FoxTalk

Июнь 2003
№ 6 (72)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

Управляйте приложениями при помощи данных

Дуг Хенниг (Doug Hennig)

Будучи VFP-разработчиками, мы привыкли хранить данные приложений в таблицах. Однако другим применением таблиц является хранение информации о поведении приложений. В этой статье Дуг Хенниг рассматривает, как ключевые фрагменты ваших приложений, управляемых данными, могут сделать их более гибкими и легче поддерживаемыми.

Рик Ходдер (Rick Hodder) поместил замечательную статью в октябрьском (2002 года) выпуске FoxTalk, озаглавленную «Создание подклассов готовых приложений». В этой статье он обсуждал механизм создания вариантов существующих приложений. Приведу примеры ситуаций, в которых подобный механизм может оказаться полезным. Представьте, что у вас есть приложение, созданное для одного клиента, но оно может быть использовано (с некоторыми изменениями) и другим клиентом. Или, возможно, у вас есть программный продукт, который вы приобрели для множества заказчиков, но для одного из них требуется видоизмененная версия. Я успешно применял эту технику в прошлом для создания вариантов приложений для разных заказчиков.

Однако есть и другой подход к этой идее, заключающийся в управлении вашими приложениями при помощи данных. Приложение является управляемым данными, если его части, отличающиеся для различных вариантов приложения, не являются жестко запрограммированными, а считываются из данных. Этот подход имеет несколько преимуществ, включая тот факт, что вы можете свести к минимуму потребность в создании подклассов и изменении кода. Вместо этого, вы помещаете информацию в таблицы, что намного проще для редактирования и поддержки.

Создание кода, управляемого данными, подобно созданию объектно-ориентированного кода, в котором вы рассматриваете задания, требуемые для отдельной операции, и абстрагируете их в универсальный набор. Вместо использования подклассов для реализации особого поведения, вы создаете записи в таблице.

Мы рассмотрим несколько областей, в которых вы можете управлять приложениями при помощи данных, чтобы сделать их более гибкими и легче обслуживаемыми: меню, настройки приложений и диалоговые окна. Однако сначала я хочу обсудить некоторые проблемы.

Июнь 2003

материал имеет отношение к соответствующей версии

материал имеет отношение к соответствующей платформе

исходные тексты программ можно скачать из Интернета

Проблемы

Прежде чем начать управлять всем в вашем приложении при помощи данных, стоит узнать о паре вещей.

- *Производительность*. Обычно для кода, управляемого данными, необходимо макрорасширение или функции, подобные EVALUATE(), TEXTMERGE() и EXECSCRIPT(). Будучи очень гибкими, эти функции выполняются медленнее, чем жестко запрограммированные команды или операторы присваивания. Большинство рассматриваемых нами продуктов, управляемых данными, используются при запуске приложения, когда производительность не столь важна. Убедитесь, что оптимизировали управляемый данными код, содержащий циклы или другие чувствительные к производительности части приложения. Используйте технику управления с помощью данных только если это действительно нужно, а не потому, что вы умеете это делать. Например, вы можете применять эту технику ко всем формам в период исполнения (runtime). Пустая форма заполняется элементами управления посредством сканирования таблицы, содержащей свойства для каждого элемента (используемый класс, значения Top, Left, Width, ControlSource и т. д.), и добавления их к форме при помощи метода AddObject. Этот подход имеет смысл в том случае, если формы должны выглядеть по-разному для разных заказчиков или если у пользователя должна быть возможность настраивать форму. Но это исключительные случаи, поскольку, очевидно, что эти формы будут инициализироваться гораздо медленнее своих жестко запрограммированных аналогов.
- *Безопасность*. Если ваши пользователи похожи на моих, то к некоторым из них можно применить фразу: «Недостаточное знание — опасная штука!» Если вы предоставляете таблицу настроек приложения, то подвергаетесь риску, что некто может случайно (или намеренно) удалить ее или изменить причудливым и необычным способом. Вам придется управлять ситуациями, когда таблица утрачена или повреждена, содержит неразрешенные или выходящие за допустимые рамки значения или содержит настройки, не устраивающие, в конечном счете, пользователя. В этом случае опция «восстановление значения по умолчанию» может оказаться очень полезной. Стоит также подумать о том, чтобы сделать вашу таблицу доступной только из вашего приложения, скажем, посредством шифрования таблицы или включения ее в EXE-файл, если ее содержание не меняется при работе приложения.

Управляемое данными меню

Одним из последних оплотов процедурного кода в VFP является система меню. Хотя VFP имеет встроенный генератор меню и нам не приходится создавать код меню вручную, все же генерируемый код является статическим. Это означает, что меню не может быть использовано повторно, оно даже не может быть программно изменено во время исполнения. VFP-разработчики в течении ряда лет просили объектно-ориентированную систему меню, но безуспешно. К счастью, можно создать свою собственную объектно-ориентированную систему меню (посмотрите мою статью «Сделайте ваши меню объектными» в выпуске FoxTalk за июнь 2002 года).

Раз уж вы можете создать подклассы на основе классов меню, реализованных в стиле ООП, или использовать процедурный код для создания специальных систем меню из этих классов, почему бы не попробовать управлять меню с помощью данных? Это значительно упростит изменение меню для отдельных вариантов приложения: просто изменим содержание таблицы, хранящей информацию об этом меню, и меню приложения изменится.

Таблица 1 показывает структуру таблицы меню, содержащей определение меню приложения. Эта таблица используется программой CreateMenu.prg для создания меню приложения. Утилита в качестве параметров принимает имя и путь к таблице описания меню и имя переменной или свойство, хранящее объектную ссылку на это меню (чтобы понять, для чего это требуется, см. указанную выше статью). CreateMenu присваивает значение элементу SFMenu, а затем открывает заданную таблицу, используя индексный ключ Order (сортирующий таблицу таким образом, что сначала идут пункты меню, а затем команды меню). Затем таблица обрабатывается так, что активные элементы добавляются к меню (пункты меню в случае «P»-записи, команды под заданными родительскими пунктами в случае «B»-записи) с атрибутами, определенными в таблице.

```
lparameters tcMenuTable, tcMenuObject
local loMenu, lnSelect, lcName, lcClass, ;
   lcLibrary, loItem, lcParent
* Создает объект меню.
loMenu = newobject('SFMenu', 'SFMenu.vcx', '', ;
   tcMenuObject)
* Открывает таблицу меню и обрабатывает все активные записи
lnSelect = select()
select 0
use (tcMenuTable) order Order
scan for Active
* Если это пункт меню, добавляет его в меню.
   if RecType = 'P'
     lcName    = trim(Name)
     lcClass   = iif(empty(Class), 'SFPad', trim(Class))
     lcLibrary = iif(empty(Library), 'SFMenu.vcx', ;
       trim(Library))
     loItem    = loMenu.AddPad(lcClass, lcLibrary, lcName)
```

Таблица 1. Структура таблицы меню.

Имя поля	Тип/Размер	Описание
RecType	C(1)	Тип создаваемого объекта меню: “P” для пункта меню или “B” для команды меню.
Active	L	.T., если эта запись должна быть использована.
Order	I	Порядок, в котором объекты должны добавляться к меню.
Name	C(30)	Имя, назначаемое объекту меню (например, “FilePad”).
Parent	C(30)	Имя родительского объекта для этого объекта (например, команда меню FileOpen может определять “FilePad” как свой родительский объект).
Class	C(30)	Класс, из которого присваивается значение объекту меню; оставьте это поле пустым, чтобы использовать SFPad для пункта меню и SFBar для команды меню.
Library	C(30)	Библиотека содержит класс, определенный в поле Class; оставьте это поле пустым, чтобы использовать класс в SFMenu.vcx.
Caption	C(30)	Заголовок для элемента меню; поместите выражение в ограничители слияния (например, “Выставление счета для <<oApp.cCustomerName>>”).
Key	C(10)	Горячая клавиша для элемента меню.
KeyText	C(10)	Текст, отображаемый в меню для горячей клавиши.
StatusBar	M	Текст, отображаемый в строке состояния для элемента меню; поместите выражение в ограничители слияния текста.
Command	M	Команда, которая будет выполняться, когда элемент меню выбран; поместите выражение в ограничители слияния текста.
Clauses	M	Использование дополнительных пунктов меню; поместите выражение в ограничители слияния текста.
PictFile	M	Имя и путь к используемому файлу изображения; поместите выражение в ограничители слияния текста.
PictReso	M	Имя для системной команды VFP, чье изображение должно использоваться для этой команды; поместите выражение в ограничители слияния текста.
SkipFor	M	Выражение SKIP FOR для элемента меню; поместите выражение в ограничители слияния текста.
Visible	M	Выражение, определяющее, будет ли элемент меню видимым или нет; оставьте это поле пустым, чтобы элементы были всегда видимыми.

```
* Если это команда меню, добавляет ее
* к определенному заданному меню.
   else
     lcName    = iif(empty(Name), sys(2015), trim(Name))
     lcParent  = trim(Parent)
     lcClass   = iif(empty(Class), 'SFBar', trim(Class))
     lcLibrary = iif(empty(Library), 'SFMenu.vcx', ;
       trim(Library))
     loItem    = loMenu.&lcParent..AddBar(lcClass, ;
       lcLibrary, lcName)
   endif RecType = 'P'
* Присваивает значения для свойств пункта или
* команды меню из таблицы меню. Обратите внимание,
* что используется функция TEXTMERGE(), которая допускает
* использование выражений в любом поле.
   with loItem
     if not empty(Caption)
       .cCaption = textmerge(trim(Caption))
     endif not empty(Caption)
     if not empty(Key)
       .cKey = textmerge(trim(Key))
     endif not empty(Key)
     if not empty(StatusBar)
       .cStatusBarText = textmerge(StatusBar)
     endif not empty(StatusBar)
     if not empty(SkipFor)
       .cSkipFor = textmerge(SkipFor)
     endif not empty(SkipFor)
     if not empty(Visible)
       .lVisible = textmerge(Visible)
     endif not empty(Visible)
     if RecType = 'B'
       if not empty(KeyText)
         .cKeyText = textmerge(trim(KeyText))
       endif not empty(KeyText)
       if not empty(Command)
         .cOnClickCommand = textmerge(Command)
       endif not empty(Command)
       if not empty(Clauses)
         .cMenuClauses = textmerge(Clauses)
       endif not empty(Clauses)
       if not empty(PictFile)
         .cPictureFile = textmerge(PictFile)
       endif not empty(PictFile)
       if not empty(PictReso)
         .cPictureResource = textmerge(PictReso)
       endif not empty(PictReso)
     endif RecType = 'B'
   endwith
endscan for Active
use
select (lnSelect)
return loMenu
```

Чтобы посмотреть, как это работает, запустите Test-Menu.prg. Она вызовет CreateMenu с указанием, что Menu1.dbf является таблицей описания меню.

Настройки приложения, управляемого данными

При использовании некоторых каркасов приложений (application framework), вы создаете новое приложение, создавая дочерние классы на основе класса application и настраивая их свойства в окне свойств или в коде. Другие каркасы, напротив, используют неизменяемый класс application и конфигурационную таблицу настроек. Каждый из этих подходов имеет свои преимущества, но иногда гораздо проще изменить содержание таблицы, чем создавать подклассы.

Механизмом для сохранения настроек приложений, управляемых данными, могут служить INI-файл, реестр Windows, таблица или их комбинация. Например, я обычно храню настройки, относящиеся к определенному пользователю, в реестре, но общие настройки — в таблице. Однако там, где пользователям необходимо иметь возможность выполнять некоторые простые настройки, в особенности извне приложения, вы не можете обойтись без INI-файла.

Для настроек, хранящихся в таблице, выбранный мною подход заключается в том, что вместо жесткого программирования списка свойств, восстанавливаемых из таблицы, я использую менеджер настройки так, чтобы не только значения свойств управлялись данными, но также и процесс их восстановления. Приведу пример. Допустим, одним из свойств вашего объекта application является его имя (например, «Visual Cash Register»). Вы всегда определяли его, вводя в окно свойств, поскольку оно не меняется в процессе исполнения. Пусть один из вариантов приложения, который вы хотите создать, требует другого имени (например, «Visual Cash Register for AccountMate»). Для восстановления этого свойства из конфигурационной таблицы вам потребуется добавить код в объект application, выполняющий эту задачу. Скорее всего, потребуется пара строчек, но они должны быть написаны и протестированы.

Используя менеджер настройки, содержание конфигурационной таблицы управляет тем, какие свойства должны быть восстановлены. Управление ранее жестко прописанным свойством начинается с добавления новой записи в таблицу.

Менеджер настройки, который я использую, это SFConfigMgr в SFPERSIST.VCX. Его свойство cFileName содержит имя конфигурационной таблицы, из которой проводится восстановление; значение по умолчанию — SFConfig.dbf. SFConfigMgr взаимодействует с другим объектом, SFPersistentTable, в том же VCX-файле (этот класс был обсужден в FoxTalk в моей сентябрьской статье 2000 года «Постоянство без проблем»), выполняющим фактическую работу по получению значения из таблицы и записи свойства объекта. Единственным public-методом этого класса является Restore, восстанавливающий все настройки в конфигурационной таблице для определенного объекта, и Reset, возвращающий менеджер в исходное состояние. Этот класс вы можете найти на сопровождающей дискете.

