FoxTalk

Май 2003
№ 5 (71)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

Поиск с использованием служб Web Services

Дуг Хенниг (Doug Hennig)

Службы Web Services облегчают доступ к функциональным возможностям других компьютеров. В настоящее время уже не один Web-сайт предоставляет услуги Web Service по поиску текста в своих архивах. Однако, каждая служба Web Service имеет свой собственный интерфейс и возвращает результаты в формате, отличном от форматов, используемых другими службами. В этом месяце Дуг Хенниг демонстрирует, как создать программную оболочку, которая обеспечит унифицированный способ обращения к различным службам Web Service для поиска с их помощью одного и того же текста.

материал имеет отношение к соответствующей версии

материал имеет отношение к соответствующей платформе

исходные тексты программ можно скачать из Интернета

Службы Web Service намерены изменить способ эксплуатации приложений. Вместо монолитных приложений, установленных на рабочих станциях или серверах, функциональность переносится на Web-серверы. Например, приложение для ввода данных о заказах могло бы передать заказ на доставку UPS, воспользовавшись услугами одной службы, и следить за тем, как осуществляется отправка по данному заказу, с помощью другой. Хотя вы и прежде могли поступать таким же образом, используя Web-сайт фирмы UPS, вам необходимо было выполнить «чистку» HTML-кода, чтобы полученные результаты были понятны для приложения. Использование Web Service аналогично обращению к функции на вашей локальной машине, за исключением того, что в действительности эта функция размещается на чьем-то еще сервере где-то в Internet. Хотя в данный момент определенное беспокойство вызывают вопросы, связанные с обеспечением безопасности и доступности таких служб, я полагаю, со временем эти проблемы будут решены.

Одной из весьма привлекательных особенностей версии VFP 7 является то, что эта версия облегчает создание ваших собственных служб Web Service. Прежде всего, вам необходимо инсталлировать пакет Microsoft SOAP Toolkit, который поставляется на одном компакт-диске вместе с версией VFP 7. Затем вы просто создаете COM DLL-библиотеку (объявите один из классов своего проекта как OLEPUBLIC и сформируйте DLL-библиотеку) и запускаете мастера Web Services Wizard (доступен из меню Tools | Wizards). Этот мастер создаст файлы, необходимые для ввода в эксплуатацию службы Web Service, включая WSDL-файл, который описывает интерфейс вашего класса. Он даже создаст виртуальный каталог сервера IIS и сконфигурирует его для вас. А если после внесения из-

менений вы заново сформировали свою DLL-библиотеку, перехватчик событий проекта (project hook) автоматически обновит файлы Web Service. Ничего не может быть проще! Подробное описание процесса создания ваших собственных служб Web Service ищите в статьях Уилла Хентцена (Whil Hentzen), опубликованных в декабрьском, январском и мартовском номерах журнала FoxTalk за 2002 и 2003 год.

Итак, большинство публично доступных служб, созданных средствами VFP, это поисковые механизмы, работающие на тех Web-сайтах, чья тематика связана с VFP:

- ProFox (www.leafe.com): созданная Эдом Лифом (Ed Leafe) и предназначенная для FoxPro-разработчиков доска сообщений на базе электронной почты. «Служебный» WSDL-файл находится по адресу www.leafe.com/profoxws.wsdl.
- FoxWiki (http://fox.wikis.com): Стивен Блэк (Steven Black) создал этот потрясающий ресурс, где можно отыскать тысячи документов по каждой мыслимой теме. Информация о службе Web Service доступна по адресу http://fox.wikis.com/wc.dll?Wiki~WikiWebServices, а WSDL-файл вы можете найти по адресу http://fox.wikis.com/wikiwebservice.wsdl.
- Universal Thread (www.universalthread.com): этот форум, организатором которого является Мишель Форнье (Michel Fournier), является тем местом, куда тысячи VFP-разработчиков заглядывают ежедневно, чтобы обменяться идеями и файлами. Документацию о Web Service ищите по адресу www.universalthread.com/WebService.asp; WSDL-файл находится по адресу www.universalthread.com/universalthread.wsdl.

В дополнение к этим сайтам, имеющим прямое отношение к VFP, фирма Google недавно создала службу Web Service для ведения поиска во всей сети Internet. Реализация этой службы не свободна от ограничений и несколько ненадежна. Спасибо Гектору Корреа (Hector Correa) за то, что он разместил на форуме FoxWiki пример программного кода, который демонстрирует, как следует обращаться к службе Google из VFP (http://fox.wikis.com/wc.dll?Wiki ~ GoogleWebServiceUsingVFP ~ VFP). Если вы хотите использовать службу Web Service фирмы Google, вы должны подписаться на эту услугу (бесплатно) и скачать соответствующий WSDL-файл (часть документации и примеры можно найти по адресу www.google.com/apis).

Некоторые службы Web Service использовать проще, чем другие. Вот как легко, например, найти все ссылки “Stonefield” на форуме FoxWiki:

```
loWS     = newobject('WSClient', ;
  home() + 'FFC\_WebServices.vcx')
loSearch = loWS.SetupClient('http://fox.wikis.com/' + ;
  'wikiwebservice.wsdl', '', '')
lcXML    = loSearch.GetTextSearch('Stonefield')
xmltocursor(lcXML)
browse
```

В этом фрагменте программного кода используется класс WSClient, который поставляется вместе с VFP и предоставляет оболочку для COM-объекта MSSOAP.SOAPClient, обеспечивающего доступ к службе. Метод SetupClient получает в качестве параметра URL-локатор, указывающий на WSDL-файл, и создает proxy-объект, интерфейс которого воспроизводит интерфейс службы, так что вы можете выполнять действия, аналогичные локальным обращениям. Однако, «за кулисами» происходит следующее: когда вы обращаетесь к методу proxy-объекта, это обращение пакуется в SOAP-оболочку, которая передается Web Service. (Чтобы получить более подробные сведения о внутреннем устройстве Web Service, выполните на Web-сайте сети разработчиков MSDN, доступном по адресу http://msdn.microsoft.com, поиск строки “SOAP” или “Web Services”.) В нашем случае мы обращаемся к методу GetTextSearch службы Web Service, организованной на Web-сайте форума FoxWiki, который возвращает VFP-курсор, упакованный в виде XML-строки. Так что мы можем воспользоваться функцией XMLToCursor(), чтобы преобразовать этот XML-код обратно в курсор и просмотреть полученные результаты.

