



AutomatedQA Corporation



with TestComplete™ 8

Notice to A4-size paper users: This document is delivered in 8.5x11 US letter format. It can be printed on A4 paper, but there will be wide bottom margins. Reformatting for A4 paper requires only a few simple steps. One, in Word open File | Layout and choose A4 from the Size tab. Two, go to the second page (anywhere in Contents) and press F9 to update. Three, go to the last page (anywhere in the Index) and press F9 again to reindex. Four, save under a new name for convenience.

The PDF version of this file is available at <http://www.automatedqa.com/downloads/testcomplete/>.

* Windows and the Windows logo are trademarks of the Microsoft group of companies.

Table of Contents

INTRODUCING AUTOMATED TESTING AND TESTCOMPLETE	3
<i>Automated Testing</i>	3
<i>Test Types</i>	3
<i>TestComplete Projects and Project Items.....</i>	4
<i>TestComplete User Interface</i>	5
<i>TestComplete Test Object Model.....</i>	6
<i>Checkpoints and Stores</i>	8
CREATING YOUR FIRST TEST	9
1. <i>Creating a Test Project.....</i>	10
2. <i>Defining Applications to Test</i>	10
3. <i>Planning Your Test</i>	12
4. <i>Recording a Test.....</i>	13
5. <i>Analyzing the Recorded Test</i>	24
6. <i>Running the Recorded Test.....</i>	27
7. <i>Analyzing Test Results</i>	30
WHERE TO GO NEXT.....	33
INDEX.....	34

Introducing Automated Testing and TestComplete

Automated Testing

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

So, the basic test sequence includes --

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

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

TestComplete provides special features for automating test actions, defining baseline data, running tests and logging test results. It 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 a functional test - the kind that is used most often. Functional tests check the interface between the application on one side, and the rest of the system and users on the other side. They verify that the application functions as expected.

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

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

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

TestComplete Projects and Project Items

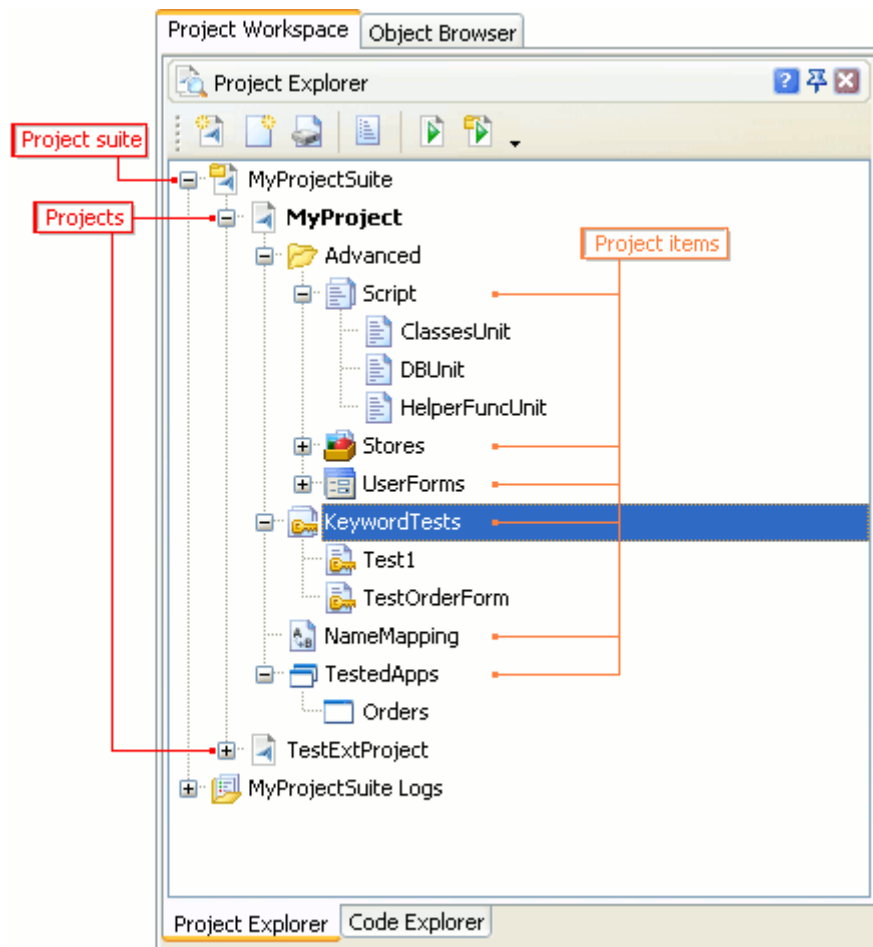
TestComplete operates with test projects and project suites. A **project** is a starting point for creating tests. It contains your tests, baseline data for checkpoints, information about tested applications and other items needed to perform testing. The project also defines the execution sequence of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

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

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

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

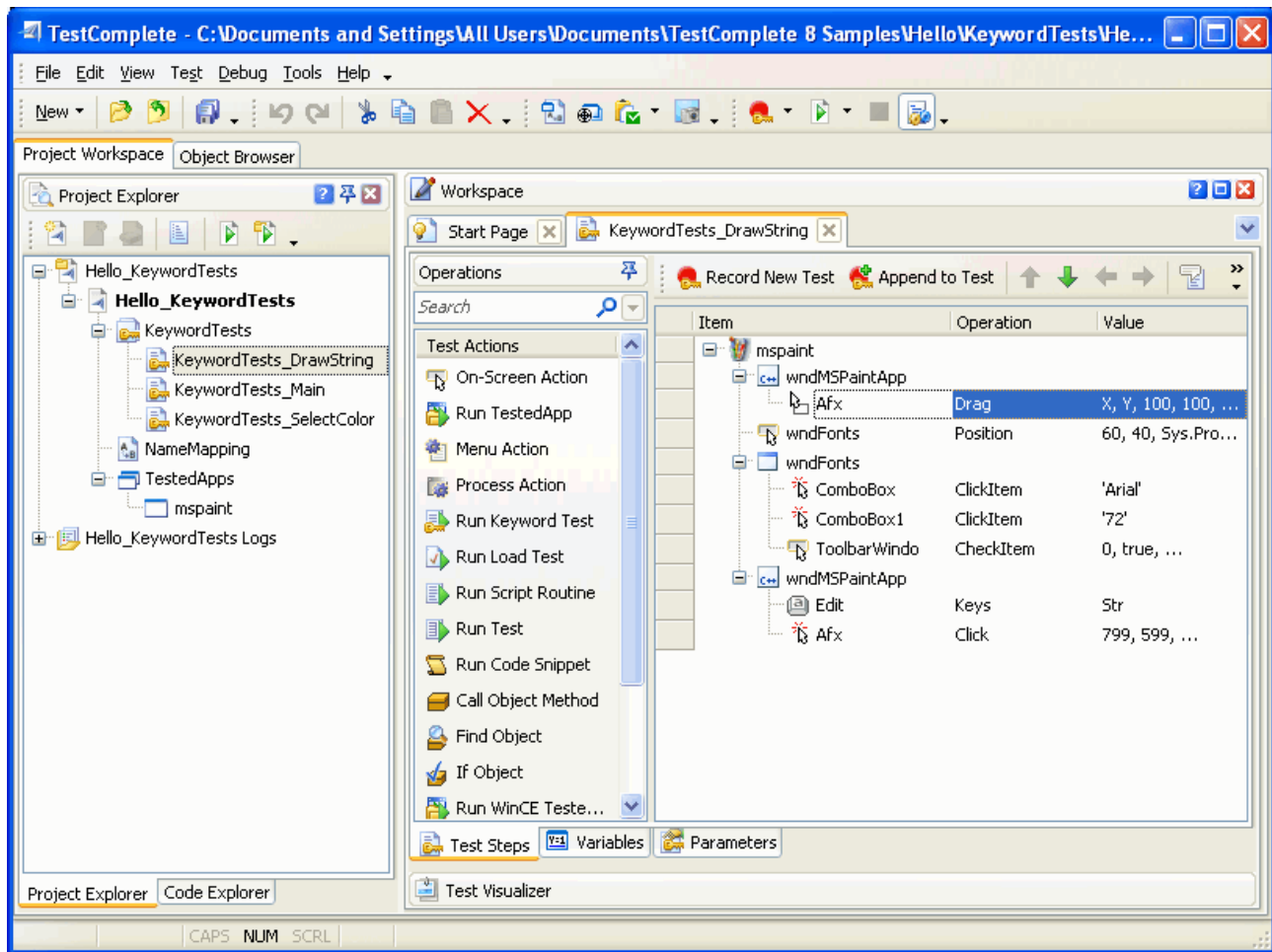
You can view and manage projects, project suites and project items in TestComplete's Project Explorer panel:



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

TestComplete User Interface

Here is a sample image of TestComplete's main window:



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

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

Besides the Project Explorer, Workspace and Test Visualizer, TestComplete contains other panels. For example, the Watch List, Locals, Breakpoints and Call Stack panels are used for test debugging. The To Do panel manages a list of tasks to be done and the Code Explorer panel provides a convenient way to explore script contents and navigate through script units.

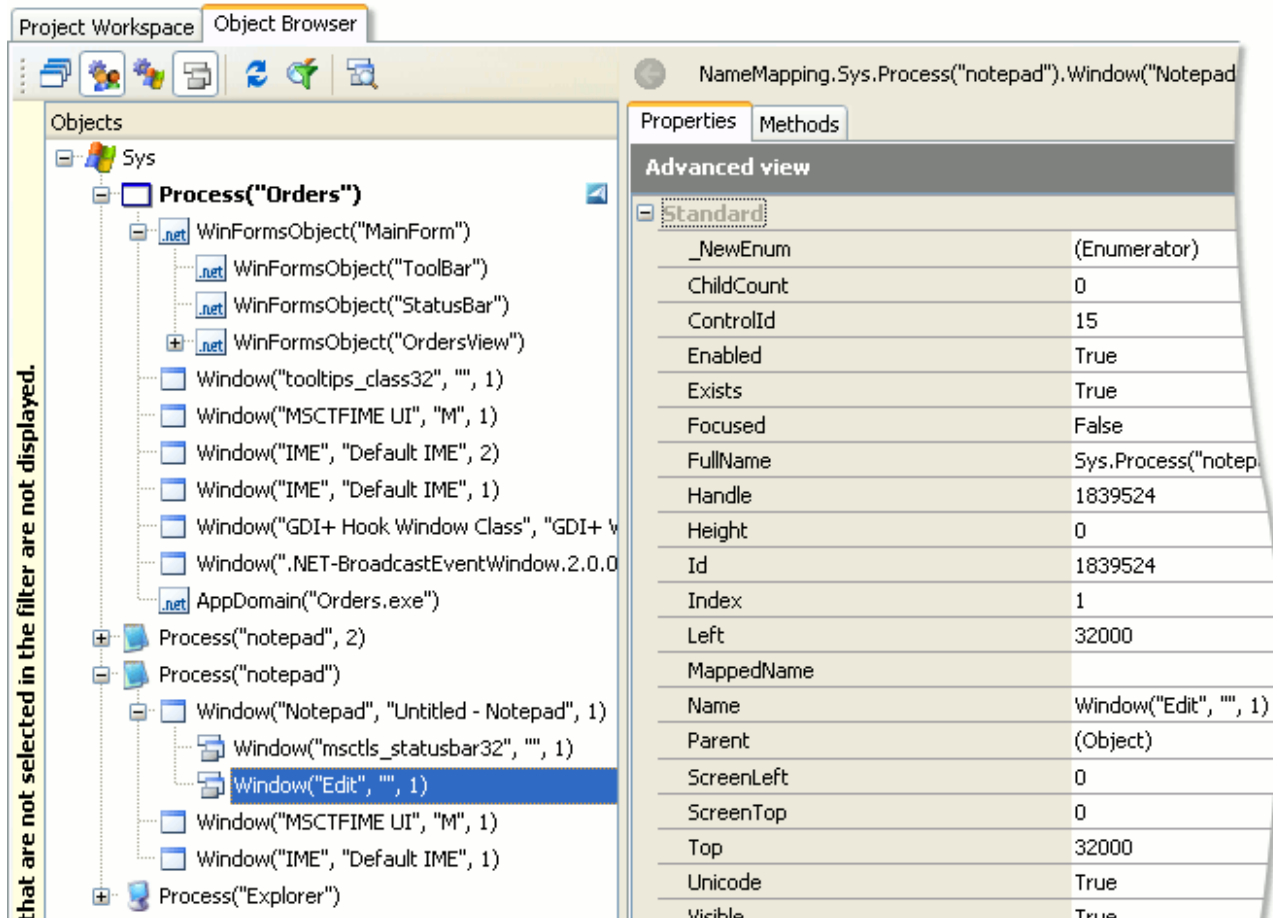
The **Object Browser** panel holds one major TestComplete function that does not belong to a specific project: it shows the list of all processes and windows that exist on the machine. For each process and window it shows methods and properties accessible externally through TestComplete facilities. In other words, the Object Browser tells you which objects, methods and properties are available for testing, and how to get to them. See *Exploring Application Properties* in TestComplete Help.

