

TestComplete基础教程.....	2
Getting Started Tutorial（快速入门教程）	3
Introducing Automated Testing and TestComplete（自动化测试与Testcomplete简介）	3
Automated Testing（自动化测试）	3
Test Types（测试的种类）	4
TestComplete Projects and Project Items（Testcomplete的项目与项目组）	4
TestComplete User Interface（Testcomplete的用户界面）	5
TestComplete Test Object Model（Testcomplete的测试对象模型）	7
Checkpoints and Stores（检查点和数据存贮）	9
Creating Your First Test（创建你的第一个自动化测试）	9
1. Creating a Test Project（创建一个测试项目）	10
2. Defining Applications to Test（定义被测程序）	11
3. Planning Your Test（制定测试计划）	12
4. Recording a Test（录制测试）	13
5. Analyzing the Recorded Test（分析录制的测试）	22
6. Running the Recorded Test（执行测试脚本）	24
7. Analyzing Test Results（分析测试结果）	25
TestComplete数据驱动测试.....	28
1.创建项目组（1: Creating a Project Suite）	29
2.存储测试数据（Creating a Data Storage）	30
3.建立一个迭代脚本（Creating One Script Iteration）	31
4.修改脚本、定制输入值（Modifying Script and Assigning Input Values）	38
把录制好的脚本拆分成不同的子程序：	38
创建Main函数：	44
修改Main函数的代码：	45
修改OpenForm, PopulateForm, Checkpoint, CloseForm, CloseApplication 子程序.....	48
获取测试数据.....	48
5.运行测试脚本，查看测试结果（Executing Script and Checking Results）	60
TestComplete测试.NET应用程序	60
Testing .NET Applications – Overview（概况）	60
.NET Open Applications（白盒测试）	62
Testing .NET Applications - Required Plug-Ins（所需插件）	63
Addressing Windows, Controls and Objects of .NET Applications（窗体、控件和对象的寻址）	63
Retrieving Data From .NET Objects（获取.Net对象的数据）	69

TestComplete 基础教程

Testcomplete 包含了一系列的教程来帮助你熟悉 testcomplete 和学习相关的测试技术。每一个教程都详细的步骤来指引你轻松完成整个项目的创建，涵括了各种类型的测试场景。

AutomatedQA Corporation



with
TestComplete™ 7

Getting Started Tutorial（快速入门教程）

这份教程的对象是初学者。它会教你如何让在 **testcomplete** 首次创建简单的功能测试。在你学习完这节教程之后，你会掌握录制、修改、回放测试的技术。

这份教程包括如下两部分：

1. 简要介绍自动化测试和 **testcomplete**
2. 创建你的第一个自动化测试：按照教程的指引步骤在 **testcomplete** 中创建你的第一个测试项目。

Introducing Automated Testing and TestComplete（自动化测试与 Testcomplete 简介）

这一节的主题是给出自动化测试和 **testcomplete** 的一个概况，包括：

自动化测试；

测试类型；

Testcomplete 项目和项目内容（Projects and Project Items）；

Testcomplete 的用户界面（User Interface）；

Testcomplete 的测试对象模型（Test Object Model）；

检查点和数据存储（Checkpoints and Stores）；

Automated Testing（自动化测试）

软件测试是一个检查应用程序并从中发现错误的一个过程。测试中需要比较应用程序的实际输出与预期输出是否一致，才可以下结论说应用程序是否实现了它应有的功能——这就是软件测试和试用软件的根本区别。换句话说，测试员不仅仅要保证应用程序显示了一系列的值，还要证实显示的值是正确的。

所以，测试的基本步骤如下：

定义预期输出；

执行测试（输入合适的值）；

收集程序的输出并与预期输出（基线数据）作出比较；

如果比较失败，要告知开发人员或者经理。

自动化测试是一种用特定的程序（无人干预或者少量的人工干预）自动执行软件测试。自动化执行保证不会跳过每一个测试动作；它使测试人员从重复执行沉闷的测试步骤中解放出来。

Testcomplete 提供了一些专为自动化测试动作、定义基线数据、运行测试和记录测试结果日志的功能。它还提供了专门的对话框和提示来帮助你在测试中自动化比较命令（或者检查

点)。

Test Types（测试的种类）

Testcomplete 支持多种测试类型和方法：单元测试、功能和图形界面测试、回归测试、分布式测试等。在这份教程里，我们将创建一个我们最常用到的测试——功能测试。功能测试会检查应用程序与其他系统、用户之间的接口。它们将验证应用程序的功能与预期是否相同。

一个典型的功能测试由一系列的测试指令组成，这些测试命令形如模拟用户的鼠标点击和键盘输入。循环运行这些测试指令来验证被测软件的功能。

在 **Testcomplete** 里，能够以关键字测试（**Keyword tests**）或者脚本（**Script**）来的形式创建功能测试。这两种类型的测试都可以用内置的编辑器来录制（**recorded**）或者从零开始创建。创建关键字测试是可视化的，非常简单且不需要任何编程基础。脚本方式创建的测试则需要理解相关脚本的命令，但你能够从中创建更强大、更灵活的测试。**Testcomplete** 支持的脚本包括 **VBScript**, **JScript**, **DelphiScript**, **C++Script** and **C#Script**，所以你可以从中挑选你最熟悉的脚本语言来使用。

在这份教程里，我们会使用关键字测试这一功能。

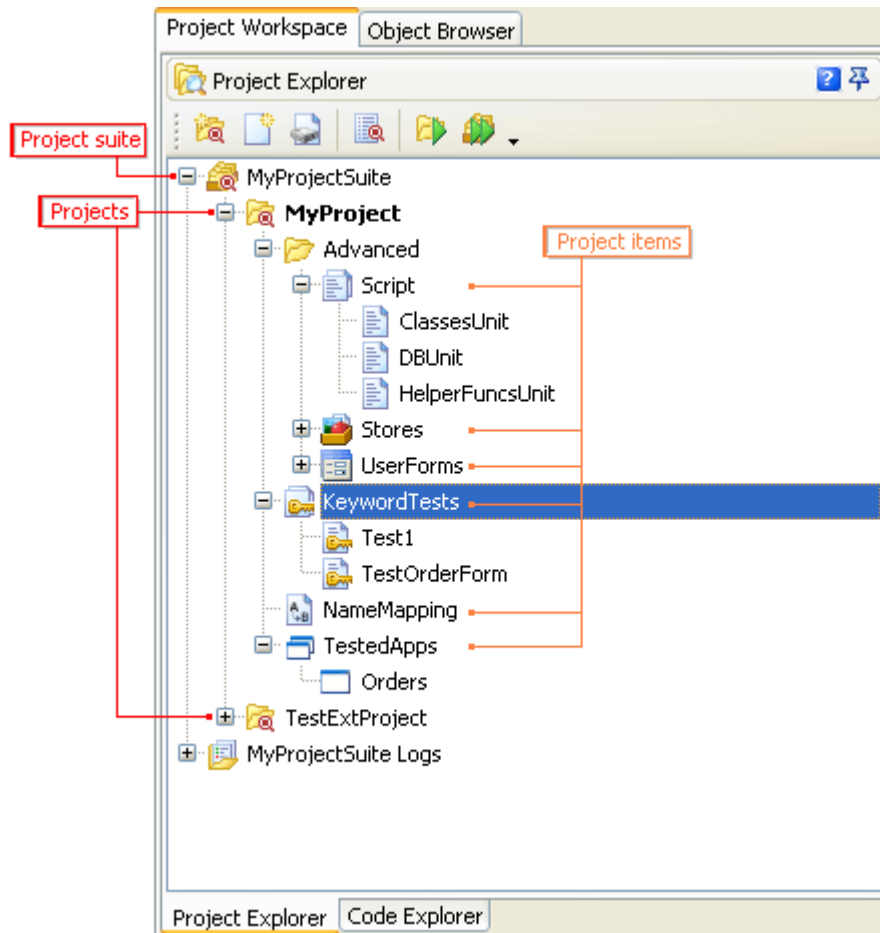
TestComplete Projects and Project Items（Testcomplete 的项目与项目组）

Testcomplete 用项目（**Projects**）和项目组（**project suites**）来管理。项目创建测试的开端。它包括了你的所有测试、检查点的基线数据、被测程序的相关信息和其在测试执行中需要用到各项内容。项目中也定义了多个测试之间的执行次序，和项目累计执行的测试日志。

一个项目可以包括对被测程序的所有测试。在复杂的被测程序中，你可以仅将一个项目专注被测程序的一部分，而让其他项目关注被测程序的另外的部分（通常是各个不同的模块）。

相关的多个项目（**Projects**）可以组织成一个项目组（**project suites**）。**Testcomplete** 会在你创建一个新项目的时候自动创建一个项目组。你也可以创建一个空的项目组，然后利用 **Testcomplete** 的对话框来对项目组添加相应的项目。

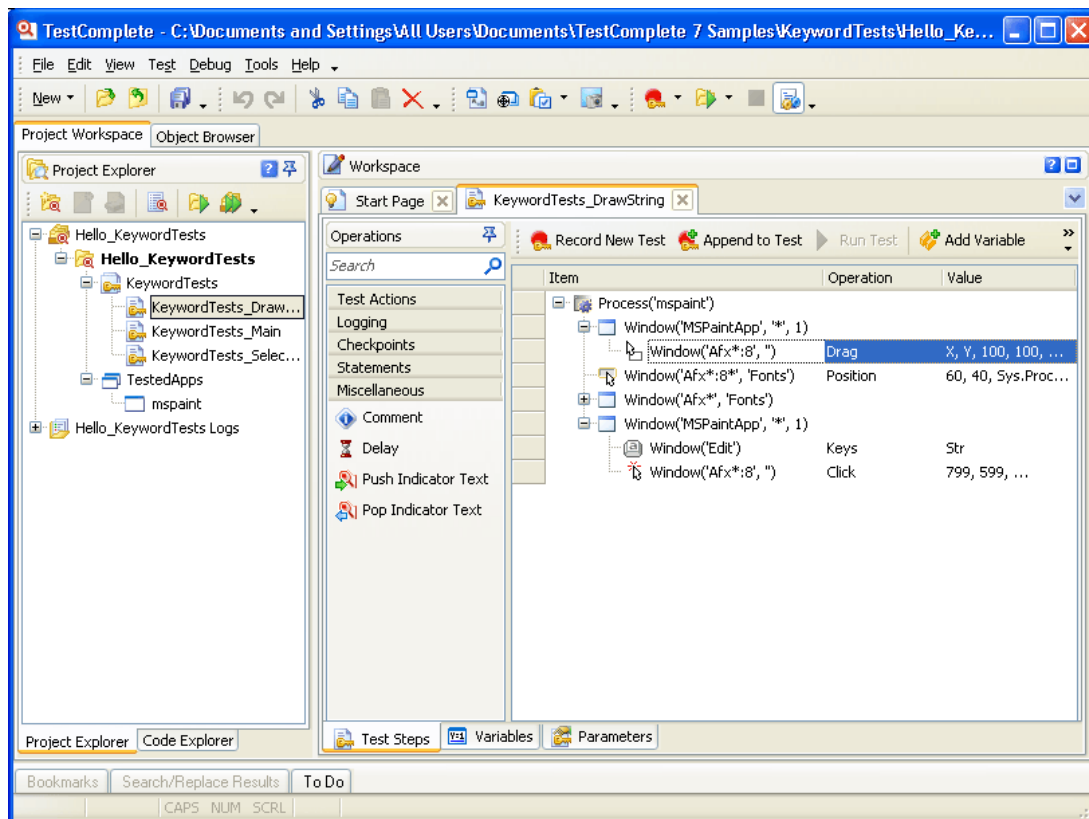
项目项（**Project items**）是用来执行或者支援测试的项目元素。你可以在**Testcomplete**里查看和管理项目、项目组、项目项，如下图所示：



如果项查看关于项目组的完整信息，请查看About Project Items.

TestComplete User Interface（Testcomplete 的用户界面）

Testcomplete 的主界面如下图所示：



如你所见，Testcomplete的用户界面组织在一系列的控制面板上。位于左边的项目浏览器面板（Project Explorer）显示了项目和项目组的内容。它也提供了可链接到测试日志的节点。

工作区面板（The Workspace panel）是你的工作台：它显示了项目和项目项的编辑器，你可以在上面创建或者修改测试、查看测试结果。例如，在上图中你可以看到关键字测试编辑器在工作区中处于打开状态。

除了项目浏览器和工作区，Testcomplete还提供了其他控制面板。例如，Watch List, Locals, Breakpoints 和 Call Stack面板供调试所用。To Do面板把将要执行的测试管理起来，Code Explorer面板则提供了浏览脚本内容、导航脚本单元的的快捷方式。

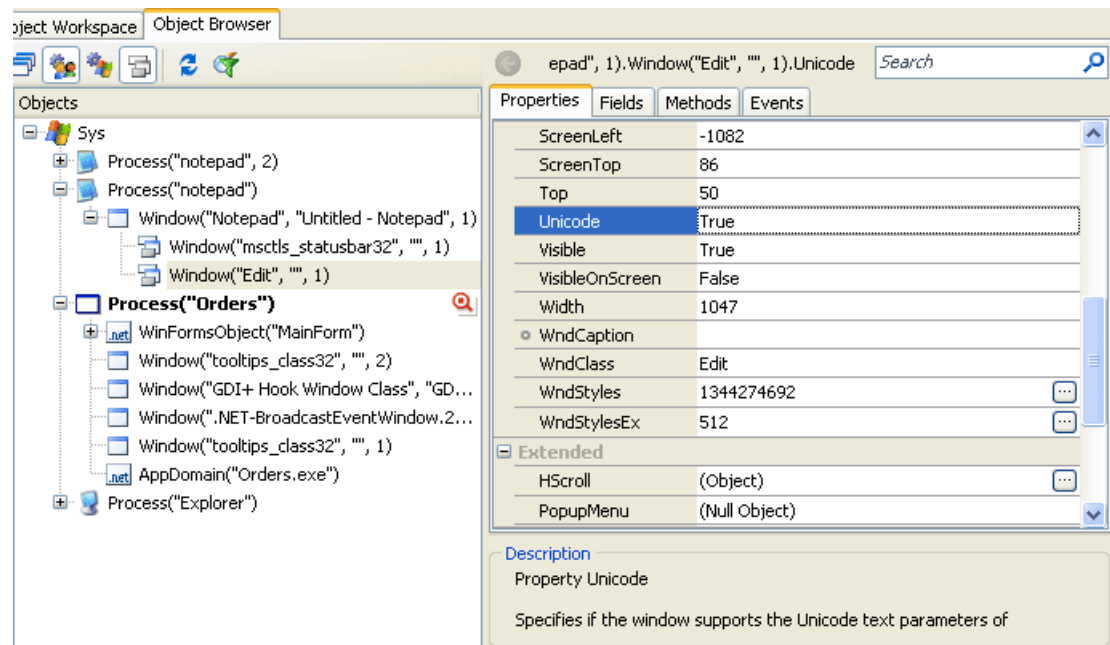
Object Browser 面板显示了 Testcomplete 中与项目无关的功能：它以列表的形式显示了当前机器上的所有进程和窗口。若对应进程和窗口的属性和方法能够被 Testcomplete 内部识别出来，Object Browser 面板就会显示。换句话说，Object Browser 能告诉你那些对象、方法和属性是可测的，和怎样取来测。

如果想跟深入了解面板的使用，在里面单击一下，然后点击 F1，就能打开对应面板的描述信息。

你可以使用菜单和工具栏来执行相应的功能。它的菜单、工具栏系统和 Microsoft Visual Studio、流行的 windows 应用程序非常类似。你可以改变工具栏的位置，把里面的内容转移到其他菜单或者工具栏中、隐藏内容、把隐藏的内容还原等等。请参见 Toolbars Customization。

TestComplete Test Object Model (Testcomplete 的测试对象模型)

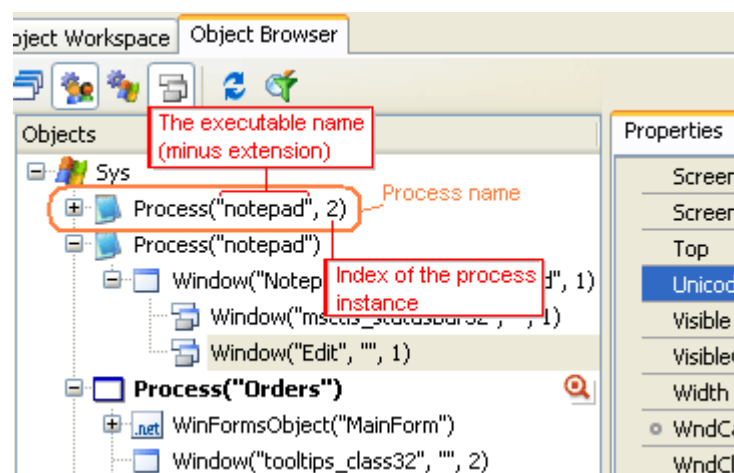
对象浏览器面板显示了对象的结构，如下图示：



Testcomplete 使用了树形模型来组织被测对象。根结点是 Sys（桌面应用程序和窗口）和 PDA（一些运行在连接你计算机的 WindowsMobile 设备的程序）

Process 对象 相当于运行在操作系统上的应用程序。我们用术语 **Process**（进程）而不是 application（应用程序），这是因为这等价于 windows 文档里有关进程的概念。

进程对象的名称包括正在执行的进程的名称和它对应的索引（index）值。注：index 只使用在多个应用程序执行的环境下：

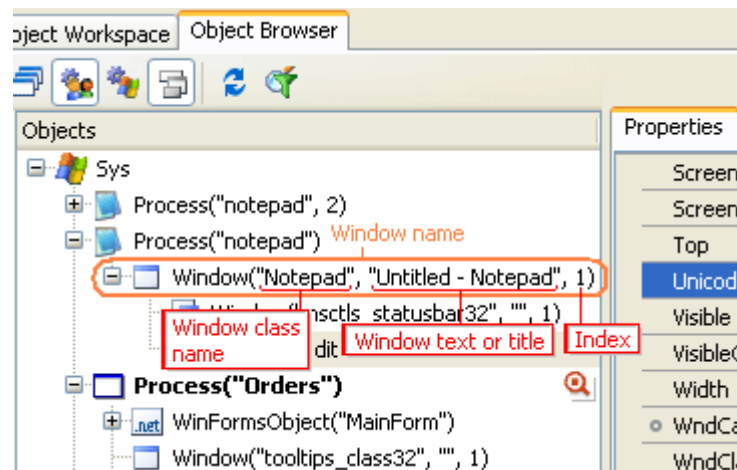


进程（Processes）有他们的子对象（窗口），也就是说（子对象的）顶层窗口。这些对象都


拥有它们对应控件的子窗口。这些窗口和控件的命名取决与测试引擎是否能够识别出被测应用程序的内部方法和属性。**Testcomplete** 支持上述的两种类型，但用不同的方法来命名它们的各总窗口的控件。

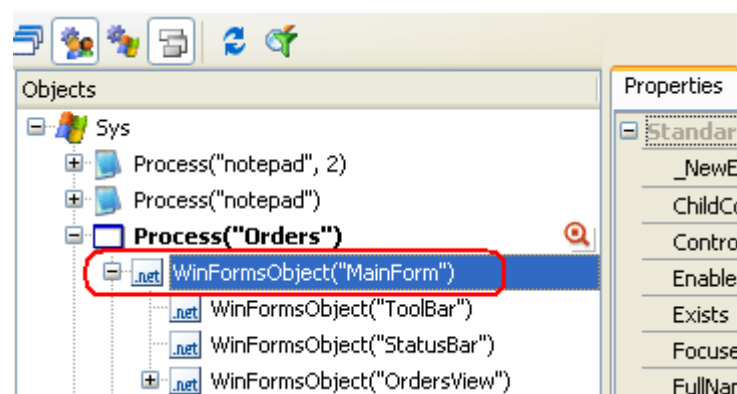
黑合应用程序：

黑合应用程序指的是不提供访问它们内部方法和属性的应用程序。它们的命名包括 window's class name, window's text 或 title (caption) , 和它的 index。控件的命名方式和窗口的命名方式类似，因为就操作系统而言，控件只是窗口的类型之一：



白盒应用程序：

那种向 **Testcomplete** 提供其内部属性和方法的应用程序叫做白盒应用程序或者开合应用程序。它们用  作标示，显示在对象浏览器上。为了突出白盒应用程序的窗口和控件，**Testcomplete** 使用了特别的命名方式，可以反映出控件或窗体的类型、在源码中定义的名称。例如，你有个用 C#调用 Microsoft WinForms 库生成的名为 MainForm 的窗体，**Testcomplete** 将以这种形式来识别：WinFormsObject ("MainForm")



注：强烈建议在有条件的情况下选择白盒应用程序来测试，而不是黑合应用程序。这使得测试引擎能够识别出被测程序的内部方法和属性，使你的测试更加有效灵活。

有些程序像.NET, WPF, Visual Basic, Java 或者 Web 对 **Testcomplete** 都是白盒应用程序，而其他的可能要通过一些特别的手段编译才行。