Работа со службой ProFox столь же проста, но обратите внимание на различия в имени и параметрах используемых методов (в данном случае метод получает в качестве параметра диапазон дат), и, кроме того, в ином формате представлены результаты поиска:

```
loWS     = newobject('WSClient', ;
  home() + 'FFC\_WebServices.vcx')
loSearch = loWS.SetupClient('http://www.leafe.com/' + ;
  'profoxws.wsdl', '', '')
lcXML    = loSearch.SearchTextLinks('Stonefield, ;
  {^2002-01-01}, date(), .F.)
xmltocursor(lcXML)
browse
```

И форум Universal Thread, и служба фирмы Google гораздо сложнее в обращении. Служба, организованная на форуме Universal Thread, требует, чтобы вы выполнили процедуру входа, и возвращает значение уникального идентификатора ID, которое вы должны указывать при каждом обращении к методу. Однако, это значение не передается как параметр; оно должно быть «встроено» в SOAP-заголовок (см. подробности в документации, предоставляемой для этой службы). Служба поиска Google не требует регистрации; при каждом обращении к соответствующему методу вы передаете значение уникального идентификатора, присвоенного вам при оформлении

подписки на эту услугу. Однако, выполняющий поиск метод возвращает массив результатов, в котором вам придется «порыться», чтобы извлечь оттуда содержимое, поэтому вы не можете просто воспользоваться обычным обращением к функции XMLTOCURSOR().

Поскольку все службы отличаются друг от друга как с точки зрения способа обращения к ним, так и тем, как выглядят результаты их работы, выработка решения проблемы поиска с помощью нескольких служб может оказаться непростым делом. Чтобы облегчить его, я создал некую программную оболочку.

Поиск, управляемый данными

Изначально, для поиска с помощью одной службы, мной была написана жестко закодированная программа-оболочка, но как только я добавил в эту программу код, позволяющий вести поиск, обращаясь к нескольким службам Web Service, то понял, что будет лучше использовать для управления этим процессом данные. Иначе говоря, таблицу, где хранятся сведения о каждой из используемых для поиска служб Web Service, и некоторый программный код универсального характера, обрабатывающий эту таблицу. Таким образом, подключить к поиску еще одну службу Web Service оказывается не сложнее, чем добавить в таблицу новую запись.

Поскольку каждая служба имеет свой собственный, отличный от других служб, способ обращения к ней и возвращает результаты, представленные в собственном формате, я решил хранить необходимый, специфичный для каждой службы, программный код в memo-поле упомянутой таблицы. Этот код исполняется при помощи реализованной в версии VFP 7 новой функции EXECSCRIPT().

Таблица, в которой определены все используемые для поиска службы Web Service и то, как к ним следует обращаться, называется WSSEARCH.DBF; ее структура показана в таблице 1.

Вот пример записи из таблицы WSSEARCH. В этой записи, относящейся к службе Web Service форума FoxWiki, хранится адрес WSDL-файла — http://fox.wikis.com/wikiwebservice.wsdl, в полях USERNAME и PASSWORD записаны пустые значения, а в поле SEARCH хранится следующий программный код:

```
lparameters toWS, tcSearch, tdBegin, tdEnd, tcUserName, ;
   tcPassword
 local lcXML
 lcXML = toWS.GetTextSearch(tcSearch)
 if not empty(lcXML)
   xmltocursor(lcXML, '_WikiResults')
   if reccount() = 0
     lcXML = ''
   else
     select NAME as TITLE, LINK as URL, '' as CONTENTS ;
       from _WikiResults into cursor _Results
     cursortoxml('_Results', 'lcXML', 1, 0, 0, '1')
     use in _Results
   endif reccount() = 0
   use in _WikiResults
 endif not empty(lcXML)
 return lcXML
```

Этот программный код обращается к методу GetTextSearch используемой службы, передавая ему в качестве параметра текстовую строку поиска. Обратите внимание на то, что в представленном программном коде не используются начальная и конечная даты, а также имя и пароль пользователя, поскольку служба Web Service форума FoxWiki не нуждается ни в чем ином, кроме строки поиска. Если возвращаемое методом GetTextSearch значение не является пустым, тогда используется функция XMLTOCURSOR() для преобразования полученных результатов в курсор и — если поиск дал какие-то результаты — используется команда SQL-SELECT для создания курсора с необходимой структурой (поле TITLE для заголовка документа, поле URL для URL-локатора документа и поле CONTENTS для содержимого документа), а также вызывается функция CURSORTOXML() для обратного преобразования созданного курсора в XML-строку. Затем рассматриваемый программный код возвраща-

Таблица 1. Структура таблицы WSSEARCH.

Имя поля	Тип/размер	Описание
ORDER	N(2, 0)	Порядок обработки записей.
ACTIVE	L	Значение .T., если данная служба должна быть задействована при поиске (вы можете указать значение .F., чтобы временно исключить из поиска ту службу, о которой вам известно, что она недоступна).
NAME	C(30)	Описательное имя службы.
WSDL	M	URL-локатор для WSDL-файла Web Service.
SEARCH	M	Программный код для обращения к тому методу Web Service, который осуществляет поиск. Этот код передает созданному для данной Web Service proxy-объекту строку поиска, начальную и конечную даты и содержимое полей USERNAME и PASSWORD. Метод должен вернуть пустую строку или XML-строку с полученными результатами, которая сформирует курсор с полями TITLE, URL и CONTENTS.
USERNAME	C(10)	Имя пользователя для доступа к Web Service.
PASSWORD	C(10)	Пароль.

ет либо обновленную XML-строку, либо пустую строку, если поиск оказался безрезультатным.

Класс WSSearch

Базовым для определенного в файле WSSearch.PRG класса WSSearch является класс Session. Класс WSSearch имеет три пользовательских свойства: свойство cErrorMessage, в котором запоминается сообщение об ошибке, полученное в случае возникновения сбойной ситуации (например, если бы мы не смогли инициализировать объект MSSoap); свойство cProcessingMessage, в котором хранится строка-индикатор прогресса поиска; и aResults — свойство-массив, в котором запоминаются результаты поиска. Метод Init открывает таблицу WSSEARCH, метод Destroy закрывает ее, а метод Error просто запоминает в свойстве cErrorMessage значение функции MESSAGE(), так что мы можем зафиксировать сообщение об ошибке.