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

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

TestComplete Test Object Model

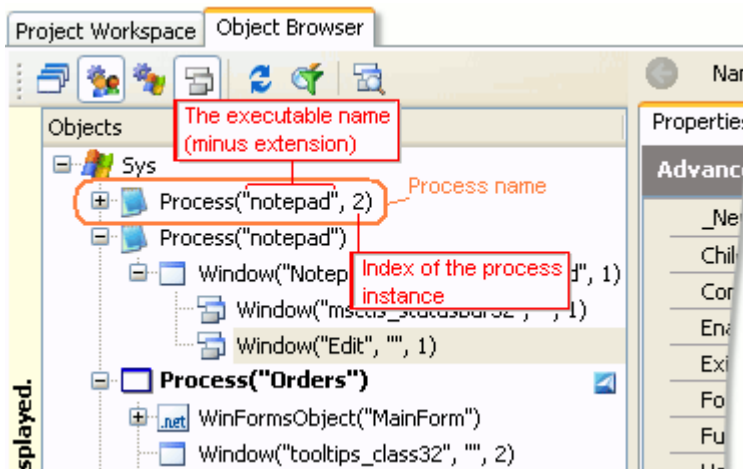
The object structure is shown in the Object Browser panel:



TestComplete uses a tree-like model for test objects. The root nodes are **Sys** for desktop applications and windows and **PDA** for programs running on Windows Mobile devices connected to your computer.

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

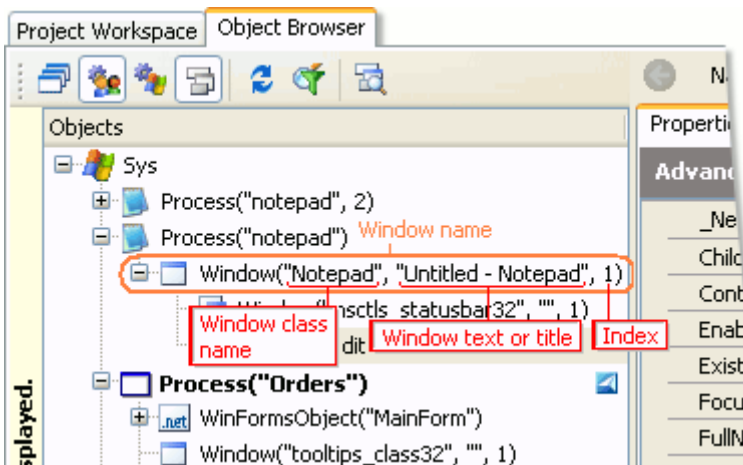
A process object's name includes the name of the process executable and its index (the index is used only if several application instances are running):




The processes have child objects – windows – that correspond to top-level windows. These objects in their turn have other child window objects that correspond to controls. The window and control names depend on whether or not the test engine has access to internal methods and properties of the application under test. TestComplete works with applications of both types, but names their windows and controls in different ways.

- Black-box applications

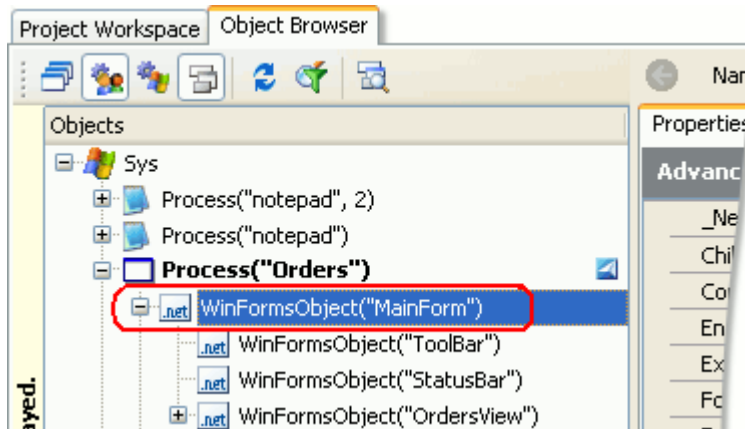
Applications that do not provide access to their internal methods and properties are called **black-box applications**. The name of each window of such applications includes the window's class name, the window's text or title (caption) and its index. Controls are named in the same manner as windows,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



- White-Box Applications

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

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



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

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

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

Checkpoints and Stores

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

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

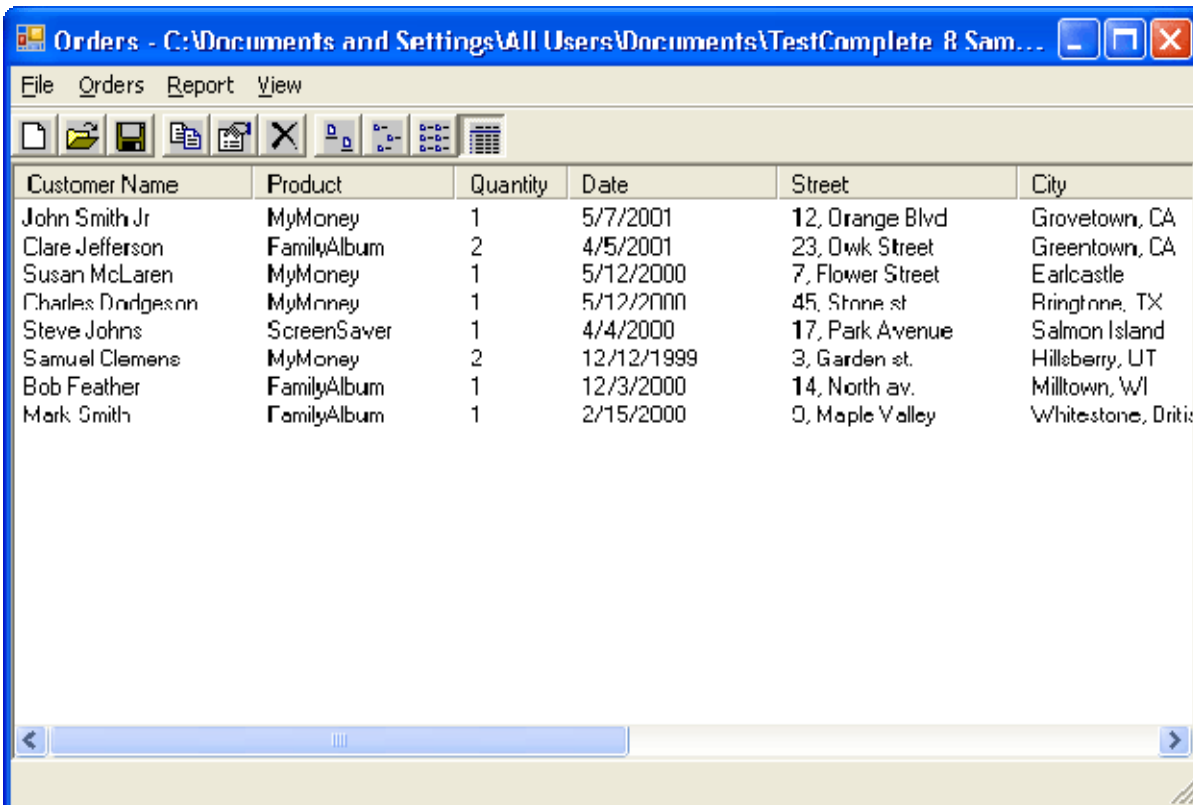
For more information on creating checkpoints and verification code, see *About Checkpoints* in TestComplete Help.

Creating Your First Test

This section provides a step-by-step tutorial that describes how to create a test project in TestComplete, record and play back a simple test, and analyze the results. The test will emulate user actions over a tested application and verify some data. Verification commands will be 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 screenshot shows a window titled "Orders - C:\Documents and Settings\All Users\Documents\TestComplete 8 Sam...". The window contains a table with the following data:

Customer Name	Product	Quantity	Date	Street	City
John Smith Jr	MyMoney	1	5/7/2001	12, Orange Blvd	Grovetown, CA
Clare Jefferson	FamilyAlbum	2	4/5/2001	23, Oak Street	Greentown, CA
Susan McLaren	MyMoney	1	5/12/2000	7, Flower Street	Earlcastle
Charles Dridgeson	MyMoney	1	5/12/2000	45, Stone st	Brington, TX
Steve Johns	ScreenSaver	1	4/4/2000	17, Park Avenue	Salmon Island
Samuel Clemens	MyMoney	2	12/12/1999	3, Garden st.	Hillsberry, UT
Bob Feather	FamilyAlbum	1	12/3/2000	14, North av.	Milltown, WI
Mark Smith	FamilyAlbum	1	2/15/2000	9, Maple Valley	Whitstone, Briti

The application is located in the following folder:

- On Windows 7, Windows Vista or Windows Server 2008:

C:\Users\Public\Documents\TestComplete 8 Samples\Open Applications

- On Windows XP, Windows Server 2003 or Windows 2000:

C:\Documents and Settings\All Users\Documents\TestComplete 8 Samples\Open Applications

Note that in Windows Explorer and in the Open and Save dialogs, the *All Users\Documents* folder may be displayed as *All Users\Shared Documents*.

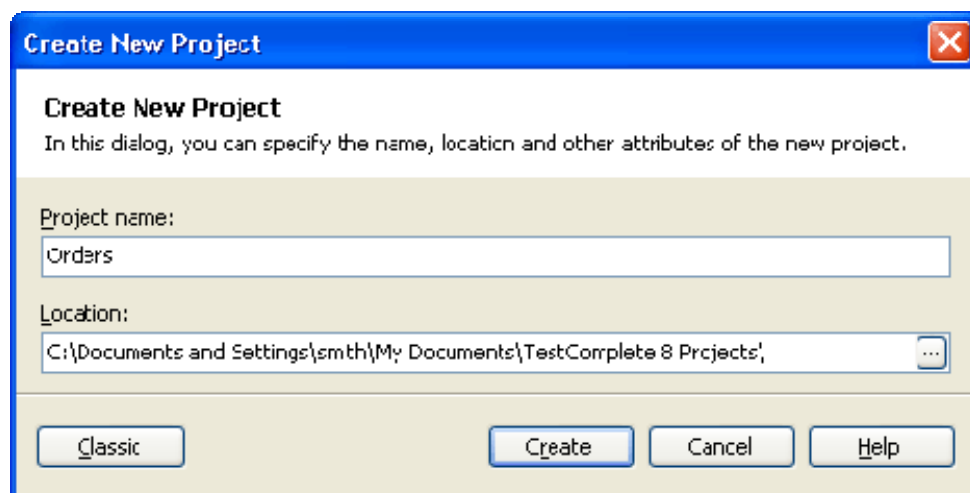
The folder stores several *Orders* projects created with different compilers: C#, Visual C++, Visual Basic, Delphi, C++Builder, Swing and so on. We will use the *Orders* application created with Visual C#.

