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 ..................................................... 5  
Automated Testing .................................................................................................................. 5  
Test Types................................................................................................................................. 5  
TestComplete Projects and Project Items ..................................................................................... 6  
TestComplete User Interface ...................................................................................................... 7  
TestComplete Test Object Model ............................................................................................... 8  
Checkpoints and Stores .............................................................................................................. 10  

TESTING DESKTOP APPLICATIONS ......................................................................................... 11  
1. Plan the Test ......................................................................................................................... 12  
2. About Recording ................................................................................................................... 12  
3. Start Test Recording ............................................................................................................ 13  
4. Define an Application to Test .............................................................................................. 14  
5. Record a Test ....................................................................................................................... 16  
6. Analyze the Recorded Test ................................................................................................. 23  
7. Run the Recorded Test ........................................................................................................ 26  
8. Analyze Test Results .......................................................................................................... 29  

TESTING WEB APPLICATIONS .................................................................................................. 33  
1. Prepare Your Web Browser .................................................................................................... 34  
2. Plan the Test ....................................................................................................................... 35  
3. About Test Recording .......................................................................................................... 35  
4. Start Test Recording ............................................................................................................ 36  
5. Define a Tested Web Page ..................................................................................................... 38  
6. Record a Test ....................................................................................................................... 39  
7. Analyze the Recorded Test ................................................................................................. 48  
8. Refine the Recorded Test ..................................................................................................... 53  
9. Run the Recorded Test ........................................................................................................ 55  
10. Analyze Test Results ........................................................................................................ 57  
11. Run the Test in Multiple Browsers .................................................................................... 60  

TESTING ANDROID APPLICATIONS ......................................................................................... 63  
1. Preliminary Steps .................................................................................................................. 64  
2. Plan the Test ....................................................................................................................... 69  
3. About Test Recording .......................................................................................................... 69  
4. Start Test Recording ............................................................................................................ 71  
5. Define a Tested Application ................................................................................................. 71  
6. Record a Test ....................................................................................................................... 73  
7. Analyze the Recorded Test ................................................................................................. 80  
8. Run the Test ....................................................................................................................... 82  
9. Analyze Test Results ........................................................................................................ 84  
10. Run Test on Multiple Devices ............................................................................................ 86  

TESTING IOS APPLICATIONS ................................................................................................... 89  

© 2018 SmartBear Software http://support.smartbear.com/
1. Prepare an iOS Device ................................................................. 90
2. Prepare iOS Certificate Files .................................................... 93
3. Plan the Test ........................................................................... 95
4. About Recording ..................................................................... 95
5. Start Test Recording ............................................................. 97
6. Define a Tested Application .................................................. 98
7. Record a Test ......................................................................... 99
8. Analyze the Recorded Test ................................................... 108
9. Run the Recorded Test .......................................................... 112
10. Analyze Test Results .......................................................... 112
11. Configure the Test to Run on Multiple Devices ................... 115
12. Run the Test on Multiple Devices ....................................... 117

WHERE TO GO NEXT .................................................................................................................. 120

TECHNICAL SUPPORT AND RESOURCES ................................................................. 122

INDEX ................................................................................................................................. 123
Introducing Automated Testing and TestComplete

Automated Testing

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

So, the basic test sequence includes –

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

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

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

Test Types

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

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

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

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

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

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

Here is a sample image of TestComplete main window:

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

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

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

![Object Browser Panel](image)

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

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

**Process** objects correspond to applications running in the operating system. We use the term *process* rather than *application* because it corresponds to the concept of processes in Windows documentation.
A process object name includes the name of the process executable and its index (the index is used only if several application instances are running):

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

- **Black-box applications**

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

- **White-Box Applications**

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

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

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

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

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

Checkpoints and Stores

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

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

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

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

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

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

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

About Tested Application

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

The application is located in the following folder:

C:\Users\Public\Public Documents\TestComplete 12 Samples\Desktop\Orders

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

In this tutorial, we will use the Orders application created with Visual C#.
For the Orders projects created with other compilers (Delphi, C++Builder, Swing, and so on), download the TestComplete Samples installation package from the [https://support.smartbear.com/downloads/testcomplete](https://support.smartbear.com/downloads/testcomplete) page of our website and run it.

### 1. Plan the Test

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

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

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

### 2. About Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually** - You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.

- **Record tests** - Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them to script lines or keyword test operations. You create a test visually, and, in one sense, you record the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

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

The recording includes three steps:

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

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

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

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

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

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

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

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

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

### 3. Start Test Recording

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

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

3. On the Start page, click Record Test. TestComplete will show the Record Test wizard:
4. On the first page of the wizard, you can specify the project name, location and test type:

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

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

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

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

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

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

5. Click **Next** to continue.

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

### 4. Define an Application to Test

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

There are several ways to add applications to your project:

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

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

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

   ![Wizard page to select tested application](image)

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

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

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

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

© 2018 SmartBear Software  http://support.smartbear.com/
4. In order for the test run to be successful, the state of the tested application must be the same at the beginning of the test run and at the beginning of test recording. If your tested application is running when you start test recording, its state may differ from its state during subsequent test runs (for example, it may have some data loaded or dialogs may be open). That is why we recommend that you close all the existing instances of your tested application and start recording your tests from launching your tested application.

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

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

### 5. Record a Test

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

![Recording toolbar](image)

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

TestComplete records the application start using a special application launch test command. You will see this command later, when we will analyze the recorded test.
3. Wait until the application starts, and the application main window appears:

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

5. In the dialog, open the *MyTable.tbl* file. It resides in the *C:\Users\Public\Public Documents\TestComplete 12 Samples\Desktop\Orders* folder. Some file managers can display the *Public Documents* folder as *Documents*.

