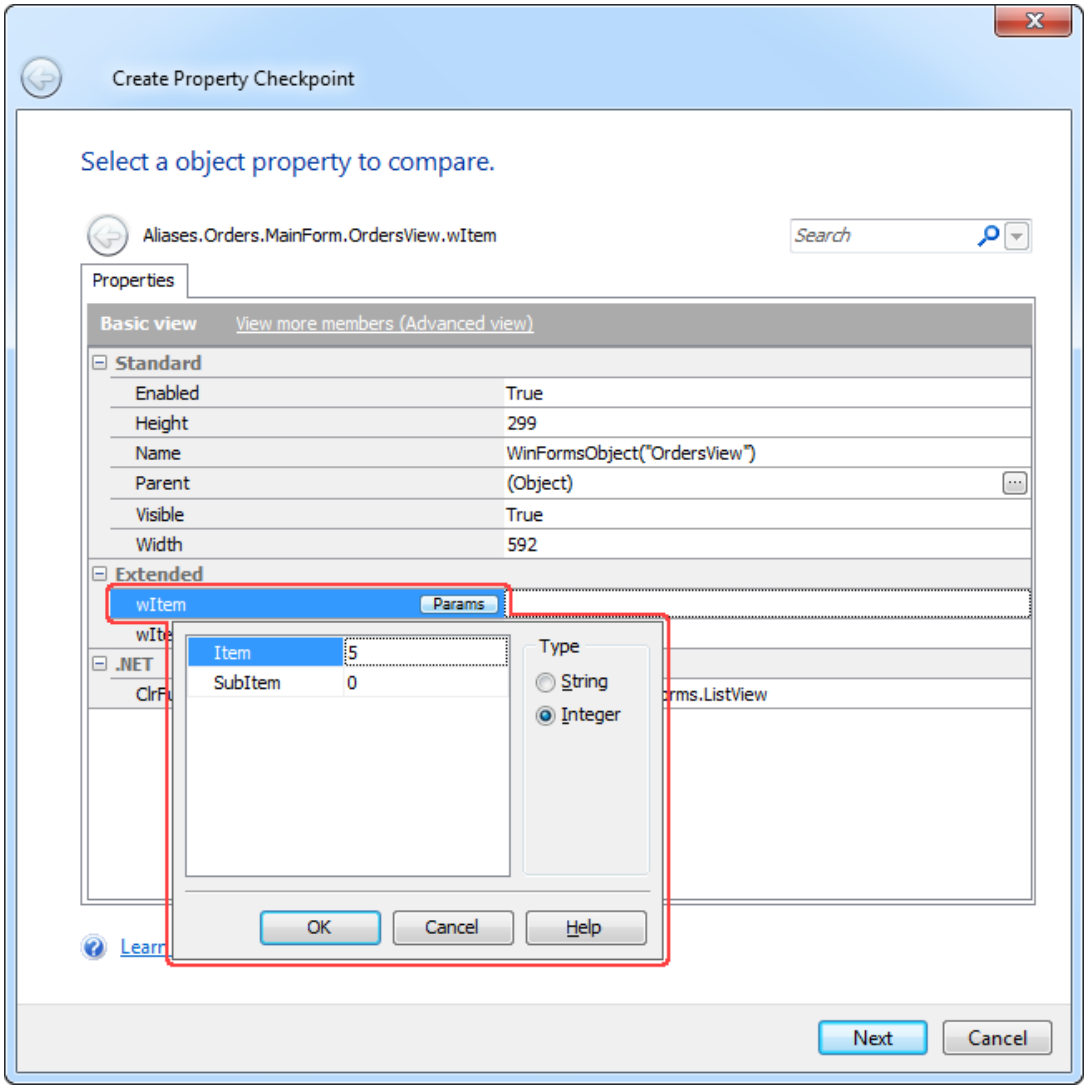


Click **Next** to continue.

- The next page of the wizard displays a list of the selected object's 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. You can see them under the **.NET** node. In our example the list contains only a basic set of properties. To view all available properties, click the **View more members (Advanced view)** link.

  TestComplete appends two groups of properties to the selected object: one group includes properties common for all tested windows and controls. You can see them under the **Standard** node. Another group includes properties that are specific to list-view controls (since the object we selected is a tree view control). The names of these properties start with the letter `w`. You can see them under the **Extended** node. To verify the data, we will use the `wItem` property. It provides access to individual items of tree view controls.

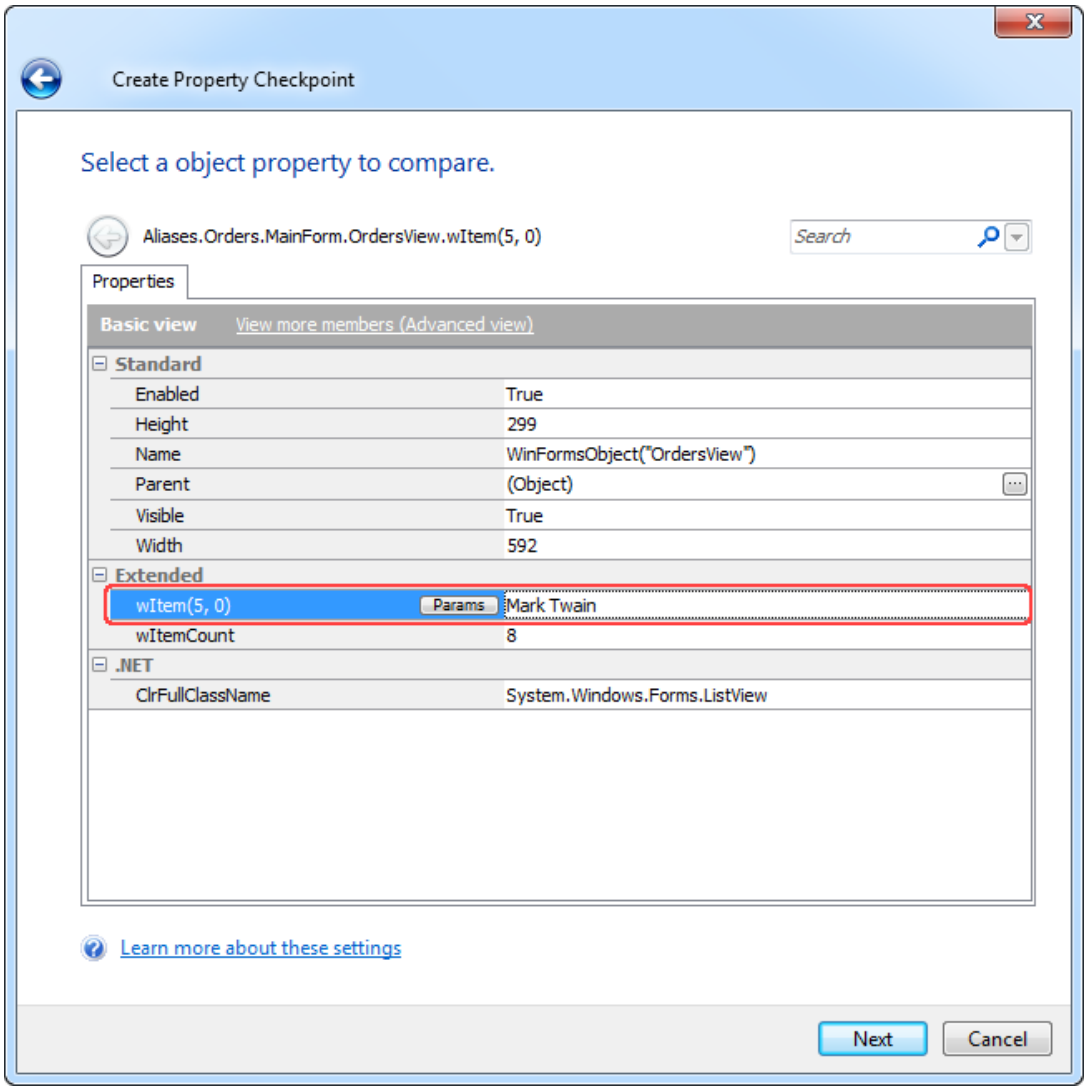
- Find the `wItem` property in the list (it is under the node **Extended**). Click its **Params** button. The following window will pop up:



In this window, specify the cell holding the *Mark Twain* string:

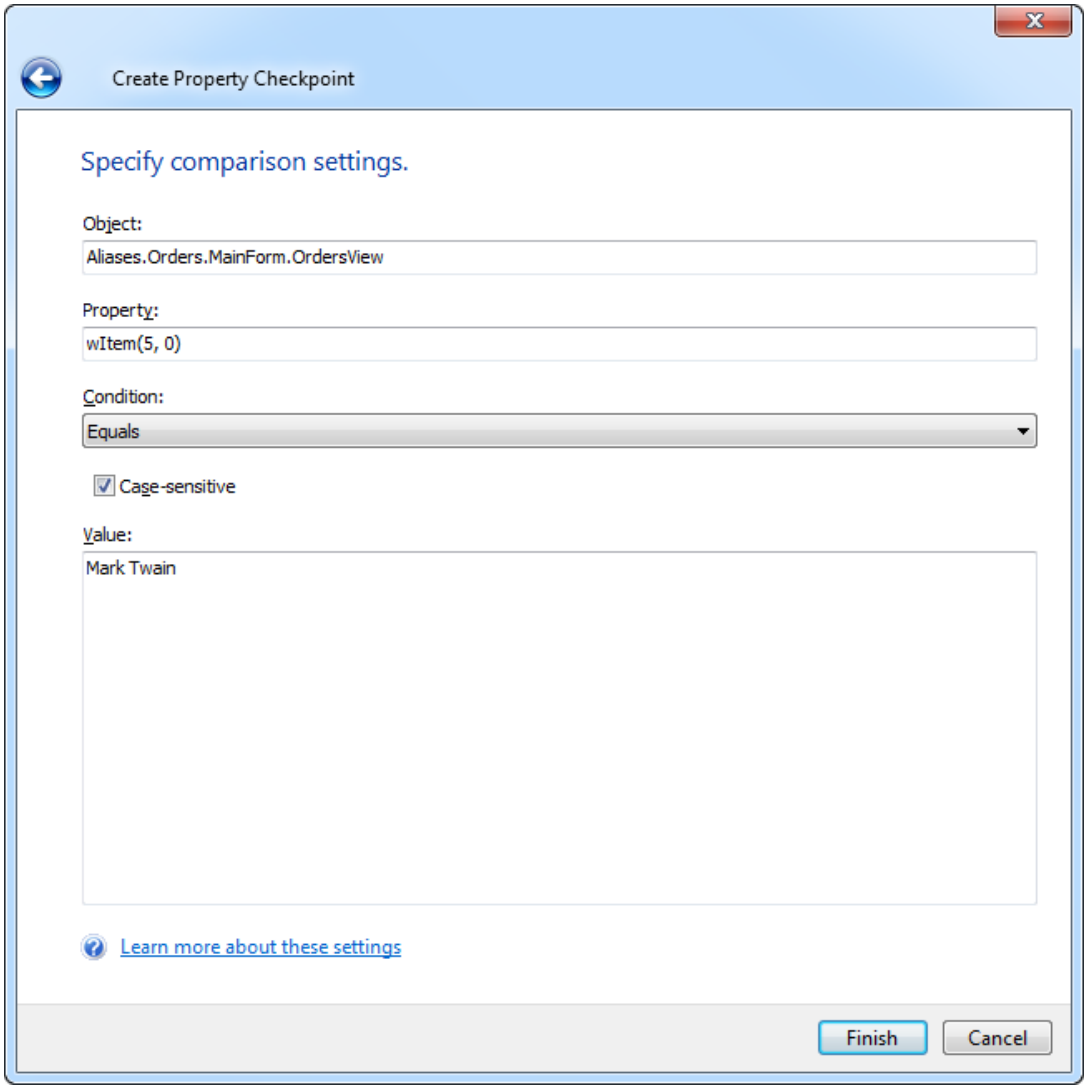
- Select **Integer** in the **Type** section
- Enter **5** into the **Item** box (5 is the index of the *Mark Twain* item in the tree view. Indexes are zero-based).
- Click **OK**.

The test engine will retrieve the item's data and display it in the property list:



Click **Next** to continue.

- On the next page of the wizard you can see the name of the property, whose value will be verified, the comparison condition and baseline data in the **Value** box:



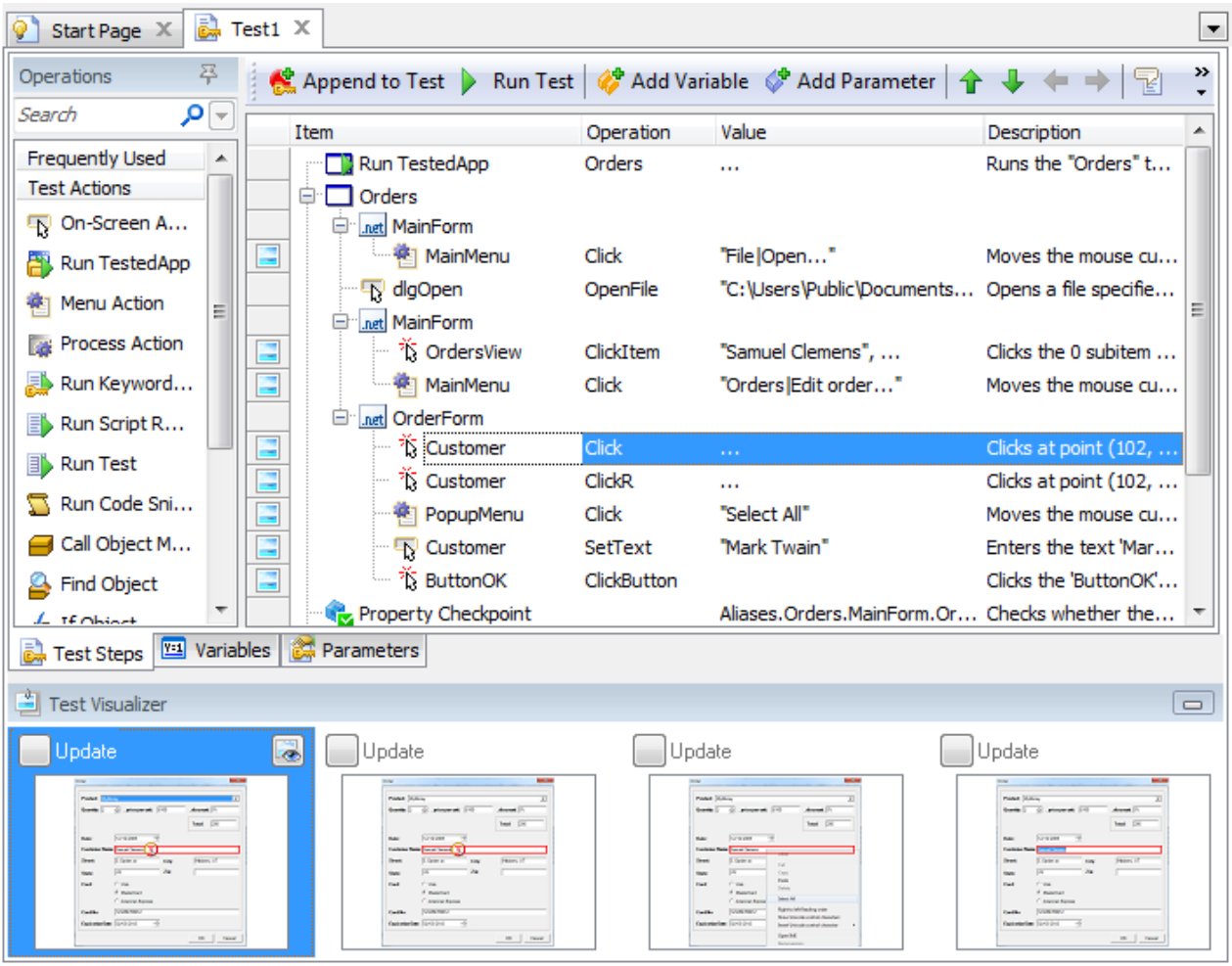
Click **Finish** to complete the checkpoint creation. TestComplete will append the checkpoint command to the recorded test.

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

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

5. Analyzing the Recorded Test

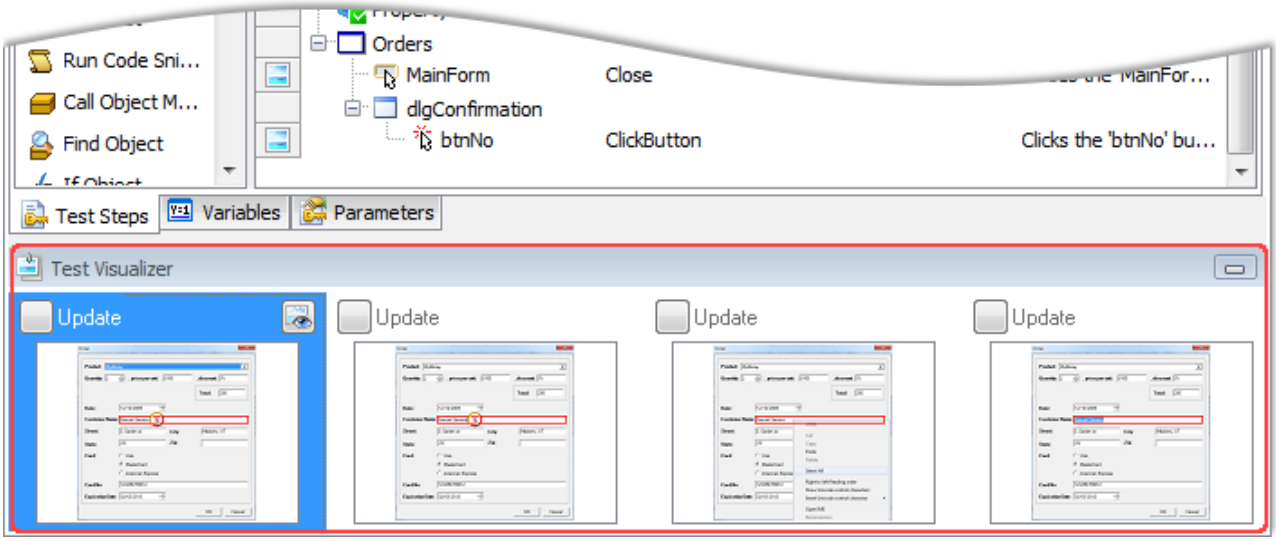
After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test's contents in the Keyword Test editor:



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 Visual C++ 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**.

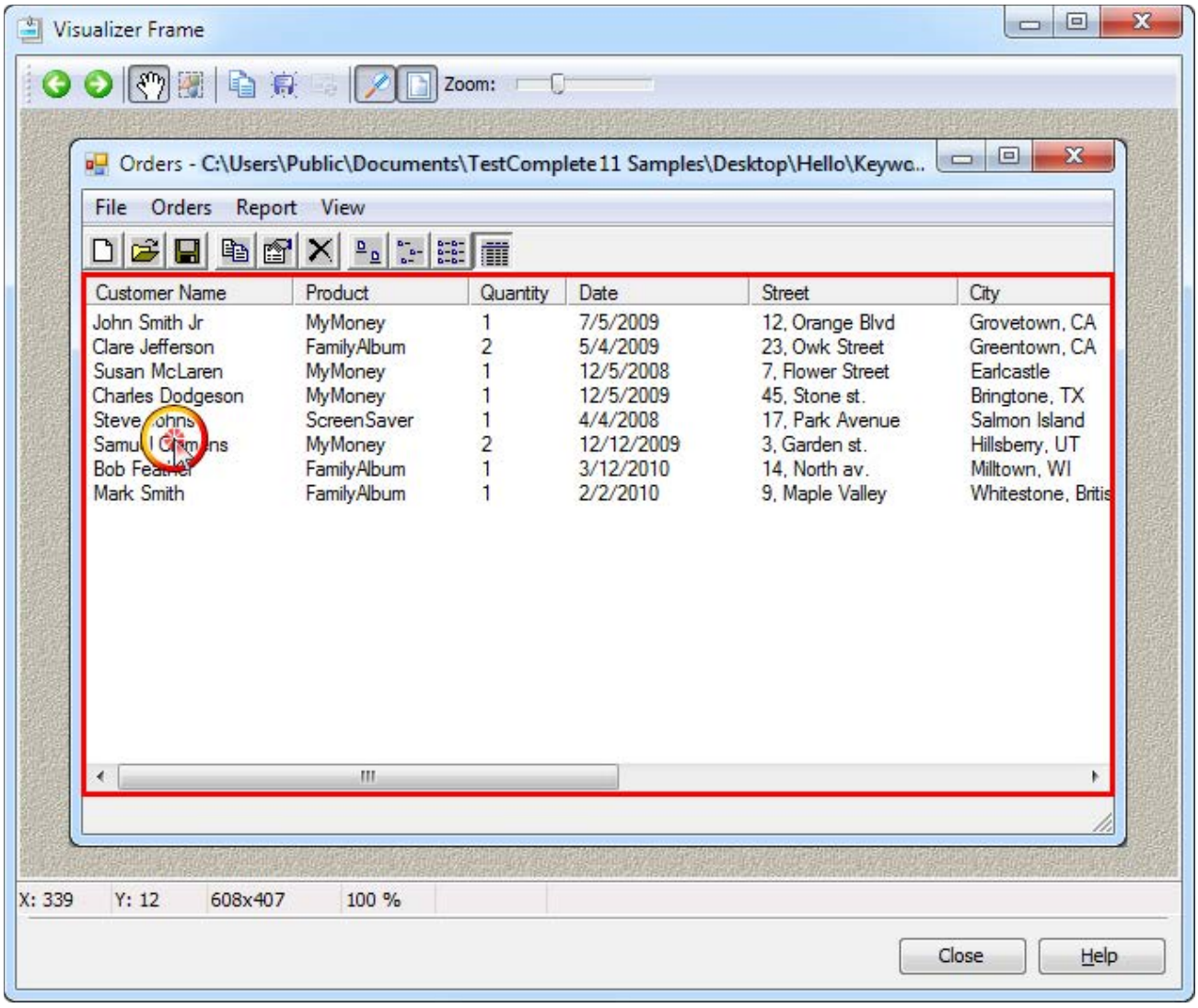
Below the commands there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:



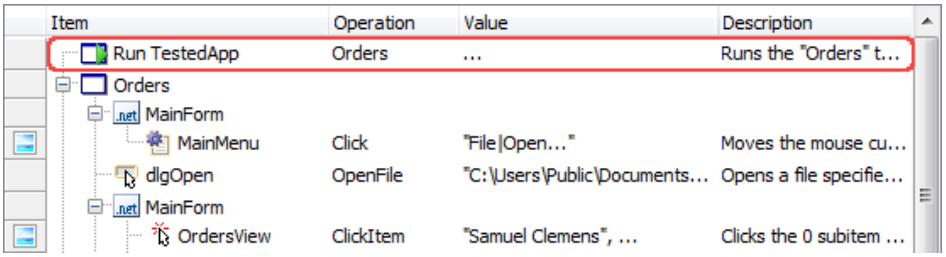
These images illustrate the recoded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (mouse clicks, typing text and so on).

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

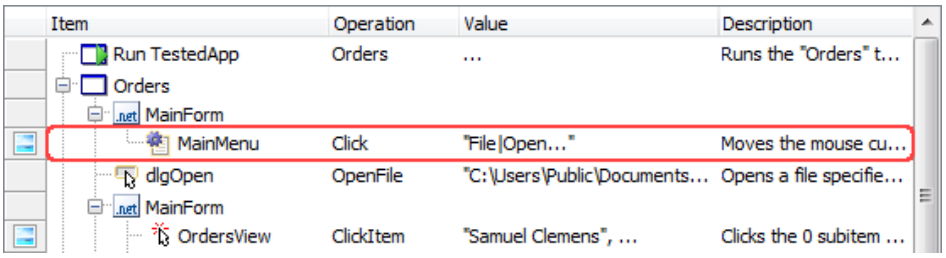
To view the needed image closely, double-click it in the Test Visualizer panel. The **Visualizer Frame** window will appear. This window lets you perform additional actions against the captured images. For example, you can zoom them in and out, save them to a file, navigate through the images, and so on. For more information, see the window description in TestComplete Help.



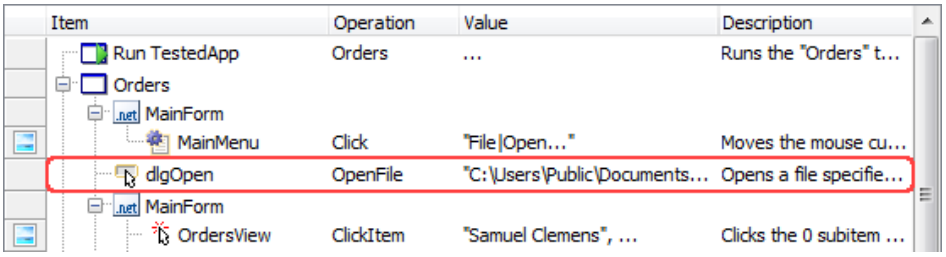
The first operation in the test is **Run TestedApp**. It is used to launch the tested application (in our case, it is the *Orders* application) from a keyword test. TestComplete automatically records this operation when it launches the application automatically or detects an application launch from the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



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



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

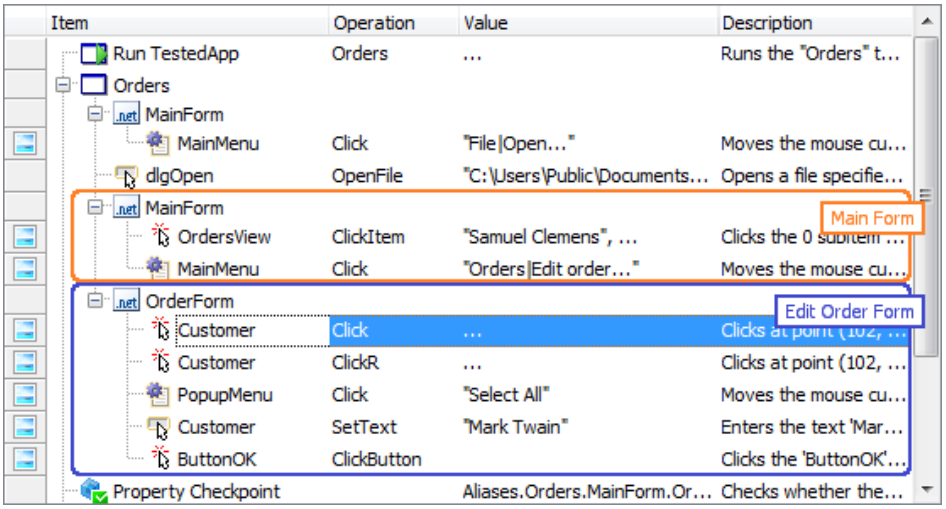


If your computer is running Windows Vista, Windows 7 or later operating system, TestComplete records a sequence of operations that simulate actions you perform when working with the Open File dialog's controls.

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's 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.

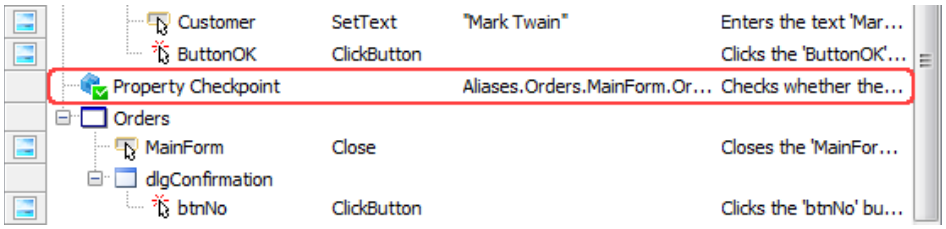
If your test contains a sequence of operations simulating actions over the Open File dialog, you can modify the test and manually replace those operations with the OpenFile method call.

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

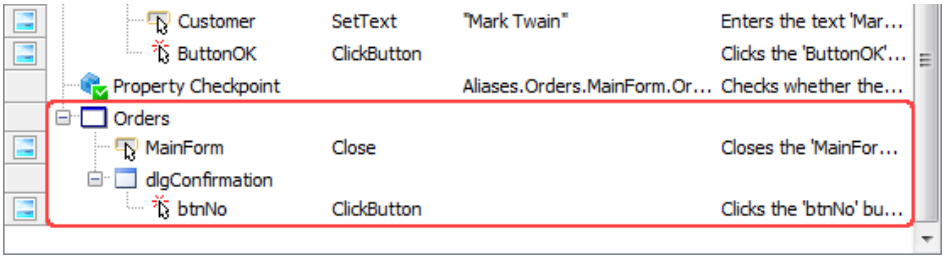


For more information on simulating mouse events, keyboard input and other actions from your scripts, see *Simulating User Actions* in TestComplete Help.

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

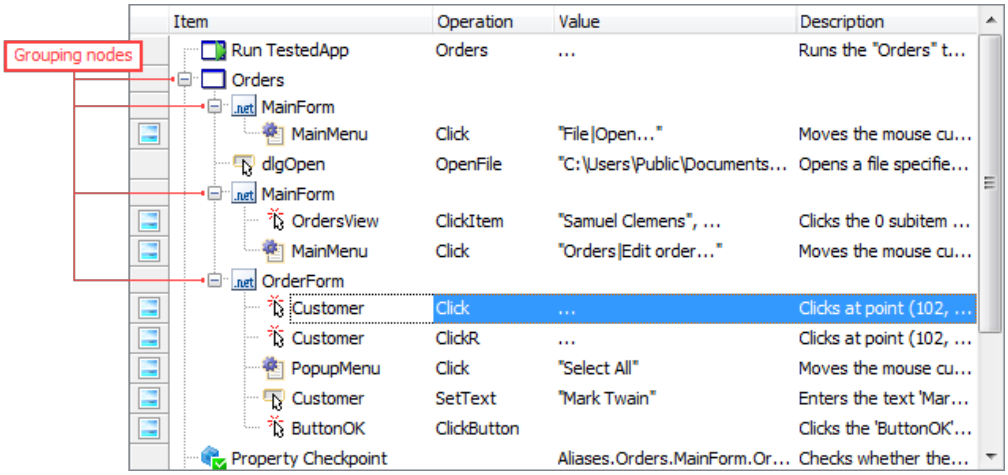


Finally, there is the operation that closes the Orders application and the operation that simulates the "No" button press in the message box.



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:



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 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 hard to determine which window or control corresponds to a name. Using mapped names makes the test

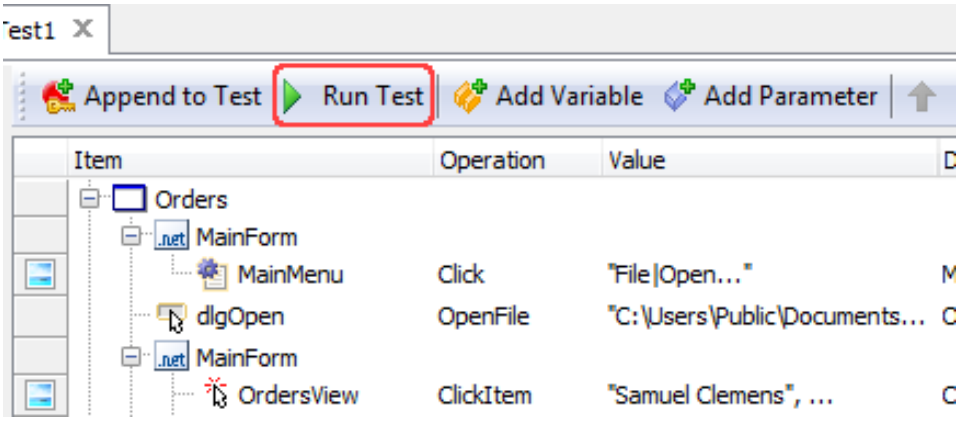
easier-to-understand and more stable. For more information on mapping names, see *Name Mapping* in TestComplete Help.

6. Running 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's IDE.

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



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

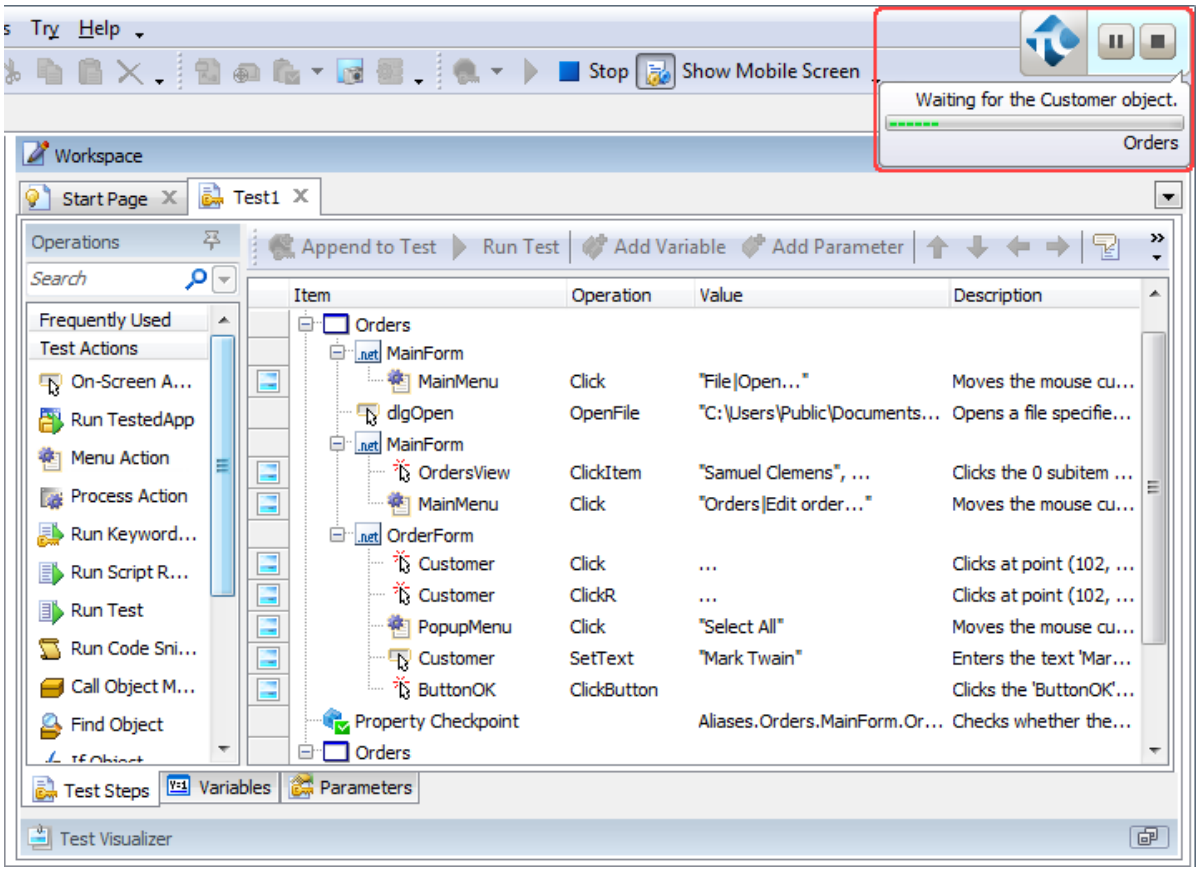
> **Note:** Don't move the mouse or press keys during the test execution. Your actions may interfere with actions simulated by TestComplete and the test execution may go wrong.

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:



  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's main menu.

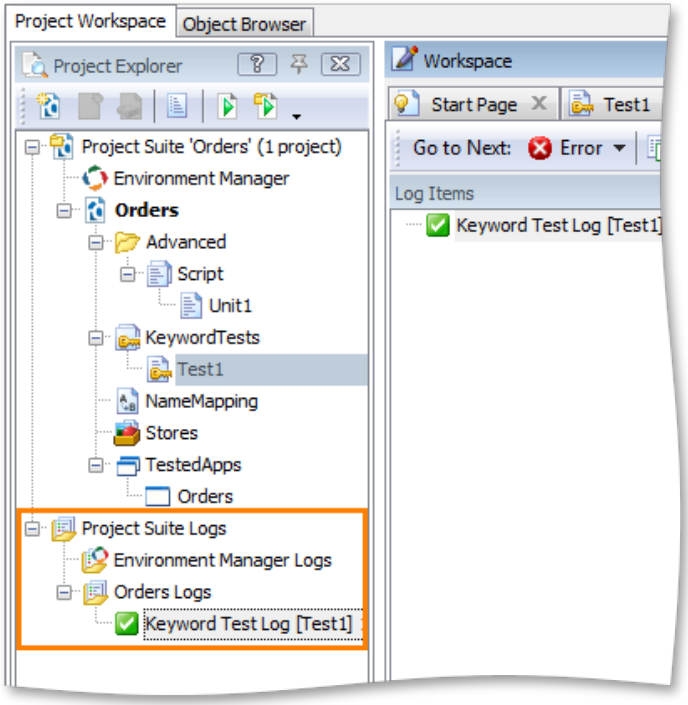
  You can pause the test execution by clicking  **Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check the test's variables and objects using TestComplete's **Watch List** or **Locals** panel or the **Evaluate** dialog (see *Debugging Tests*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

- To launch the test we used the **Run Test** button on the test editor's 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.