1. Creating a Test Project

Let's create a new test project:

1. If you have a project or project suite opened in TestComplete, close it. To do this, choose **File | Close** from TestComplete's main menu.
2. Select **File | New | New Project** from TestComplete's main menu. This will call up the **Create New Project** dialog. This dialog functions in two modes: Simple and Classic. In Simple mode, which is enabled by default, the dialog contains two edit fields in which you can specify the project name and location. In Classic mode you can also specify the project suite name, scripting language and choose the project's template and project items.

This tutorial uses the Simple mode because it is more common than the Classic mode:



3. Enter *Orders* to the **Project name** edit box. TestComplete will automatically generate the project path and display it in the **Location** field. The project folder is used to store all information generated for or by the project: keyword tests, scripts, test logs, stores, and so on. You can change the project's folder in the Location box. In our example we will keep the folder name unchanged.
4. After you specified the project name and location, press **Create**. TestComplete will create a new project, *Orders.mds*, and a project suite for it. It will then display the project suite's and project's contents in the Project Explorer panel.

2. Defining Applications to Test

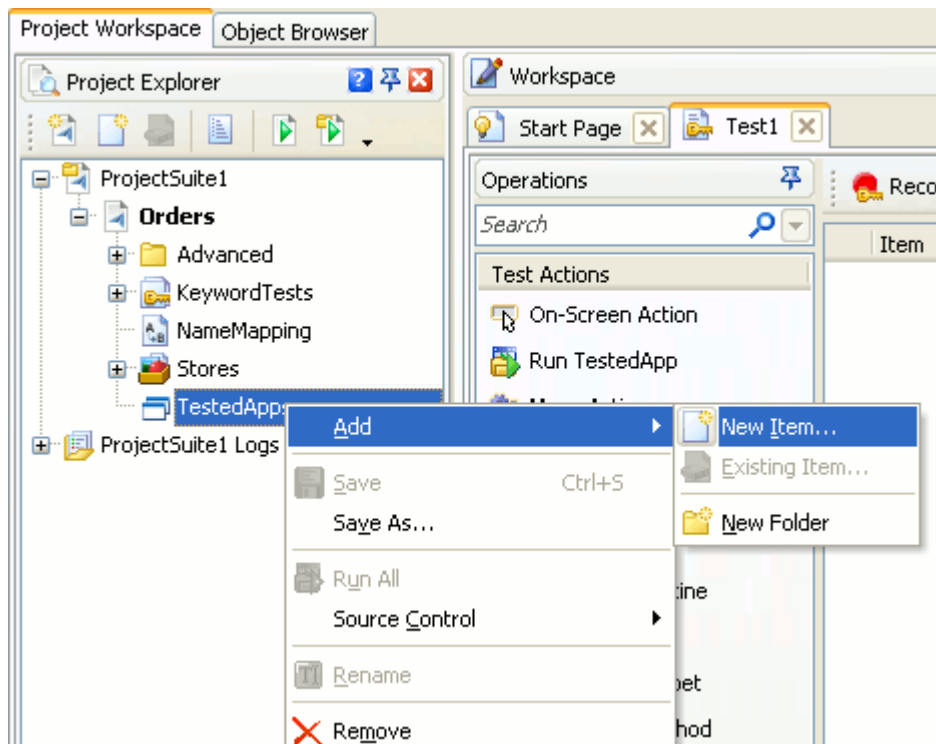
Each project may have a list of tested applications. This is a way for you to keep track of which applications the project deals with and how they are configured for testing. It also allows TestComplete to launch all applications in the list or only those applications that are enabled to be launched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

You can append applications to the list of tested applications manually, or TestComplete can do this automatically during test recording. The recorder is smart enough to detect the start of an application through the command line, Windows Explorer or any other way. After the recording is over, TestComplete will add the tested application to the list and insert the “Run Tested Application” command into the recorded test.

In this tutorial, in order to get you acquainted with the list and to demonstrate the features it provides for managing applications under test, we will add the tested application to the list manually.

Let’s add our sample application to the tested applications list:

1. Right-click the **TestedApps** node in the Project Explorer panel and choose **Add | New Item** from the context menu.



This will invoke the standard **Open File** dialog.

2. Locate *Orders.exe* using this dialog and then click **Open**.

As you may remember, we were going to use the sample Orders application created in C# and shipped with TestComplete. The path to *Orders.exe* looks like this:

- If you are working under Windows 7, Vista or Windows Server 2008:

C:\Users\Public\Documents\TestComplete 8 Samples\Open Applications\OrdersDemo\C#\bin\Debug\Orders.exe

- If you are working under Windows XP, Windows 2000 or Windows Server 2003:

C:\Documents and Settings\All Users\Documents\TestComplete 8 Samples\Open Applications\OrdersDemo\C#\bin\Debug\Orders.exe

After you added a tested application to the list, you can specify its run mode and run parameters. For more information, see *Run Modes* in TestComplete Help. In this tutorial, we will use the default settings.

3. Save your changes by selecting **File | Save** from TestComplete’s main menu.

3. Planning Your Test

General Notes on Planning Tests

It is generally considered a good practice to **plan** your test:

- **Define the test goal** and decide which application functionality should be tested. The clearer the goal and the simpler the test, the better. Large tests that deal with various aspects of the application's behavior are hard to create and maintain. It is better to create a simple test that is aimed at one objective only. Once you create several simple tests, you can always organize them into a one larger test.
- **Plan testing steps** and decide which actions the test will perform. The testing steps depend on the test purpose and the nature of the application under test. Testing steps may include actions that prepare the application for the test (that is, they put the application to some initial state). Also, testing steps may feed corresponding input to the application.
- **Plan checking actions.** Typically, after the application performed some actions, something changed in that application: data in an application window can be changed or rearranged, a new window may be created and a file can be created on or deleted from your hard disk. You should determine what criteria should be used to pass or fail a test and which checkpoints will be used to verify these criteria.
- **Log the result.** The test results can be logged in different ways, for instance, your tests can save the full test results to a file or show a message notifying you that the test run is over.

TestComplete tracks all simulated actions during the test run and saves the information on these actions to the test log. You can also post custom messages, image, files or file links to the log. The messages can be organized into folders and each message may use specific font and background settings to display. The results can be exported to a file, compressed and sent via e-mail to your co-workers. You can even create a bug report in an issue-tracking system directly from test results shown in the log. See *Test Log* in TestComplete Help for more information.

Planning a Test for the Orders Application

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

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

4. Recording a Test

Recording Tests in TestComplete

The recording includes three steps:

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

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

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



The toolbar contains items that let you perform additional actions during the recording, pause or stop recording and change the type of the recorded test (keyword test, script code, Windows or PDA low-level procedure, or HTTP traffic).

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

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


Recording Test for the Orders Application

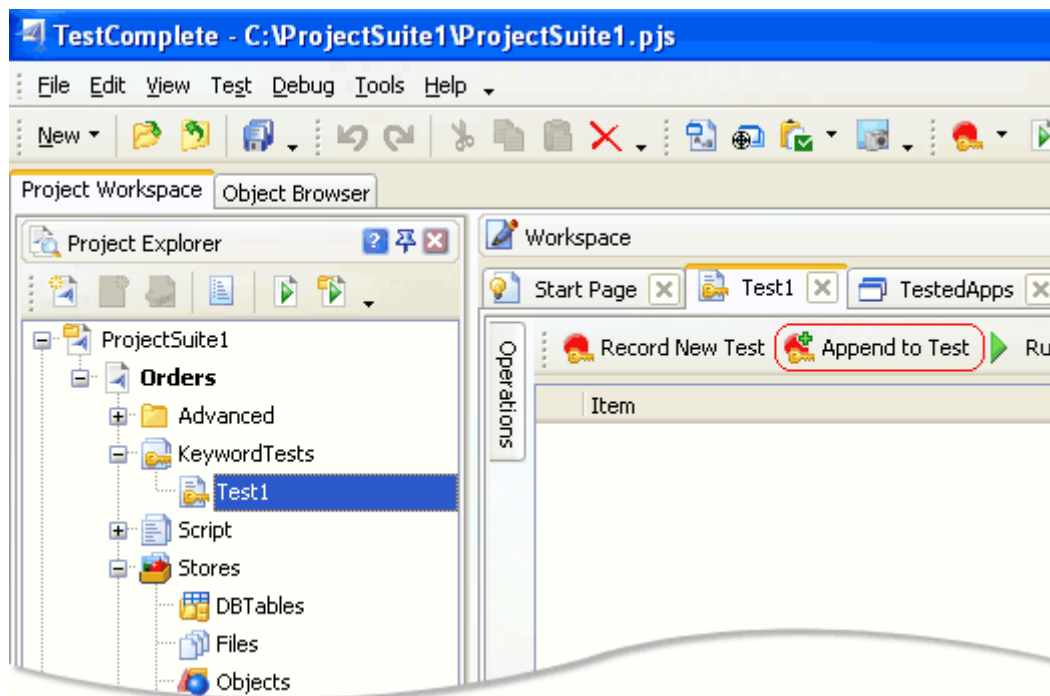
Let's now record a keyword test for the sample Orders application. The test will launch the application, load data in it, simulate clicks and keystrokes within the application's window and verify the application's data.

Note: If you are reading this document from screen, then do not switch to it during the recording. The recording engine traces and records all user actions, so the recorded test will contain commands that simulate "switching" to this document.


To see the instructions, you can print them before recording. Or, if you have two monitors, you can move this document to the other monitor.