   ![Orders - Untitled](image)

   We recommend that you type the fully-qualified file name into the **File name** box of the Open File dialog. Typing instead of using the mouse will help you avoid problems if the test is played back on a different operating system, or if the Open File dialog displays a different initial folder when the test is played back later.
6. After specifying the file in the **File name** box, click **Open**. The Orders application will load data from the file and display this data in the application main window.

7. Click the **Samuel Clemens** row in the list of orders.
8. On the Orders toolbar, click **Edit order**. This will call the *Order* dialog:

![Order dialog](image)

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

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

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

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

   To create a property checkpoint, you can use the **Create Checkpoint** wizard, or you can create a **Quick Checkpoint**. In this tutorial, we will demonstrate how to create Quick Checkpoints:
• Click 🎨 on the Recording toolbar and make sure that Quick Checkpoints are enabled:

![Recording toolbar](image)

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

![Highlighting properties](image)

• In the list, click **wItem**. This property provides access to individual items of tree view controls:
TestComplete will open the **Create Checkpoint** wizard that will show the list of object properties. This list includes properties provided by TestComplete as well as properties defined by the tested application. For instance, our tested application was created in C#, so the list includes properties of the appropriate .NET class. They are of the .Net category (see the **Available Properties** table).

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

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

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

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

- The **Value** column specifies the baseline data against which the checkpoint will compare the actual data during the test run. Leave the current value.
• Select the check box of the property:

![Create Property Checkpoint](image)

- **Specify one or multiple properties to check**

  - **Property**
  - **Condition**
  - **Value**
  - **Category**
  - **Select**

- **Learn more about these settings**

- **Finish**
- **Cancel**

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

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

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

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

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

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

The test contains the commands that correspond to the actions you performed on the Orders application during the recording. We call the test commands **operations**.
Below the commands, there is the **Test Visualizer** panel that displays images which TestComplete captured for operations during test recording:

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. To view the needed image closely, double-click it in the Test Visualizer panel.

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

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

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:

<table>
<thead>
<tr>
<th>Item</th>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainForm</td>
<td>MainMenu</td>
<td>Click</td>
<td>&quot;File</td>
</tr>
<tr>
<td>MainForm</td>
<td>dgOpen</td>
<td>OpenFile</td>
<td>&quot;Variables.Var '1', 'Table (*.txt)&quot;&quot;</td>
</tr>
</tbody>
</table>
| MainForm   | dgOpen        | variables                  | "Variables.Var '1', 'Table (*.txt)""

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

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

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

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

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

7. Run the Recorded Test

Now we can run our simple test to see how TestComplete simulates user actions.
**Before running a recorded test, make sure it starts with the same initial conditions as the recording did.**

For instance, the test almost always requires the tested application to be running. So, before simulating the user actions, you should launch the application. In our case, to launch our tested application, we use the *Run TestedApp* operation at the beginning of the test, so the test will launch it for us. Alternatively, you can run the tested application manually from TestComplete IDE.

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

![TestComplete toolbar](image)

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

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

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

Some notes about the test run:

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

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

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

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

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

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

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

![Project Explorer screenshot]

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

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

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

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). You can click View Comparison Result to see the difference between the images. This simplifies the search for errors that may occur in your test. For more information, see topics of the Test Visualizer section in TestComplete Help.
The **Call Stack** pane displays the hierarchy of test calls that led to posting the selected message to the log.

The **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 “Toolbar button 5 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 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 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

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

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

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

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

About the Tested Web Page

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

http://services.smartbear.com/samples/TestComplete12/smartstore/
This application is a sample online store where you can browse items, add them to a wish list and to a shopping cart, compare items. You can create an account, login, and create mock-up orders:

![Website screenshot](image)

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

### 1. Prepare Your Web Browser

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

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

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

- **Test purpose**: The test must check whether the selected item is added to the Shopping Cart correctly. That is, the Shopping Cart shows the added item.

- **Testing steps**: The test must add an item to the Shopping Cart, and then verify that the Shopping Cart contains this item. We will record a test simulating user actions over the application.

- **Checking and logging the test result**: If the item has been added correctly, it must be in the Shopping Cart. To check this, the test will compare the data the Shopping Cart shows with the expected value. We will add a special comparison command to the test for this purpose. This command will post the comparison results to the test log, so we will see whether the verification has failed or passed.

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

3. About Test Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually** - You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.

- **Record tests** - Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them to script lines or keyword test operations. You create a test visually, and, in one sense, you record the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

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

The recording includes three steps:

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

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

   TestComplete will switch to the recording mode and display the Recording toolbar on the screen. By default, the toolbar is collapsed and shows only the most commonly used commands you may need during the recording:
You can click the ▶ arrow button to expand the Recording toolbar and view all its buttons:

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

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

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

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

4. Start Test Recording

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

2. Switch to the TestComplete Start page. If the page is hidden, select Start Page from the TestComplete Help menu.
3. On the Start page, click **Record Test**. TestComplete will show the **Record Test** wizard:

4. On the first page of the wizard, you can specify the project name, location and test type:
   - In the **Project name** text box, enter Orders.
   - Leave the **Location** to its default value.
   - Select a test type. You can create either a keyword test, or a JavaScript or Python script test:
     - A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.
     - JavaScript and Python tests are script functions with instructions simulating user actions. In this tutorial we will show how to create a keyword test. Click **Keyword**.
   - Click **Next** to continue.

