2005 年第 7 期

在一个 **Visual FoxPro** 应用程序中的 **HTML** 编辑器会是一个困难的挑战。在这个 **West Wind** 帮助生成器应用程序技术案例研究的第二部分中，**Rick Strahl** 展示了你可以怎样通过实现接口和处理事件来解决在 **Web** 浏览器控件中进行编辑的问题。

在 VFP 应用程序里发送 SMTP 消息，第一部分 page.34

原著：**Anatoliy Mogylevets** 翻译：**CY**

在他的网站“在 **VFP** 里使用 **Win32** 函数”（**www.news2news.com**），那里有许多示例代码，涉及了 **Windows** 操作系统编程的多个方面。发送和接收邮件的主题是一直排列在网站访问者列表的顶部。在这篇文章里，**Anatoliy Mogylevets** 将为你展示他的学识，并提供了你可以应用在你的应用程序里的代码。

一个定性的结果

作者：Andy Kramek & Marcia Akins

译者：fbilo

在设计应用程序的时候，我们曾碰到过的一个更难解的问题，是怎样处理那些没有固定数据结构的或者扩展性极大的数据。传统的关系型结构太简单，不能处理这样的情况，因此我们需要换用一个更灵活的设计。在本月的专栏中，Andy Kramek 和 Macia Akins 将试图为人的资格/证书建模。

玛西亚：你知道，我一直认为自己在数据建模上还是很有一套的，但这个案例确实难倒我了。我正在试图为一个医院人事系统建立数据模型。除了一般的内容外，它还需要跟踪那些医务人员被指派的工作、以及他们工作的场所。这个部分有点困难，但还不是最难的部分。总之，我还需要跟踪关于医务人员的资格信息。

安迪：我说不清是否明白了这里的问题所在。它不就是一个多对多关系吗？我的意思是，一个人可以有多种资格，而同一个资格可以被许多人所拥有。你所需要的，是一个资格的参照表、以及一个用来“将每个人与他所拥有的资格关联起来”的分配表。

玛西亚：我也希望它会有那么简单。首先，人们所拥有的至少有三种不同类型、并且互补相关的资格：

- 教育——这些用于证明你的学术成就，同时还包括象你的大学学位、以及专业领域之类的东西。它们是静态的，因为一旦你获得了一个教育性的资格，那么你就一辈子拥有这个资格了。
- 专业——这些用于证明你的作证能力，同时还包括象医师资格证书以及专业协会的会员之类的东西。它们倾向于静态，但也可能变化，尤其是在有各种不同等级的成就或者会员的时候。
- 认证——这些说的是你被允许能做的事情，同时还包括象实践许可以及特殊技能之类的东西。它们跟专业资格的区别是：它们会过期，并要求被周期性的更新。

我们可能会有与一个人的“资格”有关的三套全然不同的信息。

安迪：好吧，不过我肯定还是没发现这里的关键，因为所有这一切都告诉我的是，有各种不同的资格类型。因此，何不给资格表增加一个“资格类别”父表呢？至于其它的细节内容，请记住分配表可以包含的东西可不止是一些键而已！所以我们可以直接把象等级、取得日期、过期日期之类的信息直接储存在分配表里（见图 1）。

玛西亚：是的，你的确还没找到这里的关键所在。首先，三种不同类型的资格中的每一种

都有截然不同的一套与之相关的属性。例如，专业协会的会员资格很少会对“专业领域”有所要求，而一个大学学位则肯定会有。此外，对于可更新的认证，我们需要跟踪其更新日期、许可证号码、以及其它可能随着时间的变化而改动的属性。

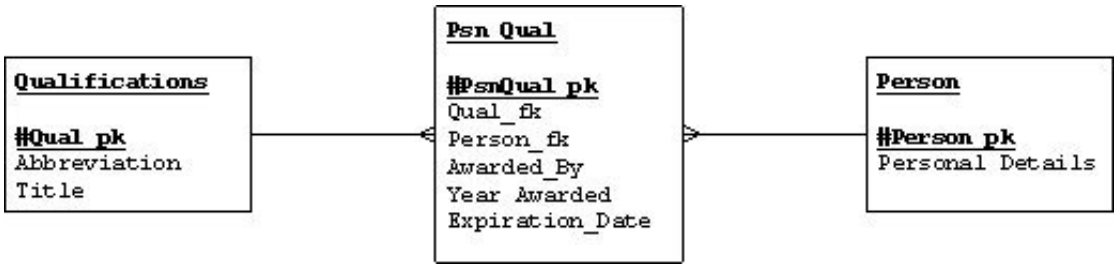

图 1、标准的关系模型

安迪：我明白了，如果我们仅用一个分配表来为之建模的话，的确会出问题。你将不得不包含三套字段（以适应不同的需求）并且在任何一条记录上只使用其中的一套字段。象图 2 中那样，使用扩展表来为每一种资格类型存储与之相关的信息怎么样？虽然这么做会让查询变得麻烦点（你需要用 Union 来连接表），但这样做可以保证对任何一种资格只有一个表中有属于它（指该种资格）的值。

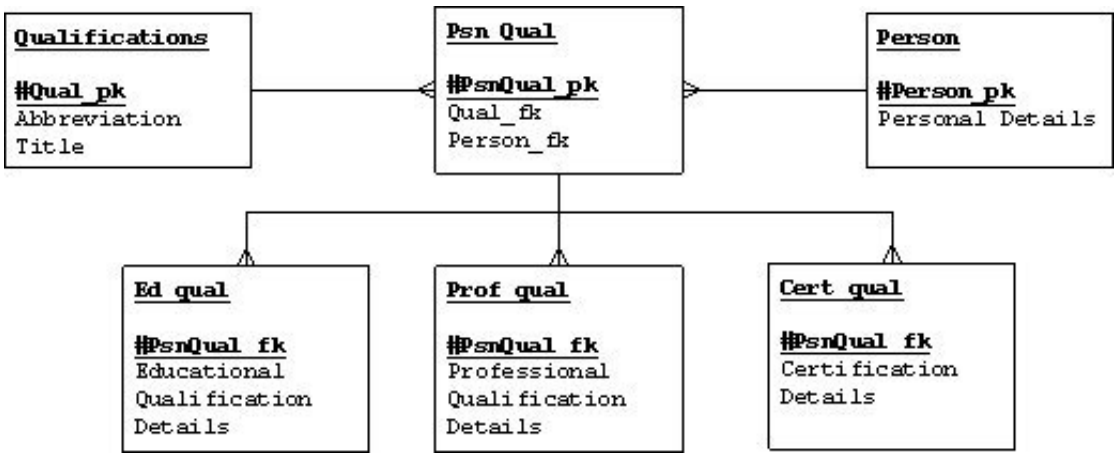

图 2、带扩展表的建模明细图

玛西亚：哦，这么做没多大的帮助！为了使用 UNION，我们将必须建立一个包含所有可能的字段的结果集，并强制把包含每一个扩展表数据的查询结果放入到这个结果集的结构中去。我们也许还可以就用一个文本文件。不过，这里还有第二个你没有考虑到的问题。

安迪：是什么？

玛西亚：我们并没有一个预先定义好的资格列表！所以，没有办法可以把它们放到一个参照表中去以开始工作。

安迪：不可能。肯定在某个地方会有一个受认可的资格列表。

玛西亚：没有！你会从什么地方找到它呢？每一种授予资格的机构都定义了它自己的一套

术语，而国家正式授予的资格范围非常的广泛（我甚至都不想考虑国际的了!）。更糟糕的是，即使是缩写词的含义也各自大不相同。

安迪：这确实是个问题。毕竟，在美国，**"RN"**表示“注册护士”，但是在联合国，相应的缩写则是“**SRN**”，“**RN**”被专门用于表示“**Royal Navy**（皇家海军）”——一种完全不同的解释。

玛西亚：那么，怎么解决？

安迪：我怎么知道？我看起来找不到什么可以用任何常用的方式处理这种情况的好办法。你在这里处理的是“可无限扩展的数据”，那通常意味着需要一个纵向表（**vertical table**）了。

玛西亚：“纵向表”是什么意思？

安迪：纵向表不是水平的通过字段来存储数据。相反，数据是被存储在行中（顾名思义），每一行还包含一个说明存储的是什么东西的描述（就是“属性”）以及一个实际的值。最基本的一个纵向表就包含这么两个字段，所以这种类型的表还有个“草号”，就叫做“属性**/**值 对”。

玛西亚：我喜欢它。通过使用一个纵向表，我们排除了预定义资格表结构的需求。我可以为一个人可能拥有的的资格存储其所需要的任何信息了。我想我们只要增加一个字段来存储对个人表（**person**）的一个外键就行了？

安迪：没错。

玛西亚：等一下。如果我得处理不同类型的数据的话怎么办？是不是所有属性的值都必须存储为字符型的值了？

安迪：没错。最简单的办法，是你给属性表增加一个属性数据类型字段。如果你需要存储更多的信息，那么再增加需要的字段（例如，为了处理显示的格式、或者输入的掩码）。

玛西亚：**That's cool.**不管怎么说，这么做的话，大多数数据将是字符型的，那么我可以建立象下面表 **1** 这样的数据了。

表 **1**、纵向表的示例数据

个人表外键	属性	值
103	**Degree**	**Bachelor of Science BS**
103	**Specialization**	**Computer Science**
103	**Year Awarded**	**1987**
103	**Awarding Body**	**University of Akron**
103	**Honors**	**Magna cum laude**

安迪：目前为止还不错。但如果某人有不止一个资格的话该怎么办？对于医务人员来说，这几乎是肯定会出现的情况了。

玛西亚：啊哈！我们还没有办法可以指定一组属性是属于一个人以外的别的东西。因此，如果某人有两个学位，就无法区分任何一个学位是由哪一个单位授予的了。

安迪：正确。这里的问题是：你的属性也有自己的属性！我们需要在这个模型的基础上再进一步，并使用一个“头部/细节”表对（a header/detail pair of tables）来完整的描述这些资格以及它们的属性。

玛西亚：那么是不是我们其实并不需要纵向表了？

安迪：不，我们需要的，但它将被用作细节表。我们只是需要增加一个头部来搜集相关的属性（见图 3）。

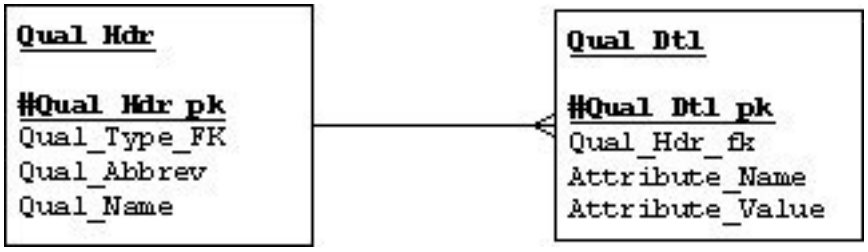

图 3、属性头/明细对

玛西亚：这不是又回到前面的“需要一个参照表来存储可用的资格列表”的情况了吗？

安迪：不，因为我们仍然不知道这些资格是什么东西，所以还是需要手动去输入它们。

玛西亚：不过这样将向我们展现出当用户直接输入数据时将会出现的所有问题！某人的数据里面可能会输入“B.S.”，另一个人的则输入了“BS”，而第三个人的则输入了“B.Sc”。天知道我们怎么才能找到所有拥有一个理学学位的人员名单？