Let's start recording:

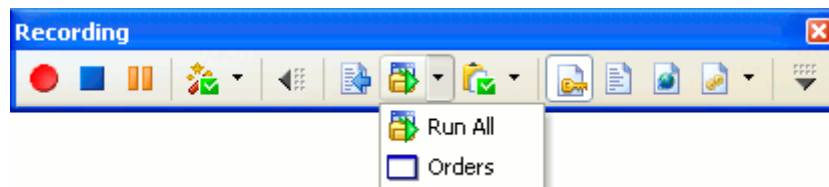
1. When creating a new project, TestComplete automatically creates an empty keyword test in this project. Let's record test commands into this test. To start recording, select the  **Append to Test** item on the test editor's toolbar:



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

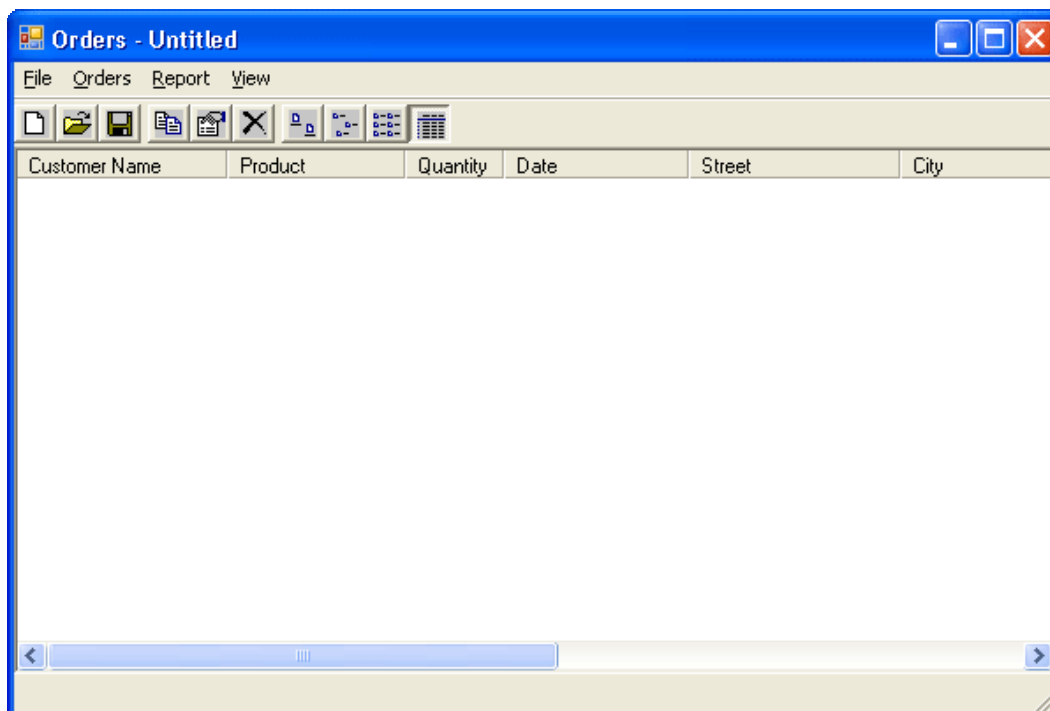
- By default, the Recording toolbar is collapsed. Click the arrow button  to expand the Recording toolbar and view all its buttons.

After that, click the down arrow of the  **Run Tested Applications** button and select Orders from the drop-down list:



TestComplete will automatically insert the application launch command into the recorded test. You will see this command later, when we will analyze the recorded test.

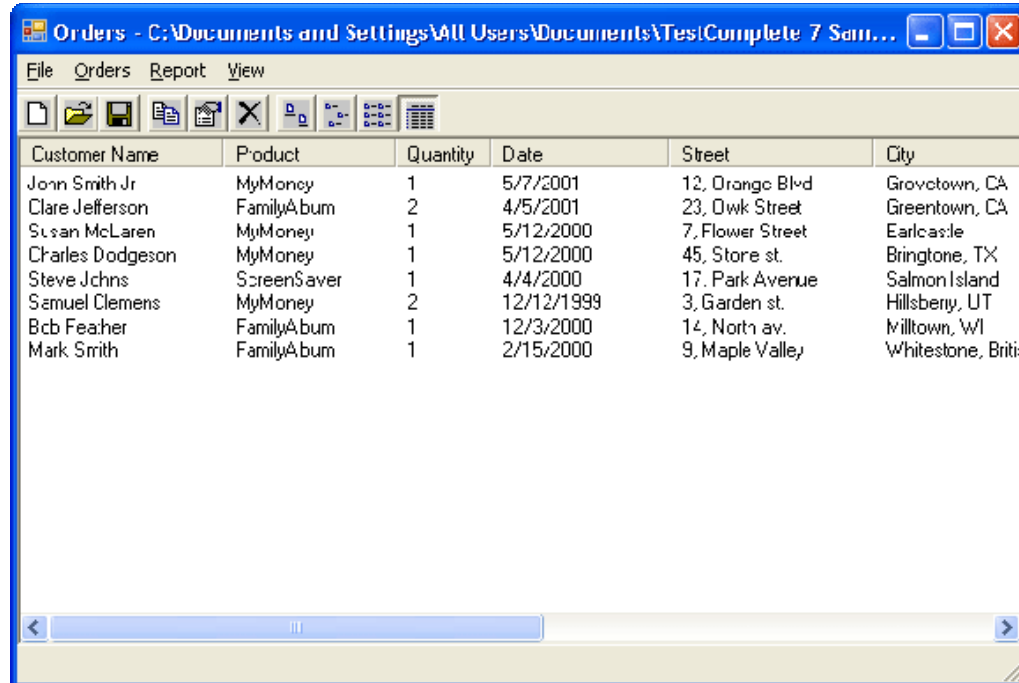
3. Wait until the application starts and the application's main window is shown:




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

4. Switch to the Orders application and select **File | Open** from its main menu. This will bring up the standard Open File dialog.
5. In the dialog, open the *MyTable.tbl* file. The location of this file depends on the operating system you use. On Windows 7, Windows Vista and Windows Server 2008 it resides in the *C:\Users\Public\Documents\TestComplete 8 Samples\Open Applications* folder. On other operating systems, it is located in the *C:\Documents and Settings\All Users\Documents\TestComplete 8 Samples\Open Applications* folder.

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



- Click the *Samuel Clemens* row in the list of orders.
- Move the mouse cursor to the Orders toolbar and press  **Edit order**. This will call the **Order** dialog:

The 'Order' dialog box contains the following fields and controls:


- Product:** MyMoney (dropdown menu)
- Quantity:** 2 (spin box)
- price per unit:** \$100 (text box)
- discount:** 0% (text box)
- Total:** 200 (text box)
- Date:** 12/12/1999 (date picker)
- Customer Name:** Samuel Clemens (text box)
- Street:** 3, Garden st. (text box)
- City:** Hillsberry, UT (text box)
- State:** US (text box)
- Zip:** (text box)
- Card:**
 - Visa
 - MasterCard
 - American Express
- Card No:** 123456789012 (text box)
- Expiration Date:** 03/02/2009 (date picker)
- Buttons:** OK, Cancel

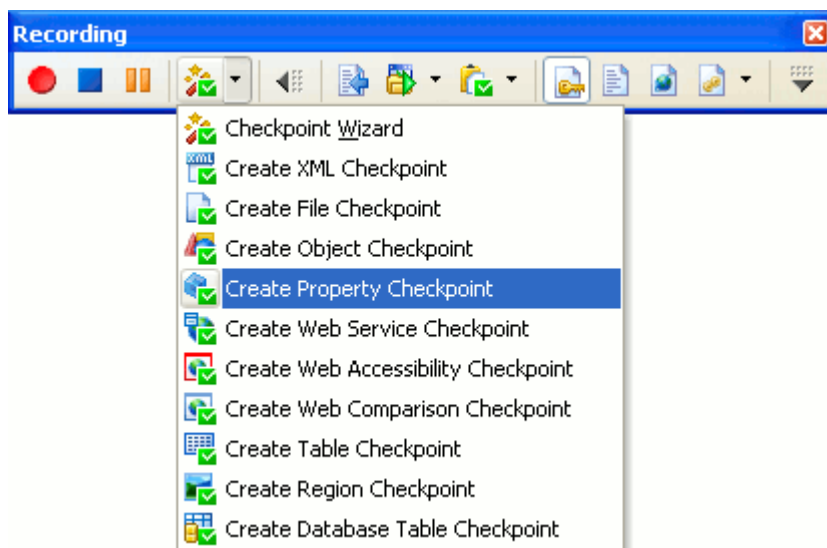
9. In the dialog, click within the **Customer Name** text box to move the insertion point there. Right-click within the Customer Name box and choose **Select All** from the context menu and then enter *Mark Twain* as the customer name.

Note: You could also choose the text in a text box by dragging the mouse, but in this case the test would become less universal, because if the text length varies, the test will have to drag for a longer or shorter distance, that is, you will have to calculate the dragging distance in your test. Using the Select All menu item shortcut frees you from this problem and makes the test more stable. A possible alternative to using this menu item is to use a shortcut (typically, CTRL+A) to choose the entire context of the text box.

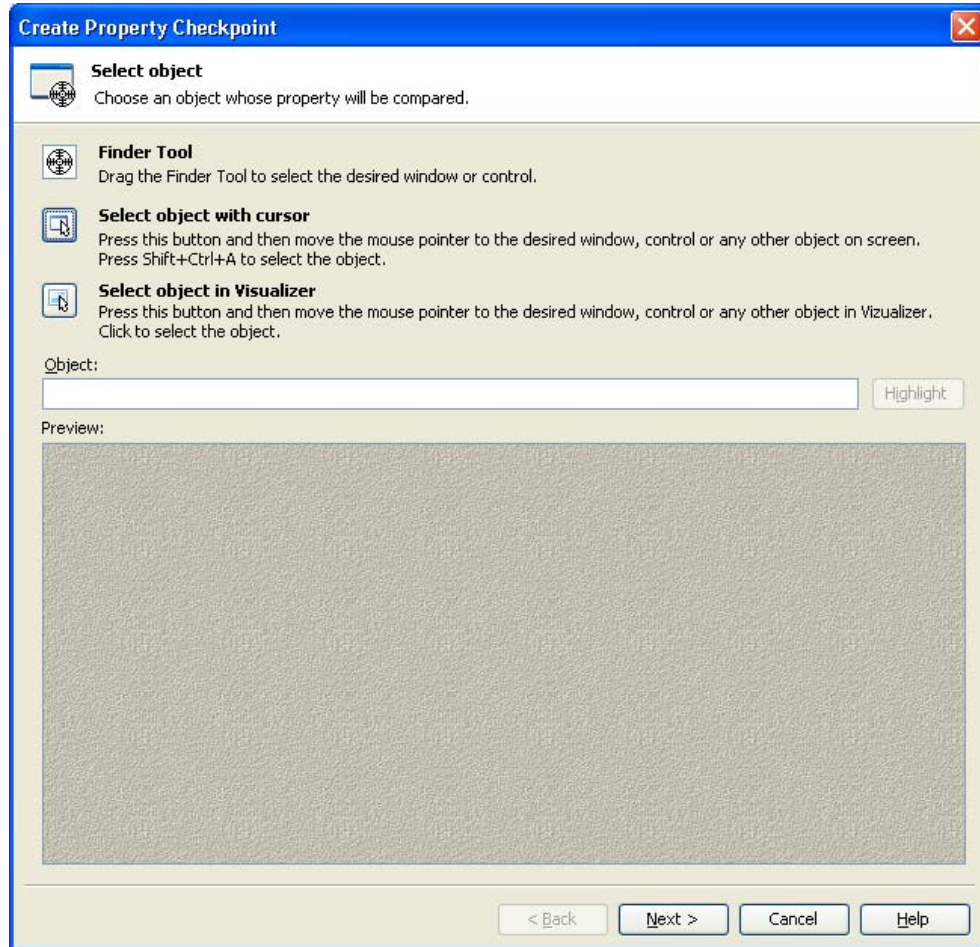
10. Click **OK** to close the dialog. TestComplete will update the customer list in the application's main window.
11. Now let's insert a comparison command into our test. It will verify that the application's customer list displays the modified name - *Mark Twain*.

