FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 12 月刊

使用集合(第二部分) — Lauren Clarke and Randy Pearson 著 Fbilo 译

Page.21

这是由三部分组成的用集合设计系列文章的第二部分。在第一部分中，Randy Pearson 和 Lauren Clarke 检讨了一些他们在使用了两年的集合以后所学到的经验。这个月，他们将通过演示集合可以怎样被用来建立接口灵活的 Visual FoxPro 对象模型，来将这些经验应用到工作中去。

表格列交换器 —Dragan Nedeljkovich 著 CY 译

Page.28

在他先前的文章里，Dragan Nedeljkovich 已经介绍了“半手工生成器”的概念，那些是简单的小小实用工具，可以帮助你变成为一个多产的 VFP 开发者。最后，他为你演示了如何创建最简单的生成器。现在来做更严肃些的，创建表格列交换器。

查看，捕捉，建档 — Richard D. Hodder 著 CY 译

Page.31

捕捉 GUI 屏幕图像来做文档，是件费力和乏味的事。在这篇文章里 Rick Hodder 讲解了一个通用屏幕捕捉程序的自动接口，Snagit，并演示如何遍历项目里的表单，生成一个屏幕快照手册。

一些旧的，一些新的

作者：Andy Kramek & Marcia Akins

译者：Fbilo

当 Visual FoxPro 9.0 的发布临近的时候，它的新功能和增强的名单已经最终被确定了，所以，现在去看看新的版本更有实际的意义。这个月，Andy Kramek 和 Marica Akins 会去探索 9.0 版将给我们什么样的语言增强和新能力。出发点是：他们突然认识到长久一来一直存在于 VFP 中的一个问题已经永远消失了。

玛西亚：嗨，安迪！我知道这听起来很蠢，可我确实忘了怎么在一个 SQL 查询的结果集中建立一个新的 Integer 数据类型的列了。

安迪：嗯，有多种办法。最简单的就是指定一个由多个 0 组成的字符串，就像这样：

```
SELECT maintable.*, 00000 AS newint;
FROM maintable ;
INTO CURSOR cur_results
```

不过，如果你这么做了，那么你得到的其实并不是一个真正的 Integer 数据类型，而是一个按照你指定“0”的格式所组成的一个数值型列。在这里，它就是 N(5,0)。

玛西亚：哦，这可不太好。因为我需要一个真正的 Integer 列，而且我也不知道我将会在其中储存多大的数字——尽管我能非常确信它不会超过一个 Integer 数据的最大值（正的最大值是 2,147,483,647）。

安迪：这种情况下，你可以用指定一个宽度为 10 的数值型字段的办法来绕过它。不过，如果你真的需要一个 Integer 字段，在 VFP 9.0 出现之前的唯一办法只有建立一个只带一条记录的假游标，并在这条唯一的记录里面定义一个 Integer 字段。然后你把这个表不带条件的与你的查询连接起来，这样，结果集中的每一条记录就都带有一个 Integer 列了，象这样：

```
CREATE CURSOR dummy (intfield I)
INSERT INTO dummy (intfield) VALUES (0)
** 在这里运行查询
SELECT maintable.* dummy.intfield AS newint ;
FROM maintable, dummy ;
INTO CURSOR cur_results
```

注意：为了实现这个目的，你必须使用上面这样的查询方式。使用不带一个 ON 子句的 “FULL JOIN”语法会产生一个语法错误，并导致代码无法编译。不过，你可以将“规范的语法和不规范的连接”结合起来使用，就象这样：

```
CREATE CURSOR dummy ( intfield I )
INSERT INTO dummy VALUES (0)
SELECT AC.cacref, AL.clocperson, intfield ;
FROM dummy, account AC ;
LEFT OUTER JOIN acctloc AL ;
ON AL.iacfk = AC.iacpk
```

玛西亚：哦，太好了。这还让我想到了这就是在我需要某个结果集中有一个备注字段时所必须要做的事情。我想我可以被我的 DBC 添加一个名为 dummy 的表，并给它一个 Integer 字段和一个 Memo 字段。这样一来，无论何时我需要哪种类型，我都能方便的得到它了。

安迪：是的。不过在 VFP 9.0 中就根本不需要这么做了。新的 CAST() 函数除了能够转换数据类型之外，还可以被用来指定 VFP 应该怎样在一个结果集中建立增加的列。那么现在，为了建立 Integer、Memo、或者其它任何指定的数据类型，你只需要用象这样的代码就行了：

```
SELECT maintable.*, CAST( 0 AS INTEGER ) AS newint ;
CAST( "" AS MEMO ) AS newmemo ;
FROM maintable ;
INTO CURSOR cur_results
```

玛西亚：这太酷了。我还注意到你在为 CAST() 函数使用 SQL Server 的语法，把数据类型指定为“Integer”而不是“I”。这是否意味着现在我们可以用 SQL Server 那样的方式指定我们的数据类型？如果那样的话，我们是否还能用老的办法来做呢？

安迪：是的，是的，就是这样。事实上，VFP 9.0 不仅允许我们使用“适当的”语法来定义列（这使得我们的表定义可以直接移植到 SQL Server 中去了），而且它还给了我们一些新的非常有用的数据类型。最有用的就是 VARCHAR（VFP 的缩写是 “V”）了。

玛西亚：你的意思是我们不再需要去 ALLTRIM()一个字符型字段以去掉后面的空格了吗？这是一个很大的进步，尤其是在使用象 SQL Server 这样的远程数据存储的时候。

安迪：是的，如果列被定义为 VARCHAR，这就基本上是正确的。

玛西亚：但是我想 VFP 给它的表使用的是一个固定的字段格式。事实上，据我所知，它的确是这么做的。那么 VARCHAR 怎么会有效呢？

安迪：这的确有一些诡异，你说的关于 VFP 使用固定长度字段的事情是正确的。在建立一个 VARCHAR 列的时候，你还是需要给它定义一个长度。但是当你访问一个 VARCHAR 字段的时候，VFP 会自动删除

在常见的字符型字段中自动填充的那些空格，这样就只返回了有效的数据。很明显，如果你在一个 **VARCHAR** 列中数据的末尾输入了空格，那么你输入的这些空格就是“真正的”空格，所以在你获取数据的时候，这些空格不会被删掉。

玛西亚：不管诡异不诡异，我自己是肯定会把我已有的数据库中许多字段改成 **VARCHAR** 了。

安迪：啊哈，现在还有个问题。如果你这么做的话，**VFP** 会假定对旧的字符型字段数据进行填充的空格是“重要的”，所以，即使你把它们定义成 **VARCHAR**，这些空格还是会保留在那里。

玛西亚：那不是什么大问题。我会在改动了数据类型以后执行下面的命令：

```
REPLACE ALL varcharfield with ALLTRIM(varcharfield) ;
IN mytable
```

然后我就永远不再需要在我的代码中执行另一个 **ALLTRIM()**了。

安迪：你也许会认为这很有趣。但是 **ALLTRIM()**函数自己也发生了一些改动，也许你就不会那么快的永远抛弃它了。现在，你可以指定从一个字符串的开头和末尾删除 **23** 字符或者其它字符串了。

玛西亚：啊哈！我可以马上就想到它的用途——删除前导或者后续的 **Tab** 字符、回车、和行齐满等等。

安迪：难道你不能用 **STRTRAN()** 通过指定开始的发生次数和要替换的次数来做到同样的事情吗？

玛西亚：不完全是。假定你有一个象这样的字符串：

```
<TAB><TAB>This is some text<TAB>And more text<TAB><TAB>
```

用 **STRTRAN()** 你怎么能够删除了前面和后面的 **Tab**、却把中间的 **Tab** 保留下来？

安迪：嗯，我理解你的意思了。你更需要去算出临近的 **Tab** 字符的数量、**Tab** 字符的总数、然后搞清楚要删除多少个以及从哪里开始删除。

玛西亚：你想过头了。我只是想要用增强的语法去一次就去掉全部前导和后续的东西，就像这样：

```
lcStr = ALLTRIM( lcStr, 1, ;
CHR(32), CHR(9), CHR(13),CHR(10),CHR(0))
```

安迪：**Okay**，这一点上是你赢了。但是 **VFP 9.0** 中还有一个很酷的新函数，我认为在 **SQL** 查询中真的是非常有用的。**ICASE()** 让你可以定义一系列的条件，并根据匹配的条件返回一个指定的值。它就像是一个嵌套的 **IIF()**，只是你就不需要去跟踪所有的括号位置了。

玛西亚：给我个例子。

安迪：当你需要处理一系列的可能值时——比如从一个选项按钮组中返回的值——该怎么办？“老的”办法时使用对 **IIF()**函数的嵌套调用：

```
lcRetVal = IIF( This.Value = 1, 'S', ;
```

```
IIF( This.Value = 2, 'P', 'X' ))
```

而有了 **ICASE()**以后，它就只要调用一个函数了：

```
lcRetVal = ICASE(This.value=1,"S", ;
This.Value = 2, "P", "X")
```

你可以看到我们不再需要中间的函数了。相反，我们只要把成对的用逗号分隔的条件和与条件相关的值传递给函数就行了。当没有一个条件匹配的时候，最后的（单独的）值就被当作是“**otherwise**”条件部分返回。

玛西亚：我觉得 **ICASE()**是更具可读性一些，但对于只有三个选项的情况来说优势并不明显。不过，如果你有大量的选项要处理的话，用它来处理是方便多了。它的另一个好处是：当你被限制在单行代码中的时候它仍然有效。

安迪：你什么时候会被限制在单行代码里的？

玛西亚：比如你要为一个必须被运算过的自定义属性指定一个值、或者在某个报表的输出字段或一个“**print when**”子句中。

安迪：**Okay**，理由已经足够了——你已经证明了你的观点。

玛西亚：你知道，正当我想到 **VFP** 是一个功能如此丰富、完整的语言的时候，**Fox** 开发组又令我惊讶的使它更完美了。关于自动增长字段的实现问题，我们已经提出的一个意见是：当时还没有办法找出一个指定的表最后生成的值。这就对使用可更新视图造成了很大的困难。