Всю основную работу выполняет метод Search. Этот метод получает три параметра: строку поиска и начальную и конечную даты, ограничивающие область поиска. Метод Search помещает результаты каждого поискового обращения к службе в строку массива aResults и возвращает число строк в этом массиве. Первый столбец массива aResults содержит имя Web Service; второй — значение .T., если поиск был успешным; и в третьем столбце запоминается возвращенная Web Service XML-строка, если поиск был успешным, или текст сообщения об ошибке, если поиск потерпел неудачу (например, сообщение о том, что служба недоступна).

Вот первая часть программного кода этого метода. Если какой-либо из параметров имеет неверный тип данных или пуст, свойство cErrorMessage получает соответствующее значение, и метод возвращает значение -1.

```
function Search(tcSearch, tdBegin, tdEnd)
local lnResults, lcName, loWS, loSearch, llOK, ;
  lcMessage, lcXML
with This

 * Убедиться в том, что у нас есть допустимые параметры.
   if vartype(tcSearch) <> 'C' or empty(tcSearch)
     .cErrorMessage = 'Invalid search string'
     return -1
   endif vartype(tcSearch) <> 'C' ...
   if vartype(tdBegin) <> 'D' or empty(tdBegin)
     .cErrorMessage = 'Invalid starting date'
     return -1
   endif vartype(tdBegin) <> 'D' ...
   if vartype(tdEnd) <> 'D' or empty(tdEnd)
     .cErrorMessage = 'Invalid starting date'
     return -1
   endif vartype(tdEnd) <> 'D' ...
```

Затем этот код инициализирует переменную, в которой запоминается номер обрабатываемой службы, и организует цикл SCAN, в котором обрабатываются активные записи из таблицы WSSEARCH. Для каждой записи о службе метод Search добавляет в массив aResults новую строку и запоминает в свойстве cProcessingMessage значение, указывающее на то, что метод пытается получить доступ к данной службе.

```
   lnResults = 0
   scan for ACTIVE

 * Добавить новую строку в массив и сформировать сообщение
 * об обработке.
     .cErrorMessage = ''
     lcName         = alltrim(NAME)
     lcCode         = SEARCH
     lcUserName     = alltrim(USERNAME)
     lcPassword     = alltrim(PASSWORD)
     lnResults      = lnResults + 1
     dimension .aResults[lnResults, 3]
     .aResults[lnResults, 1] = lcName
     .cProcessingMessage     = 'Accessing ' + lcName
```

Затем этот код создает экземпляр объекта подкласса класса WSClient (мы рассмотрим этот подкласс позже) и обращается к методу SetupClient, чтобы создать proxy-объект для обрабатываемой службы. Если он терпит неудачу (вероятно, по причине того, что вы не подключены к Internet, или Web Service в данный момент недоступна), метод Search выставляет флажок и перехватывает сообщение об ошибке от объекта WSClient.

```
     loWS         = createobject('WSClient3')
     loWS.cWSName = lcName
     loSearch     = loWS.SetupClient(alltrim(WSDL), '', ;
       '')
     lcMessage    = loWS.cErrorMessage
     llOK         = vartype(loSearch) = 'O' and ;
       empty(lcMessage)
```

Если метод Search успешно создал proxy-объект для данной службы, он обновляет свойство cProcessingMessage (чтобы указать на то, что он выполняет поиск с помощью Web Service) и обращается к программному коду, хранящемуся в memo-поле Search, передавая ему proxy-объект, строку поиска, начальную и конечную даты, а также имя и пароль пользователя. Программный код из memo-поля Search вернет либо XML-строку, содержащую результаты поиска, либо пустую строку, если произошла ошибка или поиск оказался безрезультатным, так что метод Search помещает полученный результат или сообщение об ошибке в массив и выставляет флажок, свидетельствующий об успешном или неуспешном завершении поиска.

```
     if llOK
        .cProcessingMessage = 'Searching ' + lcName
        lcXML = execscript(lcCode, loSearch, tcSearch, ;
          tdBegin, tdEnd, lcUserName, lcPassword)
        .aResults[lnResults, 2] = not empty(lcXML) and ;
          '<?xml' $ lcXML
        do case
          case not empty(lcXML)
            .aResults[lnResults, 3] = lcXML
          case not empty(.cErrorMessage)
            .aResults[lnResults, 3] = .cErrorMessage
          otherwise
```

```
            .aResults[lnResults, 3] = 'No results'
        endcase
```

Если метод Search не смог создать proxy-объект, соответствующее сообщение помещается в третий столбец текущей строки массива.

```
    else
       .aResults[lnResults, 3] = lcMessage
    endif llOK
```

Наконец, если метод Search обработал все службы, он записывает в свойство cProcessingMessage значение "done", очищает сообщение об ошибке (поскольку все сообщения об ошибках будут теперь находиться в массиве aResults) и возвращает число строк в массиве aResults.

```
  endscan for ACTIVE
   .cProcessingMessage = 'Done'
   .cErrorMessage      = ''
 endwith
 return lnResults
```

Класс WSClient3

Вместе с VFP поставляются два класса, WSClient и WSClient2, которые обеспечивают оболочку для клиентского объекта Microsoft SOAP; оба эти класса определены в библиотеке _WebServices.VCX, размещенной в подкаталоге FFC «домашнего» каталога VFP. Класс WSClient хорош для интерактивного использования (например, в случае работы из командного окна Command Window), но в приложениях лучше работать с классом WSClient2, поскольку он не использует функцию MESSAGEBOX() для выдачи на экран предупреждений в случае возникновения проблем.

При использовании класса WSClient2 я столкнулся с парой небольших проблем. Первое, если proxy-объект не может быть создан, класс WSClient2 не сохраняет сообщения об ошибке, поскольку метод ErrorAlert, вызываемый из метода Error, ничего не делает (у него есть предложение IF .F., которое прерывает работу метода). Второе, метод SetupClient не выполняет за собой надлежащую чистку; он меняет текущую рабочую область и не восстанавливает ее по завершении своей работы. Наконец, метод CheckWSDBF, который в действительности определен в классе WSClient и вызывается из нескольких методов, открывает требуемую таблицу, которая необходима классу для записи в нее данных, если она еще не была открыта, но не выбирает эту таблицу, если она уже была открыта ранее. Метод, вызвавший метод CheckWSDBF, ждет, что требуемая таблица выбрана, и если это не так, работает неправильно.