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

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



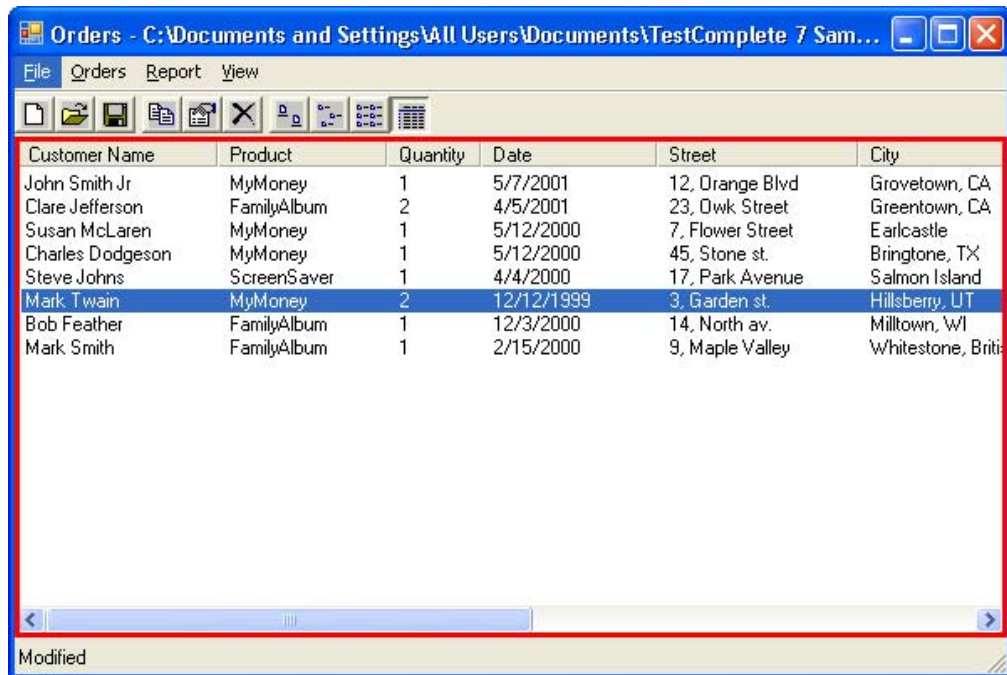
This will invoke the Create Property Checkpoint wizard. It will guide you through the process of checkpoint creation:



- On the first page of the wizard, click the Finder tool icon (🔍) with the left mouse button and keep the button depressed.

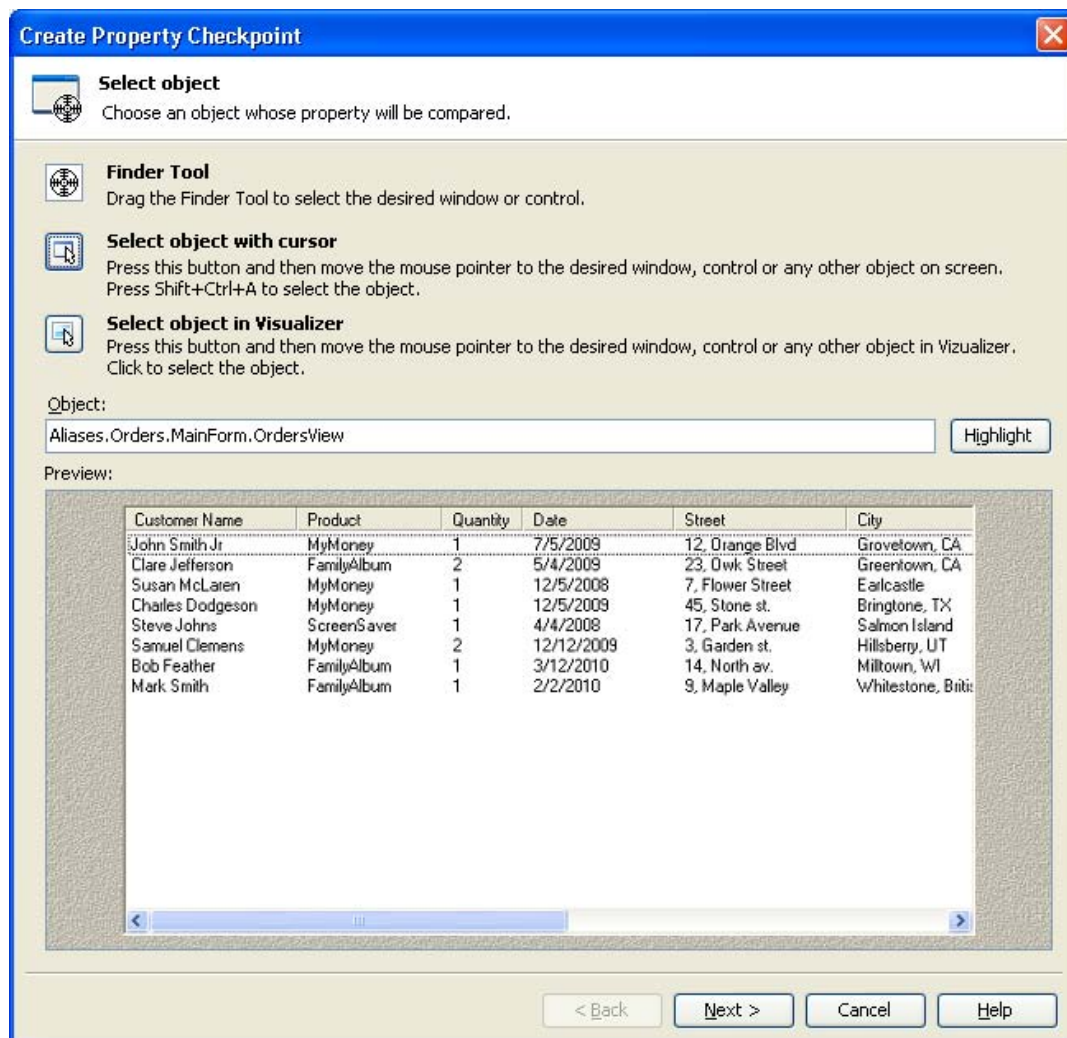
Wait until the wizard minimizes and then drag the icon to the customer list of the Orders application. While you are dragging, TestComplete will highlight the controls and windows under the mouse cursor with the red frame.

Release the mouse button when the Finder tool icon is over the customer list and it is highlighted with the red frame:



Customer Name	Product	Quantity	Date	Street	City
John Smith Jr	MyMoney	1	5/7/2001	12, Orange Blvd	Grovetown, CA
Clare Jefferson	FamilyAlbum	2	4/5/2001	23, Oak Street	Greentown, CA
Susan McLaren	MyMoney	1	5/12/2000	7, Flower Street	Earlcastle
Charles Dodgeson	MyMoney	1	5/12/2000	45, Stone st.	Bringtone, TX
Steve Johns	ScreenSaver	1	4/4/2000	17, Park Avenue	Salmon Island
Mark Twain	MyMoney	2	12/12/1999	3, Garden st.	Hillsberry, UT
Bob Feather	FamilyAlbum	1	12/3/2000	14, North av.	Milltown, WI
Mark Smith	FamilyAlbum	1	2/15/2000	9, Maple Valley	Whitstone, Briti

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

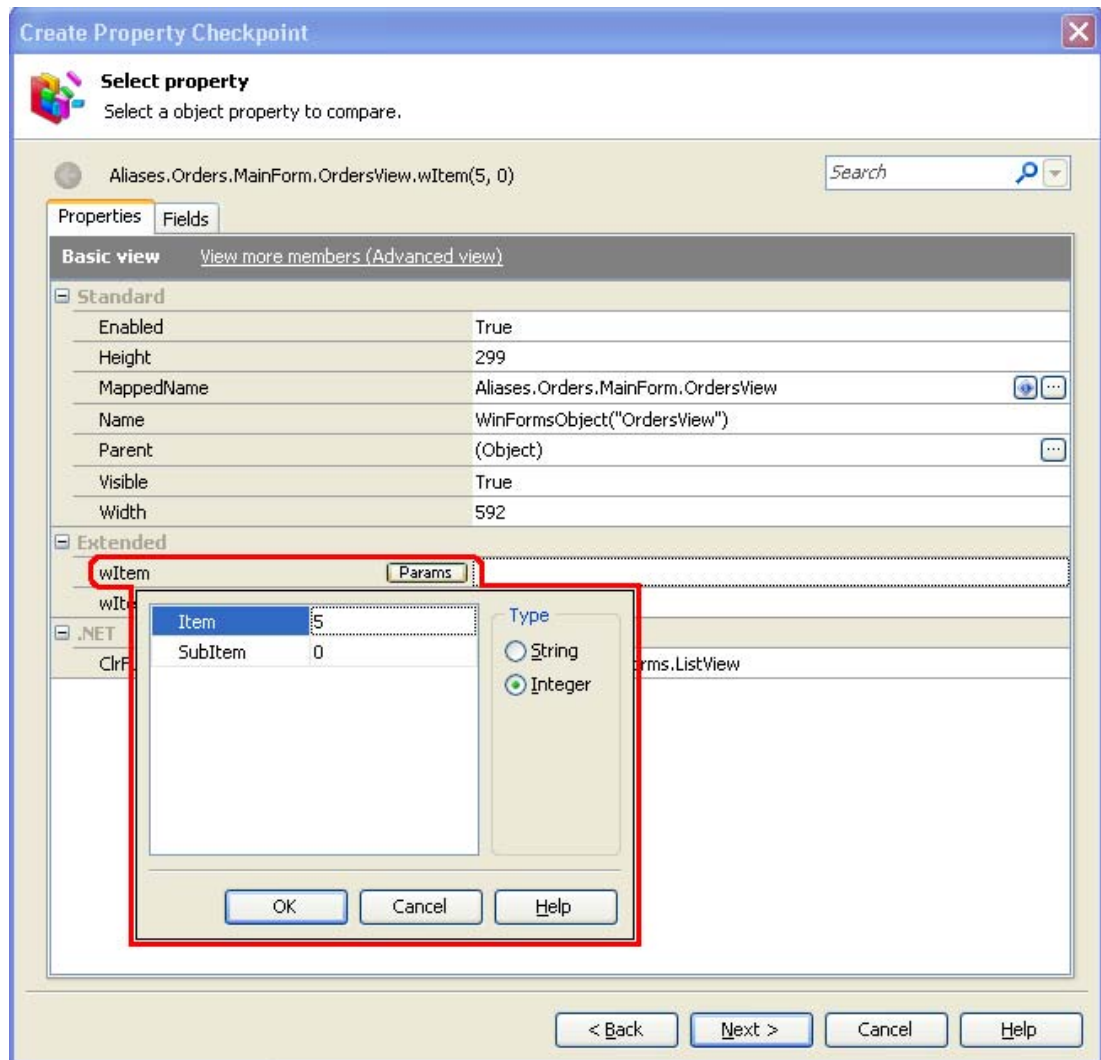


Click **Next** to continue.

- The next page of the wizard displays a list of the selected object's properties. This list includes properties provided by TestComplete as well as properties defined by the tested application. For instance, our tested application was created in C#, so the list includes properties of the appropriate .NET class. You can see them under the **.NET** node. In our example these properties are hidden, because the list contains only a basic set of properties. To view all available properties, click the **View more members (Advanced view)** link.