安迪：是的，我记得当时我们很遗憾的发现在 **VFP** 里没有 **SQL Server** 的 **@@IDENTITY** 函数的替代品。

玛西亚：是的，你肯定会高兴的知道现在它已经有了！新的 **GetAutoIncValue()** 函数会返回在一个指定数据工作期中最近生成的值。

安迪：这很整洁。就像 **SQL Server** 使用连接一样，**VFP** 是使用数据工作期来确保为一个指定用户返回的是正确的“最新的 **id**”。

玛西亚：是的，还有，如果你传递一个 **0** 作为 **nDataSessionNumber** 参数，你获得的就是在当前运行中的方法或过程中有效的值——非常类似于 **SQL Server** 的 **SCOPE_IDENTITY()** 函数。

安迪：既然我们在谈新函数的题目，那么还有一个函数是必须要提一下的：**MakeTransactable()**。

玛西亚：它是干嘛的？

安迪：就像它的名字所说的那样，它可以被用来为在一个指定工作区中的一个指定的自由表或者游标启用事务支持。

玛西亚：喔！那是另一个很好的巨变了。我明白我们应该总是使用一个数据库，但还有大量的系统（由于某些原因，大部分都是会计系统）仍然完全运行在自由表上。把它们包括在事务中的能力会让大多数人的生活变得更简单了。

安迪：是的，不过它还有一些限制。你必须要记住，如果你想让一个自由表或者游标支持事务，它就不能已经被打开、并且不能在别的数据工作期中“不支持事务”。总之，一旦自由表或者游标被设为支持事务了，你就可以在多个数据工作期中有多个该表的复件，并且它们默认都是支持事务的。

玛西亚：抱歉！请你解释的清楚一些好吗？

安迪：基本上，一旦一个自由表或者游标被打开在超过一个数据工作期中了，它的状态就不能被带动了。所以，如果你想要为自由表或者游标启用事务支持，你应该：

- 每次表被打开的时候就调用 **MakeTransactable()** 来让它启用事务支持。如果它已经是支持事务的了，这个函数就只是返回**.T.**而什么都不干。
- 使用一个例外处理器或者跟踪 **2191** 号错误——后者是当目标表已经在另一个数据工作期中被打开、并且不支持事务的时候触发的。

玛西亚：**Okay**，现在说的够清楚的了。还有别的什么数据增强能引起你的注意的吗？

安迪：我不能确定，但是我认为有一个东西潜在着极大的危险性——在一个 **SQL** 查询中对一个来源表增加的 **“WITH (BUFFERING=.T.)”**子句（是的，你必须每次都写上括号）。这会允许一个 **SQL** 查询从本地的缓冲而不总是从磁盘中去读取数据。

玛西亚：对我来说，这是一个已经被要求了多次的功能（事实上，请看本月的**“VFP** 开发组提供的技巧”专栏，里面有它的工作方式的一个演示）了。我已经在一些在线论坛上为不少人回答过为什么他们的 **SQL** 查询返回“错误的”结果的问题。对我来说，这听起来是一个不错的增强。

安迪：危险在于：它打破了一个“可重复读取”的原则。换句话来说，如果两个用户同时运行了同一个查询，他们应该得到同样的结果。使用缓冲了的（就是说，未提交的）数据的问题是双倍的。首先，它是本地的，所以它只对当前用户可见。

玛西亚：为什么这会成为问题？

安迪：想象一下你正在更新某些表的过程中间，我们老板要求我们两个去做一个显示这些数据状态的报表。除非你提交或者放弃了你的未决的、又没有完成的改动，否则我们将从同样的数据上执行同样的查询却获得了不同的结果。

玛西亚：我同意你的观点，这会极大的降低某人对系统的信心。但是另一方面，这是一个很人为的案例。你的另一个反对意见呢？

安迪：这个更加严重的多。它潜在的允许某些人生成没有被校验、检查、或者保存过（对这种情况来说，

数据甚至还没有被建立）的信息。那么，如果我脑子有毛病，我就能给一个客户的帐目结算添加一些东西，却又不保存改动。然后我可以运行查询来生成他们的帐单（显示充了水的数据）、并在打印后取消改动。即使别人对同样的数据进行查询，他们也永远看不到我的“充了水的数据，因为它从来没有出现在我的本地缓冲之外过。

玛西亚：但是你可以通过添加、然后立即删除记录来做到同样的事情。

安迪：啊哈，但那就应该有一个可以检测到的跟踪记录了（即使假定我有对那个表的删除权限）。一条记录应该已经被建立，并且别人运行同样的查询应该能看到“充了水”的数据。这就是可重复读的意思。不考虑未决改动，任何人，在任何时间运行查询都应该得到同样的结果。

玛西亚：这会让你感觉好过点：为了实现这个功能，你必须显式的为查询中的每个表使用“**WITH (BUFFERING=.T.)**”子句，默认的行为仍然是数据应该是直接从永久存储而不是本例缓冲中读取的。

安迪：而那就是另一个问题。如你所说，你必须在查询中为每个表单独的启用这个子句。如果你只给一个表启用的话，你会得到什么样的结果呢？

玛西亚：好吧，当语言变得越来越好的同时，硬币的另一面就是：现在有更多的办法打倒你自己了。但是不管怎么说，我还是对这些东西非常激动——在 **VFP 9.0** 中有一些真的很酷的语言增强，并且我确信还能发现更多。

从 XSouce 中淘金，第三部分

作者：Doug Hennig

译者：Fbilo

在这个由三部分组成的系列前两部分中，**Doug Hennig** 研究了在 **XSource--VFP** 自带的大部分 **“XBase”**工具的源代码**--**中的一些有趣的部件。这个系列的最后一篇文章讲的是两个部件：一个滚动的区域类和一个建立你自己的生成器的框架。

如我前面两篇文章讲述的那样，几乎所有 **VFP** 自带的 **Xbase** 工具的源代码是一个 **ZIP** 文件：在 **VFP** 主目录下 **Tools\XSource** 子目录下的 **XSource.ZIP**。解压这个文件会导致 **Tools\XSource** 目录下面新增加了一个名为 **VFPSource** 的目录，其中包含着 **XBase** 工具的源代码。

在第一部分中，我讨论了包含在 **XSource** 中的一些图片、你可以用来将设置保存在 **VFP** 资源文件中的部件、还有可以建立面对对象的快捷菜单的部件。第二部分向你展示了怎样在 **VFP** 表单中显示 **Video** 文件、建立进程对话框、以及使用一个模仿常见的 **Outlook bar** 控件的控件。

在这个系列的最后一篇文章中，我将向你演示怎样在一个 **VFP** 表单上建立一个滚动的区域，并检验一个用于建立你自己的生成器的 **framework(**框架**)**。

建立一个滚动区域

你也许遇到过需要在一个滚动的区域中显示一些控件的情况。例如，假定你允许你的用户能够向你的应用程序增加他们自定义的字段。这是一个强大的功能，因为它允许你的用户定义应用程序以适应他们的需要。然后问题就是怎样显示这些自定义字段。字段很少的话是不成问题的，表单上放得下。如果字段很多，你就需要一个滚动表单了。但是如果你要在表单上的别的不可滚动控件上显示字段的话怎么办？在这种情况下，你需要一个能放入字段的可滚动区域。

任务列表（**Task List**）是一个从 **VFP 7** 开始 **VFP** 自带的相对来说很少用到的应用程序。它让你可以在任务属性对话框的字段页上一个可滚动区域中添加自定义字段并显示它们。见图 **1**：