Checkpoints and Stores（检查点和数据存储）

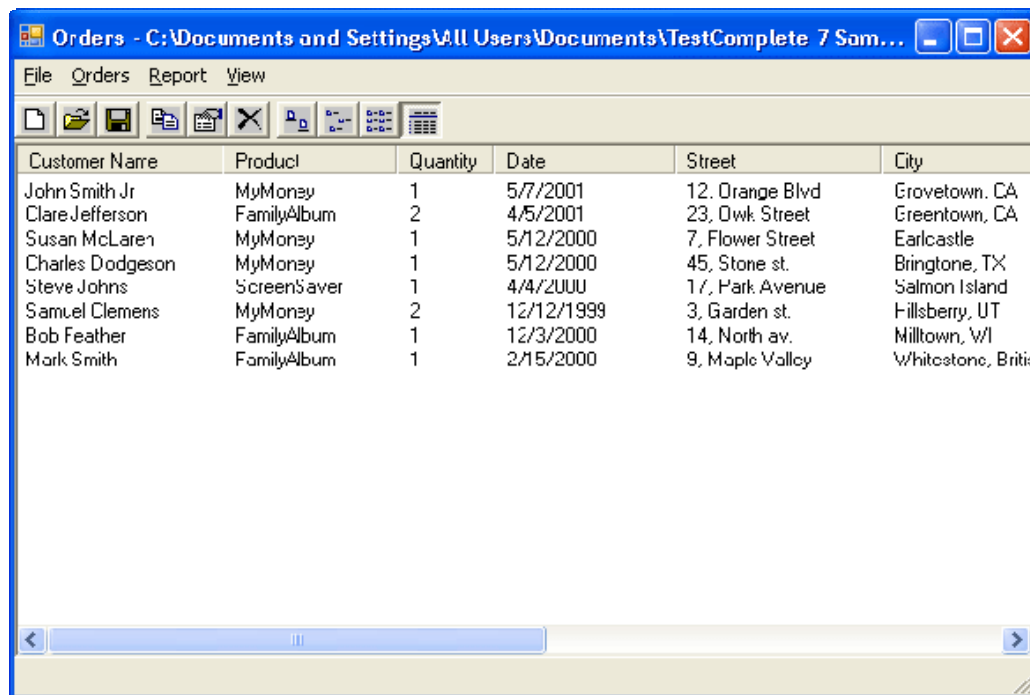
一个典型的测试中会进行多次的比较。例如，一个模拟用户输出应用程序的文件的测试场景，你需要去验证输出的数据是否有效。为了完成这一测试，你需要将输出的文件和预期的文件相比较。这仅仅是一个例子，在真实的测试中，你要作成百上千次比较。各种类型的自动化测试（回归、单元、功能等等）都需要一个参照值来对比验证。

在 **Testcomplete** 下可以很轻松地在测试中加入比较命令（或者检查点）。你可以在录制测试和设计时加入检查点。**Testcomplete** 提供比较不同类型数据的检查点：图像、文件、对象文本和属性（Object text and properties）、xml 文档、数据库表（database tables）等。**Testcomplete** 包含了数据存储（**Stores**）这个项目项（project item），可用来存放检查点的基线数据。这些项目项是一个容器，存放了图像、文件和其他用于测试比较的元素。唯一例外的是用于验证对象属性的检查点：这些基线数据是在测试中定义的。

Creating Your First Test（创建你的第一个自动化测试）

这一小节的教程提供了详细的操作步骤，告诉你如何在 **Testcomplete** 下创建项目、录制和回放简单的测试、和分析测试结果。这个测试模拟了用户使用被测软件的行为，同时验证了一些数据。验证命令会在录制测试的过程中生成。

在我们的讲解里，我们使用一个名为 **Orders** 的应用程序来作演示（它是随着 **Testcomplete** 发布的）。这一应用程序显示了一个订单列表，还包含了一些特定的功能：增加、删除、修改、输出订单。



The screenshot shows a window titled "Orders - C:\Documents and Settings\All Users\Documents\TestComplete 7 Sam...". The window has a menu bar with "File", "Orders", "Report", and "View". Below the menu bar is a toolbar with various icons. The main area 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 Dodgeson	MyMoney	1	5/12/2000	45, Stone st.	Bringstone, 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	Whitestone, Briti

这个应用程序在以下路径可以找到：

- 在 Windows 7, Windows Vista 或者 Windows Server 2008:

C:\Users\Public\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo

- 在 Windows XP, Windows Server 2003 或者 Windows 2000:

C:\Documents and Settings\All Users\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo

注意在 Windows Explorer 和打开、保存文件框下, *All Users\Documents* 文件夹可能会显示成 *All Users\Shared Documents*.

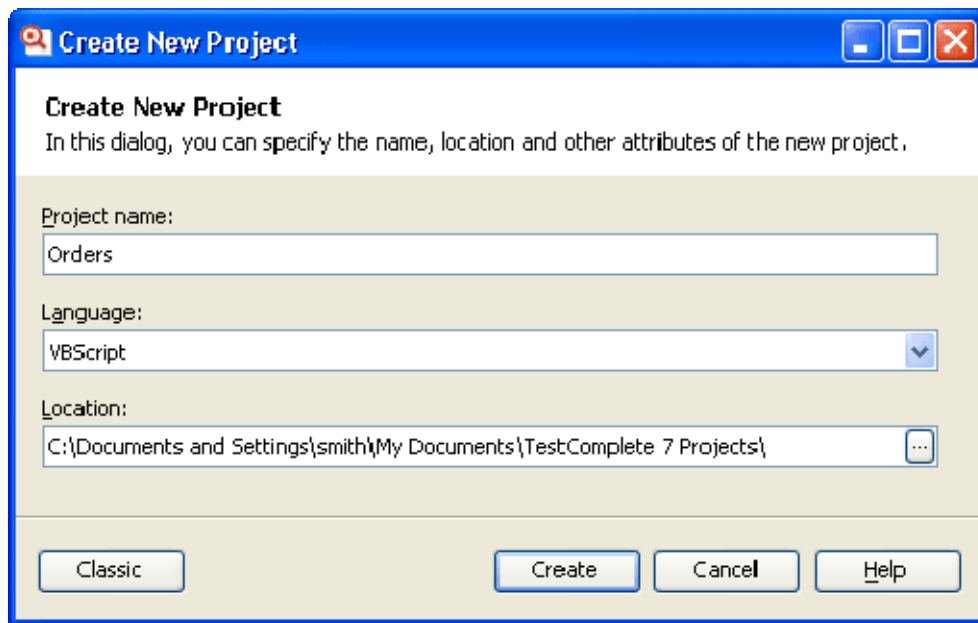
这个文件夹下存放和很多订单项目, 它们由不同的编译器创建的: C#, Visual C++, Visual Basic, Delphi, C++Builder, Swing 等等。我们采用由 C#创建的 *Orders* 应用程序。

1. Creating a Test Project (创建一个测试项目)

让我们来创建一个新项目:

1. 如果你当前在 Testcomplete 下打开了项目组或者项目, 先关闭它们。在菜单栏点击 **File | Close** 能完成这一步骤。
2. 在菜单栏选择 **File | New | New Project**。这会调用创建新项目的对话框。这个对话框由两种工作模式: 简单 (Simple) 和典型 (Classic)。在简单模式是默认的模式, 对话框包括了 3 个输入文本框, 你可以填上项目名、路径和脚本语言。在典型模式下, 你也可以定义项目组的名称 (project suite name)、选择项目模板 (project's template) 和项目项 (project items)。

这份教材用的是简单模式, 它比典型模式更常用。



3. 我们现在输入项目名、路径和脚本语言
 - 在 **project name** 输入 *Orders*: Testcomplete 会自动创建项目的路径并显示在 **Location** 这一栏上。项目文件家是用来存放项目生成或者用到的信息: 关键字测试、脚本、测试日志、数据等等。你可以在 **Location 文本框**中改变项目的路径。在我们的例子里我们用默认值。
 - 在 **Language** 文本框我们定义在测试中将要用到的脚本语言。在这里我们选择了

VBScript。即使你不使用脚本单元，脚本语言也是非常重要的。即使你准备使用关键字测试，你可能也会调用一小段的代码或者使用脚本来定义执行参数（**operation parameters**）。

说脚本语言重要，是因为它定义了对象名（**object names**）的格式，这是测试总要用到的，不管你是用脚本还是关键字测试。名称的格式是由脚本语言的语法来定义的。例如，在 **VBScript** 和 **Jscript**，**Notepad** 显示成 `Process("Notepad")`，而在 **DelphiScript** 你要把双引号变成单引号，如 `Process('Notepad')`；在 **C++Script** 和 **C#Script** 下“**Process**”必须在方括号里：`["Process"]("Notepad")`。

Testcomplete对**VBScript**, **JScript**, **DelphiScript**, **C++Script** 和 **C#Script**都支持，因此你可以选择你最熟悉的语言。无论你选择哪种语言，你都能够使用**Testcomplete**的所有特性。然而，也难为不同的语言有不同的语法，它们的脚本解释器是由不同的脚本引擎来执行的，它们会有一些例外（见*Supported Scripting Languages – Peculiarities of Usage*）。想获得更过关于语言选择的信息，请看（*Selecting the Scripting Language*）。请注意，脚本语言和被测程序使用什么语言是毫无关系的。例如，你用 **Jscript** 来测试 **Visual C++**写的程序或者用 **VBScript** 测试 **Delphi** 写的程序。然而，如果你想创建一个连接（**Connected**）或者自动自测（**Self-Testing Application**）的程序，我们建议你选择与被测程序的开发工具关联最密切的语言。例如，如果你使用 **Visual C++**或 **C++Builder**，你应该选择 **C++Script**；如果你使用的是 **Visual Basic**，你应该选择 **VBScript**，以此类推。这会使录制脚本更容易导入连接（**Connected**）或者自动自测（**Self-Testing Application**）程序。

4. 在你定义了项目名称、脚本语言和路径之后，点击 **Create** 按钮。**Testcomplete** 将会创建一个新项目，**Orders.mds**，和一个对应的项目组。它会接着在项目浏览器面板显示项目组和项目内容。

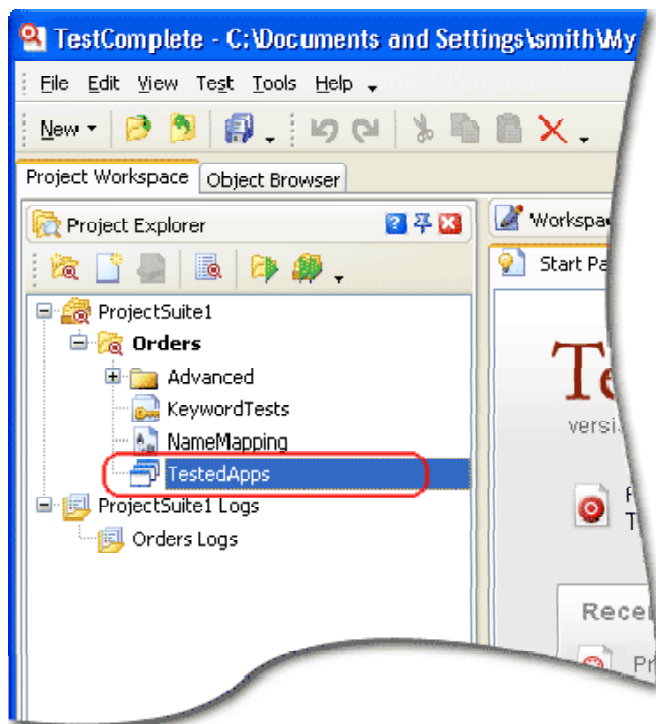
2. Defining Applications to Test（定义被测程序）

每个项目或许会有被测程序的列表。这是让你跟踪哪个项目对应哪个程序和它们在测试里如何配置。这允许 **Testcomplete** 去执行列表里面的每一个引用程序或者在内容菜单或测试里手工执行。当然，因为项目之间是相对独立的，不同的被测程序会分布在不止一个的项目里面。你可以手工向列表添加被测应用程序，或者让 **Testcomplete** 在录制测试的过程中自动完成。录制器十分聪明，能够通过命令行、**windows Explorer** 或者其他方式检测到应用程序开始执行。在录制结束之后，**Testcomplete** 会把被测程序加载到列表中，同时加载“**Run Tested Application**”到录制完的测试里面。

在这一份教程里，为了让你更熟悉列表、演示 **Testcomplete** 中管理被测程序这一特性，我们手工来向列表添加被测程序。

让我们添加一个简单的被测程序到列表里：

1. 在项目浏览面板里右键单击 **TestedApps** 节点



2. 在内容菜单里选择 **Add | New Item**，这会弹出一个标准的 **Open File** dialog。
3. 找到使用这个 **Open File** dialog 的 **Orders.exe** 可执行文件，然后点击 **Open**。
回忆一下，我们将使用 **Orders** 这个用 C#编写、随着 **Testcomplete** 发布的简单程序。它的路径像是这样的：

C:\Documents and Settings\All Users\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo\C#\bin\Debug\Orders.exe (Windows XP, Windows 2000 or Windows Server 2003), 或 C:\Users\Public\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo\C#\bin\Debug\Orders.exe (Windows 7, Vista or Windows Server 2008).

4. 在你向列表添加完被测程序后，你可以定义它的运行模式和运行参数。要获取更多的信息，请查看 *Run Modes* 这份帮助文档。在这份教材里，我们会使用默认设置。
5. 通过在主菜单点击 **File | Save** 来保存你的更改。

3. Planning Your Test（制定测试计划）

General Notes on Planning Tests（制定测试计划的注意事项）

通常，做个测试计划是个很不错的实践：

- **定义测试目标**和制定哪些功能需要被测试。目标越清晰，测试越简单、越有效。大型的、要处理大量程序行为的测试非常的难以创建和维护。创建一个目标明确的、简单的测试更有好处。一旦你建立了很多简单的测试，你可以将它们组织到一个更大的测试里面。
- **计划测试步骤**和决定运行哪些测试场景。测试步骤由测试的目的和被测程序的特性来共同决定。测试步骤可能包括测试程序的前期准备（比如被测程序的初始化）。

测试步骤也可以是给应用程序一些初始的输入值。

- **计划检查点的动作。**通常，在应用程序执行了一些操作之后，应用程序将会发生一些改变：在程序窗口的数据可能会被改变或者重新分配、一个新的窗口被创建、一个文件在硬盘里创建或者删除。你必须定义测试通过与否的标准和哪些检查点来验证这些标准。
- **记录测试结果。**测试结果能用不同的方式记录下来，例如，你的测试可以把完整的测试结果保存在一个文件里，或者弹出一个对话框来提醒你的测试已经结束了。`Testcomplete`记录了测试中模拟的所有行为同时在日志里保存了这些行为的信息。你也可以记录定制信息、图像、文件或者指向测试日志的链接。这些信息可以组织成文件夹，且每一份信息都可以用特定的字体和背景色来显示。测试结果可以输出到文件、压缩和通过电子邮件发送到你同事。你甚至可以直接通过测试结果的日志，在缺陷跟踪系统创建缺陷报告。请查阅 *Test Log* 来获取更多信息。

Planning a Test for the Orders Application（为订单程序制定测试计划）

这个 **Orders** 应用程序的例子维护了一个订单列表。假定我们要测试这个系统编辑订单的功能是否正确、是否成功修改了订单的数据，在这个例子里：

- **测试目的：**这个测试应该检查编辑订单界面是否保存了修改后的数据、改变后的数据是否能够显示出来。
- **测试步骤：**我们的测试必须模拟出修改订单的细节，同时验证订单列表里面的数据。我们将录制模拟用户操作的测试。为了简单起见，我们的测试只改变了一个订单的一个属性。
- **检查和验证测试结果：**对订单的修改能正确保存，那它应该显示在订单列表里。为了作出检查，我们的测试将列表里的数据和预期数据作出比较。我们在测试里加入特殊的指令来完成这一任务。这个指令会把对比的结果记录到测试日志里，从而我们可以验证测试通过与否。

4. Recording a Test (录制测试)

Recording Tests in TestComplete (在 Testcomplete 里录制测试脚本)

录制测试由一下三个步骤组成:

1. 在Testcomplete的主菜单或者测试引擎工具栏点击**Test | Record | Record Keyword Test** 或 **Test | Record | Record Script**。你也可以在开始页面上点击**Record a New Test**。在Testcomplete下面，你可以选择一下类型的测试录制：**keyword tests**, **scripts**, **low-level procedures** 和 **HTTP load testing tasks**。录制用的菜单项定义了主要的测试录制方式：**keyword test** 或 **script code**。其他的测试类型将会在录制开始之后启动。主要的录制方式会包含特别的指令来完成这些测试。
在你命令 Testcomplete 启动测试之后，它会切换测试类型和在屏幕上显示 **Recording toolbar**:



这个工具栏包括了一些附加功能，你可以在录制中使用，暂停或停止录制，改变录制类型(keyword test, script code, Windows or PDA low-level procedure, 或 HTTP traffic).

2. 在你启动测试之后，执行以下的测试步骤：启动被测程序（有必要的话）、在上面点击命令按钮、选择菜单项、键盘输入文本等等。
 3. 在测试动作结束后，在Recording toolbar点击  **Stop**来停止录制。
- 要获取更多的信息，请查阅帮助文档*Recording in TestComplete*；

Recording Test for the Orders Application（为订单应用程序录制脚本）

现在，我们为 **Orders** 这个应用程序录制一个关键字脚本。这个测试会运行应用程序、往里面加载数据、模拟程序界面内的鼠标点击和键盘输入、验证程序的数据。

注：如果你在屏幕里阅读这份文档，请不要在录制脚本的时候进行切换。因为录制引擎会跟踪和录制用户的所有操作，切换文档的指令也会录制到的测试脚本里。如果真的想看这些操作的话，你可以事先打印出栏。或者你用两台显示器，这样你就可以把这份文档放在另一个屏幕里。

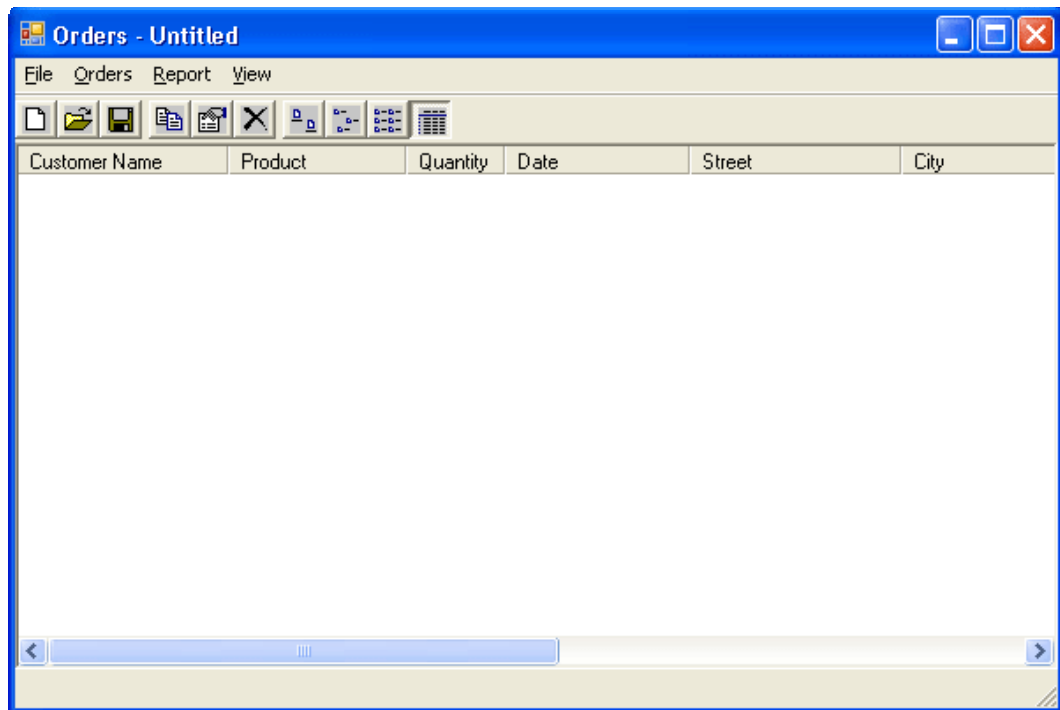
开始录制：

1. 从主菜单选择 **Test | Record | Record Keyword Test**
2. 点击展开  **Run Tested Applications** 按钮旁边的小箭头，在下拉列表选择 **Orders**：



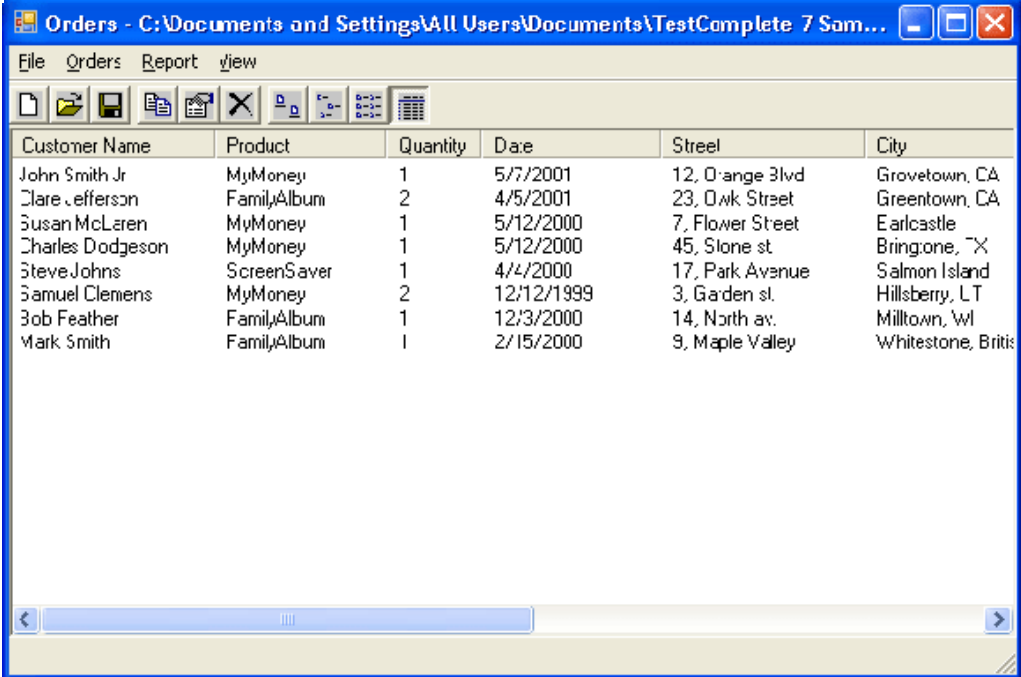
Testcomplete 会自动把启动应用程序的命令加载到录制的测试里。在后面我们分析录制测试的时候你会看到这条指令。