7. Analyzing Test Results

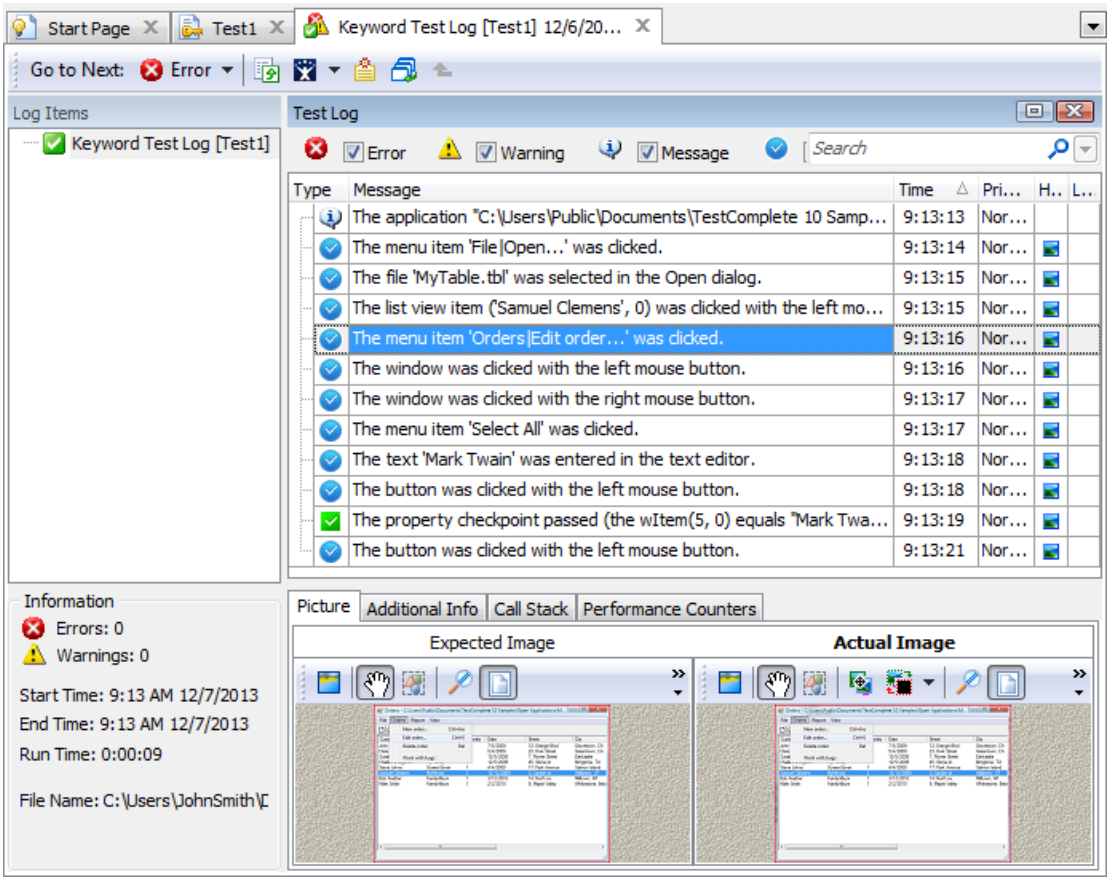
TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the **Project Explorer** panel under the **Project Suite Logs | Orders Log** node. This is the primary workspace for looking up the test history of the project and project suite. Each 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.

In our example, the log is as follows –



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.

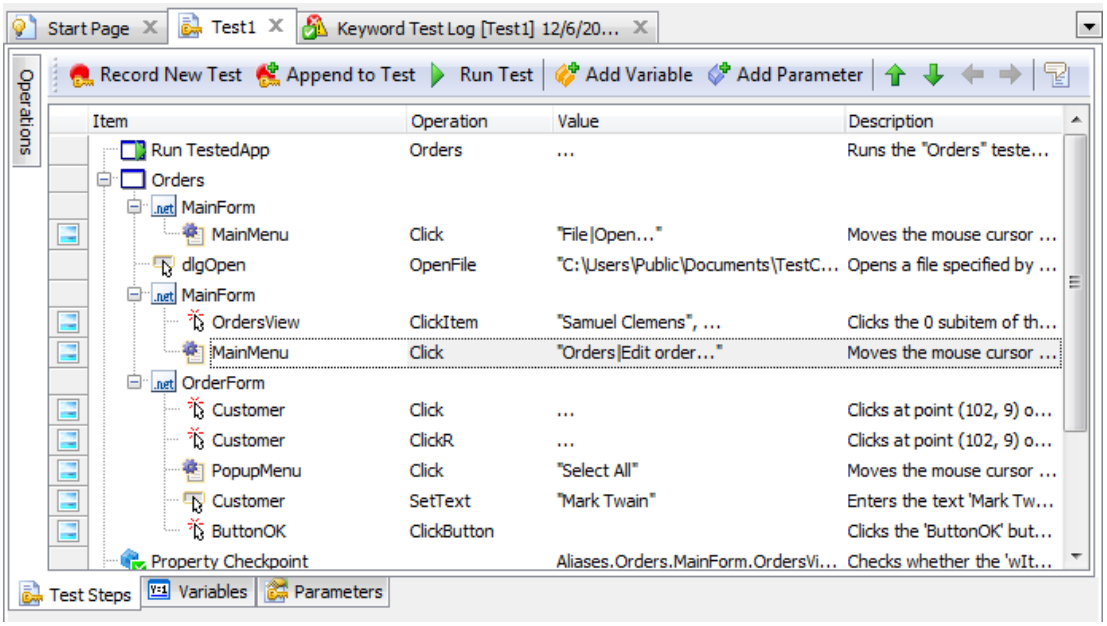
TestComplete may post additional text and images along with the message. To view them, simply select the desired message in the log and look in the **Additional Information** and **Picture** panes that are below the message list. For instance, on the image above the Picture pane displays the screenshots 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.) The test log includes a special button that lets you compare the images and easily see the difference. 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.

The log's **Call Stack** pane displays the hierarchy of test calls that led to posting the selected message to the log.

The log's **Performance Counters** pane displays values of the performance counters monitored during the test run. The values are shown in the form of graphs.

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 menu item 'Orders|Edit order…' was clicked" message in the log, TestComplete will highlight the keyword test operation that performed this action:



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 instance, developers could change the application's behavior, the recognition attributes of windows and control 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 them, and so on.

One of the most typical reasons which novice users face is the difference in the application's 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 the test, it also must be running before you run the test; if the tested web page was opened on the second tab of your web browser when you recorded your test, it should also be opened on the second tab when you run the test, and so on.

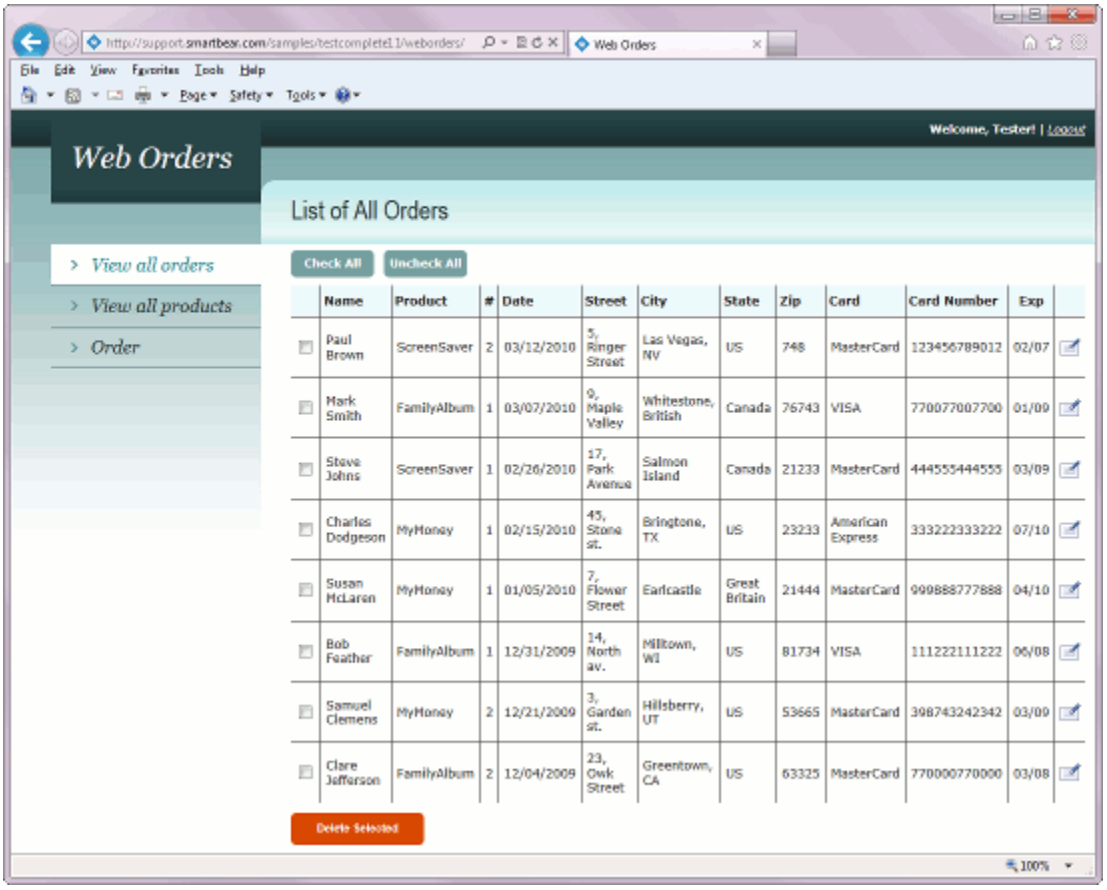
For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* in TestComplete Help.

Testing Web Applications

This section provides a step-by-step tutorial that describes how to create a test project in TestComplete, record and play back a simple web test, and analyze the results. 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 Web Orders application that can be found on our web site: http://support.smartbear.com/samples/testcomplete12/weborders/. This application displays a list of orders and contains special functions for adding, deleting, modifying and exporting orders.

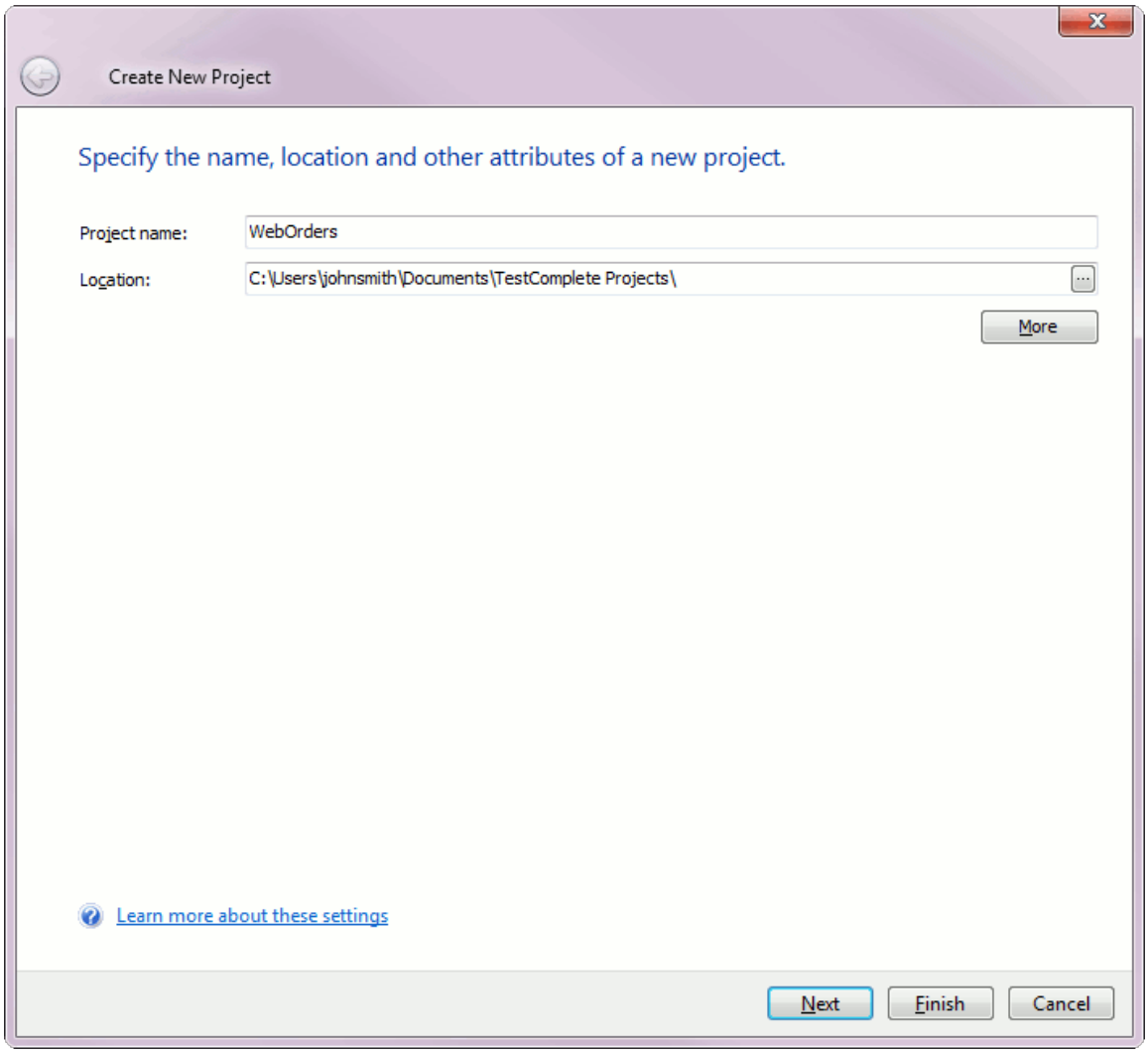


1. Creating a Test Project

Let's create a new test project:

1. If you have a project or project suite opened in TestComplete, close it. To do this, choose **File | Close** from TestComplete's main menu.

2. Select **File | New | New Project** from TestComplete's main menu. This will call up the **Create New Project** wizard:



3. On the first page of the wizard, you can specify the project name and location. Enter *WebOrders* into the **Project name** edit box. TestComplete will automatically generate the project path and display it in the **Location** field. The project folder is used to store all information generated for or by the project: keyword tests, scripts, test logs, stores, and so on. You can change the project's folder in the Location box. In our example we will keep the folder name unchanged.

   You can also specify the project suite name and its actual location by clicking the **More** button and filling in the corresponding edit fields. In our example, we will keep the project suite name and location unchanged.

4. After you specify the project name and location, click **Next** to continue.

We will continue working with the wizard and use its pages to select the needed project type and specify some project settings.

2. Selecting a Test Type

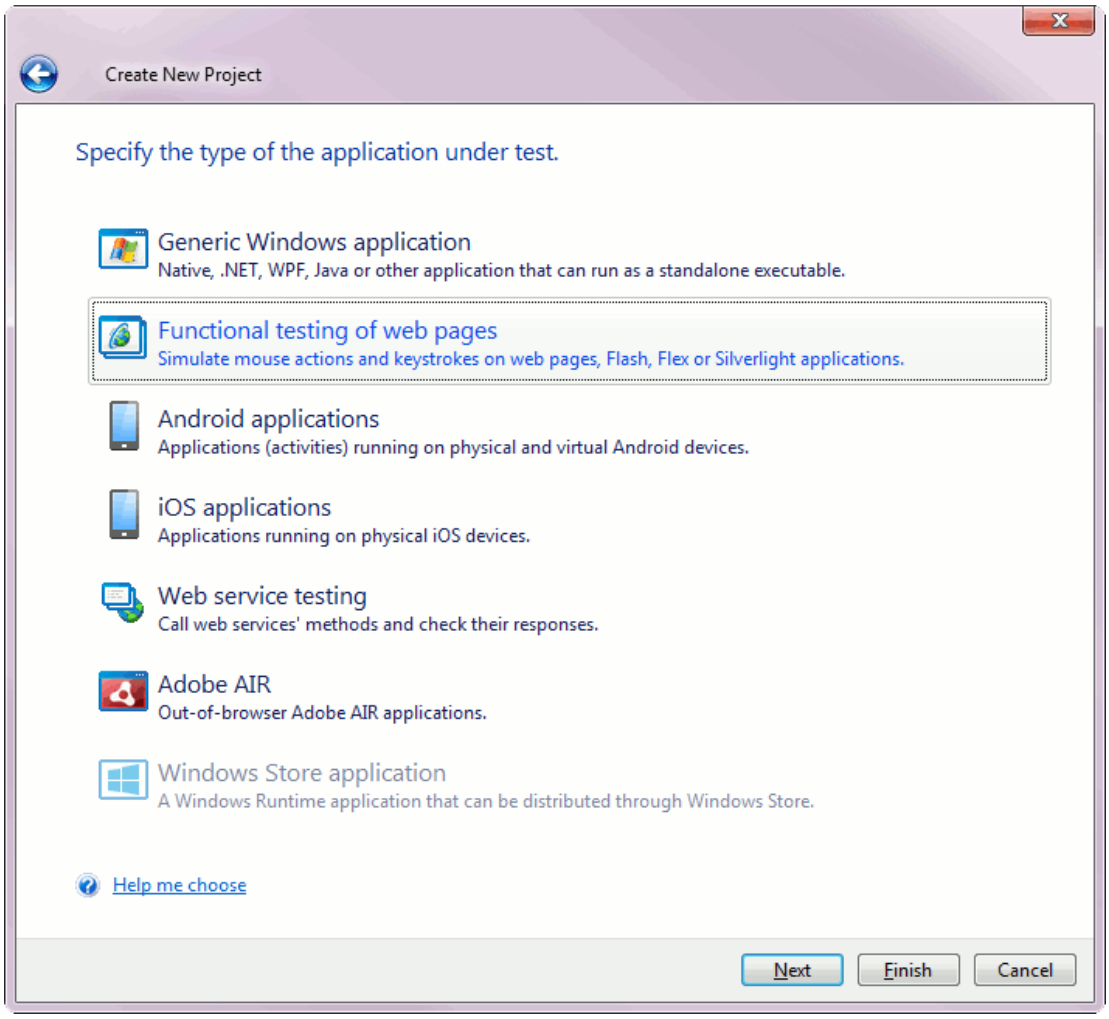
Each TestComplete project may have different sets of project items added to it. However, creating a certain type of a test (a web test, a test for Adobe AIR applications and so on) requires a certain number of added items.

There are several ways to add all required project items to the project:

- You can do that with the **Create New Project** wizard while creating the project. In this case, TestComplete will generate the project structure that corresponds to the selected test type right after you finish creating the project.
- You can do that at any time later manually by adding needed items via the context menu of the **Project Explorer** panel.

In this tutorial, we will generate the required project structure by using the Create New Project wizard.

1. After you specify the project name and location on the first page of the wizard, the wizard shows the second page where you can choose the type of your tested application:



   As you may remember, we are going to test the *Web Orders* application that is located on the web page. In the wizard, it falls under the Web category. So, click **Functional testing of web pages** and, if

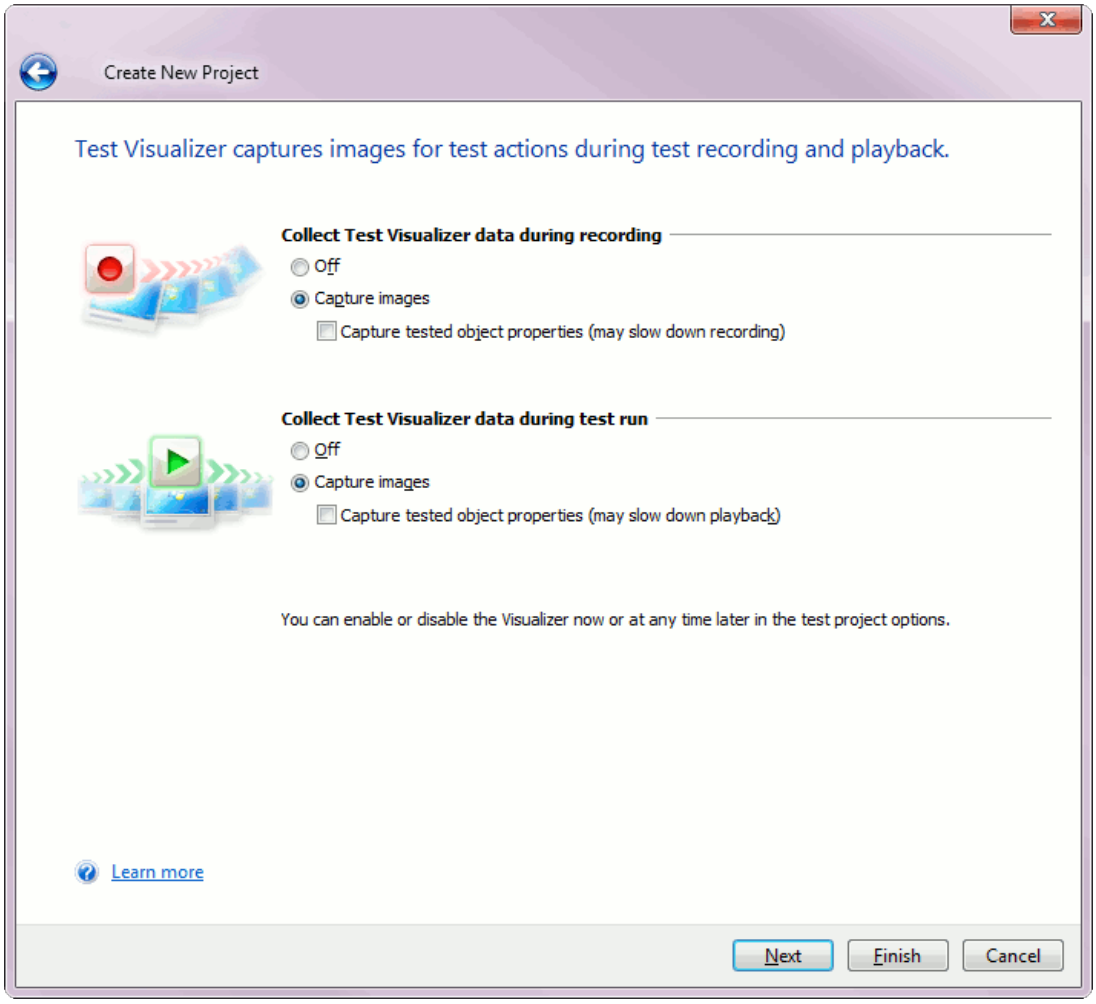
you use Windows XP, click **Next** to continue. On Windows Vista and later versions of the operating system, the wizard will switch to the next page automatically after you click the category name.

In the next topic, we will go through the rest pages of the wizard and complete the project creation.

3. Completing the Project Creation

In the previous step, we decided on the needed test type. Now, let's quickly go through the rest of the wizard's pages and complete the project creation:

1. After you have specified the type of your tested application, the wizard displays the page where you can enable or disable TestComplete's Test Visualizer functionality:



Test Visualizer captures information for test actions during test recording and playback. Depending on the selected options, Test Visualizer frames can contain screenshots only or screenshots along with information on the objects they contain.

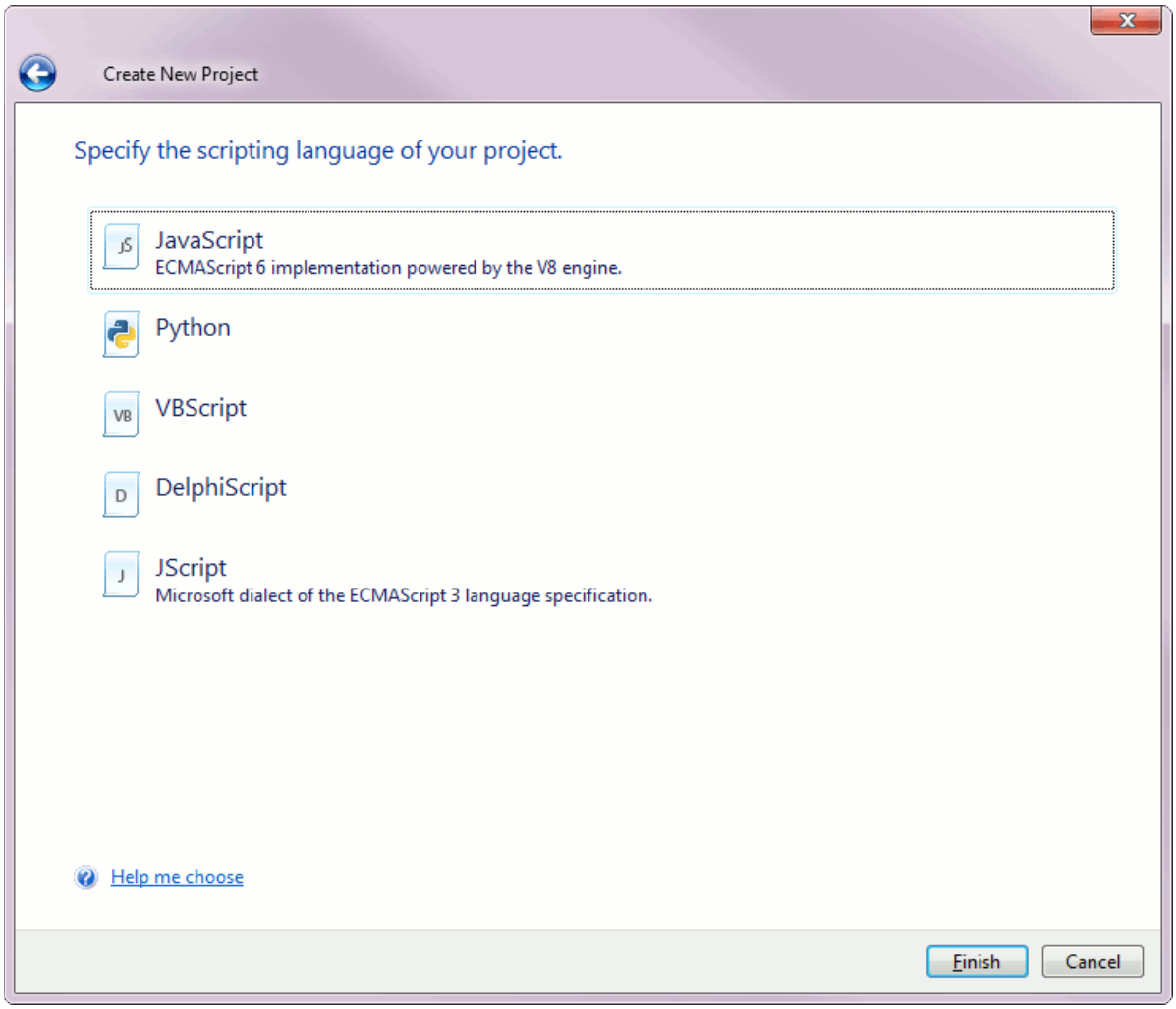
The Visualizer frames that were captured during recording help you better understand what the recorded test commands do, what is important when you have just started learning the product.

The Visualizer frames captured during test execution let you easily determine what happens to the tested application or system at that time. This information is helpful when you are debugging errors.

However, the images and object data occupy hard disk space and in large projects they may be the reason of significant increase of the size of test result files. So, you can limit the amount of collected data by capturing images only, or, if Visualizer is not needed, you may disable it and enable it at any time later using your project's settings.

In our tutorial, we select **Capture images** both for test recording and playback and leave the **Capture tested object properties** unselected. Then, click **Next** to continue.

2. On the next page of the wizard, you can choose the scripting language to be used in your project.



Every TestComplete project uses one of the supported scripting languages: JavaScript, JScript, Python, VBScript, DelphiScript, C#Script or C++Script. The scripting language is important even if you are not going to use script units in your project. Even if you are going to use only keyword tests, you may need to call code snippets or use script statements to specify operation parameters.

The scripting language is also important because it defines the format of object names with which your tests will work, regardless of whether you are using scripts or keyword tests. The name format depends on the language syntax. For instance, in JavaScript, JScript, Python and VBScript, the name of the Internet Explorer process looks like `Browser("iexplore")`. In DelphiScript you should replace double quotes with single quotes, that is, `Browser('iexplore')`; and in C#Script and C++Script, the word Process should be enclosed in brackets: `["Browser "]("iexplore")`.

For more information on choosing the scripting language, see *Selecting the Scripting Language* in TestComplete Help.

In this tutorial, we will use VBScript. So, select **VBScript** on the page. On Windows Vista and later versions of the operating system, this will close the wizard. If you are using Windows XP, click **Finish**.

TestComplete will create a new project, *WebOrders.mds*, and a project suite for it. It will then display the project suite's and the project's contents in the **Project Explorer** panel.

Now, we need to prepare the web browser for further test steps.

4. Preparing Web Browser

Creating web tests with TestComplete requires that the web browser used for recording and playing back the test be configured 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.

5. Creating a Test

Planning a Test for the Web Orders Application

The sample Web Orders application works with a list of orders. Suppose you need to test whether the application's Edit Order page functions properly and modifies data in the order list. In this case –

- **Test purpose:** The test should check whether the Edit Order page saves the modified data and the changes are visible in the order list.
- **Testing steps:** Our test should simulate modifying the order's 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.

Creating Tests in TestComplete

TestComplete provides two approaches to creating tests. You can:

- Create tests manually
- Record tests

When you are creating a test manually, you enter all the needed commands and actions that your test must perform via appropriate script objects or keyword test commands. This approach is very helpful when you need to create very powerful and flexible tests or if you are experienced in creating tests.

However, creating tests manually requires much time and does not prevent you from different kinds of problems. For example, when creating tests manually, you must know the classes and names of your application's objects you want to work with. To solve such problems, TestComplete includes a special feature that lets you create tests easily: you just perform some actions against the tested application, TestComplete recognizes these actions and then converts them to appropriate script lines or keyword test operations. We call this feature "**recording a test**", because you create a test visually and in one sense you record your actions as script or a keyword test. It is a very useful approach and it does not require much experience in creating tests. So, in this tutorial, we demonstrate how to record tests with TestComplete. For more information on recording tests, see the section below.

Recording Tests in TestComplete

The recording includes three steps:

1. You start recording by selecting **Test | Record | Record Keyword Test** or **Test | Record | Record Script** from TestComplete's main menu or from the Test Engine toolbar. You can also start recording by clicking **Record a New Test** on the Start Page.

   With TestComplete 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 is started. The main recorded test will contain special commands that will run these tests.

   After you command TestComplete to start the recording, it will switch to the recording mode and display the Recording toolbar on screen:

   The toolbar contains items that let you perform additional actions during the recording, pause or stop recording and change the type of the recorded test (from the keyword test to script code, or vice versa).

2. After you start recording, perform the desired test actions: launch the desired browser and load a page to it (if needed), then click some command buttons on the page, select menu items, type some 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 the Help.

Recording a Test for the Web Orders Application

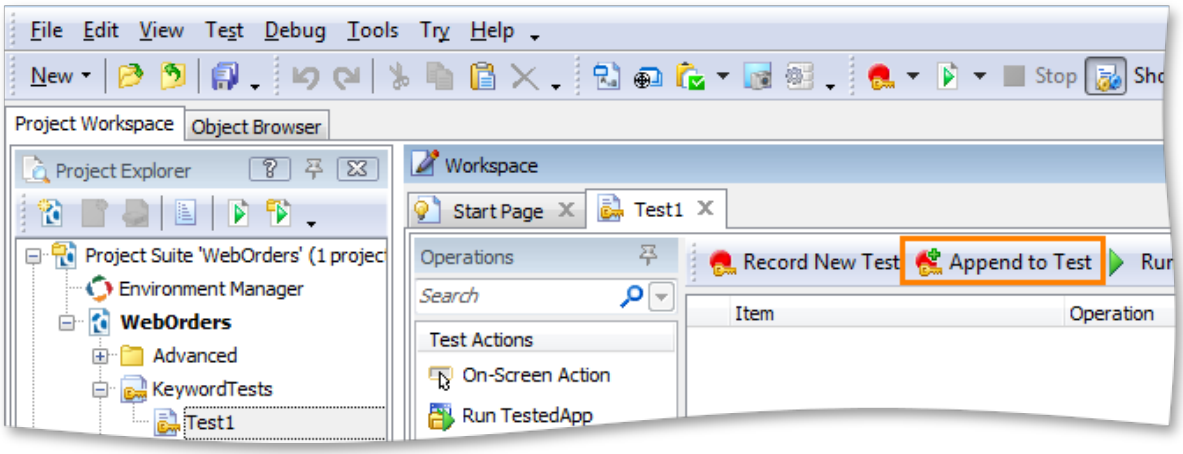
Now let's record a keyword test for the sample Web Orders application. The test will launch the browser, load the specified page to it, simulate clicks and keystrokes within the application's window and verify the application's data.

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.

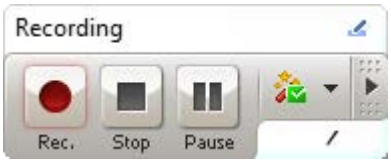
Let's start recording:

1. When creating a new project, TestComplete automatically creates an empty keyword test in this project. Let's record test commands into this test. To start recording, select the **Append to Test** item on the test editor's toolbar:



TestComplete will display the Recording toolbar on screen. If the Interactive Help panel is visible, TestComplete will also show information about the recording in it.

By default, the Recording toolbar is collapsed:

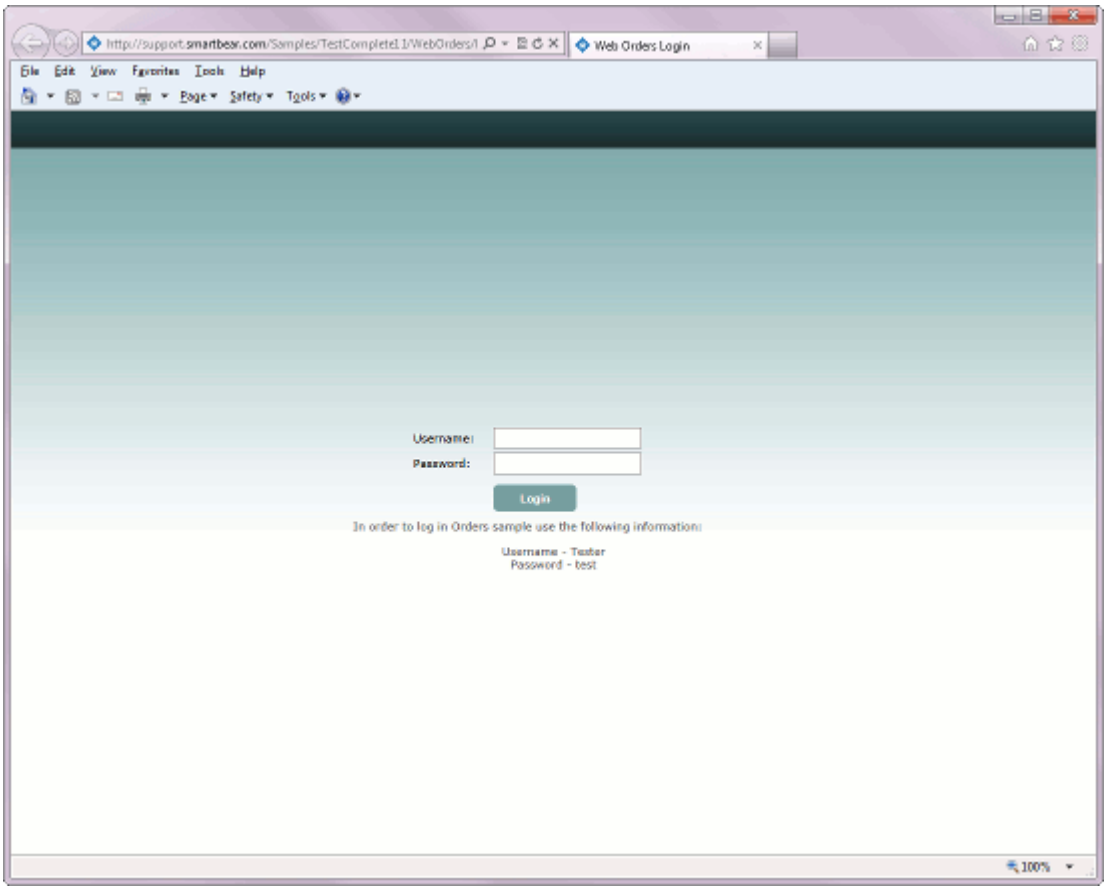


Click the ▶ arrow button to expand the Recording toolbar and view all its buttons:

2. Launch the desired browser and navigate it to the http://support.smartbear.com/samples/testcomplete12/weborders/ web page.

TestComplete records the browser launch by using a special test command. You will see the command later when we analyze the recorded test.

3. Wait until the Internet browser starts and the application's login window is shown:



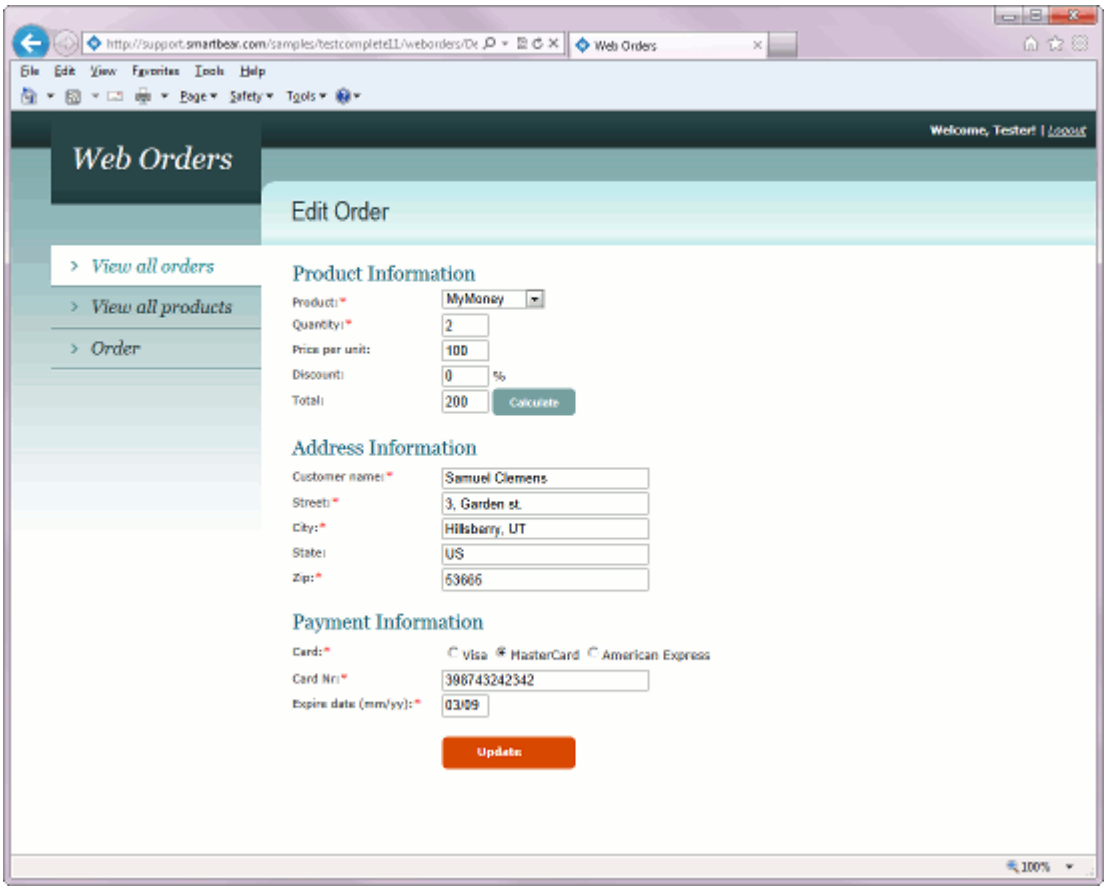
If the Interactive Help panel is visible, resize or move it so that it does not overlap the application's window. Your actions on this panel are not recorded.