Таблица 2 показывает структуру конфигурационной таблицы (SFConfig.dbf), а таблица 3 — некоторые примеры записей в таблице. Обратите внимание, что значение может быть выражением, заключенным между разграничителями слияния текста; менеджер настройки будет применять к этому значению функцию TEXTMERGE(), поэтому оно может содержать любое допустимое VFP-выражение, даже если оно вызывает некоторую функцию или метод для определения значения, например, чтение из регистра или из INI-файла. Также заметьте, что поле Group может быть использовано для определения того, какому компоненту предназначена каждая настройка. В таблице 3 вы видите, что два различных объекта восстановят свои настройки из конфигурационной таблицы — это объект Application и объект Logo.

Таблица 2. Структура SFConfig.dbf.

Имя поля	Тип/Размер	Описание
Key	C(25)	Ключ для данной записи.
Target	M	Свойство, в котором хранится значение.
Value	M	Значение для настройки.
Group	C(25)	Используется для настройки группы по типу.
Type	C(1)	Тип данных для значения. Поскольку он хранится в текстовом поле, менеджер настройки должен преобразовать строчное значение в подходящий тип данных.

Для восстановления своих настроек объект просто вызывает метод Restore менеджера настроек, передает ему имя группы для восстановления настроек и ссылку на себя, чтобы менеджер смог обновить его свойства. Следующий код, взятый из метода Init объекта SFApplication, присваивает значение “SFConfigMgr” свойству oConfigMgr и восстанавливает все настройки в группе “Application”.

```
This.oConfigMgr = newobject('SFConfigMgr', ;
   'SFPersist.vcx')
This.oConfigMgr.Restore('Application', This)
```

Класс SFLogo использует похожий код для получения имени файла изображения для логотипа.

Диалоговые окна, управляемые данными

Многие приложения имеют диалоговое окно About, отображающее номер версии, текущий каталог, расположение файлов данных, имя текущего пользователя и т. д. Вы можете создать класс диалоговых окон About, имеющий требуемое поведение и вид, и

Таблица 3. Настройки в таблице SFConfig.dbf для примера приложения.

Key	Target	Value	Group	Type
Application Name	cAppName	My Sample Application	Application	C
Version	cVersion	1.0	Application	C
Data Directory	cDataDir	K:\Apps\Data\	Application	C
Menu Table	cMenuTable	Menu.dbf	Application	C
Logo Image	Picture	<<_samples + 'tastrade\bitmaps\ttradesm.bmp'>>	Logo	C

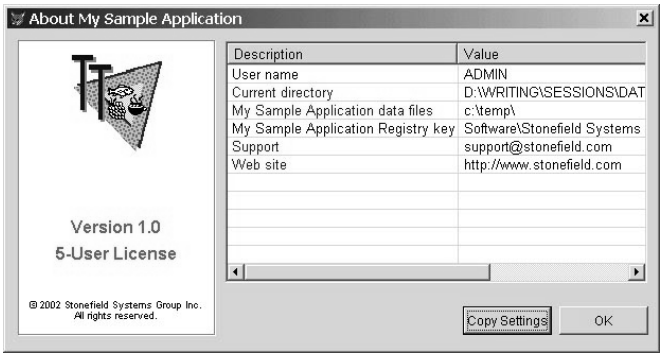

Рис. 1. Диалоговое окно About, управляемое данными, показывает настройки для примера приложения.

затем построить на его основе дочерний класс для конкретного приложения. Однако, если диалоговое окно About управляется данными, то нет необходимости создавать подклассы. Вы можете просто определить сведения, которые необходимо отображать, в диалоговом окне, используя таблицу.

Рисунок 1 показывает, как выглядит окно, построенное на основе класса SFAbout – управляемого данными класса диалоговых окон About. Этот класс использует SFAbout.dbf для определения настроек, отображаемых в списке, и других свойств объекта application (таких, например, как заголовок, версия, авторское право, адрес электронной почты и так далее).

Кроме того, что этот класс имеет список настроек, управляемых данными, также у него есть дополнительные приятные прибамбасы:

- Его размер и расположение сохраняются и восстанавливаются из реестра (используя ключи, хранимые в свойстве cRegistryKey объекта Application) при помощи объекта SFPersistentResizer на форме.
- Когда форма изменяет свои размеры, объект SFPersistentResizer также обрабатывает размер и положение элементов управления формы.
- Ширина столбца ListView также сохраняется и восстанавливается из регистра.
- Элементы в ListView могут содержать гиперссылки. Когда пользователь щелкает этот элемент, то класс использует функцию Windows API ShellExecute для «запуска» этого значения (я обсуждал эту функцию в статье за апрель 2002 года «Головокружение от интерфейса Windows API»). Такие элементы выделены синим цветом в списке. Это наиболее полезно для таких вещей, как имя каталога (запускающее Windows Explorer для этого каталога), адрес электронной почты (открывающий окно New Message почтового клиента по умолчанию) и Web-сайт (загружающий этот Web-сайт в браузере по умолчанию). Это поведение также может быть изменено для поддержки вызова таких функций в приложении, как отображение диалогового окна профиля пользователя, появляющееся при щелчке имени пользователя.
- Кнопка Copy Settings копирует настройки в буфер обмена Windows.
- Окно логотипа слева является классом (SFLogo), получающим информацию из свойств oApp и из конфигурационной таблицы (имя используемого файла изображения).

Метод LoadList, вызываемый из Init, достаточно прост: он открывает таблицу SFAbout.dbf и проходит через активные записи, вызывая метод AddSettingToList для каждой из них и добавляя ее в список.

```
local lnSelect, lcValue, lcLink
lnSelect = select()
select 0
use SFAbout order Order
scan for Active
  lcValue = evaluate(ValueExpr)
  lcLink  = iif(empty(LinkExpr), lcValue, ;
    evaluate(LinkExpr))
  This.AddSettingToList(textmerge(trim(Descrip)), ;
    lcValue, Link, lcLink)
endscan for Active
use
select (lnSelect)
```

Метод AddSettingToList добавляет специальные настройки и значения в элемент ListView. Если элемент содержит гиперссылку, он отображается синим цветом, и цель для функции ShellExecute хранится в свойстве Tag элемента списка.

```
lparameters tcDescription, tuValue, tlHyperlink, tcLink
local lcValue, loItem
lcValue = transform(tuValue)
if vartype(tcDescription) = 'C' and ;
  not empty(tcDescription) and ;
  len(tcDescription) <= 100 and ;
  not empty(lcValue) and len(lcValue) <= 255
  loItem = This.oList.ListItems.Add(, sys(2015), ;
    tcDescription)
  loItem.SubItems(1) = lcValue
  if tlHyperlink
    loItem.Forecolor = rgb(0, 0, 255)
    loItem.Tag       = tcLink
  endif tlHyperlink
endif vartype(tcDescription) = 'C' ...
return
```

Метод ListViewItemClick формы, вызываемый щелчком на элементе в ListView, просто вызывает ShellExecute для «запуска» выделенного элемента.

```
lparameters toItem
 if not empty(toItem.Tag)
   declare integer ShellExecute in SHELL32.DLL ;
     integer nWinHandle, string cOperation, ;
     string cFileName, string cParameters, ;
     string cDirectory, integer nShowWindow
   ShellExecute(0, 'Open', toItem.Tag, '', '', 1)
 endif not empty(toItem.Tag)
```

❑❑❑

Каким способом переданы данные?

Нэнси Фолсом (Nancy Folsom)

В предыдущей статье Нэнси Фолсом показала простой пример того, как форма, бизнес-объект и оболочка данных (wrapper) могут работать вместе, чтобы сделать по сути дела то, что мы делаем всегда, — отобразить некоторые данные в раскрывающемся списке. Это было, по меньшей мере, началом отделения данных от форм, и мы увидели, как можно начать разделение слоев приложения без того, чтобы полностью его переписывать. Однако данные и бизнес-логика могут быть еще дальше отнесены от пользовательского интерфейса. В своей статье Нэнси развивает эту абстракцию и создает более сложный набор классов-оболочек, приспособленных для доступа к данным как средствами FoxPro, так и ADO — хорошо знакомым нам способом.

Независимо от того, какой источник данных использует приложение FoxPro — будь-то SQL Server, Oracle, FoxPro или Access — элементы интерфейса FoxPro легко привязываются к полям курсоров переменным и свойствам. Связанные элементы представляют значительный прогресс по сравнению со SCATTER и GATHER, поскольку они упрощают привязку данных с тем, что ввел пользователь. Среда описания данных (DataEnvironment) позволяет упростить открытие таблиц, установку связей и индексов и т. д. Кроме того, DataEnvironment хорошо «убирает» за собой, после того, как форма закрыта. В сочетании с буферизацией и транзакциями, значительная часть логики, которую приходилось непосредственно кодировать при создании ранних приложений xBASE, преодолена.

Однако сегодня, чтобы продвинуться вперед, нужно отложить в сторону все эти замечательные вещи – описание окружения данных формы, напрямую привязанные к данным элементы управления и код работы с таблицами, размещенный в стиле xBASE непосредственно в форме. Иногда прогресс выглядит как оглядка назад, а не шаг вперед. Но это не возвращение в прошлое, а только перенос полезных концепций вперед, по мере нашего продвижения по тропе технологического прогресса.

Итак, может ли одна часть приложения взаимодействовать с другой? Естественный и корректный подход к работе с компонентами интерфейса пользователя в FoxPro состоит в работе с табличными курсорами, поскольку они могут создаваться из FoxPro-таблиц, объектов ADO recordset и даже из текстовых файлов XML. Не только объекты интерфейса FoxPro успешно обрабатывают курсоры, но это также хорошо делают FoxPro-разработчики. Этот подход имеет и дополнительные преимущества в силу того, что работа с элементами интерфейса и бизнес-логикой выполняется также, как и работа с таблицами.

Чтобы выполнить эту задачу, вы должны рассматривать все данные как обновляемое представление, локальное или удаленное. Этот подход работает прекрасно и, по крайней мере, одна коммерческая разработка использует его. В этой статье, тем не менее, я расскажу и о другом подходе. Этот подход основан на объектах, служащих связующим звеном между представлениями, ограничениями (constraint) и доступом к данным. Преимущество данного подхода заключается в том, что он не использует контейнеры базы данных для представлений и поэтому неуязвим в случае изменения источника данных, который потом должен быть повторно создан, как это происходит в случае представлений. Недостатком этого подхода является то, что он представляется довольно сложным и выглядит довольно странно, и он не настолько прямолинеен, как работа с курсорами. Однако этот подход помог мне понять, как я могу разделить ответственность за выполнение задач между объектами.

Примечание: «курсор» — это только имя для некоторого набора данных, расположенного в памяти, который выглядит и работает как традиционная Fox-

Бизнес-логика

«Бизнес-логика» — это иное название для ограничений или пределов, в которых данные поддаются интерпретации в заданном контексте. Смелым примером этого может служить аргумент о том, что связанные элементы являются бизнес-логикой, а не логикой данных. Например, с точки зрения таблиц, не стоит беспокоиться, если очередной элемент ссылается на инвентаризационный элемент, которого нет в инвентаризационной таблице, расположенной в базе данных. Но клерк по продажам и собственник магазина, естественно, сделают это и они находятся на бизнес-стороне вещей, что означает, что они удалят наши проверки. В сообществе FoxPro «бизнес-логика» — наиболее общий термин. Вне FoxPro вы также можете услышать о подобных идеях, упоминаемых как «ограничения».

Рис. 1. Исходная иерархия класса данных.

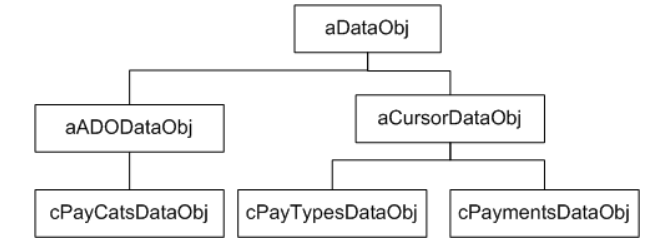

Рис. 2. Углубленная организация класса.

Pro таблица. MyCustomers.DBF является таблицей. После того, как я открыла (USE) MyCustomers.DBF, ссылка на открытую таблицу является курсором.

Позвольте мне сначала восстановить организацию класса, введенного мною в предыдущей статье. Я воспользовалась классом ComboBox, использующим бизнес-объект для взаимодействия с объектом данных для обработки типов платежей в контексте входящих платежей. Я показала пользу класса при создании рабочих форм для типов платежей, а затем, когда начальница привыкла к этому, обновила их, чтобы отобразить изменения в структуре (я добавила поле для категории типа платежа для наличных, кредита и т. д.) Этот исходный класс разговаривал на FoxPro и был разделен на подклассы для платежей и типов платежей. Теперь я могу изучить эту структуру и посмотреть, как она будет работать (или не будет) для объекта ADO recordset.