3. 稍等片刻，直到应用程序的主窗口启动，如下图示：



如果交互式帮助文档的界面可见的话，请调整它的大小或者移开它，一面挡住了应用程序的窗口。 否则你在界面上的操作是无法录制的。


4. 切换到 **Orders** 程序，点击它的主菜单 **File | Open**。这会打开一个标准的打开文件对话框（Open File dialog）。
5. 在这个对话框里，打开文件 *MyTable.tbl*。它的路径取决与你安装的操作系统。在 Windows 7, Windows Vista and Windows Server 2008 下的路径是 *C:\Users\Public\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo*。在其他操作系统，它的路径在 *C:\Documents and Settings\All Users\Documents\TestComplete 7 Samples\Open Apps\OrdersDemo*。建议你在 Open File dialog 的 **File name** 文本框用键盘输入完整的文件名。用键盘输入而不是鼠标点击，目的是防止由于 Open File dialog 显示不同的初始路径而造成回放问题。
6. 在 **File name** 文本框输入文件名后，点击 **Open**。Orders 应用程序会从文件加载数据，将其显示在主界面上：

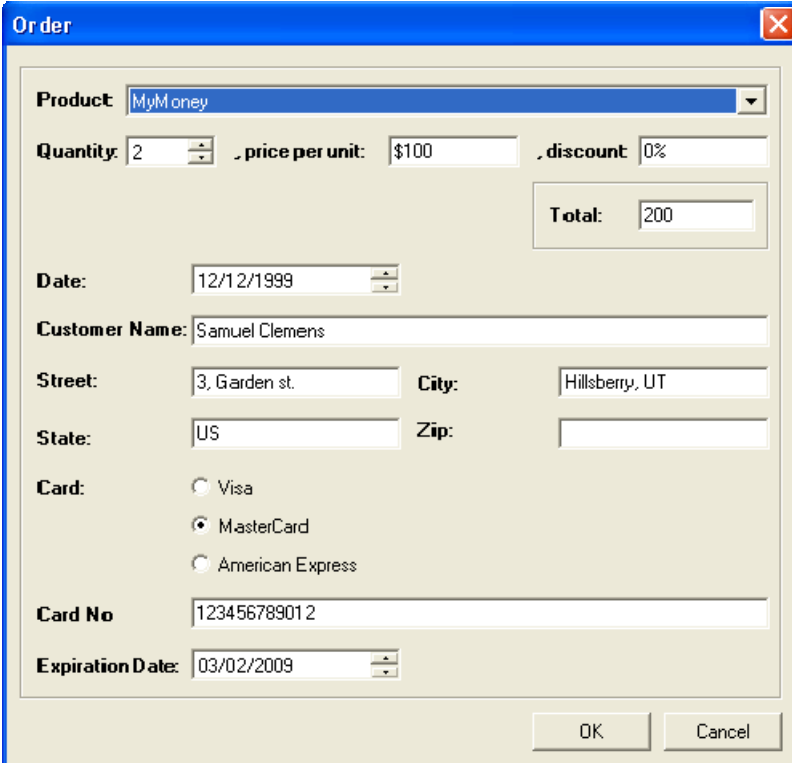


Orders - C:\Documents and Settings\All Users\Documents\TestComplete 7 Sam...

File Orders Report View

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	Bringstone, 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	Whitestone, British

7. 在订单列表点击 *Samuel Clemens* 这一行。
8. 移动鼠标到 Orders 的工具栏上，并点击  **Edit order** 调用 **Order** 对话框：



Order

Product: MyMoney

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

Total: 200

Date: 12/12/1999

Customer Name: Samuel Clemens

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

State: US Zip:

Card: ☐ Visa ☒ MasterCard ☐ American Express

Card No: 123456789012

Expiration Date: 03/02/2009

OK Cancel


9. 在对话框里，点击一下 **Customer Name** 文本框把光标插入点（insertion point）移进去。在 **Customer Name** 文本框右击鼠标，从内容菜单里选择 **Select All**，然后输入 *Mark Twain* 作为客户名称。

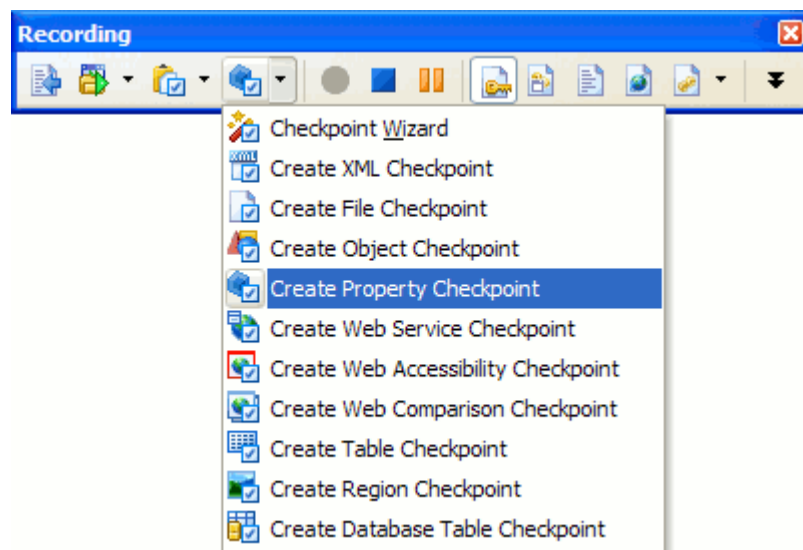
注：你可以通过鼠标拖动文本框的文本来选择，但这种情况下的测试就没那么通用了，因为如果文本长度是一个变量，在测试里我们不得不拖动长一点或者短一点，

换句话说，你不得不在测试中计算你拖动的长度。使用 **Select All** 菜单项这个快捷方式，你可以免去计算的问题，同时使你的测试更加稳定。另一种可选的途径也可以实现 **Select All** 菜单项这个功能：使用快捷键（通常是 **CTRL+A**）来选择所有文本。

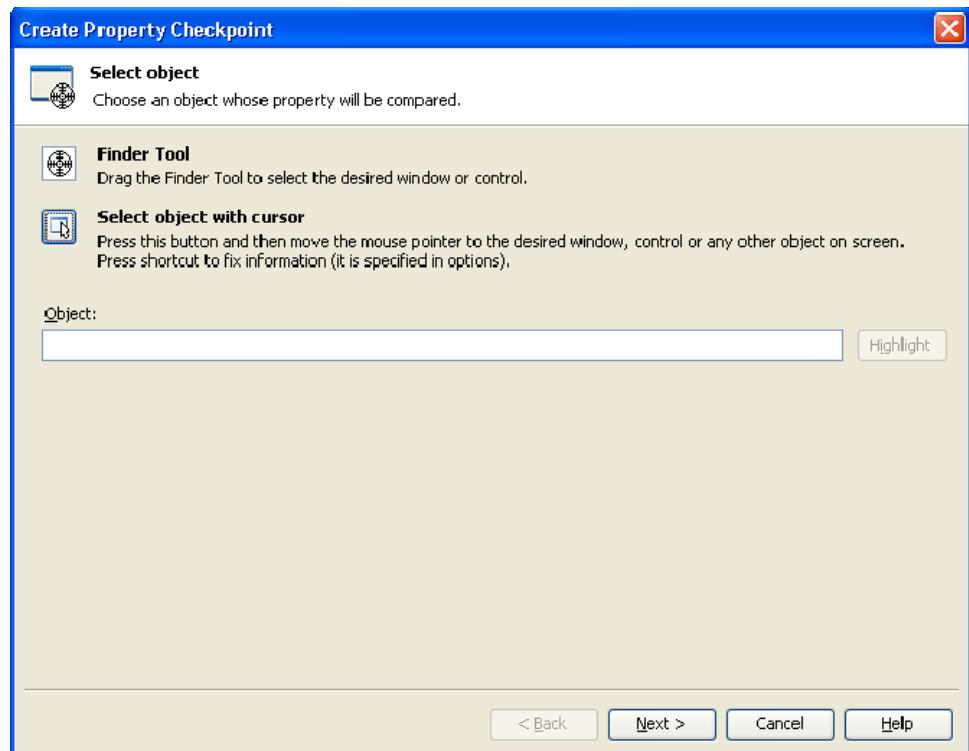
10. 点击 **OK** 来关闭对话框。**Testcomplete** 会更新应用程序主窗口里面的客户列表（customer list）。
11. 现在我们在测试中插入一个比较指令。它将会验证显示在应用程序客户列表中被修改过的名字——*Mark Twain*。

我们可以把这类比较指令称作检查点（**checkpoints**）。**Testcomplete** 提供验证不同类型数据的各类检查点。其中用得最多的检查点是属性检查点（**Property checkpoint**）。它用来验证应用程序的控件数据。我们在这份教材里使用这个检查点。

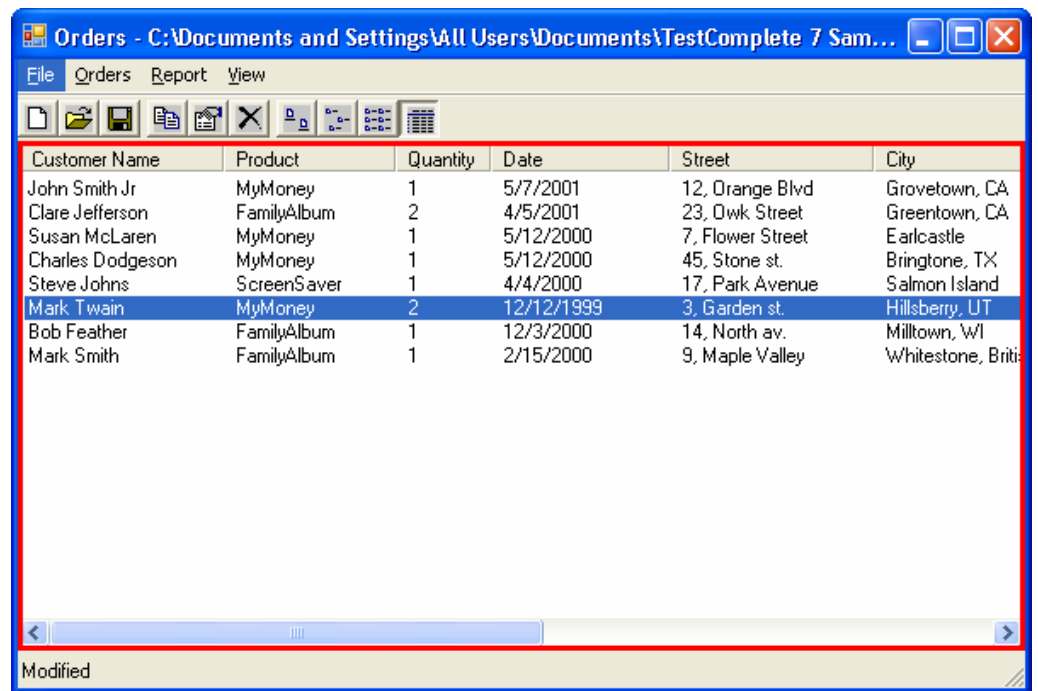
- 从录制工具栏（Recording toolbar）的 **Checkpoint** 下拉列表选择  **Create Property Checkpoint**:



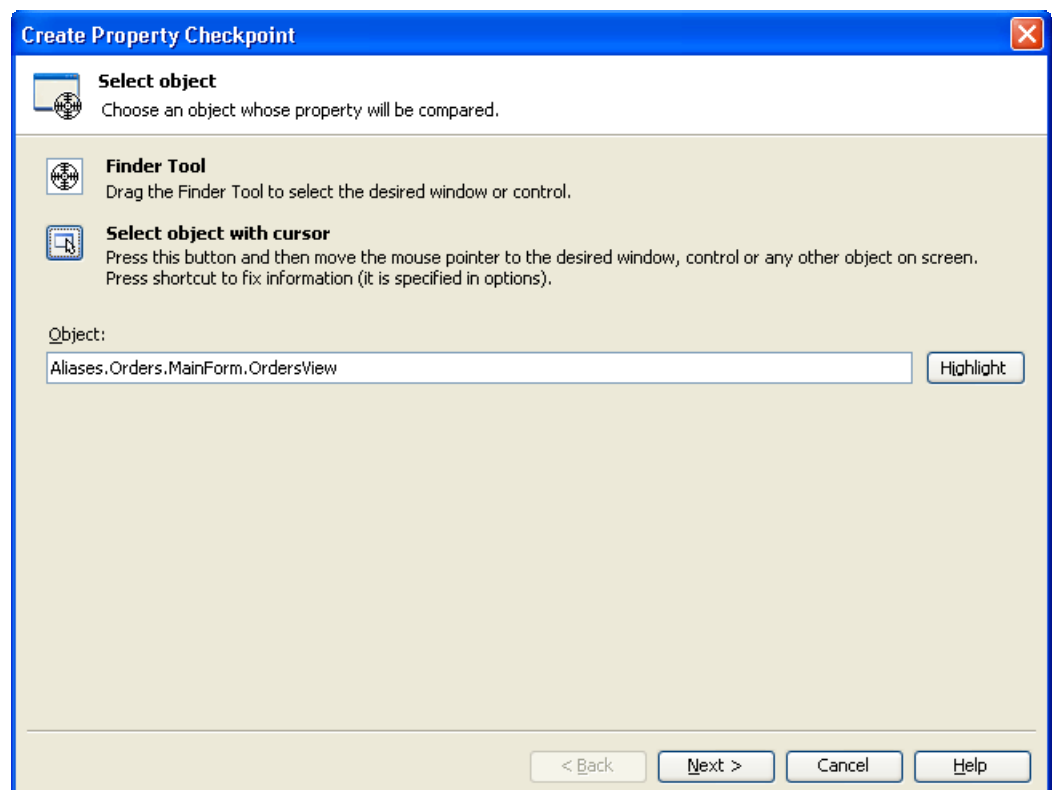
- 这会调用 **Create Property Checkpoint** 这一向导。它会指导完成创建检查点的过程：



- 在向导的第一页，用鼠标左键单击 Finder tool 这个图标 (🔍)，同时不要松开鼠标左键。
等待向导页面最小化，然拖动图标到 Orders 应用程序的客户列表。在你拖动的过程中，Testcomplete 会在红框里突出光标下面的控件和窗口。
- 当 Finder tool 图标放在客户列表上面、同时客户列表在红框里突出显示时，松开鼠标键：



- 在你松开鼠标键后，Testcomplete 会保存这个向导并把所选的对象显示在 **Object** 文本框下：

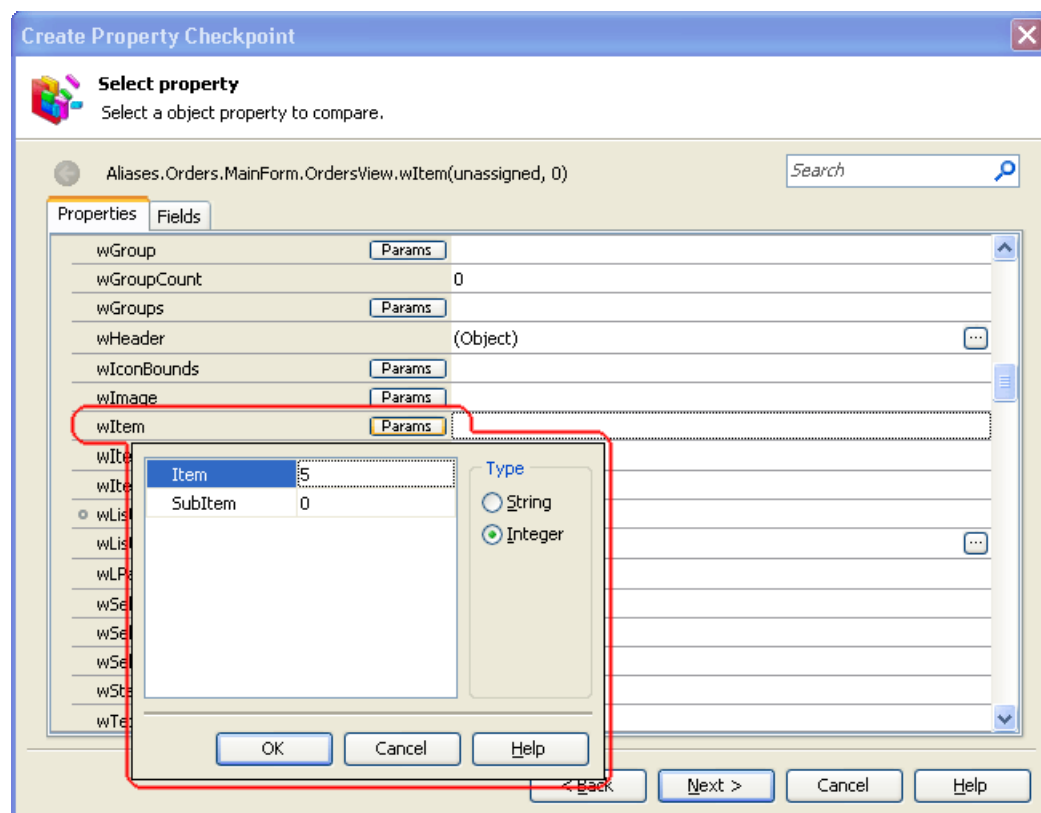


点击 **Next** 进行后续操作。

- 向导的下一页显示了被选对象的属性列表。这个列表包括了 Testcomplete 提供的属性和被测程序自己定义的属性。例如，我们的被测程序是用 C# 创建的，所以列表上显示的是对应 .Net 类库的属性。你可以在 **.NET** 节点下看到它们。

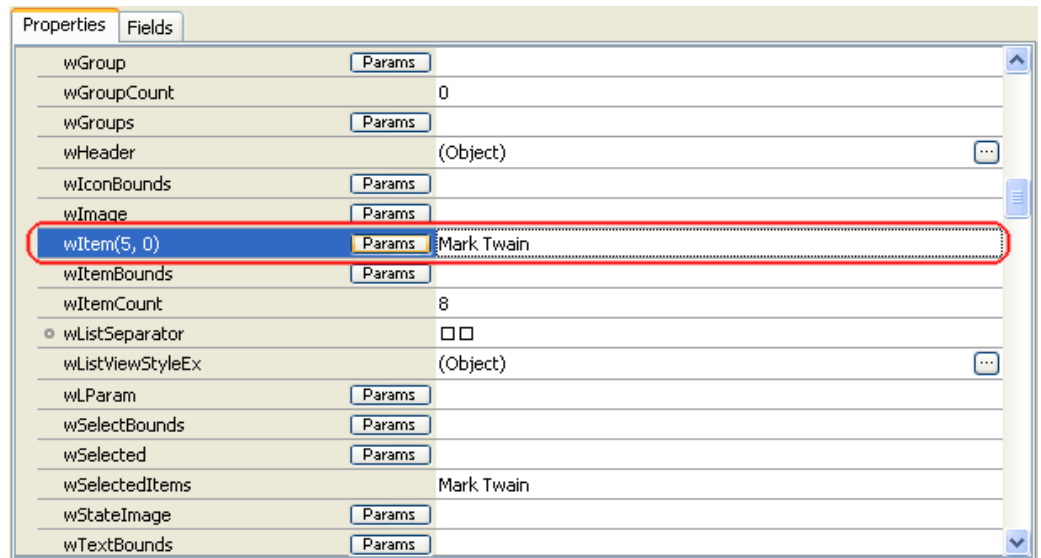
Testcomplete 提供了两组对象的属性：一组包括了被测窗口或控件的公共属性。大部分可以在 **Standard** 节点那里查看。另一组包括了 **tree-view** 控件特有的属性（因为我们选择的是 **treeview** 控件）。这些属性名称的首字母是 **w**。你可以在 **Extended** 节点下看到它们。为了验证数据，我们使用 **wItem** 属性。可以通过它访问 **tree view** 控件里每一项。