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

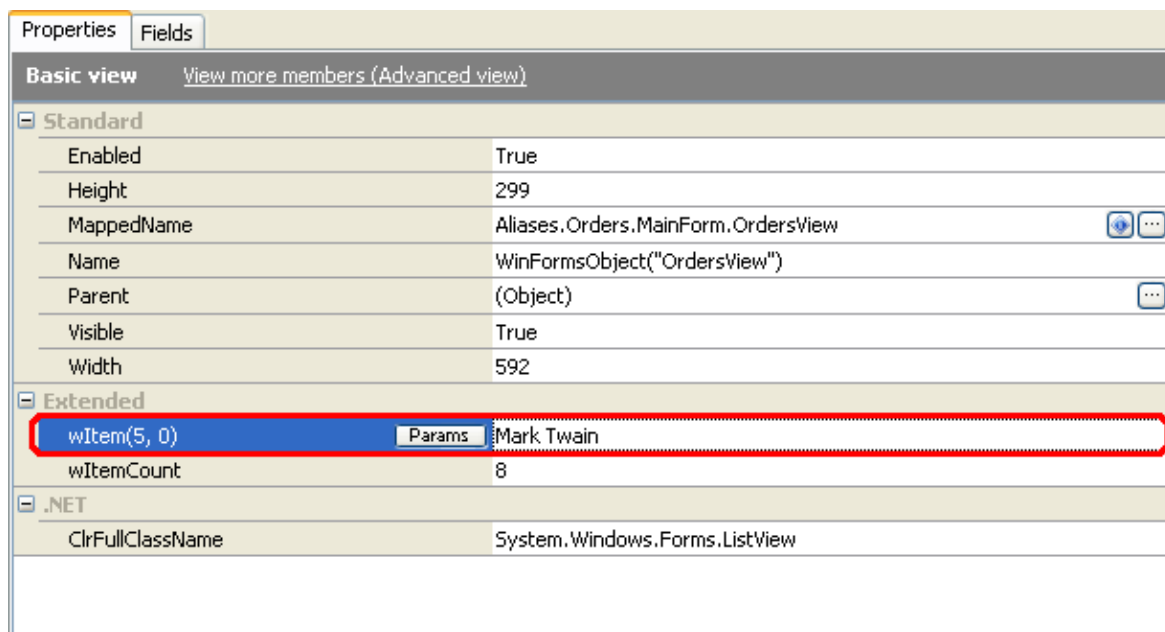
- Find the `wItem` property in the list (it is under the node **Extended**). Click its **Params** button. The following window will pop up:



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

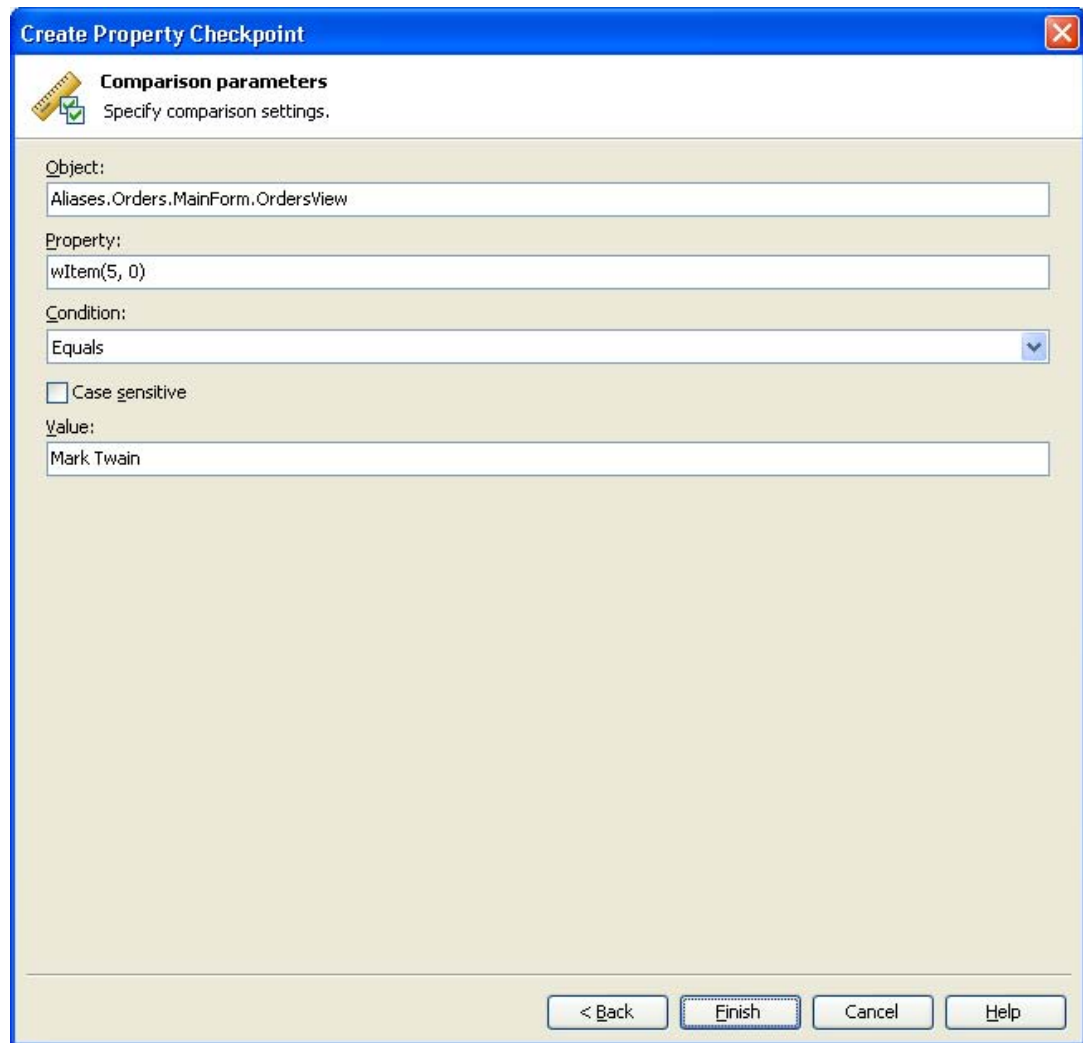
- Select **Integer** in the **Type** section
- Enter **5** into the **Item** box (5 is the index of the *Mark Twain* item in the tree view. Indexes are zero-based).
- Click **OK**.

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



Click **Next** to continue.

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

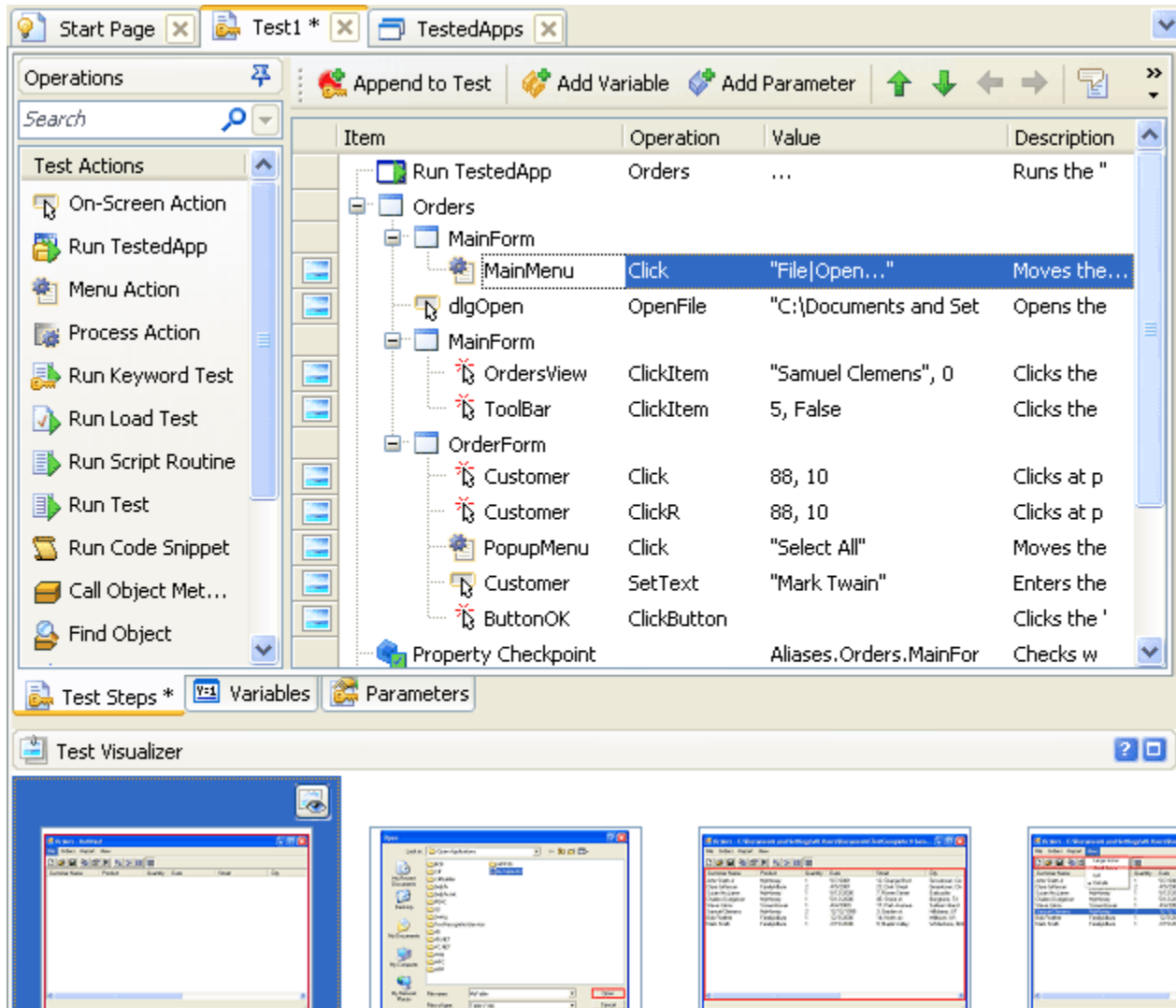


Click **Finish** to complete the checkpoint creation. TestComplete will append the checkpoint command to the recorded test.

12. Close the Orders window by clicking the X button on the window's caption bar. This will display the dialog asking if you want to save changes. Press **No**. Orders will close.
13. Press **■ Stop** on the Recording toolbar to stop the recording. TestComplete will process the recorded test commands and save them to a test.

5. Analyzing the Recorded Test

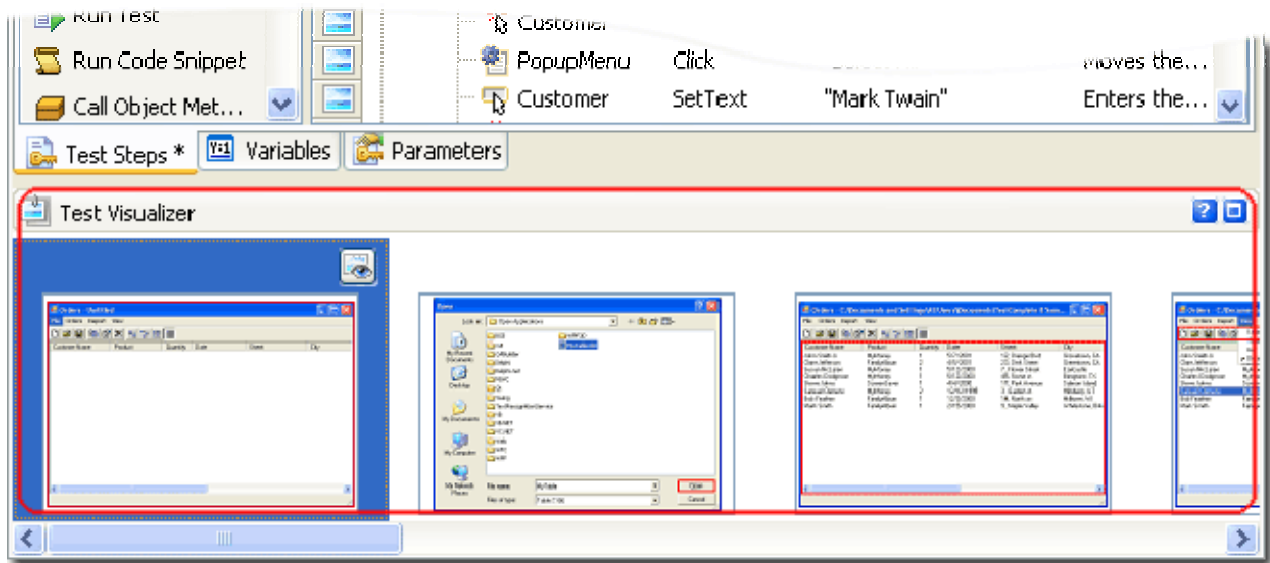
After you have finished recording, TestComplete opens the recorded keyword test for editing and displays the test's contents in the KeywordTest Editor:



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

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

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



These images illustrate the 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 in TestComplete Help.