We will continue working with the wizard to add our tested application to the project.
5. Define a Tested Web Page

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

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

3. Select one of the supported web browsers:
   - Internet Explorer
   - Firefox
   - Google Chrome
   - Microsoft Edge

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

4. In the URL text box, enter the following URL:
   
   `http://services.smartbear.com/samples/TestComplete12/smartstore/`

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

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

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

## 6. Record a Test

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

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

1. Wait until the selected web browser starts and navigates to the main page of the SmartStore application:
If the browser does not, you can launch it and open the needed web page manually.

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

2. Click in the **Search** text box and type *ball*:
3. In the search results, find the **Official game ball** item and click it:
4. The application will open the selected item page:

5. On the page, select the item color: **blue** and the item size: **3**:
6. Click **Add to cart**:
7. The application will show you your Shopping Cart with the item added to it:

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

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

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

- Click on the Recording toolbar and make sure that Quick Checkpoints are enabled:
- Move the mouse pointer to the **Color: Blue, Size: 3** line. TestComplete will highlight it with a red frame. Wait until the icon becomes and move the mouse pointer to the icon. TestComplete will show the most commonly used properties for which you can create Quick Checkpoints:
In the list, select the `contentText` property:

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

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

- You can continue recording user actions.
9. Click the **Delete** button next to the item to clear the Shopping Cart:

![Image of shopping cart with delete button]

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

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

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

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

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

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

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

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

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

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

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

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

After that, there is an operation that clears the Shopping Cart:
And, an operation that closes the web browser:

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

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

We have recorded user actions for a single browser. Therefore, we have group nodes for the browser. They contain all the actions that you performed on the browser window and controls. The actions that you performed on various web pages are organized into “page” group nodes:
You may want to pay attention to the name of the web browser, its web pages and page elements in the recorded tests:

They differ from the names you can see in the **Object Browser** panel. For instance, in the Object Browser, the name of the web browser is `Browser("iexplore")`, `Browser("edge")`, `Browser("firefox")` or `Browser("chrome")` (depending on which browser you use), while in the test, the web browser is simply called a browser. Another example - the main page of the SmartStore application. In the Object Browser it is called `Page("http://services.smartbear.com/samples/TestComplete12/smartstore/")`, while in the test its name is much shorter: `pageShop`.

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

Using Name Mapping and aliases makes tests easier to understand and more stable.
8. Refine the Recorded Test

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

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

![TestComplete recorded test operations](image)

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

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

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

Let’s modify the recorded test so that it simulates all click actions in the top left corner of the appropriate controls. This way, the test will simulate clicks correctly regardless of the browser window size and the application’s current control layout:
1. In your test, find the operation that simulates a click in the **Search** text box and click the ellipsis button in the **Value** column:

![Operation Parameters dialog](image1)

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

![Operation Parameters dialog](image2)
3. In the dialog, replace the recorded ClientX and ClientY parameters with 1. For TestComplete, this corresponds to clicking at the top left corner of the control:

Click OK to apply the changes and close the dialog.

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

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

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

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

### 9. Run the Recorded Test

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

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

![Run Test](image)

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

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

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

Some notes about the test run:

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

  ![Indicator](image)

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

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

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

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

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

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

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

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

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

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

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

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

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

The Picture panel shows the images of the expected and the actual application state when executing the selected test command: the “Expected” image was captured for the command during the test recording, the “Actual” image was captured during the test run. You can click View Comparison Results to compare the images and easily see the difference. This simplifies the search for errors that may occur in your test. For more information, see the topics of the Test Visualizer section.

The Call Stack pane displays the hierarchy of test calls that led to posting the selected message to the log. As we have run only one test that has not called other tests, the panel is empty.

The Performance Counters pane displays values of the performance counters monitored during the test run. The values are shown in the form of graphs. As we have not configured any performance counters, the panel is empty, as well.

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 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 that novice users face is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance, if the tested application had been running before you recorded 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.
11. Run the Test in Multiple Browsers

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

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

1. Open the test in the Keyword Test editor.
2. Right-click the Run Browser operation and select Make Browser Loop from the context menu.
3. In the ensuing operation parameters dialog, select Iterate Through All Browsers and click Finish.

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

Here is how the resulting test should look like:

<table>
<thead>
<tr>
<th>Item</th>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Loop</td>
<td>All Installed browsers</td>
<td><a href="http://services.smartbear.com">http://services.smartbear.com</a></td>
<td>Iterates through the s...</td>
</tr>
<tr>
<td>browser</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pageServicesSmartbearComSamplesT</td>
<td>Click</td>
<td>...</td>
<td>Clicks at point (149, 25)...</td>
</tr>
<tr>
<td>textboxInTextBox</td>
<td>SetText</td>
<td>&quot;ball&quot;</td>
<td>Sets the text &quot;ball&quot; in the textbox</td>
</tr>
<tr>
<td>textnode</td>
<td></td>
<td>...</td>
<td>Clicks at point (75, 2) o...</td>
</tr>
<tr>
<td>pageServicesSmartbearComSamplesT3</td>
<td>Wait</td>
<td></td>
<td>Waits until the browser...</td>
</tr>
<tr>
<td>label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>radioButton3</td>
<td>Click</td>
<td></td>
<td>Clicks the radioButton...</td>
</tr>
<tr>
<td>buttonAddToCart</td>
<td>ClickButton</td>
<td></td>
<td>Clicks the buttonAddToCart...</td>
</tr>
<tr>
<td>Property Checkpoint</td>
<td></td>
<td></td>
<td>Checks whether the 'c...</td>
</tr>
</tbody>
</table>

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

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

Now run the resulting test.