- 遭到列表里面的 **wItem** 属性（在 **Extended** 的节点下面）。单击 **Params** 按钮。弹出如下窗口：



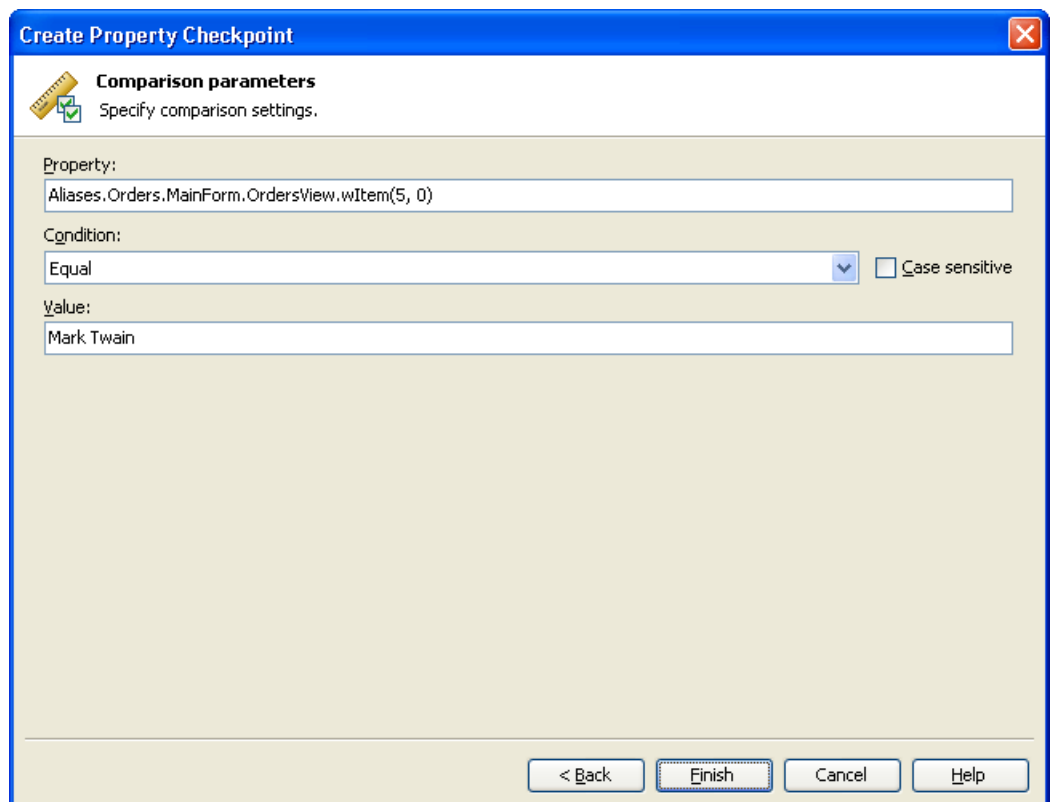
在这个窗口里面，定位到持有 *Mark Twain* 这个字符串的单元格：

- 在 **Type** 区域选择 **Integer**。
 - 在 **Item** 文本框键入 **5**。（5 是 *Mark Twain* 在 **treeview** 控件里的索引。索引的值是从 0 开始递增的）
 - 点击 **OK**。
- 测试引擎会检索到这一项的数据并显示在属性列表中：



点击 **next** 继续。

- 在向导的下一页，你会看到属性的名称（将要验证的值）。比较的条件和基线数据在 **Value** 文本框定义：

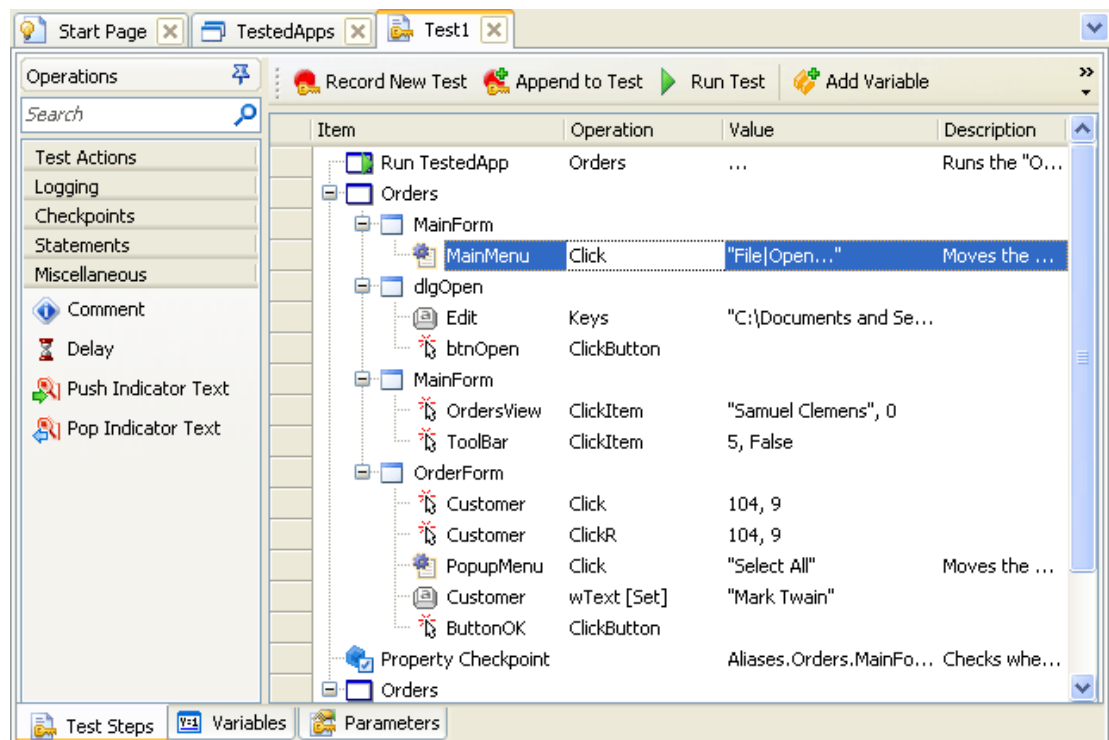


点击 **Finish** 来完成检查点的创建。Testcomplete 会将检查点的指令附加到录制的测试中。

- 点击 **window** 标题栏的 **X** 按钮来关闭 **Orders** 程序的窗口。这会弹出一个对话框来问你是否要保存变更。点击 **No**。**Orders** 程序关闭了。
- 点击录制工具栏的 **Stop** 按钮来停止录制。Testcomplete 会处理录制的测试指令并将测试保存下来。

5. Analyzing the Recorded Test (分析录制的测试)

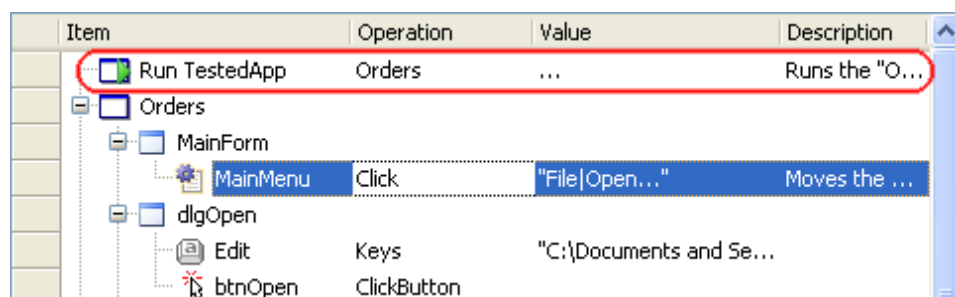
在录制完成之后，Testcomplete 会打开录制完成的关键字测试（可供编辑）并把测试的内容显示在 KeywordTest 编辑器上：



录制完成的测试与上图显示的测试非常类似。在你实际的测试中和会和这个例子会稍有出入。例如，如果你录制的是 C++ 或者 Delphi 写的应用程序，它可能会包括其他的对象名或者窗口索引。

测试指令取决于你在录制过程中怎样操作 Orders 应用程序。我们把测试指令称作**操作(operations)**。

- 在测试里执行的第一个操作是运行被测程序（Run TestedApp）。它在关键字测试中启动被测程序。（在我们的例子里，是 Orders 这个应用程序。）当 Testcomplete 检测到用户在 Testcomplete 用户界面或者在操作系统界面的其他地方运行被测程序，它会自动录制这个操作。



- 下一个操作与选择 File | Open 这个菜单有关。

Item	Operation	Value	Description
Run TestedApp	Orders	...	Runs the "O...
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the ...
dlgOpen			
Edit	Keys	"C:\Documents and Se...	
btnOpen	ClickButton		

- 接下来是一连串模拟你在打开文件对话框、应用程序主窗口、订单界面的操作：

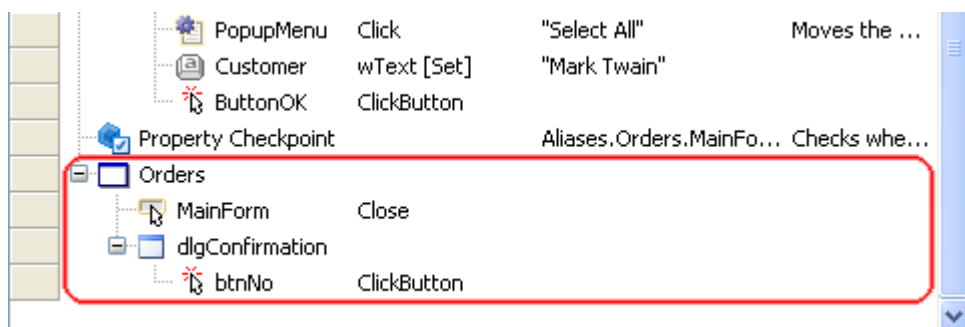
Item	Operation	Value	Description
Orders			
MainForm			
MainMenu	Click	"File Open..."	Moves the ...
dlgOpen			
Edit	Keys	"C:\Documents and Se...	
btnOpen	ClickButton		
MainForm			
OrdersView	ClickItem	"Samuel Clemens", 0	
ToolBar	ClickItem	5, False	
OrderForm			
Customer	Click	104, 9	
Customer	ClickR	104, 9	
PopupMenu	Click	"Select All"	Moves the ...
Customer	wText [Set]	"Mark Twain"	
ButtonOK	ClickButton		
Property Checkpoint		Aliases.Orders.MainFo...	Checks whe...
Orders			
MainForm	Close		

请查看 *Simulating User Actions* 这一帮助文档来获取更多关于模拟鼠标事件、键盘输入及其他动作的信息。

- 然后是在测试录制是添加进去的比较操作：

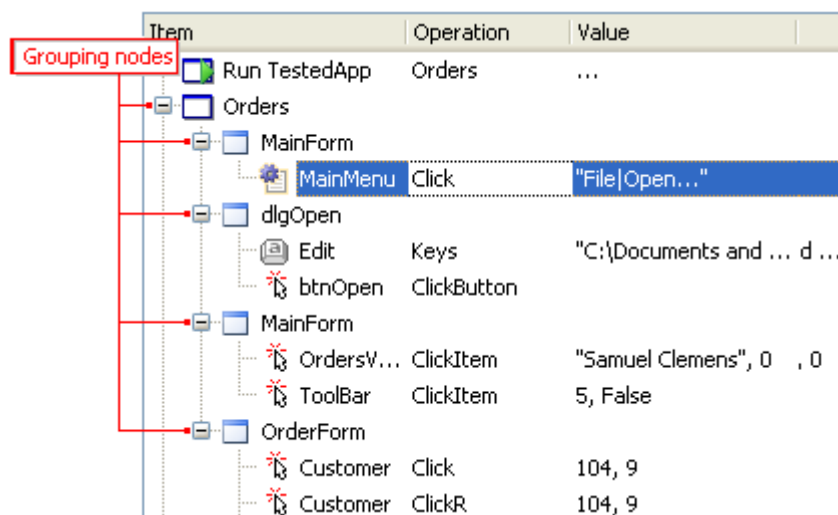
PopupMenu	Click	"Select All"	Moves the ...
Customer	wText [Set]	"Mark Twain"	
ButtonOK	ClickButton		
Property Checkpoint		Aliases.Orders.MainFo...	Checks whe...
Orders			
MainForm	Close		
dlgConfirmation			

- 最后，是关闭 Orders 应用程序和模拟点击对话框上 “No” 按钮的操作：



如你所见，Testcomplete 自动把你操作进程和窗体的步骤分组组织起来。分组的测试体系更容易理解，同时提供了一些有关被测对象层次结构的信息。

- 我们录制了在一个进程（*Orders*）的用户操作。所以，我们只有一个进程组节点，你在进程窗口和控件上的所有模拟操作都包含进去。我们在 *Orders* 进程的窗口和控件上执行的操作都被组织成一系列的“window”节点：



你可能会注意到被测进程及其窗口、控件的名称与我们之前在对象浏览器（Object Browser）看到的名称不一样。例如，在对象浏览器里我们看到的被测进程名称是 *Process("Orders")*，但在测试中它的名称是 *Orders*；主窗口的名称是 *WinFormsObject("MainForm")*，但在测试里它叫 *MainForm*，依此类推。

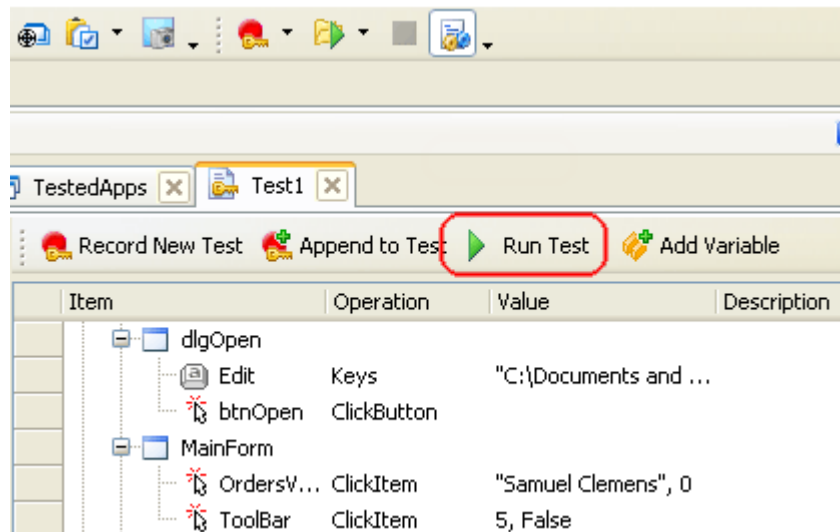
这个逻辑背后的原理是：Testcomplete默认在录制测试脚本的时候会为你的被测对象生成和使用自定义名称。生成和分配自定义名称，这个动作称为命名映射（*name mapping*）。Testcomplete之所以这么做，是因为默认的名称可能会难以理解。要确定哪个窗体和控件对应哪个名称可能会比较困难。使用命名映射使得测试更容易理解和更加可靠。请参考 *Name Mapping* 帮助文档来获取更多信息。

6. Running the Recorded Test（执行测试脚本）

现在我们可以运行这个测试例子，看看一下 Testcomplete 怎样去模拟用户操作。在回放录制的测试之前，请确保当前启动程序的初始条件和录制测试的初始条件是相同的。例如，测试几乎都需要被测程序处于运行状态。所以，在模拟用户的操作之前，你必须启动被测程序。在我们的例子中，测试开始前，我们用 *Run TestedApp* 这个操作来启动我们的被测程序，所以我们的测试会自动为我们启动被测程序。或

者，你可以手动从 Testcomplete 的集成开发环境（IDE）里面启动被测程序。

- 点击测试编辑器工具栏的  **Run Test** 来回放录制的测试：





测试引擎会自动把 Testcomplete 的窗口最小化，并启动测试指令。在我们的例子里，这个测试会简单的重复你之前录制的操作。

注：不要在测试过程中移动鼠标或者按下键盘。你这些行为可能会干涉到 Testcomplete 模拟的测试，导致这个测试出现问题。

在测试执行结束之后，Testcomplete 会还原它的窗口并显示测试结果。在下一节里我们会分析它们。

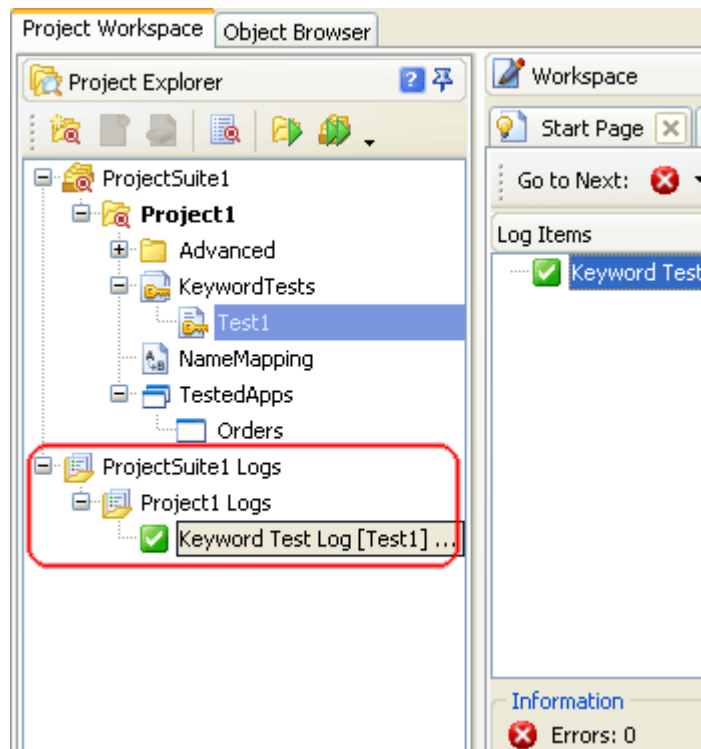
运行测试的一些注意事项：

- 之前创建的测试并非经过编译而用于测试的可执行文件。你直接从 Testcomplete 里运行测试。如果想在没有安装 Testcomplete 的计算机上执行测试，你可以使用名为 TestExecute 的共享工具。请看 *Connected and Self-Testing Applications* 这节帮助文档以获取更多信息。
- 在你开始测试之后，Testcomplete 会一直运行测试命令直到测试停止。你可以点击测试引擎工具栏上的  **Stop** 按钮或者点击主菜单上的 **Test | Stop**。
- 你可以点击  **Pause** 来暂停测试。在暂停中，你可以按需做任何操作。例如，你可以浏览测试日志或者通过 Testcomplete 的 Watch List 、 Locals panel 或 Evaluate dialog 检查测试变量和对象。（看 *Debugging Tests* 这一节帮助文档）
- 我们使用测试编辑器工具栏上的 **Run Test** 按钮来启动测试。这只是其中一种执行测试的方法（还有很多种其他方法）。你也可以在项目浏览器或其他测试里运行测试。你也可以使用项目编辑器下的 **Test Items** 页来执行批量的测试。
请查看 *Running Tests* 文档获取更多的信息。

7. Analyzing Test Results（分析测试结果）

Testcomplete 的测试日志保存了测试中的所有操作。指向测试结果的链接显示

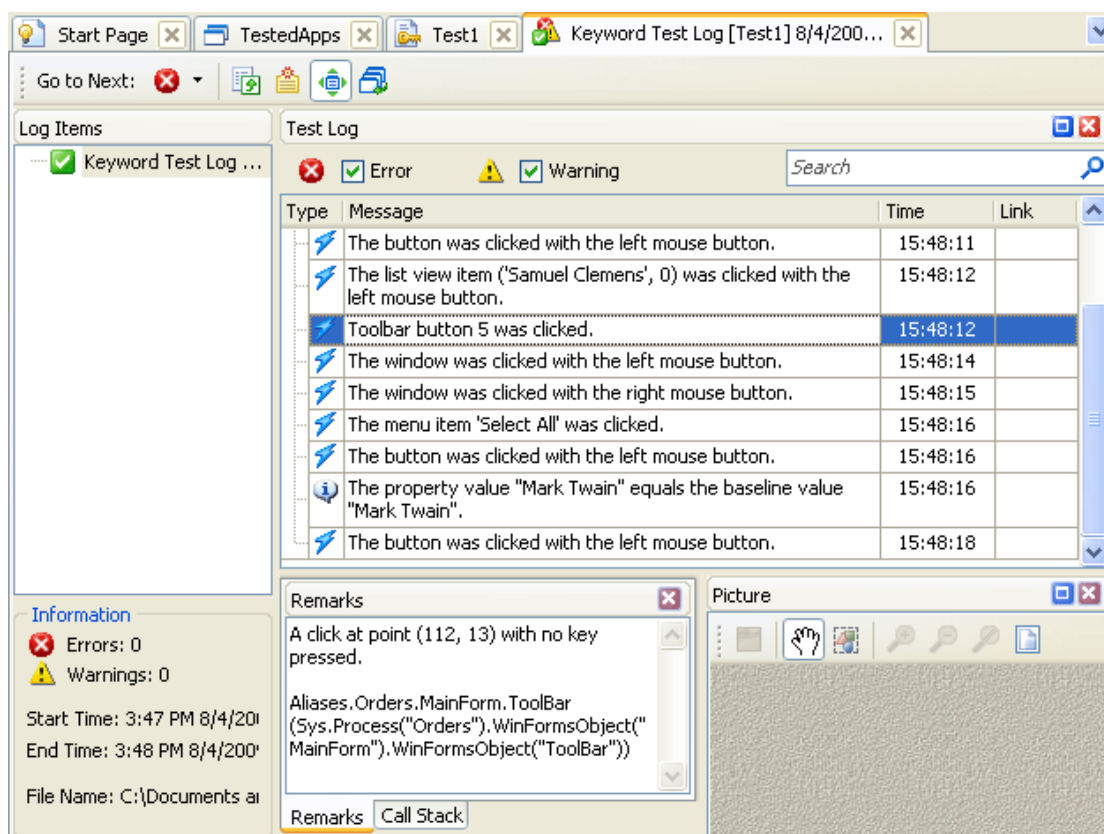
在项目浏览器界面的 **ProjectSuite1 Log | Orders Log** 节点下。这是查看项目或则项目组测试历史的主要工作区。每一个节点都与对应的测试相关。节点左边的图标指明了它对应的测试通过与否：