4. Log in to the application using the information shown at the bottom of the page:

   - **Username**: Tester
   - **Password**: test

   Click **Login** to switch to the main page of the application.

5. Now, the browser shows the default page of the application - the page that contains all the existing orders. Now, let's modify one of the . For this purpose, click the **Edit** icon next to the order made by *Samuel Clemens*. This will open the **Edit Order** page.



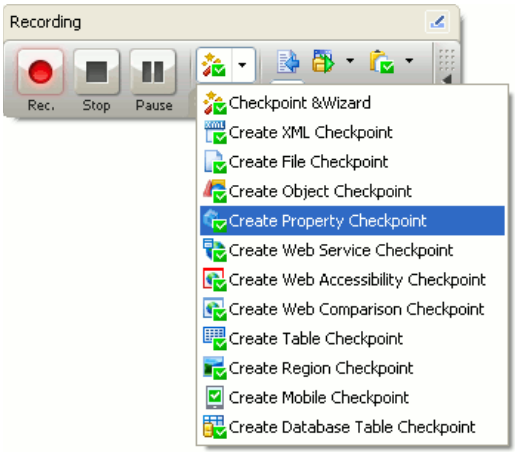
6. In the window, click within the **Customer name** text box and enter *Mark Twain* as the customer name.

7. Click **Update** to close the editing page. The application will update the customer list shown in the main window.

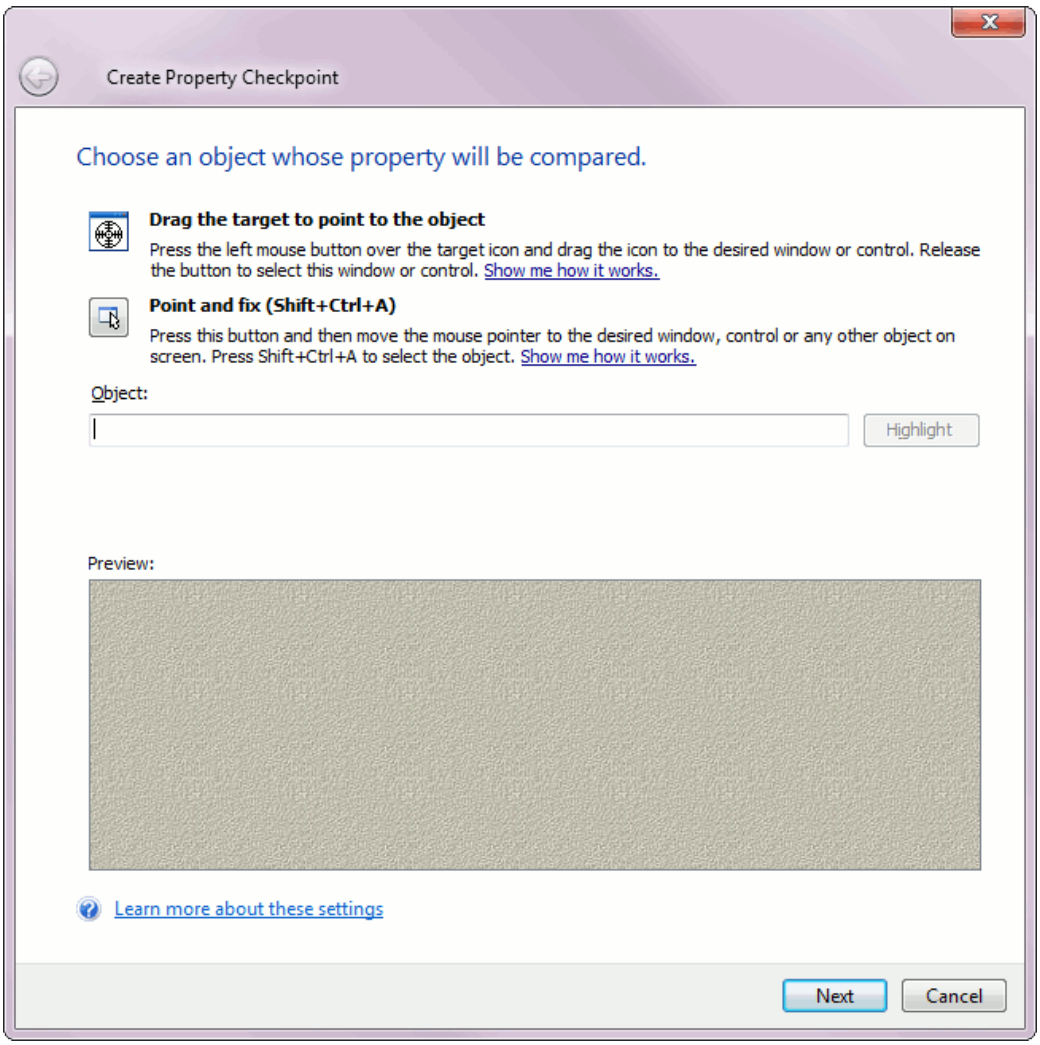
8. Now let's insert a comparison command into our test. It will verify that the application's customer list displays the modified name - *Mark Twain*.

   We call the comparison commands **checkpoints**. TestComplete offers various types of checkpoints that are suitable for verifying different types of data (see Checkpoints). 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.

- Select **Create Property Checkpoint** from the **Checkpoint** drop-down list of the Recording toolbar:



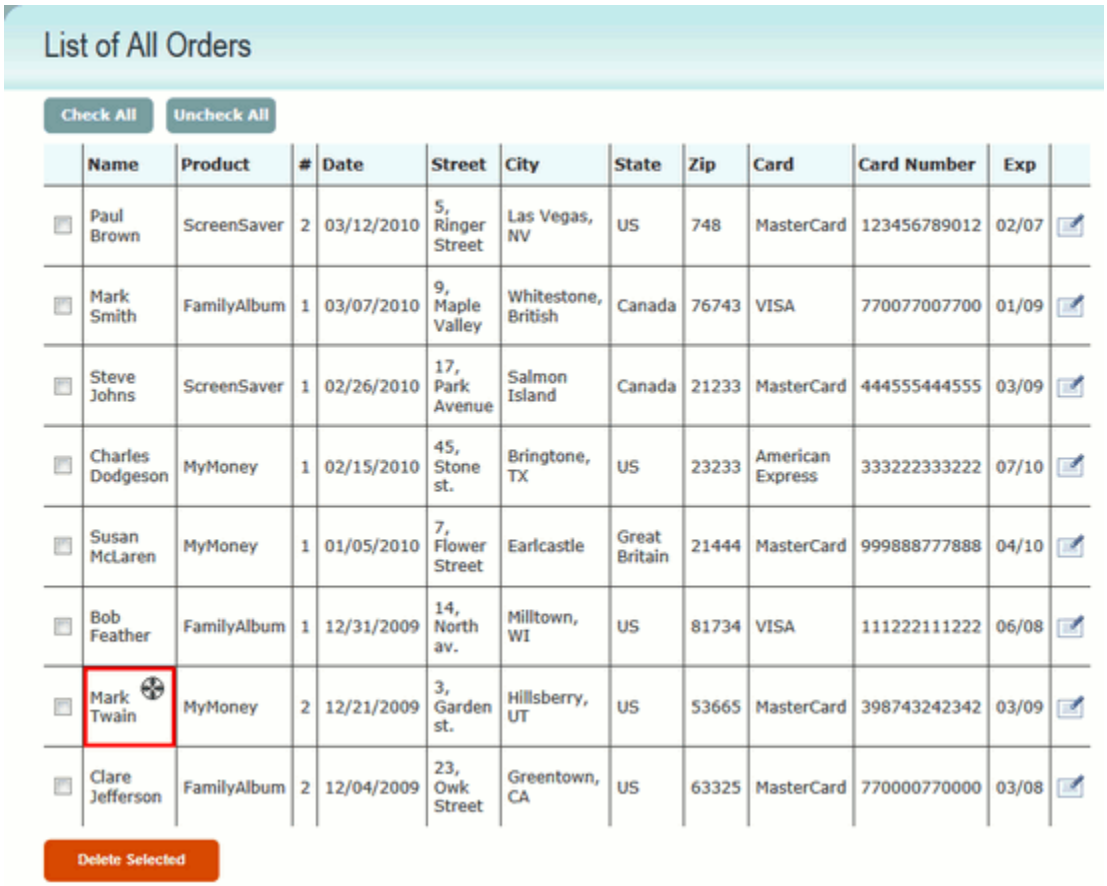
This will invoke the **Property Checkpoint** wizard. It will guide you through the process of checkpoint creation:



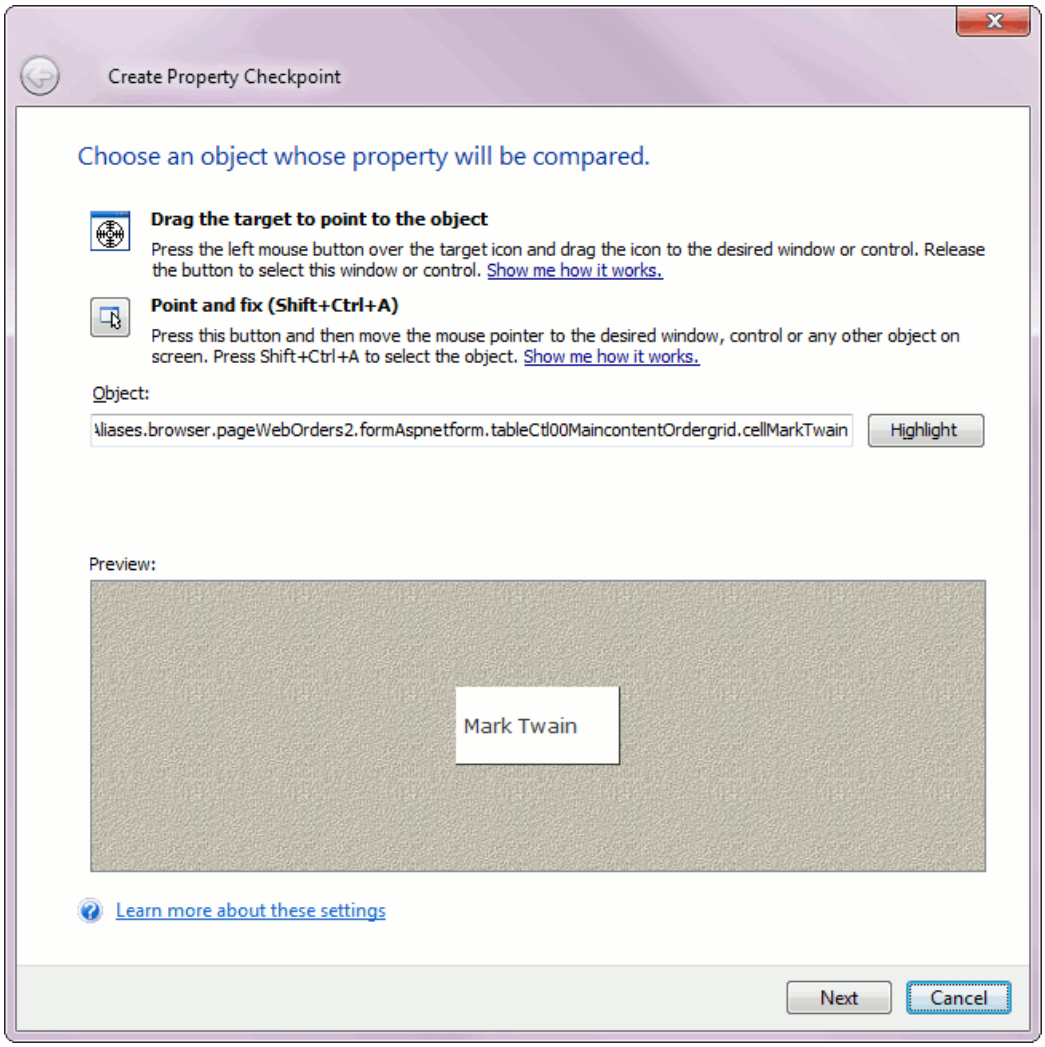
- On the first page of the wizard, click the target glyph () with the left mouse button and keep the button pressed.

  Wait until the wizard minimizes and then drag the icon to the cell that contains the *Mark Twain* string. While you are dragging, TestComplete will highlight the controls and windows under the mouse cursor with the red frame.

  Release the mouse button when the target glyph is over the first cell of the **Name** column and when it is highlighted with a red frame:



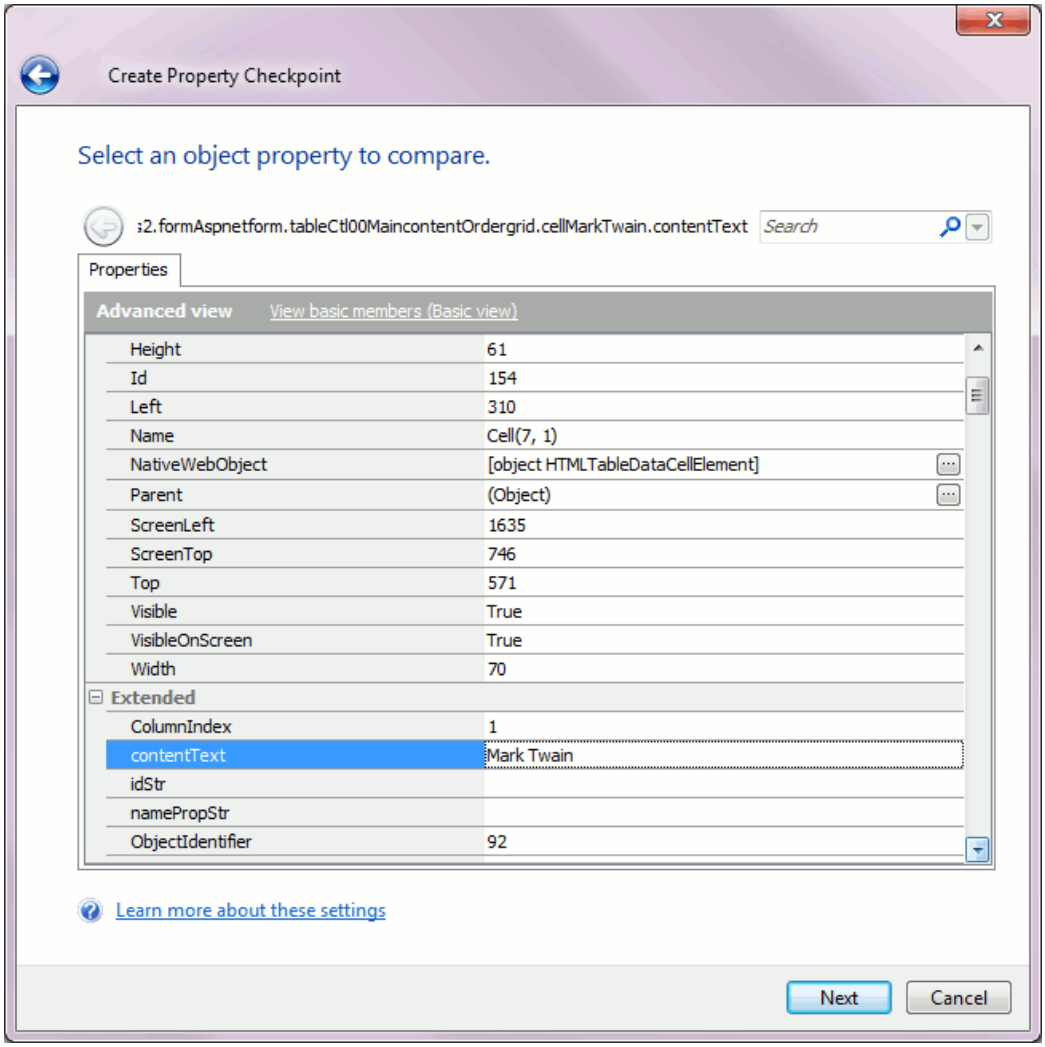
- After you release the mouse button, TestComplete will restore the wizard and display the name of the selected object in the **Object** box and the image of the object below it:



Click **Next** to continue.

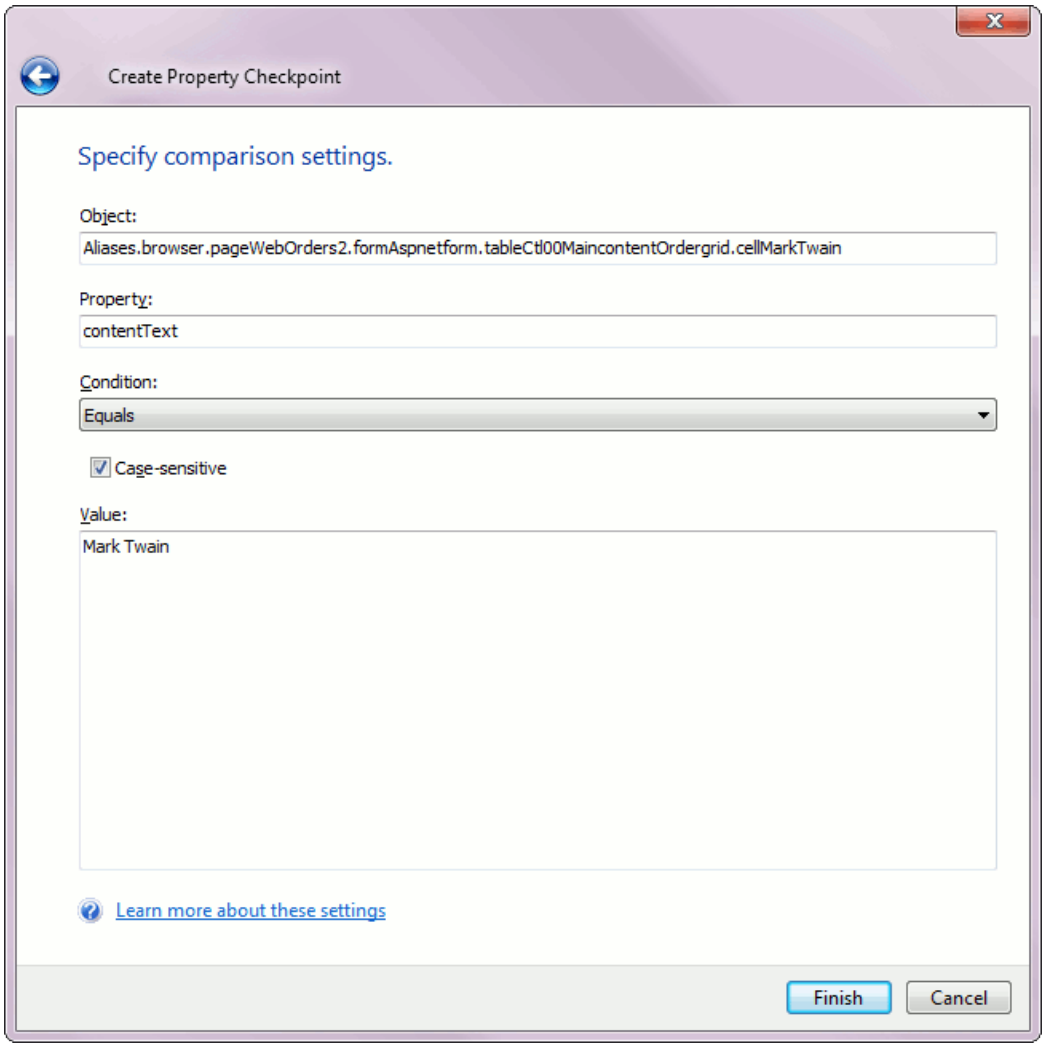
- The next page of the wizard displays a list of the selected object's properties. This list includes properties provided by TestComplete as well as properties defined by the tested application. To view all available properties, click the **View more members (Advanced View)** link.

  TestComplete appends three groups of properties to the selected object: one group includes properties common for all tested windows and controls. You can see them under the **Standard** node. Another group includes properties that are specific to grid controls (since the object we selected is a grid control). You can see these properties under the **Extended** node. The third group includes the properties provided by the Internet browser. You can see these properties under the node with the name of the used browser. To verify the data, we will use the `contentText` browser-independent property. It provides access to the text shown in the selected cell and helps you avoid problems with playing back the test in different browsers.



- Find the `contentText` property in the list. Select this property and then click **Next** to continue.

- On the next page of the wizard you can see the name of the property, whose value will be verified, the comparison condition and baseline data in the **Value** box:



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

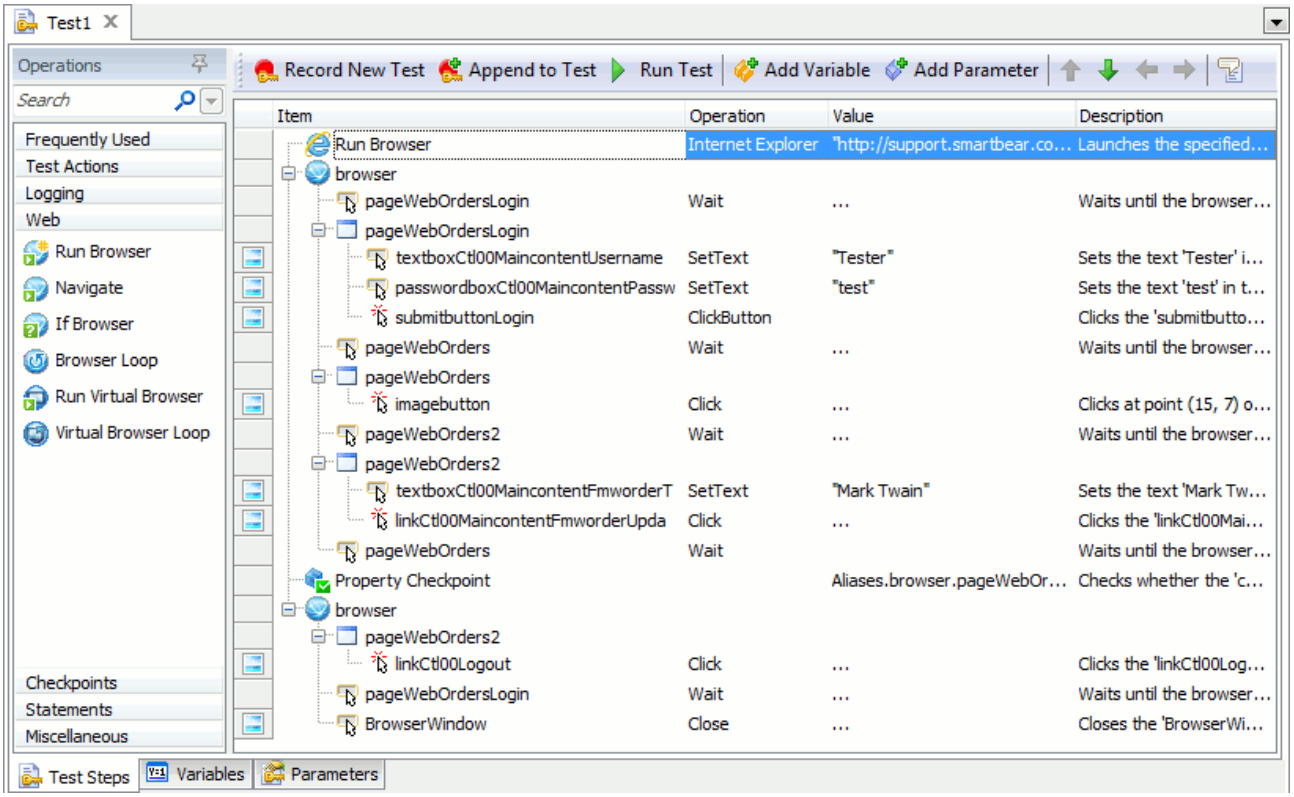
9. Click the **Logout** link in the upper-right corner of the page.

10. Close the Internet 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.

6. Analyzing the Recorded Test

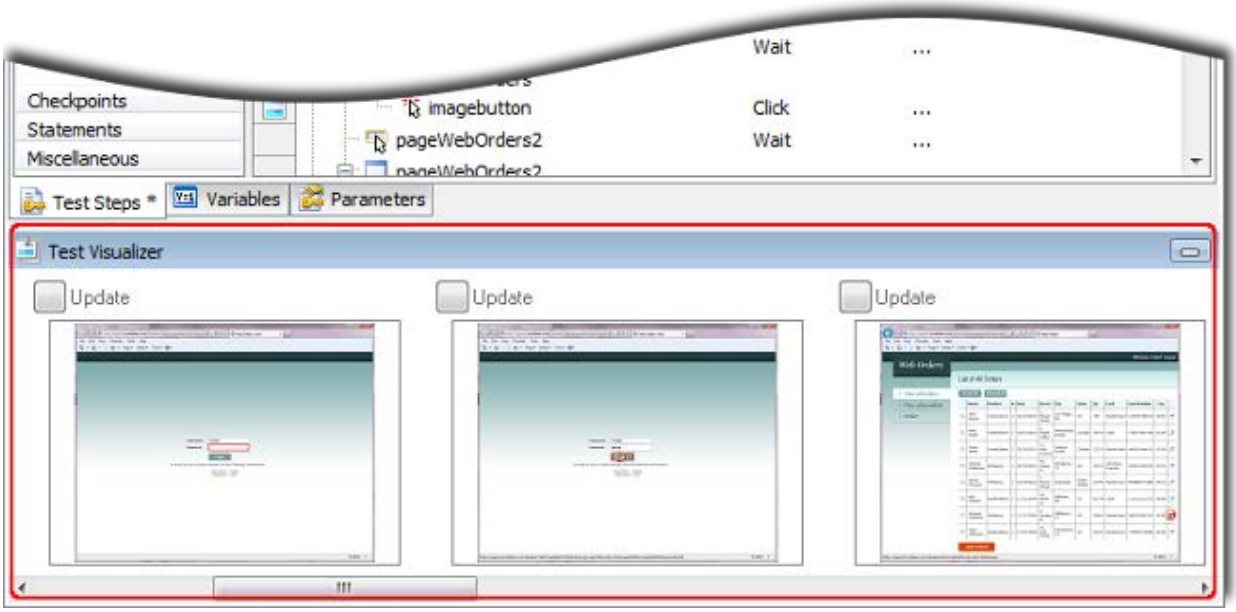
After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test's contents in the Keyword Test editor:



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 Web Orders application during the recording. We call the test commands **operations**.

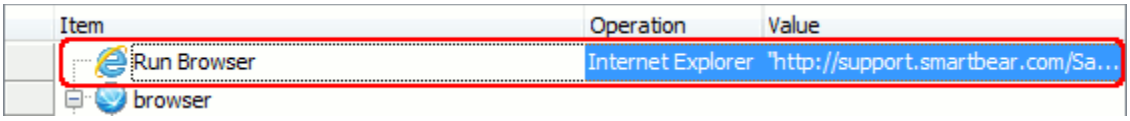
Below the commands there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:



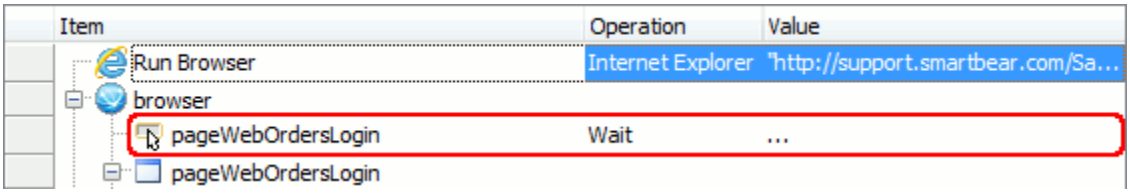
These images illustrate the recorded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (mouse clicks, typing text and so on).

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.

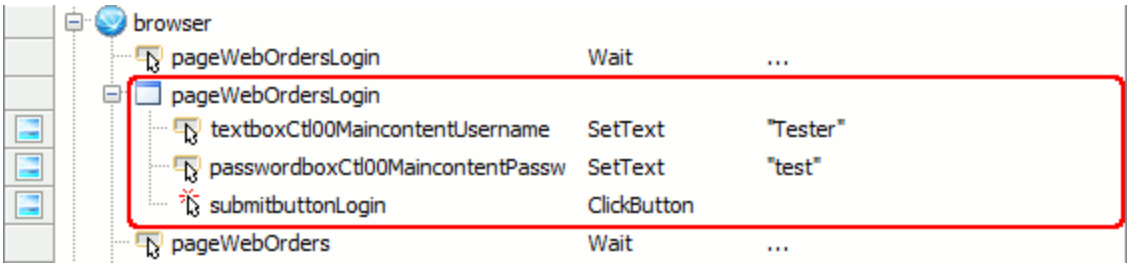
The first operation in the test is **Run Browser**. It launches the Internet browser and navigates to the specified page from a keyword test. TestComplete automatically records this operation when it detects a browser launch.



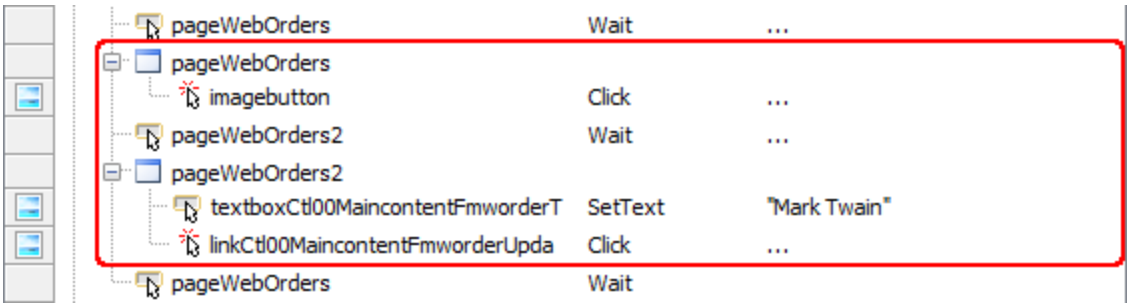
The second operation waits for the web page to load completely in the web browser. The highlighted operation waits for the Login page of the Web Orders application. The other operations of this type wait for other web pages.



The following operations work with the Login window.

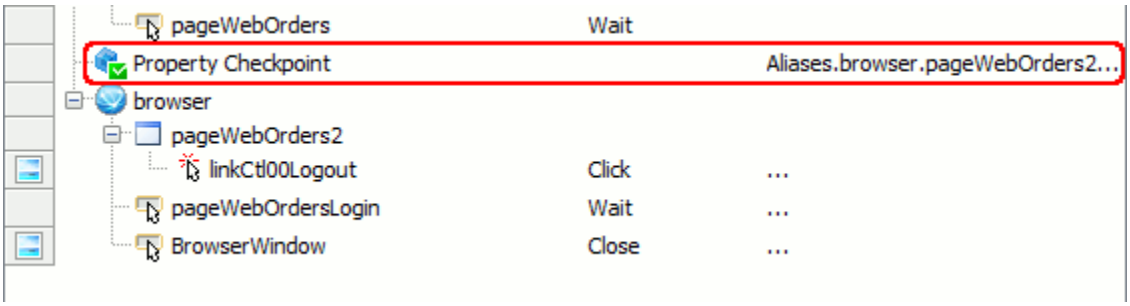


Then there are operations that simulate opening the Edit Order page and modifying the **Customer Name** field:

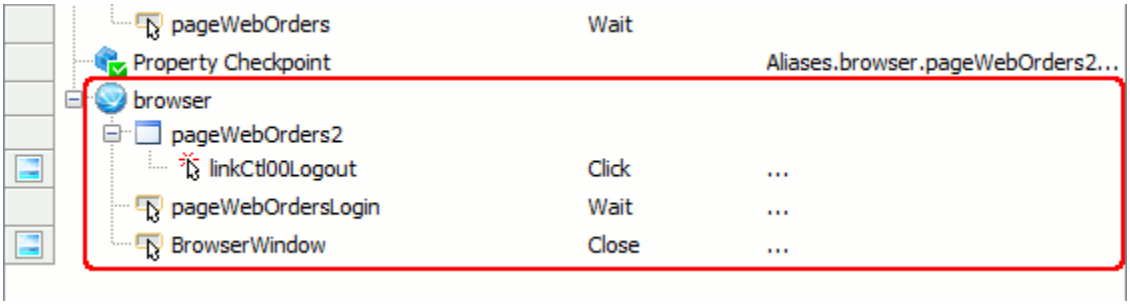


For more information on simulating mouse events, keyboard input and other actions from your scripts, see *Simulating User Actions* in TestComplete Help.

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

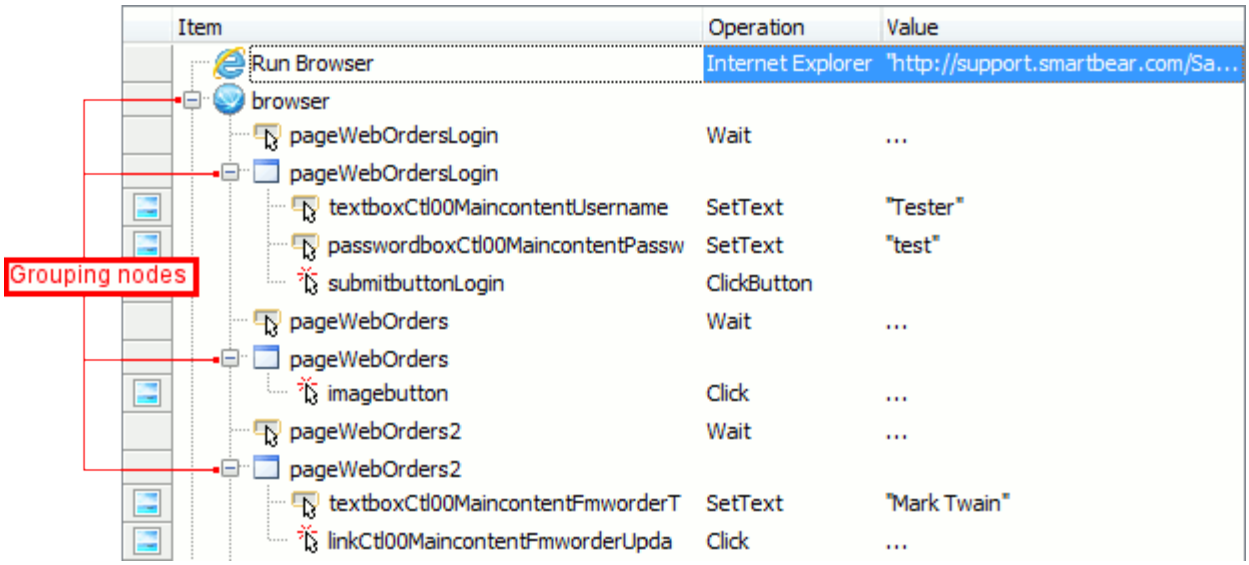


Finally, there are operations that log out from the Web Orders application and close the browser:



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've recorded user actions for one browser. So, we have a single group node for the browser. It contains all the actions that you simulated on the browser windows and controls. The actions that you performed on different web pages are organized into a number of “page” group nodes:



You may notice that the name of the Internet browser and the names of its pages and page elements differ from the names that you can see in the Object Browser panel. For instance, in the Object Browser, the name of the Internet browser is *Browser("iexplore")*, *Browser("firefox")*, or *Browser("chrome")* (depending on which browser you use), while in the test, the Internet browser is simply called a *browser*. Another example - the Login page in the Object Browser is called *Page("http://support.smartbear.com/Samples/TestComplete12/WebOrders/Login.aspx")*, while in the test, its name is much shorter: *pageWebOrdersLogin*.

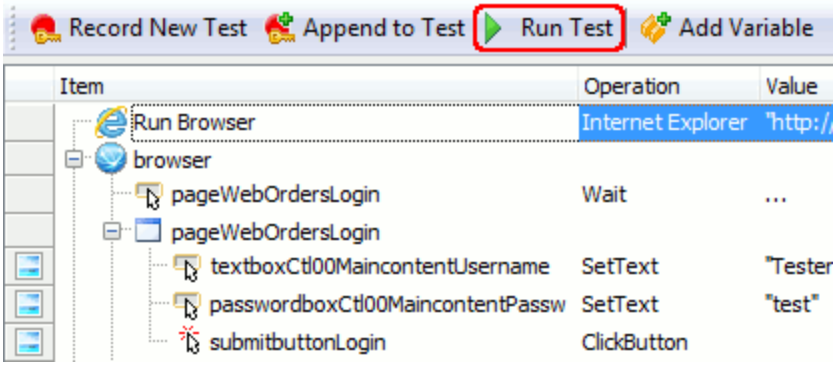
There is a logical reason for this: by default TestComplete automatically generates and uses custom names for the objects that you worked with during test recording. Generating and assigning custom names is called name mapping. TestComplete maps the names because the default names may be difficult to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

7. Running 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 an Internet browser be 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. Also do not forget to log out from the WebOrders application.