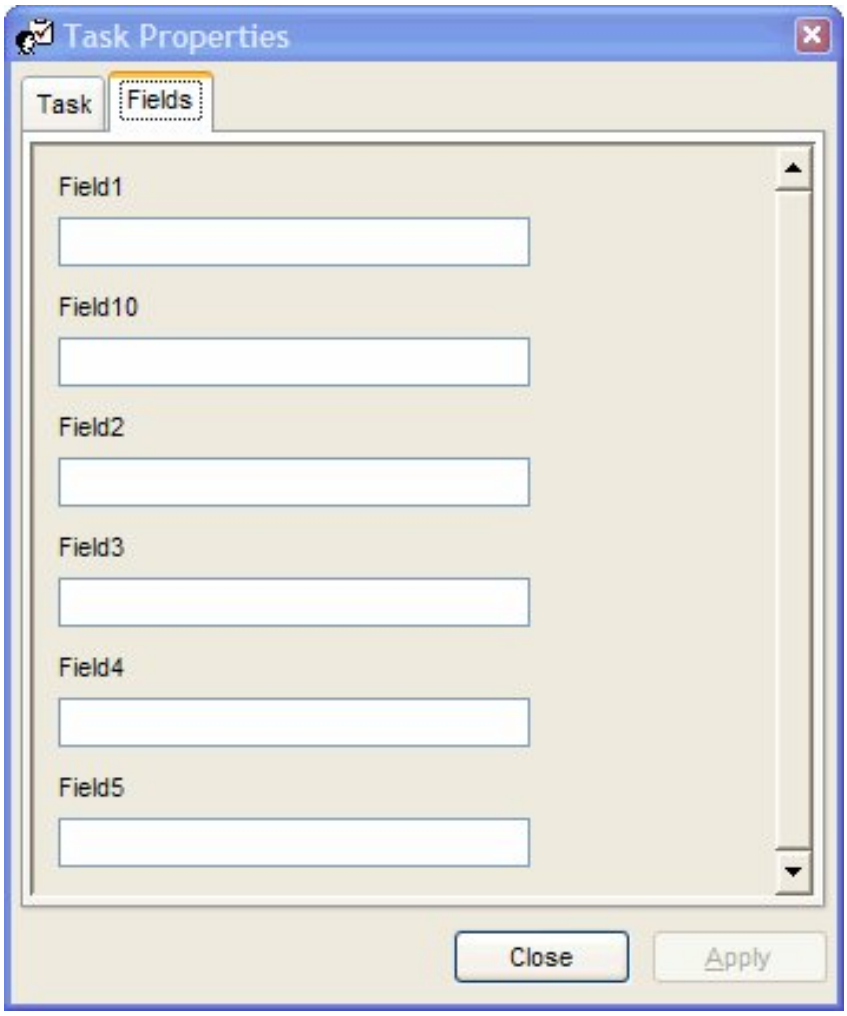

图 1

这个滚动区域是一个 **XSource** 路径 **TaskList** 目录下 **TaskListUI.VCX** 类库中的 **cntScrollRegion** 类的实例。**cntScrollRegion** 包含一个容器（控件将被放入的地方）和一个 **Microsoft Flat ScrollBar** 控件（发布应用程序的时候，你需要带上提供这个 **ActiveX** 控件的 **MSCOMCT2.OCX** 文件）。每个被添加入该容器的控件（例如图 **1** 中的 **Label** 和 **Textbox**）都也是来自 **TaskListUI.VCX** 类库中某个 ***Scrolling** 类的实例。放入容器的需要是专门的类而不是基类的原因是：当用户从一个控件移动到另一个的时候，需要根据情况自动的滚动可滚动区域。

让我们来看看怎么在你自己的应用程序中使用这些类。我决定为示例的 **NorthWind** 数据库的 **Customers** 表使用自定义字段，而不是直接给 **Customers** 表添加字段，这样会在我安装我的应用程序的一个升级版本时在更新表结构的时候造成问题，所以我建立了一个分离的表 **CustomFields**，它有一个 **CustomerID** 字段，该字段与 **Customers** 表中记录的 **CustomerID** 的值是相匹配的，此外，该表里还有我的用户增加的不管什么字段。另一个表 **CustFldDef** 是用户已经添加的自定义字段的定义（我会提供一个对话框，在该对话框里用户可以定义新的字段，根据需要，对话框会更新 **CustFldDef** 表的内容）。这个表里有着这样一些字段：**FIELDNAME**、**FIELDTYPE**、**FIELDLEN**、**CAPTION**、**ORDER**、还有 **PICTURE**，它们定义的是每个自定

义字段的信息。

在 **SFXSource.VCX** 中的 **SFCustomFieldsForm** 是一个基类，它上面有一个 **cntScrollRegion** 以及两个按钮 **OK** 和 **Cancel**（由于在 **Task List** 部分类中某些方法的需要，**OK** 按钮的 **Name** 是 **cmdApply**）。**Init** 方法打开自定义字段表（为了让这个表单通用，表的名称、使用的索引、主键字段的名称，包含在 **cCustomFieldsTable**、**cCustomFieldsOrder**、和 **cCustomFieldsKey** 属性中而不是写死在代码里的），并调用 **SetupCustomFields** 方法来设置滚动区域。

这里是 **SetupCustomFields** 方法的代码：

```
local lnSelect, ;
  lcAlias, ;
  lnTop, ;
  lcField, ;
  lcType, ;
  lnLen, ;
  lcCaption, ;
  lcPicture, ;
  lcClass, ;
  lcTemplate, ;
  lcLabel, ;
  loLabel, ;
  loField
with This.cntScrollRegion
* TaskList 滚动控件要求表单上要有一个 Task 对象，
* 而且该对象的属性应与字段的名称匹配。
* 所以，我们在这里建立它
  This.AddProperty('Task', createobject('Empty'))
* 打开包含着自定义字段信息的表，
* 并遍历每条记录
  lnSelect = select()
  select 0
  use (This.cMetaDataTable) again shared order ORDER
  lcAlias = alias()
  lnTop    = 10
  scan
* 获取字段的信息
```

```
        lcField     = trim(FIELDNAME)
        lcType      = FIELDTYPE
        lnLen       = FIELDLEN
        lcCaption   = trim(CAPTION)
        lcPicture   = trim(PICTURE)
        lcTemplate = ''
    * 根据数据类型，弄清楚为数据输入使用哪个类
    * 对于日期和日期时间型字段，
    * 你可以使用 oleDateScrolling 和 oleTimeScrolling 类，
    * 只是它们不能处理空的日期。
        do case
          case lcType = 'L'
            lcClass = 'chkScrolling'
          case lcType = 'M'
            lcClass = 'edtScrolling'
          case lcType $ 'NFIBY'
            lcClass     = 'txtScrolling'
            lcTemplate = lcPicture
          otherwise
            lcClass     = 'txtScrolling'
            lcTemplate = replicate('N', lnLen)
        endcase
    * 如果我们不是在使用一个 checkbox, 那么建立一个 label.
        if lcClass <> 'chkScrolling'
          lcLabel = 'lbl' + lcField
          .cntPane.NewObject(lcLabel, 'labScrolling', ;
            home() + 'Tools\XSource\VFPSource\TaskList\' +;
            'TaskListUI.vcx')
          loLabel = evaluate('.cntPane.' + lcLabel)
          with loLabel
            .Top        = lnTop
            .Left       = 10
            .Caption    = lcCaption
            .FontName = This.FontName
            .FontSize = This.FontSize
            .Visible    = .T.
```

```
            lnTop       = lnTop + .Height + 4
          endwith
        endif lcClass <> 'chkScrolling'
    * 建立用于该字段数据输入的一个控件
        .cntPane.NewObject(lcField, lcClass, ;
          home() + 'Tools\XSource\VFPSource\TaskList\' + ;
          'TaskListUI.vcx')
        loField = evaluate('.cntPane.' + lcField)
        with loField
          if lcClass = 'chkScrolling'
            .Caption = lcCaption
          endif lcClass = 'chkScrolling'
          .Top              = lnTop
          .Left             = 10
          .Visible          = .T.
          lnTop             = lnTop + .Height + 10
          .ControlSource = This.cCustomFieldsAlias + '.' +;
            lcField
          if inlist(lcClass, 'oleDateScrolling', ;
            'oleTimeScrolling')
            .Font.Name = This.FontName
            .Font.Size = This.FontSize
          else
            .FontName = This.FontName
            .FontSize = This.FontSize
          endif inlist(lcClass ...
          if not empty(lcTemplate)
            .Width = txtwidth(lcTemplate, This.FontName, ;
              This.FontSize) * ;
              fontmetric(6, This.FontName, This.FontSize) ;
              + 12
          endif not empty(lcTemplate)
          if lcClass = 'txtScrolling'
            .InputMask = lcPicture
          endif lcClass = 'txtScrolling'
        endwith
```

```
    * 给 Task 成员添加一个匹配字段名称的属性
        addproperty(This.Task, lcField)
      endscan
    * 刷新滚动区域
      .cntPane.Height = lnTop
      .SetViewPort()
    * 进行清理
      use
      select (lnSelect)
    endwith
```

这段代码由给表单添加一个名为 **Task** 的新属性开始，并在这个属性中放入一个 **Empty** 对象。**Task List** 中的类们要求表单的 **Task** 属性应该包含一个对象，而且该对象的属性应该与自定义字段名称相匹配（它使用这些属性来进行数据绑定，而不是绑定到表中的字段，但实际上我们还是将直接绑定到这些字段），所以后面我们将给这个 **Empty** 对象增加一些属性以匹配那些字段名。

代码打开包含着自定义字段元数据的表，表的名称被存储在 **cMetaDataTable** 属性中而不是在代码中写死。**SetupCustomFields** 根据每个字段的数据类型决定为每个字段使用哪个 ***Scrolling** 类作为显示数据的控件，如果控件不是一个 **checkbox**，那么它会给滚动区域添加一个 **lable**（使用 **lblScrolling** 类）。然后它为字段添加相应的控件并设置各种属性，包括 **ControlSource**，这个属性将控件绑定到自定义字段表中的字段。

这时，象我们前面讲到的那样，一个与字段有着同样名称的属性被添加给包含在 **Task** 属性中的 **Empty** 对象。最后，当所有的控件都已经被添加好以后，代码会根据情况设置滚动区域的 **Height**（如果有大量的控件的话，滚动区域也许会比表单更高），并调用 **cntScrollRegion** 对象的 **SetViewPort** 方法，该方法会相应的调整滚动条的大小、位置、以及其它属性。

cntScrollRegion 的 **SetViewPort** 方法里有一个 **Bug** 会导致滚动条不正确的显示。把下面这行注释掉：

```
This.oleScrollBar.LargeChange = This.Height - 20
```

TestScrolling.SCX 是一个基于 **SFCustomFieldsForm** 类的表单。它的 **cCustomFieldsTable**、**cCustomFieldOrder**、**cCustomFieldsKey**、以及 **cMetaDataTable** 属性被设置为相应的值以使用 **CustomFields** 和 **CustFldDef** 表。你可以运行这个表单并给它传递一个来自 **Customers** 表的一个 **CustomerID** 值（比如**“ALFKI”**），或者运行在这个系列文章第二部分中讨论过的 **TestToolbox** 表单、选择 **Customers** 模块、在列表中选择一个客户、然后单击 **Edit Custom Fields** 按钮。图 **2** 显示的是 **TestScrolling** 表单在滚动了一段以后的情况。

图 2

建立你自己的生成器

从 VFP 诞生起就有了生成器功能。不过，生成器还远未被充分利用过，部分原因是许多人认为它们非常难以建立。虽然事实并非如此，在建立一个生成器的时候有大量需要管理的事情，例如获得对被生成器维护的对象的一个引用、处理模式问题（如果生成器是无模式的，那么当表单或者类被关闭的时候，生成器也应该被关闭），等等。幸运的是，XSource 同样能为我们提供帮助。