Чтобы обойти эти вопросы, я создал подкласс класса WSClient2 с именем WSClient3, также определенный в программе WSSEARCH.PRG. Метод ErrorAlert подкласса WSClient3 хранит переданное ему методом Error сообщение в новом свойстве cErrorMessage, так что все, кто использует этот класс, могут определить, что произошло. Метод SetupClient хранит текущую рабочую область, использует функцию DODEFAULT(), чтобы обеспечить штатное функциональное поведение, и затем восстанавливает рабочую область. Наконец, CheckWSDBF выбирает соответствующую таблицу, если она уже была открыта.

```
define class WSClient3 as WSClient2 of ;
   (home() + 'ffc\_webservices.vcx')
   cErrorMessage = ''

   function ErrorAlert(tcMessage)
     This.cErrorMessage = tcMessage
   endfunc

   function SetupClient(tcURI, tcService, tcPort)
     local lnSelect, ;
       luReturn
     lnSelect = select()
     luReturn = dodefault(tcURI, tcService, tcPort)
     select (lnSelect)
     return luReturn
   endfunc

   function CheckWSDBF
     local llReturn
     if not empty(This.cWSAlias) and used(This.cWSAlias)
       select (This.cWSAlias)
       llReturn = .T.
     else
       llReturn = dodefault()
     endif not empty(This.cWSAlias) ...
     return llReturn
   endfunc
enddefine
```

В заключение

Чтобы облегчить обращение к классу WSSearch, я создал для него простую клиентскую форму. Форма WSSEARCH.SCX имеет поле textbox для текстовой строки и элементы управления ActiveX DateTimePicker для начальной и конечной дат. Когда вы щелкаете мышью по кнопке Search, эта кнопка использует класс WSSearch, чтобы выполнить поиск и преобразовать затем полученные результаты в HTML-документ, используя следующий программный код:

```
set textmerge on to Results.html noshow
 \\<html>
 \<head>
 \<title>Web Services Search Results</title>
 \</head>
 \<body>
 \<h1>Search Criteria</h1>
 \<b>Search for:</b> <<lcSearchFor>><br>
 \<b>From:</b> <<ldStart>><br>
 \<b>To:</b> <<ldEnd>>
 for lnI = 1 to lnResults
   \<h1><<alltrim(.oSearch.aResults[lnI, 1])>></h1>
   if .oSearch.aResults[lnI, 2]
     xmltocursor(.oSearch.aResults[lnI, 3], 'TEMP')
     \<table>
     \<tr><th>Title</th><th>Link</th>
     \\<th>Contents</th></tr>
     scan
```

```
      \<tr><td valign="top"><<alltrim(TITLE)>></td>
      \\<td valign="top"><a href="<<alltrim(URL)>>">
      \\<<alltrim(URL)>></a></td>
      lcContents = strtran(alltrim(CONTENTS), ccCRLF, ;
        '<br>')
      lcContents = strtran(lcContents, ccCR, '<br>')
      lcContents = strtran(lcContents, ccLF, '<br>')
      \\<td valign="top"><<<lcContents>>></td></tr>
    endscan
    use
    \</table>
  else
    \<p><<.oSearch.aResults[lnI, 3]>>
  endif .oSearch.aResults[lnI, 2]
next lnI
\</body>
\</html>
\
set textmerge to
```

Полученный в итоге документ имеет работающие ссылки на каждый из разделов с результатами поиска.

Разумеется, прежде чем вы действительно сможете всем этим воспользоваться, вам необходимо внести в соответствующие поля таблицы WSSEARCH.DBF свои имя и пароль пользователя для входа на форум Universal Thread и свой пользовательский идентификатор для поиска с помощью службы Google, а также поместить копию файла GoogleSearch.WSDL в тот же самый каталог, что и другие файлы, используемые в данной статье.

Предлагаемая форма (и класс, который она использует) облегчают поиск решений тех проблем, которые, как вам известно, наверняка уже решены другими VFP-разработчиками, или позволяют легко найти информацию о новых технологиях (например, о том, как использовать службы Web Services) без необходимости «ручного» запуска поискового механизма каждого Web-сайта. Надеюсь, этот инструмент окажется для вас столь же полезным, как и для меня!

❑❑❑

Понять и организовать данные

Нэнси Фолсом (Nancy Folsom)

В своей статье («Руководство к действию: как съесть слона?»), опубликованной в декабре 2002 года, Нэнси Фолсом познакомила вас с простым примером того, как разработчик может использовать стадию сопровождения проекта для проведения рефакторинга результатов традиционного структурного программирования с тем, чтобы воспользоваться преимуществами концепции ООП. В этом месяце Нэнси завершает абстрагирование типов оплаты, целью которого является демонстрация наглядного примера того, как можно отделить данные и бизнес-логику от пользовательского интерфейса UI. Особое внимание она уделяет внедрению некоторых простейших элементов n-уровневой архитектуры.

В начале были компьютеры, которые развозились на тележках и совместно использовались всеми сотрудниками. Политика по меньшей мере одной инженерной фирмы основывалась на том убеждении, что не было и никогда не появится никаких оснований к тому, чтобы инженер, не говоря уже о секретарше, имел у себя на рабочем столе свой личный компьютер: с точки зрения производительности труда это было совсем не критично. Если идея о компьютере, установленном на каждом рабочем столе, вызывала удивление, то мысль о компьютерах, объединенных в сеть, была просто радикальной.

Я надеюсь, сегодня представление о том, что компьютер является полезным, если не необходимым инструментом, больше не является революционным, также как и понимание того, что сила информации заключается в том, каким образом осуществляется ее совместное использование.

