TIB 自动化测试工作室
http://www.cnblogs.com/testware/

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合测试方法来测试.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，取决与对象能否可见：

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 and 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。