XSource 带有一个建立生成器的完整框架。它不在它自己的目录里，而是作为 CursorAdapter 和 DataEnvironment 生成器的一部分（在 XSource 下 Wizards 目录的 DEBuilder 子目录中）。这个框架使得建立你自己的生成器的工作变得简单了，因为它会处理下列所有的问题：

- 它可以被从 Builder.App 中、或者作为一个独立的表单被调用；
- 它同时支持模式和无模式对话框；
- 它会处理所有的生成器对被维护对象的了解问题，比如维护对被维护对象的一个引用；
- 它在激活（activate）的时候自动刷新，所以切换到属性窗口、做某些改动、然后重新激活生成器时，会自动对生成器进行刷新；
- 它会在表单或者类被关闭的时候自动关闭；
- 当单击它的 Cancel 按钮的时候，它会取消所做的改动；

我不会在这里讨论生成器的结构或者它是怎么工作的，但关于生成器框架的优点是：建立你自己的生成器并不要求你理解这些事情。你需要知道的信息如下：

- 要建立一个生成器，应该做一个 BuilderControls.VCX 类库中 BuilderForm 类的子类。你在这个表单中使用的控件可以是在 BuilderControls.VCX 类库中其它类的实例（比如 BuilderLabel 或者 BuilderTextbox），但这样做

并非是必要的，你可以使用自己喜欢的任何控件。

- 指定某个特定的类应该使用哪个生成器最简单的办法，是给类添加一个自定义的 **Builder** 属性，并在其中填入要被用作生成器的类的名称和该类所在的类库，使用“类库，类”的格式。

现在让我们看一个很有用的生成器的例子。**SFFile** 是在 **SFCCtrls.VCX** 类库中的一个类，它提供的是一个“用户可以输入文件名、或者单击一个按钮来显示一个他或她可以从中选择一个文件的对话框”的控件。这个类包含一个 **Label**，它提供对控件的一个提示；一个 **textbox**，用于显示文件名；一个在 **SFButton.VCX** 中的 **SFGetFile** 类的实例，它显示一个打开文件对话框。

要使用 **SFFile**，简单的把它拖放到一个表单上，并设置一些属性，例如 **cExtensions (**一个在打开文件对话框中允许的扩展名列表**)**、**cCaption(**打开文件对话框的标题**)**、和 **cControlSource**（根据需要用于绑定控件的某些东西）。不过，这里还有一对问题：

- 为了设置 **label** 的 **Caption**，你需要在 **SFFile** 对象上单击鼠标右键，选择编辑，然后在 **label** 上单击、打开属性窗口、选择 **Caption** 属性、并输入一个标题。当然，然后你可能还需要移动 **textbox** 和 **SFGetFile** 按钮以为 **Label** 腾出空间，并且如果 **SFFile** 对象的大小不够的话可能还要缩放它。

- 如果你想要放大或者缩小 **SFFile** 对象的话，你首先必须缩放它，然后进入它内部缩放文本框的大小并将 **SFGetFile** 按钮移动到适当的位置。

这些工作都不是什么“杀手工作”，但是它们确实都需要花一点时间去处理。所有这些小小的“花一点时间”的任务加起来就不得了了。任何能够提高我们生产率的事情都是受欢迎的，尤其是象这种琐碎的工作。

这里是为 **SFFile** 建立一个生成器的步骤：

1. 由于这个类已经有了一个自定义的 **Builder** 属性了，所以你就不需要再加了。不过，还是要在其中填入被用作生成器的类的名称及其所在的类库名，使用“类库，类”的格式。

2. 用你在 **SFFile.Builder** 中指定的类和类库名称建立一个 **BuilderForm** 的子类。将它的 **Caption** 设置为 **SFFile Builder**。

3. 添加一个 **BuilderLabel** 对象，并把它的 **Caption** 属性设置为 **\<Label**。

4. 在 **Label** 的边上添加一个 **BuilderTextbox** 对象，将它的 **ControlSource** 属性设置为 **Thisform.oSource.lblFile.Caption**，并将下面的代码放入它的 **Valid** 方法。（**Thisform.oSource** 是对被生成器维护的对象的一个引用，因此你可以通过这个引用访问该对象的任何属性或者成员对象）。这段代码会根据需要自动调整在 **SFFile** 中的 **textbox** 的 **left** 和 **Width** 属性以匹配 **label** 的 **Caption**：

```
with Thisform.oSource
  .txtFile.Left  = .lblFile.Width + 5
  .txtFile.Width = .cmdGetFile.Left - .txtFile.Left
endwith
```

5. 添加另一个 **BuilderLable** ，并把它的 **Caption** 设置为 **\<Width**。

6. 在新 **Lable** 的边上添加一个 **BuilderSpinner** 对象，将它的属性设置为 **Thisform.oSource.Width**，并将下列代码放入它的 **Valid** 方法。这段代码会调整在 **SFFile** 中的 **SFGetFile** 按钮的位置和 **Textbox** 的宽度以匹配该控件新的宽度：

```
with Thisform.oSource
  .cmdGetFile.Left = .Width - .cmdGetFile.Width
  .txtFile.Width   = .cmdGetFile.Left - .txtFile.Left
endwith
```

7. 保存并关闭生成器类。

现在我们来测试一下新的生成器，把一个 **SFFile** 对象拖放到一个表单上，单击鼠标右键选择生成器。改动 **Label** 的 **Caption** 并设置 **Width**，然后观察控件相应的调整。图 **3** 显示的是操作中这个控件的样子：

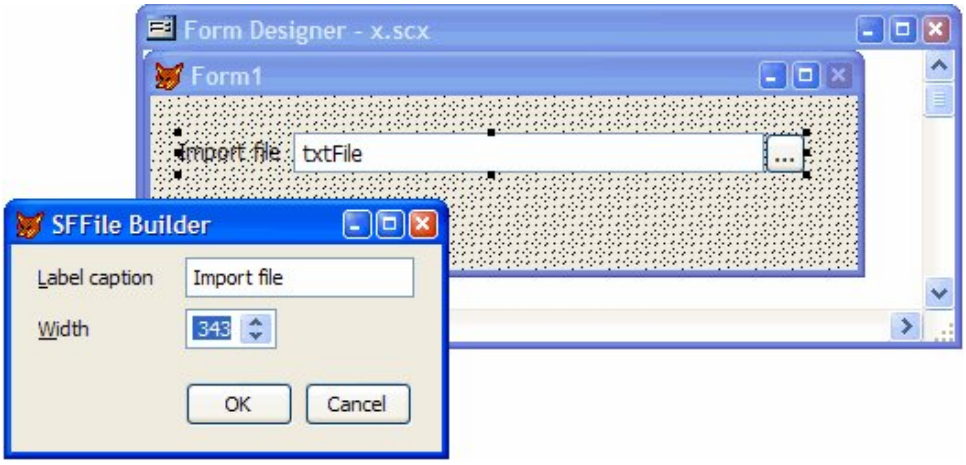

图 **3**

注意自 **VFP 8** 被发布后，这个生成器框架已经有了一些改动，所以在附带的下载文件中带有一个升级版本（该文件是 **VFP 9** 自带的）。

总结：

有趣的是，**VFP** 自带的许多工具是用 **VFP** 自己来写就的，但更酷的是，事实上我们拥有它们的源代码。这就使得“对这些工作做一些改动以满足你自己的需要、或者甚至在你自己的应用程序中重用来自这些工具的部分代码”成为可能了。在这个三部分组成的系列中，我讨论了在 **XSource** 中我认为有用的一些部件，但我还是鼓励你去在 **XSource** 目录中进行自己的探索以发现有用的技术或者代码。

附带的下载文件：**412HENNIG.ZIP**

来自 VFP 开发团队的 TIPS

原著：微软 VFP 开发团队

翻译：LQL.NET

每个月，VFP 开发团队成员都会给你带来有趣的 VFP 技术 TIPS。这个月带给你的是一些酷酷的新功能，比如同时从缓冲区和非缓冲区用 SQL SELECT 获取数据、用新的二进制类型索引提高速度等等。

同时从缓冲区和非缓冲区用 SQL SELECT 获取数据

VFP9.0 SQL SELECT 命令新提供的 WITH (BUFFERING = .T.) 子句允许你从缓冲数据中获取数据。下面的示例中 SQL SELECT 中只用了一个表，这样可以让你比较一下缓冲数据和磁盘上的数据。

```
CLOSE DATABASES ALL
OPEN DATABASE Northwind
CLEAR
SET MULTILOCKS ON
USE customers IN 0
CURSORSETPROP("BUFFERING",5,"customers")
UPDATE customers SET city="Seattle" ;
  WHERE city LIKE "B%"
* view buffer, unbuffered for same table
SELECT DISTINCT c.city, c2.city ;
  FROM customers c WITH (BUFFERING=.F.) ;
  JOIN customers c2 WITH (BUFFERING=.T.) ;
  ON c.customerID = c2.customerID
=TABLEREVERT(.T.,'customers')
```

一个测试，用来证明二进制类型索引带来的速度提升

在 VFP9 中运行一下下面的代码，看看普通索引和二进制索引效果有什么不同。

```
* BINARY (size/speed) vs. General index
CLEAR
```

```
CLOSE DATABASES all
DELETE FILE test.dbf
CREATE TABLE Test (f1 C(1))
FOR I=1 TO 10000
  APPEND BLANK
NEXT
INDEX ON DELETED() TAG deleted1
FLUSH FORCE
?"General index."
DIR test.CDX
nSeconds1=SECONDS()
DELETE ALL
RECALL ALL
nSeconds2=SECONDS()
?"Time elapsed:",nSeconds2-nSeconds1
FLUSH FORCE
DIR test.CDX
?"========Binary index.=========="
DELETE TAG all
INDEX ON DELETED() TAG deleted2 BINARY
FLUSH FORCE
DIR test.CDX
nSeconds1=SECONDS()
DELETE ALL
RECALL ALL
nSeconds2=SECONDS()
?"Time elapsed:",nSeconds2-nSeconds1
FLUSH FORCE
DIR test.CDX
CLOSE DATABASES ALL
ERASE test.dbf
ERASE test.cdx
```