安迪：我不知道，我也不关心这个。你在谈的是一个用户界面的问题，而我在谈的是数据库结构。所有我正在谈的，是不存储一个外键，而是存储实际的文本。怎么在你的用户界面上校验是你的问题。

玛西亚：我想我们可以就用一个“带有一系列“标准的”定义、并允许用户直接输入非标准的东西”的组合框。我们甚至可以使用自动完成——不过，你是对的，这些不是本文的问题了。不过，它确实令我疑惑，我该怎么写访问这种数据的查询呢？这里把字段名称都放在数据里了，我们该怎么办？

安迪：你已经把你的手指点到了纵向表最大的缺点上了。在这样的表上构造参数化视图是很麻烦的，因为它们没有定义好的字段可以用来过滤结果。唯一的办法是搜索特定的“属性=值”组合。

玛西亚：听起来就够麻烦的了——你需要在开始找之前就知道你要找的东西！我想我们是不是可以给头部（Header）表增加专用的字段，而那将允许我们去写查询？

安迪：这么做可行。事实上，你还可以在头部表上有一个 grouping（组） 字段，它将让你可以把各种资格 group （表示 Select 语句中的 Group 关键字）起来。

玛西亚：我不能确定自己是否理解了你的意思。

安迪：这么说吧！有多少种资格足以把一个人称作一个“医生”？

玛西亚：MD(Medical Doctor，药剂师)、DDS(Doctor of Dentistry，牙科医生)、DO(Doctor of Osteopathy，骨科医生)、以及 DC(Doctor of Chiropractic，按摩医生)，根据其名字，但是不多。

安迪：那么，我们所需要的，是在头部表中要有一个类别（category）列用来将上面这些学位（以及任何将来要增加的学位）标识为“医生”。然后，当你需要知道有多少“医生”的时候，就对这个字段进行查询。

玛西亚：我猜如果你真的需要把这个办法做的更精密一些的话，你可以把这个字段做成一个位图（bit map，我想，作者指的应该是一个由 0、1 组成的字符）。那将让你可以在单个字段中将一个资格分配给多种类别。尽管我并不认为我需要做到这种程度。就一套标准的类别就能满足我的需求了。不过，我确实还有另一个问题。

安迪：现在这里有一个惊喜啦！是什么？

玛西亚：我怎么知道要建立哪一些属性？

安迪：哦，这真是个好问题。我们已经知道了我们有三组要建模的资格（教育的、专业的、以及认证了的）。此外，某些属性（像“颁发单位”）可以用在所有组中，而另一些（象许可证号码）则不是。所以，我们在这里所碰到的是在属性们以及它们的组之间的一种多对多关系。

玛西亚：后面的内容我知道了。那意味着我们需要一个分配表了。

安迪：正确！我们需要三个表：一个用于描述独立的属性（“attribute”），一个用于一个属性可以从属于的各种组（“attrib_grp”），而分配表（“att_grp_alloc”）则用于将特定的属性们分配给特定的组。这个分配表也是我们存储“将属性放入其内容中”的信息的地方。

玛西亚：这话什么意思？

安迪：一个属性对某一种资格类型来说可能是“必需”的，而对另一种资格类型则可能是“可选”的，难道不是很有可能出现这种情况吗？另一个例子是顺序号码，你也许

希望“授予单位:”属性出现为一个学位后面的第一个属性，但当一个 CPR 认证时则出现为最后一个属性。

玛西亚：哦，这也会是指出另一个问题的好地方。某些属性有一个预定义值的范围，而且我喜欢为它们使用列表。那么我可以在某个属性需要一个参照表的时候给这个分配表添加一个标志。我总是把我的参照数据保存在一对名为“Lookup_Hdr”（定义类别）和“Lookup_Dtl”（允许的值）的表中。现在，只要我把那些需要参照表的属性命名的跟它们的参照类别一样，就可以简单的完成生成列表的任务。

安迪：我看不出这么做有什么区别。

玛西亚：当然有啦！如果为一个属性设置了一个标志，那么我的用户界面可能会聪明的自己去选用我的自定义 combo 类，该类已经被设置为可以从我的标准参照表中取得值去构造它自己的列表，并使用这个属性名称来生成它自己。

安迪：听起来象是另一篇文章的内容了！同时，图 4 展示了属性、组以及分配表的结构。

图 4、属性管理表

玛西亚：对我来说，它看起来非常的直观。这里面只有一件我不是很清楚的东西。我们怎样把一系列的属性关联给资格呢？

安迪：哦，我真蠢！资格明细（qualification detail）表现在需要一个对属性分配（attribute allocation）表的外键，而不是直接存储属性的名称。

玛西亚：稍等一下。这难道不会让我的查询变得更加的复杂吗？我觉得这么做光为了取得那些属性的名称就得增加三个额外的 Join。

安迪：是这样的。一开始我假设没有实际的理由不直接储存属性的名称，现在我来考虑这个问题。

玛西亚：我在这里看到的唯一问题是，如果我在属性表中改动了一个属性的名称，并且我没有键，那么就会出现问题。

安迪：是的。但到底会造成多大的问题？如果确实很重要，那么可以使用外键。如果不是，那么直接使用名称吧！在这个案例里没有什么“绝对正确”的答案；它主要是一个你更喜欢哪一种方式的问题。或者把它完全的正规化并付出需要的额外的 Join 的

代价，或者不去正规化它而是付出当属性名称改动时的代价。

玛西亚：要选择的话，我还是选使用键的那种。好处是拥有统一的名称，而且不需要担心因为同步改动而增加额外的 JOIN 从而造成的过于繁重的复杂性。那么，最终的模型就像图 5 这样。

安迪：这个模型真真的好处在于：它适用于拥有属性的任何东西。例如，你也可以以这种途径为地址建模——它可以很清晰的解决你应该提供多少“地址线（address line）”字段的问题；你可以根据需要想建多少就建多少。不过那就是完全另外一个故事了。

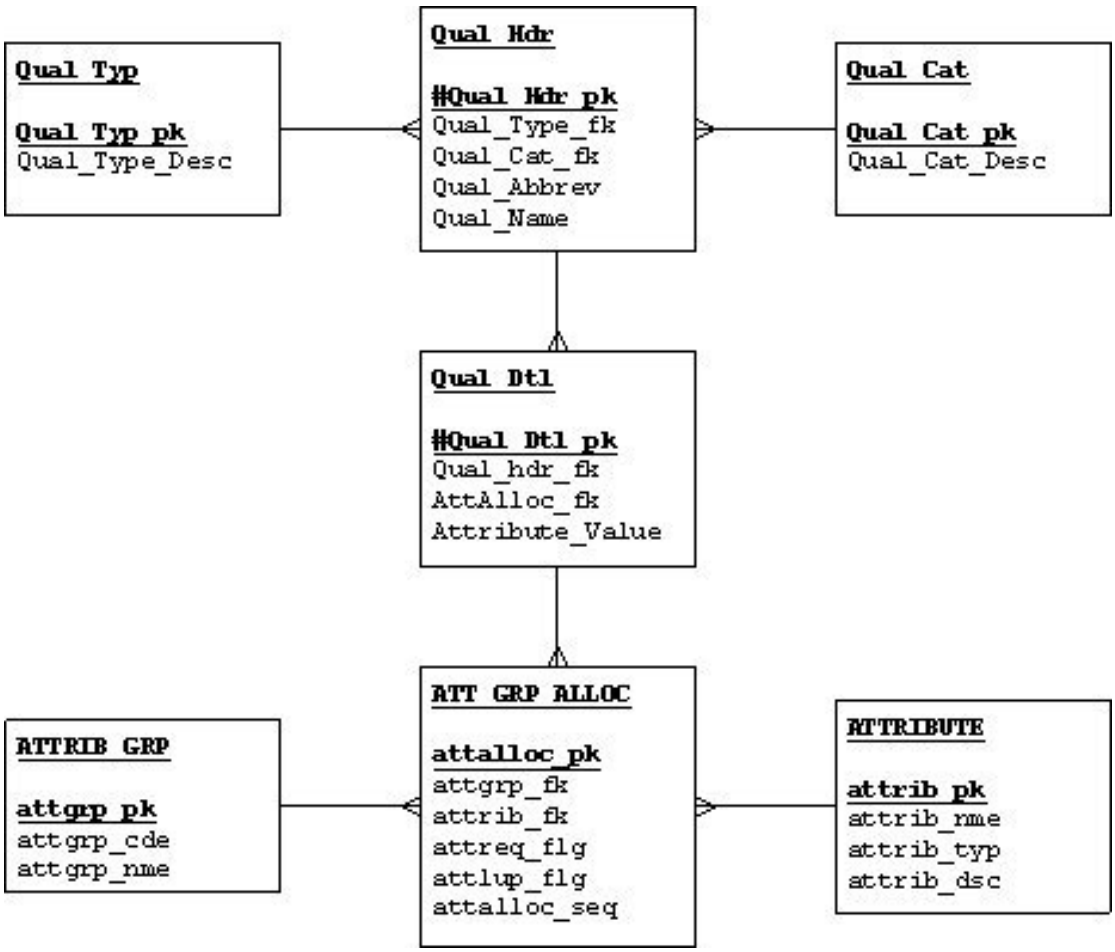

图 5、资格模型

扩展 VFP9.0

原著: David Stevenson

翻译: CY

最近在 VFP 网站上发布的 VFP 路线图，给出了些微软对 VFP 未来的计划，而更多的细节将会在六月的 VFP 会议上透露出来。

从现在到 2007 年上半年，微软 VFP 团队将会致力于为 VFP9.0 加入更多的新特性，在随后两年里将会以正常的测试版本在线公开预览。这个增强项目，代号为 Sedna，将会扩展 9.0 以使得能够与微软的操作系统平台更友好(Windows Longhorn)，以及后续的 VS2005 和 SQL2005。

如果你没有听说 Sedna，你就应该看看 VFP 的主页 http://msdn.microsoft.com/vfoxpro。看看路线图和Ken Levy的六、七月的信，同样在该网址，还包含更多的细节和VFP开发者兴趣的连接。

可以明白的是，2007 年 Sedna 版本的全部主题都是扩展 VFP9.0，以使得它在今后更多年会运行的更好。比如，在 Longhorn 里新的操作系统 API 数量将会是数以千计，而且 VFP 团队将会增加新的功能到 VFP 里，以直接在语言里展示部分功能。

然而，他们计划通过 DLLs 或 FLLs 以取代修改 VFP9 核心运行库来增加那些新的语言扩展，以尽可能的保持其稳定。其他新的特性将通过“Xbase”附件来加入，跟随于 VFP 里许多工具的模式，都是以 VFP 来编写的。这种扩展 VFP9.0 的方法对我来说似乎是非常可靠的，而且也似乎是能够有效的运用 VFP 团队的有限资源。

扩展 VFP9.0 到过去

虽然 VFP9.0 并不被 NT4 所正式支持，许多开发的 VFP 应用仍然运行在那个操作系统上。为了支持这些，并能在这些老版本的操作系统上使用新的特性，在欧洲的 VFP 社区里有部分企业成员已经张贴了修补指导和文件，以使得使用 VFP9.0 的人们可以修改其运行时库文件以实现对 NT4 的兼容。