TestComplete will repeat the test operations several times. Each time, the test actions will be performed in a different browser.
The test log contains information about which browser was used and the results of the test operations performed in each browser:

![Test Log](image)

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

TestComplete supports Android mobile applications running on physical Android devices, virtual machines and emulators.

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

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

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.

To get the application, download the TestComplete Samples installation package from the https://support.smartbear.com/downloads/testcomplete page of our website and run it. The sample will be installed to the <TestComplete Samples>\Mobile\Android\Orders\ folder.

This application lets you manage a table of purchase orders: you can view the list of existing orders, modify, or remove them, as well as add new orders to the list.
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 applications that are running on physical mobile devices (smartphones and tablets), emulators, and Android-x86 virtual machines.

For the full list of supported and non-supported devices, see the TestComplete documentation.

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 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) ver. 6 - 8 and Android SDK on your test computer.
Notes:

- Testing Android applications by using Java SE Runtime Environment (JRE) or Java SE Development Kit (JDK) ver. 9 or 10 is not supported currently.

- If you already have the Android SDK and JDK (or JRE), install the needed platform and build tools via the SDK Manager or the sdkmanager utility as described below in the To get Android Studio and the To get SDK only sections correspondingly.

1. Download the JDK from:

   For testing, it is enough to have JRE installed, but you may need to download and install JDK, as JDK includes JRE modules.

2. Run the downloaded installation and follow the instructions in the wizard.

3. Download the Android SDK from:

   You can download it as part of Android Studio (the default download) or you can download the SDK only.

To get Android Studio

- On the Download page, click Download Android Studio.

- Run the downloaded installation and follow the wizard instructions.

- After the studio is installed, launch it and select Configure > SDK Manager.

- In the SDK Manager, make sure that the following items are selected:
  - Android SDK Platform-tools
  - Android SDK Build-tools
  - Google USB Driver
• Click **OK** to install packages if needed. Accept the license agreement and proceed with the installation.

**To get the SDK only**

• On the **Download** page, scroll to the **Get just the command line tools** section and download the `sdk-tools-windows-NNN.zip` package.

• Extract the files and locate the `sdkmanager` utility.

• Use the utility to install the following items:
  
  - Android SDK Platform-tools
  
  - Android SDK Build-tools
  
  - Google USB Driver

To do this, run the utility with the following command line parameters:

```bash
sdkmanager "platform-tools" "build-tools;NN.N.N" "extras;google;usb_driver"
```

Where `NN` is the version of the API for which you want to install the tools.
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:


Installing and Configuring TestComplete Plugins

To be able to test Android applications, you must have a license for TestComplete Mobile module. This module includes the required plugins.

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 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 when 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.

   Note: If you have a Xiaomi device, you may also need to enable the Install via USB option. To enable this option, you must have an active Mi account and a SIM card must be installed in your device.

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:
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. Plan the Test

Our sample Orders application works with a list of orders. Suppose you need to test whether the application Edit Order page functions properly and modifies data in the order list. In this case, you need to define the following:

- **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.

3. About Test Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually** - You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.

- **Record tests** - Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them
to script lines or keyword test operations. You create a test visually, and, in one sense, you **record** the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

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

The recording includes three steps:

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

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

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

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

   ![Recording toolbar](image)

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

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

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

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

For instructions on recording your first test, see **4. Start Test Recording** below.
4. Start Test Recording

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

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

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

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

   In the Project name text box, enter Orders.
   Leave the default value in the Location text box.
   Select a test type. You can create either a keyword test, or a JavaScript or Python script test:
   
   - A keyword test is a series of keywords that define user actions, for example mouse clicks, text input and so on. You create keyword tests visually. No scripting background is required.
   - JavaScript and Python tests are script functions with instructions simulating user actions.

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

5. Click Next to continue.

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

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

There are several ways to add applications to your project:

- You can do this during project creation.
- You can do this at any time later, in the Project Explorer.
- (Not supported for mobile applications) TestComplete can add an application to a project automatically during test recording. The recorder is smart enough to detect the start of an application through the command line, Windows Explorer, or any other way.

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

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

2. Since the Orders application is an Android application shipped as a .apk file it falls under the Mobile application category.
   Click **Mobile**.

3. From the **Device** drop-down list, select your mobile device where you will record the test. The device must be prepared for testing and connected to TestComplete as described in *Preliminary Steps* above.
   If your device is not in the list, you can click the **Start troubleshooting** link to resolve issues preventing TestComplete from connecting to the device.

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

   ```
   C:\Users\Public\Public Documents\TestComplete 12 Samples\Mobile\Android\Orders\Orders Application\bin\Orders.apk
   ```
Note: Some file managers can display the Public Documents folders as Documents.

5. Select the **Deploy to the device on start** check box to install the application on the selected device.

6. TestComplete can access internal objects, methods and properties of an Android application only if the application is instrumented (if it is prepared for testing). The Orders application we use in this tutorial is already instrumented.

However, if you use an application that has not been prepared for testing, TestComplete will instrument it automatically upon adding it to the created project. All you need to do is to click and specify certificate data for signing your application in the resulting dialog.

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

6. **Record a Test**

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

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

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

![Recording toolbar](image)

2. TestComplete will show the **Mobile Screen** window with the screen of the connected device. 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.