Примечание: для выполнения серьезной n-уровневой разработки, я настоятельно рекомендовала бы использовать коммерческие продукты (framework), соответствующего класса. Управление удаленными модификациями, транзакционной целостностью и различиями провайдеров — непростая задача. Однако, даже если ваша серверная часть остается на FoxPro, вы можете выиграть от использования этого технического приема.

Исходная структура класса, обобщенная на рис. 1, являлась плоской. Здесь присутствовал только один абстрактный класс – тот, что устанавливает методы и свойства и реализует главную функциональность, а затем прямо соединяет подклассы. Я подозреваю, что этот подход достаточно прост. Его разгадка пришла из размышлений о том, как применить его к объектам ADO recordset.

Я люблю работать с курсорами, поэтому моим первым импульсом был использовать RSToCursor() и CursorToRS() для преобразования набора данных в курсор. На самом деле, моим первым побуждением было использовать набор данных (recordset), но в этой статье будет лучше продолжить работу с курсорами в качества рабочего формата данных. Поскольку эти функции не являются официальной частью языка, никто не даст гарантии, что они войдут в будущие версии FoxPro, или что ошибки (если они возникнут), могут быть исправлены. Кроме того, работа с этими функциями дает ощущение даже большей оторванности, чем работа с оторванными наборами данных, и поэтому я решила продолжить работу с объектом данных, подобным recordset.

Примечание: RSToCursor() и CursorToRS() не являются собственными функциями FoxPro. Они становятся доступными после создания экземпляра объекта VFPCOM. Прочтите информацию об этих функциях и библиотеке VFPCOM.DLL, а также о том, как получить их, по адресу: http://msdn.microsoft.com/vfoxpro/downloads/readme_VFPCOM.asp.

Приняв это решение, становится очевидным, что исходная иерархия класса данных не подходит для использования объекта ADO recordset, потому что она приспособлена для работы с курсорами. Например, вот фрагмент из исходного кода метода aDataObject.Insert():

```
*!* aDataObject.Insert()
 LOCAL lcPK, l_ID
 WITH THIS
   lcPK = .PRIMARYKEY
   l_ID = .GetNewPK()
   INSERT INTO (.SOURCE) (&lcPK) VALUES (l_ID)
   .SET(.PRIMARYKEY, l_ID)
 ENDWITH
 llOK = _TALLY > 0
```

Это хорошо работает для курсоров, но не для наборов данных. Поскольку “Insert” является обычной операцией над данными, тем не менее, этот метод хорош. В коде, однако, должна быть использована основанная на курсорах производная этого класса. Исправленная иерархия класса показана на рис. 2.

Принимая во внимание то, что операции и свойства являются общими для всех данных и затем выбрав те из них, что связаны с операциями курсоров и ADO, я закончила классификацию моих классов данных, описанную ниже. Чтобы проверить мой подход на источнике данных ADO, я решила использовать данные FoxPro, поскольку FoxPro 7 включает OLEDB-провайдера. Это означает, что в ходе постро-

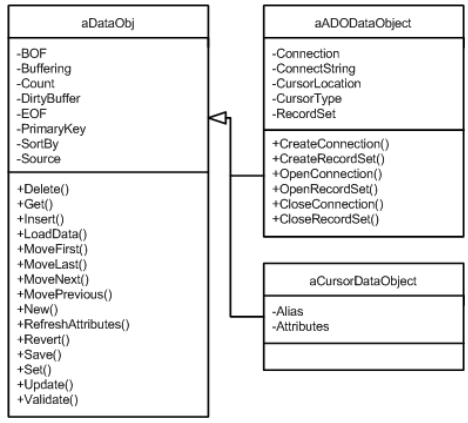

Рис. 3. Классы данных показывают, как специализируются ADO-класс и класс курсоров.

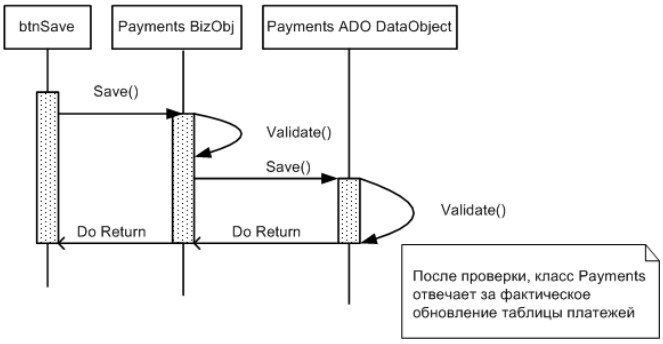

Рис. 4. Интерфейс пользователя, объекты бизнес-логики и данных взаимодействуют друг с другом для сохранения платежа.

ения и тестирования я могу работать полностью в рамках FoxPro IDE и, поскольку я чувствую себя уютно с данными Fox, в случае необходимости будет не трудно углубиться в детали этого процесса. Кроме того, чтобы проверить производительность и надежность, можно будет легко сравнить использование подхода FoxPro ADO и собственного FoxPro-подхода.

Каждый вид курсора содержит некоторые общие возможности, такие, например, как поддержка SKIP для физического перемещения по записям, независящие от типа курсора. Также и с данными ADO, вы выполняете некоторые обычные вещи, подобные созданию строк соединения. Это указание на второй уровень абстракции класса – факт заключается в том, что существует некоторое поведение, общее для каждой профессии. Можно привести в качестве примера кардиохирурга и педиатра, нуждающихся в нитях для наложении швов и скальпелях, хотя физически они выполняют очень разные вещи с данными инструментами.

Свойства и методы общего класса данных и для ADO, и для FoxPro-классов показаны на рисунке 3. Важно отметить, что различие между классом FoxPro (типа курсора) и ADO-классом не настолько велико. Другими словами, независимо от того, как это делается, происходят достаточно одинаковые вещи. Например, сравните код, помещенный мною в простые методы MoveFirst() для курсора и ADO-класса:

```
*!* aCursorDataObject.MoveFirst()
 GO TOP IN (THIS.ALIAS)
 THIS.RefreshAttributes()

 *!* aADODataObject.MoveFirst()
 THIS.Recordset.MoveFirst()
```

Обычный интерфейс любого клиента (любого бизнес-объекта) может вызвать MoveFirst(). Клиенту не нужно заботиться о том, что будет происходить. Вопрос заключается в том, как вызывается класс данных MoveFirst? Давайте посмотрим на это из интерфейса пользователя, расположенного под классом данных.

На сопровождающей дискете вы найдете пример формы, называемой SPOOPPaytypesEx7.SCX. Эта форма отображает платежи, используя, в конечном счете, класс ADO recordset для платежей. Только для забавы раскрывающийся список использует класс данных типа курсора для отображения этих данных. (Все эти данные являются данными FoxPro, что не подходит к описанному примеру). Если поле ввода не связано с control source, и payments не является курсором, как интерфейс пользователя выполнит обычные вещи, ожидаемые пользователем?

В пример класса библиотеки, SPOOP3.VCX, я включила набор кнопок с основными операциями: Add, Delete, Save и Cancel. Если щелкнуть по кнопке Save, выполняется целый каскад действий, как это показано на рисунке 4.

В следующий раз

В следующий раз я углублюсь в детали группы кнопок и полей ввода, чтобы показать, как они соединяются с бизнес-объектами.

Нэнси Фолсом (Nancy Folsom) занимается разработкой Xbase-приложений обработки бизнес-данных как для частных, так и для правительственных заказчиков, начиная с 1989 года. Она публикует свои статьи в журналах CoDe Magazine (EPS Publishing) и Virtual Fox User Group Newsletter и является автором книги Debugging Visual FoxPro Applications (издательство Hentzenwerke Publishing). Ее опыт, помимо прочего, включает разработку БД для таких сфер деятельности, как бухгалтерский учет, оценка стоимости, хозяйственное управление, управление проектом, кассовые терминалы, вексельные операции для заказчиков как частного, так и из общественного секторов. nancy@mvps.org.

❑❑❑

Удаление адресов электронной почты

Уилл Хентцен (Whil Hentzen)

WIN 7.0 6.0 3.0

Кампании продаж, использующие списки адресов электронной почты, это реальность сегодняшнего дня. Если у вас есть список клиентов или потенциальных клиентов, имеет смысл использовать этот список для информирования их о продуктах и услугах, в которых они могут быть заинтересованы. Однако проблема заключается в том, что люди меняют свои электронные адреса довольно часто, и они не любят получать нежелательную электронную почту — это тоже реальность сегодняшнего дня. Люди, которые были заинтересованы в чем-то в определенный момент времени, могут утратить этот интерес, и вам необходимо иметь возможность удалить их, а также их электронные адреса из списков рассылки почты и сделать это эффективно и аккуратно. В этой статье Уилл Хентцен предлагает один из способов решения этой задачи.

У меня большое количество электронных адресов как для текущих, так и для прошлых заказчиков, а также для заказчиков потенциальных — людей, загружавших что-либо с нашего сайта или другим способом интересующихся нашей продукцией и услугами. Я делал массовые рассылки электронных писем по этому списку по разным причинам, в основном, чтобы дать им возможность узнавать о новых книгах, а также различные новости. Некоторые из этих электронных адресов устарели — люди сменили место работы, поменялись сервера электронной почты или их учетная запись была закрыта. Кроме того, иногда люди хотят, чтобы их электронный адрес был просто удален, потому что они хотят получать почту на другой адрес или просто утратили интерес к нашим предложениям.

Важно отметить, что каждый, чей адрес содержится в нашей базе данных, имел определенные бизнес-отношения с нашей компанией – либо они покупали копию “TUFR” в 1994 году, либо были в моем списке “Mother of All User Group Lists” 1997 года или скачивали файлы для нашей книги, приобретенной ими в 2000 году. Я никогда не покупал списков рассылки у третьей стороны – в моем присутствуют только те люди, которые контактировали с нами когда-либо в прошлом.

Поэтому я чувствую большую персональную ответственность и не хотел бы досаждать тем, кто не хочет больше получать мои электронные письма. Проще всего было бы пометить этих заказчиков в моей базе данных таким образом, чтобы я не посылал им больше писем. Обратите внимание, что запросы на удаление должны быть выполнены – они попросили удалить их адреса, и я не хочу беспокоить их еще раз. Электронные письма, вернувшиеся из-за неверных адресов, конечно, никого не беспокоят, они только затрудняют работу. Зачем тратить полосу пропускания и время для рассылки писем, если они все равно будут возвращены назад? Поэтому я решил также помечать эти адреса как «плохие».

В этой статье я собираюсь обсудить, как обрабатывать запросы на удаление. В следующей я расскажу об обработке вернувшихся назад писем.

Как рассылаются электронные письма

Когда я делаю массовую рассылку электронных писем, каждое письмо состоит из четырех важных частей.

Первая из них – обратный адрес. Я использую в качестве обратного адреса remove@hentzenwerke.com, чтобы я мог идентифицировать запросы на удаление и возвращенные письма, как приходящие в ответ на массовую рассылку, а не по какой-либо иной причине. С распространением вируса Klez (что явилось следствием наличия уязвимостей в системе безопасности Outlook), мы получили огромное количество «сторонних переадресовок», появившихся как результат возврата писем на некоторые из наших опубликованных альтернативных адресов электронной почты, нашедших свой путь во множество адресных книг пользователей Интернета. Эти испорченные ответы обычно могут быть идентифицированы как таковые, поскольку они возвращаются на один из этих опубликованных альтернативных адресов, а не на специальный «удаляющий» адрес.

Второй частью является первая строка в теле электронного письма. Она сообщает читателю, что инструкция по удалению его адреса из списка рассылки расположена в конце электронного письма и выглядит так: (*Мы не хотим доставлять вам беспокойство. В нижней части этого сообщения вы найдете инструкцию по удалению вашего адреса из списка рассылки*).

Третья часть электронного письма – это основная часть сообщения (не включающая инструкцию по удалению, расположенную внизу).

И, в заключение, последней важной частью электронного письма является сама инструкция по удале-

нию адреса из списка рассылки. Вот пример того, что мы помещаем в конце каждого сообщения электронной почты:

Инструкция по удалению:
Это электронное сообщение было послано HERMAN@WERKE.COM потому, что вы приобрели у нас нечто, спрашивали о чем-то или загружали что-то с нашего web-сайта. Если вы не хотите в дальнейшем получать от нас электронные сообщения, щелкните REPLY и введите слово "REMOVE" в строку темы сообщения, чтобы быть удаленным из всех будущих рассылок Hentzenwerke. Пожалуйста, удостоверьтесь, что тело этого сообщения включено в ваш ответ, поскольку некоторые адреса в нашей базе данных могут быть перенаправлены на альтернативные адреса и мы не сможем отследить их и выполнить удаление. Спасибо!