На наших глазах аналогичный переход осуществляется в архитектуре программного обеспечения. Медленно, но неуклонно внедряется та точка зрения, что приложения должны делить между собой ответственность за данные, правила и пользовательский интерфейс. Такое разделение называется уровнями (или слоями). Трехуровневая (или, чаще, n-уровневая) архитектура программного обеспечения не нова. Те, кому посчастливилось пользоваться одним из добротных коммерческих каркасов (framework), имеют, вероятно, прочные практические познания в области n-уровневой архитектуры. Однако, для многих из нас, тех, кто работает с унаследованными приложениями, чьи функциональные возможности наращивались в течение многих лет, идентификация области приложения, подходящей для внедрения хотя бы элементарной структуры, которая позволяет отделить данные от формы, не говоря уже о внесении реальных изменений, может оказаться непростым делом. С учетом того, что имеются залежи доступной информации об архитектуре программно-

го обеспечения, а отведенное мне в этом журнале пространство ограничено, привожу свое личное и неформальное определение. N-уровневая архитектура — это некий способ разделить обязанности приложения. Как правило, эти обязанности имеют дело со следующими составными частями приложения: 1) данными; 2) ограничениями, накладываемыми системой или приложением на взаимодействие с этими данными; и 3) представлением информации для внешнего мира, будь это пользователь или клиентская система: короче говоря, обязанности приложения имеют дело с данными, бизнес-логикой и пользовательским интерфейсом (UI). В условиях конкретной архитектуры каждая из этих крупных категорий может подвергаться дальнейшему делению. Границы между уровнями не всегда очевидны, и способ реализации концепций также может оказаться непрозрачным. Например, что такое «бизнес-правило», и какое место оно занимает в пространстве между уровнями пользовательского интерфейса и данных — вопрос тонкий.

Собственно говоря, бизнес-правило — это две вещи в одном. Бизнес-правила выступают в роли посредников между внешним миром (например, пользовательским интерфейсом) и данными. Такая роль бизнес-правил предполагает, что они взаимодействуют с данными и документируют набор правил, применяемых к этим данным, и, кроме того, бизнес-правила делают эти данные осмысленными с точки зрения их использования. Унаследованные приложения являются, в некотором смысле, жертвами успешного развития СУБД FoxPro. СУБД Visual FoxPro обладает мощными и удобными инструментальными средствами, которые позволяют разработчикам в сжатые сроки завершить большой объем работы. В сравнении с программированием команд @GET и @SAY, прекрасным усовершенствованием оказались, например, буферизация и связь элементов управления с источником данных через свойство ControlSource. Среда окружения данных плюс сессии данных сделали задачу управления данными почти тривиальной. Триггеры, обеспечивающие ссылочную целостность, были восприняты как облегчение после многих лет написания своих собственных модулей, обеспечивающих ссылочную целостность.

К сожалению, то, что делает СУБД Visual FoxPro инструментом для быстрой разработки приложений, в то же время ограничивает традиционные Visual FoxPro-приложения, сдерживая взаимодействие с различными источниками данных, когда, например, один клиент «обретается» в базе данных Access, а другой использует СУБД Oracle. Возможность адаптировать новые подходы, например, программировать приложения для работы в глобальной сети Internet, ограничена. Тот факт, что данные так легко привязать прямо к интерфейсным элементам управления, оборачивается ограничением в приложениях, чье развитие протекает в этом прекрасном новом мире распределений в масштабе Internet и быстро меняющихся рынков — изменений, к которым необходимо адаптироваться нашим заказчикам.

Вы, возможно, думаете: «Какое отношение все это имеет к моему заказчику, который владеет одним магазином по продаже кухонного оборудования?» Простое. Давайте предположим, что магазин — это три этажа, заполненных покупателями. Ваш клиент был фантастически успешен. Сегодня ваш клиент взаимодействует с внешним миром такими способами, о которых и не помышляли в тот момент, когда приложение только создавалось. Например, программное обеспечение для организации поставок с помощью службы UPS находится в базе данных Access. У вашего клиента может возникнуть необходимость увеличить объем продаж по заказам, сделанным по обычной почте, что неизбежно влечет за собой повторное использование бизнес-правил. Ваш клиент мог бы заниматься подсчетом стоимости «тележки заказов» для Internet-магазина, для чего используются разнообразные возможности, которые предоставляет хранение данных на сервере. Бизнес вашего клиента претерпевает изменения, и вместе с тем остается еще много заказчиков, не располагающих средствами, достаточными для перепроектирования приложения, на создание которого уже затрачены крупные инвестиции. Не все приложения используют коммерческие каркасы, а внедрение такого каркаса в унаследованную систему может оказаться непростым делом. Наконец, многие приложения используют компоненты пользовательского интерфейса, формы например, в качестве «рабочих лошадок» приложений, «везущих» на себе весь программный код, необходимый для обеспечения доступа к данным и организации функциональных возможностей приложения. Необходимость реализации идей n-уровневого проектирования в таких условиях может привести в уныние.

Уровни — это образ мышления

Уровни не обязательно должны быть физическими по своей природе, когда, например, данные обрабатываются на одном компьютере, а общение с потребителем этих данных осуществляется на другой машине. Такой случай — это двухуровневая или клиент/серверная модель, и это вчерашний, такой вчерашний день. Количество уровней, имеющих смысл для конкретной архитектуры, может быть любым. В

этой статье используется простая трехуровневая модель. Как уже было сказано ранее, уровни — это способ представления, моделирования или разделения обязанностей в приложении. Уровни, которые делят между собой пользовательский интерфейс, бизнес-логику и данные, воспринимаются почти как заранее известное решение или закон природы. Такая модель полезна и обладает широкими возможностями, но, кроме того, она рациональна. Причина, по которой имеет смысл распределить обязанности подобным образом, заключается в том, что такое распределение не препятствует внесению в приложение малейшего изменения.

Если вы представите себе какой-либо слоистый или гранулированный материал, например, дерево, ткань или слюду, то поймете, что резать этот материал в одном направлении легче, чем в другом. По аналогии с такими материалами, «начинка», заключенная внутри каждого уровня, имеет тенденцию оставаться относительно стабильной или, по крайней мере, меняться сходным образом. Например, если пользователь приступает к работе с приложением, то ему предстоит иметь дело с набором взаимосвязанных правил (не забудьте, это рассуждение с точки зрения пользователя), которые структурируют его задачи. Пользователь принимает решение о том, что покупатель имеет право на скидки, на основе данных об «истории» приобретений, сделанных покупателем, сведений о продолжительности периода, в течение которого этот покупатель пользуется услугами данной фирмы, сумме приобретений и так далее. Он не основывает свое решение на том, является ли форма Web-страницей в intranet-сети фирмы или это локальная Fox-форма. Права покупателя не связаны с тем, хранятся ли данные о нем в наборе FoxPro-, SQL Server- и Access-файлов.