注意到 Testcomplete 会在测试结束之后自动添加节点。也就是，测试在执行的时候测试结果是会显示的（你可以通过暂停测试来查看中间结果）。

因为到目前为止我们只执行了一次测试，我们在项目浏览器里只有一个日志节点。默认下，Testcomplete 会在工作区自动打开这个节点的内容。你也可以随时查看日志。在项目浏览器右击所选的测试结果，选择菜单里的 **Open**。

- 在我们的例子里，日志是这样的：



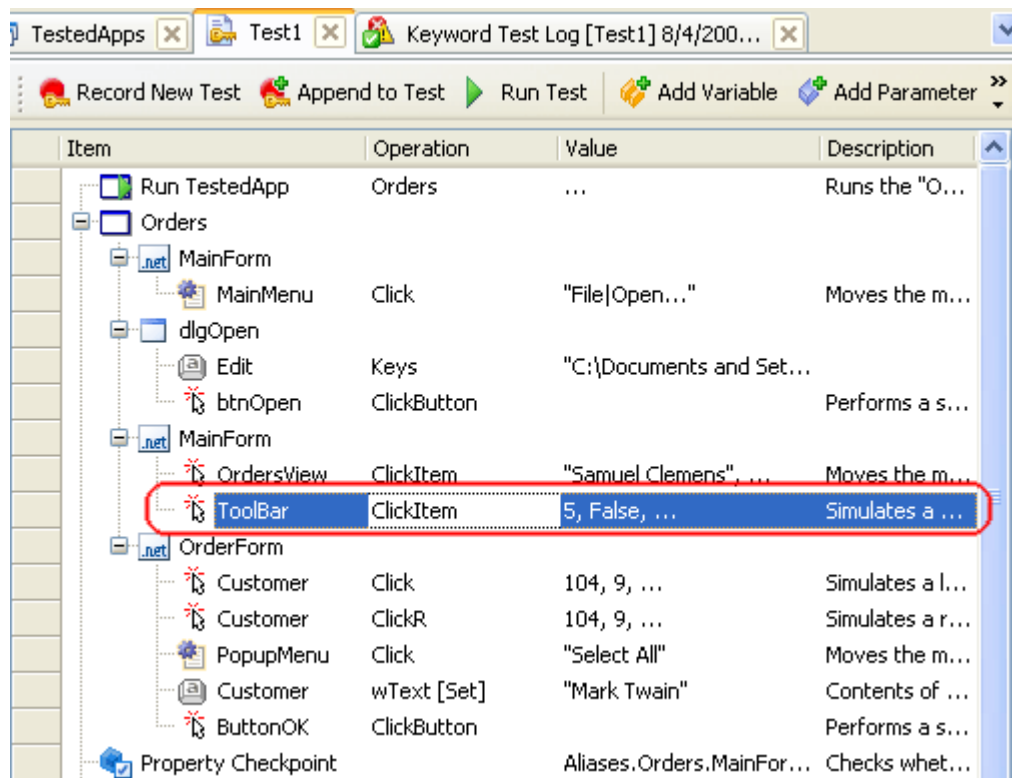
日志窗口显示了一次测试的结果。左边的窗口是一个已运行测试的树形结构；每个测试的节点都可以选择，以查看它们的结果。在我们的例子中，我们只运行了一个测试，所以这颗树只有一个节点。节点的图标显示了测试是否通过。

测试日志包括错误、警告、提示信息和其他类型的信息。左边的图标显示了信息的类型。使用信息列表顶上的单选框你可以按类隐藏或者显示信息。

Testcomplete 可能会提供附加的文本或者图像信息。点击对应的信息，并查看 **Remarks** 和 **Picture** 子窗口，你就能找到它们。例如，在上图 **Remarks** 子窗口里显示了如下附加信息 “Toolbar button 5 was clicked”。

日志的 **Call Stack** 子窗口显示了（能触发写日志行为的）测试调用的层次。

- 双击对应的日志信息来查看（触发写日志的）测试操作。Testcomplete 会在编辑器里打开相应的关键字测试，并把对应的操作高亮显示。例如，如果你在日志里双击“Toolbar button 5 was clicked”这条信息，Testcomplete 会把执行这个测试操作高亮显示出来：



请看 *Test Log* 以获取更多信息。

注：这里我们讨论的日志是 Testcomplete 里的关键字测试和脚本所生成的。其他类型的测试可能在结构上会有所不同。例如，在 HTTP 压力测试，生成的测试日志里会包含一些有关虚用户、连接、模拟请求和响应的表格。如需获取这些日志的更多细节信息，请查看相关项目项（project item）的描述，或者单击日志页面并按下 F1 。

TestComplete 数据驱动测试

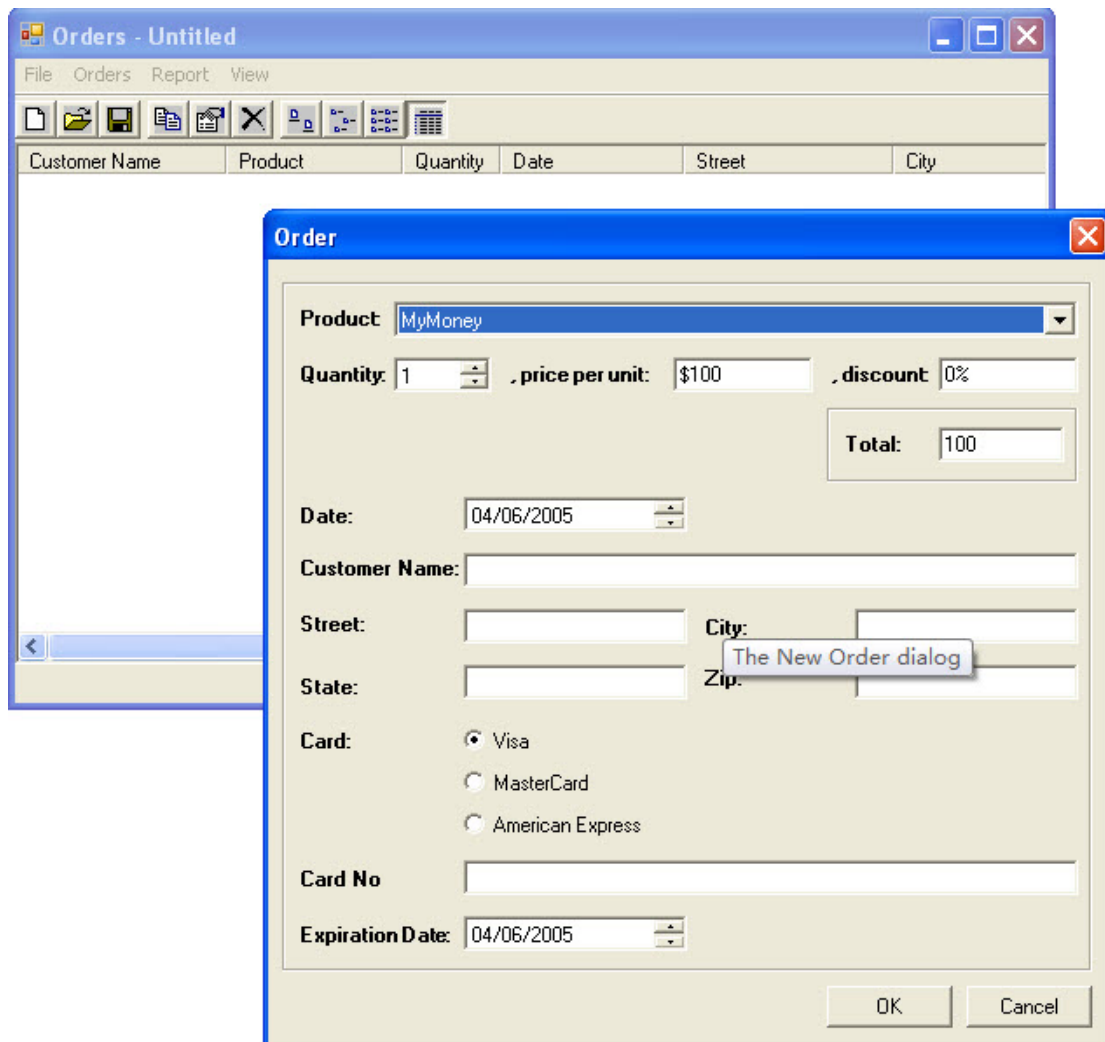
数据驱动测试的特点是把测试脚本和命令与测试数据分离。通常用一个表来存储真实的测试数据。Excle 表、数据库表、文本文件和数组都可以用作数据的载体。

这份教材演示了如果在Testcomplete下进行数据驱动测试。为了达到演示的目的，我么使用 **Orders**这个程序作为例子。你可以在以下路径找到这个例子：

<TestComplete Samples>\Open Apps\OrdersDemo\C#\bin\Debug\Orders.exe

这个应用程序显示了一个订单列表，允许用户新增、修改和删除订单。

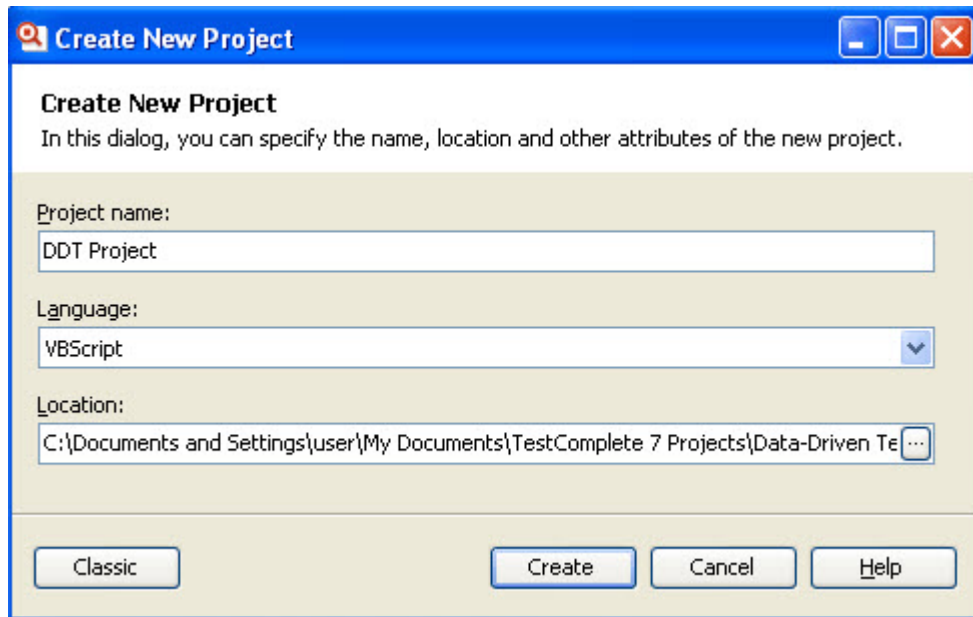
在主菜单栏选择Orders | New Order来创建一个新订单。这会调出一个订单窗口，里面含有用户自定义输入和自动计算相应数值的文本框。我们测试的目的是从存储文件里读取数据，模拟数据输入到订单窗体，然后检查程序Price Per Unit、 Discount 和 Total文本框显示的值有没有计算正确。



注：我们会采用 C#版的 Orders 程序来进行演示。如果你决定测试其他语言开发的程序，那么你就对标识窗体控件的脚本语句作出调整。

1.创建项目组（1: Creating a Project Suite）

这里我们会教会你创建整套数据驱动测试的解决方案。在你创建项目的时候会自动创建项目组。点击项目浏览器的**Add New Project**按钮来创建项目。弹出Create New Project对话框：



为这个新项目起个名字，选择你想要用的脚本语言，然后选择项目的保存路径，最后点击 **Create**，项目创建成功。

现在，你需要把Orders加进测试列表里。在项目浏览器里面选择**TestedApps**这个项目项，点击项目浏览器工具栏上的**Add Child Item**。在**Add Tested Application**对话框里选择Orders应用程序的可执行文件，点击**Open**。你可以在以下路径找到可执行文件：

<TestComplete Samples>\Open Apps\OrdersDemo\C#\bin\Debug\Orders.exe

上面提到的项目组案例，你也可以在以下路径找到：

<TestComplete Samples>\Scripts\DDT\DDTSuite.pjs

2. 存储测试数据（Creating a Data Storage）

为了实现数据驱动测试，你必须定义好测试里面要用到的数据。Excel 表、数据库表、文本文件和数组都能够存储测试数据。我们使用 Excel2007 来存储测试数据。表格存储了以下数据：

Name	Product	Quantity	Date	Street	City	State	ZIP	Credit Card	Credit Card No	Expiration Date	Price	Discount	Total
John Smith Jr	MyMoney	1	05/07/2007	12, Orange Blvd	Grovetown, CA	US	111155	AE	555777555888	11/26/2008	\$100	0%	100
Clare Jefferson	MyMoney	10	05/12/2007	23, Owk Street	Grovetown, CA	US	111155	MasterCard	444777333888	09/14/2008	\$100	8%	920
Susan McLaren	MyMoney	50	05/12/2000	7, Flower Street	Earlcastle	Great Britain	112255	AE	444555333111	10/21/2008	\$100	8%	4600
Charles Dodgeson	FamilyAlbum	1	05/12/2000	45, Stone st.	Bringtone, TX	US	662233	AE	444555333111	10/30/2008	\$80	0%	80
Steve Johns	FamilyAlbum	10	12/14/2003	17, Park Avenue	Salmon Island	Canada	665511	MasterCard	444222333888	10/26/2008	\$80	15%	680
Samuel Clemens	FamilyAlbum	50	12/12/1999	3, Garden st.	Hillsberry, UT	US	667711	MasterCard	333222555888	11/23/2008	\$80	15%	3400
Bob Feather	ScreenSaver	1	12/12/2005	14, North av.	Milltown, WI	US	668899	VISA	333444555444	12/11/2008	\$20	0%	20
Mark Smith	ScreenSaver	10	10/10/2006	9, Maple Valley	Whitestone, British Columbia	Canada	668899	VISA	333444555444	12/11/2008	\$20	10%	180
Tom Hawks	ScreenSaver	50	11/13/2002	10, Rock av.	Milltown, WI	US	334433	MasterCard	111333333444	1/15/2009	\$20	10%	900

表格里面包含了一个订单要用到的值，用于模拟输入操作。请注意，上面的数据不仅仅含有输入数据，而且还包含了验证所用到的值（见上边的最后三列）。这些数据是用作比较的基线数据。

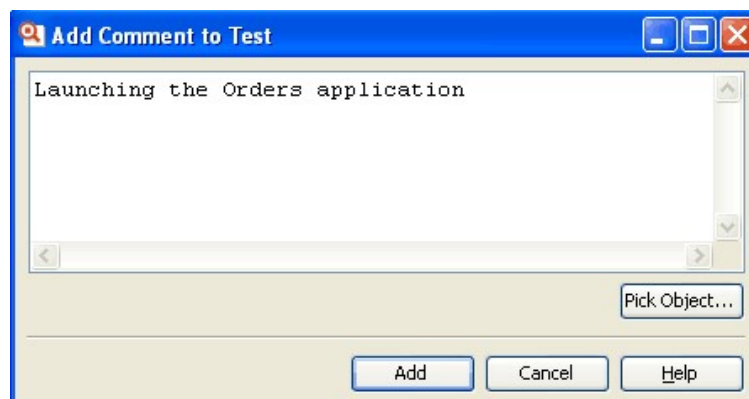
上面的数据表你可以在一下路径找到： <TestComplete Samples>\DDT\TestBook.xlsx

3. 建立一个迭代脚本（Creating One Script Iteration）

在这个步骤里，我们会录制一个执行一次测试迭代的脚本。这个脚本会启动测试程序、新增订单、验证订单生成数据的正确性，然后关闭测试程序。在录制脚本的过程中，我们会对实现不同功能的代码段作出注释。


下面是测试脚本的步骤细节：

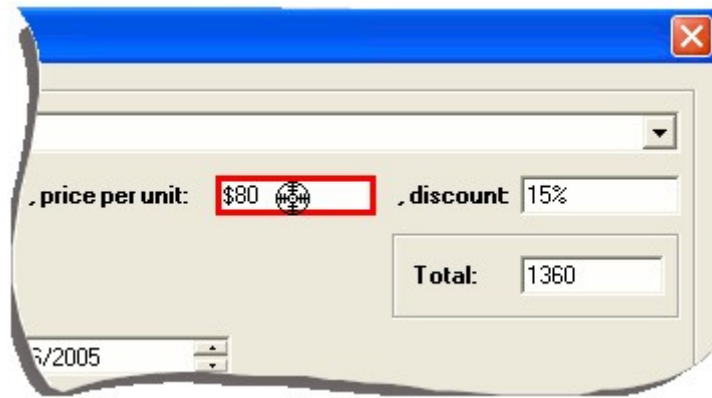
1. 在Testcomplete主菜单选择**Test | Record | Record Script**。
2. 加上“*Launching the Orders application*”这句注释：
 - 点击录制工具栏的**Add Comment to Test**按钮。弹出[Add Comment to Test](#)对话框。
 - 输入“*Launching the Orders application*”，然后点击**Add**按钮。



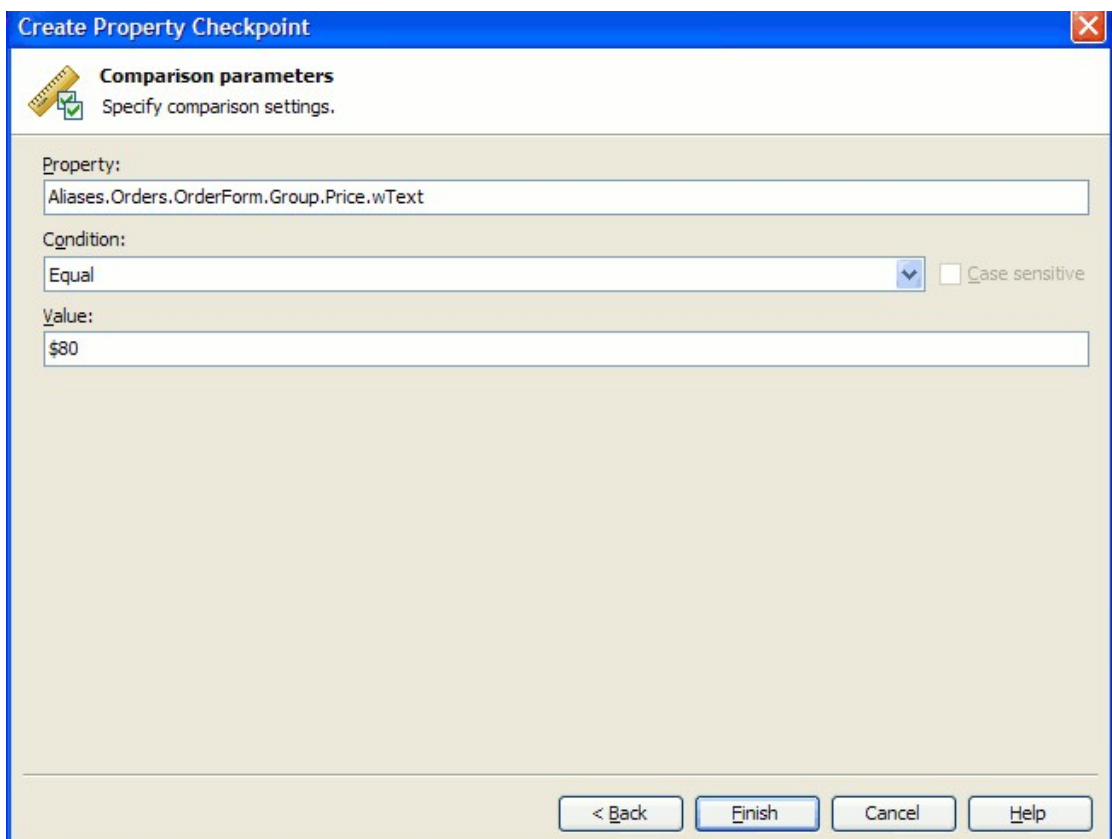
3. 点击录制工具栏上的**Run Tested Application**按钮。
4. 在上述测试里加入注释“*Opening the Order form*”。
5. 切换到Orders应用程序，点击**Orders | New order**菜单。打开订单窗口。
6. 在测试中加入注释：“*Populating the Order form*”。
7. 在订单里输入以下数值：

Field	Value
Product	Select FamilyAlbum.
Quantity	20
Date	05/06/2005
Customer Name	John Smith Jr
Street	12, Orange Blvd
City	Grovetown, CA
State	US
Zip	111155
Card	Select MasterCard.
Card No	555777555888
Expiration Date	05/06/2005

8. 向测试脚本添加注释“*Creating checkpoints*”。
9. 创建一个验证订单窗体**Price per Unit**的检查点：
 - 在录制工具栏选择**Create Property Checkpoint**。弹出[Create Property Checkpoint](#)向导。
 - 拖动  **Finder Tool** 图标到**Price per Unit**文本框里：




- 弹出 [Create Project Item](#) 对话框。利用这个对话框，在项目创建 [Name Mapping item](#)。然后 Name Mapping project item 节点会添加到项目浏览器那里。这个节点保存了测试里面用到的客户姓名。Testcomplete 会在测试录制的过程中自动映射这些姓名（因为 [Map object names automatically](#) 这个设置是默认打开的）。
- 点击 Create Property Checkpoint 向导的 next 按钮；
- 在 object properties 列表里选择 wText 属性，点击 next；
- 确保 Equals 条件被选上：



点击 **Finish** 按钮来结束向导。Testcomplete 为测试生成合适的检查点。

10. 用同样的方法，你也可以为 **Discount** 和 **Total** 文本框的 wText 属性值设置检查点。
11. 添加注释 "Closing the Order form"。
12. 点击订单窗体的 OK 按钮。