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



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

> **Note:** Don't 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:

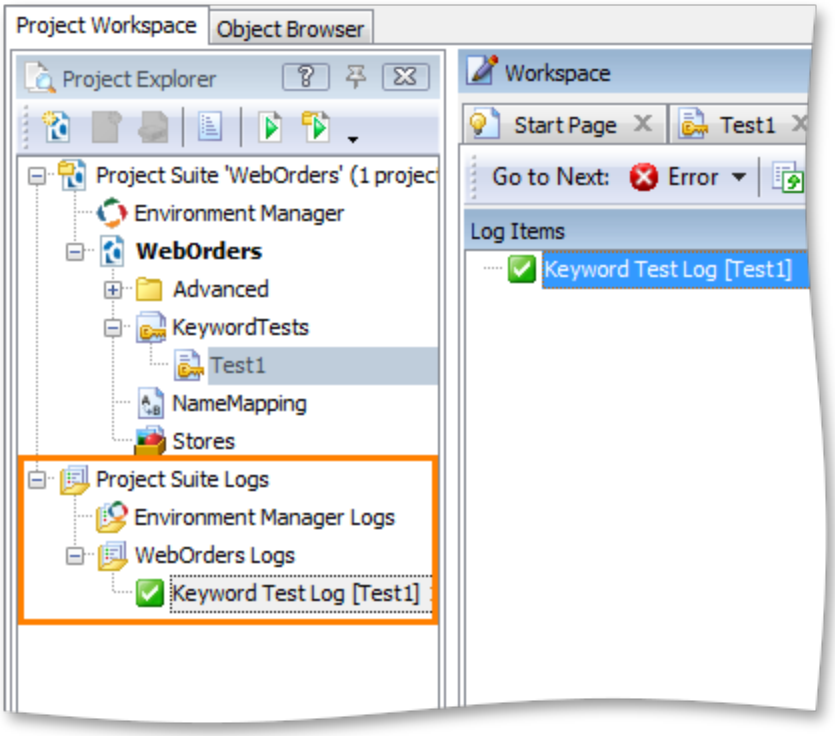
  The indicator displays messages informing you about the simulated test actions.
- TestComplete executes the test commands until the test ends. You can stop the execution at any time by pressing **Stop** on the Test Engine toolbar or in the indicator, or by selecting **Test | Stop** from TestComplete's 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's variables and objects using TestComplete's **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* in TestComplete Help.

8. Analyzing Test Results

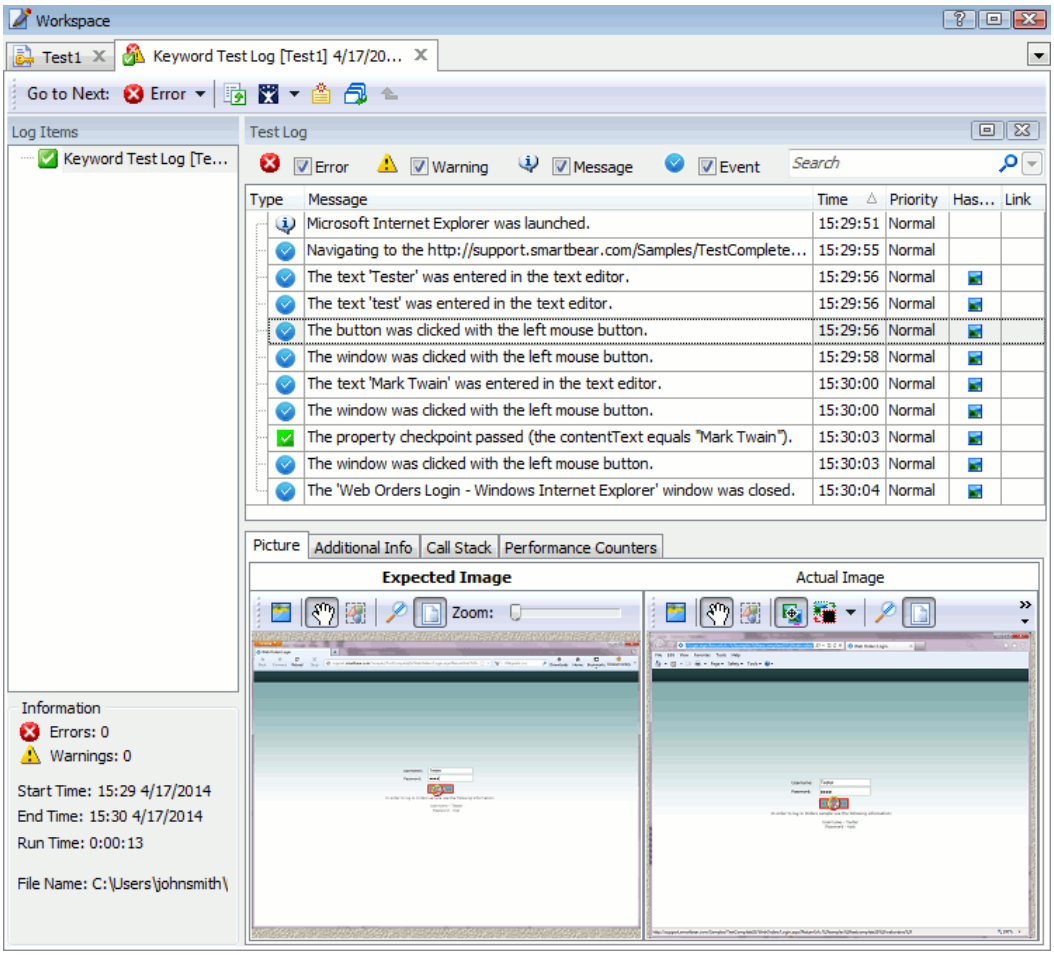
TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the Project Explorer panel under the **Project Suite Logs | WebOrders Log** node. This is the primary workspace for looking up the test history of the project and project suite. Each node 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.

In our example, the log is as follows –



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.

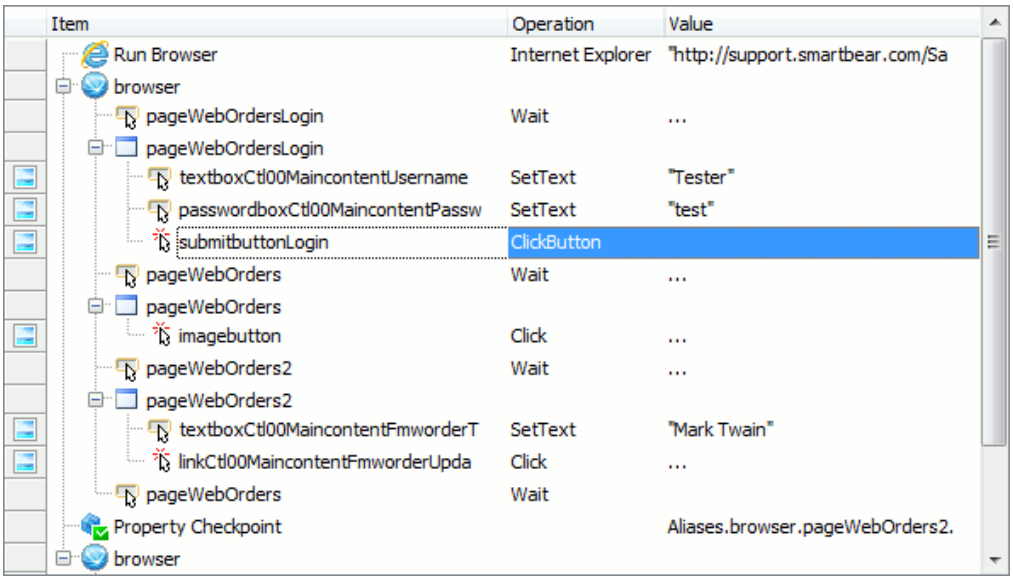
TestComplete may post additional text and images along with the message. To view them, simply select the desired message in the log and look in the **Additional Info** and **Picture** panes that are below the message list.

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.) The test log includes a special button that lets you compare the images and easily see the difference. This simplifies the search for errors that may occur in your test. For more information, see topics of the *Test Visualizer* section in TestConlete Help.

The log's **Call Stack** pane displays the hierarchy of test calls that led to posting the selected message to the log.

The log's **Performance Counters** pane displays values of the performance counters monitored during the test run. The values are shown in the form of graphs.

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:



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 instance, developers could change the application's behavior, the recognition attributes of windows and control 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 them, and so on.

One of the most typical reasons which novice users face is the difference in the application's 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 the test, it also must be running before you run the test; if the tested web page was opened on the second tab of your web browser when you recorded your test, it should also be opened on the second tab when you run the test, and so on.

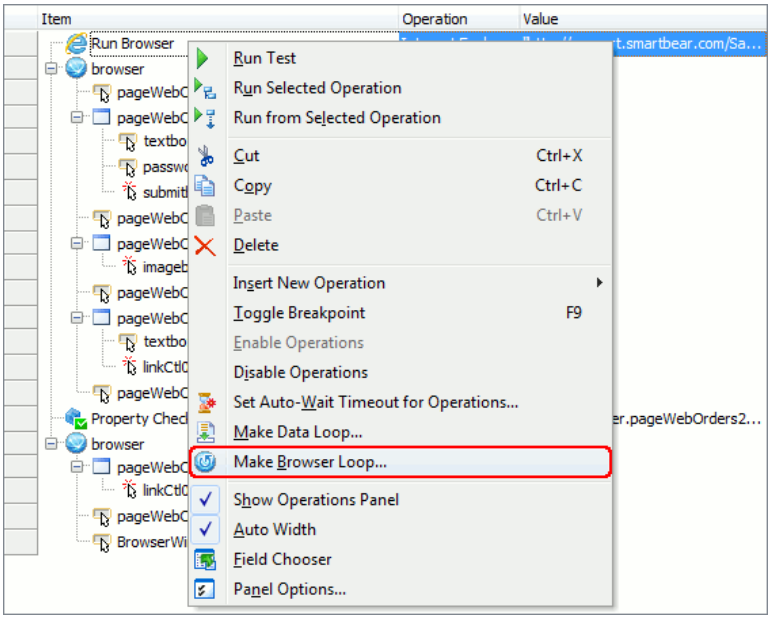
For information on searching for the cause of errors and resolving typical problems, see *Handling Playback Errors* in TestComplete Help.

9. Running the Test in Multiple Browsers

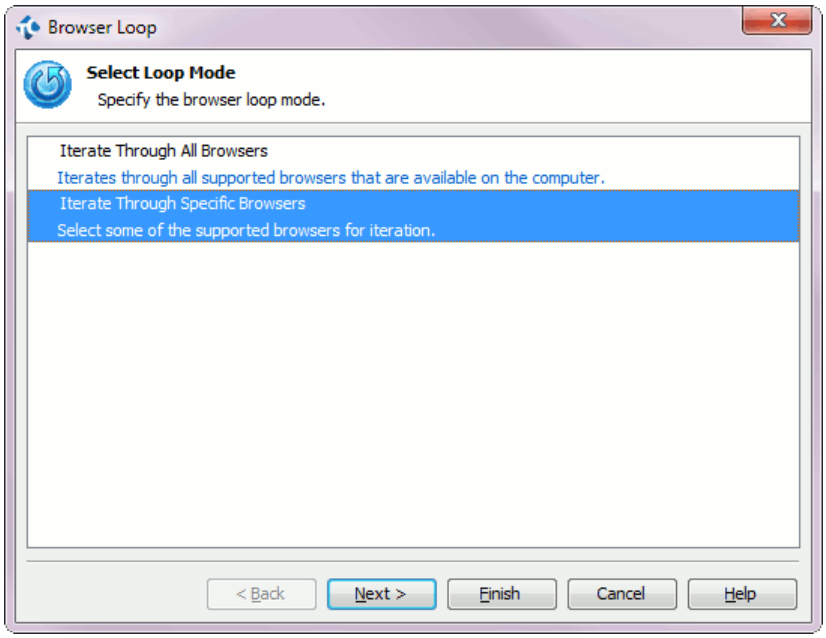
After you make sure that the test is executed successfully in the browser you used for recording, you can easily modify the test so that it can be executed in different browsers (the so called **cross-browser testing**). This helps you ensure that the web application works correctly in various browsers.

Now let's modify the test so that it runs in different browsers during the test run.

- Open the test in the Keyword Test editor.
- Right-click the Run Browser operation and select **Make Browser Loop** from the context menu.



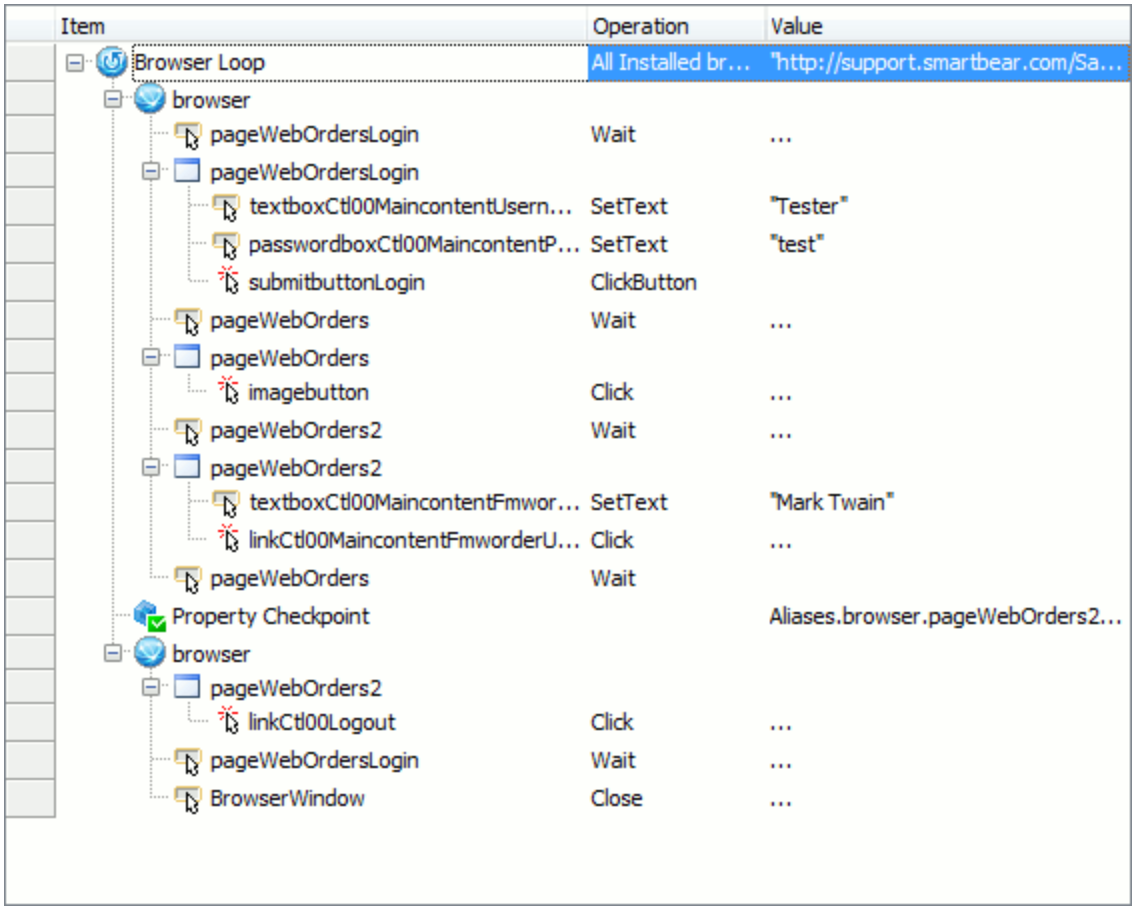
- In the ensuing operation parameters dialog, select **Iterate Through All Browsers** and press **Finish**.



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

- 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:



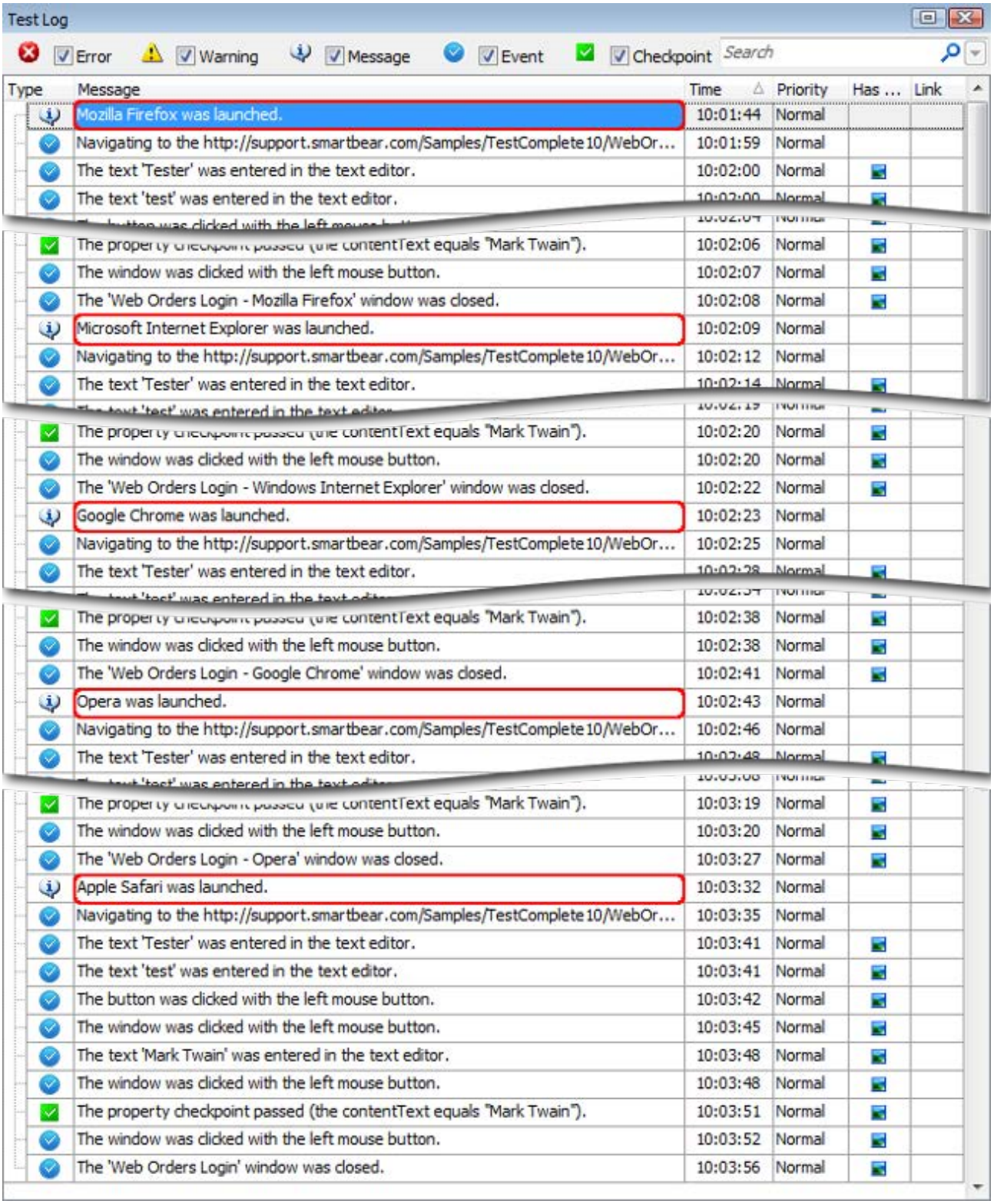
- Save the test by selecting **File | Save** from TestComplete's main menu.

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.



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. We are going to prepare the testing environment for mobile testing, connect to a mobile device and deploy the tested application to it, create and run a simple test and analyze the results. The test will emulate user actions over the mobile application and verify some data.

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.

The application should be deployed to the mobile device as described in the further steps of the tutorial.

1. Preliminary Steps

This section describes the preliminary actions you need to perform to prepare the environment for testing Android applications with TestComplete.

Note: With TestComplete you can test mobile applications that are running on physical devices, emulators and Android-x86 virtual machines. You cannot run tests on Android Wear devices.

In this section we will provide instructions on preparing for testing on a physical device. Preparatory actions for Android emulators and Android-x86 virtual machines are somewhat different (see *Preparing Devices, Emulators and Virtual Machines* section in TestComplete Help). With this exception, the rest of the testing actions are performed in the same manner.

Installing Required Software

Installing JDK and Android SDK

To test Android applications, you need to download and install Java Development Kit or Java Runtime Environment (JDK or JRE) and Android SDK on your test computer.

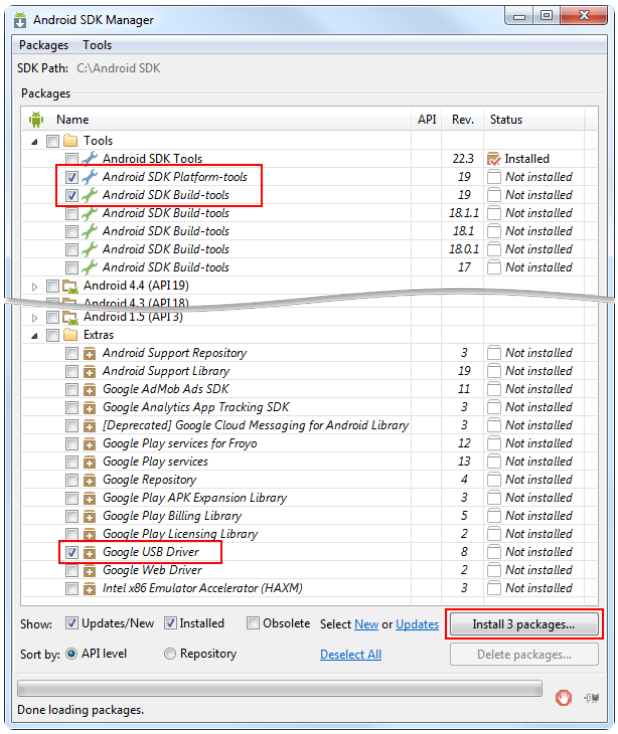
Note: If you already have the Android SDK and JDK (or JRE), go to step 5

1. Download the JDK from:
   - *http://www.oracle.com/technetwork/java/javase/downloads/index.html*
2. Run the downloaded installation and follow the instructions in the wizard.
3. Download the Android SDK from:
   - *https://developer.android.com/sdk/index.html*

   You do not need Android Studio (the default download), you need the SDK only. To get the SDK, scroll to the **Other Download Options** section, go to the **SDK Tools Only** subsection and download the file *installer_rXX-windows.exe*.

   Run the downloaded installation and follow the instructions in the wizard.
4. When Android SDK is installed, launch Android SDK Manager: Click the **Start** menu and select **All Programs | Android SDK Tools | SDK Manager**.

5. In the SDK Manager, select the following items:
   - Android SDK Platform-tools
   - Android SDK Build-tools
   - Google USB Driver



6. Click **Install packages**, accept the license agreements and proceed with the installation.
7. Close the SDK Manager.

Installing Device USB Drivers

As stated above, in this tutorial we will automate tests on the physical device. In order to interact with the device, you need to install the device USB driver on the TestComplete computer. For instructions and driver download links, see *OEM USB Drivers* in Android documentation:

http://developer.android.com/tools/extras/oem-usb.html

Installing and Configuring TestComplete Plugins

To be able to test Android applications, you need to have TestComplete's Mobile module installed. This module installs and enables the required plugins automatically.

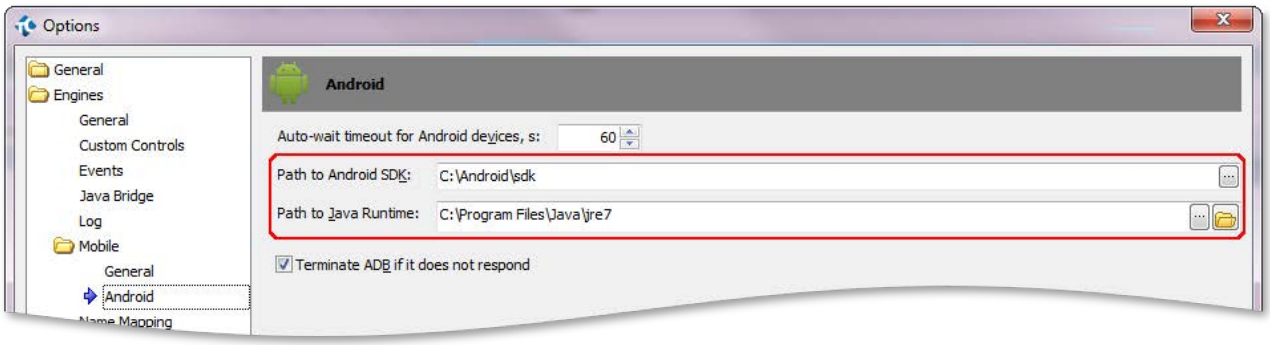
If the **Mobile** module is not available, you need to run the TestComplete installation in the Modify mode and select **Mobile** module for installation in the installation wizard. This would install and activate the required plugins.

Configuring TestComplete

Once the plugins are active, you may need to specify paths to JDK and Android SDK in TestComplete settings. The TestComplete installation wizard tries to find these paths automatically on your computer. If it is unable to do so, you may need to manually specify these paths in TestComplete.

To check whether the paths are specified correctly:

1. Launch TestComplete
2. Select **Tools | Options** from TestComplete's main menu. This will open the Options dialog.
3. On the left of the dialog, select the **Engines | Mobile | Android** category.
4. In the **Path to Android SDK** edit box, specify the folder where you installed the Android SDK on your computer.
5. In the **Path to Java Runtime** edit box, specify the folder where Java is installed on your computer.



6. Switch to the **Engines | Mobile | General** category.
7. Enable *the Automatically display Mobile Screen on OnScreen operation* option.
8. Enable *the Automatically display Mobile Screen on recording* option.
9. Click **OK** to close the dialog and save the settings.

Preparing Mobile Device

After all of the required software was installed on the desktop computer, you need to prepare the mobile device as well.

Enabling Developer Options on the Device

To perform testing on a physical Android device, you need to enable certain developer options on the device:

1. Make the developer options visible. Go to **Settings | About phone/tablet** and touch **Build number** 7 times.

   Note: The way in which you make the developer options visible may vary between devices and Android versions. Please refer to the documentation on your device for more information.

2. Go to **Settings | Developer options** and enable the following options:

- **USB debugging** - Enables communication between the device and Android Debug Bridge (ADB). In other words, it makes your device visible to ADB and TestComplete.
- **Stay awake** - Prevents the device from going into sleep mode during debugging.

3. Save the changes.

Connecting the Device

Turn on your device and connect it to your computer with the USB cable. Android Debug Bridge will detect the connection and automatically connect to the device.

Checking Whether Device Is Connected

To verify whether the mobile device has connected successfully:

- Click the **Show Mobile Screen** button on the TestComplete toolbar.

The **Mobile Screen** window will open if at least one mobile device or virtual machine is connected to Android Debug Bridge.

If you have several devices or virtual machines connected, a click on the Show Mobile Screen item will open the **Select Current Device** dialog that lists the connected devices. Check whether your device or virtual machine is on the list, select it there and click OK to view its screen.

Installing TestComplete Android Agent

TestComplete Android Agent is a helper application used to exchange data with Android applications prepared for testing with TestComplete. The Agent should be installed on the mobile device.

To install the Agent:

- Open the **Mobile Screen** window (if you have not opened it yet)
- Press the  **Install Android Agent** button on the window toolbar.

Preparing Application for Testing

In order for TestComplete to recognize objects of an Android application, that is, to make it "open" to TestComplete, you need to prepare the application in a special way.

To learn how to prepare applications manually or automatically, see *Instrumenting Android Applications* in TestComplete Help.

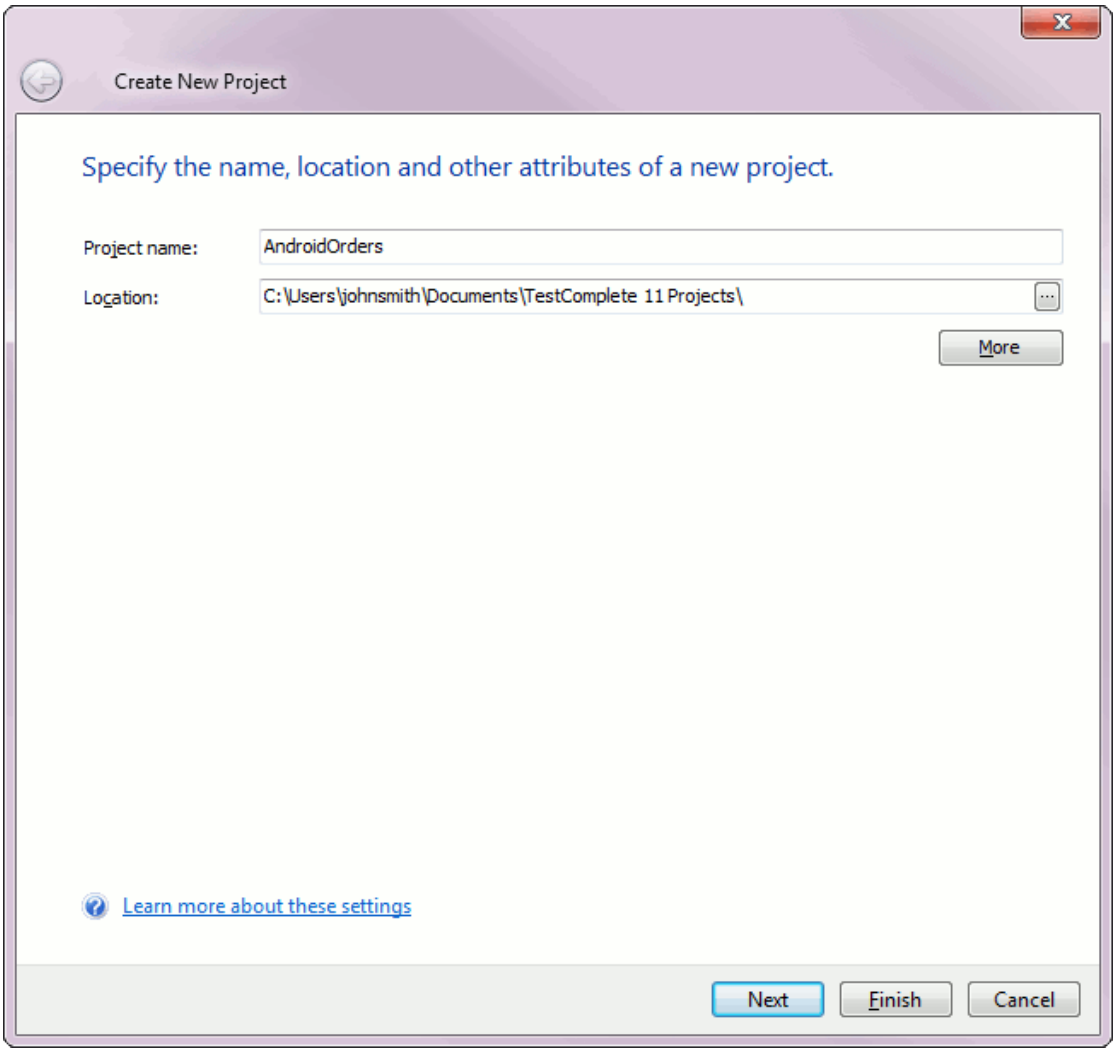
The Orders application which will be used in this tutorial is already prepared for white-box testing. Therefore the preparation actions will not be needed this time.

Now TestComplete, the mobile device and the application are prepared for testing. In the following sections, we will describe how to create a test project, create a simple test and play it back.

2. Creating a Test Project

Let's create a new test project:

1. If you have a project or project suite opened in TestComplete, close it. To do this, choose **File | Close** from TestComplete's main menu.
2. Select **File | New | New Project** from TestComplete's main menu. This will call up the **Create New Project** wizard.
3. In the wizard, enter the project name and the location where you want to store the project. In this tutorial, we will name the project *AndroidOrders* and use the default location.



4. On the next page, select *Android applications*.
5. On the next page, we can add our tested Android application to the project. To do this:
   - Click the **Add**. The **Add Android Tested Application** wizard will appear.
   - In the wizard, click the button near the **Android application package file** edit box and browse for the Android Orders package file. Its path is as follows:

- On Windows Vista, Windows 7 or later Windows versions:

  C:\Users\Public\Public Documents\TestComplete 12 Samples\Mobile\Android\Orders\Orders Application\bin\Orders.apk

- On Windows XP or Windows Server 2003:

  C:\Documents and Settings\All Users\Shared Documents\TestComplete 12 Samples\Mobile\Android\Orders\Orders Application\bin\Orders.apk

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

- Select the *Deploy to the device on start* check box to install the application on the device every time you launch the application from TestComplete.
- Clear the *Keep the data and cache directories on removal* check box. This will guarantee the same initial state of the tested application.
- Click **Next** and then **Finish** to close the wizard.
- Make sure that the *Autorun* check box on the list is selected. If it is selected, TestComplete will automatically launch the Orders tested application on the mobile device when you start recording tests. If the check box is clear, then, to record user actions over your application, you will have to launch the application manually.

Click **Next** to continue.

6. On the next page, you can enable or disable *Test Visualizer* for test recording and playback. To do this, check or uncheck the corresponding check box. Click **Next**.
7. On the last page, choose the preferable scripting language and click **Finish**.

TestComplete will create a new project and show its files in the Project Explorer panel.

3. Creating a Test

Creating Tests in TestComplete

TestComplete allows you to create tests in two ways. You can:

- Create tests manually
- Record tests

When you create a test manually, you enter all the needed commands and actions that your test must perform via appropriate script objects or keyword test commands. This approach is very 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 different problems. For example, while creating a test manually you must know the classes and names of your application's objects you want to work with. To solve such problems, TestComplete includes a special feature that lets you easily create tests. 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. We call this feature "**recording a test**", because you create a test visually and in one sense you record the performed actions to a script or keyword test. It is a very useful approach and it does not require much experience in creating tests. So, in this tutorial we will demonstrate how to record tests with TestComplete.

Recording Tests in TestComplete

The recording includes three steps:

1. You start recording by selecting **Test | Record | Record Keyword Test** or **Test | Record | Record Script** from TestComplete's main menu or from the Test Engine toolbar. You can also start recording by clicking **Record a New Test** on the Start Page.

   After you command TestComplete to start the recording, it will switch to the recording mode. In this mode, TestComplete minimizes its main window, displays the Recording toolbar and, optionally, displays the Mobile Screen window.

   The Recording toolbar contains items that let you perform additional actions during the recording, pause or stop recording and change the type of the recorded test (from the keyword test to script code, or vice versa).

   The Mobile Screen window displays the screen of the connected mobile device. This window is used to record tests against mobile applications. The window is displayed if the *Automatically display Mobile Screen on recording* option is enabled.

4. After starting the recording, perform the desired test actions: launch the tested application (if needed), work with the application as you normally do: select menus, touch buttons and other controls and so on.

5. 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 the Help.

Planning a Test for the Android Orders Application

The sample Android Orders application works with a list of orders. Suppose you need to test whether the application's Edit Order page functions properly and modifies data in the order list. In this case --

- **Test purpose:** The test should check whether the Edit Order page 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 data in the order list. 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 the expected value. We will add a special comparison command to the test for this. This command will post comparison results to the test log, so we will see whether verification failed or passed successfully.

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

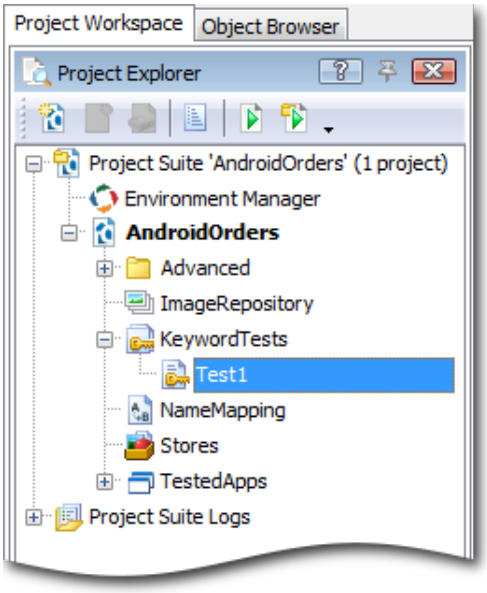
Recording a Test for the Android Orders Application

Now let's record a keyword test for the sample Android Orders application.

1. Open the Mobile Screen window. TestComplete records only those actions over mobile applications that you perform in this window and ignores the actions that you perform on the device, emulator or virtual machine.
   - Click the **Show Mobile Screen** button on the Test Engine toolbar.
2. Open the keyword test in the editor.

   When creating a new project, TestComplete automatically creates an empty keyword test in this project. We will add test commands into this test.

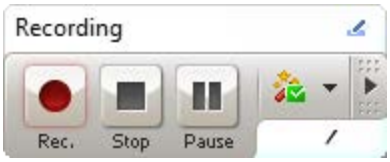
   To open the keyword test:
   - Switch to the **Project Explorer** panel.
   - Expand the **KeywordTests** node.
   - Double-click the **Test1** node.



3. To start the recording, select the **Append to Test** item on the test editor's toolbar.

TestComplete will display the Recording toolbar on screen. If the Interactive Help panel is visible, TestComplete will also show information about the recording in it.