3. TestComplete will automatically deploy the Orders application to the mobile device and start this application. If the application does not start, you can launch it manually by selecting it from the **Run App** menu of the Recording 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.
4. 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:

5. In the Mobile Screen window, click Samuel Clemens’s order.
6. Click the Edit button of the Orders application. This will invoke the Edit Order page:

7. 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.

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

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

   We call the comparison commands **checkpoints**. TestComplete offers various types of checkpoints that are suitable for verifying different types of data (see *Checkpoints* section in TestComplete Help). One of the most frequently used checkpoints is the **Property checkpoint**. It is used to check data of application controls. We will use this checkpoint in our tutorial.
• Click \textbf{Add Check} on the Recording toolbar:

\begin{center}
\includegraphics[width=0.6\textwidth]{checkpoint_wizard}
\end{center}

This will invoke the \textbf{Checkpoint} wizard. It will guide you through the process of checkpoint creation:

\begin{itemize}
  \item On the first page of the wizard, click \textbf{Object property}.
  \item On the next page of the wizard, click the target glyph (\includegraphics[width=0.05\textwidth]{object_property_glyph}) with the left mouse button and keep the button pressed.
\end{itemize}

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 properties suggested for checking:

This list includes properties provided by TestComplete, as well as properties defined by the tested application. You can click More Properties to view all available properties.
In the table, make sure the `ControlText` property is selected. Make sure that the `Equals` (case-sensitive) condition is selected in the `Condition` column:

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

10. Click **Stop** on the Recording toolbar to stop the recording. TestComplete will process the recorded test commands and save them to a test.
7. Analyze the Recorded Test

After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test 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. Along with mapping names, TestComplete also stores images of the mapped objects in the Name Mapping repository. This helps you understand which window or control one or another mapped object matches. For more information on mapping names, see `Name Mapping` in TestComplete Help.

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

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

Some notes about the test run:

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

  ![Indicator Image]

  The indicator displays messages informing you about simulated test actions.

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

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

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

TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the **Project Explorer** panel under the **Project Suite Logs** | **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:

![Project Explorer Screenshot]

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

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

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

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 TestComplete Help.

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

The **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 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 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.

10. Run 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).

You need to 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 click **Finish**.

![Device Loop Operation Parameters Dialog](image)

• 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:

<table>
<thead>
<tr>
<th>Item</th>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Loop</td>
<td>All connected devices</td>
<td></td>
<td>Iterates through the specified mobile device. Runs the &quot;Orders&quot; tested application.</td>
</tr>
<tr>
<td>Run TestedApp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>processOrders</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListView_list1</td>
<td>TouchItem</td>
<td>5</td>
<td>Simulates a single touch on the specified item.</td>
</tr>
<tr>
<td>Button.Edit</td>
<td>TouchButton</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EditText_cust_name</td>
<td>SetText</td>
<td>&quot;Mark Twain&quot;</td>
<td></td>
</tr>
<tr>
<td>Button.Ok</td>
<td>TouchButton</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Checkpoint</td>
<td></td>
<td></td>
<td>Checks that the 'ControlText' property is set correctly.</td>
</tr>
</tbody>
</table>

• Save the test by selecting **File > Save** from TestComplete 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Message</th>
<th>Time</th>
<th>Priority</th>
<th>Has Pic</th>
<th>Link</th>
<th>Time Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄 Error</td>
<td>The device with the name &quot;VirtualBox&quot; and index 1 has been made current.</td>
<td>12:34...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The Android application &quot;Orders&quot; has started.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.24</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The listview item 5 was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>0.33</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The button was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>1.75</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The text 'Mark Twain' was entered in the text editor.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>1.38</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The button was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.08</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The property checkpoint passed (the ControlText equals &quot;Mark Twain&quot;).</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.46</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The device with the name &quot;Nexus 7&quot; and index 1 has been made current.</td>
<td>12:34...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The Android application &quot;Orders&quot; has started.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.45</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The listview item 5 was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.57</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The button was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.66</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The text 'Mark Twain' was entered in the text editor.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.22</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The button was touched.</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.51</td>
</tr>
<tr>
<td>🔄 Error</td>
<td>The property checkpoint passed (the ControlText equals &quot;Mark Twain&quot;).</td>
<td>12:35...</td>
<td>Nor...</td>
<td></td>
<td></td>
<td>2.79</td>
</tr>
</tbody>
</table>
Testing iOS Applications

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

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

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 the following software:
  - TestComplete with an active license for the Mobile module.
  - Apple iTunes. iTunes is needed, because it contains the USB driver that TestComplete uses to connect to iOS devices. We recommend that you use the latest version of iTunes.

- One of the supported devices running iOS 9.x, 10.0 - 10.3.2, 11.0 - 11.4.

- A Mac computer with Xcode, iOS SDK, and an iOS development license to compile the sample Orders application and retrieve certificate files.
About Tested Application

In our explanations, we will use the iOS version of the Orders application.

To get the application, download the TestComplete Samples installation package from the https://support.smartbear.com/downloads/testcomplete/samples/ page of our website and run it. The sample will be installed to the <TestComplete Samples>\Mobile\iOS\Orders\ folder.

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. Prepare an iOS Device

Before you can test iOS applications with TestComplete, you need to:
• Adding Test Devices to the Application's Provisioning Profile.
• Connecting Device to TestComplete.

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 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:
  

• 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 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. Prepare iOS Certificate Files**

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 and build the application.

**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 Certificate Files

To instrument iOS applications, you also need a number of certificate files:

- a developer certificate,
- the PEM file,
- a provisioning profile.