13. 添加注释“*Closing the Orders application*”。
14. 关闭 Orders 程序。一个弹出框问你是否保存所有修改。
15. 点击 **No** 按钮。
16. 点击录制工具栏的  **Stop** 按钮。

录制的脚本如下：

VB Script

Sub Test1

```
Dim orders
Dim mainForm
Dim orderForm
Dim groupBox
Dim numericUpDown
Dim textBox

' Launching the Orders application
TestedApps.RunAll

' Opening the Order form
Set orders = Aliases.Orders
Set mainForm = orders.MainForm
Call mainForm.MainMenu.Click("Orders|New order...")

' Populating the Order form
Set orderForm = orders.OrderForm
Set groupBox = orderForm.Group
Call groupBox.ProductNames.ClickItem("FamilyAlbum")
Set numericUpDown = groupBox.Quantity
Call numericUpDown.Click(37, 9)
numericUpDown.wValue = 20
groupBox.Date.wDate = "5/6/2005"
Set textBox = groupBox.Customer
Call textBox.Click(39, 7)
textBox.wText = "John Smith Jr"
Set textBox = groupBox.Street
Call textBox.Click(66, 7)
textBox.wText = "12, Orange Blvd"
Set textBox = groupBox.City
Call textBox.Click(72, 9)
textBox.wText = "Grovetown, CA"
Set textBox = groupBox.State
Call textBox.Click(98, 9)
textBox.wText = "US"
Set textBox = groupBox.Zip
Call textBox.Click(31, 12)
textBox.wText = "111155"
```

```

groupBox.MasterCard.ClickButton
Set textBox = groupBox.CardNo
Call textBox.Click(53, 12)
textBox.wText = "555777555888"
groupBox.ExpDate.wDate = "5/6/2005"
' Creating checkpoints
Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, "$80",
False)
Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual,
"15%", False)
Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, "1360", False)
' Closing the Order form
orderForm.ButtonOK.ClickButton
' Closing the application
mainForm.Close
orders.dlgConfirmation.btnNo.ClickButton
End Sub

```

Jscript:

```

function Test1()
{
    var orders;
    var mainForm;
    var orderForm;
    var groupBox;
    var numericUpDown;
    var textBox;
    // Launching the Orders application
    TestedApps.RunAll();
    // Opening the Order form
    orders = Aliases.Orders;
    mainForm = orders.MainForm;
    mainForm.MainMenu.Click("Orders|New order...");
    // Populating the Order form
    orderForm = orders.OrderForm;
    groupBox = orderForm.Group;
    groupBox.ProductNames.ClickItem("FamilyAlbum");
    numericUpDown = groupBox.Quantity;
    numericUpDown.Click(8, 7);
    numericUpDown.wValue = 20;
    groupBox.Date.wDate = "5/6/2005";
    textBox = groupBox.Customer;
    textBox.Click(69, 12);
}

```

```

    textBox.wText = "John Smith Jr";
    textBox = groupBox.Street;
    textBox.Click(4, 8);
    textBox.wText = "12, Orange Blvd";
    textBox = groupBox.City;
    textBox.Click(12, 14);
    textBox.wText = "Grovetown, CA";
    textBox = groupBox.State;
    textBox.Click(21, 6);
    textBox.wText = "US";
    textBox = groupBox.Zip;
    textBox.Click(14, 14);
    textBox.wText = "111155";
    groupBox.MasterCard.ClickButton();
    textBox = groupBox.CardNo;
    textBox.Click(35, 15);
    textBox.wText = "555777555888";
    groupBox.ExpDate.wDate = "5/6/2005";
    // Creating checkpoints
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, "$80",
false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual,
"15%", false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText, cmpEqual,
"1360", false);
    // Closing the Order form
    orderForm.ButtonOK.ClickButton();
    // Closing the application
    mainForm.Close();
    orders.dlgConfirmation.btnNo.ClickButton();
}

```

DelphiScript:

```

procedure Test1;
    var orders : OleVariant;
    var mainForm : OleVariant;
    var orderForm : OleVariant;
    var groupBox : OleVariant;
    var numericUpDown : OleVariant;
    var textBox : OleVariant;
begin
    // Launching the Orders application
    TestedApps.RunAll;
    // Opening the Order form

```

```

orders := Aliases.Orders;
mainForm := orders.MainForm;
mainForm.MainMenu.Click('Orders|New order...');
// Populating the Order form
orderForm := orders.OrderForm;
groupBox := orderForm.Group;
groupBox.ProductNames.ClickItem('FamilyAlbum');
numericUpDown := groupBox.Quantity;
numericUpDown.Click(37, 9);
numericUpDown.wValue := 20;
groupBox.Date.wDate := '5/6/2005';
textBox := groupBox.Customer;
textBox.Click(67, 17);
textBox.wText := 'John Smith Jr';
textBox := groupBox.Street;
textBox.Click(31, 5);
textBox.wText := '12, Orange Blvd';
textBox := groupBox.City;
textBox.Click(47, 15);
textBox.wText := 'Grovettown, CA';
textBox := groupBox.State;
textBox.Click(82, 4);
textBox.wText := 'US';
textBox := groupBox.Zip;
textBox.Click(10, 8);
textBox.wText := '111155';
groupBox.MasterCard.ClickButton;
textBox := groupBox.CardNo;
textBox.Click(46, 9);
textBox.wText := '555777555888';
groupBox.ExpDate.wDate := '5/6/2005';
// Creating checkpoints
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, '$80',
false);
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual, '15%',
false);
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText, cmpEqual,
'1360', false);
// Closing the Order form
orderForm.ButtonOK.ClickButton;
// Closing the application
mainForm.Close;
orders.dlgConfirmation.btnNo.ClickButton;
end;

```

C++Script, C#Script:

```
function Test1()
{
    var orders;
    var mainForm;
    var orderForm;
    var groupBox;
    var numericUpDown;
    var textBox;
    // Launching the Orders application
    TestedApps["RunAll"]();
    // Opening the Order form
    orders = Aliases["Orders"];
    mainForm = orders["MainForm"];
    mainForm["MainMenu"]["Click"]("Orders|New order...");
    // Populating the Order form
    orderForm = orders["OrderForm"];
    groupBox = orderForm["Group"];
    groupBox["ProductNames"]["ClickItem"]("FamilyAlbum");
    numericUpDown = groupBox["Quantity"];
    numericUpDown["Click"](37, 9);
    numericUpDown["wValue"] = 20;
    groupBox["Date"]["wDate"] = "5/6/2005";
    textBox = groupBox["Customer"];
    textBox["Click"](43, 7);
    textBox["wText"] = "John Smith Jr";
    textBox = groupBox["Street"];
    textBox["Click"](31, 11);
    textBox["wText"] = "12, Orange Blvd";
    textBox = groupBox["City"];
    textBox["Click"](24, 6);
    textBox["wText"] = "Grovetown, CA";
    textBox = groupBox["State"];
    textBox["Click"](31, 8);
    textBox["wText"] = "US";
    textBox = groupBox["Zip"];
    textBox["Click"](24, 17);
    textBox["wText"] = "111155";
    groupBox["MasterCard"]["ClickButton"]();
    textBox = groupBox["CardNo"];
    textBox["Click"](19, 7);
    textBox["wText"] = "555777555888";
    groupBox["ExpDate"]["wDate"] = "5/6/2005";
```

```

// Creating checkpoints
aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Price"]["wText"],
cmpEqual, "$80", false);
aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Discount"]["wText"],
cmpEqual, "15%", false);

aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["groupBox1"]["Total"]["wText"], cmpEqual, "1360", false);
// Closing the Order form
orderForm["ButtonOK"]["ClickButton"]();
// Closing the application
mainForm["Close"]();
orders["dlgConfirmation"]["btnNo"]["ClickButton"]();
}

```

4.修改脚本、定制输入值（Modifying Script and Assigning Input Values）

把录制好的脚本拆分成不同的子程序：

我们看一下刚才录制好的脚本。注释划分了不同的测试步骤：执行 **Orders** 程序、打开订单窗体、弹出窗体、验证订单字段的值、关闭窗体、关闭程序。我们现在可以把不同的代码段划分为不同的子程序。由 **Main** 函数来调用这些子程序。

下面是一系列的子程序：

- OpenForm子程序用来打开订单窗体。拷贝注释"*Opening the Order form*" 后面的代码到OpenForm子程序里面，如下所示：

VBScript:

Sub OpenForm

Set orders = Aliases.Orders

Set mainForm = orders.MainForm

Call mainForm.MainMenu.Click("Orders|New order...")

End Sub

Jscript:

- **function** OpenForm()
{
orders = Aliases.Orders;
mainForm = orders.MainForm;
mainForm.MainMenu.Click("Orders|New order...");
}

DelphiScript:

- **procedure** OpenForm;
 begin
 orders := Aliases.Orders;
 mainForm := orders.MainForm;
 mainForm.MainMenu.Click('Orders|New order...');
 end;

C++Script, C#Script:

- **function** OpenForm()
 {
 orders = Aliases["Orders"];
 mainForm = orders["MainForm"];
 mainForm["MainMenu"]["Click"]("Orders|New order...");
 }

● PopulateForm子程序用来弹出订单窗体。拷贝注释"*Populating the Order form*"后面的代码到PopulateForm子程序, 如下:

VBScript:

Sub PopulateForm

Set orderForm = orders.OrderForm

Set groupBox = orderForm.Group

Call groupBox.ProductNames.ClickItem("FamilyAlbum")

Set numericUpDown = groupBox.Quantity

Call numericUpDown.Click(37, 9)

 numericUpDown.wValue = 20

 groupBox.Date.wDate = "5/6/2005"

Set textBox = groupBox.Customer

Call textBox.Click(39, 7)

 textBox.wText = "John Smith Jr"

Set textBox = groupBox.Street

Call textBox.Click(66, 7)

 textBox.wText = "12, Orange Blvd"

Set textBox = groupBox.City

Call textBox.Click(72, 9)

 textBox.wText = "Grovetown, CA"

Set textBox = groupBox.State

Call textBox.Click(98, 9)

 textBox.wText = "US"

Set textBox = groupBox.Zip

Call textBox.Click(31, 12)

 textBox.wText = "111155"

```

groupBox.MasterCard.ClickButton
Set textBox = groupBox.CardNo
Call textBox.Click(53, 12)
textBox.wText = "555777555888"
groupBox.ExpDate.wDate = "5/6/2005"
End Sub

```

Jscript:

```

function PopulateForm()
{
    orderForm = orders.OrderForm;
    groupBox = orderForm.Group;
    groupBox.ProductNames.ClickItem("FamilyAlbum");
    numericUpDown = groupBox.Quantity;
    numericUpDown.Click(8, 7);
    numericUpDown.wValue = 20;
    groupBox.Date.wDate = "5/6/2005";
    textBox = groupBox.Customer;
    textBox.Click(69, 12);
    textBox.wText = "John Smith Jr";
    textBox = groupBox.Street;
    textBox.Click(4, 8);
    textBox.wText = "12, Orange Blvd";
    textBox = groupBox.City;
    textBox.Click(12, 14);
    textBox.wText = "Grovetown, CA";
    textBox = groupBox.State;
    textBox.Click(21, 6);
    textBox.wText = "US";
    textBox = groupBox.Zip;
    textBox.Click(14, 14);
    textBox.wText = "111155";
    groupBox.MasterCard.ClickButton();
    textBox = groupBox.CardNo;
    textBox.Click(35, 15);
    textBox.wText = "555777555888";
    groupBox.ExpDate.wDate = "5/6/2005";
}

```

DelphiScript:

```

procedure PopulateForm;
begin
    orderForm := orders.OrderForm;
    groupBox := orderForm.Group;

```



```

groupBox.ProductNames.ClickItem('FamilyAlbum');
numericUpDown := groupBox.Quantity;
numericUpDown.Click(37, 9);
numericUpDown.wValue := 20;
groupBox.Date.wDate := '5/6/2005';
textBox := groupBox.Customer;
textBox.Click(67, 17);
textBox.wText := 'John Smith Jr';
textBox := groupBox.Street;
textBox.Click(31, 5);
textBox.wText := '12, Orange Blvd';
textBox := groupBox.City;
textBox.Click(47, 15);
textBox.wText := 'Grovetown, CA';
textBox := groupBox.State;
textBox.Click(82, 4);
textBox.wText := 'US';
textBox := groupBox.Zip;
textBox.Click(10, 8);
textBox.wText := '111155';
groupBox.MasterCard.ClickButton;
textBox := groupBox.CardNo;
textBox.Click(46, 9);
textBox.wText := '555777555888';
groupBox.ExpDate.wDate := '5/6/2005';
end;

```

C++Script, C#Script:

```

function PopulateForm()
{
    orderForm = orders["OrderForm"];
    groupBox = orderForm["Group"];
    groupBox["ProductNames"]["ClickItem"]("FamilyAlbum");
    numericUpDown = groupBox["Quantity"];
    numericUpDown["Click"](37, 9);
    numericUpDown["wValue"] = 20;
    groupBox["Date"]["wDate"] = "5/6/2005";
    textBox = groupBox["Customer"];
    textBox["Click"](43, 7);
    textBox["wText"] = "John Smith Jr";
    textBox = groupBox["Street"];
    textBox["Click"](31, 11);
    textBox["wText"] = "12, Orange Blvd";
    textBox = groupBox["City"];
}

```

```

        textBox["Click"] (24, 6);
        textBox["wText"] = "Grovetown, CA";
        textBox = groupBox["State"];
        textBox["Click"] (31, 8);
        textBox["wText"] = "US";
        textBox = groupBox["Zip"];
        textBox["Click"] (24, 17);
        textBox["wText"] = "111155";
        groupBox["MasterCard"]["ClickButton"] ();
        textBox = groupBox["CardNo"];
        textBox["Click"] (19, 7);
        textBox["wText"] = "555777555888";
        groupBox["ExpDate"]["wDate"] = "5/6/2005";
    }

```

- Checkpoint 子程序用来检查订单窗体的值是否正确:

VBScript:

```

    Sub Checkpoint
        Call
        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, "$80", False)
        Call
        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual, "15%", False)
        Call
        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.groupBox1.Total.wText, cmpEqual, "1360", False)
    End Sub

```

Jscript:

```

    function Checkpoint ()
    {

        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, "$80", false);

        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual, "15%", false);

        aqObject.CompareProperty (Aliases.Orders.OrderForm.Group.groupBox1.Total.wText, cmpEqual, "1360", false);
    }

```

DelphiScript:

```

    procedure Checkpoint;

```

```
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual, '$80', false);
```

```
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText, cmpEqual, '15%', false);
```

```
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText, cmpEqual, '1360', false);  
end;
```

C++Script, C#Script:

```
function Checkpoint()  
{
```

```
aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Price"]["wText"], cmpEqual, "$80", false);
```

```
aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Discount"]["wText"], cmpEqual, "15%", false);
```

```
aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["groupBox1"]["Total"]["wText"], cmpEqual, "1360", false);  
}
```

- CloseForm 子程序用来关闭订单窗体:

VBScript:

```
Sub CloseForm  
    orderForm.ButtonOK.ClickButton  
End Sub
```

Jscript:

```
function CloseForm()  
{  
    orderForm.ButtonOK.ClickButton();  
}
```

DelphiScript:

```
procedure CloseForm;  
begin  
    orderForm.ButtonOK.ClickButton;  
end;
```

C++Script, C#Script:

```
function CloseForm()
```

```

        {
            orderForm["ButtonOK"]["ClickButton"] ();
        }
    }

```

- CloseApplication 子程序用来关闭 Orders 程序:

VBScript:

```

Sub CloseApplication
    mainForm.Close
    orders.dlgConfirmation.btnNo.ClickButton
End Sub

```

Jscript:

```

function CloseApplication()
{
    mainForm.Close();
    orders.dlgConfirmation.btnNo.ClickButton();
}

```

DelphiScript:

```

procedure CloseApplication;
begin
    mainForm.Close;
    orders.dlgConfirmation.btnNo.ClickButton;
end;

```

C++Script, C#Script:

```

function CloseApplication()
{
    mainForm["Close"] ();
    orders["dlgConfirmation"]["btnNo"]["ClickButton"] ();
}

```

创建 Main 函数:

我们按上述步骤创建完子程序。最后，我们创建一个 Main 函数来调用这些子程序:

VBScript:

```

Sub Main
    TestedApps.RunAll
    OpenForm
    PopulateForm
    Checkpoint
    CloseForm
    CloseApplication
End Sub

```

Jscript:

```
function Main()
{
    TestedApps.RunAll();
    OpenForm();
    PopulateForm();
    Checkpoint();
    CloseForm();
    CloseApplication();
}
```

DelphiScript:

```
procedure Main;
begin
    TestedApps.RunAll;
    OpenForm;
    PopulateForm;
    Checkpoint;
    CloseForm;
    CloseApplication;
End
```

C++Script, C#Script:

```
function Main()
{
    TestedApps["RunAll"]();
    OpenForm();
    PopulateForm();
    Checkpoint();
    CloseForm();
    CloseApplication();
}
```

修改 Main 函数的代码:

orders, mainForm, orderForm, groupBox 里面含有很多对象的引用，它们存放在子程序的变量里，这些子程序通过 Main 函数来调用。为了访问这些子程序的变量，你要在 Main 函数里对这些变量定义并赋值。在 OpenForm 子函数里，选择变量 orders 和 mainForm 的赋值语句；在 PopulateForm 子函数里，选择变量 orderForm 和 groupBox 的赋值语句。把这些变量的赋值语句剪切到 Main 函数里。

为了访问脚本里面的数据，你要把那些字符串（这些字符串是用来访问数据的[DDTDriver](#) 对象）放在子程序的开头。这些字符串定义了一个 DDTDriver 对象到 Driver 变量的引用。这

个变量必须在Main函数里定义。

➤ 留 意 一 下 注 册 表 *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Excel\FirstRowHasNames* 的键值。这个键值决定了表的第一行是表头还是数据。如果你把表的第一行作为表头，那么注册表的键值是 01。如果第一行是用作数据，注册表的键值是 00。如果有一行是非空的，那么它将成为第一行，否则，将会被忽略。请查看[Using DDT Drivers](#)获取更多信息。

为了多次执行测试，我们需要创建一个循环。循环体里要调用OpenForm、PopulateForm、Checkpoints和CloseForm子函数。[DDTDriver.EOF](#)属性为true时，循环结束。我们每次调用[DDTDriver.Next](#)方法来取得数据表中的一行数据，直到循环结束。
所有修改完成之后的 Main 函数如下：

VBScript

```
Sub Main
    Set Driver = DDT.ExcelDriver("../TestBook.xlsx", "TestSheet", True)
    TestedApps.RunAll
    Set orders = Aliases.Orders
    Set mainForm = orders.MainForm
    While Not Driver.EOF
        OpenForm
        Set orderForm = orders.OrderForm
        Set GroupBox = orderForm.Group
        PopulateForm
        Checkpoint
        CloseForm
        Driver.Next
    Wend
    CloseApplication
End Sub
```

JScript

```
function Main()
{
    Driver = DDT.ExcelDriver("../TestBook.xlsx", "TestSheet", true)
    TestedApps.RunAll();
    orders = Aliases.Orders;
    mainForm = orders.MainForm;
    while(!Driver.EOF())
    {
        OpenForm();
        orderForm = orders.OrderForm;
```

```

        groupBox = orderForm.Group;
        PopulateForm();
        Checkpoint();
        CloseForm();
        Driver.Next();
    }
    CloseApplication();
}

```

DelphiScript

```

procedure Main;
begin
    Driver := DDT.ExcelDriver(' ../TestBook.xlsx', 'TestSheet', true)
    TestedApps.RunAll;
    orders := Aliases.Orders;
    mainForm := orders.MainForm;
    while not Driver.EOF do
        begin
            OpenForm;
            orderForm := orders.OrderForm;
            groupBox := orderForm.Group;
            PopulateForm;
            Checkpoint;
            CloseForm;
            Driver.Next;
        end;
        CloseApplication;
    end

```

C++Script, C#Script

```

function Main()
{
    Driver = DDT["ExcelDriver"]("../TestBook.xlsx", "TestSheet", true)
    TestedApps["RunAll"]();
    orders = Aliases["Orders"];
    mainForm = orders["MainForm"];
    while (!Driver["EOF"]())
    {
        OpenForm();
        orderForm = orders["OrderForm"];
        groupBox = orderForm["Group"];
        PopulateForm();
        Checkpoint();
    }
}

```

```

        CloseForm();
        Driver["Next"]();
    }
    CloseApplication();
}

```

Main 函数差不多可以用了。你值需要粘贴输入参数来调用其他的子函数。

修改 OpenForm, PopulateForm, Checkpoint, CloseForm, CloseApplication 子程序

其他子函数里会用到 Main 函数里的变量值。你要声明每个子函数里面变量，作为它的输入参数（变量 numericUpDown 和 textBox 除外）。变量 numericUpDown 和 textBox 仅仅在子程序 PopulateForm 里用得上，所以它们要定义成局部变量。因此，这些子函数会拥有以下输入参数：OpenForm(mainForm), PopulateForm(groupBox), Checkpoint(), CloseForm(orderForm), CloseApplication(orders, mainForm)。但是 PopulateForm 和 Checkpoint 还有一个额外的输入参数，下一节会提到。

把变量 orders, mainForm, orderForm, groupBox 的赋值语句从 OpenForm and PopulateForm 子函数移动到 Main 函数。