Обратите внимание на то, что я включил электронный адрес, на который электронное сообщение было послано, в его тело, поэтому, если получатель следует инструкции, я смогу правильно определить этот адрес и отметить его. Это важно, поскольку (и это написано в самой инструкции по удалению) наибольшей проблемой в выполнении запроса на удаление является то, что люди часто пересылают сообщение с одной учетной записи на другую и, в некоторых случаях, сами забывают, как они это сделали. Потом они отвечают, используя вторую учетную запись, но, поскольку этот электронный адрес не существует в нашей базе данных, попытка автоматически удалить его не сработает, и исходный электронный адрес пользователя останется неотмеченным.

Люди остаются людьми, поэтому это не всегда срабатывает. Некоторые из них просто не следуют инструкции. Они щелкают кнопку ответа, но не используют слово "remove", поскольку не уверены, что знают, что с ним делать. Другие включают "remove" в тело сообщения, так что его не так просто найти в массе возвращаемых писем, приходящих на адрес remove@hentzenwerke.com.

Остальные же следуют инструкции, но только отчасти. Они включают "REMOVE" в строку темы сообщения, но в то же время удаляют тело сообщения. И всегда найдутся такие, кто ответит не с того адреса электронной почты, который содержится в нашей базе данных, так что не будет никакой возможности сопоставить их с адресами, на которые электронное сообщение было послано. Кроме того, некоторые системы электронной почты настроены так, что используют другие электронные адреса в качестве «исходящего адреса», хотя пользователь уверен, что он использует свой собственный электронный адрес. Например, вы можете считать, что у вас следующий электронный адрес: herman_werke@mycompany.com. Но когда вы отвечаете на электронное сообщение, ваш «исходящий» адрес будет выглядеть так herman_werke@e-mail.mycompany.com. В каждой из этих особых ситуаций требуется применить «ручной» подход.

Итак, все вышесказанное означает, что задача автоматизации процесса обработки запросов на удаление не является такой простой. Как и прежде, большинство запросов на удаление может быть обработано только в псевдо-автоматическом режиме. В этой статье я расскажу о найденном мною наилучшем подходе и о том, как его применить.

Использование автоматизации

Поскольку все эти запросы на удаление приходят в Outlook PST, моей первой мыслью было выполнить эту задачу через PST, используя автоматизацию Outlook и разбирая каждое сообщение индивидуально. Процесс должен был выбирать электронный адрес из запроса на удаление, искать его в моей базе данных CUST и помечать его для удаления. Я предполагал также помещать каждое успешно разобранное сообщение в отдельную папку, чтобы знать, для каких сообщений электронные адреса были помечены.

Я провел около половины рабочего дня за написанием этого кода... ладно, сознаюсь, на самом деле я провел три часа и 45 минут, изучая справочный файл по автоматизации и доставая друзей, и 15 минут за написанием кода. И хочу сказать: это был ужасный опыт и вот почему:

- Он работает медленно. Выполняя удаление через PST-файл, мы ищем сообщение об удалении и затем разбираем его – это не слишком длительный процесс. Но я разработчик, использующий FoxPro. Я не привык видеть, что за время обработки моих данных вырастает новая трава!
- Он громоздкий. Необходимо проделать массу работы для того, чтобы найти нужный элемент, пройти по дереву вниз, открыть элемент, работать с ним и затем переместить его в другую папку. Это не требует интеллектуальных затрат, но раздражает.
- Все примеры по автоматизации используют VB. Предпринимать спелеологический спуск в недра файла справки по автоматизации и распутывать бесспорно редкие примеры — это серьезное испытание, но преобразование синтаксиса из VB в Fox — еще худшая работа. Да, они схожи, но помните, что достаточно одной точки с запятой, чтобы вызвать взрыв космического корабля.

Рис. 1. Первое окно мастера Import/Export программы Outlook.

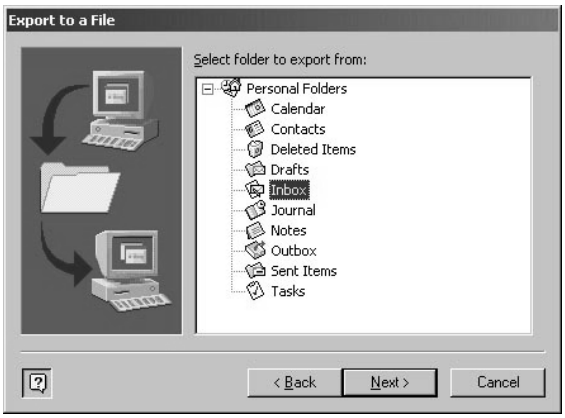

Рис. 2. Выполняя двойные щелчки, опускаемся по дереву до папки, содержащей сообщения, которые вы хотите экспортировать.

По этой причине я решил экспортировать электронные адреса для запросов на удаление из PST в DBF-файл... а уж DBF я и во сне прочитаю. Это не совершенное решение, поскольку функциональность Outlook Export оставляет желать много лучшего. Например, присоединенные документы и даты не могут быть экспортированы. Но это решение может оказаться более быстрым, чем продираться через код автоматизации на VB и продолжать сталкиваться с множеством проблем, экспортируя ключевые элементы (электронный адрес «от кого» и тело сообщения) и затем работая с ними в Fox.

Экспорт DBF из Outlook

Для того, чтобы экспортировать электронное сообщение, откройте Outlook, запустите мастер Import and Export Wizard из меню File | Import/Export и выберите Export to a File, как это показано на рис. 1.

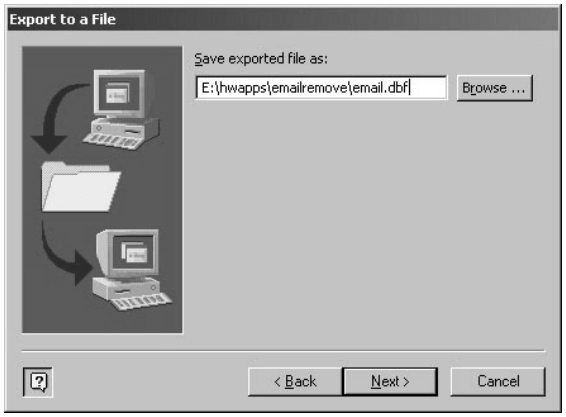

Рис. 3. Определяем DBF-файл для экспортируемых сообщений.

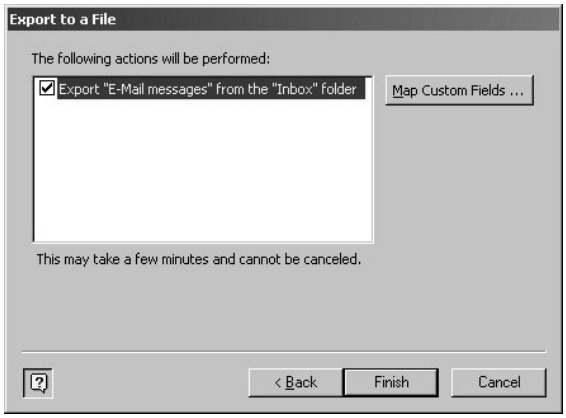

Рис. 4. Выбираем, какое действие вы хотите выполнить.

После этого щелкните Next, спуститесь вниз по списку типов файлов и выберите Microsoft FoxPro.

Затем еще раз щелкните Next, и вы увидите дерево Personal Folders, как это показано на рисунке 2. Двойными щелчками мыши доберитесь до папки, содержащей сообщения, которые вы хотите экспортировать (в моем случае, они находятся в папке E-mail-Remove) и щелкните Next.

Мастер запросит имя файла, в который вы хотите экспортировать ваши сообщения. Используйте поле ввода и/или кнопку Browse для перехода к папке, в которой вы хотите создать DBF, и для ввода имени самого файла DBF (см. рис. 3). Я назвал эту таблицу E-MAILREMOVE.DBF (когда я буду экспортировать все вернувшиеся письма в следующей статье, вы узнаете, что я назвал таблицу для них E-MAILBAD.DBF).

Пришло время немного разочаровать вас. Мастер попросит выбрать, какие действия вы хотите выпол-

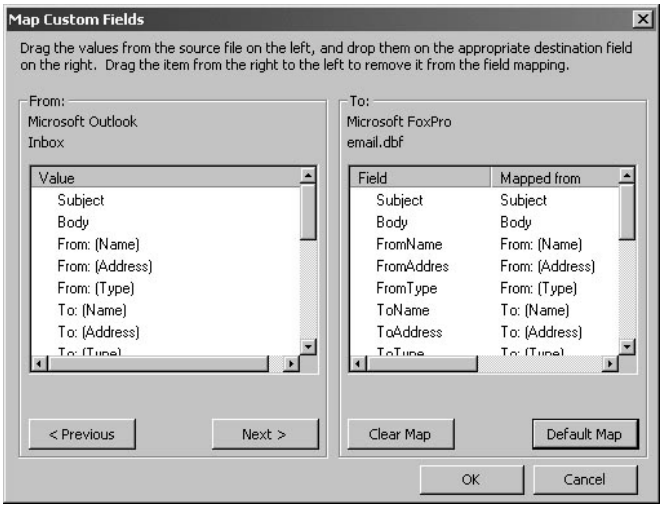

Рис. 5. Кнопка Map Custom Fields пытается убедить вас, что вы можете сопоставить все поля Outlook соответствующим полям в вашем FoxPro DBF-файле.

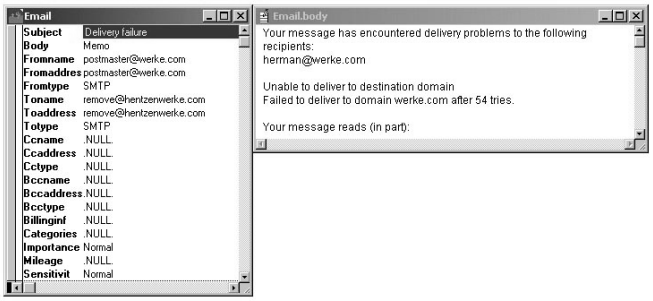

Рис. 6. Типичная запись для электронного сообщения после экспорта.

нить, хотя в списке “The following actions will be performed” (следующие действия будут выполнены) я увидел только один вид действия, как это и отображено на рисунке 4.

Если вы щелкните кнопку Map Custom Fields (установить соответствие для выбранных полей), то получите окно, показанное на рисунке 5. Оно постарается убедить вас поверить, что вы можете сопоставить все поля Outlook соответствующим полям вашего FoxPro DBF-файла, но это не совсем так. Доступные поля в левой стороне диалогового окна являются только подмножеством атрибутов, принадлежащих электронному сообщению. Такие атрибуты, как Date Sent и Attachment, не доступны для экспорта. Соответствующие функциональные возможности отсутствуют и в Outlook 2000, третьем воплощении Outlook.

В результате вы можете просто игнорировать диалоговое окно Map Custom Fields и пойди дальше, щелкнув кнопку Finish для запуска экспорта. Выполнив это, вы получите диалоговое окно хода процесса.

Когда процесс экспорта завершится, вы получите файл DBF со следующей структурой:

```
E-MAILREMOVE.DBF

  1  SUBJECT         Character             210
  2  BODY            Memo                    4
  3  FROMNAME        Character             210
  4  FROMADDRES      Character             210
  5  FROMTYPE        Character             210
  6  TONAME          Character             210
  7  TOADDRESS       Character             210
  8  TOTYPE          Character             210
  9  CCNAME          Character             210
 10  CCADDRESS       Character             210
 11  CCTYPE          Character             210
 12  BCCNAME         Character             210
 13  BCCADDRESS      Character             210
 14  BCCTYPE         Character             210
 15  BILLINGINF      Character             210
 16  CATEGORIES      Character             210
 17  IMPORTANCE      Character             210
 18  MILEAGE         Character             210
 19  SENSITIVIT      Character             210
```

Спросите у меня, почему поля BillingInf и Mileage включены как часть экспортного процесса для электронной почты, а поля, содержащие даты, нет. Я отвечу вам, что не знаю. Достаточно интересно открыть эту таблицу в окне браузера и просмотреть записи. Типичная запись показана на рисунке 6.

Нас интересует всего несколько полей в этой таблице. Поле Body содержит тело электронного сообщения, посланное по адресу remove@hentzenwerke.com. Оно может содержать что-либо, а может и не содержать. Если оно не содержит ничего, то может быть пустым, а может содержать значение .NULL. Поле FromName содержит фактическое имя человека, пославшего электронное сообщение, введенное им в его клиентской программе электронной почты. Для некоторых людей, таких, например, как пользователи AOL, это имя совпадает с электронным адресом. Поле FromAddres является фактическим электронным адресом, с которого было послано сообщение. Обратите внимание, что в имени стоит только одна буква “s”, поскольку это свободная таблица, и имена столбцов ограничены 10 символами.

В наших целях мы хотим использовать два других поля в этой таблице, игнорируя информацию, содержащуюся в них. Мы будем передвигаться по этой таблице и попытаемся найти каждую запись, содержащую электронные адреса из нашей таблицы Customer. Если наша попытка не увенчается успехом, мы попробуем искать такие записи вручную. Мы будем использовать поле FromType для того, чтобы пометить эту запись в E-MAILREMOVE.DBF

как “not found” (не найдено), так, чтобы мы могли вернуться к ней позднее.