德国FoxPro用户组的Rainer Becker（也是德国年度开发者大会的组织者），发布了关于如何利用这些资源以扩展VFP9.0 到过去平台的信息。这个补丁可以在目录

http://portal.dfpug.de/dFPUG/Dokumente/Freeware/的dFPUG文档入口找到。

需要注意的是，你要认真考虑微软的最终用户许可协议(EULA)以决定是否在你的环境里使用这个补丁。

这里是 Rainer 所讲述的如何修补和分发规则：

- vfp90nt4.dll 复制到 Windows 系统目录
- vfp9r.dll 和 vfp9t.dll 在 VFP 运行时文件夹“Shared Files” 需要修补
- vfp9.exe 在 VFP 程序目录需要修补
- vfp9.exe, vfp9r.dll, and vfp9t.dll 在同一个目录需要修补
- 备份所有被修补的文件（扩展名为 001,002,等等）

修补过的 VFP9 文件可以运行在任何操作系统上，只要它可以在 Windows 系统目录或当前目录下找到 vfp90nt4.dll 文件。这些补丁程序可以传送给用户，但你不可以把它作为你自己的下载。请链接到上面提及的目录。修补运行时可能会与病毒过滤和/或许可协议产生问题。对补丁程序和结果不作任何担保。

扩展 VFP 功能到 SQL Server

加拿大蒙特利尔的 Igor Nikiforov 已经张贴了一个可免费下载的，它好象看起来对 VFP 开发者作 SQLServer 开发有用的。Igor 已经模仿 VFP 函数编写了数个 SQLServer 用户自定义函数，以解决 TSQL 语言的短处。

包含在 Igor 的免费下载文件里的是 GETWORD COUNT(), GETWORDNUM(), AT(), RAT(), CHRTRAN(), OCCURS(), PADL(), PADR(), PADC(), PROPER(),等等。

下载文件在:

www.universalthread.com/wconnect/wc.dll?LevelExtreme~2,2,27115。

更正

那些从六月刊里下载了 Lisa Slater Nicholls 的伴随着 PDF 文章的源代码的人，当运行于在路径里包含有空格时会有小问题。新版本的下载已经发布，它修正了会导致软件出现这样特别况的一行代码。请从 FoxTalk 2.0 网站的源代码区域下载。

创建你自己的属性编辑器

原著：Doug Hennig

翻译：CY

VFP9 可以很容易让你为类和表单的自定义属性创建自己的编辑器。本月，Doug Hennig 将介绍一个此类编辑器的框架，可以让你只关注于编辑器本身，而不是在编辑器里所挂钩的必要管道。

自从 VFP 发布以来，VFP 所提供的编辑器可以让你很容易的在属性窗口里来指定属性的值。某些属性，比如 BorderStyle，就有一个可接受的值的组合框来让你从中选择。其它还会有一个省略号可以运行对应的对话框，比如对于 BackColor 和 ForeColor 就有个 Color 对话框。这些年来，VFP 开发者都在游说微软为我们自定义属性支持特定的编辑器。在 VFP9 里微软实现了。

你可以在属性的元数据(Member Data)的脚本属性里为属性指定一个编辑器。元数据是 VFP9 的新事物，它是一个 XML 字符串，它定义了对象成员的多个属性，包括什么情况它要被显示，或是否要出现在属性窗口的收藏夹里（我在 FoxTalk 2.0 2004.06 的专栏里详细讨论了元数据，《元数据和自定义属性编辑器》）。

元数据存储在两个地方：一个是对象的自定义属性_MemberData，它包含有对象的多个成员的元数据，或者是智能感知表的记录里(FoxCode.DBF)，每个记录包含一个成员的元数据。

不再象输入 XML 那样乏味，VFP 包含一个元数据编辑器，可以从表单和类菜单里访问到。要为属性指定一个属性编辑器，从元数据编辑器里选择一个属性，打开“有元数据”设置，然后输入在脚本编辑框里被调用的该属性的编辑器所要执行的代码。

元数据的一个有趣的事是它的可扩展性，你可以加入自定义属性到 XML 里。其中的一个用途就是我们把它用作为我们属性编辑器来存储元数据。元数据编辑器使得它可以很容易为成员加入自定义属性—在用户自定义页（参见图 1），点击“增加”以加入一个新的自定义属性，然后在值编辑框里输入该属性的值。

在我们观看如何创建属性编辑器前，这里有一些事你需要知道：

- 你可以以两种方法来引用某个属性的属性编辑器：如果已经为某个属性指定了编辑器，当省略符出现时你可以点击按钮，或者双击属性窗口里的属性行。

- 属性编辑器难于调试：如果你在代码里设置 SET STEP ON，会在这点上暂停执行。然而，即使你选择了单步(Step)，也会不停止的连续执行。为防止这样，可以使用 SYS(2030, 1)，它将允许在系统组件上作调试，仅在 SET STEP ON 前。
- 如果你想在先前版本的 VFP 里使用类，你可能将无法使用_MemberData。即使你可以，但是_MemberData 也肯定少于 255 个字符，或者你根本无法在先前的版本里打开类。

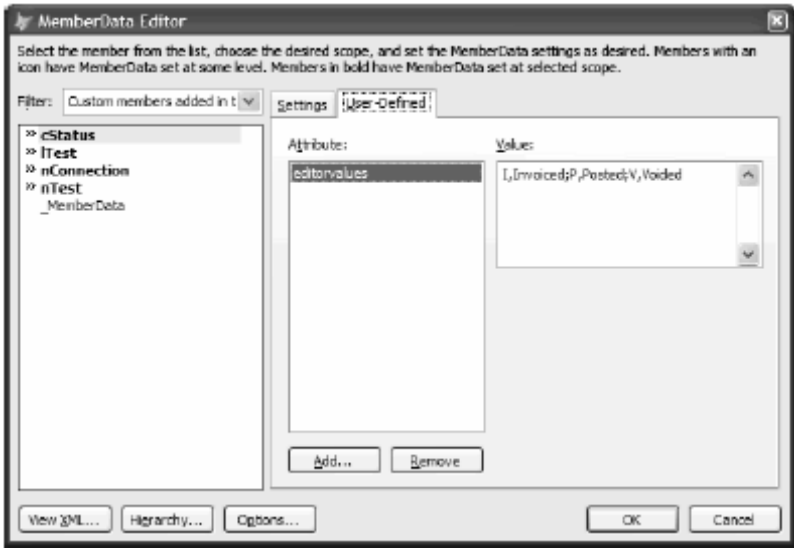

图 1：你可以在元数据编辑器的用户定义页里定义自定义的元数据。

PropertyEditor.PRG

因为编辑属性的许多工作从一个编辑器到下一个是一样的，我创建了一个属性编辑器的框架。它的开始点是 **PropertyEditor.PRG**，在属性的元数据里指定这个程序作为属性编辑器。**PropertyEditor.PRG** 接受编辑器名作为参数，编辑器将要编辑的属性名是可选的。

PropertyEditor.PRG 所做的并不太多，它只是简单的作了例化，并为指定的编辑器名调用相应的类。编辑器类是注册在 **PropertyEditors.DBF** 里，它与 **PRG** 在相同目录下的简单表，它有 **ID, NAME, CLASS,** 和 **LIBRARY** 列。

这里是在 **Test** 类 **Test.VCX** 里的部分属性的元数据。**1Test** 属性，它包含一个逻辑值，需要 **Toggle** 编辑器（注意：你可能需要指定 **PropertyEditor.PRG** 的路径，但是为了简化在本例里不作考虑）。

```
<memberdata name="ltest"
type="property" display="lTest"
script="do PropertyEditor with 'Toggle', 'lTest'"/>
```

nTest 包含数值，从 **1** 到 **3**，因此它也指定 **Toggle**，但使用了自定义的最小值和最大

值属性来指定值的范围。

```
<memberdata name="ntest"
type="property" display="nTest"
script="do PropertyEditor with 'Toggle', 'nTest'"
lowvalue="1" highvalue="3"/>
```

cStatus 指定了发票的状态：“**I**”表示已开票，“**P**”表示已过帐，“**V**”表示空白。因为这些是可枚举值，**cStatus** 使用了 **Enumerated** 编辑器，并且它的自定义属性提供了一个按照格式的列表，“值，显示的描述；值，显示的描述；等等”。

```
<memberdata name="cstatus"
type="property" display="cStatus"
script="do PropertyEditor with 'Enumerated',
'cStatus'"
editorvalues="I,Invoiced;P,Posted;V,Voided"/>
```

PropertyEditor.PRG 开始时将根据需要来打开 **PropertyEditors.DBF**，在 **NAME** 字段查找指定的名，如果找到，就例化 **CLASS** 和 **LIBRARY** 字段里所指定的类。注意到它是对 **LIBRARY** 使用 **TEXTMERGE()**，因此你可以以“**HOME() + ‘LibraryName.VCX’**”的方式来指定一个 **VFP** 根目录下的类库。同样，如果不指定目录，**PropertyEditor.PRG** 将在它的相同目录下查找类库。在例化类后，**PropertyEditor.PRG** 将调用编辑器的 **EditProperty** 属性来继续工作。

```
lparameters tuEditorName, ;
    tcProperty
    local lcDirectory, ;
    lcLibrary, ;
    loEditor

* 判断我们运行时的目录

lcDirectory = addbs(justpath(sys(16)))

* 创建指定的编辑器，并处理未指定或未注册的情况

do case
    case not vartype(tuEditorName) $ 'CN' or ;
         empty(tuEditorName)
         messagebox('Must specify editor name.', 48, ;
              'Property Editor')
         return
    case not OpenEditorsTable(lcDirectory)
         return
    case (vartype(tuEditorName) = 'C' and ;
         seek(upper(tuEditorName), 'PropertyEditors', ;
```

```
            'Name')) or (vartype(tuEditorName) = 'N' and ;
            seek(tuEditorName, 'PropertyEditors', 'ID'))
            lcLibrary = textmerge(PropertyEditors.Library)

            if not file(lcLibrary)
                lcLibrary = forcepath(lcLibrary, lcDirectory)
            endif not file(lcLibrary)

            if file(lcLibrary)
                loEditor = newobject(PropertyEditors.Class, ;
                    lcLibrary)
            else
                messagebox('The library specified for ' + ;
                    transform(tuEditorName) + ;
                    ' cannot be located.', 48, 'Property Editor')
                return
            endif file(lcLibrary)
        otherwise
            messagebox(transform(tuEditorName) + ;
                ' is not a registered editor.', 48, ;
                'Property Editor')
            return
    endcase
    use in PropertyEditors

    * 如果属性名是指定的（比如，一个编辑器可以用于多个属性），
    * 将编辑器的 cProperty 属性设为它。
    if vartype(tcProperty) = 'C' and not empty(tcProperty)
        loEditor.cProperty = tcProperty
    endif vartype(tcProperty) = 'C' ...

    * 让编辑器作它的事情
    loEditor.EditProperty()
```