By default, the Recording toolbar is collapsed:



Click the ▶ arrow button to expand the Recording toolbar and view all its buttons:

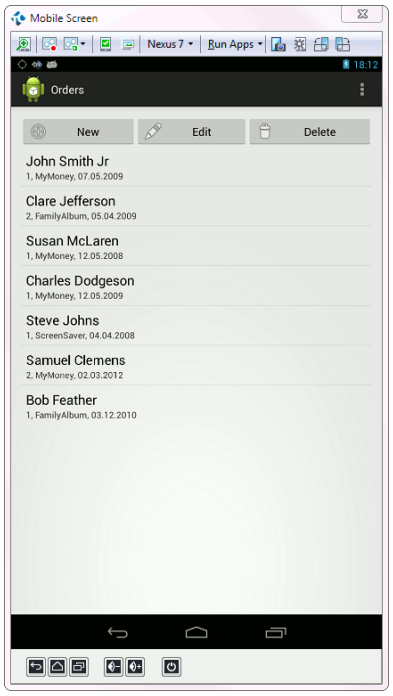
4. After you start the recording, TestComplete will automatically deploy the Orders application to the mobile device and start this application. This happens, because we have enabled the application's *Deploy to the device on start* and *Autorun* settings while creating the test project on the previous step.

If we had disabled the *Autorun* property, we would have had to launch the application manually. You can do this by selecting the **Run Tested Application** command from the Recording toolbar:

You can also launch the application from the **Run Apps** drop-down menu on the Mobile Screen window's toolbar.

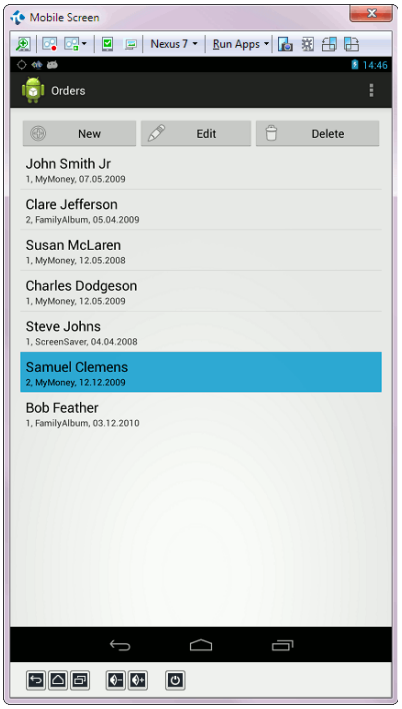
TestComplete records the application start using a special application launch test command. You will see this command later, when we will analyze the recorded test.

5. Wait until the Android Orders application installs and starts on the mobile device. The Mobile Screen window will display the initial window of the application:

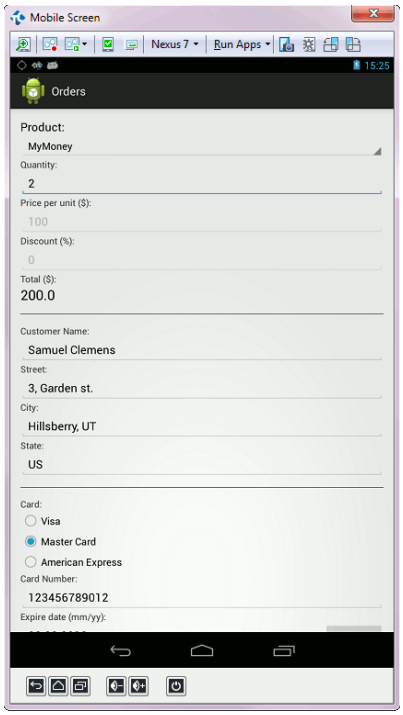


If the Interactive Help panel is visible, resize or move it so that it does not overlap the Mobile Screen window. Your actions on this panel are not recorded.

6. In the Mobile Screen window, click Samuel Clemens's order.



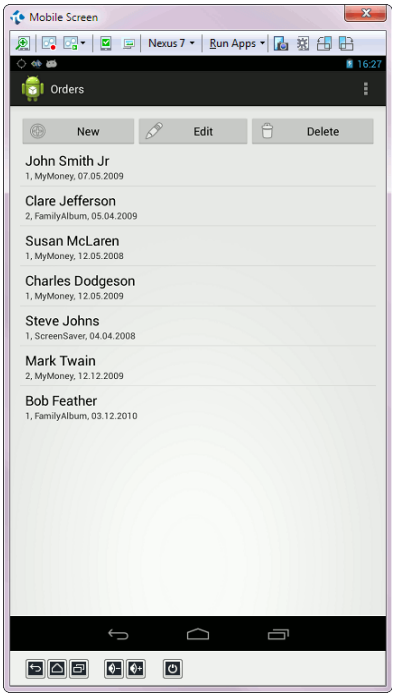
7. Click the Edit button of the Orders application. This will invoke the Edit Order screen:



8. Let's change the customer name in the order details.

   Clear the *Samuel Clemens* text and type *Mark Twain* instead. Use your desktop keyboard to enter text in the Mobile Screen window.

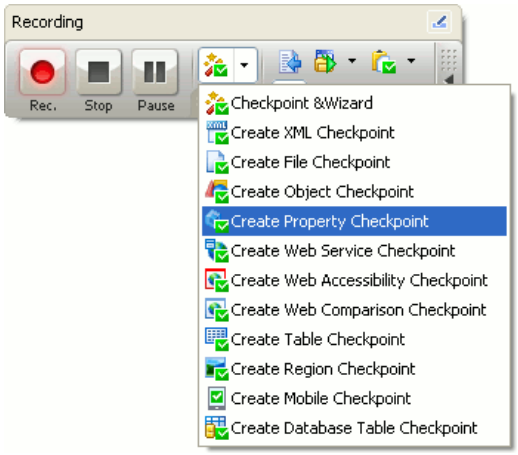
9. Scroll down the Edit Order Screen till the Ok button becomes visible and click this button. This will save the order changes and return back to the orders list.



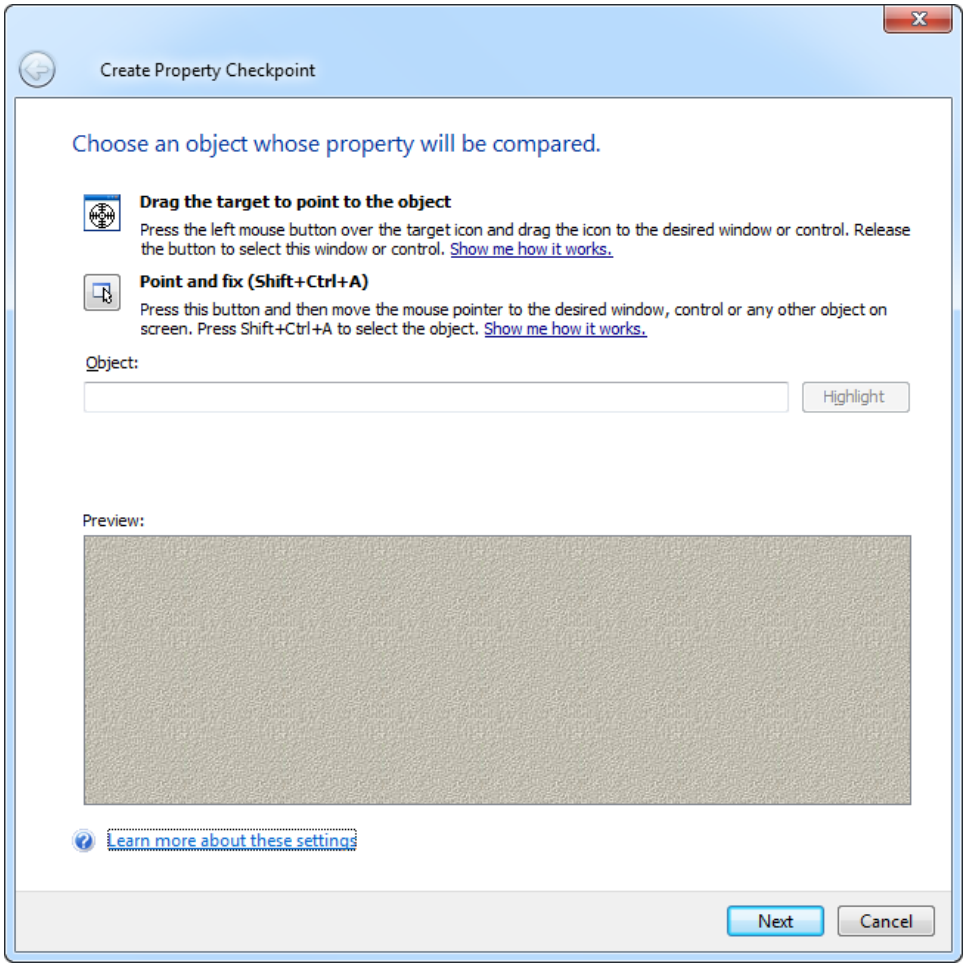
10. Now let's insert a comparison command into our test. It will verify that the application's customer list displays the modified name - *Mark Twain*.

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 the **Property checkpoint**. It is used to check data of application controls. We will use this checkpoint in our tutorial.

- Select **Create Property Checkpoint** from the **Checkpoint** drop-down list of the Recording toolbar:



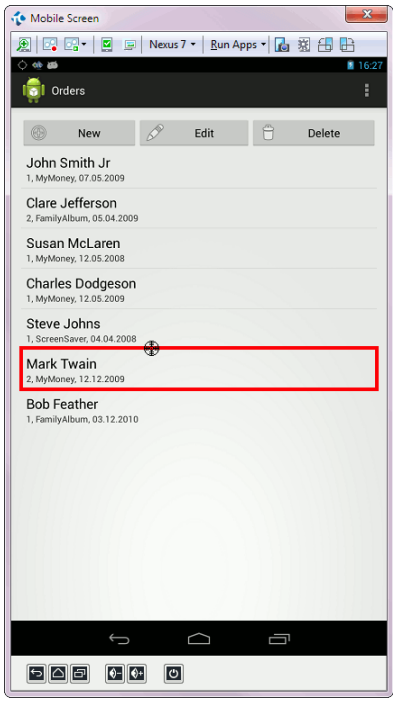
This will invoke the **Property Checkpoint** wizard. It will guide you through the process of checkpoint creation:



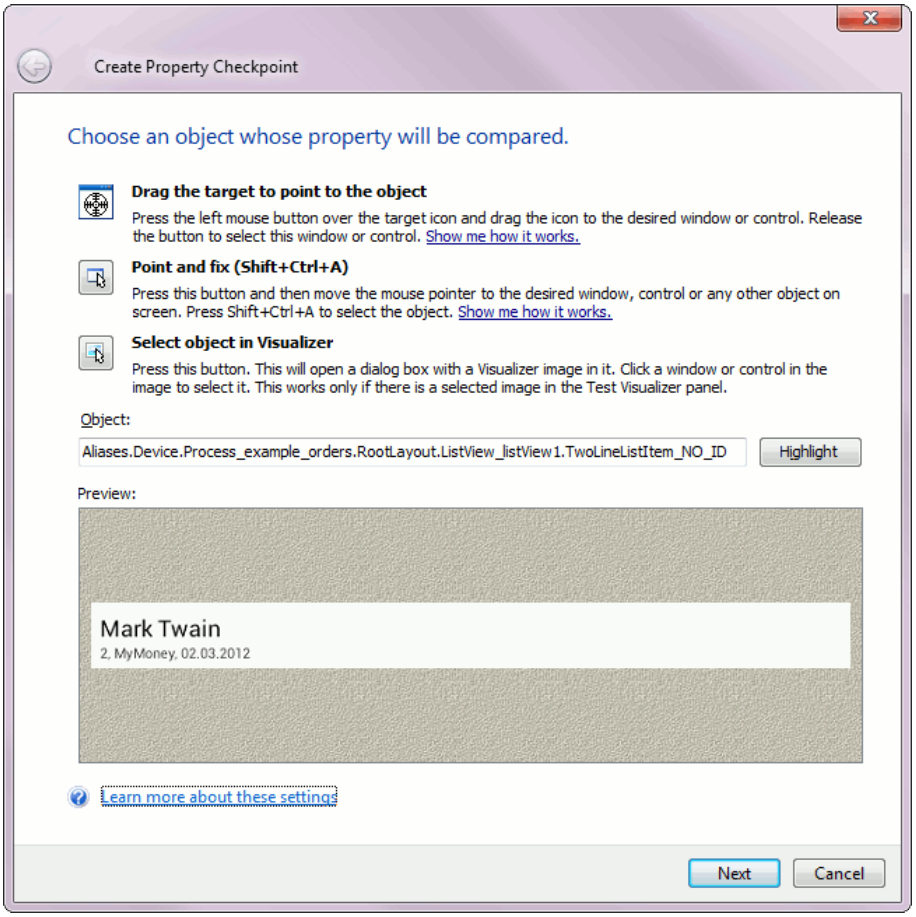
- On the first page of the wizard, click the target glyph () with the left mouse button and keep the button pressed.

  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 the red frame.

  Release the mouse button when the target glyph is over the order made by Mark Twain and when the entire item is highlighted with the red frame:



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

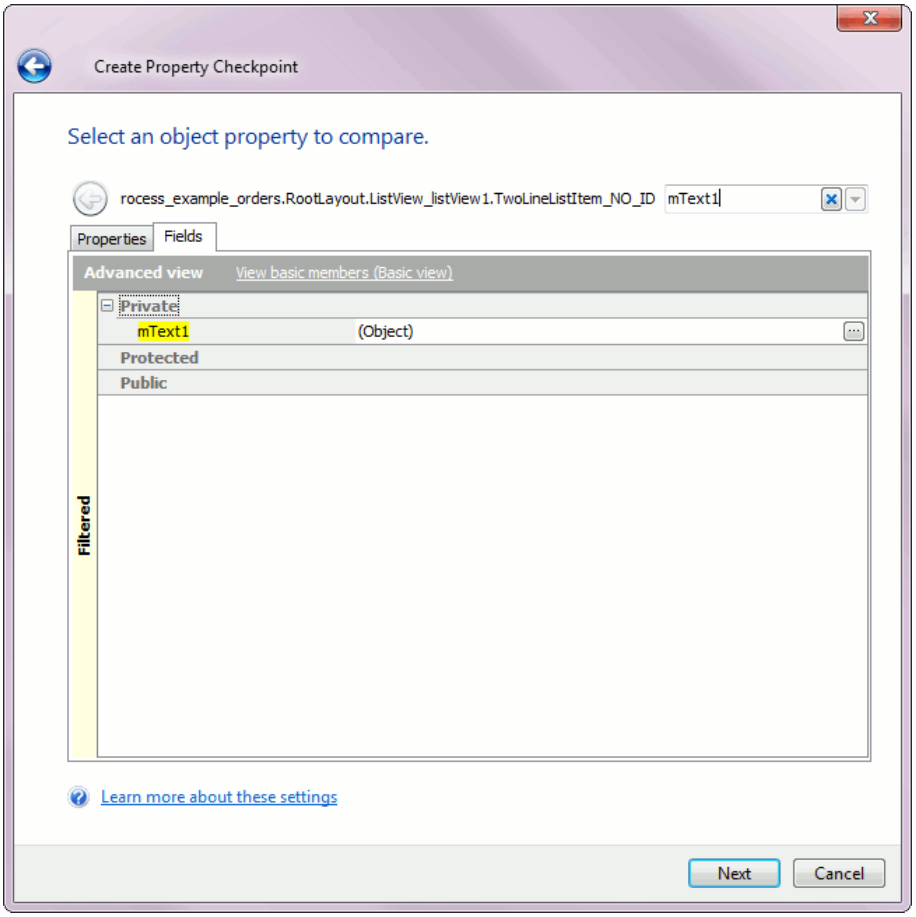


Click **Next** to continue.

- The next page of the wizard displays a list of the selected object's properties and fields. This list includes properties provided by TestComplete, as well as properties and fields defined by the tested application.

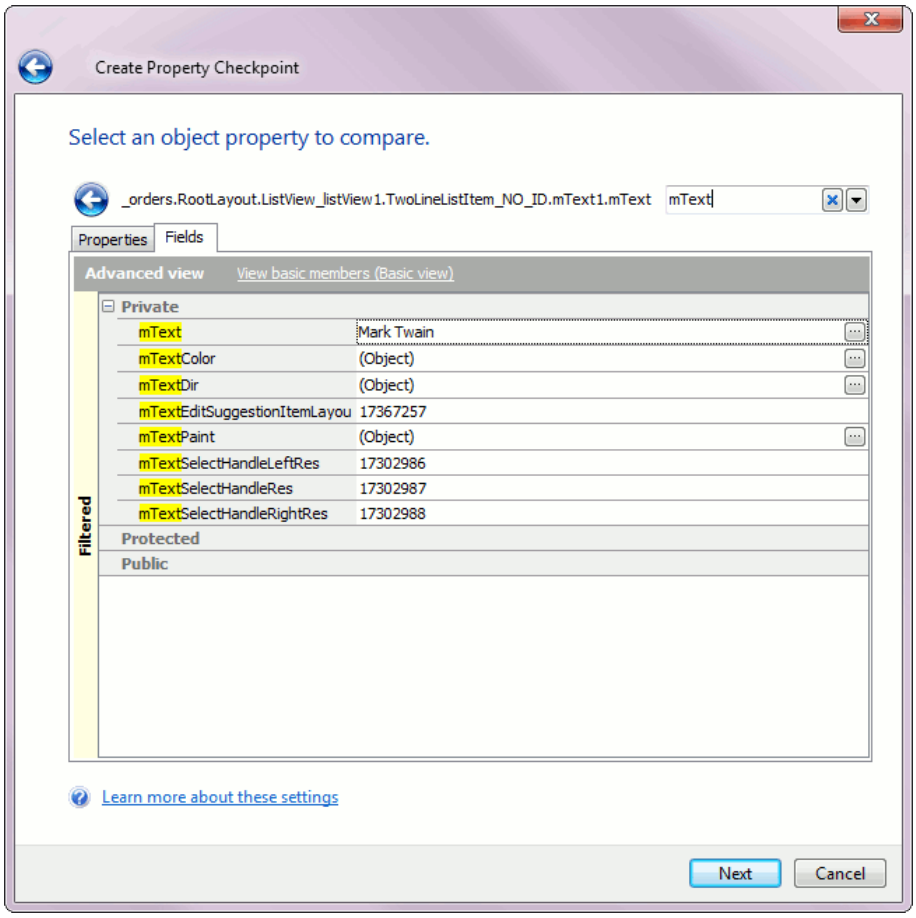
  To verify the data, we will use several native fields defined by the application. To view the object's fields, click the **View more members (Advanced View)** link and then switch to the **Fields** tab.

- Type `mText1` in the **Search** box. TestComplete will filter out the members according to the text you type.



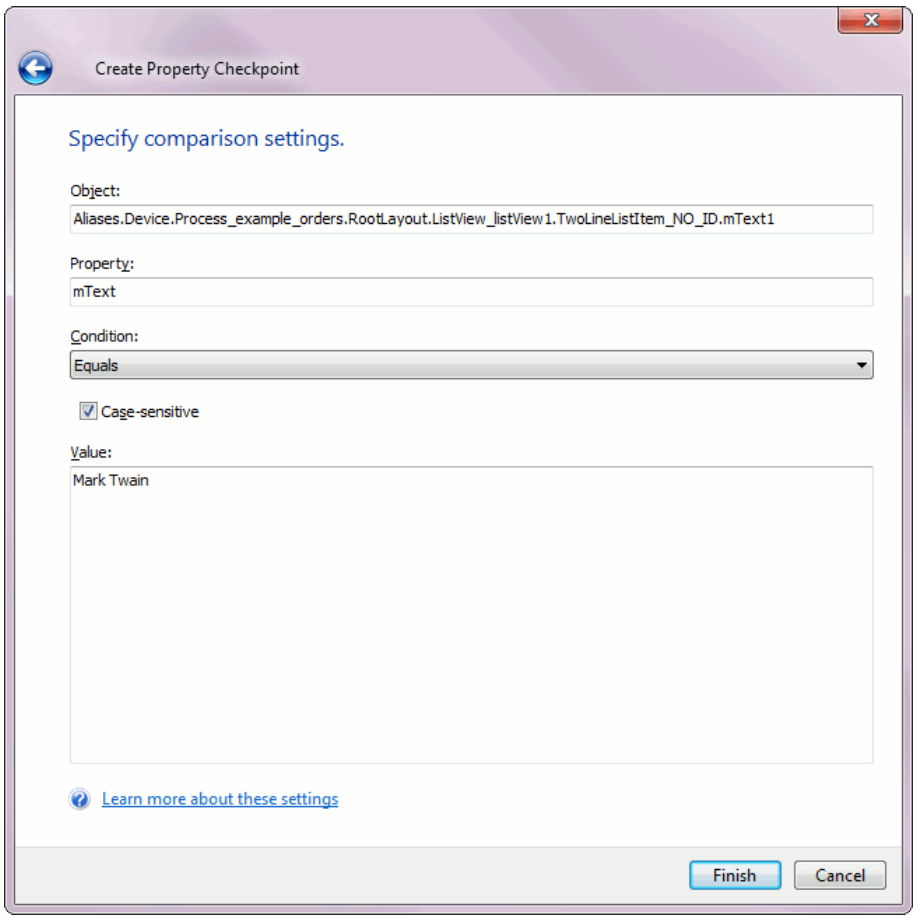
- Select the `mText1` field. This field refers to an object. To view the members of this object, press the ellipsis button.

- Switch to the Fields tab and type `mText` in the Search box.



- Find the `mText` property in the list. Select this property and then click **Next** to continue.

- On the next page of the wizard you can see the name of the property whose value will be verified, the comparison condition and baseline data in the **Value** box:

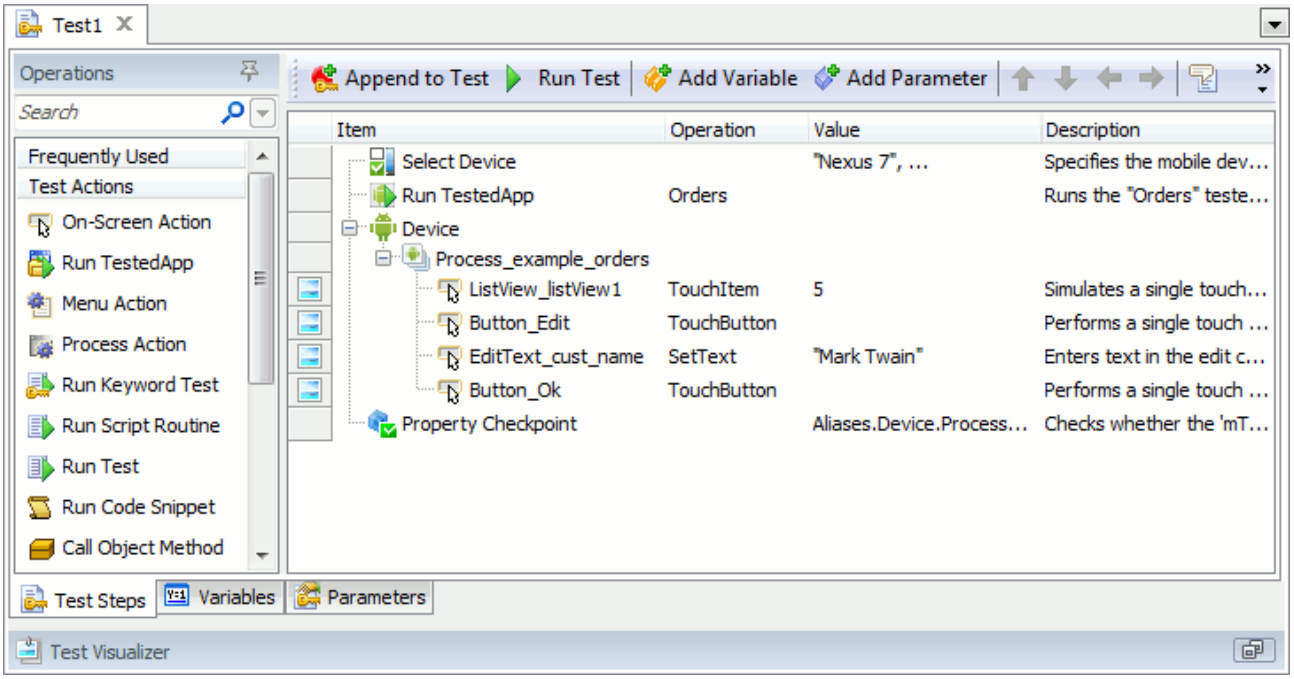


Click **Finish** to complete the checkpoint creation.

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

4. Analyzing the Recorded Test

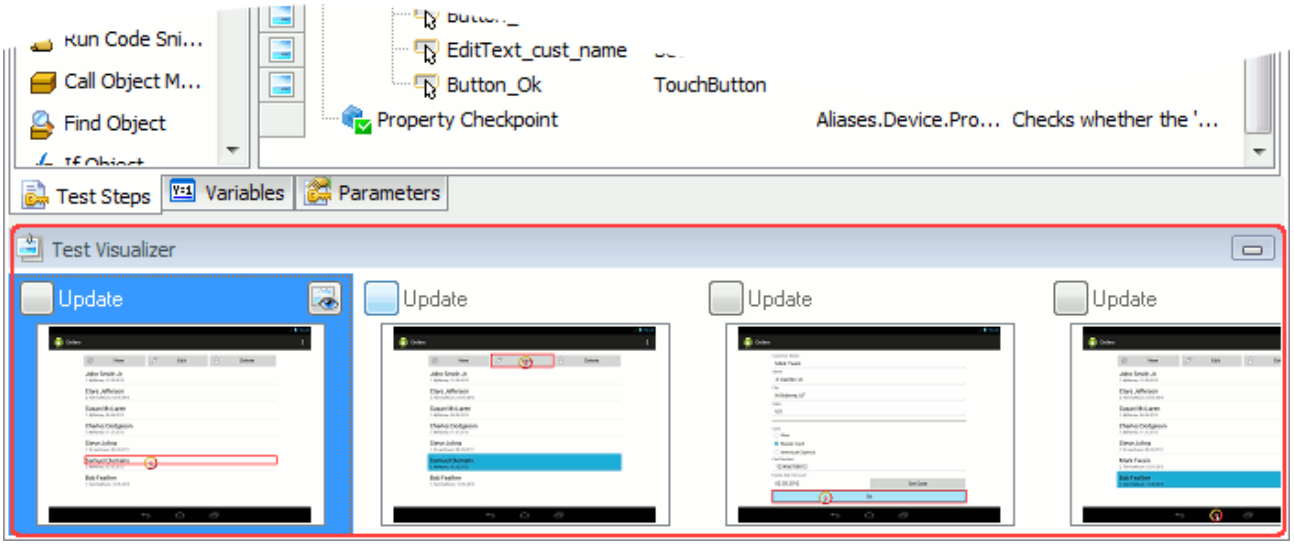
After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test's contents in the Keyword Test editor:



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 the recording. We call the test commands **operations**.

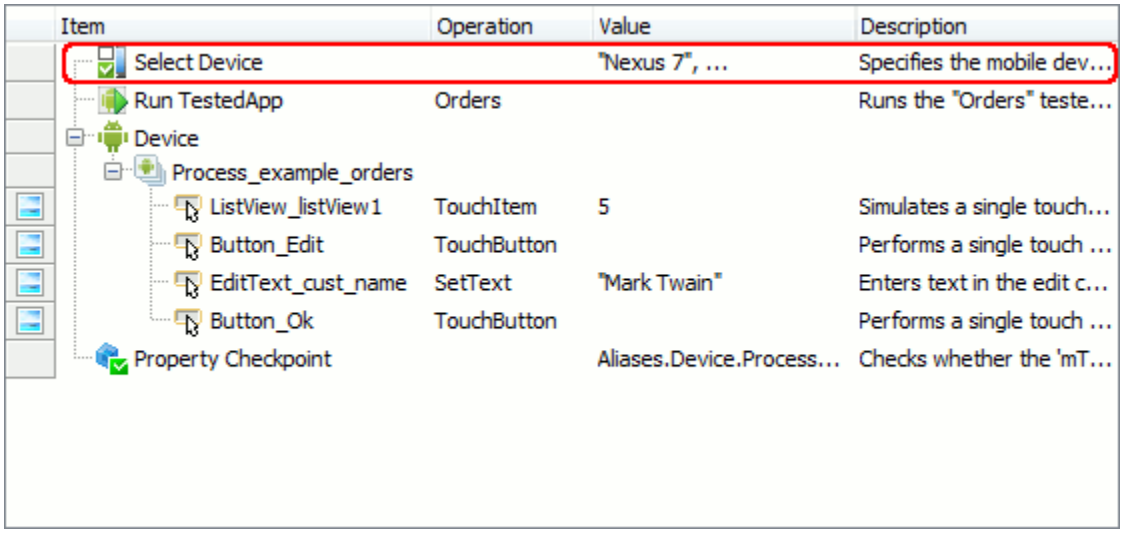
Below the commands there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:



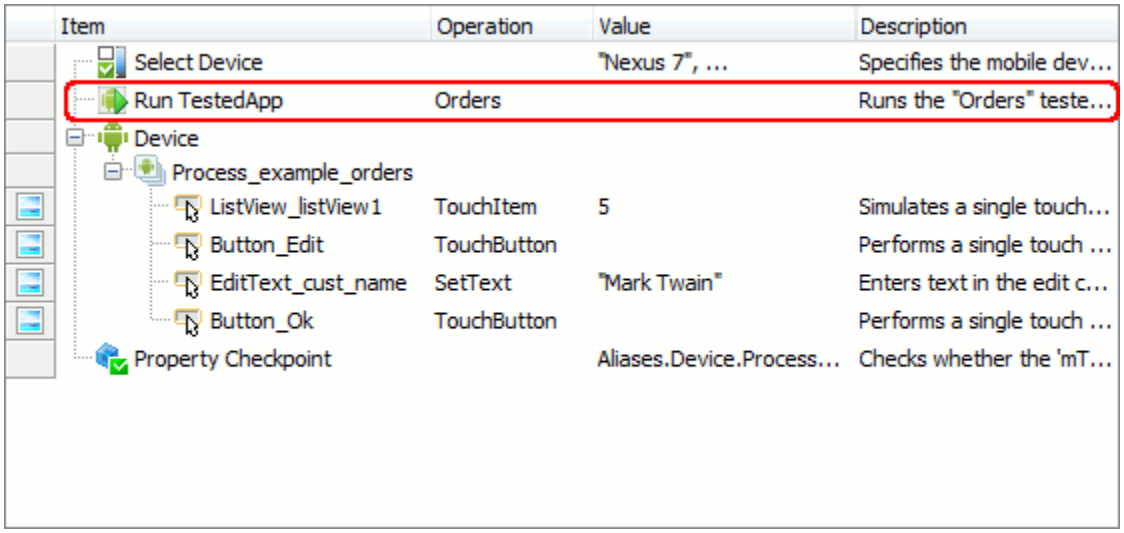
These images illustrate the recorded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (touching, dragging, typing text and so on).

When you choose an operation in the editor, Test Visualizer automatically highlights 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 in TestComplete Help.

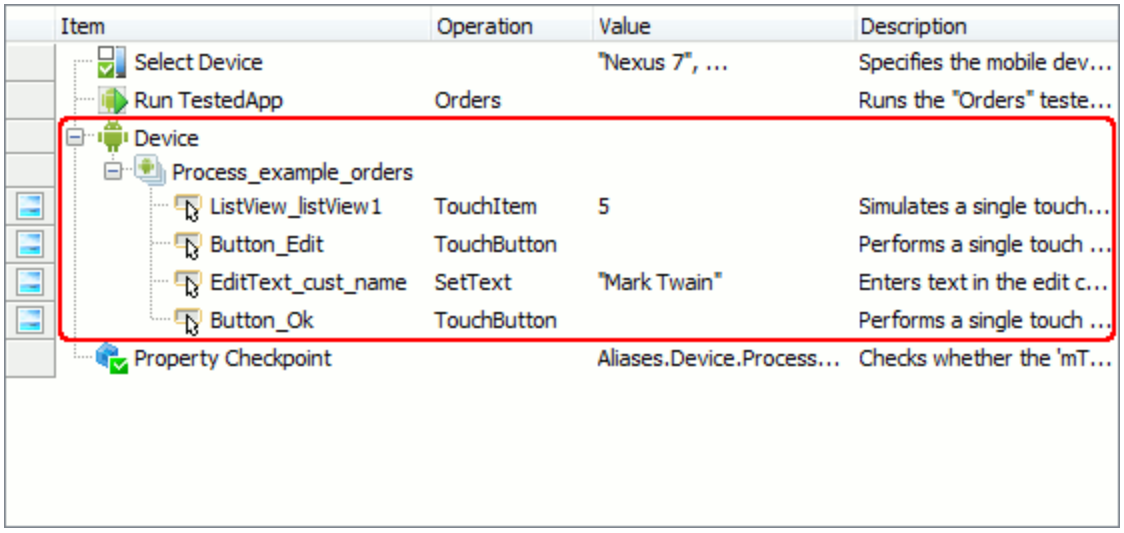
The first operation in the test is **Select Device**. It specifies a mobile device the test should work with. All further test operations will refer to this device.



The second operation 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.

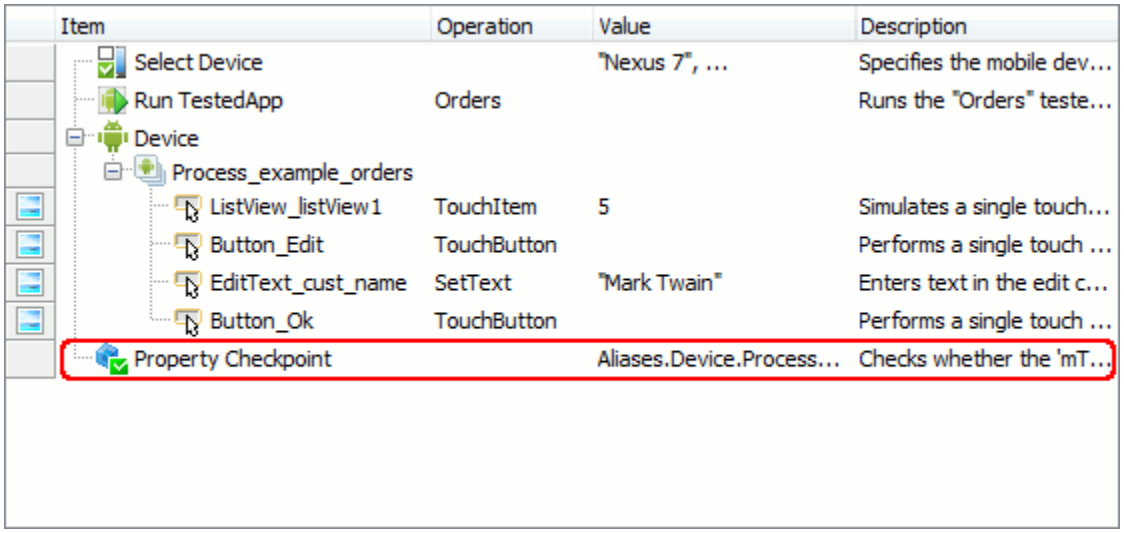


After that, there are operations that simulate your actions with the application. These operations select an item in the orders list, press the Edit button, change the value of the text field and press the Ok button.



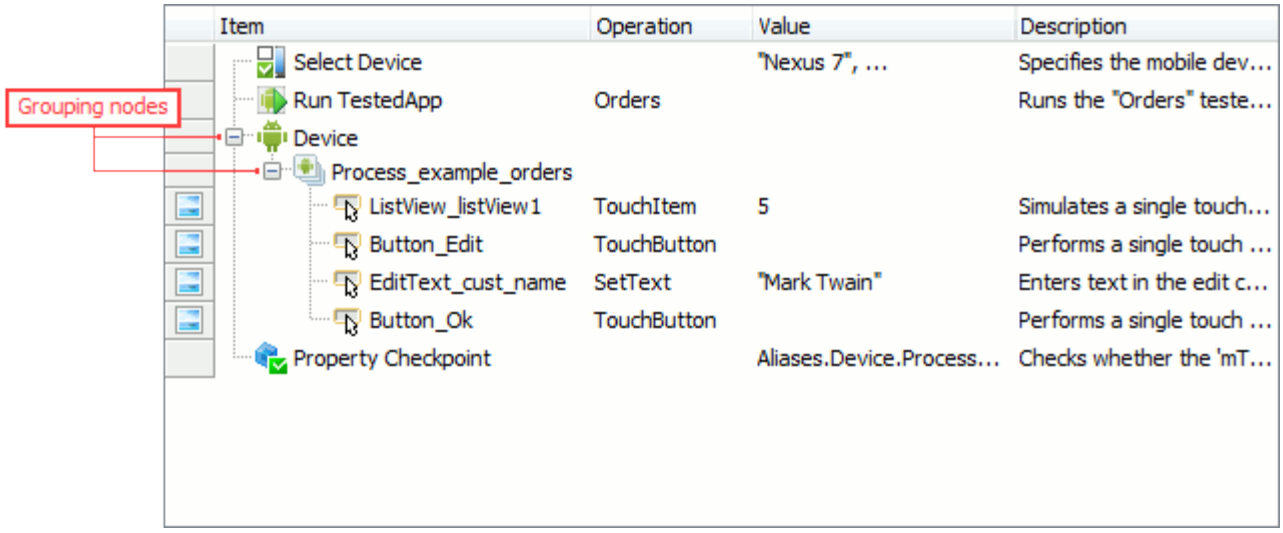
For more information on simulating touch events, text input and other user actions from your tests, see *Simulating User Actions Over Android Devices* in TestComplete Help.

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



As you can see, TestComplete automatically organizes the operations into groups that correspond to mobile devices and processes 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 mobile device and one process. So, we have two group nodes. The “device” node groups processes that were launched on the same device. The “process” node contains all of the actions that you simulated on the process windows and controls.



You may notice that the names of the tested process and its windows and controls differ from the names that you can see in the Object Browser panel. For instance, in the Object Browser, the tested process was named *Process("smartbear.example.orders")* while in the test it is called *Process_example_orders*; the Edit button was called *Button("editButton")* while in the test it is called *Button_Edit*, and so on.