Инструкция по удалению в теле сообщения, которое мы рассылаем, включает электронный адрес, на который было послано данное сообщение. Мы извлечем этот адрес из тела сообщения и поместим его в символьное поле, которым проще управлять, когда сравниваешь записи при помощи CUST. Мы используем поле CCADDRESS для хранения действительных электронных адресов – если он есть в поле, значит, он есть. Если тело пусто, мы вставим в CCADDRESS семафор, показывающий, что получатель не последовал инструкции и удалил тело сообщения при составлении ответа.

И в заключение, для достижения целей этой статьи, я собираюсь предположить, что наша таблица заказчиков содержит поля для имени и фамилии (cNaF, cNaL), адреса электронной почты (cE-mail), поле для статуса электронной почты (cStatE-mail), содержащее запись “REMOVE” или “BAD” для тех, кому мы вообще не посылали сообщение, и поле, описывающее почему и когда в это поле было внесено значение “REMOVE” или “BAD” (cWhyBad).

Поиск совпадений и маркировка запросов на удаление

Итак, перейдем к делу.

Первым шагом будет получение «правильных» электронных адресов, помещенных в E-MAILREMOVE, так чтобы их можно было проще сравнивать с полем cE-mail таблицы CUST.DBF. Этот шаг просто замещает содержание поля FROMADDRES его версией в нижнем регистре только для того, чтобы убедиться, что в дальнейшем у нас не будет проблем, связанных с регистром. Второй шаг извлекает адреса электронной почты, по которым были разосланы исходные сообщения, и заполняет ими поля CCADDRESS. Вот используемый код:

```
use E-MAILREMOVE
 replace fromaddres with lower(fromaddres) all
 go top
 scan
  if isnull(e-mailremove.body) or ;
     empty(alltrim(e-mailremove.body))
   * не помечает запись, если она пуста
   replace ccaddress with "No body"
  else
   * где начинается электронный адрес
   m.liStart = at("was sent to", ;
     lower(e-mailremove.body)) + 12
   * проходит символ за символом до тех пор,
   * пока не найдет пробел
   for m.li = m.liStart to m.liStart+100
    if substr(e-mailremove.body, m.li) = " "
     * m.li метит конец адреса
     m.liLast = m.li
     exit
    else
     * продолжает работу
    endif
   next
   m.liLength = m.liLast - m.liStart
   m.lcAddress = substr(e-mailremove.body, ;
     m.liStart, m.liLength)
   replace ccaddress with lower(m.lcAddress)
  endif
 endscan
```

Неплохо, не так ли? Теперь у нас есть два поля в E-MAILREMOVE.DBF, могущие содержать адреса электронной почты, для которых мы ищем совпадения в CUST.DBF.

Сначала мы будем искать в поле CCADDRESS, потому что по этим адресам мы производили рассылку. Если адрес в CCADDRESS существует, мы всегда сможем найти его и в CUST.DBF. Если не сможем, скорее всего это произойдет потому, что CCADDRESS является пустым, потому что получатели наших электронных сообщений не включали тело сообщения в свои запросы на удаление.

Поэтому приведенный ниже отрывок кода движется через каждую запись в E-MAILREMOVE, ища содержание CCADDRESS в поле cE-mail таблицы CUST.DBF. Если данные из CCADDRESS будут найдены в CUST, тогда в поле FromType в E-MAILREMOVE будет помещено значение “FoundBodyInCust”, в поле CUST.cStatE-mail – значение “NEVER”, а в cWhyBad – следующее объяснение, почему и когда был установлен флажок NEVER: “RequestedRemove CCYYMMDD”.

```
* двигается через E-MAILREMOVE и пытается найти совпадения
* CCADDRESS с CUST.ce-mail
m.liNumRec = reccount()
scan
  * Сначала ищет адреса в теле сообщения,
  * поскольку это те адреса, на которые мы посылали письма
  if "@" $ ccaddress
   wait wind nowait "Looking for REMOVE e-mail in CUST " ;
   + transform(m.liNumRec - recno())
   m.lcE-mail = upper(alltrim(e-mailremove.ccaddress))
   * создает строку, поясняющую, почему этот адрес
   * был помечен флажком
   m.lcWhyBad = "RequestedRemove " + + dtos(date()) + ;
     "_" + strtran(time(), ":", "")
   select CUST

   if seek( m.lcE-mail )
    * если мы нашли электронный адрес, помечаем его флажком
    if empty(cWhyBad)
     replace cWhyBad with m.lcWhyBad
     replace cstate-mail with "NEVER"
     select E-MAILREMOVE
     replace FromType with "FoundBodyInCust"
    else
     m.lcWhyBadOld = CUST.cWhyBad
     replace cWhyBad with alltrim(alltrim(m.lcWhyBad) ;
       + " " + alltrim(m.lcWhyBadOld))
     replace cstate-mail with "NEVER"
     select E-MAILREMOVE
     replace FromType with "FoundBodyInCust"
    endif && empty(cWhyBad)
   else
    * мы не нашли этот электронный адрес в CUST,
    * поэтому помечаем его в
    * E-MAILREMOVE.DBF специальным указателем
    select E-MAILREMOVE
    replace FromType with "NotFoundBodyInCust"
   endif && seek (m.lcE-mail)
  else
```

```
   * в ccaddress нет электронного адреса (поскольку тело
   * было пустым или содержало значение null)
   select E-MAILREMOVE
   replace FromType with "NotFoundBodyInCust"
  endif && "@"$ccaddress
endscan
```

Этот FoxPro-код работает достаточно тривиально, но это более бизнес-логика, чем изощренное программирование, которое больше интересует меня. (Я не проверял, но бьюсь об заклад, что этот код будет работать в FoxPro 2.0.).

Теперь представим себе, что мы не можем найти CCADDRESS в CUST.cE-mail. Помните, что получатель нашего электронного сообщения мог удалить тело сообщения перед отправкой, поэтому все, что у нас есть, — это значение FromAddres, которое может быть (а может и не быть) адресом, входившим в массовую рассылку. Приведенная ниже порция кода будет обрабатывать эту ситуацию, за исключением того, что мы хотим использовать FromAddres только в том случае, если не нашли совпадения значению CCADDRESS, и эти два значения различны.

```
* если вы не нашли ccaddress, и
* fromaddres отличается от ccaddress,
* то ищем fromaddres
 select E-MAILREMOVE
 go top
 scan
  wait window nowait "Looking for REMOVE e-mail in CUST " ;
  + transform(m.liNumRec - recno())
  if FromType <> "Found" and ;
     alltrim(e-mailremove.fromaddres) <> ;
     alltrim(e-mailremove.ccaddress)
   m.lcE-mail = upper(alltrim(e-mailremove.fromaddres))
   m.lcWhyBad = "RequestedRemove " + + dtos(date()) + ;
     "_" + strtran(time(), ":", "")
   select CUST
   if seek( m.lcE-mail )
    * если мы нашли электронный адрес, помечаем его
    if empty(cWhyBad)
     replace cWhyBad with m.lcWhyBad
     replace cstate-mail with "NEVER"
     select E-MAILREMOVE
     replace FromType with "FoundFromInCust"
    else
     m.lcWhyBadOld = CUST.cWhyBad
     replace cWhyBad with alltrim(alltrim(m.lcWhyBad) ;
       + " " + alltrim(m.lcWhyBadOld))
     replace cstate-mail with "NEVER"
     select E-MAILREMOVE
     replace FromType with "FoundFromInCust"
    endif && empty(cWhyBad)
   else
    * мы не нашли электронный адрес в CUST,
    * поэтому помечаем его в
    * E-MAILREMOVE.DBF специальным указателем
    select E-MAILREMOVE
    replace FromType with alltrim(FromType) ;
      + iif(!empty(FromType), "", "") ;
      + "NotFoundFromInCust"
   endif && seek (m.lcE-mail)
  endif && alltrim(e-mailremove.fromaddres) <>
 endscan
 wait clear
```

В моих тестах эти два прохода через базу данных E-MAILREMOVE находили большинство из запросов на удаление – обычно более 90 процентов. Но остаются еще 10 процентов, которые предстоит обрабатывать вручную. И они должны быть обработаны, потому что эти клиенты могут просто не понимать, что они не последовали инструкции. Кроме того, они могли пренебречь инструкцией не по злому умыслу, а просто не понимая, что нарушают свои собственные интересы. Думаю, они расстроятся, если будут по-прежнему получать электронные сообщения.

Итак, пришло время засучить рукава и выполнить оставшуюся работу вручную. Прежде всего, настройте фильтр на E-MAILREMOVE так, чтобы вы видели только записи, для которых FromType = "NotFound". Далее используйте значение, которое выглядит как фамилия человека в поле FromName для поиска в CUST – вы зачастую обнаружите, что электронный адрес позволяет с достаточной уверенностью утверждать, что он принадлежит именно этой персоне. И в заключение, если останутся те, кого не удастся обнаружить этим способом, попробуйте копать через доменное имя. Например, если вы ищете john_doe@mycompany.com, но не можете найти “DOE” в поле для фамилии или в электронном адресе, попытайтесь поискать в домене mycompany. Я часто находил записи следующего вида: john_l_doe@mycompany.com или jdoe@e-mail.mycompany.com.

Простое следование этим двум шагам позволит позаботиться о подавляющем большинстве оставшихся запросов на удаление. Как же быть с остальными? Что делать, они просто будут получать электронные письма снова и снова, пока не последуют инструкции должным образом.

И в заключение, скопируйте E-MAILREMOVE в резервный каталог базы данных. В случае, если к вам явится разгневанная персона, выражающая недовольство по поводу того, что вы не удалили его (или ее) адрес, вы сможете выловить адрес в ваших данных и покажете, что все возможное было сделано. Большинство людей, независимо от того, насколько они были недовольны первоначально, увидев, что вы предприняли героические усилия для их поиска, успокоятся. А как поступить с теми несколькими, которых не удовлетворит и это? Что ж, среди массы людей всегда найдется и такой, но не стоит терять из-за него сон.

Еще одно достоинство такого подхода заключается в том, что при последующих массовых рассылках число возвращающихся назад запросов на удаление будет уменьшаться. Уверен, они будут появляться, но после нескольких рассылок основная масса людей, которые не хотят получать от вас сообщения, ответит вам, и вы пометите их адреса с тем, чтобы они не получали больше электронную почту от вас.

❑❑❑

Найти меня…

Энди Крамек и Марсиа Акинс (Andy Kramek and Marcia Akins)

WIN 7.0 DOWNLOAD

Одной из наиболее распространенных задач, с которыми мы имеем дело, является поиск определенной записи в таблице, курсоре или представлении. Несмотря на то что эта задача, на первый взгляд, не имеет простого решения, Энди Крамек и Марсиа Акинс нашли, после некоторого размышления, что любое текстовое поле на любой форме может быть использовано как базовый компонент для поиска. Как всегда, ключевая проблема заключается в разработке класса, выполняющего эту работу.

Энди: Знаешь что? Я на самом деле сыт по горло написанием кода для выполнения поиска. Мне кажется, что я провожу половину моего рабочего времени, тестируя и выискивая ошибки в коде, позволяющем пользователю ввести значение и затем сравнивающим его с полем, расположенным в некоторой записи.

Марсиа: Понимаю, что ты имеешь в виду. После того, как я три раза подряд написала такой или, по меньшей мере, очень похожий код, я пришла к идее, что, возможно, я должна абстрагировать его и создать класс.

Энди: Веришь ли, но я также хотел создать класс для этой задачи, но проблема заключается в том, что каждый поиск сугубо индивидуален. Я имею в виду, что иногда у вас есть индекс на поле (и тогда имеет смысл использовать функцию SEEK()); в другой раз вы не можете использовать индекс, поэтому вам необходимо выполнить команду LOCATE. Иногда вам необходимо выполнить действие “Find Next”. Кроме того, у вас возникают проблемы, связанные с фактическим определением того, какое поле искать и, конечно, с определением самой таблицы. Поэтому не похоже, что здесь есть какой-либо практический путь для абстрагирования.

Марсиа: Да, но если ты будешь придерживаться моего стандарта, всегда именуя индексный тег на поле точно также, как и само поле, то проблема определения наличия у поля связанного индекса разрешается достаточно просто.

Энди: Но ты не сможешь сделать этого, если будешь использовать длинные имена поля, поскольку существует 10-символьное ограничение на имена тегов. И что тогда?

Марсиа: Я никогда не использую длинные имена для полей, которые собираюсь индексировать. Это не так трудно осуществить, ты знаешь, особенно если ты предварительно проектируешь свои таблицы.

Энди: Ну хорошо. Итак, если мы примем твое правило, то сможем проверить наличие тега, имеющего имя поля. Но как мы узнаем в каком поле осуществлять поиск, если только это не жестко запрограммировано?

Марсиа: Давай остановимся на этом. И вникнем в детали. На самом деле, мы еще не определили проблему или не установили ответственности для этого класса. Ты же знаешь, нужно самому практиковаться в том, чему учишь других.