SFPropertyEditor

SFPropertyEditor，定义于同名的 **VCX** 文件，是属性编辑器的基类。**SFPropertyEditor** 并不被直接使用，对它子类化以创建所需的编辑器。因而 **PropertyEditor.PRG** 调用编辑器的 **EditProperty** 属性，并让它开始运行。

第一个任务是获得在表单或类设计器里所选择的一个对象或多个对象的指针。**GetObjectReference** 方法填充并返回一个指针的集合，因此 **EditProperty** 把集合放入 **oObjects** 属性。然后，**EditProperty** 调用 **ValidateProperty** 以确认属性名已经在

cCproperty 属性里指定，并且指定的属性是所选择对象的成员（**PropertyEditor.PRG** 把接收到的第二个参数放入 **cProperty** 属性，对于单个编辑器，在属性窗口里为编辑器的子类填入一个值）。

GetCurrentValue 返回属性的当前值（因此编辑器可以显示，如果需要），第一个选择的对象放入到 **uCurrentValue** 属性，并且 **cDataType** 包含了该值的数据类型。然后 **EditProperty** 调用 **GetAttributes** 以读取元数据属性名和值，并放入 **oAttributes** 集合里，因此编辑器可以在需要时使用它。如果你的编辑器需要测试这些设置是否完成（比如，**cDataType** 包含编辑器所期望的数据类型），把代码放入 **TestCustomAsserts** 方法里，它会被 **EditProperty** 所调用，**TestCustomAsserts** 在这个类里是抽象的。最后，**PropertyEditor** 方法，它在这个类里也是抽象的，它被调用以实现真正的编辑任务。

```
local loException as Exception
with This
try

* 获取选择对象的指针

.oObjects = .GetObjectReference()

* 确认我们有一个有效属性名，并取得其当前值和在_HemberData 里任何属性

.ValidateProperty()
.uCurrentValue = .GetCurrentValue()
.cDataType = vartype(.uCurrentValue)
.oAttributes = .GetAttributes()

* 测试其他申明，然后调用“真正”的编辑器

.TestCustomAsserts()
.PropertyEditor()

* 处理失败的申明

catch to loException ;
when not empty(loException.UserValue)
.Warning(loException.UserValue)

* 处理其他类型错误

catch to loException
.Warning(loException.Message)
```

```
endtry
endwith
```

我们并没有查看在 **SFPropertyEditor** 里的其他方法，尽管那些也包含有重要或有趣的代码。

GetObjectReference 返回一个所选择对象的集合。它利用 **ASELOBJ()**把这些对象填充到数组里，并复制指针到它所要返回的集合里。注意到对 **Assert** 方法的调用，这个方法简单的测试了指定的条件，并且如果它出错了，就抛出错误消息。**Assert** 方法是类似于 **ASSERT** 命令，但是它可以让我们来合并所有的申明错误到同一个地方：在 **EditProperty** 里的第一个 **CATCH** 语句。

```
local laObjects[1], ;
    loObjects, ;
    lnI

This.Assert(aselobj(laObjects) > 0 or ;
    aselobj(laObjects, 1) > 0, ;
    'No object is selected.')
loObjects = createobject('Collection')
for lnI = 1 to alen(laObjects)
    loObjects.Add(laObjects[lnI])
next lnI
return loObjects
```

GetAttributes 以编辑器正在编辑的属性所定义的所有元数据属性的名和值来填充为集合。它使用一个 **XML DOMDocument** 对象完成少量的查找和拆分的工作。注意到如果对象没有**_MemberData** 属性，或者它为空，明显的它必须要有“全局”元数据，它存储在智能感知表内，或者其他属性编辑器还没有被运行。因此，在那样的情况下，元数据是从智能感知表的相应记录的 **TIP** 备注字段里读取出来的。

```
local loAttributes, ;
    lnSelect, ;
    loDOM as MSXML2.DOMDocument, ;
    loObject, ;
    loNode as MSXML2.IXMLDOMElement, ;
    loAttribute as MSXML2.IXMLDOMAttribute, ;
    lcName, ;
    lcValue

loAttributes = createobject('Collection')

try

* 创建一个 XHL DOH 对象，并装载所选择的第一个对象的_MemberData 忏悔里的
```

```
XHL。
    * 如果那个属性不存在或为空，查看是否在智能感知表里有任何全局的Hember Data。

    loDOM = createobject('MSXML2.DOMDocument.4.0')
    loDOM.async = .F.
    loObject = This.oObjects.Item(1)
    if pemstatus(loObject, '_MemberData', 5) and ;
        not empty(loObject._MemberData)
        loDOM.loadXML(loObject._MemberData)
    else
        select 0
        use (_foxcode) again shared alias __FOXCODE
        locate for TYPE = 'E' and ;
            upper(ABBREV) = upper(This.cProperty)
        if found()
            loDOM.loadXML(TIP)
        endif found()
        use
    endif pemstatus(loObject, '_MemberData', 5) ...

    * 查找我们所编辑在属性节点。如果有，装载所有的属性到集合里。

    loNode = loDOM.selectSingleNode('//memberdata' + ;
        '[@name="' + lower(This.cProperty) + '"]')
    if vartype(loNode) = 'O'
        for each loAttribute in loNode.attributes
            lcName = loAttribute.name
            lcValue = loAttribute.value
            loAttributes.Add(lcValue, lcName)
        next loAttribute
    endif vartype(loNode) = 'O'

    catch
        use in select('__FOXCODE')
    endtry
    select (lnSelect)
    return loAttributes
```

UpdateProperty 并不是从 **EditProperty** 里被调用的，但是可以从你的子类的 **EditProperty** 里作调用，为每个所选择的对象写一个新值到属性里。

```
    lparameters tuValue
        local loObject, ;
        lcProperty
    for each loObject in This.oObjects
```

```
        lcProperty = 'loObject.' + This.cProperty
        store tuValue to (lcProperty)
    next loObject
```

SFPropertyEditorToggle

让我们来看看一个实际的例子。当 **VFP9** 还是 **Beta** 版本时，我看到 **VFP** 的领袖 **Rick Schummer** 在属性窗口里双击一个属性可以把它从**.F.**改变为**.T.**。除了作双击，没做什么处理，我所实现的是对自定义属性而不是他所双击的内部属性。我问他是否已经有些新的特性可以解决我的关注，而他给我一个狡猾的笑并说，“不，它是自定义属性编辑器”。

在我对它作了些思考后，我实现这样一个编辑器，它对数值属性是很有用的，因此他们只需要双击一个带有预定义值范围的内部属性，比如 **BorderStyle**。于是我创建了**SFPropertyEditorToggle**。

PropertyEditor 方法检查并确认是否有自定义的最小值和最大值元数据属性存在，是否两者都有，并且都是有效值（如果需要你可以自由改变这个范围）。然后通过以相应的值来调用 **UpdateProperty** 以轮换属性：对于逻辑属性则对当前值取非；对于数值属性，如果未达到最大值，则取当前值加 **1**，否则就取为最小值。

```
    local lnValue, ;
        lnLowValue, ;
        lcLowValueType, ;
        lnHighValue, ;
        lcHighValueType

    with This

    * 确认最小值和最大值是否都存在，并且是有效值。

    if .oAttributes.GetKey('lowvalue') > 0
        lnValue = .oAttributes.Item('lowvalue')
        lnLowValue = int(val(lnValue))
    endif .oAttributes.GetKey('lowvalue') > 0

    lcLowValueType = vartype(lnLowValue)
    if .oAttributes.GetKey('highvalue') > 0
        lnValue = .oAttributes.Item('highvalue')
        lnHighValue = int(val(lnValue))
    endif .oAttributes.GetKey('highvalue') > 0

    lcHighValueType = vartype(lnHighValue)
    .Assert(lcLowValueType = lcHighValueType, ;
```

```
        '_MemberData must specify both lowvalue and ' + ;
        'highvalue attributes.')
    .Assert(lcLowValueType = 'L' or ;
        (lcLowValueType = 'N' and ;
        between(lnLowValue, 0, 10)), ;
        'The _MemberData lowvalue attribute must be ' + ;
        '0 - 10.')
    .Assert(lcHighValueType = 'L' or ;
        (lcHighValueType = 'N' and ;
        between(lnHighValue, 0, 10)), ;
        'The _MemberData highvalue attribute must ' + ;
        'be 0 - 10.')

    * 轮换属性值。

    do case
        case .cDataType = 'L'
            .UpdateProperty(not .uCurrentValue)
        case lcHighValueType = 'N' and ;
            .uCurrentValue >= lnHighValue
            .UpdateProperty(lnLowValue)
        otherwise
            .UpdateProperty(.uCurrentValue + 1)
    endcase
    endwith
```

若想看看这个编辑器的动作，打开 **Test.VCX** 里的 **Test** 类，并双击 **1Test** 和 **nTest** 属性。正如我们前面所见到的，这两个属性的元数据都指定 **Toggle** 作编辑器的名并传递给 **PropertyEditor.PRG**，并且 **Toggle** 是注册在 **PropertyEditors.DBF** 里作为 **SFPropertyEditorToggle** 类。同样，**nTest** 也有一个最小值和最大值属性，它为 **nTest** 指定了值的范围为 **1** 到 **3**。双击 **nTest** 多次就可以看到整个范围的值。

SFPropertyEditorEnumerated

某些属性包含“代码”值，比如 **BorderStyle**，在这里 **0** 代表无边框，**1** 代表固定单线边框，**2** 代表固定对话边框，**3** 代表可缩放边框。因为这是预定义的值列表和所代表的意思，这些是枚举属性。你可以有你自己的枚举属性，比如一个发票状态属性，它包含“**I**”代表已开票，“**P**”代表已过帐，“**V**”代表空白。

为不再强迫开发者记住这些可取值，可利用 **SFPropertyEditorEnumerated** 作为属性的编辑器。正如你在图 **2** 所见，它显示了一系列的值及其描述，类似于 **VFP** 组合框为内部枚举属性在属性窗口里所显示的。

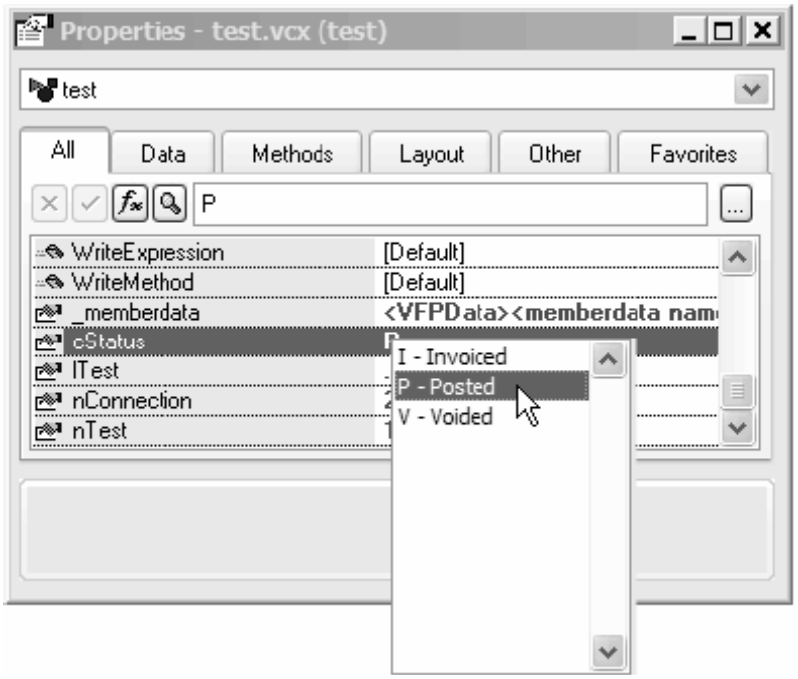

图 2：SFPropertyEditorEnumerated 显示了一个为枚举属性的可取值及其描述的列表。

SFPropertyEditorEnumerated 需要一列用于显示的值，因此 **PropertyEditor** 方法从元数据里获取自定义 的 **editorvalues** 属性的值，并把它拆分为分离的 **valuedescription** 集。然后 **PropertyEditor** 例化了 **SFEnumeratedValueForm**，一个简单的由列表框和不多的代码组成的表单，然后调用它的 **AddValue** 方法来为列表框加入每个值和描述，并且调用调用它的 **SelectValue** 方法以确认属性的当前值是默认选择的。