给你的自由表加上事务，拥有回滚的能力

用 VFP9 新的 MAKETRANSACTABLE()函数让你的自由表拥有事务和回滚的能力。

```
CLEAR
CLOSE DATABASES ALL  && look ma, no DBC!
ERASE test1.dbf
CREATE TABLE test1 (ID INT AUTOINC, DATE D)
MAKETRANSACTABLE()
BEGIN TRANSACTION
FOR i = 1 TO 10
   INSERT INTO test1 (Date) VALUES (DATE())
ENDFOR
? RECCOUNT()
ROLLBACK
? RECCOUNT()
```

示例文件：412TEAMTIPS.ZIP

用集合设计，第二部分

作者：Randy Pearson & Lauren Clarke

译者：Fbilo

这是由三部分组成的用集合设计系列文章的第二部分。在第一部分中，Randy Pearson 和 Lauren Clarke 检讨了一些他们在使用了两年的集合以后所学到的经验。这个月，他们将通过演示集合可以怎样被用来建立接口灵活的 Visual FoxPro 对象模型，来将这些经验应用到工作中去。

Collection 基类的引入，在设计类的层次结构方面向我们提供了更多的能力，在这种结构中，可以通过使用对象模型来引用所有的对象。这种能力使得我们能够为程序员们设计出更符合潮流的设计和更容易使用的接口。

在这篇文章中，我们要讲一个设计的示例，它来自一个对 **Visual FoxPro** 程序员来说非常著名的领域：数据库容器。在开始前，先让我们回忆一下这个系列文章第一部分中的两个观点：

- 集合构成了面对对象的黏合剂，它让全然不同的类们可以被吸收入一个更容易访问的结构（一个“对象模型”）中去。
- 集合应该作为其它对象的成员被发布，增加的行为应该封装在父（容器）对象中。

在这个月的示例中我们将遵循这两个观点的指导。在第三部分中，我们将探讨由此导致的任何限制，并考虑扩展 **Collection** 基类自身行为的技术。

一个 DBC 的 DOM

在设计软件系统的过程中，一个常见的需求是给一组有一个逻辑的、层次结构的实体们建模。一个例子是一个文件系统，在这个系统里逻辑驱动器拥有文件夹，而每个文件夹里可能同时拥有文件和子目录。另一个例子是一个由菜单、子菜单、各个菜单项组成的菜单系统。所有的 **Microsoft Office** 软件包中的产品在对文档进行工作的时候都使用了一种复杂的对象层次结构。

在这篇文章中，我们想要选择一个 **Visual FoxPro** 程序员们都理解的领域来建立一个对象模型。我们选择了数据库容器（**DBC**）。数据库容器虽然定义的很好，但用起来总是有些不便，而且不是面对对

象的。举个例子，要访问在 **Orders** 表中的 **Freight** 字段的 **Caption** 属性，在读取这个属性之前，你必须首先确保该数据库已经被打开并且被选中了：

图 **1**

```
?DBGETPROP("ORDERS.FREIGHT", "Field", "Caption")
```

如果我们有了一个 **DBC** 的对象模型，我们就可以使用下面两种方案之一来访问同一个属性：

```
?oDBC.Fields("ORDERS.FREIGHT").Properties('Caption')
?oDBC.Tables('ORDERS').Fields('FREIGHT').;
Properties('Caption')
```

尽管这种对象模型的途径在读取一个属性方面也许并没有什么优势，但是考虑一下以下好处：

- 一旦加载以后，在读取属性值之前，你就不再需要去打开或者设置当前数据库了；
- 对象模型让你可以使用 **FOR EACH** 遍历所有的集合，例如所有的表；
- 通过设计你自己的类，接口可以被增强到超过 **Visual FoxPro** 函数所允许的水平。

最后一点可能需要更多的解释。假设你需要知道一个字段的标题、数据类型、和长度。这个标题是被储存在 **DBC** 中的，而且可以通过使用 **DBGETPROP()** 来读取，可另外两个却只能通过打开表然后使用 **AFIELDS()** 或者 **TYPE()** 来判定。而有了一个基于集合的 **DOM** 以后，我们就可以应用上一些需要的语法来平滑的访问数据库、表和字段，而放弃那些通常写代码时必须的笨拙写法。然后这个设计将让你可以使用同一种语法访问另两个属性：

```
? oDBC.Fields("ORDERS.FREIGHT").Properties('Caption')
? oDBC.Fields("ORDERS.FREIGHT").Properties('Type')
? oDBC.Fields("ORDERS.FREIGHT").Properties('Length')
```

如果没有这种模型，代码可能会是这样的内容：

```
SET DATABASE TO Northwind
? DBGETPROP("ORDERS.FREIGHT", "Field", "Caption")
USE Orders AGAIN ALIAS __Orders IN SELECT("__Orders")
AFIELDS(laFields, "__Orders")
? TYPE("__Orders.Freight")
? laFields[ASCAN(laFields, 'FREIGHT', 1, -1, 1, 15), 3]
USE IN SELECT("__Orders")
```

前面的途径产生了更容易维护的代码，并去掉了令人头疼的记住哪个属性储存在数据库里而哪个又需要使用 **AFIELDS()** 来返回的问题。

了解了这些好处以后，现在我们来设计一个 **DBC** 对象模型的第一块。这个第一次尝试仅涉及数据库，加上它的表、字段和视图。我们的目标，不是建立一个完整的产品，而是演示集合可以怎样被用来

建立一种类似的对象模型。

在我们的对象模型中，我们不是去为每一种节点的类型（数据库、表、字段、视图）设计一个完全独立的类，而是采取了一种乐观的观点，即它们应该有一些通用的行为，允许某些代码可以被重用。因此，该设计使用了一个抽象的 **DbcDOMElement** 类，其它的类都从这个类来继承。表 **1** 是在我们模型中类们的列表：

表 1、在 DBC 文档对象模型中的类

类名	用途
DbcDOMElement	抽象类
DbcDOM	用于数据库容器本身的根节点
DbcDOMTable	包含在数据库中的每个表
DbcDOMView	包含在数据库中的每个视图
DbcCOMField	在一个表中的每个字段

这里是一个发布和使用这些在一个对象模型中类们的示例，它演示了这个工作会是多么的简单：

```
SET PROCEDURE TO CollectDBC ADDITIVE
LOCAL lcDbc, loDbc, loItem
lcDbc = HOME(2) + "northwind\northwind.dbc"
loDbc = CREATEOBJECT("DbcDOM")
loDbc.Load(m.lcDbc)
```

在这里，对象们都被加载，并且可以通过 **DOM** 来访问。现在让我们来使用这个模型：

```
? "数据库中表的数量:", loDbc.Tables.Count
FOR EACH loItem IN loDbc.Tables
? "Table:", loItem.cName, ;
"备注:", loItem.Properties("Comment")
ENDFOR
```

在我们的模型中，集合被使用在某几个位置。首先，数据库类有两个集合，一个用于表，一个用于视图：

```
DEFINE CLASS DbcDOM AS DbcDOMElement
cType = "DATABASE"
ADD OBJECT Tables AS Collection
ADD OBJECT Views AS Collection
```

与此类似，表类（**DbcDOMTable**）有一个字段集合。对于每个这样的集合，在 **DBC** 层次中的“父

类”有一个或多个方法是被设计为用来建立正确的“子”对象、并将子对象加载到集合中去的。例如，数据库类中有 **LoadTables()** 和 **LoadViews()** 方法。

在每个节点上的一个属性集合

除了象 **Tabels** 和 **Fields** 这样的名称集合以外，每个类都是从 **DbcDOMElement** 类派生出来的，而后者自身就有一个 **Properties** 集合。这个集合与其它三个属性（**cType**、**cName**、和 **cProps**）以及 **LoadProperties()** 方法结合起来提供了我们所期望的可重用代码。让我们看一下下面这段代码，它位于抽象类中，其功能可以用于表、字段和视图。

```
DEFINE CLASS DbcDOMElement AS Container
cType = NULL && 必须覆盖
cName = "" && 在 Load 方法中设置
cProps = "" && 用逗号分隔的 DBC 属性列表
ADD OBJECT Properties AS Collection
* ------------------------------------------------ *
FUNCTION LoadProperties(toRef)
LOCAL lnWord, lcWord, lvVal
WITH THIS
FOR lnWord = 1 TO GETWORDCOUNT(.cProps, ",")
lcWord = GETWORDNUM(.cProps, m.lnWord, ",")
lvVal = DBGETPROP(.cName, .cType, m.lcWord)
.Properties.Add(m.lvVal, m.lcWord)
ENDFOR
ENDWITH
ENDFUNC
```

简而言之，每种类型的对象都包含它自己的一套用逗号分隔的 **DBC** 属性列表。这个列表在 **LoadProperties()** 方法被调用的时候被分析。每个属性都用 **DBGETPROP** 来从 **DBC** 中查询到，然后被作为 **Properties** 集合的数据项存储起来。

一种可以考虑的替代办法是使用 **ADDPROPERTY** 来把每个 **DBC** 属性简单的当作一个相应 **DOM** 对象的属性。尽管这种办法在简化接口方面有一些优点，用这种办法你将不需要去显式的使用 **Properties** 集合，但我们还是使用集合的设计，因为前者可能与 **Visual FoxPro** 内建的属性名称存在着潜在的冲突。