To get these files, you can use the Certificate Exporter utility that is shipped with TestComplete. The utility is located at: "<TestComplete>/Bin/Extension/iOS/CertificateExporter.zip."

**Note:** To learn how to get these files manually, see Getting Certificate Files topic in TestComplete Help.

To get the files by using the utility:

1. Unpack the archive on a Mac computer.

   ! Do not unpack the utility on a Windows machine. This will corrupt the utility.

2. Run CertificateExporter.app. You will see a list of available provisioning profiles.

3. Select a profile from the list. The utility will show you the assigned certificates in the list below.

4. Select the certificate you want to export, click Export and specify the desired directory.

5. If a dialog appears asking you to allow the utility to access your keychain, click Allow.
3. Plan the Test

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

- **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.

4. About Recording

In TestComplete, you can create tests in two ways:

- **Create tests manually** - You enter all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests or if you have good experience in creating tests. However, creating tests manually requires a lot of time and
does not prevent you from various problems. For example, you must know the classes and names of your application objects you want to work with.

- **Record tests** - Allows you to create tests easily. You can perform some actions against the tested application once, and TestComplete will automatically recognize these actions, and then convert them to script lines or keyword test operations. You create a test visually, and, in one sense, you record the performed actions to a script or keyword test. This approach does not require much experience in creating tests.

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

The recording includes three steps:

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

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

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

   ![Recording Toolbar]

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

   ![Expanded Recording Toolbar]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5. Click **Next** to continue.

   We will continue working with the wizard to add our tested application to the project.
6. Define a Tested Application

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

There are several ways to add applications to your project:

- You can do this during project creation.
- You can do this at any time later, in the Project Explorer.
- (Not supported for Mobile applications) TestComplete can add an application to a project automatically during test recording. The recorder is smart enough to detect the start of an application through the command line, Windows Explorer, or any other way.

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

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

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

3. From the **Device** drop-down list, select your mobile device where you will record your test. The device must be prepared for testing and connected to TestComplete as described above.

   If your device is not in the list, you can click the **Start troubleshooting** link to resolve issues preventing TestComplete from connecting to the device.

4. In the **Application** box, click the ellipsis button. In the resulting **Select Tested Application** dialog, locate the Orders.ipa file. The path is the following:
5. Select the **Deploy to the device on start** check box to install the application on the selected device.

6. TestComplete can access internal objects, methods and properties of an iOS application only if the application is **instrumented** (that is, if it is prepared for testing). TestComplete will instrument your application automatically upon adding it to the created project. All you need to do is to click ![image](image1.png) and specify the files holding certificate data for signing your application in the resulting dialog:

These are certificate files you have gotten on step 2. *Prepare iOS Certificate Files* above.

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

### 7. Record a Test

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

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

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

2. TestComplete will show the **Mobile Screen** window with the screen of the connected device. 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.
3. TestComplete will automatically deploy the Orders application to the mobile device and start this application. If the application does not start, you can launch it manually by selecting it from the **Run App** menu of the Recording 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.

4. 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:
5. In the Mobile Screen window, click the **Edit** button:

![Mobile Screen window](image)

This will switch the Orders application to the edit mode.
6. Click the second order in the list made by Samuel Clemens. This will display the Edit Order panel:

7. 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.

8. Click the Save button in the Edit Order panel and confirm the changes.

9. 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.
10. Click **Done** to switch off the edit mode.

![Image of the orders list](image)

The orders list will return to normal mode.

11. Insert a comparison command into our test. It will verify that the application 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.

- Click **Add Check** on the Recording toolbar:
This will invoke the **Checkpoint** wizard. It will guide you through the process of checkpoint creation:

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

![Checkpoint Wizard](image)

- Click the target glyph (靶) 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 contains a list of properties suggested for checking:

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

• Clear the Selected check box of the property selected by default.

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 `wItem` property in the table and click its **PAR** button. Enter 1 in the **Index** column and 0 in the **Section** column. The wizard will retrieve the data to check. Select the **Selected** check box of the `wItem` property:

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

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

### 8. Analyze the Recorded Test

After you finish recording, TestComplete opens the recorded keyword test for editing and displays the test 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:

![Test Visualizer panel](image)

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 TestCompleteHelp.
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 click 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.

You have recorded user actions on one mobile device and one process. So, you 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 the test recording. Generating and assigning custom names is called name mapping. TestComplete maps the names because the default names may be difficult to understand. It may be difficult to determine which window or control corresponds to a name. Using mapped names makes the test easier to understand and more stable. Along with mapping names, TestComplete also stores images of the mapped objects in the Name Mapping repository. This helps you understand which window or control one or another mapped object matches. For more information on mapping names, see Name Mapping section in TestComplete Help.
9. Run the Recorded 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 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 screen during the test run to avoid interference with the test actions.
- During the test run, the TestComplete indicator in the top right corner of the computer screen displays information about the test operations performed.

The indicator displays messages informing you about simulated test actions.

- You can stop the execution at any time by pressing Stop on the Test Engine toolbar or in the indicator, or by selecting Test > Stop from TestComplete main menu. You can pause the test execution by clicking Pause in the indicator. During the pause, you can perform any actions needed. For instance, you can explore the test log or check test variables and objects using TestComplete Watch List or Locals panel or the Evaluate dialog (see Debugging Tests in TestComplete Help).

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

10. Analyze Test Results

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

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

The Picture panel contains images that show the 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.

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

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.