机巧的代码包含着决定表单的下一步动作。通常，我喜欢它被直接放在鼠标定位 **(MROW()** 和 **MCOL()**将告诉我们这些)，但是如果属性窗口被固定了，它将无法工作，因为表单不能被放在属性窗口的顶上。

因此，代码使用了部分 **Windows API** 功能来描绘属性窗口的定位位置（我们将不在这里讨论这那些方法代码），判断是否把表单放到属性窗口的左边或右边，并相应的设置表单的 **Top** 和 **Left** 属性。最后，它调用表单的 **Show** 方法，并从中返回，检索用户从表单里选择的值并根据所要求的数据类型来更新属性。

```
local lcValues, ;
    lcDirectory, ;
    loForm, ;
    laLines[1], ;
    lnLines, ;
    laValues[1], ;
    lnI, ;
    lcValue, ;
    lcDescription, ;
    lnTop, ;
```

```
        llDockable, ;
        lnhWnd, ;
        lcBuffer, ;
        lnWLeft, ;
        lnWRight, ;
        lnLeft

    with This

    * 获取 editorvalues 属性

    .Assert(.oAttributes.GetKey('editorvalues') > 0, ;
        'There is no editorvalues attribute in ' + ;
        '_MemberData.')
    lcValues = .oAttributes.Item('editorvalues')

    * 创建一个表单以显示值，并拆分为单个的值

    lcDirectory = sys(16)
    lcDirectory = addbs(justpath(substr(lcDirectory, ;
        at(' ', lcDirectory, 2) + 1)))
    loForm = newobject('SFEnumeratedValueForm', ;
        lcDirectory + 'SFPropertyEditor.vcx')
    lnLines = alines(laLines, lcValues, 1, ';')
    .Assert(lnLines > 0, 'The editorvalues ' + ;
        'attribute in _MemberData does not contain a ' + ;
        ' valid set of values.')

    dimension laValues[lnLines]
    for lnI = 1 to lnLines
        lcValue = laLines[lnI]
        lcDescription = strextract(lcValue, ',')
        lcValue = strextract(lcValue, '', ',')
        laValues[lnI] = lcValue
        loForm.AddValue(lcValue, lcValue + ' - ' + ;
            lcDescription)
    next lnI

    * 选择当前值
    loForm.SelectValue(.uCurrentValue)

    * 描绘出表单要放置的位置。如果属性窗口被固定（无论当前是否被固定），我们将
不得不把表单旋转在它旁边，
    * 因为我们不能放到顶上。在这样的情况下，我们将不得不利用部分 Windows API
```

函数来定位属性窗口，并判断它的位置。

* 如果我们有足够的空间来旋转表单到属性窗口的右边，就继续照做；否则就把它放到左边。

```
lnTop = min(mrow('', 3), _screen.Height - ;
    loForm.Height)
llDockable = wdockable('Properties')

if llDockable
    lnhWnd = .FindWindow(0, 'Properties')
    if lnhWnd <> 0
        lcBuffer = replicate(chr(0), 16)
        if GetWindowRect(lnhWnd, @lcBuffer) <> 0
            lnWLeft = ctobin(left(lcBuffer, 4), 'rs')
            lnWRight = ctobin(substr(lcBuffer, 9, 4), 'rs')
        endif GetWindowRect(lnhWnd, @lcBuffer) <> 0

        if _screen.Width - loForm.Width > lnWRight
            lnLeft = lnWRight
        else
            lnLeft = lnWLeft - loForm.Width - 5
        endif _screen.Width - loForm.Width > lnWRight
    else
        lnLeft = 0
    endif lnHWnd <> 0
else
    lnLeft = min(mcol('', 3), _screen.Width - ;
        loForm.Width)
endif llDockable

loForm.Top = lnTop
loForm.Left = lnLeft

* 显示表单并返回选择值。

loForm.Show()
lcValue = laValues[loForm.nSelectedValue]
do case
    case vartype(loForm) <> 'O' or ;
        loForm.nSelectedValue = 0
    case .cDataType = 'C'
        .UpdateProperty(lcValue)
    otherwise
        .UpdateProperty(int(val(lcValue)))
```

```
    endcase
endwith
```

为查看编辑器的动作，打开 Test.VCX 里的 Test 类，并双击 nConnection 或 cStatus 属性。正如我们先前所见到的，这两个属性的元数据都指定了 Enumerated 作为编辑器名并传递给 PropertyEditor.PRG，并且 Enumerated 是注册在 PropertyEditors.DBF 里作为 SFPropertyEditorEnumerated 类。两个属性都有一个编辑值属性，它指定了值和描述对。

其他属性编辑器的思考

另一个普通的属性编辑器，SFPropertyEditorGetFile，是用于包含文件名的属性。创建名为 fileext 和 extdescrip 的自定义属性，它包含有文件的扩展名和扩展名描述（比如，“DBF”和“表”），然后指定“GetFile”作为编辑器名以传递给 PropertyEditor .PRG，并传递属性名作为第二个参数。我们将不讨论这个编辑器的代码，因为它是非常简单的。看看 Test 类的示例里是如何使用 cFileName 属性的元数据。

并不是所有你所创建的属性编辑器都要象 SFPropertyEditorToggle，SFPropertyEditorEnumerated，和 SFPropertyEditorGetFile，你的属性编辑器可以指定给单个属性。在那样的情况下，设置 cProperty 为属性的名，并且不需要传递第二个参数给 PropertyEditor.PRG。同样，一个属性编辑器并不是只能影响一个属性，如果一个属性的值与另一个有影响，可以在同一个时间自由改变两者。

因为在属性窗口里的属性有 8K 字符的限制，而且，当你有许多成员应用于冗长的 XML 时，_MemberData 会快速增长，你可能想通过改名 PropertyEditor.PRG 为 PE.PRG 以在 XML 里保存空格，并把它放入到你的 VFP 目录里，然后通过 ID 而不是名来引用一个指定的编辑器。比如，你可以用 script=“PE(1,‘PropertyName’)” 来使用 SFPropertyEditorToggle，它的 ID 为 1，以作为属性的编辑器。

特别感谢 Rick Schummer 对此的测试，和提供有价值的建议，并且，当然，还有原始的灵感火花。

总结

事实是你可以利用 VFP 代码来自定义和扩展 VFP IDE，以尽可能的提高你的无限产生力。我在本文中所展示的属性编辑器框架可以帮助你建立编辑器，以减少创建大型 VFP 应用时所花费的时间。

下载文件：507HENNIG.ZIP

案例研究：West Wind HTML Help Builder 的开发

第二部分

作者：Rick Strahl

译者：fbilo

在一个 Visual FoxPro 应用程序中的 HTML 编辑器会是一个困难的挑战。在这个 West Wind 帮助生成器应用程序技术案例研究的第二部分中，Rick Strahl 展示了你可以怎样通过实现接口和处理事件来解决在 Web 浏览器控件中进行编辑的问题。

在我的上一篇文章中（**Foxtalk 2.0 2005 年 5 月刊中**），我展示了帮助生成器中用于内容编辑的基础文本编辑界面。文本编辑可能是输入文本的最有效率的途径，但某些人可能更愿意在他们输入的时候就看到输出的结果。为了实现这个目的，帮助生成器通过使用 **Web 浏览器控件，实现了为标题内容的主体支持所见即所得的编辑。图 1 展示了帮助生成器在富 HTML 编辑模式下的样子。**

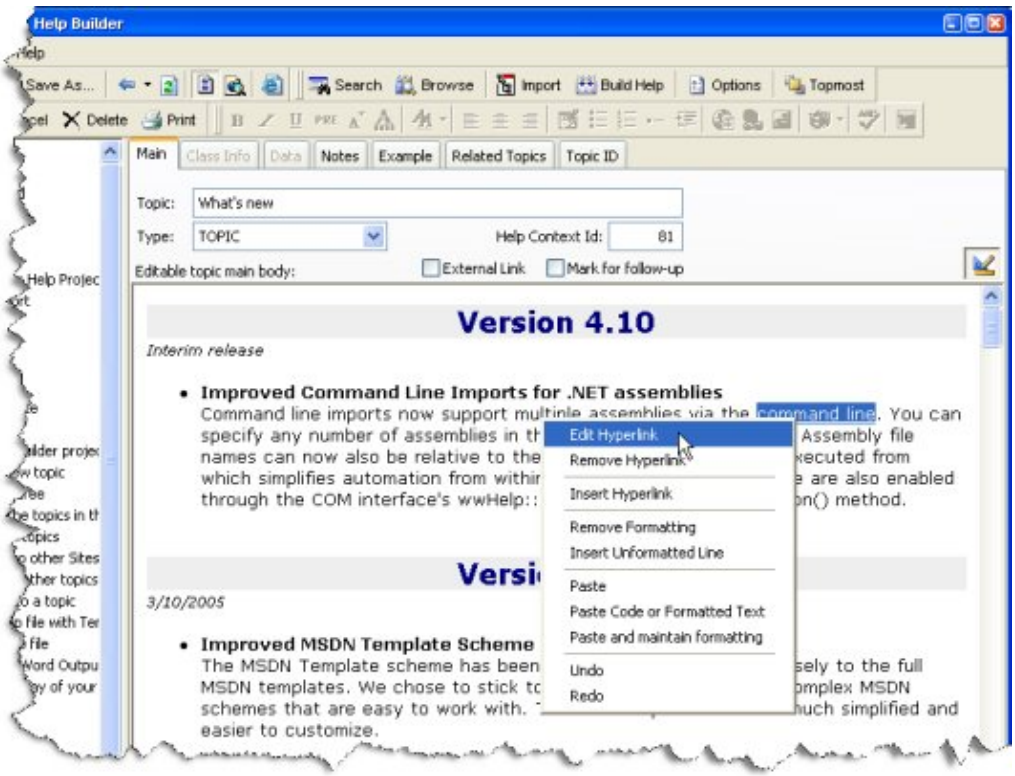

图 1、为所见即所得编辑采用 Web 浏览器控件——可不是个简单的任务

如果你看得更仔细一些，你将发现编辑被限制在当前主题的实际主题内容部分。例如，完整的主题栏就没有显示。在编辑控件里，你可以向 **HTML** 文档中输入文本和任何格式，它们都会立即被应用给这个文档，让你可以马上看到你输入的文本格式化过以后的效果。因此，如果你插入了一幅图片或者执行了一个屏幕捕捉，这幅图片也会立即显示出来。如果你用工具栏上的某个选项标记了某些文本以改变文本的一些属性或者应用一个样式，这个改动也会立即显示出来。