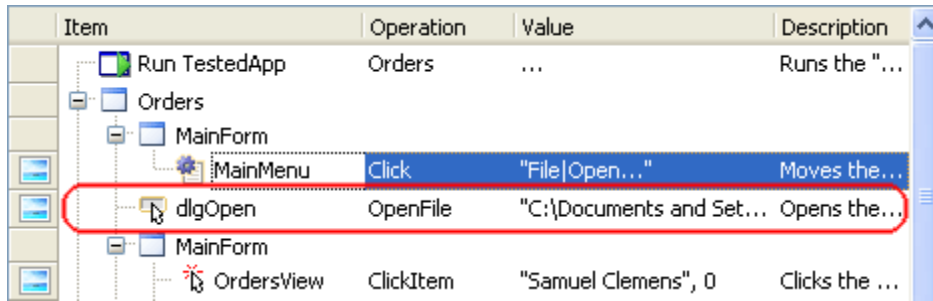
The first operation in the test is Run TestedApp. It is used to launch the tested application (in our case, it is the *Orders* application) from a keyword test. TestComplete records this operation automatically when it detects that a user launches the application from TestComplete's UI or somewhere from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

Item	Operation	Value	Description
Run TestedApp	Orders	...	Runs the "..."
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the ...
dlgOpen	OpenFile	"C:\Documents and Set..."	Opens the ...
MainForm			
OrdersView	ClickItem	"Samuel Clemens", 0	Clicks the ...

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

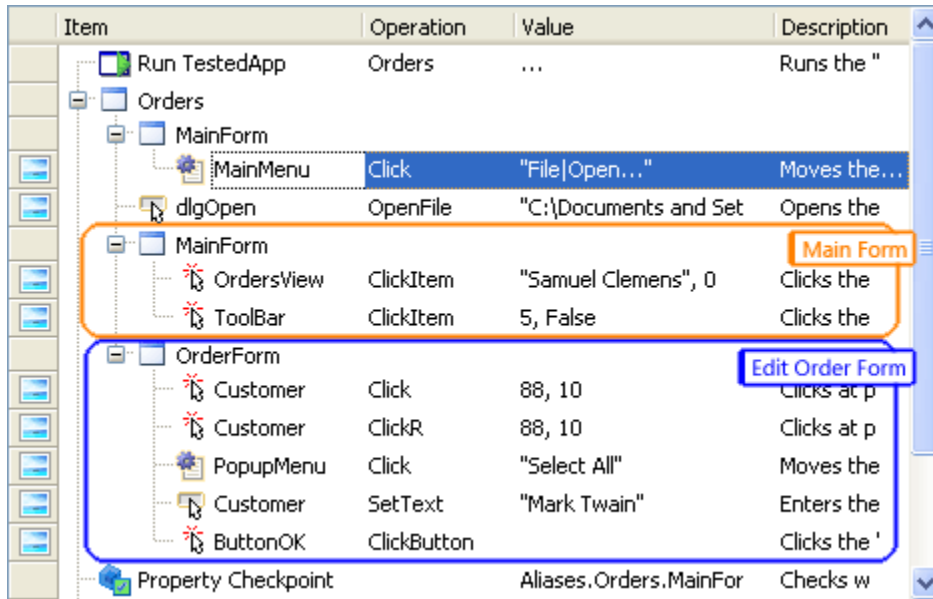
Item	Operation	Value	Description
Run TestedApp	Orders	...	Runs the "..."
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the ...
dlgOpen	OpenFile	"C:\Documents and Set..."	Opens the ...
MainForm			
OrdersView	ClickItem	"Samuel Clemens", 0	Clicks the ...

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



Item	Operation	Value	Description
Run TestedApp	Orders	...	Runs the "...
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the...
dlgOpen	OpenFile	"C:\Documents and Set..."	Opens the...
MainForm			
OrdersView	ClickItem	"Samuel Clemens", 0	Clicks the ...

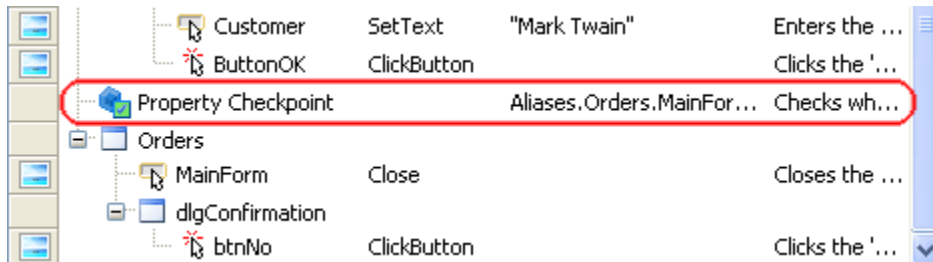
After that, there follow operations that simulate your actions with the application's main window and the Order form:



Item	Operation	Value	Description
Run TestedApp	Orders	...	Runs the "
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the...
dlgOpen	OpenFile	"C:\Documents and Set..."	Opens the
MainForm			
OrdersView	ClickItem	"Samuel Clemens", 0	Clicks the
ToolBar	ClickItem	5, False	Clicks the
OrderForm			
Customer	Click	88, 10	Clicks at p
Customer	ClickR	88, 10	Clicks at p
PopupMenu	Click	"Select All"	Moves the
Customer	SetText	"Mark Twain"	Enters the
ButtonOK	ClickButton		Clicks the '
Property Checkpoint		Aliases.Orders.MainFor	Checks w

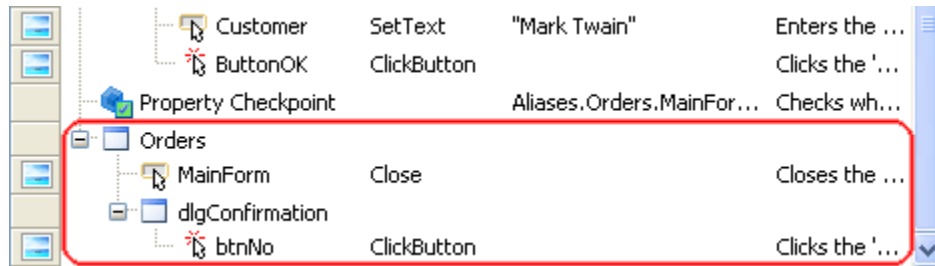
For more information on simulating mouse events, keyboard input and other actions from your scripts, see *Simulating User Actions* in TestComplete Help.

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



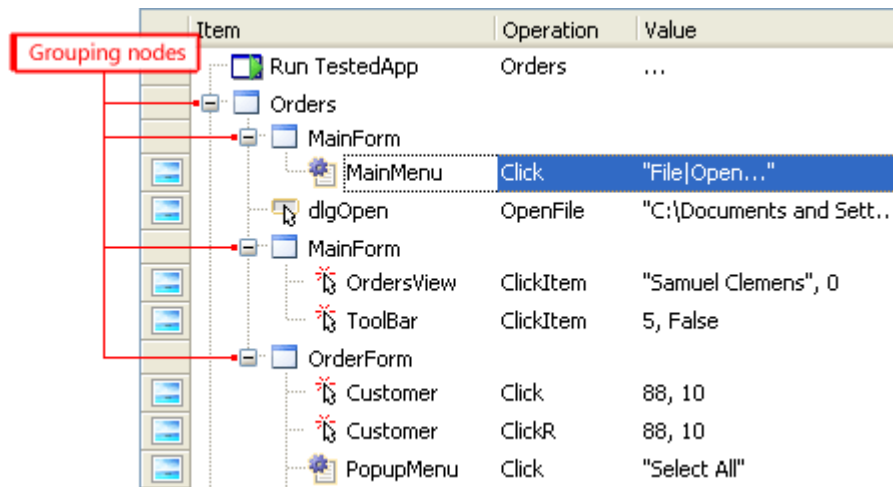
Customer	SetText	"Mark Twain"	Enters the ...
ButtonOK	ClickButton		Clicks the '...
Property Checkpoint		Aliases.Orders.MainFor...	Checks wh...
Orders			
MainForm	Close		Closes the ...
dlgConfirmation			
btnNo	ClickButton		Clicks the '...

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



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

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



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

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

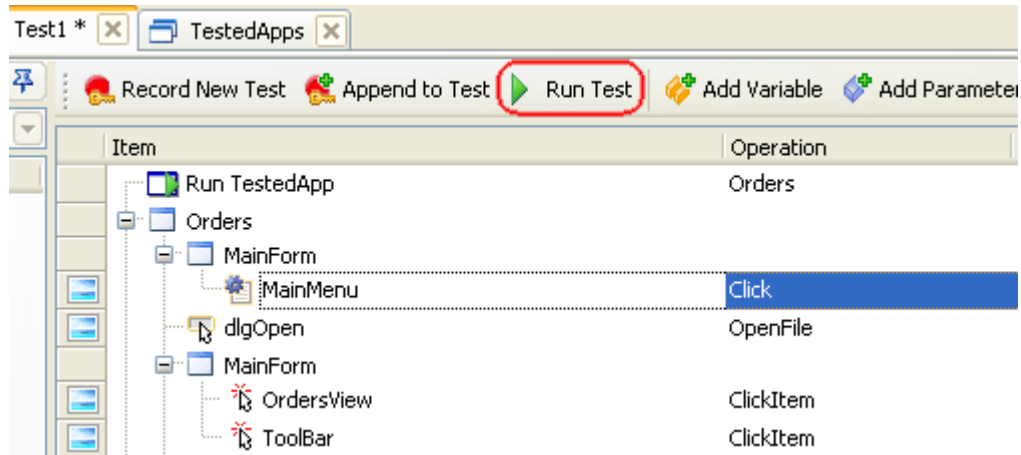
6. Running the Recorded Test

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

Before running a recorded test, make sure it starts with the same initial conditions as the recording did. For instance, the test almost always requires the tested application to be running. So, before simulating the user actions, you should launch the application. In our case, to launch our tested application, we use the

Run TestedApp operation at the beginning of the test, so the test will launch it for us. Alternatively, you can run the tested application manually from TestComplete's IDE.