获取测试数据

访问测试数据的驱动（Driver）已经在Main函数里面定义了。声明Driver变量作为PopulateForm和Checkpoint子函数的输入参数来使用这个驱动。为了将外部测试数据输入到订单窗体，你要修改以下子函数PopulateForm的代码。找到录制脚本里面定义好的数值，改为调用Driver.Value的属性值（每个值都对应一个索引），如下：

Field name	Recorded value	Driver.Value property
Customer	"John Smith Jr"	Driver.Value (0)
Product	"FamilyAlbum"	Driver.Value (1)
Quantity	"20"	Driver.Value (2)
Date	"06.05.2005"	Driver.Value (3)
Street	"12, Orange Blvd"	Driver.Value (4)
City	"Grovetown, CA"	Driver.Value (5)
State	"US"	Driver.Value (6)
ZIP	"111155"	Driver.Value (7)
Card No	"555777555888"	Driver.Value (9)
Expiration Date	"06.05.2005"	Driver.Value (10)

你还要改变子函数 groupBox.MasterCard.ClickButton 这一行代码，这一行用来点击按钮。把 WinFormsObject(Driver.Value(8))对象插入到 MasterCard。这允许你通过按钮的名字来点击。

按钮的名字在 **Driver.Value(8)**已经定义好了。

通过修改子函数 **Checkpoint()**的代码，来实现订单窗体的字段值和外部（读取）数据的比较。找到录制脚本里定义好的数值，改为 **Driver.Value** 属性值（每个值都有对应的索引值），如下表示：

Field name	Recorded value	Driver.Value property
Price per unit	"\$80"	Driver.Value (11)
Discount	"15%"	Driver.Value (12)
Total	"1360"	Driver.Value (13)

子函数 **PopulateForm** 和 **Checkpoint** 的所有修改如下：

VBScript

```
Sub PopulateForm(groupBox, Driver)
    Dim numericUpDown, textBox

    Call groupBox.ProductNames.ClickItem(Driver.Value(1))
    Set numericUpDown = groupBox.Quantity
    Call numericUpDown.Click(37, 9)
    numericUpDown.wValue = Driver.Value(2)
    groupBox.Date.wDate = Driver.Value(3)
    Set textBox = groupBox.Customer
    Call textBox.Click(39, 7)
    textBox.wText = Driver.Value(0)
    Set textBox = groupBox.Street
    Call textBox.Click(66, 7)
    textBox.wText = Driver.Value(4)
    Set textBox = groupBox.City
    Call textBox.Click(72, 9)
    textBox.wText = Driver.Value(5)
    Set textBox = groupBox.State
    Call textBox.Click(98, 9)
    textBox.wText = Driver.Value(6)
    Set textBox = groupBox.Zip
    Call textBox.Click(31, 12)
    textBox.wText = Driver.Value(7)
    groupBox.WinFormsObject(Driver.Value(8)).ClickButton
    Set textBox = groupBox.CardNo
    Call textBox.Click(53, 12)
    textBox.wText = Driver.Value(9)
    groupBox.ExpDate.wDate = Driver.Value(10)
End Sub
```

```

Sub Checkpoint(Driver)
    Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText,
cmpEqual, Driver.Value(11), False)
    Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,
cmpEqual, Driver.Value(12) , False)
    Call
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), False)
End Sub

```

JScript

```

function PopulateForm(groupBox, Driver)
{
    var numericUpDown, textBox;

    groupBox.ProductNames.ClickItem(Driver.Value(1));
    numericUpDown = groupBox.Quantity;
    numericUpDown.Click(8, 7);
    numericUpDown.wValue = Driver.Value(2);
    groupBox.Date.wDate = Driver.Value(3);
    textBox = groupBox.Customer;
    textBox.Click(69, 12);
    textBox.wText = Driver.Value(0);
    textBox = groupBox.Street;
    textBox.Click(4, 8);
    textBox.wText = Driver.Value(4);
    textBox = groupBox.City;
    textBox.Click(12, 14);
    textBox.wText = Driver.Value(5);
    textBox = groupBox.State;
    textBox.Click(21, 6);
    textBox.wText = Driver.Value(6);
    textBox = groupBox.Zip;
    textBox.Click(14, 14);
    textBox.wText = Driver.Value(7);
    groupBox.WinFormsObject(Driver.Value(8)).ClickButton();
    textBox = groupBox.CardNo;
    textBox.Click(35, 15);
    textBox.wText = Driver.Value(9);
    groupBox.ExpDate.wDate = Driver.Value(10);
}

```

```

function Checkpoint(Driver)
{

```

```

    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual,
Driver.Value(11), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,
cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), false);
}

```

DelphiScript

```

procedure PopulateForm(groupBox, Driver);
    var numericUpDown, textBox;
begin
    groupBox.ProductNames.ClickItem(Driver.Value(1));
    numericUpDown := groupBox.Quantity;
    numericUpDown.Click(37, 9);
    numericUpDown.wValue := Driver.Value(2);
    groupBox.Date.wDate := Driver.Value(3);
    textBox := groupBox.Customer;
    textBox.Click(67, 17);
    textBox.wText := Driver.Value(0);
    textBox := groupBox.Street;
    textBox.Click(31, 5);
    textBox.wText := Driver.Value(4);
    textBox := groupBox.City;
    textBox.Click(47, 15);
    textBox.wText := Driver.Value(5);
    textBox := groupBox.State;
    textBox.Click(82, 4);
    textBox.wText := Driver.Value(6);
    textBox := groupBox.Zip;
    textBox.Click(10, 8);
    textBox.wText := Driver.Value(7);
    groupBox.WinFormsObject(Driver.Value(8)).ClickButton;
    textBox := groupBox.CardNo;
    textBox.Click(46, 9);
    textBox.wText := Driver.Value(9);
    groupBox.ExpDate.wDate := Driver.Value(10);
end;

```

```

procedure Checkpoint(Driver);
begin
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual,
Driver.Value(11), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,

```

```

cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), false);
end;

```

C++Script, C#Script

```

function PopulateForm(groupBox, Driver)
{
    var numericUpDown, textBox;

    groupBox["ProductNames"]["ClickItem"](Driver.Value(1));
    numericUpDown = groupBox["Quantity"];
    numericUpDown["Click"](37, 9);
    numericUpDown["wValue"] = Driver.Value(2);
    groupBox["Date"]["wDate"] = Driver.Value(3);
    textBox = groupBox["Customer"];
    textBox["Click"](43, 7);
    textBox["wText"] = Driver.Value(0);
    textBox = groupBox["Street"];
    textBox["Click"](31, 11);
    textBox["wText"] = Driver.Value(4);
    textBox = groupBox["City"];
    textBox["Click"](24, 6);
    textBox["wText"] = Driver.Value(5);
    textBox = groupBox["State"];
    textBox["Click"](31, 8);
    textBox["wText"] = Driver.Value(6);
    textBox = groupBox["Zip"];
    textBox["Click"](24, 17);
    textBox["wText"] = Driver.Value(7);
    groupBox["WinFormsObject"](Driver.Value(8))["ClickButton"]();
    textBox = groupBox["CardNo"];
    textBox["Click"](19, 7);
    textBox["wText"] = Driver.Value(9);
    groupBox["ExpDate"]["wDate"] = Driver.Value(10);
}

```

```

function Checkpoint(Driver)
{
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Price"]["wText"], cmpEqual, Driver.Value(11), false);
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Discount"]["wText"], cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["groupBox1"]

```

```
["Total"]["wText"], cmpEqual, Driver.Value(13), false);  
}
```

完整的代码如下：

VBScript

```
Sub OpenForm(mainForm)
```

```
    Call mainForm.MainMenu.Click("Orders|New order...")
```

```
End Sub
```

```
Sub PopulateForm(groupBox, Driver)
```

```
    Dim numericUpDown, textBox
```

```
    Call groupBox.ProductNames.ClickItem(Driver.Value(1))
```

```
    Set numericUpDown = groupBox.Quantity
```

```
    Call numericUpDown.Click(37, 9)
```

```
    numericUpDown.wValue = Driver.Value(2)
```

```
    groupBox.Date.wDate = Driver.Value(3)
```

```
    Set textBox = groupBox.Customer
```

```
    Call textBox.Click(39, 7)
```

```
    textBox.wText = Driver.Value(0)
```

```
    Set textBox = groupBox.Street
```

```
    Call textBox.Click(66, 7)
```

```
    textBox.wText = Driver.Value(4)
```

```
    Set textBox = groupBox.City
```

```
    Call textBox.Click(72, 9)
```

```
    textBox.wText = Driver.Value(5)
```

```
    Set textBox = groupBox.State
```

```
    Call textBox.Click(98, 9)
```

```
    textBox.wText = Driver.Value(6)
```

```
    Set textBox = groupBox.Zip
```

```
    Call textBox.Click(31, 12)
```

```
    textBox.wText = Driver.Value(7)
```

```
    groupBox.WinFormsObject(Driver.Value(8)).ClickButton
```

```
    Set textBox = groupBox.CardNo
```

```
    Call textBox.Click(53, 12)
```

```
    textBox.wText = Driver.Value(9)
```

```
    groupBox.ExpDate.wDate = Driver.Value(10)
```

```
End Sub
```

```
Sub Checkpoint(Driver)
```

```
    Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText,  
cmpEqual, Driver.Value(11), False)
```

```
    Call aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,
```

```

cmpEqual, Driver.Value(12) , False)
    Call
aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), False)
End Sub

```

```

Sub CloseForm(orderForm)
    orderForm.ButtonOK.ClickButton
End Sub

```

```

Sub CloseApplication(orders, mainForm)
    mainForm.Close
    orders.dlgConfirmation.btnNo.ClickButton
End Sub

```

```

Sub Main
    Dim orders, mainForm, orderForm, groupBox
    Set Driver = DDT.ExcelDriver("../TestBook.xlsx", "TestSheet", True)
    TestedApps.RunAll
    Set orders = Aliases.Orders
    Set mainForm = orders.MainForm
    While Not Driver.EOF
        Call OpenForm(mainForm)
        Set orderForm = orders.OrderForm
        Set groupBox = orderForm.Group
        Call PopulateForm(groupBox, Driver)
        Call Checkpoint(Driver)
        Call CloseForm(orderForm)
        Driver.Next
    Wend
    Call CloseApplication(orders, mainForm)
End Sub

```

JScript

```

function OpenForm(mainForm)
{
    mainForm.MainMenu.Click("Orders|New order...");
}

```

```

function PopulateForm(groupBox, Driver)
{
    var numericUpDown, textBox;

    groupBox.ProductNames.ClickItem(Driver.Value(1));
}

```

```

numericUpDown = groupBox.Quantity;
numericUpDown.Click(8, 7);
numericUpDown.wValue = Driver.Value(2);
groupBox.Date.wDate = Driver.Value(3);
textBox = groupBox.Customer;
textBox.Click(69, 12);
textBox.wText = Driver.Value(0);
textBox = groupBox.Street;
textBox.Click(4, 8);
textBox.wText = Driver.Value(4);
textBox = groupBox.City;
textBox.Click(12, 14);
textBox.wText = Driver.Value(5);
textBox = groupBox.State;
textBox.Click(21, 6);
textBox.wText = Driver.Value(6);
textBox = groupBox.Zip;
textBox.Click(14, 14);
textBox.wText = Driver.Value(7);
groupBox.WinFormsObject(Driver.Value(8)).ClickButton();
textBox = groupBox.CardNo;
textBox.Click(35, 15);
textBox.wText = Driver.Value(9);
groupBox.ExpDate.wDate = Driver.Value(10);
}

```

```

function Checkpoint(Driver)
{
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual,
Driver.Value(11), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,
cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), false);
}

```

```

function CloseForm(orderForm)
{
    orderForm.ButtonOK.ClickButton();
}

```

```

function CloseApplication(mainForm, orders)
{
    mainForm.Close();
}

```

```

    orders.dlgConfirmation.btnNo.ClickButton();
}

function Main()
{
    var orders, mainForm, orderForm, groupBox, Driver;

    Driver = DDT.ExcelDriver("../TestBook.xlsx", "TestSheet", true);
    TestedApps.RunAll();
    orders = Aliases.Orders;
    mainForm = orders.MainForm;
    while(!Driver.EOF())
    {
        OpenForm(mainForm);
        orderForm = orders.OrderForm;
        groupBox = orderForm.Group;
        PopulateForm(groupBox, Driver);
        Checkpoint(Driver);
        CloseForm(orderForm);
        Driver.Next();
    }
    CloseApplication(mainForm, orders);
}

```

DelphiScript

```

procedure OpenForm(mainForm);
begin
    mainForm.MainMenu.Click('Orders|New order...');
end;

procedure PopulateForm(groupBox, Driver);
var numericUpDown, textBox;
begin
    groupBox.ProductNames.ClickItem(Driver.Value(1));
    numericUpDown := groupBox.Quantity;
    numericUpDown.Click(37, 9);
    numericUpDown.wValue := Driver.Value(2);
    groupBox.Date.wDate := Driver.Value(3);
    textBox := groupBox.Customer;
    textBox.Click(67, 17);
    textBox.wText := Driver.Value(0);
    textBox := groupBox.Street;
    textBox.Click(31, 5);
    textBox.wText := Driver.Value(4);

```



```

    textBox := groupBox.City;
    textBox.Click(47, 15);
    textBox.wText := Driver.Value(5);
    textBox := groupBox.State;
    textBox.Click(82, 4);
    textBox.wText := Driver.Value(6);
    textBox := groupBox.Zip;
    textBox.Click(10, 8);
    textBox.wText := Driver.Value(7);
    groupBox.WinFormsObject(Driver.Value(8)).ClickButton;
    textBox := groupBox.CardNo;
    textBox.Click(46, 9);
    textBox.wText := Driver.Value(9);
    groupBox.ExpDate.wDate := Driver.Value(10);
end;

procedure Checkpoint(Driver);
begin
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Price.wText, cmpEqual,
Driver.Value(11), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.Discount.wText,
cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases.Orders.OrderForm.Group.groupBox1.Total.wText,
cmpEqual, Driver.Value(13), false);
end;

procedure CloseForm(orderForm);
begin
    orderForm.ButtonOK.ClickButton;
end;

procedure CloseApplication(orders, mainForm);
begin
    mainForm.Close;
    orders.dlgConfirmation.btnNo.ClickButton;
end;

procedure Main;
var orders, mainForm, orderForm, groupBox, Driver;
begin
    Driver:=DDT.ExcelDriver(' ../TestBook.xlsx', 'TestSheet', true);
    TestedApps.RunAll;
    orders := Aliases.Orders;
    mainForm := orders.MainForm;

```

```

while not Driver.EOF do
begin
    OpenForm(mainForm);
    orderForm := orders.OrderForm;
    groupBox := orderForm.Group;
    PopulateForm(groupBox, Driver);
    Checkpoint(Driver);
    CloseForm(orderForm);
    Driver.Next;
end;
    CloseApplication(orders, mainForm);
end;

```

C++Script, C#Script

```

function OpenForm(mainForm)
{
    mainForm["MainMenu"]["Click"]("Orders|New order...");
}

```

```

function PopulateForm(groupBox, Driver)
{
    var numericUpDown, textBox;

    groupBox["ProductNames"]["ClickItem"](Driver.Value(1));
    numericUpDown = groupBox["Quantity"];
    numericUpDown["Click"](37, 9);
    numericUpDown["wValue"] = Driver.Value(2);
    groupBox["Date"]["wDate"] = Driver.Value(3);
    textBox = groupBox["Customer"];
    textBox["Click"](43, 7);
    textBox["wText"] = Driver.Value(0);
    textBox = groupBox["Street"];
    textBox["Click"](31, 11);
    textBox["wText"] = Driver.Value(4);
    textBox = groupBox["City"];
    textBox["Click"](24, 6);
    textBox["wText"] = Driver.Value(5);
    textBox = groupBox["State"];
    textBox["Click"](31, 8);
    textBox["wText"] = Driver.Value(6);
    textBox = groupBox["Zip"];
    textBox["Click"](24, 17);
    textBox["wText"] = Driver.Value(7);
    groupBox["WinFormsObject"](Driver.Value(8))["ClickButton"]();
}

```

```

    textBox = groupBox["CardNo"];
    textBox["Click"](19, 7);
    textBox["wText"] = Driver.Value(9);
    groupBox["ExpDate"]["wDate"] = Driver.Value(10);
}

function Checkpoint(Driver)
{
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Price"]["wText"], cmpEqual, Driver.Value(11), false);
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["Discount"]["wText"], cmpEqual, Driver.Value(12), false);
    aqObject.CompareProperty(Aliases["Orders"]["OrderForm"]["Group"]["groupBox1"]["Total"]["wText"], cmpEqual, Driver.Value(13), false);
}

function CloseForm(orderForm)
{
    orderForm["ButtonOK"]["ClickButton"]();
}

function CloseApplication(mainForm, orders)
{
    mainForm["Close"]();
    orders["dlgConfirmation"]["btnNo"]["ClickButton"]();
}

function Main()
{
    var orders, mainForm, orderForm, groupBox, Driver;

    Driver = DDT["ExcelDriver"]("../TestBook.xlsx", "TestSheet", true);
    TestedApps["RunAll"]();
    orders = Aliases["Orders"];
    mainForm = orders["MainForm"];
    while(!Driver.EOF())
    {
        OpenForm(mainForm);
        orderForm = orders["OrderForm"];
        groupBox = orderForm["Group"];
        PopulateForm(groupBox, Driver);
        Checkpoint(Driver);
        CloseForm(orderForm);
        Driver["Next"]();
    }
}

```

```
}  
    CloseApplication(mainForm, orders);  
}
```

5. 运行测试脚本，查看测试结果（Executing Script and Checking Results）

右键点击项目浏览器的对应节点，选择菜单里面的 **Run | Run Main** 来运行脚本。

在测试结束之后，你可以分析测试日志（Test Log）。所有测试脚本执行时遇到的警告和错误都会记录到日志里。如果日志里面出现了“The property value does not equal the template value”，这就表明了程序的输出值与正确值有出入（被测程序有问题）。你可以把你想要的信息包含在测试日志里，比如，你在错误发生时，制定所有的输入参数。

完整的例子可以在

<TestComplete Samples>\Scripts\DDT 路径下找到。

TestComplete 测试.NET 应用程序

Testing .NET Applications – Overview（概况）

Testcomplete 提供对 Windows 应用程序的支持（包括 .Net 应用程序）。你可以通过模拟用户操作、录制和回放、或者其他特殊功能来执行黑盒测试，或者直接进行白盒测试。

如果安装了 [.NET Open Application Support](#) 插件，.Net 应用程序对 Testcomplete 就是开放的，这意味着你可以在测试中直接访问他们的内部属性和方法。默认设置下，.NET Open Application Support (tcClrOpenApp.pls) 插件安装在 <TestComplete>\Bin\Extensions 目录下。当插件安装了以后，你就可以访问 .NET 应用程序对象的内部方法和属性。你可以在 [Object Browser](#) 面板访问到它们。

Note: 这个插件仅对桌面的 .Net 应用程序有效（不包括 PDA 的 .Net 程序）。如果想在 PDA 上访问对象的内部方法和属性，必须使用 Testcomplete 专门为 WinCE 控件提供的特殊方法和属性（请参考 [Working With WinCE Controls](#)）。

为了使用对象的属性和方法，你必须识别出它们（通常是进程、窗体和控件）。Testcomplete 提供了特定的函数 [WinFormsObject](#) 和 [VCLNETObject](#) 来让你识别出 .Net 应用程序下的对象。访问 [Addressing Windows, Controls and Objects of .NET Applications](#) 来获取更多的信息。

当你控制了被测对象，你可以调用它们的方法和属性，例如 object_name.method_name。

VBScript

```

Dim p1, w1
' Obtain the list view control
Set p1 = Sys.Process("MyNETApp")
Set w1 = p1.WinFormsObject("MainWindow").WinFormsObject("UltraListView")
' Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3))

' Call control-specific method
Call w1.ClickItem 3

```

Jscript

```

var p1, w1;
// Obtain the list view control
p1 = Sys.Process("MyNETApp");
w1 = p1.WinFormsObject("MainWindow").WinFormsObject("UltraListView");
// Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3));

// Call control-specific method
w1.ClickItem(3);

```

DelphiScript

```

var p1, w1: OleVariant;
begin
// Obtain the list view control
p1 := Sys.Process('MyNETApp');
w1 := p1.WinFormsObject('MainWindow').WinFormsObject('UltraListView');
// Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3));

// Call control-specific method
w1.ClickItem(3);

```

C++Script, C#Script

```

var p1, w1;
// Obtain the list view control
p1 = Sys["Process"]("MyNETApp");
w1 = p1["WinFormsObject"]("MainWindow")["WinFormsObject"]("UltraListView");
// Obtain the value of control-specific property and post it to the test log
Log["Message"](w1["wSelected"](3));

```

```
// Call control-specific method  
w1["ClickItem"](3);
```

注意到被测对象可能包括重载 (overloaded) 的方法。Testcomplete为这些方法添加索引, 以便区分。浏览[Object Browser](#)可以面板找出对象重载方法的索引。请浏览[Retrieving Data From .NET Objects](#)这一节。

Testcomplete提供了一个特殊的dotNET对象, 允许你直接从脚本里调用.NET assemblies 里面的函数。为了使用这些函数, 你要先把要用到的assembly的名字加到项目的[CLR Bridge](#)选项里。当Assemblies加载到CLR Bridge列表后, 里面定义的函数就可以被Testcomplete的脚本调用了。它们显示在[Code Completion](#)窗口下 (作为dotNET对象的子节点)。请查看[Calling Functions From .NET Assemblies](#)获取更多的信息。

.NET Open Applications (白盒测试)

当[.NET Open Application Support](#)插件安装后, Testcomplete可以访问.Net对象的*public*, *protected*, *private*, *internal* 和 *protected internal*的属性和方法。默认下, *protected*和 *private*的属性在Object Browser下是隐藏的, 不能再一些项目项使用, 但可以通过脚本访问他们。请访问[Access to Properties](#)获取更多的信息。如果插件没有安装, 你同样可以通过录制回放等黑盒测试方法来测试.Net程序。

默认下, .NET Open Application Support plug-in (tcClrOpenApp.pls) 安装在 <TestComplete>\Bin\Extensions 目录下。如果想知道如何安装这个插件, 请看 [Installing Extensions](#).