通过 **Web** 浏览器的 **HTML** 编辑在理论上可以很简单的通过设置来完成。你可以简单的设置：

```
thisform.oBrowser.Document.Body.ContentEditable ="true"
```

为了读和写后台 **HTML** 文档的内容，你必须与 **document** 对象进行交互以返回文本：

```
lcHTML = thisform.oBrowser.Document.Body.innerHtml
```

为了写入内容，你可以对选择范围（**Selection Range**）进行操作以读取文本，更新它，然后再写回去：

```
loRange = ;
    thisform.oBrowser.Document.Selection.CreateRange()
lcText = loRange.HtmlText
lcText = [<b>] + lcText + [</b>] && 标记选中部分
loRange.PasteHtml(lcText)
```

对于基本的内容编辑，这么干没问题。不过，如果你想做些象一个真的编辑器那样的活，就需要大量的工作以使得 **Web** 浏览器控件编辑器能够正常的表现。让我们看看其中的部分问题。

象我上一篇文章中提到的那样，帮助生成器为 **Body** 字段的编辑界面使用了可切换的显示“视图”。它里面有一个标准的编辑框（**editbox**）供用于文本输入，还有一个 **HTML** 编辑控件可以被覆盖在 **Edit** 控件的顶上。编辑模式是通过表单上的一个按钮、或者在主题内容中一个文本命令来激活的。

对于大段的文本编辑，帮助生成器使用了一个名为 **wwHTMLEditBox** 的自定义 **EditBox** 类。这个类使用选中文本范围处理从 **Editbox** 中读取和写入内容的工作。它的 **InsertHtml** 方法接收将近 **50** 种不同的参数类型，这些参数类型表示的是可以被用来插入内容到文本文档中的各种途径。它处理象插入图片、执行屏幕捕捉（直接将捕捉到的图片插入到文本中）之类的任务。大多数操作其实都很简单，就象这样：

```
CASE lcAction="BOLD"
    lcText=lcLTag + "b"+lcRTag +lcSelText+lcLTag + ;
        "/b"+lcRTag
```

不过，某些象屏幕捕捉或者代码片断的语法标记这样的东西，会更复杂一些，并且由

于插入的需要要打开一些表单。所有这一切都是为了纯文本工作的。

在 **wwHtmlEditBoxVisual** 类中有另一种专门的用途。这个类通过在 **Editbox** 的顶上覆盖一个 **Web** 浏览器控件来提供 **HTML** 编辑支持。这个类管理 **HTML** 编辑界面，以及将内容从表单控件“推入”到 **HTML** 编辑器、再倒推回去的工作。

因此，导航到一个已经启用了 **HTML** 编辑的新主题会更新 **HTML** 编辑控件，而当 **HTML** 编辑控件失去焦点时则会更新后台的 **EditBox** 控件。这是为了让 **HTML** 控件和 **EditBox** 控件总是保持同步。下面的代码示例演示了实现这个目的的一些关键方法：

```
Function SetViewMode()
LPARAMETER lnMode
LOCAL loHelp, llError, loEdit

loHelp = ThisForm.oHelp

IF lnMode = 2
   *** If the control exists already remove it!
   IF !VARTYPE(this.Parent.oHtmlEdit) == "O"
      TRY
         THIS.parent.AddObject("oHTMLEdit",;
            "wwBrowser_editor")
      CATCH
         llError = .t.
      ENDTRY

      IF llError
         thisform.LockScreen = .f.
         THIS.SetViewMode(1)
         WAIT WINDOW ;
            "不能将控件设置为所见即所得编辑模式..." ;
         nowait
         RETURN
      ENDIF

      loEdit = THIS.parent.oHTMLedit
      loEdit.visible = .T.

      *** 强制在可见后进行一次缩放
      loEdit.Height = THIS.height +1
      loEdit.Height = THIS.height -1
   ENDIF

   *** 现在，从文本框取出 HTML 放到这个控件里
   THIS.UpdateHTMLEditDisplay()
```

```
    THIS.nViewMode = 2
    THIS.visible = .F.

    THISFORM.Lockscreen = .F.
ELSE
    this.nViewMode = 1

    *** 我们正处于文本模式下 - 不需要切换
    IF VARTYPE(this.Parent.oHtmlEdit) != "O"
        RETURN
    ENDIF

    THISFORM.lockscreen = .T.

    THIS.Parent.oHTMLEdit.visible = .F.
    THIS.visible = .T.

    *** 显示文本框，并从设计模式读取数据
    THIS.SaveHTMLToText()

    this.SetFocus()

    THIS.parent.RemoveObject("oHTMLEdit")
    THIS.nViewMode = 1
    THISFORM.LockScreen = .F.
ENDIF

FUNCTION UpdateHtmlDisplay
LOCAL loHelp, loEdit, lcUrl, lcHTML

loHelp = THISFORM.oHelp
loEdit = THIS.PARENT.oHTMLEdit

lcUrl = FULLPATH( ADDBS(JUSTPATH(loHelp.cFileName)) + ;
    "__editor.htm" )

*** 从这个主题中读取模板
*** 并只返回头部
lcHTML = File2Var(loHelp.SetTemplate())
IF (EMPTY(lcHTML) )
    lcHtml = "<html><head></head><body style=" +;
        "'font-family:verdana;font-size:10pt;'>" +;
        "</body></html>"
```

```
ENDIF

*** 取出头部，并添加空白的主体部分
lcHTML = SUBSTR(lcHTML,1,ATC("<body",lcHTML))

*** 写入到磁盘上，以便我们可以打开浏览它
*** - empty document
File2Var(lcUrl,lcHTML)
loEdit.Navigate(lcUrl)

*** 等待文档被加载完毕
lnSeconds = SECONDS()
DO WHILE TYPE("loEdit.Document.Body.innerHtml") # "C" ;
   AND SECONDS() - lnSeconds < 4
   DOEVENTS
ENDDO

lcValue = this.Value
* 需要在这里加一些 HB 分析代码

*** 进入编辑模式
loEdit.Document.Body.ContentEditable = "true"
DO WHILE TYPE("loEdit.Document.Body.innerHtml") # "C" ;
   AND SECONDS() - lnSeconds < 3
   DOEVENTS
ENDDO

loEdit.Document.Body.innerHTML = ;
   loHelp.FormatHTML(lcValue,.f.,.t.) && Don't parse

*** 将文本容器事件绑定到文档主体
*** 允许捕捉大多数常用文档事件
THIS.oTextContainerEvents = .f.
this.oTextcontainerevents = ;
   CREATEOBJECT("IHtmlTextContainerEvents")
this.oTextContainerEvents.oDoc = loEdit.Document
EVENTHANDLER(loEdit.Document.Body, ;
   this.oTextContainerEvents)

FUNCTION SaveHtmlToText
LOCAL lcValue

lcValue = THIS.parent.oHTMLEdit.Document.Body.innerhtml
```

```
lcValue = STRTRAN(lcValue,"&lt;%=","<%=")
lcValue = STRTRAN(lcValue,"%&gt;","%>")

* 在这里做一些额外的 HTML 补充
this.Value = FixBasePath(lcValue, ;
    this.Parent.oHtmlEdit.Document.Body)
```

SetViewMode 在编辑和 **HTML** 模式之间切换时触发。**UpdateHtmlDisplay** 和 **SaveHtmlToText** 在控件之间转换 **HTML** 和文本。然后，这些方法在 **LostFocus** 发生时、 **EditBox** 控件的值被改动时（因为 **EditBox** 是绑定到数据的控件）被触发。

注意，显示在控件中的 **HTML** 是从磁盘上加载来的。我这么做是为了确保 **Web** 浏览器会加载包含 **HTML** 头的完整 **HTML** 文档，**HTML** 头里面包含着对样式表的链接。它是通过为被编辑的主题加载相应的模板来达到这个目的的。所以，显示在浏览器控件中的格式是与定义在模板中的格式相匹配的。如果用户有自定义的 **CSS** 样式表，则会使用这些样式表中的样式，以保证内容看起来就象它在正式的帮助文档中将会出现的样子。

当从控件中取出 **HTML** 主题的时候，有一件重要的事情是修正所有相关的嵌入链接的基本路径。**Internet Explorer** 有一个坏习惯，它会自动把所有它加载到 **DOM** 里的路径都扩展为完整的路径。假设你现在有这么一个链接：

```
<a href='test.htm'>Some Page</a>
```

那么，控件所返回的却会是：

```
<a href='file:///d:\temp\hbproj\test.htm'>Some page</a>
```

很明显，我们可不希望这种东西出现在我们的帮助文档中。为了解决这个问题，我建立了一个 **FixBasePath()** 方法，该方法会弄清楚当前文档的基础路径，然后使用 **STRTRAN** 来替换掉文档中的路径部分：

```
FUNCTION FixBasePath( lcElementPath, loCtl )
LOCAL lnAt, lcPathOnly
*** Strip to last /
lnAt = RAT("/",loCtl.Document.location.toString())
lcPathOnly = SUBSTR(loCtl.Document.location.toString, ;
    1,lnAt )
RETURN STRTRAN(lcElementPath,lcPathOnly,"",1,-1,1)
```

处理 HTML 文档事件

采用 **Web** 浏览器控件的 **HTML** 编辑器真正困难的地方，在于怎样让它表现得象一个真正的编辑器那样。你需要能够选择在文档中的文本，并打开一个匹配该对象的编辑器。该控件默认的行为只提供了很少一些默认的编辑器和处理器，但它们有很多的限制。唯一有用的只有一个用于常用的键盘操作（粗体、斜体等等操作用的）和弹出超级链接（**Ctlr+K**）

的处理器。

例如，该编辑器允许你选择一个表（指 HTML 文件中的表），但当你单击鼠标右键的时候，却不会出现什么菜单来帮助你对这个表进行操作。如果你想要实现象图 2 中显示那样的有用的界面，你需要做大量额外的工作：