除了 DBC 属性之外的属性

前面我们提到过使用集合的方式在引用不存储在 DBC 中的属性方面可以通过使用同样的接口而具有潜在的优势。对于字段，这个目标的实现是通过向 Properties 集合添加用于类型、长度、小数位的数据项来实现的。事实上这些代码写在表类而不是字段类中，因为在表类中使用 AFIELDS() 可以临时的获得这些信息。查看本月下载文件中 DbcDOMTable 类的 Load 方法以了解这个目标是怎样实现的。

在多个集合中引用一个对象

在这个系列文章第一部分中我们曾说到：“在设计对象模型的时候，同时在多个集合中引用同一个对象是很常见的事情”。我们已经通过给我们的 DOM 增加一个省力的功能将这个理念用到工作中了，靠这种办法，你可以直接访问一个字段对象而不需要使用 Tables 集合。

这个目标的实现只要求两个小改动：

8. 1、给数据库类增加一个 Fields 集合；
9. 2、当每个字段对象被建立的时候，除了将其引用添加给相应表对象的 Fields 集合以外，还要添加给数据库对象的 Fields 集合。为了提供这种省力的需求，只要在 DbcDOMTable 类的 Load() 方法中增加一行代码，该代码就能增强集合所能提供的能力和简便性！

```
* 将对象添加给表的 Fields 集合:
THIS.Fields.Add(m.loObj, m.lcFld)
* 还要添加给数据库的 Fields 集合:
toRef.Fields.Add(m.loObj, ;
THIS.cName + "." + m.lcFld)
```

此外，还要注意当同一个对象被添加给不同集合时所使用的不同的 key 值。一个数据项被用于几个不同集合时没有理由使用同样的键值。而且在这个案例里有个使用不同键值的很好的理由，即使在多个表中出现同样的字段名，数据库级别的字段集合仍然可以正常的工作而不出现错误。我们的设计简单的使用类似于别名的前缀作为 key 的一部分，这样就可以使用类似与以下的语法直接访问一个字段：

```
? oDBC.Fields('ORDERS.FREIGHT').Properties('Caption')
```

而在我们增加这个功能前，就需要一种多层的嵌套：

```
? oDBC.Tables('ORDERS').Fields('FREIGHT'). ;
```

```
Properties('Caption')
```

XML

在这篇文章中，我们已经使用了 **DOM(Document Object Model,**文档对象模型**)**这个名词，尽管这里从来没有一个物理文档。事实上，拥有一种将一个 **DBC** 中的信息作为一个文档描述出来的途径可能是一个非常有用的功能。**XML** 能够为说明层次结构的信息提供一种非常好的格式，所以我们的下一步增强是为我们的 **DOM** 添加这种能力以生成一个它自己的文档。

在文档中的每个节点都包含一个 **Serialize()** 方法，通过遍布在 **DOM** 中的这个方法，我们就可以生成一个完整的文档。一旦 **DBC** 对象被加载以后，可以使用下面这样的代码来建立 **XML** 文档：

```
STRTOFILE(loDBC.Serialize(), "Tastrade.XML")
```

Serialize() 是一个调用低层次的 **SerializeProperties()** 方法的模板方法，同时还提供了一个空的 **SerializeItems()**方法、以及两个 **before** 和 **After** 挂钩方法，以向不同类型的节点提供让它们可以自己定制的能力。尽管跟集合的题目没什么关系，你也许还是会想要看看用于生成 **XML** 输出的技术。在本月下载文件中的 **Tastrade.XML** 文件提供了一个这种输出的例子。

有了你的 **DBC** 的一个 **XML** “快照”之后，你就可以使用象 **XPATH** 那样的技术来对你的 **DBC** 进行基于相当复杂标准的信息查询了。下面的代码示例会从 **DBC** 中选出所有默认值不为空的字段：

```
loXMLDOM.loadXML( loDBC.Serialize() )
loProps = loXMLDOM.selectnodes(
'//FIELD/properties/property[' + ;
'@name="DefaultValue" and string-length(@value)>0]')
FOR EACH loProp IN loProps
loField = loProp.ParentNode.ParentNode
? loField.getAttribute("name"), ;
loProp.getAttribute("value")
ENDFOR
```

这段代码会生成一个很方便的数据库中所有默认值设置的列表：

```
CATEGORY.CATEGORY_ID newid()
EMPLOYEE.EMPLOYEE_ID newid()
EMPLOYEE.PASSWORD "Tastrade"
ORDERS.ORDER_ID newid()
ORDERS.ORDER_NUMBER newid("order_number")
ORDERS.ORDER_DATE DATE()
ORDERS.DELIVER_BY DATE()+7
```

```
ORDERS.EMPLOYEE_ID defaultemployee()
ORDER_LINE_ITEMS.QUANTITY 1
PRODUCTS.PRODUCT_ID newid()
SHIPPERS.SHIPPER_ID newid()
SUPPLIER.SUPPLIER_ID newid()
```

直到下一次

尽管我们可以用少量的代码就建立一个相当强大的模型，使用 **Visual FoxPro** 的 **Collection** 基类还是存在着一些限制。例如，由于集合使用大小写敏感的事实，强制要求你的访问 **DOM** 的代码需要知道在属性被加载的时候使用了哪种大小写形式。在第三部分中，我们将会把注意力转向在这方面以及其它情况下有帮助的建立集合的子类的技术。

下载文件：**412PERSON.ZIP**

表格列交换器

原著: Dragan Nedeljkovich

翻译: CY

在他先前的文章里，Dragan Nedeljkovich已经介绍了“半手工生成器”的概念，那些是简单的小小实用工具，可以帮助你变成为一个多产的VFP开发者。最后，他为你演示了如何创建最简单的生成器。现在来做更严肃些的，创建表格列交换器。

我为这篇文章精选的生成器，可以交换表格中的列。在如下这样的情况下，我的表格的方法里有多个的代码，并且是通过VFP以nIndex参数来传递（你可以这样认为，nIndex可以是“列号”，但是它真正的意思是“列序号”）。这个代码是处于框架级，可以交换列到不同的位置，以前都是需要重建表格的。

生成器做了什么？它仅是交换了两个列—只是在程序里实际交换是做不到的。生成器在内存里使用了最大的工作区：它为第一个列创建了一个另外的副本，复制第二个列到第一个列里，再复制另外的列到第二个列里，最后删除另外的列。生成器完整的代码（含注释）在本文的下载里，我仅在这里解释最重要的部分。

这是生成器的程序主体：

```
* colswap.prg
Lparameters og As Grid, nCol1, nCol2
Local cName1, cName2
og.AddColumn()
og.Columns[og.ColumnCount].Name="coltemp"
objclone(og.Columns[nCol1],og.coltemp)
objclone(og.Columns[nCol2],og.Columns[nCol1])
objclone(og.coltemp, og.Columns[nCol2])
cname1=og.Columns[nCol1].Name
cname2=og.columns[nCol2].Name
og.Columns[nCol1].Name = Sys(2015)
og.Columns[nCol2].Name=cname1
```

```
og.Columns[nCol1].Name=cname2
og.RemoveObject("coltemp")
```

首先，我需要传递表格自身作为参数(og)。怎么做？很简单—用我上次介绍的最简单的生成器。我假定在你的路径里已经有cuform.prg。如果还没有，这里就是：

```
* curform.prg
ASELOBJ(aa,1)
RETURN aa[1]
```

现在，点击你的表单项在它上面的任何顶层对象（表格自身也可以），然后在命令窗口里输入如下：

```
ox=curform()
og=ox.grid1
```

即使你的表格是长名，如grdGridWithWrongOrderOfColumns，智能感知也会在几次击键后找出它来，它是很有帮助的。于是我现在已经有了og，它是一个引用我正在编辑的表格对象，并且我正要交换第2列和第5列：

```
Do colswap with og, 2, 5
```

可以很容易的见到生成器所做的：它加入一个列并给它命名为“coltemp”，为什么样要改名？或许它会被命名为Column7或者其他类似这样的，但是我的代码将不得不做大量的evaluate()调用，或者创建一个新的变量以引用这个列。为了透明的缘故，我只好改变它的命名并且以新的命名来引用它。

克隆对象

下面三行是最常用的交换内容游戏，正如你要交换两个变量的内容，a和b：

```
c=a
a=b
b=c
release c
```

我也是使用这个模型来交换列，但是这里“a”是我要交换的第一列，“b”是第二列，而“c”是在交换操作过程中用到的新的临时列。对于这个命名，是分别处理，因为你并不是象这样的指定一个列的

名给另一个列。如果你在任何时候试图把两个列命名为相同的名（在设计时或运行时），VFP将会产生错误。

同样，我不能只使用赋值—我已经编写了一个objclone()过程（可见于同名PRG）以创建一个列的副本。注意它并不是命名为columnclone()，因为它可以工作于任何类型容器对象（我已经把它试用于页框，并且它复制到了从第3页到第7页，包含标题和文本框里的代码）。

Objclone()是如何工作的？它使用了三个循环。第一个循环将目标对象的所有属性复位为默认值，并且移去了它所有的成员对象。这里包含了错误检查，因为部分的属性可能是固定的，并拒绝被复位的，会对报错的。这些仅需要保留原样。

第二个循环从源对象里复制了所有的成员对象到目标对象里，通过以递归的方法调用objclone()来创建并复制它们所有的PEM。

第三个循环遍历了源对象的所有非默认值的PEM，并复制它们到目标对象，当然除了.Name属性。其它例外是成员对象，它们是在第二个循环是被复制的。除此以外，包括方法代码，也被复制。最后，成员对象被从源对象里移出。这也不是必须的，因为这些将会在第一个循环里被移去，当那个对象被作为目标对象来运行时。

对象外部排序的最重要工具是AMEMBERS()函数，它有重要的最后一个参数，cFlag。我已经把它设为“UC”（用户自定义和/或改变的PEM），它为我节省了大量的PEMSTATUS()检查，和许多的叠代循环次数。它是一个切割工具，在后面又会把它缝补在一起，这些常用的
是 .NewObject()、.RemoveObject()、 .AddProperty()、.ResetToDefault()、.ReadMethod()、.WriteMethod()，当赋值一个属性时，每一个都使用名字表达式以避免宏替换：

```
Store Evaluate("o1."+cWhatName) To ("o2."+cWhatName)
```

回到原始问题：当我们把列加入表格时，如何让它们显示的顺序变成等同于它们的索引？你只需要按照你的需要来多次交换列，或者你可以扩展你的半手工生成器来完成它。

查看，捕捉，建档

原著: Richard D. Hodder

翻译: CY

捕捉GUI屏幕图像来做文档，是件费力和乏味的事。在这篇文章里Rick Hodder讲解了一个通用屏幕捕捉程序的自动接口，Snagit，并演示如何遍历项目里的表单，生成一个屏幕快照手册。

正视它，我们都讨厌创建文档。捕捉可视信息和文本用于帮助手册、CHM文件、开发指南等等，是件艰巨的任务。这个任务随着加入到应用程序里的每一个表单而增加，并且随着应用可视界面的改变，整个过程的内容都要重复。

SnagIt!