Бизнес-правилом может стать все, что не зависит от способа представления и хранения информации. Возвращаясь на минуту назад, «бизнес» — несколько обманчивое определение, поскольку оно подразумевает, что этот уровень применим только для бизнес-приложений, таких как система кассовых терминалов в розничной торговле или система управления запасами. В другом месте я использую как эквивалент альтернативное понятие «ограничение» (constraint). В самом общем смысле, ограничение означает, что даже правила ссылочной целостности могут рассматриваться как бизнес-правила. Я не настаиваю на такой точке зрения, но упоминаю о ней, поскольку такое представление заостряет внимание на наличии «серой» области, лежащей между уровнями.

Уровни не обязательно должны быть физическими за счет того, что вся обработка, предусмотренная для каждого из них, осуществляется на отдельных компьютерах. Уровни даже не обязательно должны быть поделены между программными модулями. Другими словами, даже если правила ссылочной целостности являются бизнес-правилами, а не правилами данных, это не означает, что сервер (back end) не может применять бизнес-правила. Так же, если рефакторингу подвергается существующая часть приложения, может случиться, что пределом мечтаний будет распределение программной логики по определяемым на базе уровней логическим единицам, таким, например, как методы формы. Программный код не обязан находиться в разных программных модулях или в разных местах с точки зрения его физического хранения. Программный код, как минимум, централизован и подготовлен к наступлению того момента, когда представится удобный случай для его перепроектирования.

Эти уровни можно уподобить линиям, нарисованным на песке, или занавесу, который опускался в перерыве между показом пары голливудских кинофильмов в 1940-х годах. Даже если создание «совершенного» решения — это непозволительная роскошь, все-таки продвижение вперед на такой шаг, который может быть осмыслен, запрограммирован и протестирован, может оказаться полезным. Я не всегда должна иметь классы, которые заняты одной только проверкой логики (проверить, указан ли тип оплаты), работать с базами данных, которые выполняют лишь проверку допустимости хранящихся в полях таблиц значений (сумма оплаты > 0) и создавать формы, которые ограничиваются тем, что предоставляют продавцу анкету для сбора необходимых данных. Зачастую я начинаю распределять обязанности в ходе работы с готовыми задачами: при реализации необходимой заказчику новой функциональной возможности. Я поняла, что мне не надо для этого ждать появления совершенно нового приложения или заказчика. Обязанности можно распределять постепенно.

В прошлый раз

Чтобы напомнить содержание предыдущей статьи: заказчику необходимо было добавить новый тип оплаты в размещенный в форме список. Изначально этот список был заполнен жестко запрограммированными значениями. Оказалось, что, имея полчаса свободного времени, сравнительно несложно воспользоваться имеющейся задачей как удобным случаем для извлечения из формы жестко запрограммированной информации о данных (типы оплаты). Я по шагам продемонстрировала, как осуществляется переход от исходной ситуации через добавление но-

вого класса (класс ComboBox) к такому представлению на экране типов оплаты, которое не зависит от того, какие именно форма или иной элемент управления могли бы их использовать. Затем я создала для типов оплаты бизнес-класс, который вначале просто перемещал жестко запрограммированный список из формы в бизнес-объект. Я добавила класс ComboBox, способный создать экземпляр объекта упомянутого бизнес-класса и обратиться к этому объекту с просьбой о предоставлении списка допустимых типов оплаты. Хотя такая адаптация может показаться не слишком значительным достижением, она не потребовала много времени, позволила перенести жестко запрограммированные данные в класс, так что этот класс мог, по крайней мере, быть использован повторно (инкапсуляция), и облегчила тестирование вносимых изменений.

В заключение я полностью высвободила жестко запрограммированные данные и создала таблицу для хранения типов оплаты. Класс ComboBox попрежнему запрашивал список типов оплаты у бизнес-объекта, но теперь бизнес-объект загружал этот список из справочной таблицы. Смысл реализации такого поступательного подхода заключался в том, чтобы обеспечить отклик на требование заказчиков (они быстро получали некоторую порцию того, что им было необходимо), я могла тестировать не тотальные, а небольшие изменения в программном коде, и этим предварялась возрастающая потребность в более гибкой организации данных. Точнее говоря, такой подход предугадывал, что в некоторый момент от типов оплаты потребуется не одна только систематизация ежедневных денежных поступлений. Упреждающее изменение — вот одна из целей n-уровневой разработки и одно из ее обещаний. Говорить о будущем трудно, и, тем не менее, направления развития приложений и данных вполне предсказуемы. Выявляя кандидатуры для n-уровневого рефакторинга, полезно ответить на вопросы: «Как это могло бы измениться в будущем? Как бы я приспособился к такому изменению?»

Постепенно наращиваемое тестирование вселяло в меня уверенность в том, что форма обеспечивает выдачу на экран типов оплаты путем общения с бизнес-объектом, и ей безразлично, откуда эти данные были получены. При этом проведение тестирования реализации справочной таблицы попадает в единственно возможную временную точку, которая имеет дело только с одной задачей. Даже если бы приложение могло всю жизнь прожить с тем механизмом обработки данных, который обеспечивает Visual FoxPro, все-таки существует вероятность того, что использованные структуры претерпят изменения, дабы соответствовать новым потребностям. В любом случае, даже такое простое дело требует подведения фундамента под создание классов, которые могут использовать иные, более сложные данные. Я себе это представляю как практическое моделирование (live prototyping).

Выражаясь точнее, трехуровневая разработка для программиста, работающего исключительно с FoxPro, означает, что элементы управления Form, Text Box, ComboBox и так далее не содержат ссылок на конкретные таблицы. Аналогичным образом, элементы управления не реализуют правила, которые накладывают ограничения на способ взаимодействия внешнего мира с данными. Классы данных несут ответственность только за взаимодействие с данными, размещенными на диске — в любом месте постоянного хранения данных. В идеале, ограничения содержатся в классах, отвечающих только за управление потоком информации, которая поступает «в», исходит «из» и проходит «через» приложение. Ограничения отчасти напоминают полицейских, регулирующих уличное движение. Регулировщики только руководят движением, и делают это из некоторого центра. Если бы они осуществляли свое руководство, будучи сами в движении, или если бы они переходили от одного перекрестка к другому, поток движения оказался бы субоптимальным. Поэтому, в идеале, настоящие эксплуатируемые данные имеют класс для общения с ними — подобно кинозвезде, у которой есть ассистент для чтения писем от поклонников. Класс данных, в свою очередь, общается с классом ограничений, который действует как камердинер, отбирающий и сортирующий костюмы, предназначенные для отдыха, от тех, которые соответствуют деловой встрече — в идеале, но не обязательно. Как было показано ранее, даже в иногда беспорядочном мире унаследованных приложений существует все-таки возможность реализовать n-уровневые концепции прежде, чем идеальная ситуация сама проявит себя.