- 处理 HTML 文档事件
- 建立能够识别当前正在处理的是什么对象的处理器
- 实现让你修改行为的编辑器

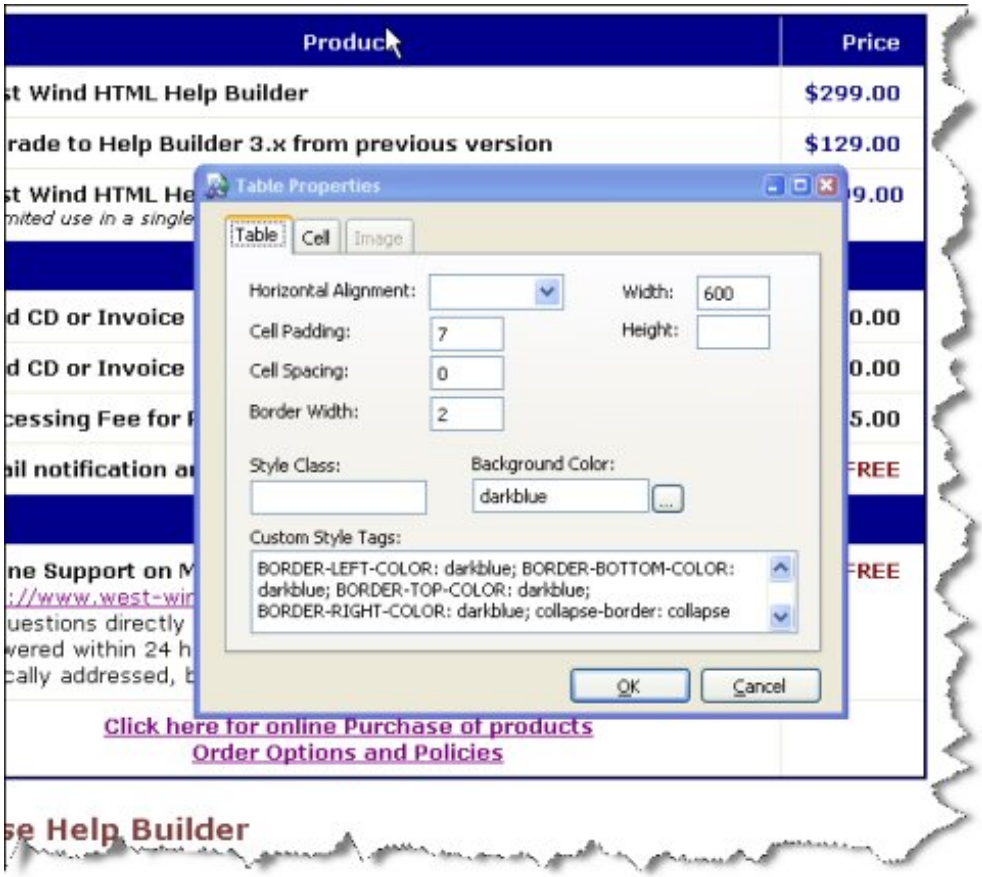

图 2、要在编辑控件上弹出自定义的菜单和编辑器，你必须处理大量的文档事件，并弄清楚活动对象的类型

如果你仔细看了 UpdateHtmlDisplay() 方法，你会从中发现一些绑定 Web 浏览器的文档事件的代码：

```
this.oTextcontainerevents = ;
    CREATEOBJECT("IHtmlTextContainerEvents")

this.oTextContainerEvents.oDoc = loEdit.Document
EVENTHANDLER(loEdit.Document.Body, ;
    this.oTextContainerEvents)
```

这些代码将事件绑定给了 HtmlTextContainerEvents 对象，该对象是一个 实现了你系统目录下的 MSHTML.TLB 文件的 COM 对象。这个强大的对象是顶层容器，它会处

理文档的所有事件，包括键盘和鼠标事件、帮助请求、以及活动的事件们。

这是在 **IE** 里的“所有事件之母”。要实现这个接口，你可以使用对象浏览器，打开 **MSHTML.TLB** 文件、找到 **IHtmlTextContainerEvents** 对象，然后把这个接口拖动到一个 **PRG** 文件中。这么做会实现这个接口，其中带有大量的空白方法。

这里是用于处理表格标记的代码（注意这里的代码是经过删节的专用于处理表格标记的相关代码；完整的代码在下载文件中）：

```
DEFINE CLASS IHtmlTextContainerEvents AS ;
    MSHTMLEventBase OLEPUBLIC
    IMPLEMENTS HTMLTextContainerEvents IN "MSHTML.TLB"

* 这里忽略了所有其它方法
PROCEDURE HTMLTextContainerEvents_oncontextmenu() ;
    AS LOGICAL
    LOCAL lcResult, loBrowser
    PRIVATE loCtl, loEvent

    TRY
        loBrowser = _VFP.ActiveForm.ActiveControl
        IF loBrowser.BaseClass != "Olecontrol"
            loBrowser=.null.
        ENDIF
    CATCH
    ENDTRY

    loCtl = this.GetCurrentElement()
    loEvent = this.GetEventObject()

    lcNodeName = loCtl.NodeName

    PUBLIC gnMenuResult
    DO CASE
    CASE lcNodeName == "TD"
        lcResult = qk_Menu( "Edit Cell Properties,"+;
            "Edit Table Properties,"+;
            "\-,"+;
            "Insert Row below,"+;
            "Insert Row above,"+;
            "Insert Row at bottom,"+;
            "Delete Row,"+;
            "\-,"+;
            "Insert Hyperlink,"+;
            "\-," + ;
```

```
                "Remove Formatting," +;
                "Insert Unformatted Line," +;
                "\-,"+;
                "Paste," +;
                "Paste Code or Formatted Text,"
                "\-," +;
                "Undo," +;
                "Redo")
        CASE lcNodeName == "TABLE"
            lcResult = qk_Menu( "Edit Table Properties,"+;
                "Delete Table,"+;
                "\-,"+;
                "Insert Row")
        ENDCASE

        IF EMPTY(lcResult)
            RETURN .F.
        ENDIF

    DO CASE
        *** 超级链接操作
        CASE lcResult = "Remove Hyperlink"
            loCtl.outerHtml = loCtl.innerText
            RETURN .t.

        *** 表选项
        CASE lcResult = "Delete Table"
            loCtl.RemoveNode(.t.)

        CASE lcResult = "Insert Row" AND lcNodeName = ;
            "TABLE"
            lnCellCount = loCtl.Rows.Item( ;
                loCtl.Rows.Length-1 ).Cells.Length
            loNewRow = loCtl.InsertRow()
            FOR lnX = 1 TO lnCellCount
                loCol = loNewRow.InsertCell()
                loCol.VAlign="top"
            ENDFOR

            *** 任何无标题的 CASE 选项必须遵照标题设置

            *** 表格单元格专用选项
        CASE lcNodeName = "TD"
            this.GetDocumentMap(loCtl)
```

```
            loRow = this.FindControlInMap(loCtl,"TR")
            loTable = this.FindControlInMap(loCtl,"TABLE")

            DO CASE
                CASE lcResult == "Delete Row" AND ;
                    !ISNULL(loRow) AND !ISNULL(loTable)
                    loTable.DeleteRow(loRow.RowIndex)
                CASE lcResult = "Edit Cell"
                    DO FORM TableProperties WITH loCtl, loTable
                CASE lcResult = "Edit Table Properties"
                    DO FORM TableProperties WITH null,loTable
            ENDCASE

            IF lcResult = "Insert Row"
                DO CASE
                    CASE lcResult = "Insert Row at bottom"
                        loNewRow = loTable.InsertRow()
                    CASE lcResult = "Insert Row above"
                        loNewRow =loTable.InsertRow(loRow.RowIndex)
                    CASE lcResult = "Insert Row below"
                        loNewRow = ;
                            loTable.InsertRow(loRow.RowIndex + 1)
                ENDCASE

                lnCellCount = loRow.Cells.Length
                FOR lnX = 1 TO lnCellCount
                    loNewRow.InsertCell()
                ENDFOR
            ENDIF
    ENDCASE

    RETURN .F.
    ENDPROC
```

你可以看到，这里涉及到了什么。首先，在头上的一些代码是用来处理找到当前选中的控件和关于当前事件的信息的。在基类中有大量的辅助方法可以轻松的向这里展示的处理代码提供这些信息。根据元素的类型，会显示一个菜单、然后返回选中的菜单项。

然后，是对每一种选择的处理。你可以看到对 **TableProperties** 表单的调用带着对 **loCtl** 和 **loTable** 两个对象的各一个引用。该表单接着从 **TABLE** 和 **TD** 元素中取出信息显示在表单上。在结束以后，该表单会将值回写到这些元素对象中去。

这个控件是纯粹的对象，所以被设置的值会立即将改动反映到文档中去。当我在表单中将一个单元格的文本对齐方式修改为居中对齐的时候，一旦 **loCtl.TextAlignment** 被

用这个值更新了，该单元格就立即会反映出这种改动。

象你也许期望的那样，这种办法可以被用来快速的处理大量的工作，因为你可能需要处理大量的对象：**Hyperlink**（超级链接）、**Image**（图片）、**Table**（表格）、选中的文本，等等。

除了我已经讲述过的对内容敏感的右键菜单以外，你还需要操心键盘捕捉的问题。在帮助生成器的主表单中，按键可以很容易的用 **ON KEY LABEL** 来捕捉，但是在 **Web** 浏览器控件中，这些控制键就不再起作用了。这是因为，从技术上来说，**Web** 浏览器控件不是你的表单的一部分，而是一个独立的 **Windows** 窗口。所以，你需要处理这些热键按键操作中的每一个，并以某种方式把它们传递给你的表单。

你也许还希望在特定的控件被激活的时候采取某种自定义的行为。例如，我实现了让表格中的 **TAB** 键跳到下一个单元格而不是将焦点移动到表格边上的另一个控件。这又是件需要整个晚上工作的活！如果你也计划这么做的话，可以看一下下载文件中的 **MSHTMLEVENTS.PRG** 文件。尽管它是应用程序专用的而不是通用的东西，它还是体现了许多你可以用在处理 **Web** 浏览器控件事件上的事情。

HTML 编辑是一项好功能，但我自己个人来说更喜欢编辑纯文本。**HTML** 编辑有一些笨拙，而且即使已经做了大量的工作，我还是没能实现按我希望的方式处理所有的任务：

还有大量没有实现的功能：

- 用不同的方式处理行中断；
- 正确的将焦点返回给控件（如果你切换到另一个应用程序后又切换回来的话，鼠标光标会丢失）；
- 使得代码片断的编辑更少错误；

显然，这些问题是从控件继承过来的。某些编辑行为的问题甚至在象 **FrontPage** 和 **DreamWeaver** 这样的高级编辑器中也能碰到。

总结

在 **Web** 浏览器中的 **HTML** 编辑模式还缺少某些关键的功能，你可以通过处理该控件的大量编辑事件并提供自定义代码来增加这些功能。帮助生成器使用这种办法来提供一种丰富的编辑经历，你也可以做到。使用在下载文件中的构想来作为你自己的起点吧！

下载：**507STRAHL.ZIP**

在 VFP 应用程序里发送 SMTP 消息，第一部分

原著：Anatoliy Mogylevets

翻译：CY

在他的网站“在 VFP 里使用 Win32 函数”（www.news2news.com），那里有许多示例代码，涉及了 Windows 操作系统编程的多个方面。发送和接收邮件的主题是一直排列在网站访问者列表的顶部。在这篇文章里，Anatoliy Mogylevets 将为你展示他的学识，并提供了你可以应用在你的应用程序里的代码。

如果我想在 **VFP** 应用程序里自动发送邮件，我该选择什么？有多种方法可以实现这个任务，并可以归纳为三类：

1.自动化一个默认的邮件客户端，比如 **Outlook Express**。

2. 使用第三方类库或控件，比如 **CDO, Blat, West Wind Web Connection**，在网站服务器上通过脚本来发送邮件，也是非常类似于使用第三方类库。