SnagIt是TechSmith公司的一个令人惊讶的图像捕捉和处理的实用工具。简单的说，SnagIt是在皮质类固醇上的屏幕打印。如果SnagIt还没有加入你的开发工具包，那它就应该要加入的。如果你已经有了自己的SnagIt，请确认你已经是最新版本（V7），它有着奇异的新特性；图1-3展示了产品的作用。你可以从www.techsmith.com/products/snagit下载30天的试用版。这个产品单用户版仅花$39.95，升级版为$29.95—这是很便宜的，正如你所见到的。

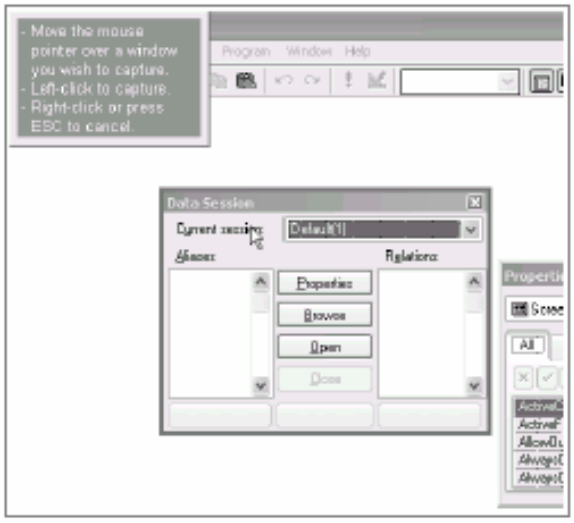

图1:窗口捕捉演示了窗口选择器和帮助窗口。

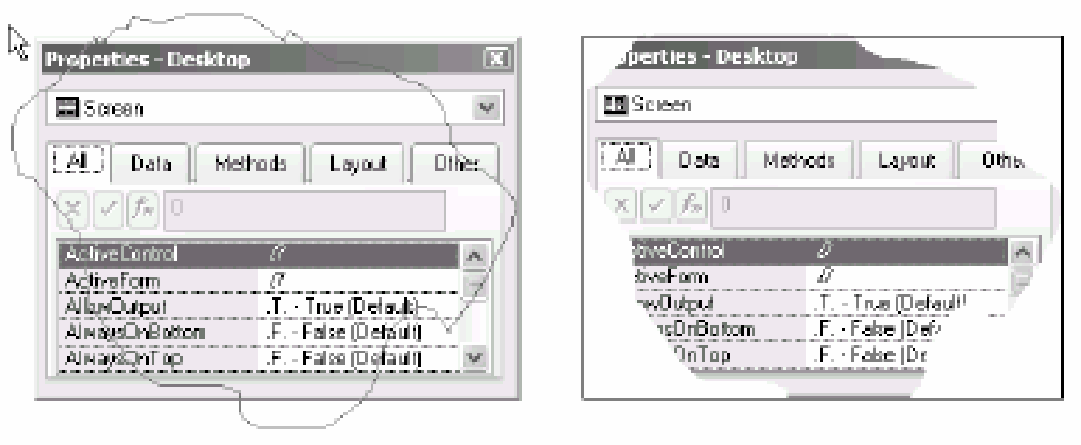

图2:自由手绘形状捕捉演示捕捉与结果。

图3:菜单捕捉包括菜单条，但仅限于弹出框的宽度。

本篇文章将聚集于这个产品的COM自动服务方面，通过一个简单的自动示例来遍历项目的表单，捕捉表单图形为文件，然后组合一个快速的基于HTML的“手册”。COM服务器的简要概述可以在www.techsmith.com/products/devtools/snagitcomserver.asp上找到。关于COM服务器帮助文件有CHM和PDF格式，与示例一样，可以在www.techsmith.com/download/comserverdefault.asp上找到。

这里是利用内置于Windows的PrtScrn功能来捕捉单个表单的步骤：

1.运行表单
2.按PrtScrn键
3.打开MS Paint
4.粘贴到Paint
5.修剪图形
6.保存图形

当下次你改变了你的应用程序屏幕的部分内容时，你要重新从第1步开始。这看起来好象并不是很多步骤，但是乘以30或更多的你的应用程序表单时，将会有大量沉闷的工作等着你。

现在来看看当使用SnagIt时的下列步骤：

1.运行表单
2.按下Ctrl+Shift+P
3.保存图形

简单多了！你还可以设置诸如色数、色调、对比度，甚至是捕捉文件存放的文件夹。这大大地简化了过程，在看到实际节省的时间和工作，你就会想用这个自动特性的。

注意：这篇文章并没有全面的讲解SnagIt，但是你完全可以相信这个产品的许多特性（编辑工具、组织工具、增加结构、多种捕捉类型，和图形捕捉文件），这样的低费用对于这个软件包是值得的。关于特性的更多细节和关于输入、输出和过滤选项更多的详细说明，可参见我已经包含在本文的下载文件里的单独DOC文件。

自动化

我是自动化的极大爱好者。首先，我将告诉你一些关于我潜心于自动化接口的问题，然后我将会给你代码以文档化你的项目。

体验

就象是圣诞节的小孩，我深入于SnagIt COM服务器的帮助文件，但是我发现它稍微有点混乱。帮助文件把COM服务器介绍为一系列分离的API（接口）而不是作为对象。它感觉就象是试图看着一辆拆散的汽车来学开车。

比如，有一个名为IImgeCaputre的接口，它带有一个Capture方法，但是该方法却没有任何属性可以让我用于寻找捕捉图像的对象。它没有输出文件名属性和输出文件类型属性。更让我混乱的是，有一个名为IImageFile的接口，它带有我希望IImageCapture所要有的所有任何属性，但是它却没有Capture方法。

我想我将不得不创建一组VFP类以实现组合这些接口。并且，帮助文件里IImageCapture接口的主题有个子标题“包含接口”，它说IImageCapture的一个包含接口是OutputImageFile（IImageFile）。

当我看完帮助文件，我怀疑包含接口或许实际上就是对象属性。我决定放置COM服务器到一个表表单上以作演示，但是发现SnagIt库并没有出现在ActiveX控件列表里以放置到表单上。然后，我打开VFP里便利的对象浏览器，点击按钮以打开类型库，并选择COM库选项。SnagIt并没有出现在列表里，于是我点击浏览按钮并指向ProgramFiles\Techsmith\SnagIt7目录下的SnagIt32.EXE，最后我访问到了SnagIt的信息。

注意：当我最初下载COM服务器信息时，帮助信息里的示例告诉我如何具体例化捕捉对象，但是却没有说怎么用它。TechSmith最近已经加入完整的示例到网站上（没有VFP的）.

我查看了类而不是在浏览器里查接口（最终你例化的将是类，而不是接口）。我查看了ImageCapture

类，并发现OutputImageFile是一个对象属性，它实现了IimageCapture接口（正如我所怀疑）。

你可以自动化做什么？

这里有两个类可以直接具体例化：ImageCapture和Textcapture（你也可以例化其他的，但SnagIt会挂起）。类名的第一部分（“Iamge”和“Text”）指明了捕捉的输出：调用ImageCapture类的Capture方法返回捕捉的图像。调用TextCapture类的Capture方法返回捕捉的文本块。

可以注意到没有VideoCapture和WebCapture。自动接口仅可以捕捉图像和文本。如果你对自动视频捕捉有兴趣，TechSmith在网站上有Screen Record SDK（www.techsmith.com/download/sdkfreetrial.asp）。你可以利用Rick Strahl的West Wind Client 工具结合SnagIt来创建你自己的网站捕捉（www.west-wind.com/wwClientTools.asp）。

当开始用SnagIt捕捉前，你必须告诉SnagIt你要捕捉什么（输入）和放在哪里（输出），并且设置相应的属性。比如，如果你告诉捕捉对象通过邮件发送输出，你要设置outputEmailOptions对象属性的邮件地址、标题和信息文本属性。

捕捉对象只有一个方法：Capture。捕捉对象的属性很简单，正因为很简短，我把它们一起介绍。属性的类型出现在括号里跟在属性名后。下列数值属性的有效取值列表可以在附随的下载文件的包含文件SnagIt.H里找到。

• 输入（N）--告诉SnagIt要捕捉什么。所有的图像捕捉常数在SnagIt.H里都是以“sii”开头（比如，siiWindow代表捕捉窗口图像）。所有文本捕捉输入常数都以“sti”开头（例如，stiWindow代表捕捉窗口文本）。这个属性的某些更有趣的常数包括可以捕捉窗口的壁纸，从图像文件里捕捉图像，通过定义多边形捕捉图像，捕捉剪粘板，从TWAIN设备捕捉（象数码照相机扫描仪），并可以捕捉滚动窗口—实际上它是滚动窗口以捕捉所有内容。