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

---

**Resolving Errors**

Your test may fail. There can be several possible reasons for this. For instance, developers could change the application 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 faces by novice users is the difference in the application state during the test creation and playback. To avoid this problem, make sure that the initial conditions of the test run correspond to those you had when creating the test. For instance:

- If the tested application had been running before you recorded the test, it must be also 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.
• If you edited any data in the application, and then saved it, you need to revert the changes, for the test run to pass successfully.

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

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

For more information about finding and fixing errors, see Handling Playback Errors and Debugging Tests sections in TestComplete Help.

11. Configure the Test to Run 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 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.

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.

• Ensure that TestComplete can find this object 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 ![button](image).

  The Selected properties on the left should look like this:

  ![Screenshot](image)

  - 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.

  ![Screenshot](image)

  - If a confirmation dialog appears, click **Yes**.
• In the Aliases tree, drag the `textfield0` object 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.
• 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` 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.

### 12. Run the 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 click **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:

<table>
<thead>
<tr>
<th>Item</th>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Loop</td>
<td>All connected</td>
<td></td>
<td>Iterates through the...</td>
</tr>
<tr>
<td></td>
<td>devices</td>
<td></td>
<td>Runs the &quot;Orders&quot; te...</td>
</tr>
<tr>
<td>Run TestedApp</td>
<td>Orders</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>navigationbar1</td>
<td>TouchItem</td>
<td>Performs a touch on...</td>
</tr>
<tr>
<td></td>
<td>tableview1</td>
<td>TouchItemXY</td>
<td>Simulates a touch at...</td>
</tr>
<tr>
<td></td>
<td>textfield0</td>
<td>Touch</td>
<td>Simulates a touch on...</td>
</tr>
<tr>
<td></td>
<td>textfield0</td>
<td>SetText</td>
<td>Enters the text &quot;Mark...&quot;</td>
</tr>
<tr>
<td></td>
<td>navigationbar0</td>
<td>TouchItem</td>
<td>Performs a touch on...</td>
</tr>
<tr>
<td></td>
<td>navigationbar1</td>
<td>TouchButton</td>
<td>Performs a touch on...</td>
</tr>
<tr>
<td></td>
<td>Property Checkpoint</td>
<td>TouchItem</td>
<td>Performs a touch on...</td>
</tr>
</tbody>
</table>

• Save the test by selecting **File > Save** from TestComplete 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Message</th>
<th>Time</th>
<th>Priority</th>
<th>Has Picture</th>
<th>Link</th>
<th>Time Diff (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>The device with the name &quot;Andrews iPad Air 2&quot; and index 1 has been made current.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>✗</td>
<td>The iOS application &quot;Orders&quot; has started.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>6.12</td>
</tr>
<tr>
<td>✗</td>
<td>The toolbar item 'Edit' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>2.96</td>
</tr>
<tr>
<td>✗</td>
<td>The 'Samuel Clemens' item in the 0 section was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>1.74</td>
</tr>
<tr>
<td>✗</td>
<td>The control was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.93</td>
</tr>
<tr>
<td>✗</td>
<td>The text 'Mark Twain' was entered in the text editor.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>1.01</td>
</tr>
<tr>
<td>✗</td>
<td>The toolbar item 'Save' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.88</td>
</tr>
<tr>
<td>✗</td>
<td>The 'Yes' button was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.29</td>
</tr>
<tr>
<td>✗</td>
<td>The toolbar item 'Orders' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>✗</td>
<td>The toolbar item 'Done' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>✓</td>
<td>The property checkpoint passed (the [item(Empty, 1)] equals &quot;Mark Twain&quot;).</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>1.18</td>
</tr>
<tr>
<td>✓</td>
<td>The device with the name &quot;John Smith's iPad Air 2&quot; and index 1 has been made current.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>✓</td>
<td>The iOS application &quot;Orders&quot; has started.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>3.39</td>
</tr>
<tr>
<td>✓</td>
<td>The toolbar item 'Edit' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>2.21</td>
</tr>
<tr>
<td>✓</td>
<td>The 'Samuel Clemens' item in the 0 section was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>2.01</td>
</tr>
<tr>
<td>✓</td>
<td>The control was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>1.32</td>
</tr>
<tr>
<td>✓</td>
<td>The text 'Mark Twain' was entered in the text editor.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.24</td>
</tr>
<tr>
<td>✓</td>
<td>The toolbar item 'Save' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.59</td>
</tr>
<tr>
<td>✓</td>
<td>The 'Yes' button was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>✓</td>
<td>The toolbar item 'Orders' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.08</td>
</tr>
<tr>
<td>✓</td>
<td>The toolbar item 'Done' was touched.</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.40</td>
</tr>
<tr>
<td>✓</td>
<td>The property checkpoint passed (the [item(Empty, 1)] equals &quot;Mark Twain&quot;).</td>
<td>15:30:...</td>
<td>Normal</td>
<td></td>
<td></td>
<td>0.16</td>
</tr>
</tbody>
</table>
Where to Go Next

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

Common

- **Recording in TestComplete**
  
  This section contains information on recording tests in TestComplete.

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

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

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

- **Handling Playback Errors**
  
  Explains how to handle errors that occur during the test run.

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

Specific for Testing Desktop Applications

- **Naming Objects**
  
  This section provides information on how TestComplete names processes, windows, and controls.

- **Simulating User Actions**
  
  This section describes how to simulate mouse clicks, keystrokes and selecting menu items with TestComplete.

- **Working With Application Objects and Controls**
  
  This section contains topics that explain how to perform specific actions over test objects and retrieve data from them.

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

**Specific for Testing Web Applications**

- *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.

- *Common Tasks for Web Testing*
  
  Describes various methods for locating web page elements.