3.直接在 **VFP** 代码里调用 **Winsock API** 函数。

第一类的解决方法可以是这样的短小和简单：

```
#DEFINE SW_SHOWNORMAL 1
LOCAL lcMail

DECLARE INTEGER ShellExecute IN shell32;
    INTEGER hWindow, STRING lpOperation,;
    STRING lpFile, STRING lpParameters,;
    STRING lpDirectory, INTEGER nShowCmd

lcMail = "mailto:buddy1@somewhere.com" +;
    "?CC=buddy2@somewhere.com" +;
    "&Subject=Test message" +;
    "&Body= This is a test message.%0A" +;
    "Please do not respond."
```

```
= ShellExecute(0, "open", lcMail, "", "",;
    SW_SHOWNORMAL)
```

这是另一个代码：

```
DECLARE INTEGER MAPISendDocuments IN mapi32;
    INTEGER ulUIParam, STRING lpszDelimChar,;
    STRING lpszFullPaths, STRING lpszFileNames,;
    INTEGER ulReserved

* 附件文件列表，请确保这些文件的确存在

cFiles = "c:\logs\ex041007.log;" +;
    "c:\logs\ex041008.log;" +;
    "c:\logs\ex041009.log"

= MAPISendDocuments(_screen.HWnd, ";", cFiles, "", 0)
```

我认为，那是非常惊人的短。尽管，这两个方法都留有许多可改进的地方。当使用简单 **MAPI** 时这将会是第一类方法里最好的功能。这篇文章有不同的焦点，是因为我需要比在第一种方法里所提供的对发送过程作更多的控制。

第二种方法有很好的效率和功能，就如 **CDO** 或 **West Wind Web Connection**。典型的对象有一个接口，可以很容易使用和理解，并且它关注了在创建和发送消息的过程中的所有棘手的事。然而，不利的是，需要有外部模块与你的 **VFP** 应用程序一起发布。

那第三种又怎么样？直接以纯 **VFP** 代码调用 **WinsockAPI** 函数？它几乎有与第二种同样的功能，并提供比第一种方法对发送消息的过程更佳的控制。**WinsockAPI** 库是操作系统的组成部分，并在大多数 **Windows** 计算机上是默认安装的。

也有些不利的地方，比如需要在 **VFP** 里对 **Winsock** 方法作大量编程。同时，最大的好处是，因为你可以学到更好的理解邮件发送的里里外外。这新的知识或许与今天不相关，但将来某时它将会变得不可缺少的。

使用 **WinsockAPI** 的方法，直接发送命令和数据到邮件服务器，数据交换的协议叫作 **SMTP**，简单邮件传送协议的简称。

SMTP 消息

创建和发送电子消息的过程可以用非常普通的术语来解释，就如填写在信封上的“来自：”和“寄往：”区域，是用纸来填写信息，并把信封投递到附近的邮局。

在 **VFP** 应用程序里，**SMTP** 消息是个字符串或备注字段，是以封包开头后面包含着实

体。有时实体用特定的边界符分成多个部分。一个 **SMTP** 消息通常包含的字符是从 **CHR(32)** 到 **CHR(127)**，以及一些界外符。它防止消息内容与 **SMTP** 服务器所使用的控制字符相混淆。因为这个原因，包含这个范围外字符的部分消息必须在发送端被编码，然后在接收端被解码。我将在下面章节解释更多的细节。

SMTP 包头

包头是由名和值来成对构成的，叫做 **SMTP** 头。这里有一个简单包头的示例：

```
From: <sender@somedomain.com>
To: <somebody@somewhere.com>
Subject: Test Message
Date: Thu, 12 May 2005 14:01:58 -0400
```

下面创建包头的 **FoxPro** 代码是很显而易见的：

```
#DEFINE CRLF CHR(13)+CHR(10)
cEnvelope = "From: <devicecontext@msn.com>" + CRLF +;
"To: <somebody@somewhere.com>" + CRLF +;
"Subject: Test Message" + CRLF +;
"Date: Thu, 12 May 2005 14:01:58 -0400"
```

日期头是可选项的，大多数的 **SMTP** 服务器会对此作检查，如果它不存在就会加入的。一个包头可以包含许多其它的头。

SMTP 消息体

消息体可以包含多个部分，包括下列：

- 无格式 ASCII 文本
- 格式化文本，比如 HTML
- 内联文件
- 附加文件

当然，消息可以只由一个部分组成，如无扩展字符的无格式 **ASCII** 文本。在这样的情况下，在包头里只需要最小的 **SMTP** 头。

无格式文本的 SMTP 消息

在下面的示例中，前九行生成了包头，最后一行是消息体。那个空行表示包头的结束和消息体的开始。

```
From: <sender@somedomain.com>
To: <receiver@somedomain.com>
```

```
Cc: <another_receiver@somedomain.com>
Subject: Test message
Date: Sun, 29 May 2005 21:50:44 -0400
MIME-Version: 1.0
Content-Type: text/plain;
charset="Windows-1252"
Content-Transfer-Encoding: 7bit

This is a test message. Please do not respond.
```

当然，你可以在 **VFP** 里很容易的创建这个消息：

```
TEXT TO cMessage NOSHOW
    From: <sender@somedomain.com>
    To: <receiver@somedomain.com>
    Cc: <another_receiver@somedomain.com>
    Subject: Test message
    Date: Sun, 29 May 2005 21:50:44 -0400
    MIME-Version: 1.0
    Content-Type: text/plain;
    charset="Windows-1252"
    Content-Transfer-Encoding: 7bit

    This is a test message. Please do not respond.
ENDTEXT
```

正如你所见，“来自：”、“寄往：”和“抄送”部分包含有以单方括号包围的邮件地址。它也可能是如下把一个名字放在邮件地址前：

```
From: "Barney Gumble" <barney@boozehound.com>
```

更可能的，消息接收者将会见到 **Barney** 名字而不是邮件地址。

你或许注意到了上面消息里的 **MIME** 版本头。**MIME**，它是多用途互联网邮件扩展的缩写，允许非 **US** 的 **ASCII** 文字消息，非文字消息，多分部的消息体，和在消息头的非 **US** 的 **ASCII** 信息。这里不需要改变 **MIME** 版本头。

带选择部分的 SMTP 消息

在无格式文本的邮件消息里没有什么有趣的事，正如许多发送者想要作下划线、黑体字，甚至是在消息里有些彩色部分。因而，它是如何工作的？它非常简单：消息包含两个部分。一部分仍然是无格式文本，但第二部分是加入的 **HTML** 格式文本。

两个部分都可以传递同样的文本内容。另一方面，它们是可互换的，虽然那对 **SMTP**

服务器并不是强制的。注意 **Content** 类型头是与先前显示的无格式文本消息是不同的。

```
From: <sender@somedomain.com>
To: <receiver@somedomain.com>
Subject: Test message
Date: Sun, 29 May 2005 21:47:51 -0400
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="----=_NextPart_000_002F_01C56498.1002E940"

This is a multi-part message in MIME format.

------=_NextPart_000_002F_01C56498.1002E940
Content-Type: text/plain;
charset="Windows-1252"
Content-Transfer-Encoding: quoted-printable

This is a test message. Please do not respond.

------=_NextPart_000_002F_01C56498.1002E940
Content-Type: text/html;
charset="Windows-1252"
Content-Transfer-Encoding: quoted-printable

<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
</HEAD>
<BODY>
<DIV><FONT face=3DArial size=3D2>
This is a test message. Please do not=20
respond.</FONT></DIV></BODY></HTML>

------=_NextPart_000_002F_01C56498.1002E940--
```

这里有个问题：谁来处理 **HTML** 格式？如果你使用 **Outlook Express** 或类似的邮件客户端来发送消息，是客户端程序来完成的。如果你是自己用 **VFP** 编写的邮件程序，你的邮件程序来作的，或者至少它应该做的。

这里有另一个问题：是否只有 **HTML** 格式可以用于可选择的部分？不，事实并非如此，虽然是经常选这个的。为了使事情更复杂些，一个消息并不限于只能有一个选择部分。发送者可以根据需要创建多个选择部分。接收者的邮件客户端程序来判定哪一个选择部分更适合于显示，或者它将此决定权交给用户。

注意 **Content** 类型头包含了一个新的参数—边界，它分离开选择部分。一个边界必须

是唯一的，对于消息的其它部分来说是突出的。

这里的代码利用 **GUID** 通过 **API** 调用创建了一个边界： **FUNCTION GetBoundary**

```
DECLARE INTEGER CoCreateGuid IN ole32;
STRING @pguid

DECLARE INTEGER StringFromGUID2 IN ole32;
    STRING rguid, STRING @lpsz, INTEGER cchMax

LOCAL cGUID, cBuffer, nBufsize

cGUID = REPLICATE(CHR(0), 16)
nBufsize=128
cBuffer = REPLICATE(CHR(0), nBufsize*2)
= CoCreateGuid(@cGUID)

= StringFromGUID2(cGUID, @cBuffer, nBufsize)
cBuffer = SUBSTR(cBuffer, 1, AT(CHR(0)+CHR(0),;
    cBuffer))
RETURN STRCONV(cBuffer,6)
```

在消息里的所有边界都是统一的。他们左边填有两个破折号。另外右边也是两个破折号。它应是多分部或带选择部分的消息的最后一行。

注意在先前示例里用到的新的 **Content-Transfer-Encoding**（内容-转换-编码）的值：**quoted-printable**（可引用于打印）。这个 **quoted-printable** 和 **Base64** 是两种必要的 **MIME** 内容转换编码。它们是以特定序列的 **ASCII** 字符来来代替非 **ASCII** 字符。**VFP8** 里的 **STRCONV()**可以很容易实现与 **Base64** 的互相转换。下面的代码可将字符转换为可打印格式：

```
FUNCTION ToQuotedPrintable(cBuffer As String) As String
#DEFINE SOFT_LINE_BREAK "=" + CHR(13) + CHR(10)
#DEFINE MAX_LINE_LENGTH 71

LOCAL cResult, nIndex, ch, nLineLength
cResult = ""
nLineLength=0

FOR nIndex=1 TO LEN(m.cBuffer)
    ch = SUBSTR(m.cBuffer, m.nIndex, 1)
    nLineLength = nLineLength + 1

    DO CASE
        CASE BETWEEN(m.ch, CHR(62), CHR(126))
            * do nothing
```

```
            CASE BETWEEN(m.ch, CHR(32), CHR(60))
                * do nothing
            CASE INLIST(m.ch, CHR(10), CHR(13))
                nLineLength=0
            CASE m.ch = CHR(9)
                * do nothing
            OTHERWISE
                ch = "=" + RIGHT(TRANSFORM(ASC(m.ch), "@0"),2)
        ENDCASE

        cResult = cResult + m.ch
        IF nLineLength >= MAX_LINE_LENGTH
            cResult = cResult + SOFT_LINE_BREAK
            nLineLength=0
        ENDIF
    NEXT

    cResult = STRTRAN(m.cResult, " "+CHR(13), ;
        "=20"+CHR(13))
    cResult = STRTRAN(m.cResult, CHR(9)+CHR(13), ;
        "=09"+CHR(13))
    RETURN m.cResult
```

这里结束文章的第一部分。至此时我确信你已经知道如何创建一个 **SMTP** 消息。在第二部分，我将继续讨论如何创建一个 **SMTP** 消息和展示如何与 **SMTP** 服务器通信发送消息。

下载文件：**507ANATOLIY.ZIP**