В действительности, масштаб реализации n-уровневых архитектур может быть разным. Например, я могла бы промоделировать процесс, или осуществлять рефакторинг для некоторой части унаследованного приложения. В таком случае я могу поместить программный код, связанный с конкретной таблицей, в форму, или я могла бы добавить метод на уровне формы, который определяет полноту записи. Однако, я буду изолировать любое конкретное правило, позволяющее определить, достаточно ли информации ввел пользователь для данной записи, так, чтобы оно в полной мере было учтено в методах формы.

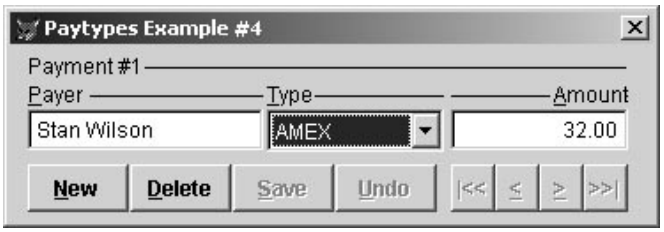

Рис. 1. Форма ввода платежей.

Снова возвращаюсь к форме для ввода данных об оплате (см. рис. 1). В этой форме представлены три логических типа данных: плательщики, суммы платежей и типы оплаты. Я пока сосредоточусь на типах оплаты. Эта форма не управляла типами оплаты, и в исходном проекте не было иного способа для организации такого управления, кроме как внести изменения в программный код формы. Извлечение типов оплаты из формы для ввода данных о платежах следовало рука об руку с нормализацией данных. Поскольку пока еще отсутствует механизм для выполнения данного требования, имеется хорошая возможность внедрить еще один «фрагмент» n-уровневого и объектно-ориентированного программирования.

Даже несмотря на возможность существования такого представления, что типы оплаты — это всего лишь вопрос названия (AMEX, Visa/MC и так далее), есть естественный способ дополнить это понятие таким образом, чтобы оно было более полезным. Я расскажу об этом дополнении немного позже. Однако, даже если это понятие навсегда сохраняет свое обычное толкование, заложить фундамент для других, более сложных элементов, намного проще, используя для этого простой пример. При разработке все внимание может быть сосредоточено на реализации и тестировании компонентов. Таким образом, одна из преследуемых целей — получить в конечном итоге нечто такое, что может быть повторно использовано для других типов данных, таких, например, как плательщики или платежи. Важно учесть этот принцип, разрабатывая первичный план работы.

Типы оплаты представлены в классе ComboBox. Очевидно, что класс ComboBox не связан со способом хранения данных (эти типы могли бы храниться в текстовом файле) или с тем, как организованы типы оплаты. Исходя из того, что класс ComboBox остается в форме payment — и он там останется — возникает вопрос: за что тогда несет ответственность пользовательский интерфейс? Пользовательский интерфейс отвечает за получение списка допустимых типов оплаты и представление его на экране для пользователя. Поскольку все обращения к данным осуществляются через посредничество бизнес-объекта, получение данных должно быть функцией бизнес-объекта. Так же точно, допустимость вариантов выбора не имеет ничего общего с тем, как эти варианты представлены на экране, поэтому эта обязанность также возлагается на бизнес-класс. Класс ComboBox, однако, несет ответственность за демонстрацию списка пользователю. Элемент управления ComboBox, встроенный в Visual FoxPro, может поддерживать для такого списка несколько различных источников данных, которые варьируются от случая отсутствия источника (no source) (список формируется вручную) до курсора (cursor). Выбор источника — это то занятие, которое можно отложить «на потом». В любом случае решение этого вопроса не должно оказаться критичным, поскольку существует вероятность того, что типы могли бы быть представлены на экране несколькими различными способами, в зависимости от контекста.

С точки зрения пользовательского интерфейса, на бизнес-класс типов оплаты возложены элементарные обязанности: получить данные и выбрать допустимые в данном контексте данные. Помните, пользовательский интерфейс — форма payments и элемент управления ComboBox — это лишь один пример взаимодействия внешнего мира с приложением. Рассмотрим сначала вторую обязанность. Выбор допустимых для данного контекста типов оплаты — это обязанность бизнес-класса. Этот выбор не входит в обязанности интерфейса, как только что было продемонстрировано, и он не является обязанностью данных, поскольку со всеми типами данных база данных, в которой эти данные хранятся, обращается одинаково, почти одинаково. Следовательно, выбор допустимого типа оплаты будет функцией бизнес-класса.

Другая обязанность класса ComboBox — это получение списка типов оплаты. Бизнес-класс мог бы использовать типы оплаты непосредственно. Так почему бы ему не сделать это? Потому что место хранения данных или способ их хранения не влияют на то, каким образом внешний мир захочет с ними взаимодействовать. Поэтому мы переложим эту обязанность на класс данных. Еще один ключ к тому, почему получение списка является обязанностью класса данных — это опять-таки понятие текстуры. Изменение способа хранения не обязательно потребует чего-то большего, нежели создание нового класса данных, способного перебросить мост через пропастью между ожиданиями бизнес-класса и тем, как данные хранятся физически.

Разделяй и властвуй

На данный момент у нас есть, по крайней мере, две необходимых вещи: один бизнес-класс и один класс данных. Поскольку целью является повторное использование, я делаю предположение, что в конечном итоге у меня будет несколько таких классов для каждого типа. Следовательно, я начну с создания абстрактного бизнес-класса и абстрактного класса данных в дополнение к классам типов оплаты.

Замечание: простоты ради, я опустила уровень класса данных, который обычно присутствует. Примеры начинаются с базового FoxPro-класса.