- *Cross-Browser Testing in TestComplete*
  
  Explains how to create browser-independent tests in TestComplete.

- *Testing Mobile Web Applications Using Emulator – Tutorial*
  
  A step-by-step tutorial for creating a sample cross-browser mobile web test.

**Specific for Testing Mobile Applications**

**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

## A
- Analyzing recorded test
  - Android: 80
  - Desktop: 23
  - Web: 48
- Analyzing recorded tests: 108
- Analyzing test results
  - Desktop: 29
  - Mobile: 84, 112
  - Web: 57
- Android applications testing: 63
- Automated testing: 5

## B
- Black-box applications: 9

## C
- Checkpoints: 10, 19, 75
- Creating: 19, 75
- Creating tests
  - Desktop: 16
  - Mobile (Android): 71, 73, 99
  - Web: 39

## D
- Desktop applications testing: 11

## F
- Functional testing: 5

## I
- iOS applications testing: 89

## K
- Keyword tests: 5

## L
- Log
  - Desktop: 29
  - Jump to source: 31, 58, 85
  - Web: 57

## M
- Mapping object names: 26, 51
- Mobile applications testing: 63, 89
- Mobile Screen: 74

## N
- Name mapping: 26, 51
- NameMapping: 115
  - Correcting identification properties: 115
- Naming objects: 8

## O
- Object Browser panel: 7
- Object model: 8
- Object naming: 8
- Open Applications: 9

## P
- Panels: 7
- Preparing
  - Android application: 69
  - Android device: 68
  - iOS application: 93
  - iOS device: 90
- Preparing for mobile testing: 64, 90
- Project Explorer panel: 7
- Project items: 6
- Project suites: 6
- Projects: 6
  - Desktop: 13
  - Mobile (Android): 63
  - Mobile (iOS): 89
  - Web: 33

## R
- Recording tests
  - Desktop: 16
  - Web: 39
- Running tests
  - Batch runs: 28
  - Desktop: 26
  - Different browser: 60
<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial conditions</td>
<td>27, 55, 82</td>
</tr>
<tr>
<td>Mobile</td>
<td>82, 112</td>
</tr>
<tr>
<td>Multiple devices</td>
<td>86, 117</td>
</tr>
<tr>
<td>Pausing</td>
<td>28, 56, 83, 112</td>
</tr>
<tr>
<td>Stopping</td>
<td>28, 56, 83, 112</td>
</tr>
<tr>
<td>Web</td>
<td>55</td>
</tr>
<tr>
<td>Scripts</td>
<td>5</td>
</tr>
<tr>
<td>Simulating user actions</td>
<td>25</td>
</tr>
<tr>
<td>Stores</td>
<td>10</td>
</tr>
<tr>
<td>Support and resources</td>
<td>122</td>
</tr>
<tr>
<td>Technical support and resources</td>
<td>122</td>
</tr>
<tr>
<td>Test log</td>
<td></td>
</tr>
<tr>
<td>Desktop</td>
<td>29</td>
</tr>
<tr>
<td>Jump to source</td>
<td>31, 58, 85</td>
</tr>
<tr>
<td>Web</td>
<td>57</td>
</tr>
<tr>
<td>Test object model</td>
<td>8</td>
</tr>
<tr>
<td>Test projects</td>
<td>6</td>
</tr>
<tr>
<td>Creating desktop project</td>
<td>13</td>
</tr>
<tr>
<td>Creating mobile project (Android)</td>
<td>71</td>
</tr>
<tr>
<td>Creating mobile project (iOS)</td>
<td>89, 97</td>
</tr>
<tr>
<td>Creating web project</td>
<td>33</td>
</tr>
<tr>
<td>Test results</td>
<td></td>
</tr>
<tr>
<td>Desktop</td>
<td>29</td>
</tr>
<tr>
<td>Jump to source</td>
<td>31, 58, 85</td>
</tr>
<tr>
<td>Mobile (Android)</td>
<td>84</td>
</tr>
<tr>
<td>Mobile (iOS)</td>
<td>112</td>
</tr>
<tr>
<td>Resolving errors</td>
<td>31, 59, 86, 114</td>
</tr>
<tr>
<td>Web</td>
<td>57</td>
</tr>
<tr>
<td>Tested applications</td>
<td></td>
</tr>
<tr>
<td>Desktop</td>
<td>14</td>
</tr>
<tr>
<td>Mobile</td>
<td>71, 97</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>About automated testing</td>
<td>5</td>
</tr>
<tr>
<td>Test types</td>
<td>5</td>
</tr>
<tr>
<td>Tests</td>
<td></td>
</tr>
<tr>
<td>Adjusting</td>
<td>53</td>
</tr>
<tr>
<td>Analyzing recorded test</td>
<td>23, 48, 80, 108</td>
</tr>
<tr>
<td>Analyzing results</td>
<td>29, 57, 84, 112</td>
</tr>
<tr>
<td>Creating</td>
<td>16, 39, 71, 73, 99</td>
</tr>
<tr>
<td>Recording</td>
<td>16, 39</td>
</tr>
<tr>
<td>Running</td>
<td>26, 55, 82, 112</td>
</tr>
<tr>
<td>Test types</td>
<td>5</td>
</tr>
<tr>
<td>UI testing</td>
<td>5</td>
</tr>
<tr>
<td>User interface overview</td>
<td>7</td>
</tr>
<tr>
<td>Web applications testing</td>
<td>33</td>
</tr>
<tr>
<td>White-box applications</td>
<td>9</td>
</tr>
</tbody>
</table>