•输出（N）--告诉SnagIt捕捉结果发送哪里。所有的图像捕捉输出常数在SnagIt.H里都以“sio”开头（比如，sioFile意味着捕捉的内容将被写入到一个图像文件）。所有文本捕捉输出常数在SnagIt.H里都以“sto”开头（例如，stoFile意味着捕捉的内容将被为写入到一个文本文件）。这个属性的常数可以发送捕捉的信息到打印机、剪粘板、FTP服务器、e-mail地址等等。

•包含光标（L）--告诉SnagIt是否在捕捉里包含鼠标光标

•捕捉多区域（L）--告诉SnagIt是否在一次捕捉里捕捉多个区域

•背景色（N）--仅应用于图像捕捉。它是一个RGB值，用以作为捕捉区域外的背景色

•允许预览窗口（L）--允许为下次捕捉使用SnagIt预览窗口。如果设为.T.，将允许用户在输出前查看捕捉的图像

•前景预览（L）--告诉SnagIt强制捕捉窗口到顶层（仅当EnablePreviewWindow 为.T.时可用）

• 使用放大镜窗口（L）--若允许时，当定义精确区域捕捉时将在光标周围出现放大镜窗口。这个属性仅在区域输入模式下有效。

下列是对象属性：

• 自动滚动选项--包含对滚动捕捉的设置。你可以指定你所需要的滚动类型（无，水平，垂直，两者）和滚动行的延迟

• 剪粘板选项--包含相关于全屏幕DOS和剪粘板捕捉输入的属性（WidthInPixels）。设置值为零将告诉SnagIt使用Windows桌面的宽度和高度作为图像尺寸

• 延迟选项--允许捕捉在指定的数秒后被取消。这个对象也有一个逻辑属性以允许窗口倒计数。

• 输入菜单选项--包含所有对菜单捕捉的设置（输入模式设为siiMenu）：捕捉级联菜单和弹出菜单上的菜单条的属性

• 输入对象文本选项--包含所有文本对象捕捉的设置

• 输入区域选项--包含所有区域捕捉的设置（输入模式设为siiRegion或stiRegion）。它包含定义区域的属性：左上角坐标，高度和宽度

• 输入TWAIN选项--包含所有TWAIN捕捉的设置（输入模式设为siiTWAIN）。它也有一个方法来拉出对话框以选择TWAIN设备

• 输入窗口选项--包含用于捕捉窗口时的设置。属性包含如如何选择窗口（抓取活动窗口，让用户选择窗口，利用HWND来选择窗口，或者选择位于X,Y坐标的窗口）。

• 输出FTP选项--包含用于发送输出到FTP网址时的设置。这个对象有服务器名、文件名、端口、用户名、密码和代理服务器设置的属性。它甚至可以配置以显示一个进度条。

• 输出图像文件--允许输出图像文件的配置。你可以指定如文件类型（23种文件类型），文件名，和色数。

• 输出邮件选项--包含相关于e-mail输出的设置。你可以指定姓名，邮件地址，主题，和信息文本。你甚至可以让SnagIt拉出窗口以提示用户这些数据项

• 输出打印机选项--包含用于打印机输出的属性。它有个方法可以用于设置输出打印机到其它非默认打印机。

• 输出打印机设置选项--包含指定格式化输出到打印机的属性。这些属性是页边空，输出页的位置（左上，右上，等等），和比例

• 文本文件选项--包含所有输出文件文件的设置（stoFile）。输出的文件，不管使用的何种文件命名方法，都将会是.txt扩展名。

自动化示例

我最初到TechSmith的网站是因为我正在给我的VFP项目做文档。这个项目是不断变动的，用户评估用户接口并加以改变，我并不想维护这个有70个表单的应用程序的文档。当我看到SnagIt可以自动做，我的思想就开始嗡嗡作响。

几天内我就能自动捕捉表单的屏幕快照，并生成一个屏幕快照的文档以发布给测试员以标记文档。他们非常喜欢。我甚至可以为客户生成一个PowerPoint简报。

这也是一个作原型设计的很好的技术。当你会见客户并在当场在VFP里创建一个原型应用程序。事后，

在你的酒店房间里，你开始运行代码捕捉表单就可以在睡觉前做好文档。然后，第二天在客户那儿，在你给他们演示你所编制的丰富文档时，你就可以揉揉你疲惫的双眼，擦干你额头的汗。

在附随的下载里，你可以找到名为text的项目，它包含一个基于SCX的表单，一个名为test的类库，它包含有三个表单类，还有一个名为test的程序。也还有一个名为findforms.prg的程序。当findform运行时，它将在项目里遍历，查找表单。它查找SCX，和类库里的表单类（查找以表单作为基类的类），和任何以表单为基类的基于PRG的表单类。

在它遍历项目时它将运行每一个表单，捕捉屏幕快照到一个目录下。

Findform.prg将为表单和它们所存储的位置信息来创建一个游标（SCX、VCX、PRG），然后在游标里循环：

```
SCAN
*-- 运行表单，但暂不显示
DO CASE
CASE cSource = "SCX"
DO FORM (ALLTRIM(cFormName)) NOSHOW ;
NAME loForm LINKED
CASE cSource = "VCX" OR cSource = "PRG"
loForm = NEWOBJECT(ALLTRIM(cFormName), ;
ALLTRIM(MLibrary))
ENDCASE

*-- 设置表彰模式以创建等待
loForm.WindowType= 1

*-- 传递参数给表单的计时器以捕捉表单
poCaptureTimer.oForm = loForm

*-- 打开计时器
poCaptureTimer.Enabled=.T.

*--显示表单
loForm.Show()
ENDSCAN
```

目前，仅注意到计时器给表单作快照。我立刻就要进入怎么做了。当首次试图捕捉表单时，我问我

自己，“何时是执行捕捉的正确时刻？”最合理的回答是，“当表单已经完成加载时”。这个变化，取决于表单上的控件的数量，动态加入表单的控件数量，和重现表单Activate、Refresh和Show方法的过程。我试着在_Screen的FormCount变化时用BindEvent()来触发事件，在表单显示后它将会重现。唉，它不能运行。

我发现在表单的Show方法里触发捕捉是最多的情况。想不到的是要有一个等待状态以触发捕捉。这就是为什么先前的代码要设置表单模式--以创建一个等待的状态。现在计时器开始上演：计时器将会在表单开始等待后立即触发，调用它的捕捉方法：

```
PROCEDURE Capture
*-- 捕捉表单图像
LOCAL loImageCapture AS SnagIt.ImageCapture
loImageCapture = ;
CreateObject("SnagIt.ImageCapture")
WITH loImageCapture
.IncludeCursor=.F.
.Input=1 && siiWindow
.Output= 2 && sioFile

WITH .InputWindowOptions
.SelectionMethod = 2 && swsmHandle
.InputWindowOptions.Handle = this.oform.HWnd
ENDWITH

*-- 存储为PNG文件并命名
WITH .OutputImageFile
.FileNamingMethod = 1 && sofnmFixed
.FileType = 5 && siftPNG
.Directory = SYS(5)+CURDIR()+"\capturedimages"

IF "\"$cFormName
lcImage = ALLTRIM(cFormName)
lcImage = SUBSTRC(lcImage,RATC("\",lcImage)+1)
lcImage = SUBSTRC(lcImage,1,RATC(".",lcImage)-1)
.OutputImageFile.FileName = lcImage
ELSE
.OutputImageFile.FileName = ALLTRIM(cFormName)
```

```
ENDIF
ENDWITH
.Capture()
ENDPROC
```

ImageCapture类已经具体例化，并且捕捉是设置为隐藏鼠标光标，所以它是不会出现在捕捉的快照上的。下一步，输入被设置为Window，输出被设置为图形文件。选择方法被设置为利用一个HWND来捕捉窗口（相对于让用户以红色矩形来选择表单）。然后对于需要捕捉的表单HWND（句柄）被设置为这个表单的HWND。输出文件类型被设置为PNG文件，输出目录被设置为在当前目录下的capturedimages文件夹，表单名用于创建快照文件名。最后，调用捕捉。非常直截了当！（注意：确认你所运行的表单的目录下有一个名为*capturedimages*的子目录。）

继续运行

这样方式的自动捕捉是很有用的，但是它却有限制，我将会在将来的文章里讨论。在那个文章里，我将以VFP脚本的自动捕捉应对下列挑战：

- 捕捉表单部分内容--有时你想捕捉表单的一部分而不是整个表单（比如，页首，页脚，或者你所想要显示的特别控件）
- 捕捉多页表单--表单可能包含有页框。这个先前的代码仅能捕捉当表单打开时页框所激活的页。为了完整文档化表单，它必须捕捉所有页的图像。另一个示例是向导表单，它有许多步骤，在特定的情况下某些内容将不会出现：例如，在增加打印机的向导，选择加入一个本地打印机就会跳过在网络上查找打印机
- 捕捉不可见内容—表单可能有不可见的或仅是在特定情况下才可见的元素。应用安全问题就是个很好的例子：你可能有仅当用户以管理员访问登录时可见的控件。
- 如何戏剧化创建文档—为什么只是文档而不是使用应用的演示？如何戏剧化将会创建一个“连续剧”的过程。“如何增加新客户”的剧情将会调用客户表单，捕捉一个图像，调用表单的新增方法，再捕捉一个图像，填入一个示例客户，再捕捉一个图像，调用保存方法并对确认保存的信息框捕捉图像
- 支持应用代码—应用程序有关于挑剔上述的简单捕捉示例。比如，框架里的表单经常与应用对象通讯。如果你没有运行应用程序，应用对象并不存在。