Энди: Ты права! Давай вернемся на ступеньку назад и попытаемся определить проблему в общих терминах. То, о чем мы говорим, является формой, имеющей один или несколько редактируемых элементов (это либо поле ввода (text box), либо поле редактирования (edit box)), связанных с источником данных. Мы хотим, чтобы пользователь мог установить фокус на любом из этих элементов, а затем щелкнуть кнопку Search, которая позволит ему определить значение для поиска в любом поле, какое бы он ни выбрал.

Марсиа: Кхм... Звучит так, что нам необходима всплывающая форма, имитирующая поведение собственного диалогового окна Find в VFP. Думаю, будет нетрудно получить те параметры, что ты обрисовал.

Энди: Я рад это услышать. Итак, с чего ты хочешь начать?

Марсиа: Первая проблема, с которой я столкнулась, заключается в том, что нам нужно переслать детали таблицы и поля в поисковую форму. Поскольку мы имеем дело только со связанными элементами, мы можем получить эту информацию из свойства ControlSource. Закавыка состоит в том, что когда вы щелкаете кнопку Search, она немедленно становится активным элементом управления, поэтому на основ-

ной форме нам необходим некоторый способ отслеживания, какой из «способных искать» элементов последним находился в фокусе.

Энди: Ага! Таким образом, теперь нам нужно два класса: всплывающая форма и новая «способная искать» основная форма.

Марсиа: Я предупреждала, что будет довольно трудно поместить это все в один класс. На самом деле, всплывающая форма может быть просто SCX; и нам необходим только класс форм и новый класс полей ввода, который может регистрировать себя в качестве активного элемента всякий раз, когда он получает фокус.

Энди: Я вижу, куда ты клонишь. Форма должна иметь свойство, хранящее ссылку на последний «способный искать» элемент, получивший фокус.

Марсиа: На самом деле, будет лучше иметь класс «способных искать» полей ввода, устанавливающих это свойство при получении фокуса, и любые поля ввода, не способные выполнять поиск, сбрасывающие это свойство (предполагая, что оно присутствует на этой форме). Это должно выполняться на достаточно высоком уровне иерархии классов, может быть даже в нашем корневом классе.

Энди: Нет. Думаю, это классический пример подкласса. Для окна ввода необходимо свойство (lSearchable), имеющее значение FALSE по умолчанию (поскольку только определенные элементы должны быть «способными искать»). Если оно имеет значение TRUE, то будет искать соответствующее свойство формы. Приведенный ниже код в GOTFOCUS() должен выполнять эти задачи:

```
*** Обладает ли форма необходимым свойством?
IF PEMSTATUS( ThisForm, 'oSearchControl', 5 )
  *** Может ли это поле ввода быть использовано
  *** в качестве источника для поиска?
  IF This.lSearchable
    *** Устанавливает это свойство для данного элемента
    ThisForm.oSearchControl = This
  ELSE
    *** Устанавливает для свойства значение NULL
    ThisForm.oSearchControl = NULL
  ENDIF
ENDIF
```

Марсиа: Да, на мой взгляд это нормально. Выполняя эти действия окружным путем, мы получаем уверенность, что даже если поле ввода помечено как «способное искать», но не находится на «способной искать» форме, ничего не изменится. Знаешь, думаю это можно сделать и в корневом классе.

Энди: Единственная загвоздка состоит в том, что мы вынуждены выполнять вызов PEMSTATUS() каждый раз, когда одно из полей ввода получает фокус. Это не является обязательным поведением для стандартного поля ввода, поэтому его не стоит помещать в корневой класс. Если бы мы первым делом проверяли это свойство поля ввода, то смогли бы избежать дополнительной работы и, возможно, корневой класс был бы уместен. Однако, как ты указала, это может также просто испортить все дело. В целом, думаю, специализированный подкласс является лучшим решением. А теперь, что мы можем сказать о классе основной формы?

Марсиа: Итак, класс формы должен иметь свойство oSearchControl и кнопку Search, вызывающую метод Search(). Мы можем применить метод Assign для этого свойства, чтобы контролировать включение и блокировку кнопки Search, следующим способом:

```
LPARAMETERS tuSource
 WITH This
   *** Обновляет свойство
   .oSearchControl = tuSource
   *** Настраивает кнопку Search соответствующим образом
   .cmdSearch.Enabled = NOT ISNULL( .oSearchControl )
 ENDWITH
```

Энди: Теперь я вижу, почему ты хотела, чтобы кнопка Search являлась частью класса формы. Мы используем метод Assign для контроля готовности этой кнопки! Теперь все, что нам нужно, это код в самом методе Search. По-видимому, именно он вызывает всплывающую поисковую форму?

Марсиа: Да, он может использовать свойство формы oSearchControl для получения подробностей о свойстве ControlSource и передачи их в качестве параметров во всплывающую форму:

```
WITH ThisForm
* Определяет детали таблицы и поля, используя
* свойство ControlSource «способного искать» элемента
lcSource = .GetParameters()
lcTable = JUSTSTEM( lcSource )
lcField = JUSTEXT( lcSource )
* Вызывает поисковую форму (она может уже существовать)
IF VARTYPE( .oSchForm ) # "O"
  * Запускает эту форму и устанавливает свойство
  DO FORM frmsearch NAME junk LINKED
  .oSchForm = junk
ELSE
  * Активирует поисковую форму
  .oSchForm.Show()
ENDIF
* Передает параметры для поиска
.oSchForm.SetParams( lcTable, lcField )
ENDWITH
```

Энди: Я заметил, что ты предполагаешь, что описание источника данных всегда выполнено в виде Table.Field. Обоснованно ли это?

Рис. 1. Общая поисковая форма.

Марсиа: Да, ты сказал, что мы имели дело только со связанными элементами, и я в самом деле не вижу, почему мы должны беспокоиться о чем-либо еще (об условном control source или о том, что какой-то элемент связан со свойством формы). Кроме всего, это нормальный способ для связывания элементов. Также мы предусматриваем другие возможности, абстрагируя это поколение параметров поиска в их собственный метод. Если вам нужен другой механизм, просто используйте подкласс и переделайте метод GetParameters().

Энди: Это выглядит достаточно обоснованно. Между прочим, я обратил внимание, что ты оставила форму как SCX. Были ли какие-то особенные причины поступить так?

Марсиа: На самом деле нет, но, поступая так, мы можем показать еще один способ использования SCX, но продолжать работать с ним, как с объектом. Кроме того, люди склонны забывать, что SCX является только особой формой VCX, и DO FORM является особым случаем CREATEOBJECT().

Энди: Так вот почему ты использовала конструкцию LINKED команды DO FORM. Я был удивлен, так как используемое тобой фактическое имя (junk) выходит из области видимости сразу после завершения этого метода, но, поскольку ты назначила имя свойству вызываемой формы, сама поисковая форма не исчезает до тех пор, пока это свойство не освободится. Это достаточно хитрый путь для управления «сборкой мусора». Теперь давай посмотрим на действующую поисковую форму (см. рис. 1).

Марсиа: Первое, что нам потребуется, это ссылка на родительскую форму, поскольку она может обновить форму после успешного поиска. Мы можем извлечь эту ссылку в функции LOAD() формы следующим образом:

```
*** Извлекаем ссылку на родительскую форму
This.oParent = IIF( VARTYPE( _Screen.ActiveForm ) ;
  = "O", _Screen.ActiveForm, NULL )
```

Энди: Вот еще одна полезная подсказка. Это работает потому, что когда создается новая форма, свойство _Screen.ActiveForm не обновляется до тех пор, пока метод Init() новой формы не будет завершен. Вы можете использовать это обстоятельство, когда одна форма вызывает другую, чтобы получать ссылку на вызывающую форму без необходимости явного перехода к ней.

Марсиа: Хорошо, вот фрагмент кода для метода SetParams() поисковой формы, устанавливающий свойства, используемые для поиска, и добавляющий имя целевого поля в заголовок формы:

```
LPARAMETERS tcAlias, tcField
WITH ThisForm
  *** Сохраняет параметры в свойствах формы
  .cAlias = IIF( EMPTY( tcAlias ), "", tcAlias )
  .cField = IIF( EMPTY( tcField ), "", tcField )

  *** Сохраняет номер текущей записи в случае,
  *** если поиск провалился
  .nRecNo = IIF( EMPTY( tcAlias ), 0, RECNO( tcAlias ))

  *** Устанавливает значение для заголовка формы
  .Caption = "Search for: " + ALLTRIM( tcField )
ENDWITH
```

Обрати внимание, если ничего не передается, то значения этих свойств будут содержать пустые строки. Вот почему когда мы связываем таблицу и имя поля, мы не получаем сообщения об ошибке. Отсюда, мы можем использовать функцию TYPE(), чтобы убедиться, что целевое поле на самом деле существует, прежде чем будем выполнять в нем поиск.

Энди: Это выглядит убедительно. Я обратил внимание, что на нашей форме есть две кнопки — Find и Find Next, но обе они вызывают один и тот же метод Find() на форме.

Марсиа: Да, хотя заметь, кнопка Find Next на самом деле пересылает параметр “NEXT”, в то время как кнопка Find не пересылает ничего. Если ты остановишься и подумаешь об этом, то поймешь, что код, необходимый для функционирования Find и Find Next, отличается только диапазоном поиска. Поэтому наш код может выглядеть так:

```
LPARAMETERS tcNext
LOCAL lnSelect, lcField, luValue

WITH ThisForm
*** Проверяет, что источник поиска определен корректно
IF TYPE( .cAlias + "." + .cField ) = "U"
  *** У нас нет допустимого источника,
  *** поэтому просто отбрасываем его.
  *** Может также отобразить сообщение здесь!
```

```
  RETURN
ENDIF

*** Сохраняет текущую рабочую область
lnSelect = SELECT()

*** Получает значение, которое мы собираемся искать,
*** из поля ввода
luValue = IIF( VARTYPE( .txtSearchString.Value ) = 'C', ;
  ALLTRIM( .txtSearchString.Value ), ;
  .txtSearchString.Value )

*** Определяет диапазон поиска
*** и устанавливает начальное положение
IF EMPTY( tcNext )
  *** Если мы находим первое совпадение,
  *** то возвращаемся в начало файла
  GO TOP
ELSE
  *** Если мы находим следующее совпадение,
  *** то должны перейти к следующей записи
  SKIP
  IF EOF()
    GO BOTTOM
  ENDIF
ENDIF

*** Получаем имя поля из свойства
lcField = .cField

*** Просто используем команду LOCATE REST
*** для нахождения точного совпадения
IF VARTYPE( EVAL( lcField ) ) = 'C'
  LOCATE FOR UPPER( &lcField ) =  luValue REST
ELSE
  LOCATE FOR &lcField  =  luValue REST
ENDIF

*** Если совпадение найдено
IF FOUND()
  *** Сохранить номер для совпавшей записи
  .nRecNo = RECNO( .cAlias )
  *** Обновляем родительскую форму
  .oParent.Refresh()
ELSE
  WAIT WINDOW 'No Match Found!' NOWAIT
  *** Восстанавливаем указатель на запись
  GOTO .nRecNo
ENDIF

*** И восстанавливаем рабочую область
SELECT ( lnSelect )
ENDWITH
```

Энди: Неплохо! Я и забыл, что мы можем использовать предложение REST с командой LOCATE и, раз уж мы все равно используем LOCATE, это, несомненно, упрощает код, и нам даже не приходится беспокоиться об индексах. Следует заметить, что мы должны сначала явно переместить указатель записи на следующую запись, а потом использовать предложение REST. Это происходит потому, что диапазон, определяющий поиск, начинается с текущей записи. Если эта запись совпадет с искомой, ничего не произойдет (в этом отличие от использования команды CONTINUE).

Марсиа: Точно! Конечно, команда LOCATE будет использовать индекс (если таковой доступен), и хотя первое совпадение находит не так быстро, как SEEK, она позволит нам избежать целого круга потенциальных проблем.

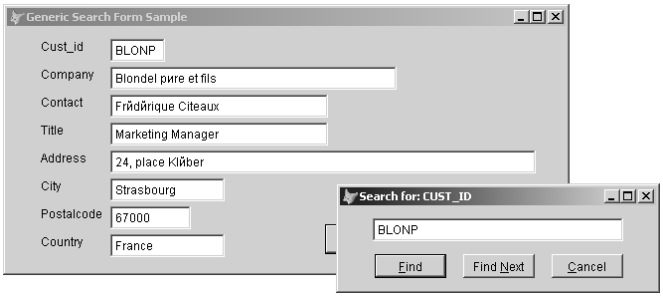

Рис. 2. Пример поисковой формы.

Энди: Положительной стороной этого подхода является то, что он также применим в случаях, когда отсутствует индекс! Мне это нравится, это жизненное решение проблемы, которое, я думаю, удержит меня от дальнейших попыток создания общей поисковой функциональности. На сопровождающей дискете вы найдете пример формы (schdemo), использующей классы, и поисковые формы, описанные в этой статье (см. рис. 2).