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



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

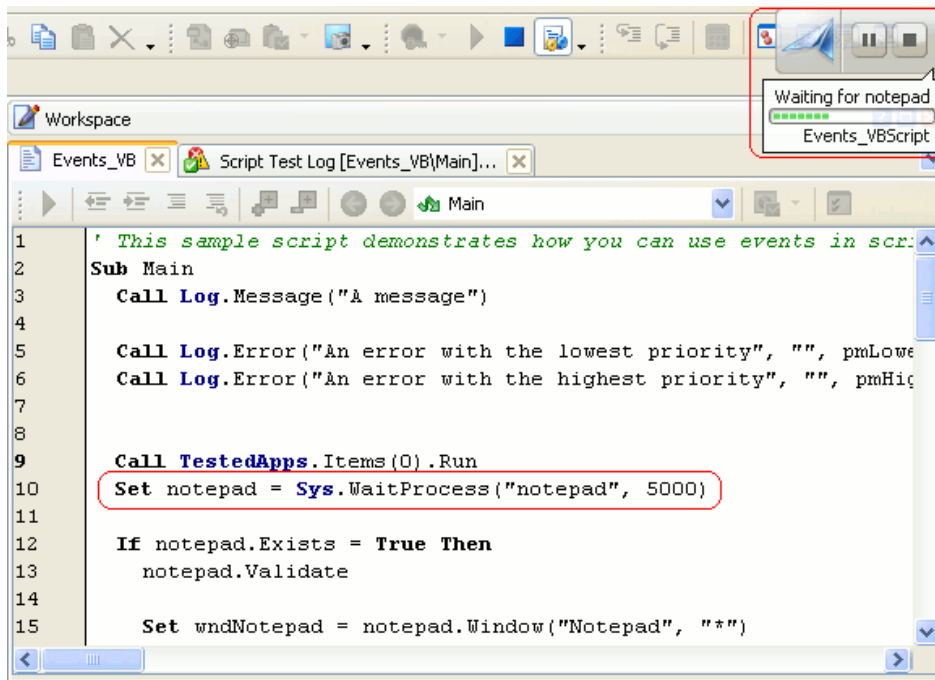
Note: Don't move the mouse or press keys during the test execution. Your actions may interfere with actions simulated by TestComplete and the test execution may go wrong.

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

Some notes about the test run:

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

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



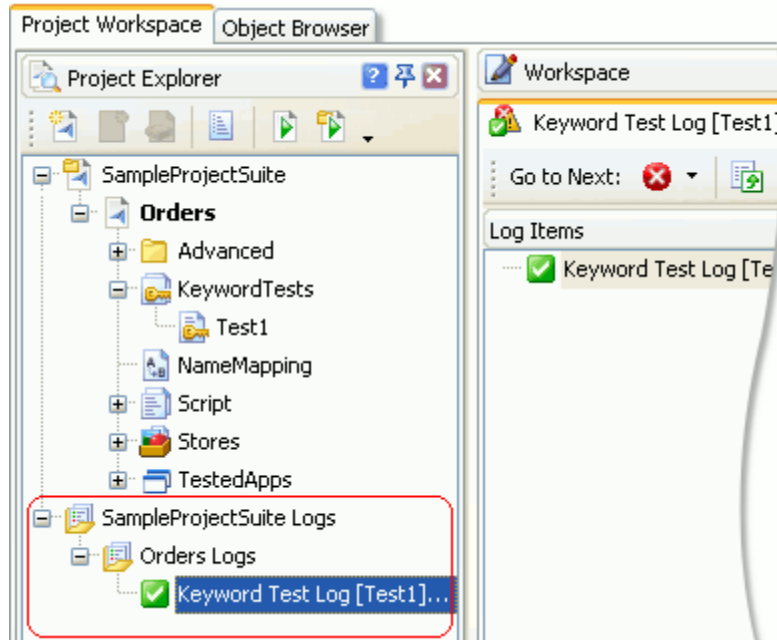
The indicator displays messages informing you about the simulated test actions.

- TestComplete executes the test commands until the test ends. You can stop the execution at any time by pressing **■ Stop** on the Test Engine toolbar or select **Test | Stop** from TestComplete's main menu. You can pause the test execution by clicking **▢ Pause**. During the pause, you can perform any actions needed. For instance, you can explore the test log or check the test's variables and objects using TestComplete's Watch List or Locals panel or the Evaluate dialog (see *Debugging Tests* in TestComplete Help).
- To launch the test we used the Run Test button on the test editor's toolbar. This is only one of several possible ways to run the test. You can also run tests from the Project Explorer, or from another test. You can also use the Test Items page of the project editor to create a batch run.

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

7. Analyzing Test Results

TestComplete keeps a complete log of all operations performed during testing. The links to test results are shown in the Project Explorer panel under the **ProjectSuite1 Log | Orders Log** node. This is the primary workspace for looking up the test history of the project and project suite. Each node corresponds to a test run. An image to the left of the node specifies whether the corresponding test run passed successfully:



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

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

In our example, the log is as follows --

The screenshot shows the Keyword Test Log window for a test run on 2/2/2010. The window is divided into several sections:

- Log Items:** A tree view on the left showing the test structure. The selected test is "Keyword Test Log [Test1]".
- Test Log:** A table of messages with columns for Type, Message, Time, Priority, and H... L... The messages include application start, menu clicks, file selection, list view clicks, toolbar clicks, mouse clicks, menu selection, text entry, and button clicks.
- Information:** Summary statistics: Errors: 0, Warnings: 0. Start Time: 11:58 AM 2/2/2010, End Time: 11:59 AM 2/2/2010, File Name: C:\Documents and Se...
- Picture:** A comparison of application states. The "Expected Image" and "Actual Image" panels show screenshots of the application's "Orders" window. The "Remarks" pane below the images contains the text: "Toolbar button 5 was clicked".

Type	Message	Time	Priority	H... L...
	The application "C:\Documents and Settings\All Users\Documents\TestComplete 8 Samples\Open Applications\C#\bin\Debug\Orders.exe" started.	11:59:00	Normal	
	The menu item 'File Open...' was clicked.	11:59:16	Normal	
	The file 'MyTable.tbl' was selected in the Open dialog.	11:59:19	Normal	
	The list view item ('Samuel Clemens', 0) was clicked with the left mouse button.	11:59:20	Normal	
	Toolbar button 5 was clicked.	11:59:21	Normal	
	The window was clicked with the left mouse button.	11:59:25	Normal	
	The window was clicked with the right mouse button.	11:59:26	Normal	
	The menu item 'Select All' was clicked.	11:59:27	Normal	
	The text 'Mark Twain' was entered in the text editor.	11:59:27	Normal	
	The button was clicked with the left mouse button.	11:59:28	Normal	
	The property value "Mark Twain" equals the baseline value "Mark Twain".	11:59:28	Normal	
	The button was clicked with the left mouse button.	11:59:30	Normal	

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.

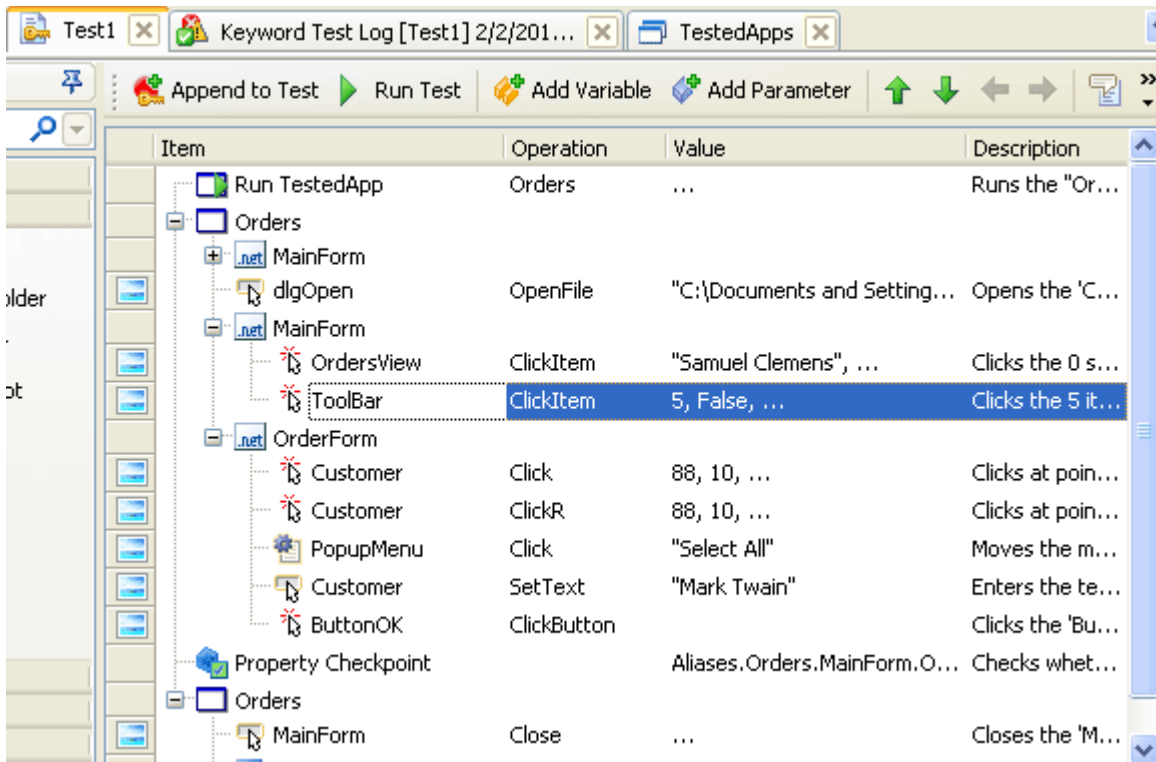
TestComplete may post additional text and images along with the message. To view them, simply select the desired message in the log and look in the **Remarks** and **Picture** panes that are below the message list. For instance, on the image above the Remarks pane displays the additional text for the message "Toolbar button 5 was clicked".

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

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

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

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 the *Test Log* section's topics 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 instance, the HTTP load tests form a log that contains tables with information about simulated virtual users, connections, simulated requests and their responses. For detailed information about these logs, see the description of the appropriate project item, or simply click within the log page and press F1.

Where to Go Next

This concludes the Getting Started tutorial. We hope it helped you to get acquainted with TestComplete. Now you can move onto learning 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:

- *Recording in TestComplete*
This section contains information on recording tests in TestComplete.
- *Naming Objects*
This section provides information on how TestComplete names processes, windows and controls.
- *Checkpoints*
This section describes various checkpoint types offered by the test engine and explains how to create checkpoints during test recording or test design.
- *Simulating User Actions*
This section describes how to simulate mouse clicks, keystrokes and selecting menu items with TestComplete.
- *Working With Applications' Object and Controls*
Explains how to perform specific actions over test objects and retrieve data from them.
- *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.

Technical Support and Resources

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

- <http://www.automatedqa.com/support/message>

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 **newsgroup** and on the **Community** web site, find answers to your question in the list of the **frequently asked questions**, read **technical papers** and take part in TestComplete **training seminars** offered by AutomatedQA.

For information on our support offerings, please visit our web site at <http://www.automatedqa.com/support>.

Index

A		R	
Analyzing recorded test.....	24	Recording tests	13
Analyzing test results.....	30	Running tests	27
Automated testing.....	3	Batch runs.....	29
B		Initial conditions.....	27
Black-box applications	7	Pausing	29
C		Stopping	29
Checkpoints	8, 17	S	
Creating	17	Scripts	3
F		Simulating user actions.....	26
Functional testing	3	Stores	8
K		Support and resources.....	33
Keyword tests	3	T	
L		Technical support and resources.....	33
Log.....	30	Test log	30
Jump to source.....	32	Jump to source.....	32
M		Test object model.....	6
Mapping object names.....	27	Test projects.....	4
N		Creating	10
Name mapping.....	27	Test results.....	30
Naming objects	6	Jump to source.....	32
O		Tested applications	10
Object Browser panel	5	Testing	
Object model	6	About automated testing.....	3
Object naming	6	Test types.....	3
Open Applications	7	Tests	
P		Analyzing recorded test.....	24
Panels.....	5	Analyzing results.....	30
Planning tests.....	12	Creating	9
Project Explorer panel	5	Planning.....	12
Project items	4	Recording	13
Project suites.....	4	Running	27
Projects	4	Test types.....	3
Creating	10	U	
R		UI testing	3
S		User interface overview.....	5
T		W	
U		White-box applications.....	7