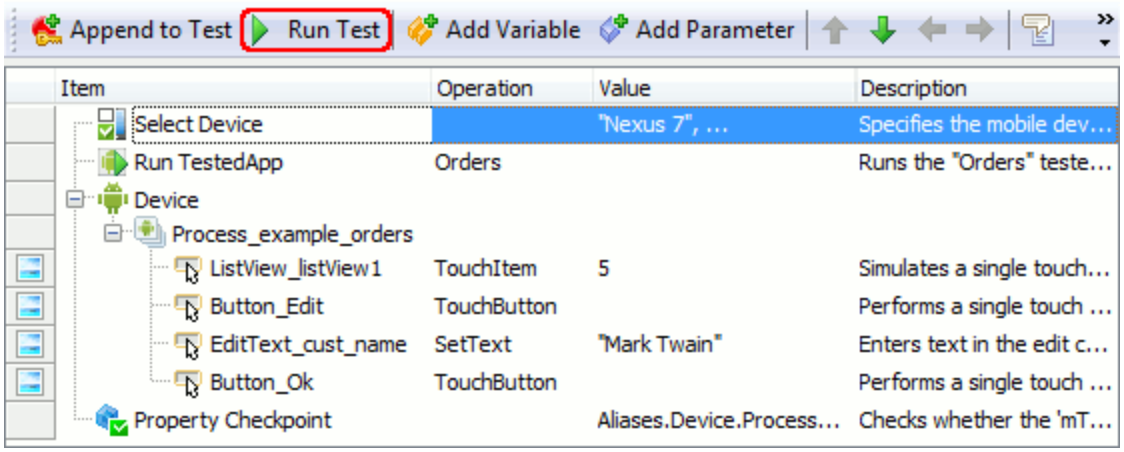
There is a logical reason for this: by default, TestComplete automatically generates and uses custom names for the objects you worked with during the test recording. Generating and assigning custom names is called name mapping. TestComplete maps the names because the default names may be difficult to 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. For more information on mapping names, see *Name Mapping* in TestComplete Help.

5. Running the Test

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

Before running a test, make sure it starts with the same initial conditions that existed when the test was created. For instance, the test designed for mobile applications must select which of the mobile devices it will currently work with. For this purpose we placed the **Select Device** keyword operation in the beginning of the test. Besides that, the test almost always requires that the tested mobile application be deployed onto the device and launched on it. In our case, to launch and deploy our tested application, we used the **Run TestedApp** keyword operation and enabled the *Deploy to the device on start* parameter of the tested application.

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



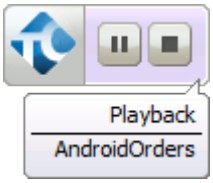
TestComplete will start executing test commands. In our case, the test will open the order and change the customer name from *Samuel Clemens* to *Mark Twain* and then verify whether the name has changed.

> **Note:** Don't 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:

  The indicator displays messages informing you about simulated test actions.

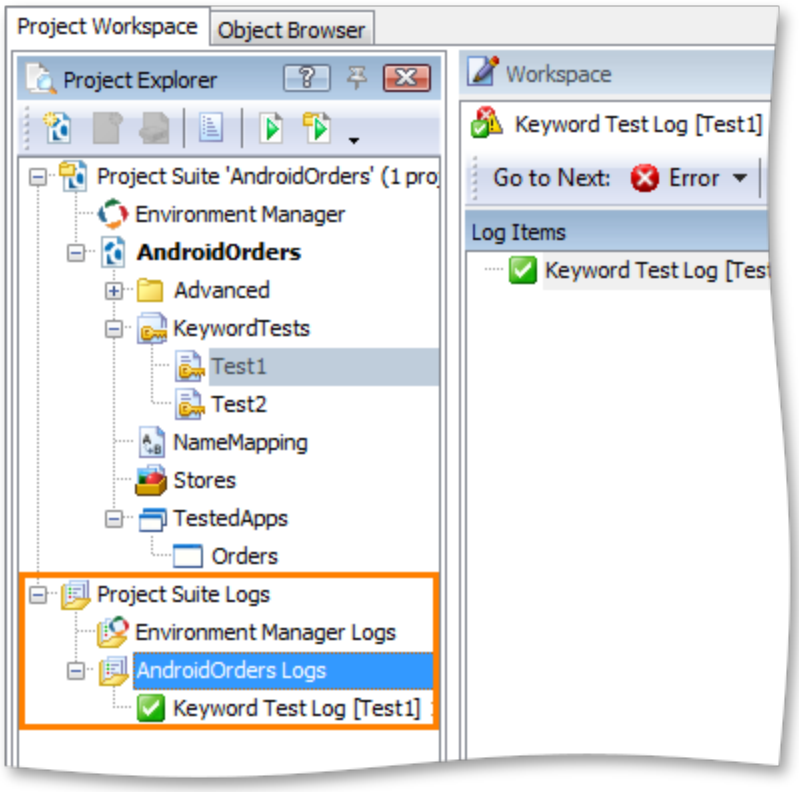
- TestComplete executes test commands until the test ends. You can stop the execution at any time by pressing  **Stop** on the Test Engine toolbar or in the indicator, or by selecting **Test | Stop** from TestComplete's main menu.

  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's 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.

6. Analyzing Test Results

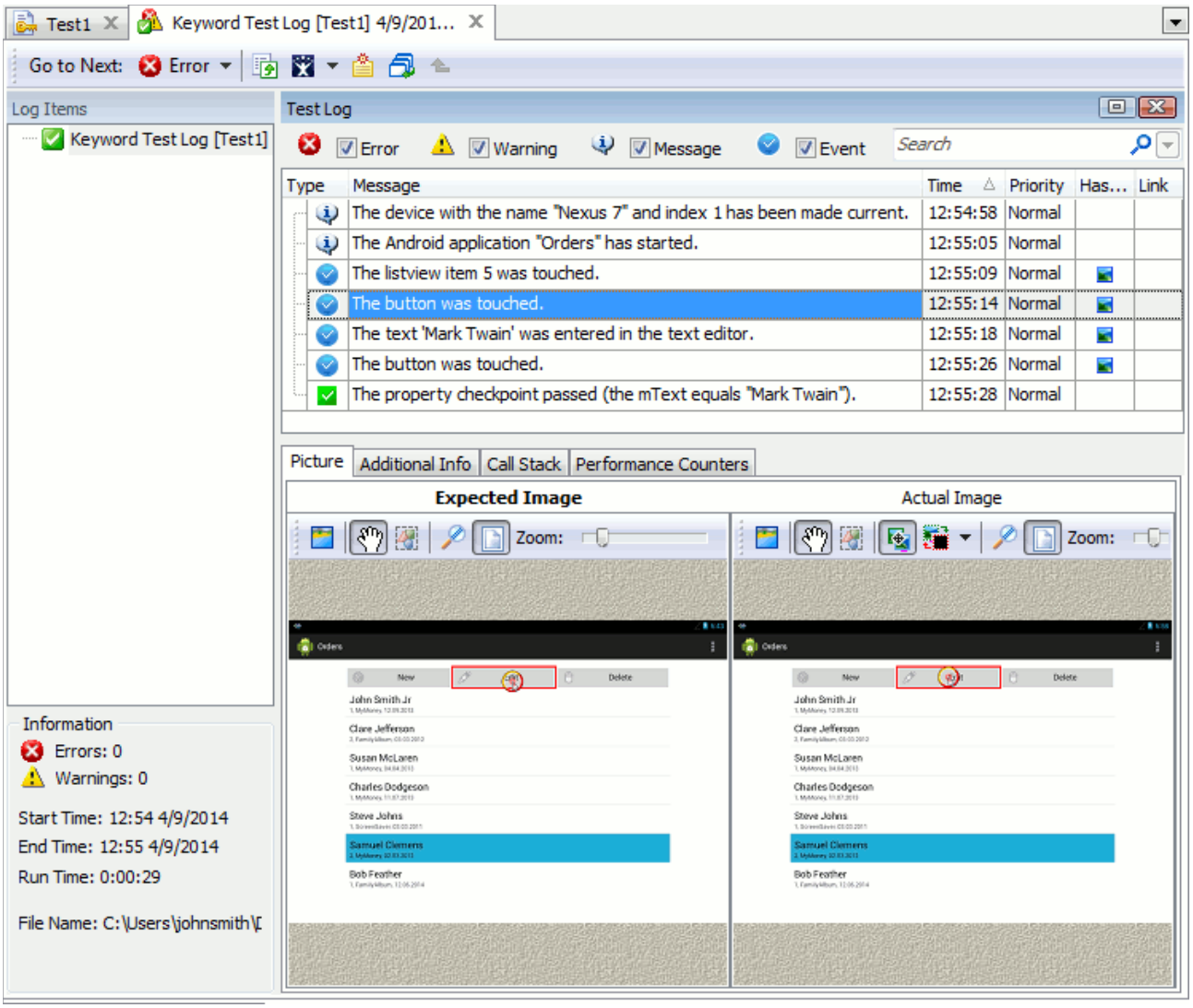
TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the **Project Explorer** panel under the **Project Suite Logs | AndroidOrders Log** node. This is the primary workspace for looking up the test history of the project and project suite. 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.

In our example, the log is as follows –



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 **Additional Info** and **Picture** panes that are below the message list.

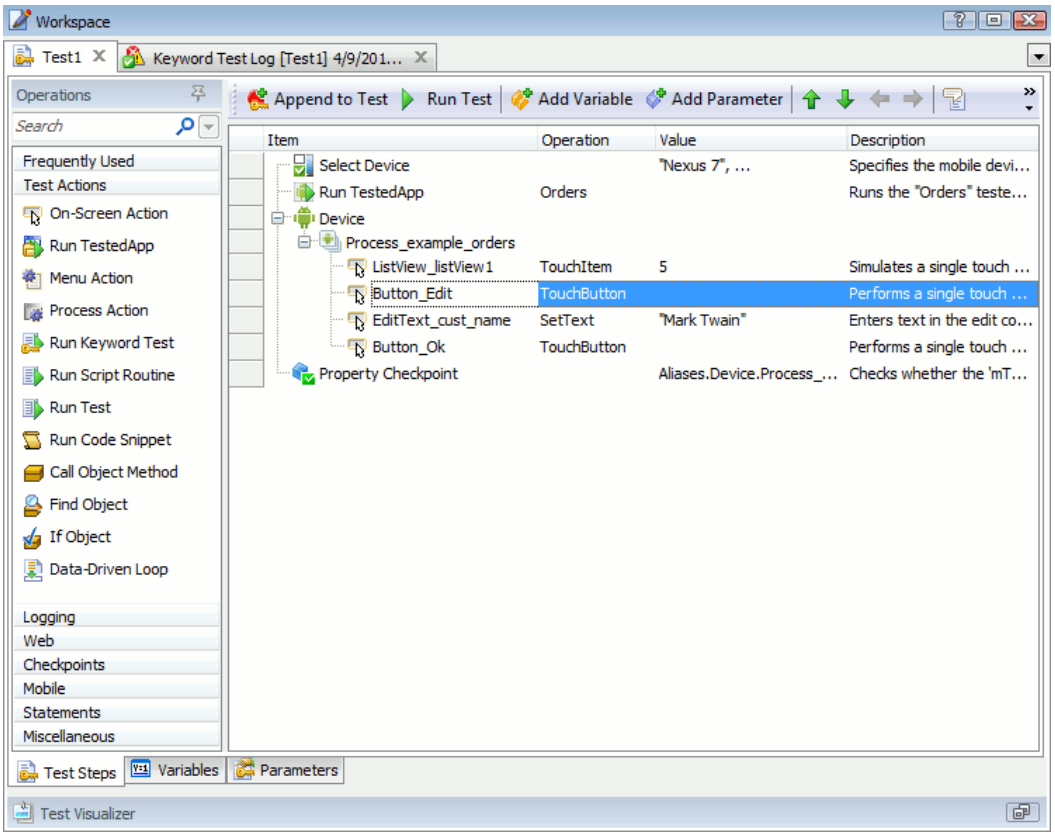
The Picture panel displays the images that show the expected and actual application state before executing the selected test command. "Expected" is the image that was captured for the command during test recording, "actual" means the image that was captured during test run.) The test log includes a special button that lets you

compare the images and easily see the difference. This simplifies the search for errors that may occur in your test. For more information, see topics of the *Test Visualizer* section.

The log's **Call Stack** pane displays the hierarchy of test calls that led to posting the selected message to the log.

The log's **Performance Counters** pane displays values of the performance counters monitored during the test run. The values are shown in the form of graphs.

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 touched" message in the log, TestComplete will highlight the keyword test operation that performed this action:



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 instance, developers could change the application's behavior, the recognition attributes of windows and control 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 them, and so on.

One of the most typical reasons faced by novice users is the difference in the application's 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 the test, it must also be running before you run the test; if the test actions were performed on a particular screen of the application, you should also open that screen when you run the test, and so on.

If your test ran with errors, examine and fix them to make the test pass.

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

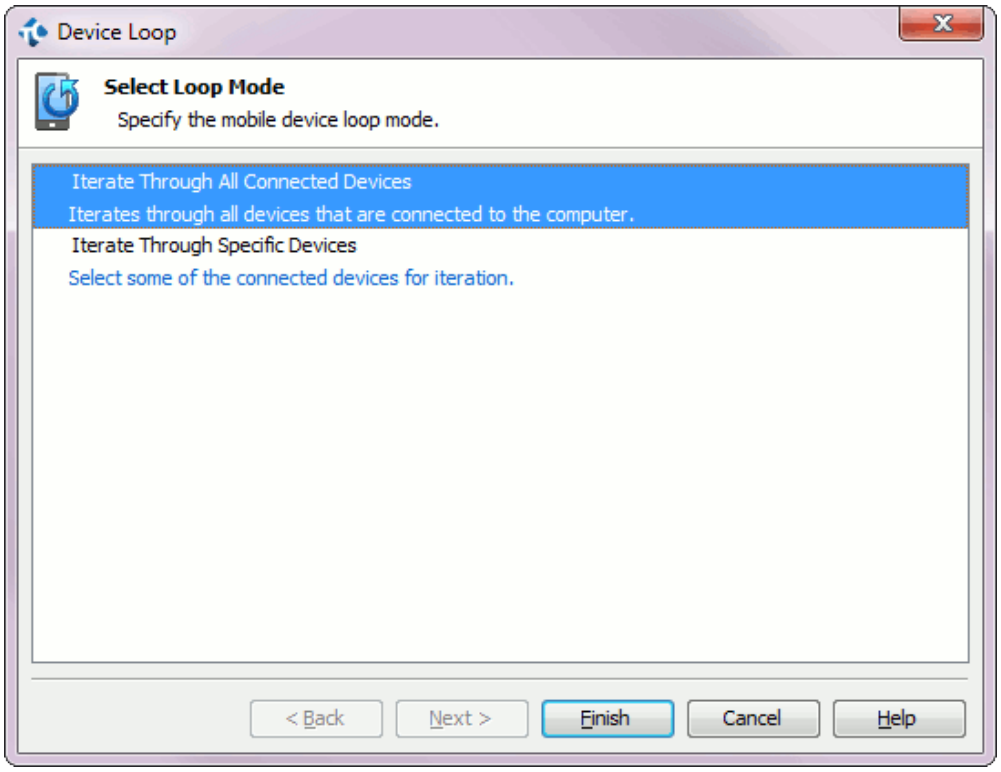
7. Running Test on Multiple Devices

After you have checked that the test is executed successfully on one of the mobile devices, you can modify it to be executed on multiple mobile devices. In this way you can ensure that the mobile application works correctly on different types of mobile devices (smartphones, tablets and so on).

Let's modify the test so it can run on different mobile devices, one after another.

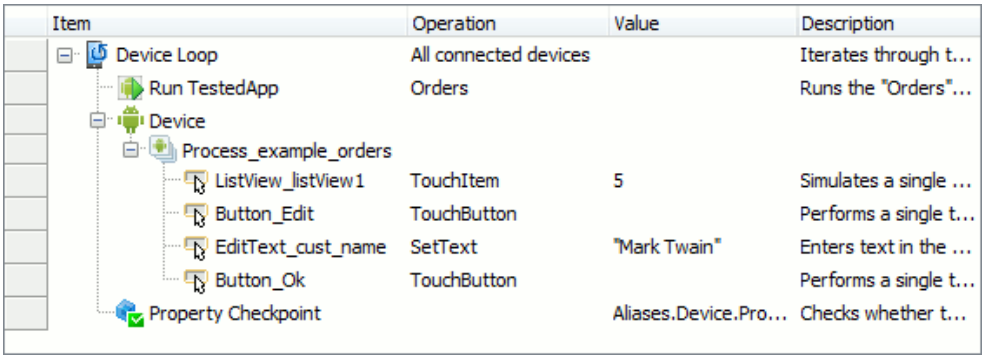
- Open the test in the Keyword Test Editor.
- Drag the **Device Loop** operation from the **Mobile** category of the Operations panel to the Test Area. Drop the operation at the very beginning of the test (above all other operations). This will add the operation to the test.

In the ensuing operation parameters dialog, select **Iterate Through All Connected Devices** and press **Finish**.



- Delete or disable the Select Device operation from the test. It is no longer needed because the Device Loop operation iterates through the mobile devices.
- Select all test operations that go after the Device Loop operation and click ➡ to move them inside the loop. Now these operations will be executed on each loop iteration.

  Here is what the resulting test should look like:



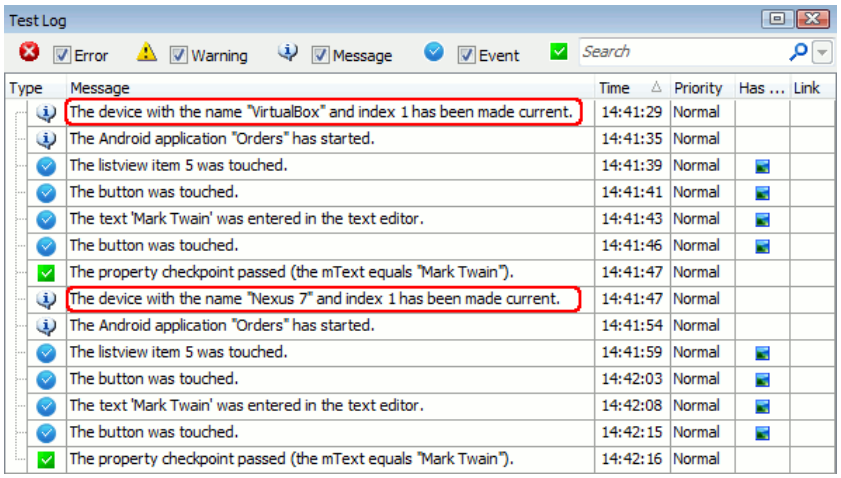
- Save the test by selecting **File | Save** from TestComplete's main menu.

Prepare and connect another mobile device as described in the "Preparing Mobile Device" section of *Preliminary Steps*.

Now run the resulting test.

TestComplete will repeat the test operations several times. Each time the test actions will be performed on a different mobile device.

The test log contains information about which mobile device was used and results of the test operations performed on each device.



Testing iOS Applications

This tutorial explains how to test *iOS applications* with TestComplete. We are going to prepare a testing environment for mobile testing, connect to a mobile device and deploy the tested application to it, create and run a simple test and analyze the results. The test will emulate user actions over the mobile application and verify some data.

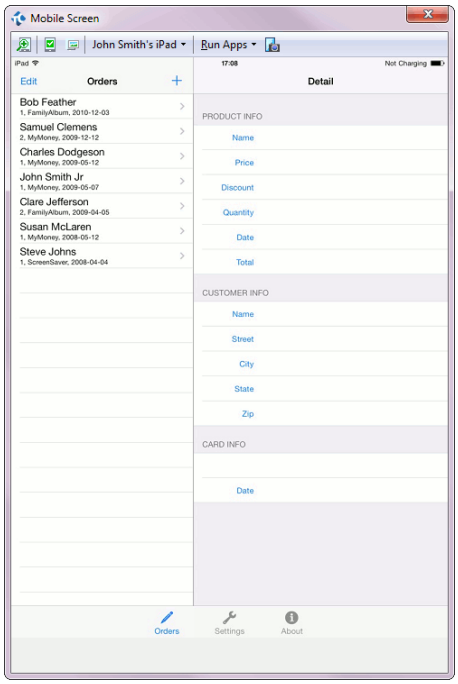
Requirements

To complete this tutorial, you need the following:

- A Windows computer with TestComplete Mobile module and Apple iTunes. iTunes is needed, because it contains the USB driver that TestComplete uses to connect to iOS devices.
- An iPhone, iPad or iPod touch running iOS 8.0 - 8.4.1 and 9.0 - 9.3.
- A Mac computer with Xcode, iOS SDK and an iOS development license - to compile the sample Orders application.

About Tested Application

In our explanations we will use the iOS version of the *Orders* application that is shipped along with TestComplete. The 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.



The application should be compiled on your Mac and then deployed to the mobile device as described in this tutorial.

1. Preparing iOS Device

Before you can test iOS applications with TestComplete, you need to:

- Add your iOS device to the application provisioning profile (for the iOS Developer Program only).
- Connect the device to the TestComplete computer via a USB cable.

Adding Test Devices to the Application's Provisioning Profile

If you have the iOS Developer Program, you need to add the test device to the provisioning profile of the tested iOS application to install the application on this device. You can register test devices on Apple's Member Center web site:

- Get the device identifier (UDID). For this purpose, launch iTunes, select your iOS device and on the **Summary** tab, click **Serial Number**. The serial number will be replaced with the UDID.

- Open the Devices section of the Member Center:

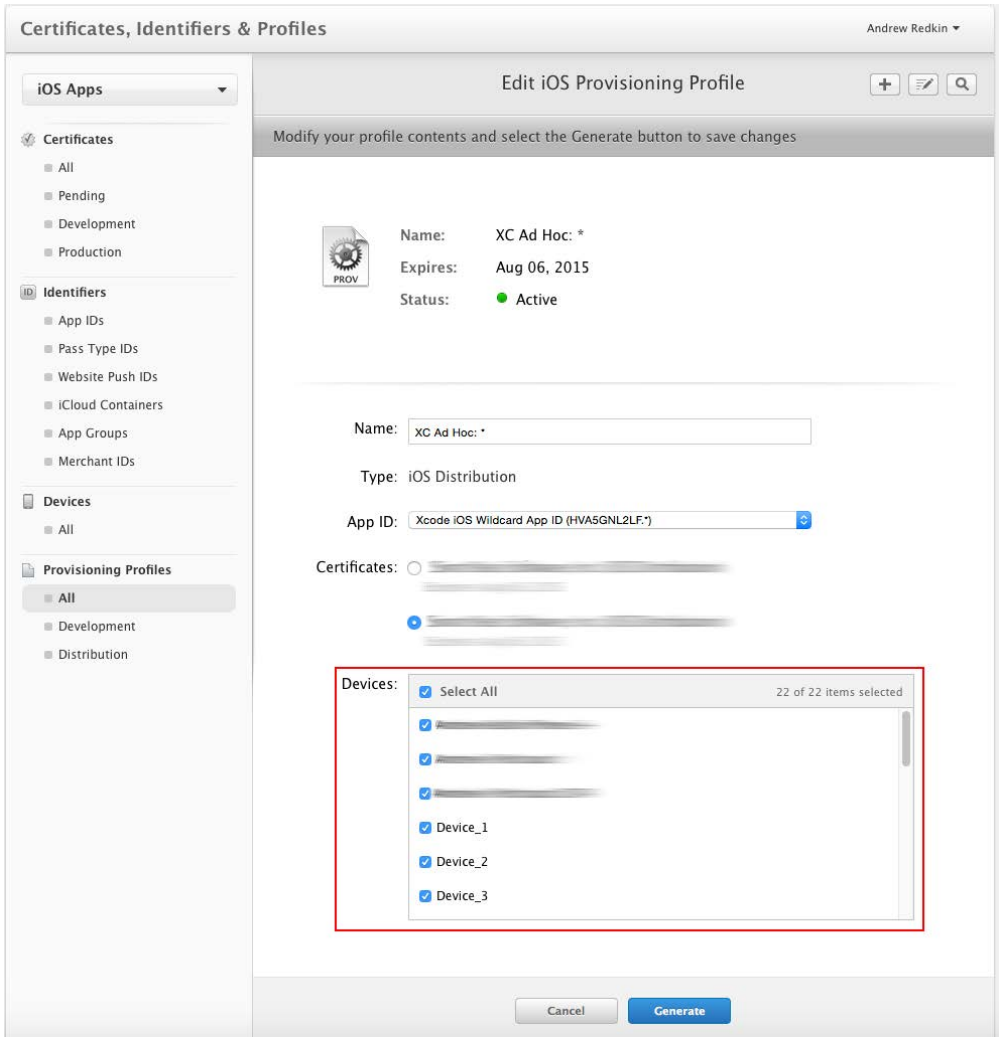
  ⇨ http://developer.apple.com/ios/manage/devices/index.action

- Click the  button in the upper-right corner.
- Select the **Register Device** radio button.
- Enter the device name and UDID.

- Click **Continue** and then **Register**.

You have registered your device for testing. Now you need to add it to the application's provisioning profile.

- Select **Provisioning Profiles** from the menu on the left.
- Select the profile to which you want to add the device and click **Edit**.
- In the **Devices** list, select all the devices where you want to test the application.



- Click **Generate** to save the changes.

Connecting Device to TestComplete

Connect your iOS device to the TestComplete computer via a USB cable.

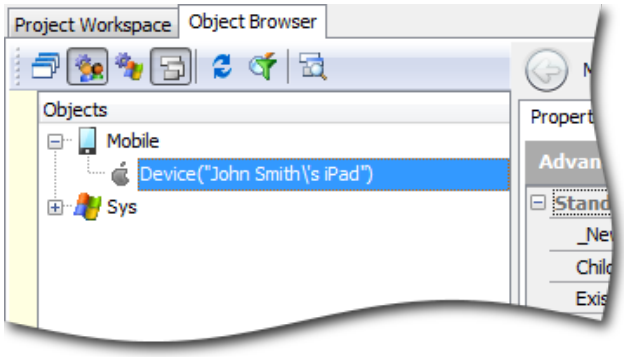
If the device shows a dialog asking whether you trust this computer, tap **Trust**.

Also, we recommend that you change the following settings of the device:

- Go to **Settings | General**.
- Tap **Auto-Lock**.
- Select **Never**.
- Go to **Settings | Wallpapers & Brightness**.
- Move the brightness slider to the minimum value.

This will prevent the device from locking the screen during the test run and reduce battery consumption of the connected device.

To make sure TestComplete "sees" the connected device, switch to the Object Browser. You should see the **Device("*your_device_name*")** object under the **Mobile** object.



On the next step, we will prepare the sample Orders application for testing.

2. Preparing iOS Application for Testing

To test your iOS application with TestComplete, you need to prepare it using the TestComplete library. You can instrument iOS applications in TestComplete, but you will need user certificate files from your Mac in this case. These files verify that the application was created by a specific developer and can be run on specific devices. You need to do this only once. You will then be able to use these certificate files to instrument all your iOS applications.

You can instrument your application directly in Xcode. To do this, you add the TestComplete static library to your project, build the application and then resign it using the script provided with TestComplete

1. Get the Apple WWDR Certification Authority File

The Apple Worldwide Developer Relations (WWDR) certificate links your development certificate to Apple. You can download it from here:

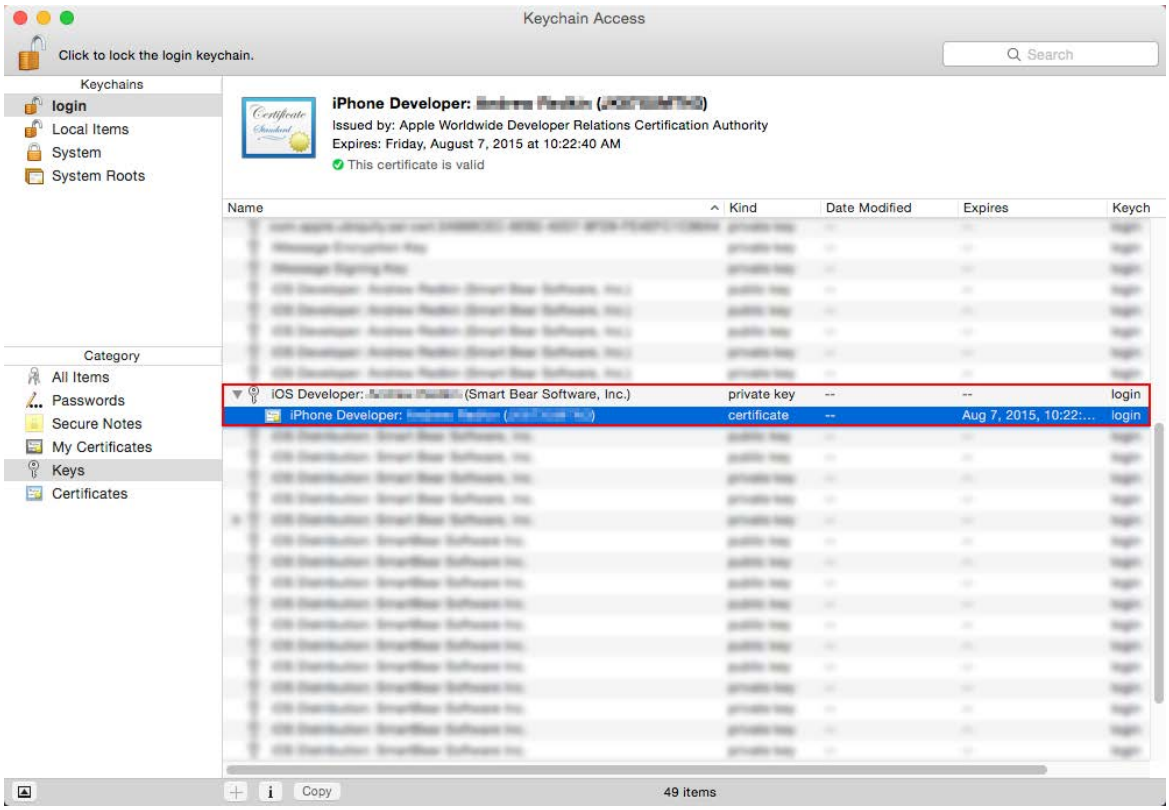
- *http://developer.apple.com/certificationauthority/AppleWWDRCA.cer*

2. Get Developer Certificate and PEM Files

A developer certificate verifies that the application was created by a specific developer. The PEM file contains the keys used to sign the application created by this developer.

To get developer certificate and PEM files, you need a Mac with the needed certificate installed. The PEM file must contain keys for the developer certificate you will export.

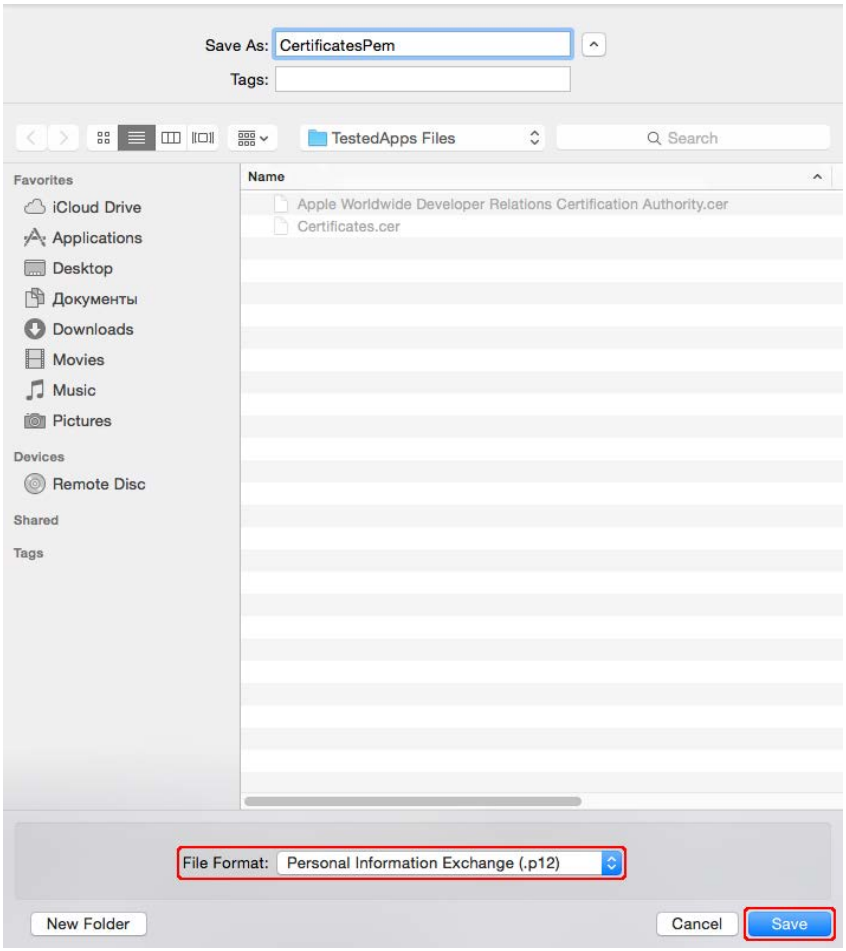
1. On your Mac, open the KeyChain Access application and switch to the Certificates category.



2. Right-click the needed developer profile and choose Export.
3. Make sure that you are exporting the certificate in the .cer format and click **Save**. You have created a developer certificate file.
4. To create a PEM file, switch to the **Keys** category.
5. Right-click the key associated with the developer and choose **Export**.

Tip: To make sure you are downloading the correct key, click the drop-down arrow. The expanded item should mention the same developer certificate that you have exported. See the image above.

6. Choose the Personal Information Exchange (.p12) format and click **Save**.



7. Enter the password that is used to protect the items you are exporting and click **OK**. You will need this password later to convert the file to the PEM format. TestComplete does not need this password.

8. Enter your Keychain Access password and click Allow.

9. After you export the file, you need to convert it to the PEM format using the command line:

   - Open the command line window and navigate to the folder to which you exported the file.
   - Run the following command line:

     ```
     openssl pkcs12 -in CertificatesPem.p12 -out Certificates.pem -nodes
     ```

     You will need to enter the password for the .p12 file that you specified earlier.

3. Get a Provisioning Profile

A provisioning profile is a collection of certificates that combine developers and devices into one development team and allow using these devices for testing.

To perform testing with TestComplete, we recommend that you use the iOS App Development profile. In this case, you will be able to automate the deployment of your tested applications to your iOS devices and launching the applications on them with specific TestComplete scripting methods or keyword-test operations.

The provisioning profile must belong to the same developer as the certificate and PEM files.