Энди Крамек (Andy Kramek) — опытный FoxPro-разработчик со стажем, имеет статус FoxPro MVP, является независимым контрактором и время от времени выступает как автор книг и статей. Родом он из Англии, но в настоящее время проживает в Акроне, шт. Огайо.
andykr@compuserve.com

Марсиа Акинз (Marcia Akins) имеет статус FoxPro MVP, является независимым консультантом и совладельцем фирмы Tightline Computers Inc., находящейся в Акроне, шт. Огайо. «Заслуженный» спикер многих конференций, она часто публикует свои работы и хорошо известна как активный участник форумов CompuServe и Universal Thread.
marciagakins@compuserve.com

❑❑❑

Подсказка в форме выноски

Предраг Боснич (Predrag Bosnic)

WIN 7.0 DOWNLOAD

После того, как Предраг Боснич создал в прошлом месяце элемент Multi-Line ToolTip (многострочная подсказка), большинство скажет, что это только один из возможных вариантов, и преобразовать эту многострочную подсказку в элемент «подсказка в форме выноски» (Balloon ToolTip) довольно просто. Почему же, в таком случае, совершенно новая статья? Очень просто. Хотя в ней Предраг будет использовать многие из основных принципов, примененные им для многострочной подсказки, но все же не стоит забывать, что он делает нечто, не совсем обычное для Visual FoxPro. На этот раз он вводит функции Windows API и покажет, как они могут помочь, когда эта помощь необходима.

Прежде всего, давайте вспомним, где можно найти элемент «подсказка в форме выноски» (Balloon Tooltip). Для написания статей я использую программу MS Word, в которой меню AutoShape/Callouts содержит несколько различных стилей для выносок. На рис. 1 показаны эти выноски, и я хочу отметить, что каждая из них чем-то привлекательна.

Вторым приложением, используемым мною, является MS Visio, но оно использует подсказки в форме выноски для отображения текста подсказок. На рис. 2 показана реализация элемента «подсказка в форме выноски» в MS Visio, и я хочу отметить, что она мне нравится, она очень элегантна. Кроме того, этот элемент может отображать заголовок и падающую тень.

В данном случае я выбрал упрощенную версию элемента MS Visio. Заголовок и падающую тень оставим до будущих времен.

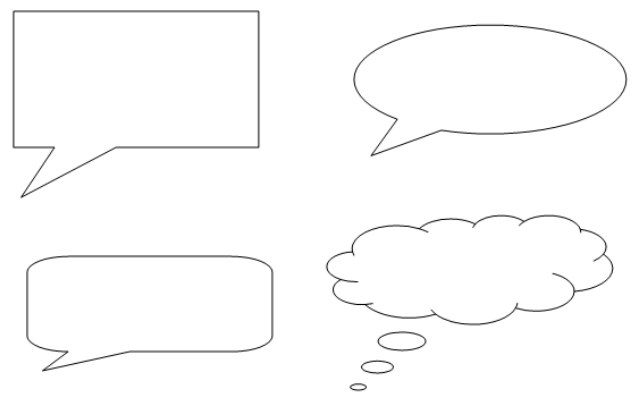

Рис. 1. Выноски MS Word.

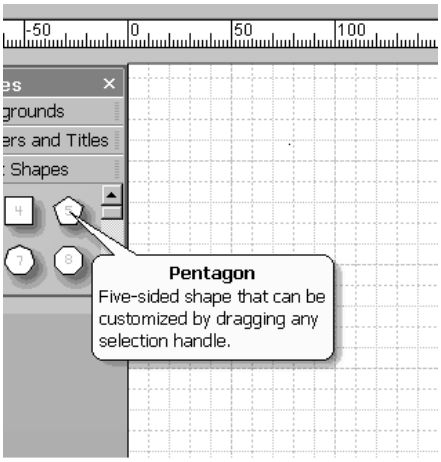

Рис. 2. Элемент интерфейса MS Visio «подсказка в форме выноски».

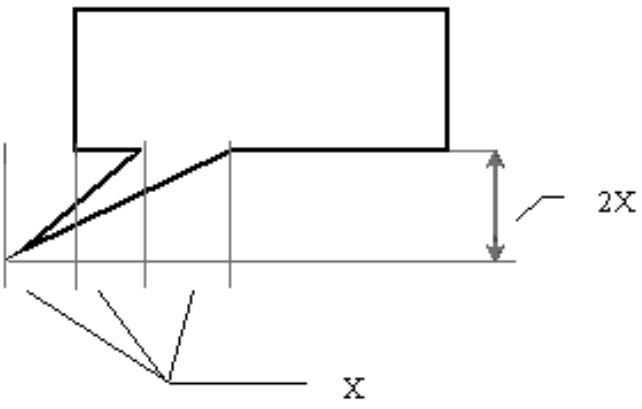

Рис. 3. Основные пропорции для элемента «подсказка».

На рис. 3 показан дизайн, которого я хотел бы добиться, и основные пропорции, которых я придерживаюсь. Изменить значение “X” возможно в самом коде. Максимальная ширина управляющего элемента составляет около 363 пикселей, и это достаточно для размещения около 60 символов. Существует че-

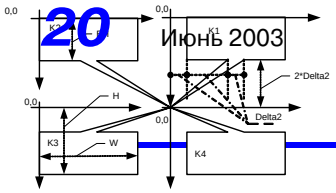

Рис. 4. Четыре возможные ориентации для элемента «подсказка в форме выноски».

тыре возможные ориентации, которые может иметь подсказка (см. рис. 4). Которая из них будет использована, зависит от положения мыши на экране. Положение по умолчанию — K1.

Требования к дизайну

Пожалуйста, посмотрите требования для многострочной подсказки в моей предыдущей статье.

Замысел и реализация

Для решения этой задачи я использовал функции Windows API. Причина этого очень проста: я не знаю, как это можно сделать при помощи собственных элементов управления Visual FoxPro 7. По моему мнению, это хороший пример того, когда нужно использовать функции Windows API. Когда язык программирования, используемый мною, не может помочь разрешить проблему (или, по крайней мере, не существует простого ее решения), можно использовать функции Windows API. Конечно, этот язык программирования должен уметь вызывать Windows API и передавать соответствующие параметры. Visual FoxPro не является идеальным языком для вызова Windows API, но с каждой новой версией он делает это все лучше и лучше. В версии Visual FoxPro 7 вводится hWnd (windows handle) как свойство для элементов управления формами. Указатель окна используется в сотнях функций Windows API. Также был улучшен оператор DECLARE и, если я добавлю к сказанному несколько статей об использовании Windows API в Visual FoxPro (например, статьи Кристофа Лэнга (Christof Lange) из общедоступной сети и статьи Дуга Хеннига (Doug Hennig) из FoxTalk), с таким набором можно сделать многое.

Я создал библиотеку классов wbToolTipB, которая будет служить контейнером для моих классов. Эта библиотека содержит два класса, wbToolTipB и ttScr2. На этот раз я сначала опишу класс wbToolTipB. Этот класс основывается на элементе управления Timer и имеет несколько определенных пользователями свойств.

- *oObjRef* — ссылка на объект (на экземпляр, кнопку, раскрывающийся список и т. д.)
- *ToolTipForeColor* — подсказка для основного цвета.
- *ToolTipBackColor* — подсказка для цвета фона.

Метод Init содержит только объявления для функций Windows API.

```
Declare Integer CreateRectRgn in "gdi32"
Declare Integer CreateRoundRectRgn in "gdi32"
 ...
```

Стоит упомянуть, что этот класс будет в форме, использующей элемент «подсказка в форме выноски», поэтому применение его метода Init для объявления функции Windows API является корректным. Метод Init выполняется только один раз, и пользователь не должен заботиться об объявлении необходимых функций Windows API. Я дам вам описание использования некоторых функций API, но хочу сказать, что я не использую все из объявленных функций. Хочу посоветовать вам почитать о следующих функциях Windows API: CreateRectRgn, CreateRoundRectRgn, CreateEllipticRgn, CreatePolygonRgn, FillRgn, PaintRgn, CombineRgn, SetWindowRgn, DeleteObject, ReleaseCapture, GetDC, ReleaseDC, CreateSolidBrush, Polygon, SelectObject, CreateFontIndirect, TextOut, SetTextColor, GetTextColor и FrameRgn.

По-настоящему важным является только метод Timer, и вот его код:

```
* Событие Timer
 LOCAL joObject, jnLeft, jnTop, jcToolTip, joX

 joX = SYS(1270)
 IF TYPE('joX') <> 'O' or ISNULL(joX)
   thisform.wb_LIsToolTipActive = .f.
   thisform.KillToolTip()
   this.Enabled = .f.
   RETURN
 ENDIF
 *
 IF joX # this.oObjRef
   thisform.wb_LIsToolTipActive = .f.
   thisform.KillToolTip()
   this.Enabled = .f.
   return
 endif
 *---
 IF thisform.wb_LIsToolTipActive = .t.
   RETURN .f.
 endif
```

```
*
joObject = this.oObjRef
jcToolTip = ALLTRIM(joObject.wb_cToolTipText)
IF EMPTY(jcToolTip)
  this.Enabled = .f.
  return
ENDIF
*
jnLeft = Mcol('',3)
jnTop  = Mrow('',3)
thisform.wb_oToolTip = CREATEOBJECT('ttscr2',thisform,;
  thisform.wb_nToolTipType,jcToolTip,jnLeft,;
  jnTop,this.ToolTipForeColor,this.ToolTipBackColor)
thisform.wb_LIsToolTipActive = .t.
Thisform.wb_oToolTip.Visible = .T.
thisform.wb_oToolTip.Show()
*------------------------
*** Не убирайте таймер отсюда.
*** Он должен быть активным
*** в ходе всего сеанса подсказки!!!
```

Запомните, что таймер активен только в ходе сеанса подсказки, т. е. в то время, пока ее видно. Когда метод Timer запущен, он определяет положение указателя мыши.

Код в первой команде IF деактивирует подсказку и затем сам деактивируется, если указатель мыши не находится над объектом.

Вторая команда IF изучает объект, находящийся под указателем мыши. Если это не тот же объект, который активировал подсказку (когда подсказка активирована, код устанавливает свойство oObjRef и использует его), код деактивирует его, поскольку пользователь переместил указатель мыши с элемента управления, создающего подсказку.

Третий оператор IF проверяет свойство wb_LisToolTipActive. Если подсказка уже активна, он просто возвращает управление из кода события Timer.

Четвертая команда IF проверяет текст подсказки. Если подсказка пуста, IF возвращает управление из кода события Timer. Если достигнут этот блок кода, я знаю, что подсказка неактивна и код должен ее активировать. Теперь я изучаю положение курсора мыши, вызываю код подсказки и устанавливаю флажок:

```
wb_lIsToolTipActive = .T.
```

Пришло время создать определение класса для подсказок — ttScr2.

Как можно видеть на рис. 5, этот класс содержит те же элементы управления, что и класс ttScr1, используемый для элементов «многострочная подсказка». Однако метод Init для него будет другим.

```
LPARAMETERS toParentForm, ;
            tnToolTipType, ;
            tcToolTipText, ;
            tnLeft, ;
            tnTop, ;
            tnForeColor, ;
            tnBackColor
```

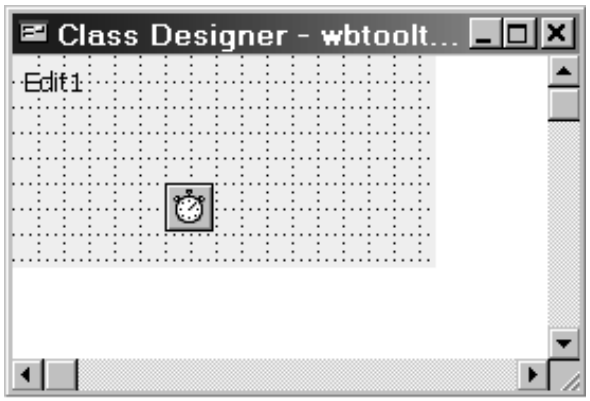

Рис. 5. Класс wbToolTipB в Class Designer.

```
* toParentForm - ссылка на родительскую форму
* tnToolTipType - тип подсказки, 1 = прямоугольник
* tcToolTipText - текст подсказки
* tnLeft - x-координата для указателя мыши
* tnTop  - y-координата для указателя мыши
* tnForeColor - передний цвет
* tnBackColor - задний цвет
*--------------------------------------------

WITH thisform
  .oParentForm = toParentForm
  .nObjCenterX = tnLeft
  .nObjCenterY = tnTop
  .comment = ALLTRIM(tcToolTipText)
  .Edit1.Value = .comment
  .ToolTipType = tnToolTipType
  IF !EMPTY(tnForeColor)
    .Edit1.ForeColor = tnForeColor
  ENDIF

  IF !EMPTY(tnBackColor)
    .BackColor = tnBackColor
  endif
ENDWITH

WITH thisform
  .Left = tnLeft
  .Top = tnTop
  .Skin(this)
ENDWITH
```