插件所提供的功能不限于特定的编译器。插件同时支持微软和非微软的编译器。它需要 **Microsoft .NET Framework v. 1.0.3705** 或者更高的版本, 如果你使用的是 Microsoft Visual Studio 2005 (v. 8.0), 你需要安装 SP1 补丁。

能否访问到.Net 应用程序的对象, 取决于对象能否可见:

Visual Objects (可见对象)

可见对象的属性、方法、这些对象相关的事件都显示在[Object Browser](#)面板。

使用Testcomplete提供的[WinFormsObject](#) 和 [VCLNETObject](#)方法来访问.Net程序的对象。这些方法允许你使用对象名、对象的类名、标题或者索引来访问用Windows Forms 和 VCL .NET 类库创建的对象。请查看 [Addressing Windows, Controls and Objects of .NET Applications](#)获取更多的信息。

Non-Visual Objects (不可见的对象)

通常, 只要一直被引用, 对象就会一直存在。为了访问一个不可见的对象, 你需要找到一个属性 (property)、来自其他对象的字段 (field) 或者方法 (必须能够返回此对象的引用)。你可以通过[Object Browser](#)找到这些信息。你也可以通过AppDomain(...).dotNET语句, 使用静态字段、属性或者方法来获得某一对象的引用。

注: 如果安装了[TestComplete 3 Compatibility](#)插件, TestComplete会模拟一个TestComplete 3.x 的函数, .Net对象名的命名方式与产品的版本一致。

在获得指定对象后, 就可以调用它的方法和属性了。请查看[Retrieving Data From .NET Objects](#)获取更多的信息。注意到, 一些数据类型只能部分支持:

1. 小数按照浮点数来处理。
2. Tecomplete 为被测对象增加 Item 属性, 支持有索引的属性。但是, 在 Object Browser 里

面你不能指定索引属性的值（针对那些用类做参数的属性）。例如，某个索引属性使用了 `string` 作为索引参数，你可以在脚本上获取或者设置这些属性，但你不能在 `Object Browser` 上看到它们。那些采用数值类型作索引（比如 `int`、`long`、`bool` 等）的索引属性，`Object Browser` 和脚本都可以很好的支持。

3. 默认设置下，`protected` 和 `private` 类型的属性在 [Object Browser](#) 是隐藏的，在 [Name Mapping](#) 和 [Stores](#) 项目项里面也不可用。这样做是因为访问这些类型的属性可能会引发一些不安全的操作。尽管如此，你还是可以通过脚本来访问它们的。如果你想在 `Object Browser` 下面是用 `protected` 和 `private` 类型的属性，可以在 [Engines - General Options](#) 对话框里面把 `Show hidden properties` 设置成 `enable`。

`TestComplete` 提供了一系列关于 `.NET Open Applications` 的例子，请查阅 [Open Application Samples](#)。

Testing .NET Applications - Required Plug-Ins（所需插件）

若访问 `.Net` 程序的内部对象、方法和属性，下列的插件是必须的（它们放在 `<TestComplete>\Bin\Extensions` 目录下）。

.NET Open Application Support（文件名：`tcClrOpenApp.pls`）：在脚本里提供访问 `.Net` 程序内部字段、属性、方法的功能。

.NET Classes Support（文件名：`tcClrOpenApp.pls`）：增加 [dotNET](#) 对象来访问 `.NET assemblies`，和它们内部定义的数据类型和成员。

如何安装插件，请查看 [Installing Extensions](#)。

Addressing Windows, Controls and Objects of .NET Applications（窗体、控件和对象的寻址）

当 [.NET Open Application Support](#) 插件安装和配置完成之后，`.NET` 程序对 `TestComplete` 开放了，这意味着你可以访问 `.NET` 程序内部对象的方法、属性和字段。下面的内容会逐一解释如何寻址 `.NET` 程序的窗体、控件、对象。这种方法与 `.Net` 程序的编程语言和编译工具无关，无论是 `C#`、`Visual C++`、`.NET`、`Visual Basic`、`.NET`、`J#`、`C#Builder` 或 `Delphi`（`.Net` 版本）等。

下面的方法可用于关键字测试或脚本。增加 [Call Object Method](#) 命令到测试里，选择合适的方法、用 `Operation Parameters` 向导给参数赋值，来获取一个对象（一个进程、一个窗体、一个控件等）。[Call Object Method](#) 的使用方法请查阅 [Waiting for a Process Activation](#)。

Obtaining Processes

窗体、控件、对象的寻址是有层次的，这意味着，在持有任意 `.Net` 程序窗体的对象之前，你必须先持有对应的进程。可以使用 [Sys.Process](#) 或者 [Sys.WaitProcess](#) 方法来放回一个 [Process](#)（进程）对象。这两个方法的区别是，`Sys.Process` 立即返回一个对象；`Sys.WaitProcess` 会使脚本延时执行，直到指定对象出现或者达到预设的延时值。

如果指定的进程不存在，`Process` 方法会向测试日志发出了一个错误信息，然后返回一个空对象。如果你在调用方法或者属性时使用了这个对象，`Testcomplete` 会给测试日志发出一个错误信息或者直接在屏幕上显示一个错误信息。（取决于 [When referring to a non-existent](#)

[object](#) 选项)。使用[Exists](#)属性来检查指定的进程是否存在。

VBScript

```
If Not Sys.WaitProcess("MyNETApp").Exists Then
    Log.Warning "The specified process does not exist."
End If
```

JScript

```
if (! Sys.WaitProcess("MyNETApp").Exists )
    Log.Warning("The specified process does not exist.");
```

DelphiScript, Delphi

```
if not Sys.WaitProcess('MyNETApp').Exists then
    Log.Warning('The specified process does not exist.');
```

C++Script, C++, C#Script, C#

```
if (! Sys["WaitProcess"]("MyNETApp")["Exists"] )
    Log["Warning"]("The specified process does not exist.");
```

请浏览[Naming Processes](#)查看更多有关如何持有进程的信息。

Obtaining Windows, Controls and Objects

在你持有了进程之后，你就能获取整个程序的窗体或者程序的其他对象。寻址对象和寻址窗体非常相似：Testcomplete提供了[WinFormsObject](#)和[VCLNETObject](#)方法。WinFormsObject方法可以访问Microsoft Windows Forms Objects类库所创建的对象。VCLNETObject方法可以访问Borland VCL.NET类库创建的对象。

除了WinFormsObject 和 VCLNETObject方法，Testcomplete还提供了[WaitWinFormsObject](#) 和 [WaitVCLNETObject](#)方法。没有wait前缀方法和它们的区别是在于：没有wait前缀的直接返回指定的对象；有wait前缀的方法会延时脚本的执行直到对象可用，或者到达超时时间。它们的使用方法是类似的，后面的例子我们采用没有wait后缀的方法作示范，但你也可以把它们替换成相应的有Wait前缀的方法。

注：以上介绍的所有方法，只有.NET Open Application Support 插件安装后才可用。注意到你不能通过 Window 方法来持有相应的程序窗体。你只能通过 WinFormsObject 或者 VCLNETObject 才能寻址到它们。

相反的，如果插件没有安装，.Net程序会党组黑合程序来对待，WinFormsObject 或者 VCLNETObject方法都不能够使用。这种情况黑合程序一样，使用Windows方法，或者采用Process对象的[Child](#),[Find](#),[FindChild](#) 和[FindId](#)方法来搜索。

通过[aqEnvironment.IsPluginInstalled](#)函数来确认插件是否已经安装。

WinFormsObject 和 VCLNETObject方法（或者它们带Wait前缀的版本）所返回的对象，除了它们自己的方法和属性之外，还包含了TestComplete增加的方法、属性和动作。一些“应用程序”的方法和属性会被特殊对待。同样，TestComplete会对重载的方法加上索引。请浏览[.NET Open Applications](#)获取更多的信息。

WinFormsObject和VCLNETObject方法都有三种不同的输入参数：1.窗口名；2.类名和标题；

3.类名、标题和索引。展开[Object Browser](#)就可以查看当前TestComplete用的是哪一种参数。事实上，不同的实现方式体现了两种不同的模型：通过对象名寻址或者通过类名、标题和索引寻址。至于TestComplete用哪一个模型，取决于[Use native object names for TestComplete object names](#)这个项目设置。通过对象名寻址看起来更方便，但如果对象名没有在应用程序的代码里定义，或者存在两个或者多个相同的对象名，就用不了了。通过类型、标题和索引来寻址，可以唯一地表示一个对象，但缺点就是脚本代码更长了，降低了可读性。下面来详细研究一些这两种模型：

对象名寻址：

如果*Use native object names for TestComplete object names*项目设置打开了，你可以用对象名来寻址对象或者窗体（Name属性的值）。你把name作为参数传进WinFormsObject 或者VCLNETObject 方法里，之后它们会返回相应的对象。

下面是代码的例子，如果你的被测程序是用VCL.NET创建的，把WinFormsObject换成VCLNETObject就可以了。

VBScript, Visual Basic

```
Set p = Sys.Process("My_NET_App")
Set w = p.WinFormsObject("MainForm") ' Obtains MainForm
```

JScript

```
var p, w;
p = Sys.Process("My_NET_App");
w = p.WinFormsObject("MainForm"); // Obtains MainForm
```

DelphiScript, Delphi

```
var
  p, w : OleVariant;
begin
  p := Sys.Process('My_NET_App');
  w := p.WinFormsObject('MainForm'); // Obtains MainForm
end;
```

C++Script, C#Script, C++, C#

```
var p, w;
p = Sys["Process"]("My_NET_App");
w = p["WinFormsObject"]("MainForm"); // Obtains MainForm
```

你可以把通配符（*和?）传进WinFormsObject或者VCLNETObject方法里。

如果应用程序的代码里面没有NAME这个属性，又或者由于一些特殊的原因TestComplete没有办法把它识别出来，TestComplete会采用第二种寻址方式（类名、标题和索引）。你可以在Object Browser里面看到。

然而，如果你的.Net 应用程序包含了多个同名的窗体（例如，你同时打开了多个 EditOrder 窗体），Object Browser 会显示两个同名的节点。这是因为 TestComplete 不会自动检测同名的窗体，所以没能够自动地采取第二种寻址方式。

所以，如果你想要录制这种重名窗体的脚本，你要在后期修改这些脚本。或者，你可以取消 Use native object names for TestComplete object names 这个项目设置，来默认使用第二种寻址方式。

除了使用这种方法，你也可以在设置了 Use native object names for TestComplete object names 的前提下，可以手动去修改你的脚本。

在你的程序有多个重名窗体的情况下，你可以：1.使用 [Name Mapping](#) 功能；2.用 FindId, Find 或者 FindChild 方法来寻址对应的窗体。FindId 返回对应 id 的字对象，Find 和 FindChild 返回对应属性值的字对象。3.遍历整个进程窗体，检查每个窗体的属性，最终把想要的窗体找出来。可以使用 process 对象的 ChildCount 和 Child 属性来遍历。

类名、标题和索引寻址：

如果 Use native object names for TestComplete object names 设置为无效，Testcomplete 采用第二种方式寻址。把对应的参数传到 WinFormsObject 或 VCLNETObject 方法里即可返回相应的对象。

窗体的类名是你的应用程序里面定义的，与操作系统无关。你不可以用父类的命名空间作为类名。例如，应使用 TextBox 而不是 System.Windows.Forms.TextBox；

窗体标题是一个标识窗体给用户的文本。例如一个文本编辑器控件的标题，是一个描述性的文本说明。

索引是用来区分同一个父类下重名的子类。索引不一定从 1 或者 0 开始。TestComplete 有一套自己的算法来处理，索引可以从任意值开始。例如，第一个索引值是 3，那么，第二个索引值就是 4，第三个索引值是 7。查看 [Object Browser](#) 面板就能够清楚看到索引值是从哪里开始的。如果能够通过类名和标题来唯一标识一个对象，最好不要使用索引了。

你可以在类名和标题这两个参数里使用通配符（*或?）。

下面是脚本的例子：

VBScript, Visual Basic

```
Set p = Sys.Process("My_NET_App")
Set w = p.WinFormsObject("MainFormClass", "Form Caption", 1) ' Obtains MainForm
```

JScript

```
var p, w;
p = Sys.Process("My_NET_App");
w = p.WinFormsObject("MainFormClass", "Form Caption", 1); // Obtains MainForm
```

DelphiScript, Delph

```

var
  p, w : OleVariant;
begin
  p := Sys.Process('My_NET_App');
  w := p.WinFormsObject('MainFormClass', 'Form Caption', 1); // Obtains MainForm
end;

```

C++Script, C#Script, C++, C#

```

var p, w;
p = Sys["Process"]("My_NET_App");
w = p["WinFormsObject"]("MainFormClass", "Form Caption", 1); // Obtains MainForm

```

上面的例子展示了如何使用 WinFormsObject 方法。如果你的应用程序是用 VCL.NET 类库创建的，用 VCLNETObject 替换一下就 OK 了。

当你持有了窗体对象后，你就能获得这个窗体的控件。你可以采用上述例子的寻址模型，也可以采用下面例子的按名称寻址的模型：

VBScript, Visual Basic

```

Set p = Sys.Process("My_NET_App")
Set w1 = p.WinFormsObject("MainForm") ' Obtains MainForm
Set w2 = w1.TextBox1 ' Obtains the TextBox1 control

```

JScript

```

var p, w1, w2;
p = Sys.Process("My_NET_App");
w1 = p.WinFormsObject("MainForm"); // Obtains MainForm
w2 = w1.TextBox1; // Obtains the TextBox1 control

```

DelphiScript, Delphi

```

var
  p, w1, w2 : OleVariant;
begin
  p := Sys.Process('My_NET_App');
  w1 := p.WinFormsObject('MainForm'); // Obtains MainForm
  w2 := w1.TextBox1; // Obtains the TextBox1 control
end;

```

C++Script, C#Script, C++, C#

```

var p, w1, w2;
p = Sys["Process"]("My_NET_App");
w1 = p["WinFormsObject"]("MainForm"); // Obtains MainForm
w2 = w1["TextBox1"] // Obtains the TextBox1 control

```

上述脚本返回了一个经过封装的 TextBox1 对象。这个对象持有 TextBox1 的方法和属性，但不包含 Testcomplete 提供的属性和方法（Child, FindChild,等）。为了返回的对象同时拥有自

身的和 TestComplete 提供的属性和方法，最好使用 WinFormsObject (或 VCLNETObject)方法。

VBScript, Visual Basic

```
Set p = Sys.Process("My_NET_App")
Set w1 = p.WinFormsObject("MainForm") ' Obtains MainForm
Set w2 = w1.WinFormsObject("TextBox1") ' Obtains the TextBox1 control
```

JScript

```
var p, w1, w2;
p = Sys.Process("My_NET_App");
w1 = p.WinFormsObject("MainForm"); // Obtains MainForm
w2 = w1.WinFormsObject("TextBox1"); // Obtains the TextBox1 control
```

DelphiScript, Delphi

```
var
  p, w1, w2 : OleVariant;
begin
  p := Sys.Process('My_NET_App');
  w1 := p.WinFormsObject('MainForm'); // Obtains MainForm
  w2 := w1.WinFormsObject('TextBox1'); // Obtains the TextBox1 control
end;
```

C++Script, C#Script, C++, C#

```
var p, w1, w2;
p = Sys["Process"]("My_NET_App");
w1 = p["WinFormsObject"]("MainForm"); // Obtains MainForm
w2 = w1["WinFormsObject"]("TextBox1") // Obtains the TextBox1 control
```

如果对应的控件或者对象没有 NAME 属性（例如，NAME 属性为空），TestComplete 会采用 WinFormsObject 或 VCLNETObject 这种实现方式。

如果一个父级的控件拥有两个或者更多重名的子控件，你要采用一些特殊的方法：1.第二种寻址方式；2.命名映射；3.使用父窗体的 Find, FindChild 或 FindId 方法。4. 用 Child 和 ChildCount 属性遍历子窗体。

注意：为了给非窗体的 VCL.NET 控件寻址（例如 TLabel 控件），你也可以采用以下的实现方法：
`WindowObj.VCLNETObject(className, Index);`

className 控件的类名，Index 是它相对于父控件的索引（从 1 开始算起）。

如果 TestComplete 不能根据名称来寻址非窗体的空间，只能采用这种办法。

获取不可见的对象：

只要有引用，对象就会一直存活下去。为了访问不可见的对象，你需要找到一个别的对象，通过它的属性、字段或者方法来返回指定对象。你可以在 [Object Browser](#) 里找到这些对象。

同样的，你可以通过静态的字段、属性或者方法来返回特定对象的引用（通过 `AppDomain(...).dotNET` 语句）。`AppDomain` 是 `TestComplete` 初始化进程的时候加进去的。和 `Microsoft .NET Framework` 提供的 `AppDomain` 对象只有一个区别：它包含了额外的 `dotNET` 属性。这个属性允许你通过以下方法寻址：

```
Sys.Process("MyApp").AppDomain("MyApp.exe").dotNET.namespace_name.class_name.property_name;
```

注意到 `dotNET` 属性允许你访问任意在程序中定义好的类。例如，你可以写脚本来创建一个程序里已有的类的实例。这在单元测试里非常有用。

Retrieving Data From .NET Objects（获取.Net 对象的数据）

在测试 .Net 应用程序的时候，你可能需要用到被测控件和对象的数据。例如，你可能需要了解单选按钮当前的状态才能执行下一步的动作。`TestComplete` 能够访问大量的属性和方法。你可以从 [Object Browser](#) 或者脚本来访问它们。你也可以这样子去访问一个对象的属性或者方法：`object_name.method_name`。

VBScript

```
Dim p1, w1
' Obtain the list view control
Set p1 = Sys.Process("MyNETApp")
Set w1 = p1.WinFormsObject("MainWindow").WinFormsObject("UltraListView")
' Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3))

' Call control-specific method
Call w1.ClickItem 3
```

JScript

```
var p1, w1;
// Obtain the list view control
p1 = Sys.Process("MyNETApp");
w1 = p1.WinFormsObject("MainWindow").WinFormsObject("UltraListView");
// Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3));

// Call control-specific method
w1.ClickItem(3);
```

DelphiScript

```
var p1, w1: OleVariant;
begin
```

```
// Obtain the list view control
p1 := Sys.Process('MyNETApp');
w1 := p1.WinFormsObject('MainWindow').WinFormsObject('UltraListView');
// Obtain the value of control-specific property and post it to the test log
Log.Message(w1.wSelected(3));

// Call control-specific method
w1.ClickItem(3);
```

C++Script, C#Script

```
var p1, w1
// Obtain the list view control
p1 = Sys["Process"]("MyNETApp");
w1 = p1["WinFormsObject"]("MainWindow")["WinFormsObject"]("UltraListView");
// Obtain the value of control-specific property and post it to the test log
Log["Message"](w1["wSelected"](3));

// Call control-specific method
w1["ClickItem"](3);
```

TestComplete 可以不同程度地识别出 .NET 程序的属性和方法，这取决于你所安装的插件。

控件的公共属性和方法：

TestComplete提供了一系列的属性，允许你获得有关窗体的标题、类名、索引和句柄等信息，与此同时，你可以改变它们的大小和位置，给它们赋予焦点等。请查看Window和Process对象的描述。这些属性和方法对所有测试程序都适用。

内部.Net 属性：

如果[.NET Open Application Support](#)安装了，你可以通过脚本直接访问.Net对象的内部属性和方法。注意到，有时候这些内部属性名会跟TestComplete提供的属性相冲突。与TestComplete相一致的属性和方法，会放在NativeClrObject命名空间下（在Object Browser可以看到）。你可以通过命名空间来寻址这些属性。请查看[Using Namespaces](#)获取更多的信息。

TestComplete 为.Net 对象增加的方法和属性：

TestComplete 提供了两个从脚本获得.Net 对象名的方法。

为常用的.Net 控件提供的属性和方法：

TestComplete 为常用的第三方控件提供了支持。前提是，你安装了相应的插件。在录制脚本的过程中，TestComplete分析空间的类名，找出匹配的程序对象。如果你的被测程序采用了第三方控件，TestComplete可能识别不出来，所以不能够找到合适的程序对象来匹配它们。如果第三方控件的类名是继承与已知的对象，你也可以手工从Object Mapping中关联它们。**Windows Class Names**持有了控件的类列表，与**Objects List**列表中支持的对象是相关联的。增加关联时，我们从**Objects List**选择对象的类型，接下来通过键盘输入或者从屏幕映射来把对应的类名添加进去。如果你的控件用的是ClrClassName属性，你可以定制类名。请查阅Object Mapping。

一旦建立了类的映射，就可以取到所有由TestComplete提供的方法和属性了。注意到，如果

控件关联了不正确的对象类型，录制和回放脚本都会引发错误。你可以按[Project Properties - Object Mapping Options](#)所说的方法来解除不正确的关联。