1. Open the following web page in your browser and log in to the Member Center:

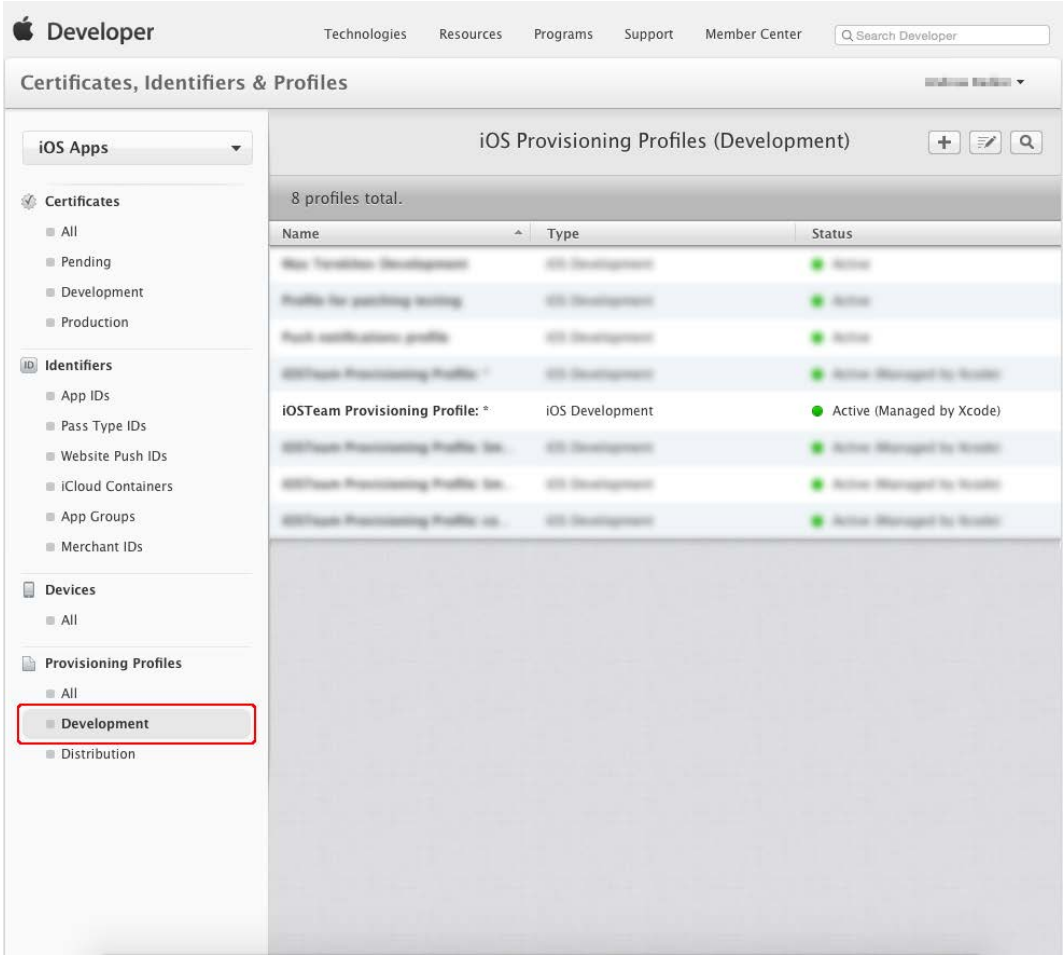
   https://developer.apple.com/membercenter/

2. In the **Developer Program Resources** section of the next page, click the **"Manage your certificates..."** link:

3. Click **Provisioning Profiles** in the **iOS Apps** section.

4. On the subsequent page, select **Provisioning Profiles | Development**.

   You will see a list of provisioning profiles of the development type:

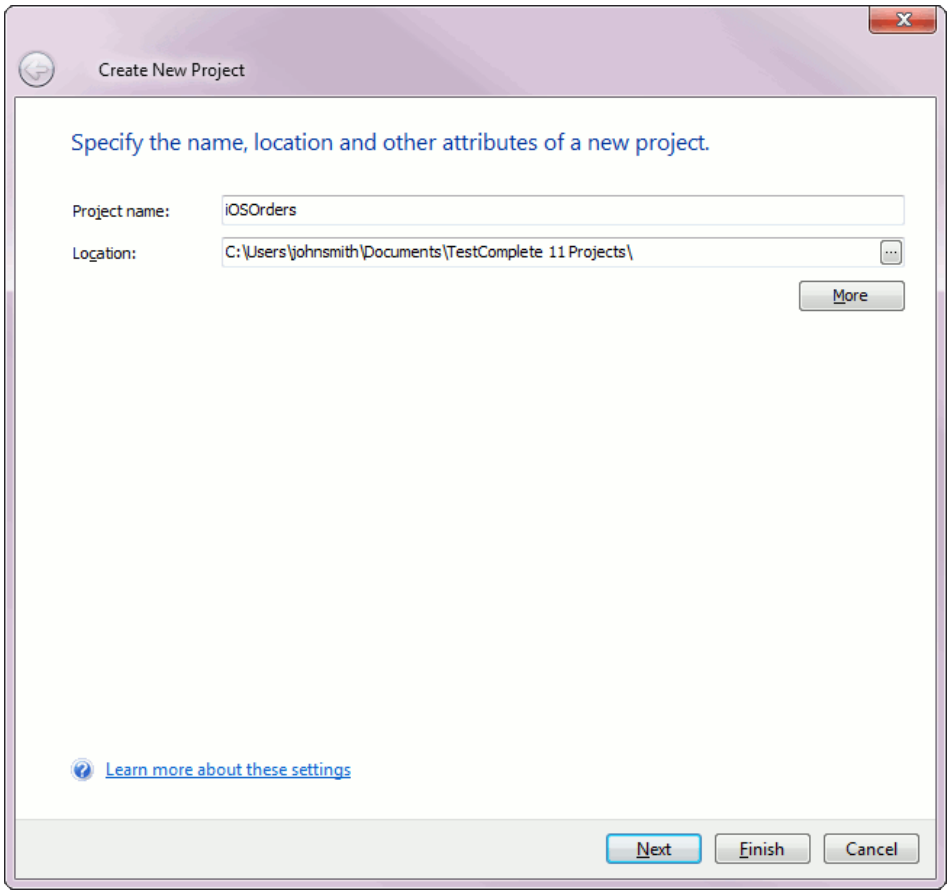


5. Click the needed profile and click **Download** in the expandable section.

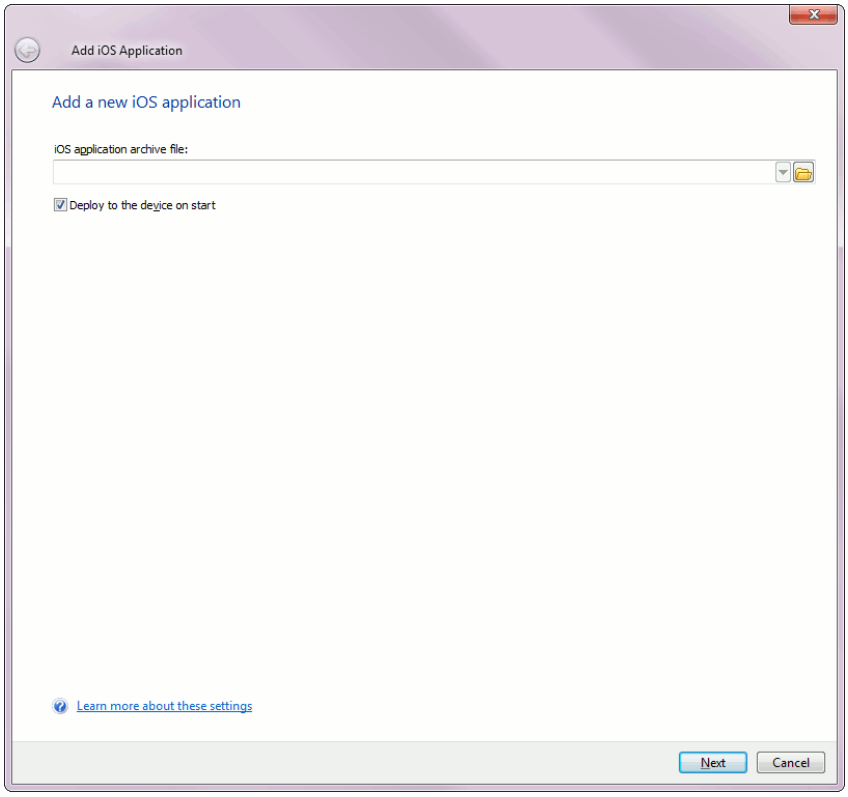
3. Creating a Test Project

Let's create a new test project in TestComplete:

1. If you have an open project or project suite, close it by selecting **File | Close** from TestComplete's main menu.
2. Select **File | New | New Project** from TestComplete's main menu. The Create New Project wizard will appear.
3. In the wizard, enter the project name and the location where you want to store the project. In this tutorial, we will name the project *iOSOrders* and use the default location.



4. On the next page, select **iOS applications**. We can now add the tested iOS application to the project. To do this:
    1. Click **Add**. The Add iOS Application wizard will appear.
    2. On the first page of the wizard, specify the path to the *Orders.ipa* file (copy this file from the Mac computer to the computer where TestComplete is installed, if you have not done this yet).

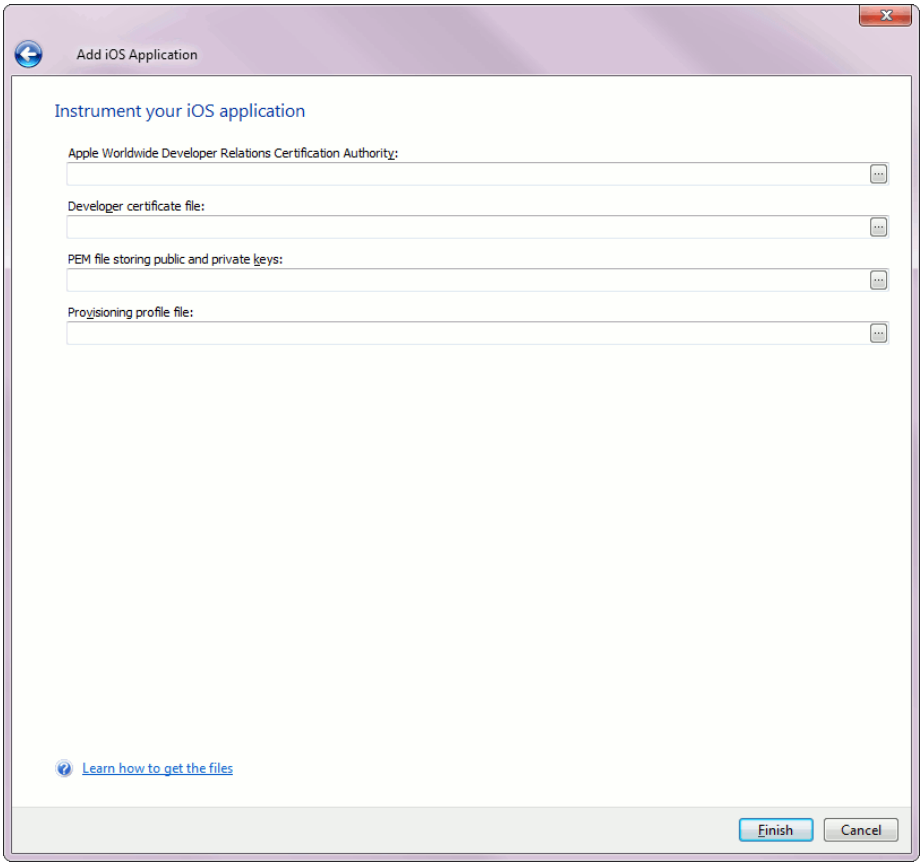


3. Make sure the *Deploy to the device on start* option is selected and click **Next**.

4. On the next step, TestComplete will check whether your application is instrumented or not. If it is not instrumented, the wizard will ask whether you want to instrument the application. Choose to instrument.

5. On the next step, the wizard will suggest backing up the original version of the application's .ipa file. The backup copy is useful if you need the uninstrumented version of your application. If you do not need it, clear the **Backup the original .ipa file** checkbox. Click **Next** to continue.

6. On the last page of the wizard, you specify the path to the files holding certificate data for signing your application.



7. Click **Finish**. TestComplete will recompile your iOS application, and add it to the Tested Applications list of your project.

8. Make sure that the *Autorun* check box is selected. If it is selected, TestComplete automatically launches the Orders tested application on the mobile device when you start test recording. If the check box is clear, then, to record user actions over the application, you will have to launch the application manually.

5. On the next page, you can enable or disable Test Visualizer functionality. To do this, check or uncheck the corresponding check box. Click **Next**.

6. On the last page, choose the preferable scripting language and click **Finish**

TestComplete will create the project and show its files in the Project Explorer.

Now you can install and launch the Orders application on the connected iOS device directly from TestComplete. To do this, right-click the application in the Process Explorer and select **Run** from the context menu.

4. Creating a Test

Now let's record our first test.

Creating Tests in TestComplete

TestComplete allows you to create tests in two ways. You can:

- Create tests manually
- Record tests

When you create a test manually, you enter all the needed commands and actions that your test must perform via appropriate script objects or keyword test commands. This approach is very 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 different problems. For example, while creating a test manually you must know the classes and names of your application's objects you want to work with. To solve such problems, TestComplete includes a special feature that lets you easily create tests. 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. We call this feature "**recording a test**", because you create a test visually and in one sense you record the performed actions to a script or keyword test. It is a very useful approach and it does not require much experience in creating tests. So, in this tutorial we will demonstrate how to record tests with TestComplete.

Recording Tests in TestComplete

The recording includes three steps:

1. You start recording by selecting **Test | Record | Record Keyword Test** or **Test | Record | Record Script** from TestComplete's main menu or from the Test Engine toolbar. You can also start recording by clicking **Record a New Test** on the Start Page.

   After you command TestComplete to start the recording, it will switch to the recording mode. In this mode, TestComplete minimizes its main window, displays the **Recording** toolbar and, optionally, displays the **Mobile Screen** window.

   The Recording toolbar contains items that let you perform additional actions during the recording, pause or stop recording and change the type of the recorded test (from the keyword test to script code, or vice versa).

   The Mobile Screen window displays the screen of the connected mobile device. This window is used to record tests against mobile applications. The window is displayed if the *Automatically display Mobile Screen on recording* option is enabled.

2. After starting the recording, perform the desired test actions: launch the tested application (if needed), work with the application as you normally do: select menus, touch buttons and other controls 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.

Planning a Test for the iOS Orders Application

The sample iOS Orders application works with a list of orders. Suppose you need to test whether the application's Edit Order page functions properly and modifies data in the order list. In this case --

- **Test purpose**: The test should check whether the Edit Order page 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 data in the order list. 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 the expected value. We will add a special comparison command to the test for this. This command will post comparison results to the test log, so we will see whether verification failed or passed successfully.

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

Recording a Test for the iOS Orders Application

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 have the help instructions at hand, you can print them before starting recording. Or, if you have two monitors, you can move the window of TestComplete's help system to the other monitor.

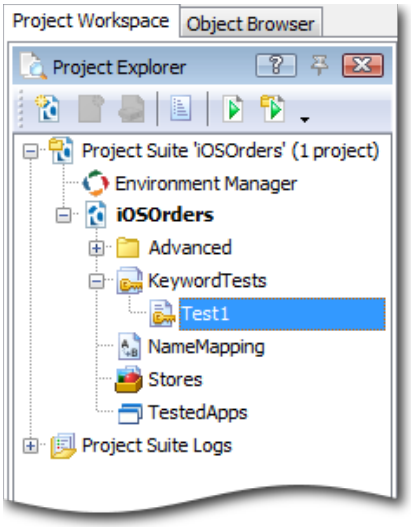
Now let's record a keyword test for the sample iOS Orders application.

1. Before recording a test, open the Mobile Screen window. TestComplete records only those actions over mobile applications that you perform in this window.
   - Click the **Show Mobile Screen** button on the Test Engine toolbar.
2. Open the keyword test in the editor.

   When creating a new project, TestComplete automatically creates an empty keyword test in this project. You will add test commands to this test.

To open the keyword test:

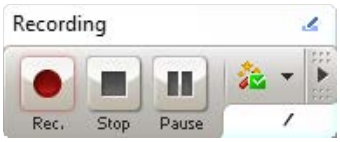
- Switch to the **Project Explorer** panel.
- Expand the **KeywordTests** node.
- Double-click the **Test1** node.



3. To start recording, select the **Append to Test** item on the test editor's toolbar.

   TestComplete will display the Recording toolbar on screen. If the **Interactive Help** panel is visible, TestComplete will also show information about the recording process in it.

   By default, the Recording toolbar is collapsed:



   Click the arrow button to expand the Recording toolbar and view all its buttons:

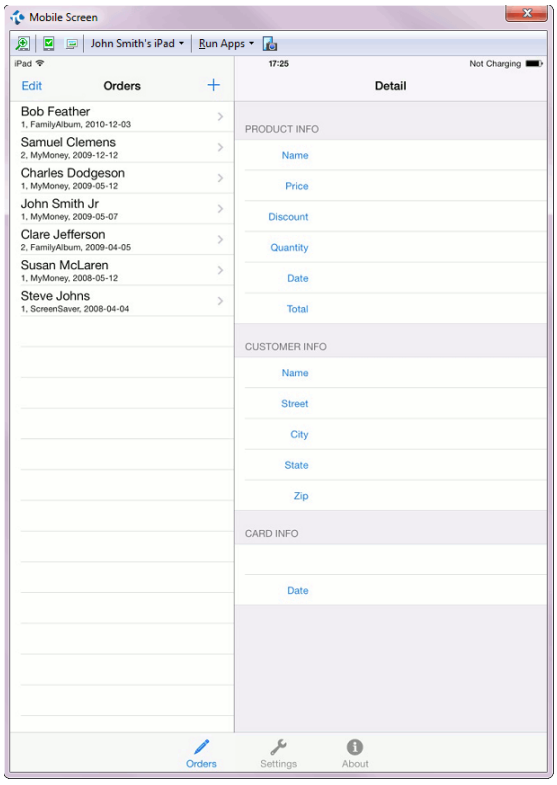
4. After you start recording, TestComplete will automatically deploy the Orders application to the mobile device and start this application. This will happen because you enabled the application's *Deploy to the device on start* and *Autorun* options while creating the test project on the previous step.

   If the *Autorun* option is disabled, you will have to launch the application manually. To do this, select the **Run Tested Application** command from the Recording toolbar:

You can also launch the application from the Run Apps drop-down menu on the Mobile Screen window's toolbar.

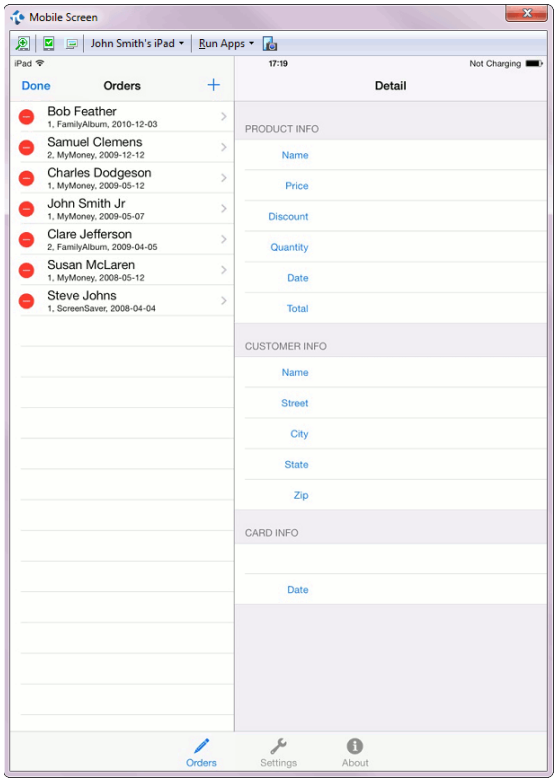
TestComplete records the application start using a special application launch test command. You will see this command later when analyzing the recorded test.

5. Wait until the iOS Orders application installs and starts on the mobile device. The Mobile Screen window will display the initial window of the application:

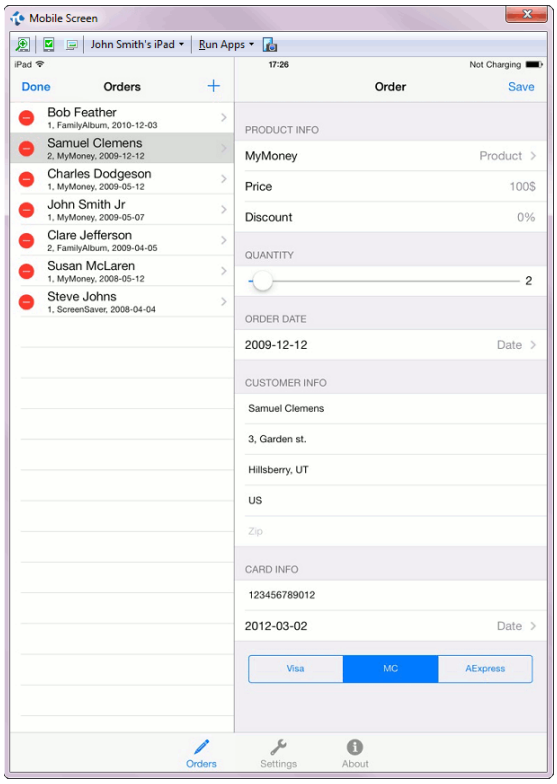


If the **Interactive Help** panel is visible, resize or move it so that it does not overlap the Mobile Screen window. Your actions on this panel are not recorded.

6. In the Mobile Screen window, press the Edit button. This will switch the Orders application to the edit mode.



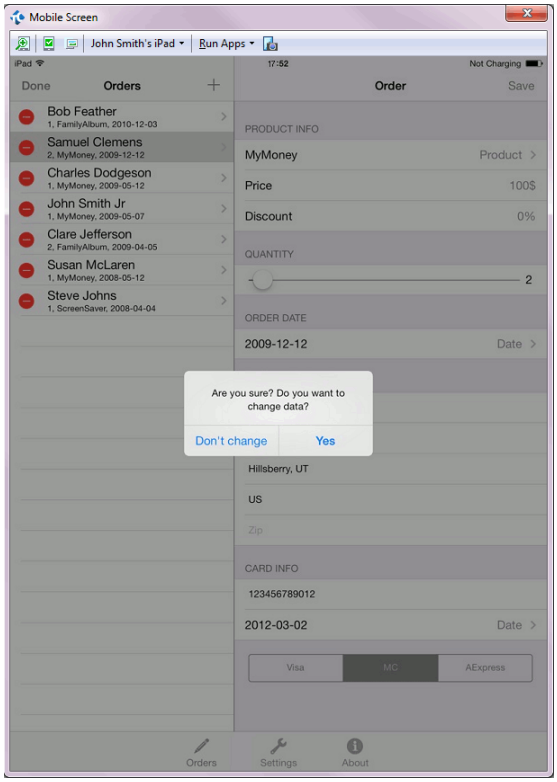
7. Click the second order in the list made by *Samuel Clemens*. This will display the Edit Order panel:



8. Let’s change the customer name in the order details.

In the CUSTOMER INFO section, clear the *Samuel Clemens* text, type *Mark Twain* and press Enter. Use your desktop keyboard to input text in the Mobile Screen window.

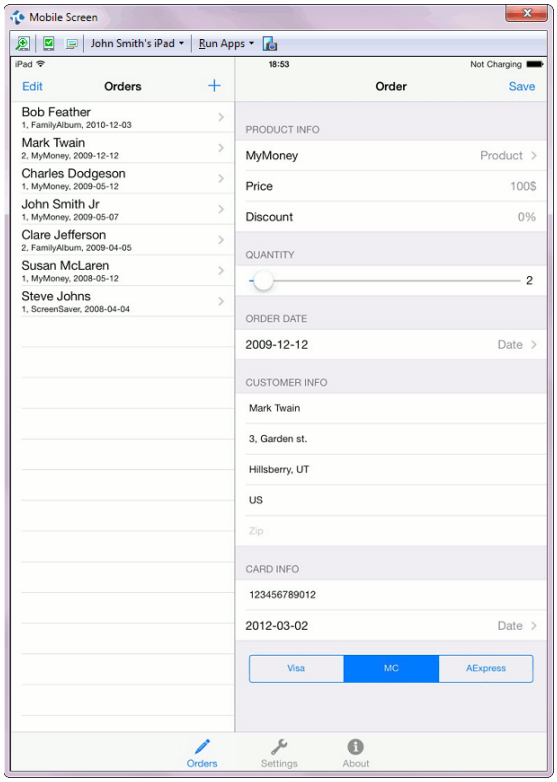
9. OK, now press the **Save** button in the Edit Order panel. This will invoke the confirmation dialog:



10. To confirm the changes, press the **Yes** button.
11. Now, if you are recording the test on an iPhone or iPod touch 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.
12. Press the Done button to switch off the edit mode.

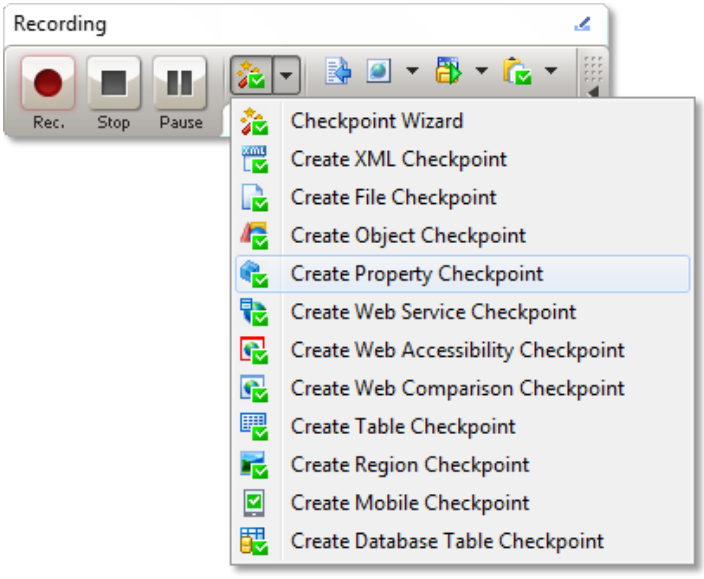
    The orders list will return to normal mode:



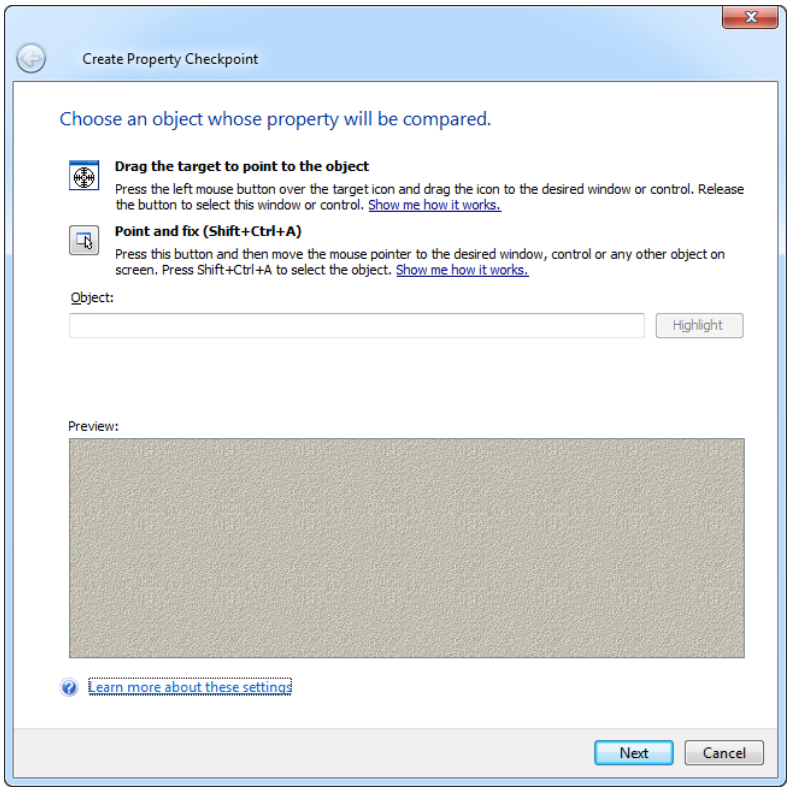
13. Now let's insert a comparison command into our test. It will verify that the application's customer list displays the modified name - *Mark Twain*.

    We call comparison commands **checkpoints**. TestComplete offers various types of checkpoints that can be used to verify different types of data (see *Checkpoints* in TestComplete Help). One of the most frequently used checkpoints is the **Property checkpoint**. It is used to check data of application controls. We will use this checkpoint in our tutorial.

    - Select **Create Property Checkpoint** from the **Checkpoint** drop-down list of the Recording toolbar:



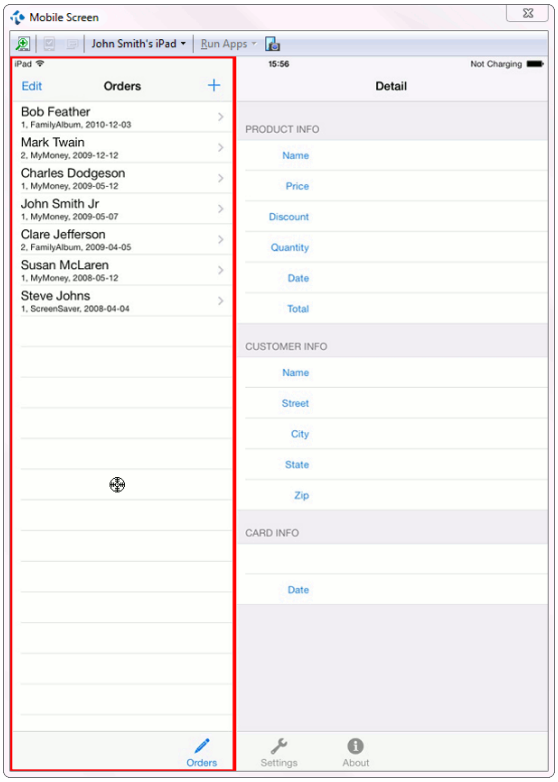
- This will invoke the **Property Checkpoint** wizard. It will guide you through the process of checkpoint creation.



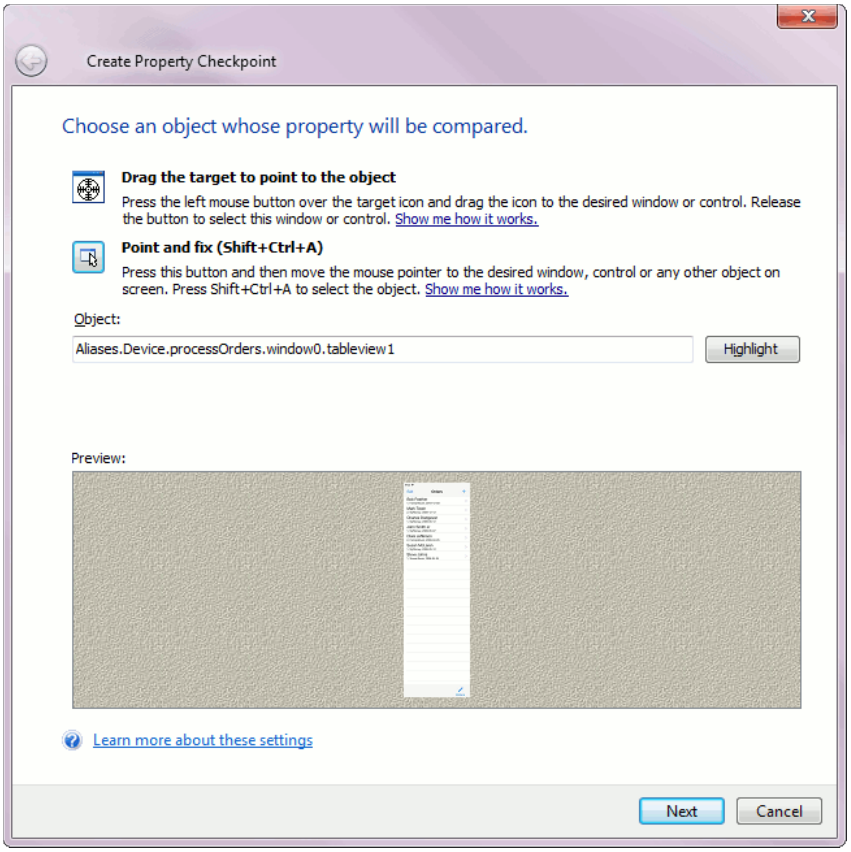
- On the first page of the wizard, click the target glyph () with the left mouse button and keep the button pressed.

  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 orders list and when the entire list is highlighted with the red frame:



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



Click **Next** to continue.

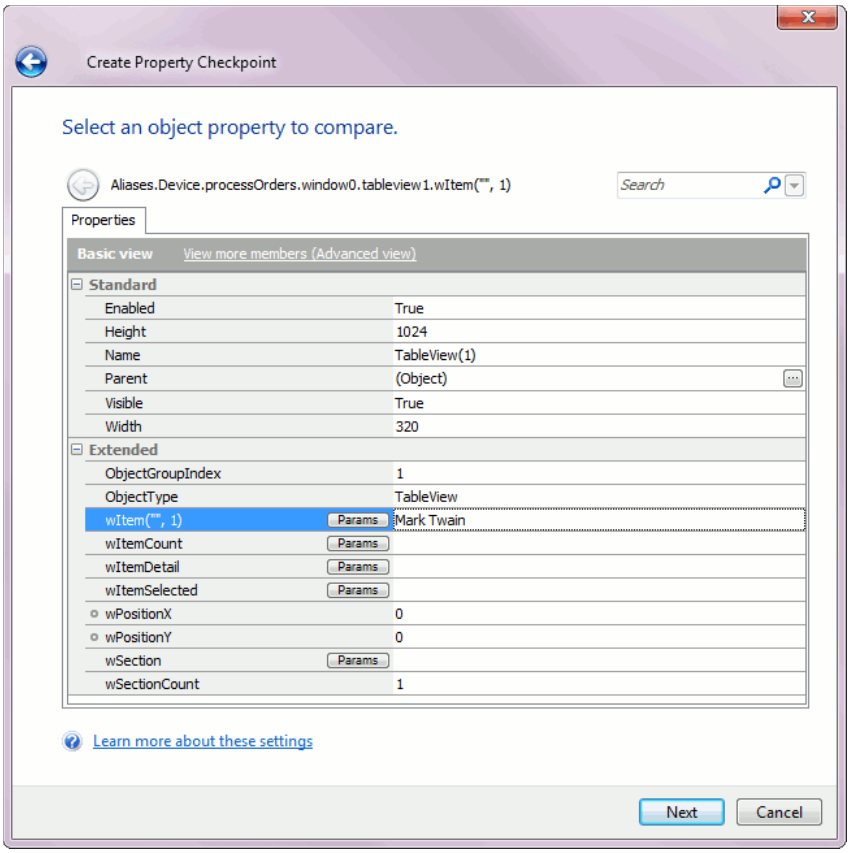
- The next page of the wizard displays a list of the selected object's properties. This list includes the properties provided by TestComplete and the properties defined by the tested application. To view all the available properties, click the **View more members (Advanced View)** link.

  TestComplete appends three groups of properties to the selected object: one group includes properties common for all tested windows and controls. You can see them under the **Standard** node. Another group includes properties that are specific to mobile controls. You can see these properties under the **Extended** node. Finally, all native properties of the corresponding control are displayed under the **iOS** node.

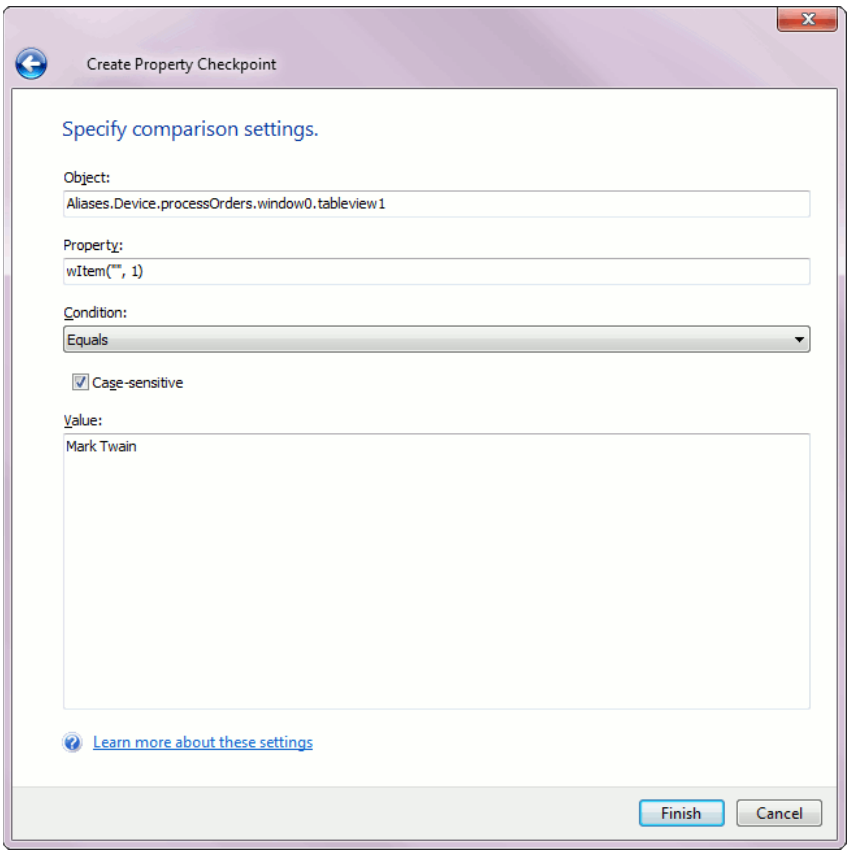
  To verify the data, we will use the `wItem` property of the parent `TableView` object. It provides access to the text shown in the specified item of the selected table view.

- Find the `Parent` property in the list (it is under the **Standard** node). Click the ellipsis button next to it. It makes the parent object selected, so the dialog shows its properties.

- In the list of the `TableView` object's properties, find the `wItem` property (it is under the **Extended** node). Click the **Params** button and enter 1 in the **Index** field. Click **OK** and then **Next** to continue.



- On the next page of the wizard, you can see the name of the property whose value will be verified, the comparison condition and the baseline data, which is in the **Value** box:

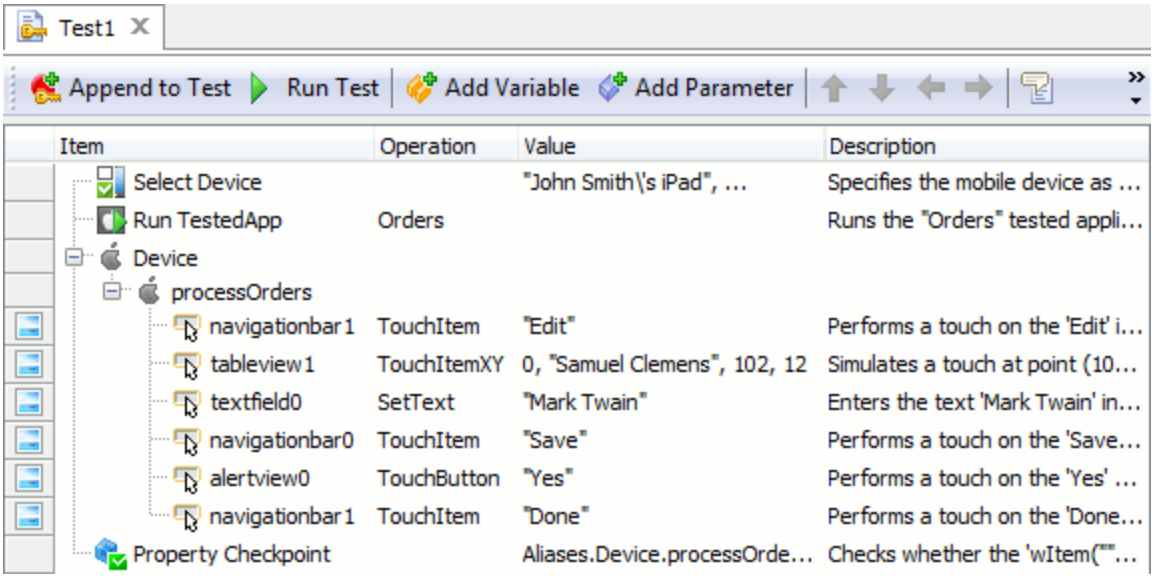


- Click **Finish** to complete the checkpoint creation. TestComplete will append the checkpoint command to the recorded test.