Класс подсказок ttScr2 принимает несколько параметров, устанавливает цвет подсказки и вызывает метод Skin. После чего появляется подсказка. Если вы сравните этот код с кодом для многострочной подсказки, то можете заметить разницу в методе Skin.

```
* ttScr2, метод Skin
 LPARAMETERS tnWidth, tnHeight

 * tnWidth - необязательный, ширина подсказки
 * tnHeight - необязательный, высота подсказки

 #DEFINE RGN_AND 1
 #DEFINE RGN_OR 2
 #DEFINE RGN_XOR 3
 #DEFINE RGN_DIFF 4
 #DEFINE RGN_COPY 5
```

```
*
LOCAL W, H, NoOfLines, jaDots, EH, K, strDots,;
  Delta, jnX, jnY, xMin, xMax
LOCAL yMin, yMax, nSkinRgn, jnDots, DeltaCut

*-- Устанавливает для ширины значение по умолчанию ---
IF EMPTY(tnWidth) or tnWidth < 363
  W = 363
ELSE
  W = tnWidth
ENDIF
*
NoOfLines=1
DO CASE
  CASE thisform.ToolTipType = 1
    NoOfLines = Int(LEN(toForm.comment)/60) + 1
    EH = 21 + 15*(NoOfLines - 1)
    Delta = 10
    Delta2= 2*Delta
    thisform.Edit1.Height = EH - 4
    *-------------------------------------
    H = EH + Delta2
    IF NoOfLines = 1
      W = thisform.textwidth(thisform.Comment)+25
    ELSE
      W = W + Delta
    ENDIF
    WITH thisform
      .width  = W
      .Height = H
      .Edit1.Width = .width-6-Delta-8
      .Edit1.Left = Delta + 3
    ENDWITH
  OTHERWISE
    *
ENDCASE

*--------------------------------
IF NoOfLines = 1
  thisform.Edit1.Alignment = 0
ENDIF
*--------------------------------
jnX = thisform.nObjCenterX
jnY = thisform.nObjCenterY

Xmin = W
Xmax = _screen.Width-W
Ymin = H
Ymax = _screen.Height - H
*
Xscr = _screen.Width
Yscr = _screen.Height

*              I
*       K2     I       K1
*              I
*  ------------------------------
*       K3     I       K4
*              I
*              I
*
DO case
  CASE jnX > Xmax and jnY < Yscr/2
    K=3
    WITH thisform
      .Left = jnX - W
      .Top = jnY + Delta2
      .Edit1.Top = H - EH + 2
    ENDWITH
    thisform.edit1.Left = 3

  CASE jnX > Xscr/2 and jnX < Xmax and jnY < Ymin
    K=3
    WITH thisform
      .Left = jnX - W
      .Top  = jnY + Delta2
      .Edit1.Top = H - EH + 2
    ENDWITH
    thisform.edit1.Left = 3

  CASE jnX > Xmin and jnX < Xscr/2 and jnY < Ymin
    K=4
    WITH thisform
      .Left = jnX
      .Top  = jnY + Delta2
      .Edit1.Top = H - EH + 2
    ENDWITH
    thisform.Edit1.Width = thisform.width-6-Delta

  CASE jnX < Xmin and jnY < Yscr/2
    K=4
    WITH thisform
      .Left = jnX
      .Top  = jnY + Delta2
      .Edit1.Top = H - EH + 2
    ENDWITH
    thisform.Edit1.Width = thisform.width-6-Delta

  CASE jnX < Xmin and jnY > Yscr/2
    K=1
    thisform.Left = jnX
    thisform.Top  = jnY -H - Delta2
    thisform.Edit1.Width = thisform.width-6-Delta

  CASE jnX > Xmin and jnX < Xscr/2 and jnY > Ymax
    K=1
    toForm.Left = jnX
    toForm.Top  = jnY - H - Delta2
    thisform.Edit1.Width = thisform.width-6-Delta
  CASE jnX > Xscr/2 and jnX < Xmax and jnY > Ymax
    K=2
    thisform.Left = jnX - W
    thisform.Top  = jnY - H - Delta2
    thisform.edit1.Left = 3
    thisform.Edit1.Width = thisform.width-6-Delta

  CASE jnX > Xmax and jnY > Yscr/2
    K=2
    thisform.Left = jnX - W
    thisform.Top  = jnY - H - Delta2
    thisform.edit1.Left = 3
    thisform.Edit1.Width = thisform.width-6-Delta

  OTHERWISE
    K=1
    thisform.Left = jnX
    thisform.Top  = jnY -H - Delta2
    thisform.Edit1.Width = thisform.width-6-Delta
ENDCASE
*
DO CASE
  * По умолчанию: прямоугольник - Visio
  CASE thisform.ToolTipType = 1
    jnDots = 7
    DIMENSION jaDots(jnDots,2) && Region
    thisform.DotsRectangle_v(;
      @jaDots, k, h, delta, eh, w, delta2 )

  OTHERWISE
    WAIT WINDOW "Error in ToolTip call."
    RETURN
ENDCASE
*
strDots  = this.CreateDots(@jaDots)
nSkinRgn = CreatePolygonRgn(strDots,jnDots,1)
*

With thisform
  Local HDC, hRgnO1, jnRez
  HDC = GetDC( .HWND )
  Local lnPrevBrush, lnBrush
  lnBrush = CreateSolidBrush(RGB( 0,0,0))
  lnPrevBrush = SelectObject(m.HDC,m.lnBrush)
  jRez = FrameRgn(HDC,nSkinRgn,lnBrush,1,1)

  SelectObject( m.HDC, m.lnPrevBrush )
  DeleteObject( m.lnBrush )
  DeleteObject(hRgnO1)
  ReleaseDC( .HWND, m.HDC )
ENDWITH

=SetWindowRgn(thisform.HWnd, nSkinRgn, .f.)

RETURN
```

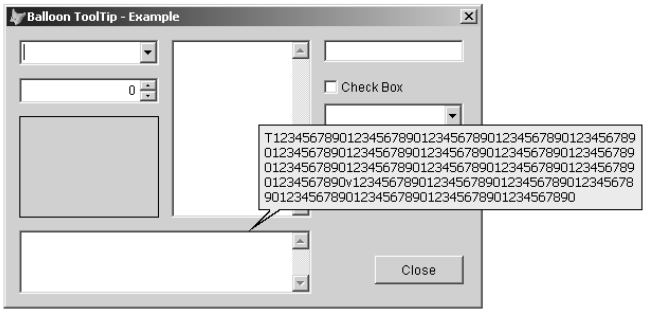

Рис. 6. Форма управляет поведением подсказки.

По сути я рисую подсказку на форме. Прежде всего, код рассчитывает ширину и высоту элемента и затем его ориентацию (K1, K2, K3 или K4). Следующей важной строкой является вызов функции CreatePolygonRgn. Эта функция создает область и возвращает указатель на него. В моем случае, область находится в контуре элемента «подсказка», и эта область определена методом DotsRectangle_v. Метод CreateDots создает шестнадцатеричное представление координат элемента. Код, заключенный внутри структуры With...EndWith, использует несколько функций Windows API для вычерчивания рамки вокруг элемента, ее раскраски и, в заключение, функция SetWindowRegion объединяет форму и область элемента. Последняя часть кода делает этот объект видимым, но оставляет саму форму прозрачной. На рис. 6 показан пример подсказки в форме выноски в действии.

Важно отметить, что эта форма должна вызывать метод Skin каждый раз, когда она рисуется. Метод Paint содержит следующий код:

```
thisform.Skin(this)
```

Библиотека wbToolTipB

Эта библиотека содержит два класса: ttScr2 и wbToolTipB. Последний класс должен быть добавлен на эту форму. Все элементы управления на форме поддерживают подсказки в форме выноски (Balloon Tooltip).

Класс ttScr2

На рис. 7 представлен класс ttScr2 с его методами и свойствами, описанными в таблице 1 и таблице 2 соответственно.

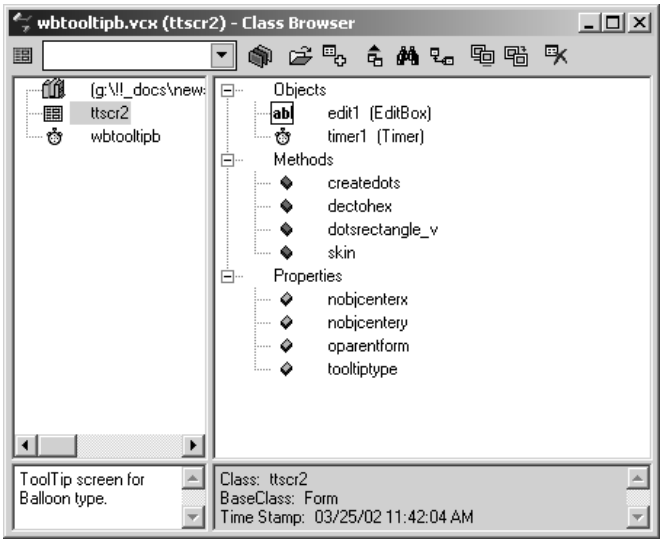

Рис. 7. Класс ttScr2, его методы и свойства.

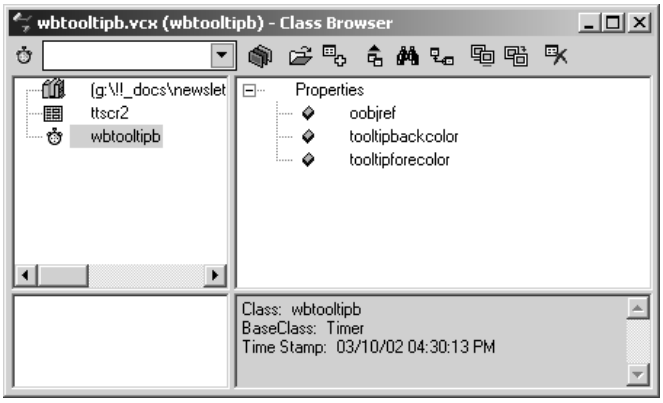

Рис. 8. Класс wbToolTip с его методами и свойствами.

Таблица 1. Методы класса ttScr2.

Имя	Тип	Описание
Skin	Private	Используется внутри класса. Вычисляет, вычерчивает и перерисовывает элемент «подсказка».
CreateDots	Private	Используется внутри класса. Создает массив точек для вычерчивания многоугольника.
DecToHex	Private	Выполняет преобразование десятичных чисел в шестнадцатеричные.
DotsRectangle_v	Private	Определяет семь значимых точек для вычерчивания контура элемента.

Таблица 2. Свойства класса ttScr2.

Имя	Тип	Описание
nObjCenterX	Int	Используется внутри класса. Положение указателя мыши, координата X.
nObjCenterY	Int	Используется внутри класса. Положение указателя мыши, координата Y.
oParentForm	Int	Ссылается на родительскую форму. Класс wbToolTipB.

На рис. 8 представлен класс wbToolTip с его методами и свойствами. Свойства класса описаны в таблице 3.

Таблица 3. Свойства класса wbToolTip.

Имя	Тип	Описание
nObjRef	Int	Используется внутри класса. Ссылается на объект в этой форме.
ToolTipBackColor	Int	Цвет фона подсказки.
ToolTipForeColor	Int	Основной цвет подсказки.

Усовершенствования

Для улучшения существующего кода подумайте о следующих вещах:

- Подсказка со скругленными углами.
- Подсказка с заголовком и, возможно, с изображением (см. рис. 2).
- Подсказка в виде облака.

Заключение

Подсказка в форме выноски является очень привлекательным элементом интерфейса и может применяться во всех приложениях. Это решение не является сложным и обладает достаточной скоростью. Скорее всего, вы обнаружите много общего между многострочной подсказкой и подсказкой в форме выноски. Если вы решили свести оба эти класса в один, я бы не советовал вам этого делать. Возможно, вы используете только один тип подсказки, а оставшаяся часть излишне перегружает код. В следующий раз я покажу вам, как создать очень специфический элемент интерфейса — подсказка в форме раскрывающегося списка (ComboBox Item ToolTip).

Предраг Боснич начал IT-карьеру в 1979 году с UNIVAC 1100, Fortran и Mapper. В течение 20 лет он постоянно живет в мире персональных компьютеров, dBase, Clipper и Fox, изредка возвращаясь домой, в Лондон, где работает ведущим разработчиком в компании Westwood Forster Ltd. и проделывает невероятные штуки с Visual FoxPro и SQL Server.
Его адрес: misobosnic@aol.com.

❑❑❑

FoxTalk *русское издание*

Печатается ежемесячно

Учредитель и издатель:

Страничка в Интернете: http://newsletter.narod.ru или
http://msnhomepages.talkcity.redmondave/dartemov/foxtalk.htm

(095) 325-5278
E-mail: foxtalk@online.ru
115304 Москва, а/я 208

Главный редактор: Д. Артемов
E-mail: dartemov@hotmail.com

Журнал зарегистрирован комитетом
Российской Федерации по печати.

Регистрационное свидетельство
№ 015520 от 17.12.1996

Подписано в печать 20/06/03. Формат 60x90 1/8. Тираж 330 экз.