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

5. Analyzing the Recorded Test

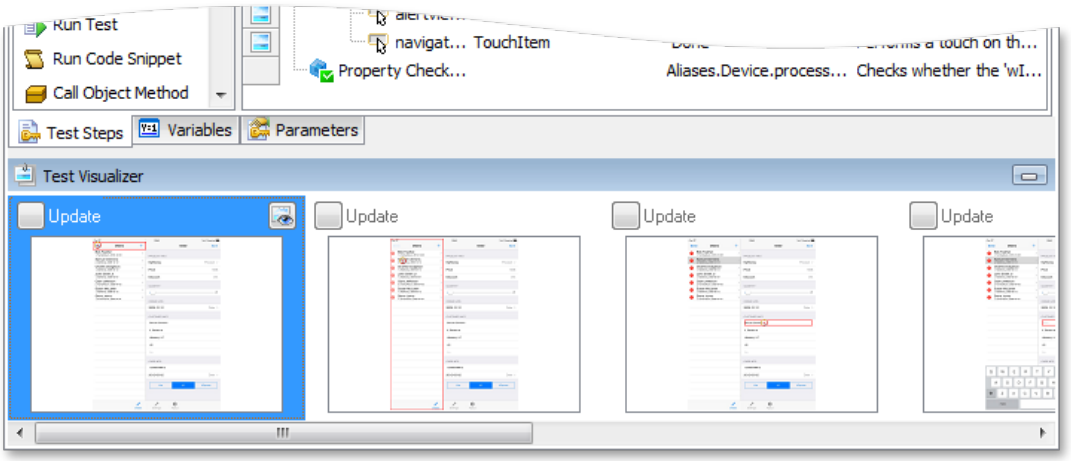
After you finish recording, TestComplete opens the recorded keyword test for editing and displays the test's contents in the Keyword Test editor:



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**.

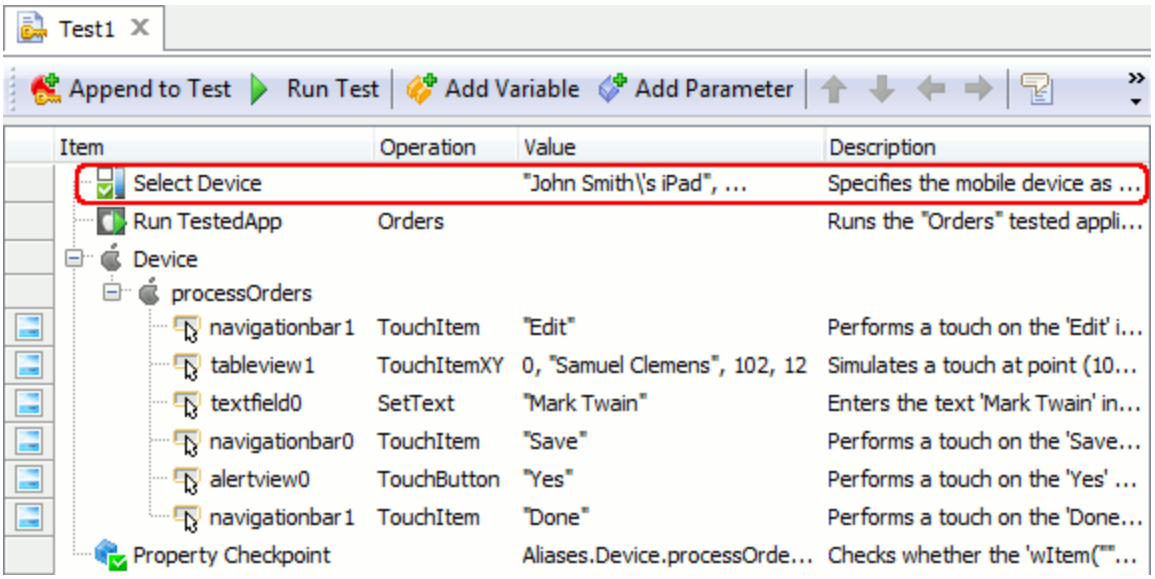
Below the commands there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:



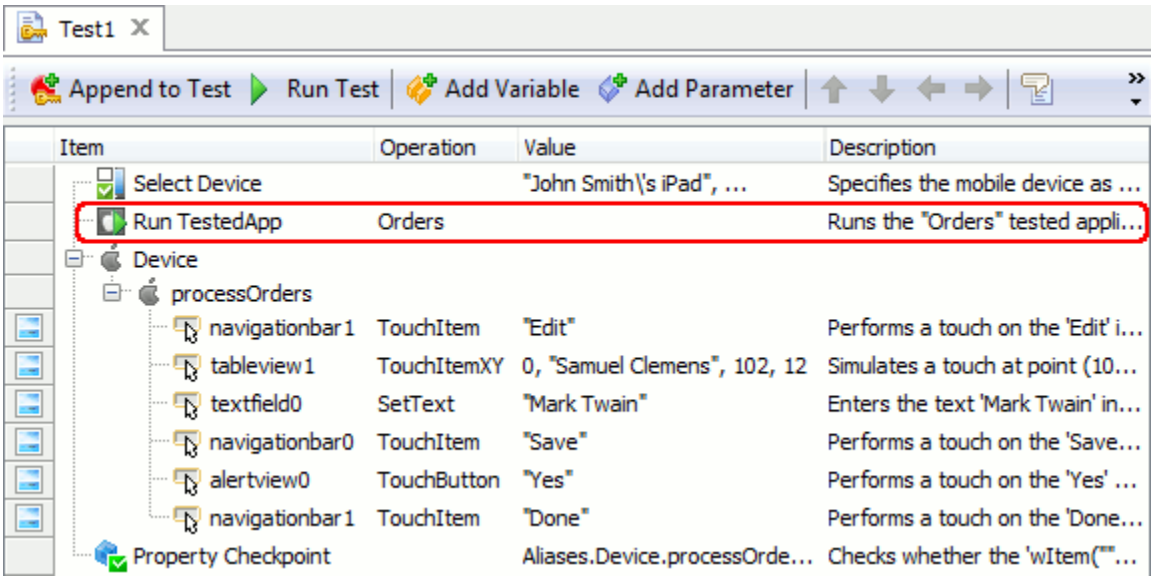
These images illustrate the recorded operations and help you better understand which action the operation performs. TestComplete captures images only for those operations that correspond to user actions (touching, dragging, typing text and so on).

When you choose an operation in the editor, **Test Visualizer** automatically highlights 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 in TestComplete documentation.

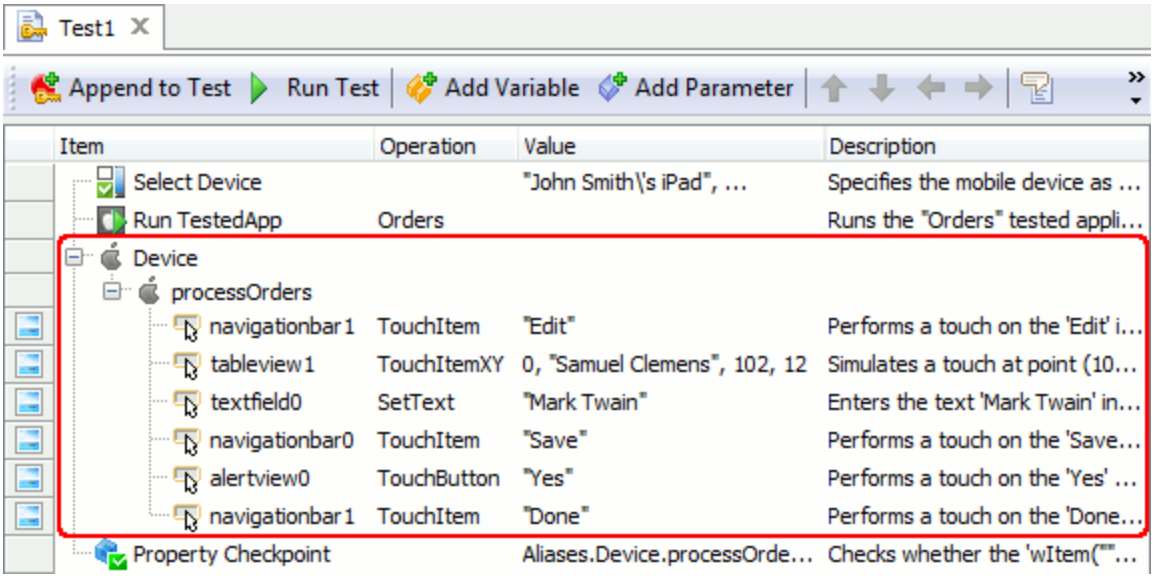
The first operation in the test is **Select Device**. It specifies the mobile device the test should work with. The other test operations refer to this device.



The second operation 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 when it detects that the application has been launched from the **Recording** toolbar.

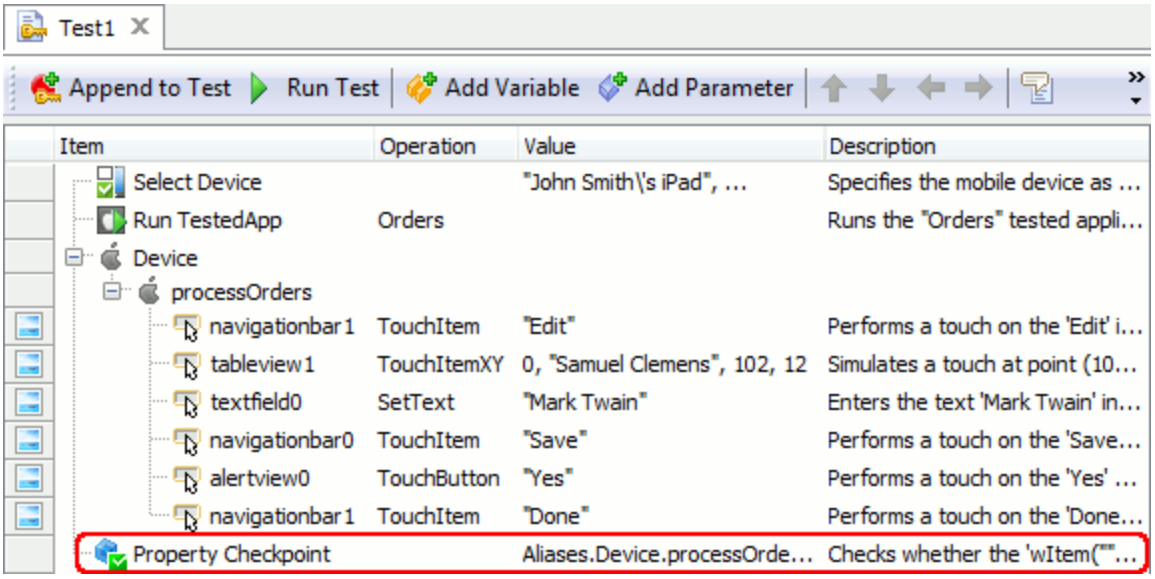


Then there are the operations that simulate your actions with the application. These operations press the Edit button, select an item in the orders list, change the value of the text field, save the changes and press the Done button.



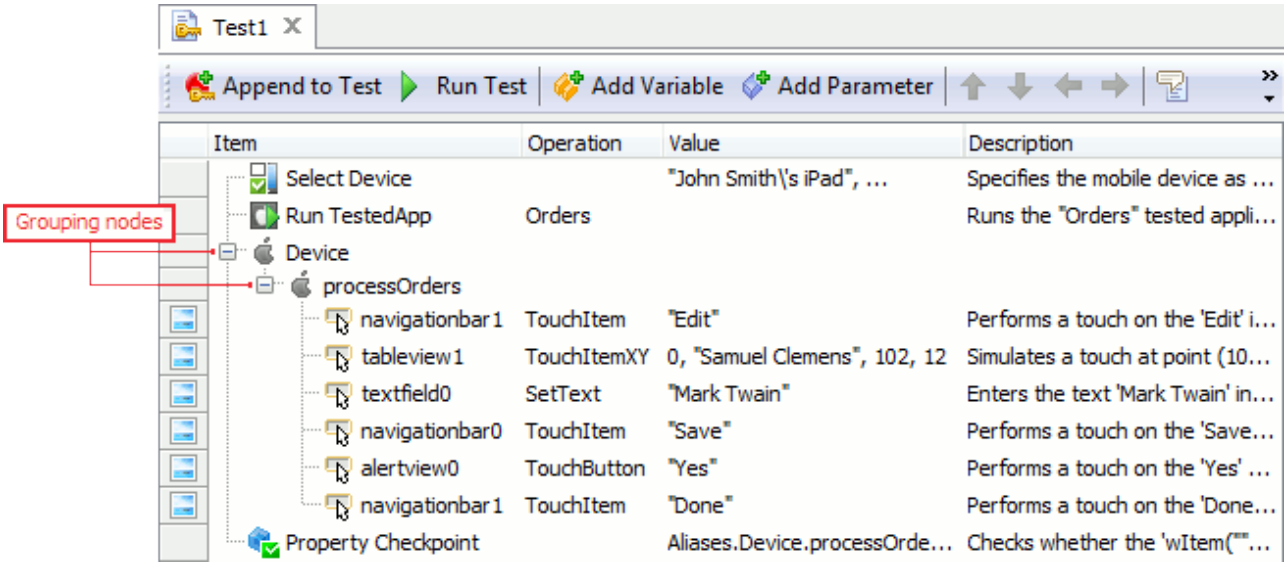
For more information on simulating touch events, text input and other user actions from your tests, see *Simulating User Actions on iOS Applications* and *Working With iOS Controls* in TestComplete Help.

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



As you can see, TestComplete automatically organizes the operations into groups that correspond to the mobile devices and processes 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've recorded user actions on one mobile device and one process. So, we have two group nodes. The “device” node groups processes that were launched on the same device. The “process” node contains all of the actions that you simulated on the process windows and controls.



You may notice that the names of the tested process and its windows and controls differ from the names that you can see in the Object Browser panel. For instance, in the Object Browser, the tested process was named *Process("Orders")* while in the test it is called *ProcessOrders*; the navigation bar was called *NavigationBar(1)* while in the test it is called *navigationbar1*, and so on.

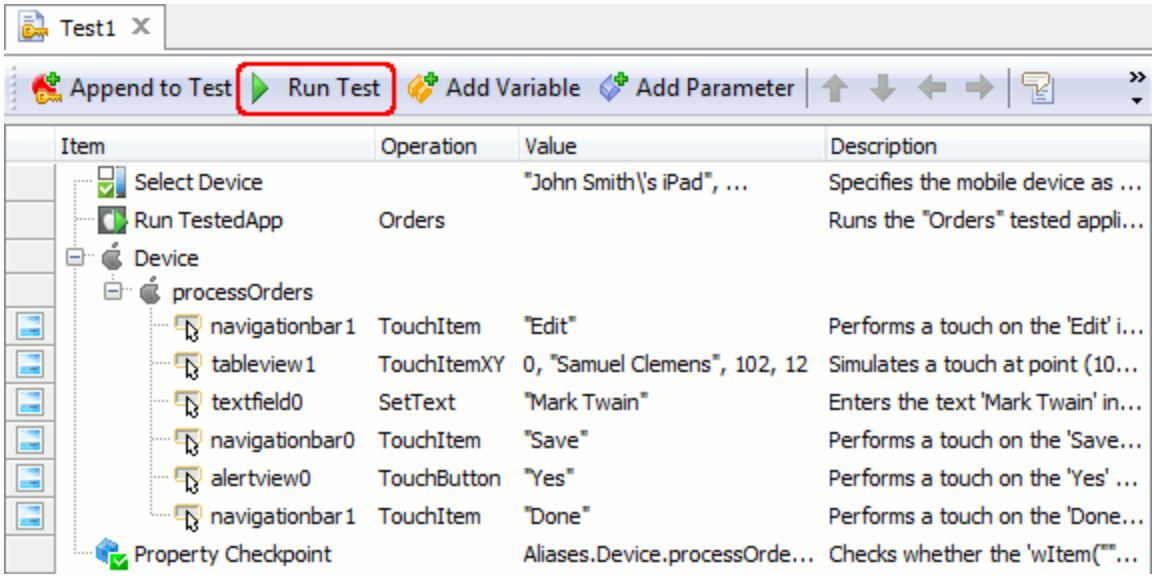
There is a logical reason for this: by default, TestComplete automatically generates and uses custom names for the objects you worked with during test recording. Generating and assigning custom names is called name mapping. TestComplete maps the names because the default names may be difficult to understand. 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. For more information on mapping names, see Name Mapping section in TestComplete Help.

6. Running the Test

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

Run the Test

To run the test, click Run Test on the test editor's toolbar:



TestComplete will launch the Orders application on the device and perform the test actions on 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's screen during the test run to avoid interference with the test actions.
- During the test execution, TestComplete displays an indicator in the top right corner of the screen:

  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's main menu.

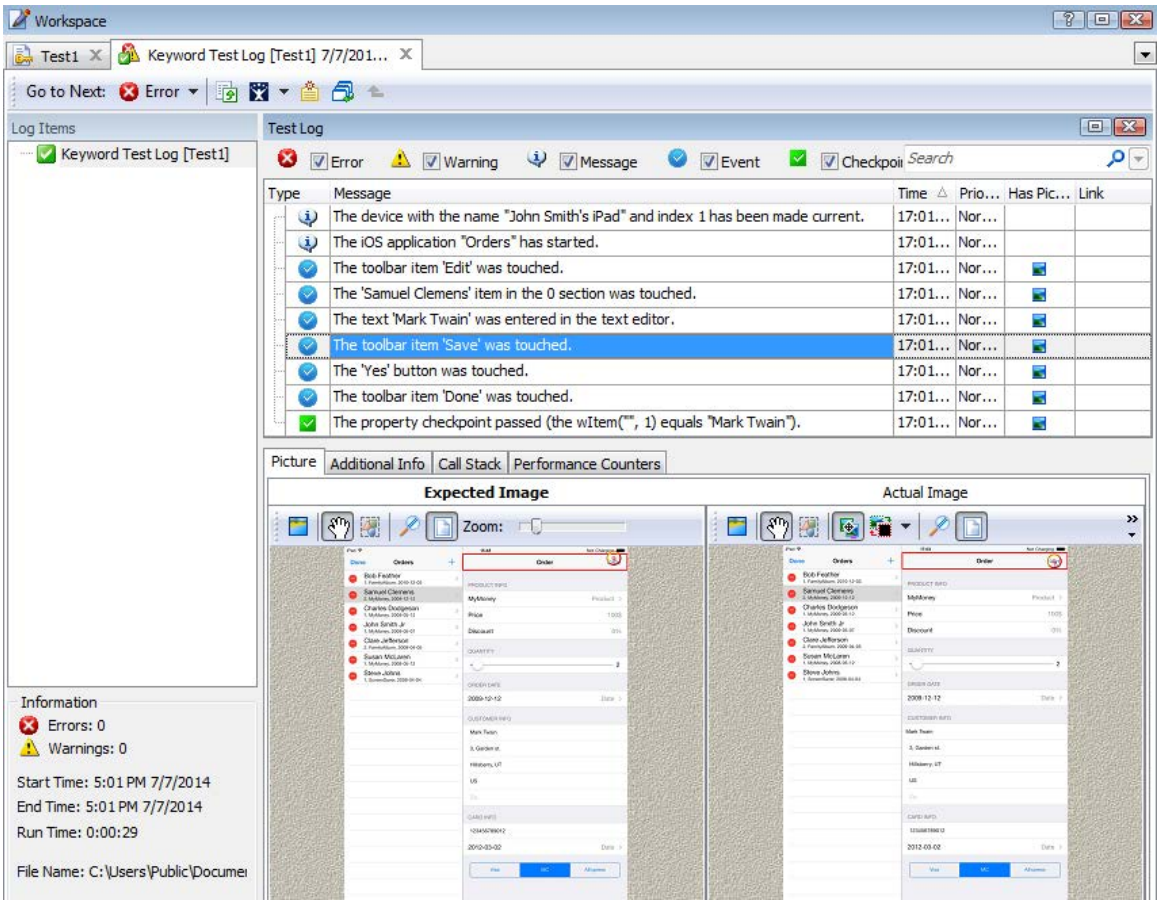
  You can pause the test execution by clicking  **Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check test variables and objects using TestComplete's Watch List or Locals panel or the **Evaluate** dialog (see *Debugging Tests* in TestComplete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

After the test finishes, TestComplete shows the test log with the results of all test operations.

The results of our test looks 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.

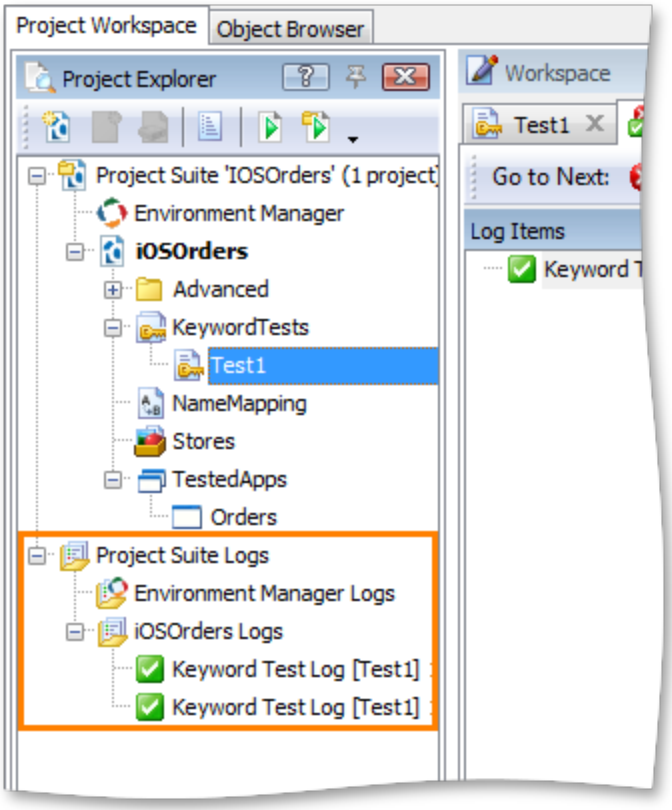
The **Picture** panel contains images that show the expected and the actual application states before the selected test command is executed ("Expected" is the image that was captured for the command during test recording, and "Actual" is the image that was captured during the test run.) The Actual Image toolbar has a special button that lets you highlight the difference between the images (if any). The images help you understand what happened in the application during the run, view the differences and find errors faster. For more information on capturing images, see the topics of the **Test Visualizer** section in TestComplete help.

There are two more panes at the bottom of the log:

- **Call Stack** is useful when debugging tests that call each other. It shows the sequence of test calls that led to the current operation or error.

- **Performance Counters** shows local or remote computer metrics (CPU load, memory usage and so on) measured during the test run. We do not use performance counters in our tutorial, but if you have a client-server iOS application, you can set up performance counters to track your server metrics.

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



Sometimes, tests fail. This can happen if the object properties in the application were changed and no longer match the identification properties specified in Name Mapping. You need to troubleshoot the failed tests to find and fix the cause of the errors. For more information about finding and fixing errors, see *Handling Playback Errors* and *Debugging Tests* sections in TestComplete Help.

8. Adjusting the Test for Running on Multiple Devices

After you check that the test runs successfully on a mobile device, you can modify it so that it can run on multiple mobile devices.

A test that will run on multiple iOS devices should be able to handle the following:

- **Different application layouts on tablets and smartphones.** An application's user interface may vary depending on whether it runs on devices with a small screen (iPhones or iPods) or on devices that have a larger screen (iPads). On small screens, the application may divide its UI elements into several layouts (panels, pages, views, tabs and so on). So, you may need to perform different actions to access the same control on a tablet and on a smartphone. As a result, a test created on an iPad will work on other iPads, but may fail on iPhones or iPod touch devices, and vice versa - a test created on an iPhone will work on other iPhones and iPod touch devices, but may fail on iPad devices.

There are several ways to handle this situation. The easiest way is to create two variants of your test - one for tablets, and one for smartphones. A more complex solution is to adapt the test for both types of devices. This may require creation of conditional test structures and probably adjusting the criteria of object recognition.

- **Different object hierarchy on different versions of iOS.** Different versions of iOS may have a different object hierarchy. So, a test may not find the needed object on another version of iOS. To resolve this issue, you may need to correct the identification properties of such mismatching objects in the Name Mapping project item (the object repository).

The Orders application, which we use in this tutorial, also changes its layout depending on whether it is running on a tablet or on a smartphone. To make the tutorial simple, we will not adapt the test for both types of devices, so please use devices of the same type (either iPhones or iPads) when running your test.

Also, the Orders application has a bit different object hierarchy when running on iOS 9 and on earlier iOS versions. The test you have created works the *textfield0* object that corresponds to the *Customer name* editor of the *Edit Order* panel. The full mapped name of this object is different on iOS 9 and on iOS 8:

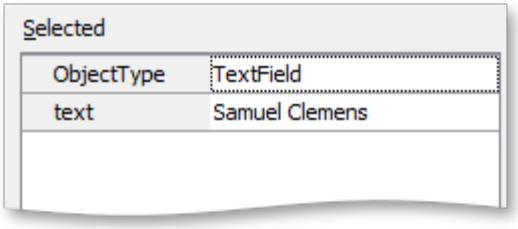
- `NameMapping.Mobile.Device.processOrders.window0.tableview0.scrollview0.TableViewCell3_0.textfield0` (on iOS 9)
- `NameMapping.Mobile.Device.processOrders.window0.tableview0.tableviewcell6.scrollview0.textfield0` (on iOS 8).

As you can see, there is no `TableViewCell3_0` object after the `scrollview0` object, but there is the `tableviewcell6` object before it.

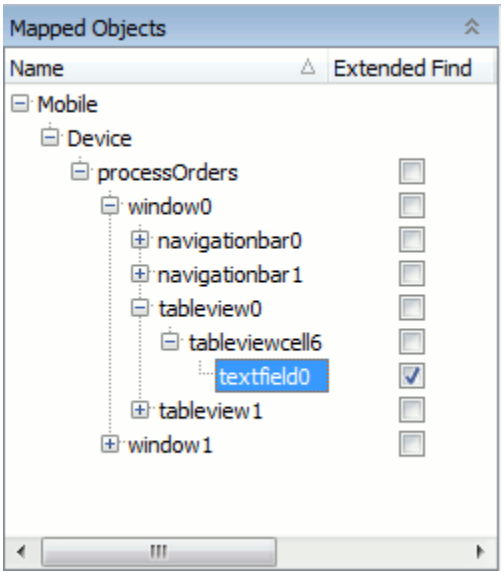
To make the test consistent, you can apply the extended search criteria to the textfield0 object. When you move an object upper in the hierarchy, you may need to clarify finding criteria to avoid wrong recognition. In our case, we must add the `text` property in order to distinguish the desired object from the rest TextField objects.

- First, you need to ensure that this object can be found by TestComplete. Switch to the Mobile Screen window and do the following:
  - Press the **Edit** button in the Orders application.
  - Select the *Mark Twain* order in the list. This will display the Edit Order panel that contains the *Customer name* editor.
  - Rename *Mark Twain* back to *Samuel Clemens*.
  - Do not close the Edit mode.
- Double-click the **NameMapping** item in the Project Explorer to open the Name Mapping editor. Here you can view and edit the identification properties of mapped objects.
- Open the **Mapped Objects** pane.
- Expand the object tree and locate the `textfield0` object.
- Double-click the `textfield0` object.
- In the dialog that appears, select the text property in the list on the right and click  .

The Selected properties on the left should look like this:



- Click **OK** to close the Edit Name Mapping Item dialog.
- In the Mapped Objects pane, drag the `textfield0` object to make it an immediate child of the `tableviewcell6` object.
- Enable the **Extended Find** check box for the `textfield0` object.



- If a confirmation dialog appears, click **Yes**.
- In the **Aliases** tree, drag the `textfield0` object's alias to the `tableviewcell6` alias (like you did in the Mapped Objects tree).
- Delete the unneeded `scrollview0` object from the Mapped Objects and Aliases trees. To delete an object, right-click it and select **Delete** from the context menu.
- Now, you need to update the object name in the keyword test.
  - Open Test1 in the Keyword editor.
  - Double-click the operation used to modify the customer name.
  - Change the object name from `Aliases.Device.processOrders.window0.tableview0.scrollview0.TableViewCell3_0.textfield0` (on iOS 9)
    or
    `Aliases.Device.processOrders.window0.tableview0.tableviewcell6.scrollview0.textfield0` (on iOS 8)

to
`Aliases.Device.processOrders.window0.tableview0.tableviewcell6.textfield0`

- Press **Finish**.

- In the Mobile Screen window, revert the Orders application to its initial state. Discard the changes made to the order (if any), close the Edit Order panel and press **Done**.

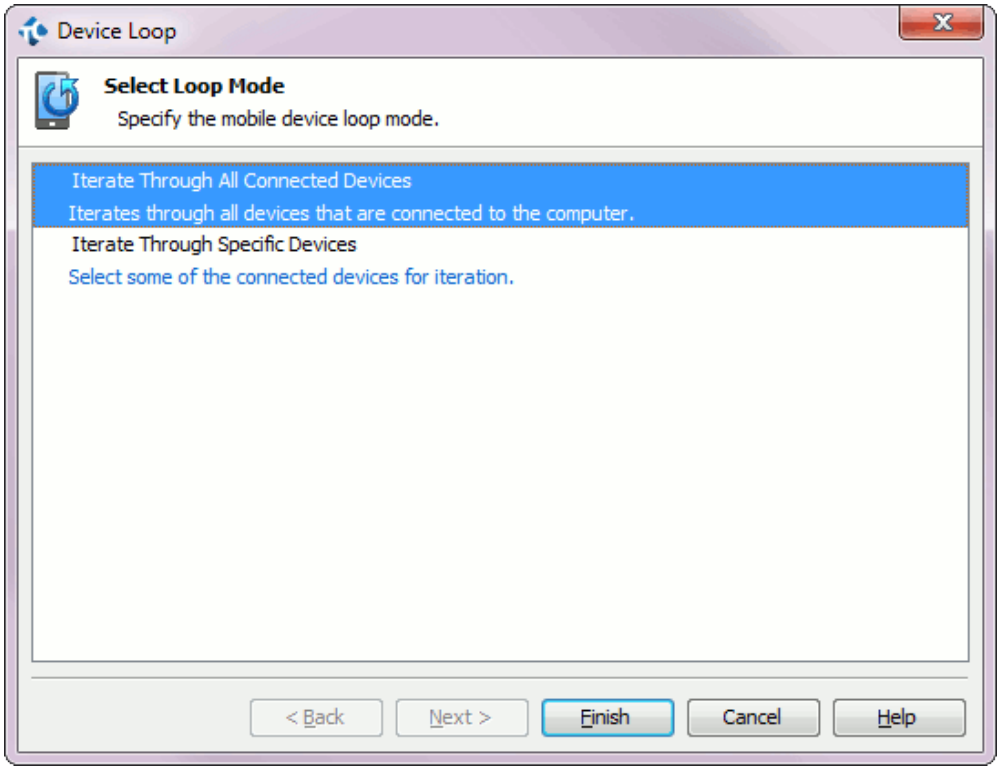
Now you can run the test on multiple devices.

9. Running Test on Multiple Devices

Let's modify the test so it can run on different mobile devices, one after another.

- Add the **Device Loop** operation from the **Mobile** category. Place it in the very beginning of the test.

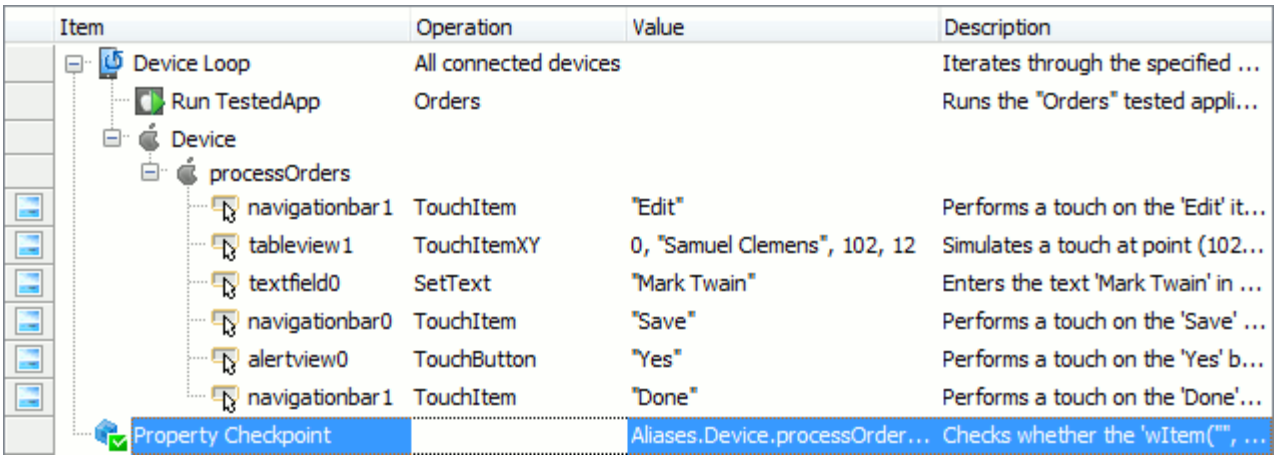
  In the ensuing operation parameters dialog, select **Iterate Through All Connected Devices** and press **Finish**.



- Delete or disable the Select Device operation from the test. It is no longer needed because the Device Loop operation iterates through the mobile devices.

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

Here is what the resulting test should look like:



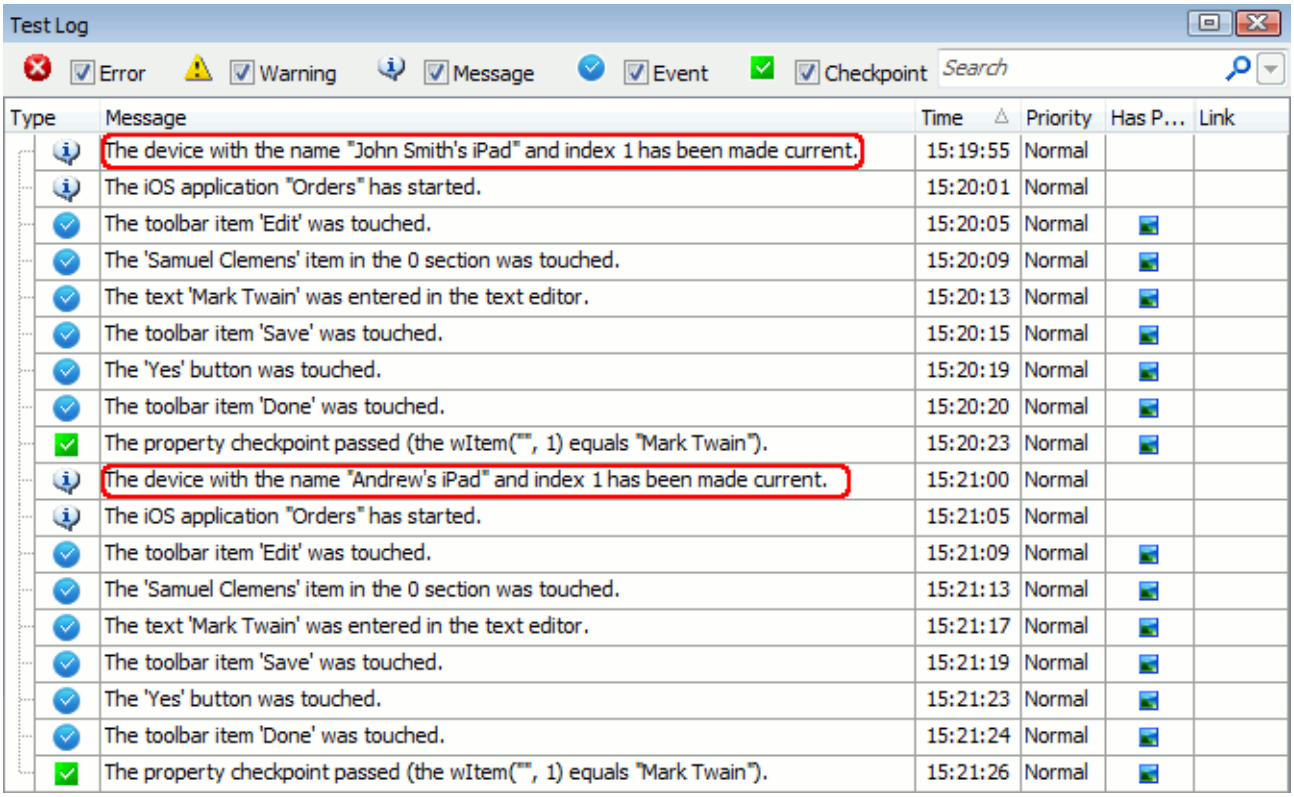
- Save the test by selecting **File | Save** from TestComplete's main menu.

Prepare and connect another mobile device as described in the "Preparing iOS Device" step. The type of the devices you use must be the same as the type of the devices you used when creating the test: either iPhones or iPads.

Now run the resulting test.

TestComplete will repeat the test operations several times. Each time the test actions will be performed on a different mobile device.

The test log contains information about which mobile device was used and results of the test operations performed on each device.



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.

- *Teamwork and Integration Into QA Process*

  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 Applications' Object and Controls*

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

- *Enhancing Tests*

  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

- *Testing Web Applications*

  Contains basic information on testing mobile applications with TestComplete.

- *Creating and Recording Web Tests*

  Explains the basics of recording and creating automated tests for web applications.

- *Finding Objects on Web Pages*

  Describes various methods for locating web page elements.

- *Cross-Browser Testing*

  Explains how to create browser-independent tests in TestComplete.

Specific for Testing Mobile Applications

Android Applications

- *Testing Android Applications*

  Contains basic information on testing Android applications with TestComplete.

- *Creating Tests for Android Open Applications*

  Provides complete information on testing Android applications that were prepared for TestComplete.

- *Simulating User Actions Over Android Devices*

  Explains how to simulate various user actions like touches, swipes and so on over your Android devices.

iOS Applications

- *Testing iOS Applications*

  Contains basic information on testing iOS applications.

- *Preparing Applications, Devices and Test Computers for iOS Testing*

  Explains how to prepare iOS applications, devices, test computers and TestComplete for testing.

- *Simulating User Actions on iOS Applications*

  Explains how to simulate user actions like touches and keystrokes on iOS applications.

Technical Support and Resources

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

http://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 **http://support.smartbear.com/**.

Index