Как правило, у меня есть иерархия следующего вида: 1) абстрактный класс данных; 2) абстрактный класс FoxPro-данных; и 3) класс для представления конкретного объекта внешнего мира. Первый класс создает свойства и методы, которые являются общими для обработки данных в целом, такие, например, как сохранение, добавление, редактирование, удаление, сортировка и поиск. Второй класс будет являться потомком первого, и поэтому будет иметь свойства и методы, полученные в наследство от своего родителя. Этот класс также является абстрактным, но он будет реализовывать функциональные возможности, специфичные для работы с FoxPro-данными. Например, этот класс мог бы иметь, среди прочих, свойство-алиас Functionality, которое является общим для более конкретных видов данных, таких, например, как класс, предназначенный для работы с FoxPro-таблицами, или источники данных, доступные для объектной модели ADO (которые, разумеется, могли бы быть FoxPro-таблицами). Третий класс — это представление конкретных объектов внешнего мира, например, платежи, заказчики, библиотеки, книги и так далее.

В рассматриваемом примере я пройду весь путь по формированию требований и выработке решения, начиная от класса данных вплоть до пользовательского интерфейса. В данный момент описываемого процесса принимаются несколько решений, включая следующие:

- Какой тип хранения данных должна я использовать на этом первом шаге?
- Какие функциональные возможности являются общими в требованиях к данным?
- Что я хочу реализовать сейчас и что я хочу отложить на будущее?

Конечно, для своей первой реализации я буду использовать данные в формате FoxPro. Почему нет? Мне они нравятся. Однако, я попытаюсь сдержать свой восторг, поскольку должна оставаться объективной по отношению к FoxPro, за исключением осознания того, что независимо от происхождения данных, представленные в FoxPro инструменты прекрасно работают с курсорами. Помните, ранее я упоминала о том, что класс ComboBox поддерживает многообразие форматов: от массивов до жестко запрограммированных списков и FoxPro-таблиц? Поскольку курсор является одним из поддерживаемых форматов, и поскольку курсоры легко создать из разнообразных источников, и поскольку в действительности курсоры — это именно то, с чем должны работать специально для этого созданные компоненты FoxPro, я выберу... барабанная дробь, пожалуйста... курсоры! То есть, во время работы с бизнес-уровнями и уровнями пользовательского интерфейса я обязана помнить о том, что меня не должно заботить происхождение курсора. Курсор мог бы появиться как результат использования сквозного SQL-запроса к серверу Oracle или как результат работы функции XMLTOCURSOR() применительно к текстовому файлу с XML-кодом в кодировке Unicode. (Я бы вам настоятельно рекомендовала ознакомиться со статьей Эрика Мура (Erik Moore) о передаче данных с уровня на уровень, опубликованной в декабрьском и январском номерах за 2002 и 2003 год).

Поскольку одной из моих целей, преследуемых в этой статье, является иллюстрация способа коммуникации между уровнями, я не буду описывать все те функциональные возможности, которыми мог бы обладать класс данных... или бизнес-класс при рассмотрении данной темы. Приложение к этой статье на дискете, сопровождающей журнал, включает три примера форм и библиотеку классов, которые представляют некоторые распространенные функциональные возможности. На данной стадии рекомендуется определить методы и свойства, даже если не все они

Абстракция против конкретики

Абстрактные классы определяют общие методы и свойства, а также действия группы классов. Конкретные классы реализуют конкретные функциональные возможности. Аналогичным образом, абстрактными или конкретными могут быть свойства и методы. Согласно общему практическому правилу, абстрактные методы не имеют программного кода. Абстрактное свойство может получать только пустое значение определенного для него типа: пробел для символьного типа, нуль для числовых типов и так далее. Любой программный код, размещенный в абстрактном методе, должен быть или принципиально необходимым для объекта данного типа, или таким ненужным, что его без вреда можно заменить. (В этой статье я не буду касаться вопросов защищенных и скрытых методов и свойств.)

будут реализованы немедленно. Такой подход, кстати, обеспечивает создание того, что в языках программирования C# или C++ было бы интерфейсом.

Выбрать, какие еще функциональные возможности, помимо интерфейса, должен иметь этот класс, непросто. Например, в абстрактном классе я могла бы определить метод с именем Sort(), поскольку мне известно, что это распространенное действие, которое должны поддерживать данные, но немедленная реализация этого метода не является обязательной. Для иллюстрации, я сосредоточусь на действиях по загрузке данных, которые потребуются форме (вы не забыли об этой форме?).

Другой вопрос, на который надо ответить при проектировании классов: «В каком классе будет размещаться программный код?» Может показаться очевидным, что код должен размещаться на самом верхнем из тех уровней абстракции в иерархии классов, которым необходим этот код, но для меня не всегда очевидно, что это за уровень. Я обнаружила, что для выявления наиболее подходящего уровня требуется создать и протестировать небольшую модель. Поскольку предполагается, что функциональность ядра класса данных — это загрузка данных из хранилища в объект таким образом, что эти данные становятся доступны теперь другим объектам, соответствующее обращение может находиться в абстрактном классе.

```
PROCEDURE aDataObject.INIT
   IF THIS.BUFFERING > 1
     SET MULTILOCKS ON
   ENDIF
   THIS.LoadData()
 ENDPROC
```

Обратите внимание на то, что этой процедуре неизвестно, какого типа данные загружаются: она знает только о том, что что-то загружается. Программный код, выполняющий реальную загрузку данных, содержится в конкретном классе:

Листинг 1. Метод cPayTypesDataObj.LoadData()

```
PROCEDURE cpaytypesdataobj.loaddata
   LOCAL lcOrderBy, lcAlias, lcSource
   lcOrderBy = THIS.sortby
   lcAlias = THIS.ALIAS
   lcSource = THIS.SOURCE

   SELECT ;
     TYPE, ;
     ID ;
     FROM &lcSource ;
     INTO CURSOR &lcAlias READWRITE ;
     ORDER BY &lcOrderBy

   INDEX ON ID TAG ID
   INDEX ON TYPE TAG TYPE

   THIS.MoveFirst()

   THIS.PostLoadData()
 ENDPROC
```

О методах MoveFirst() или PostLoadData() пока не беспокойтесь. Первый метод является одной из наиболее распространенных функций обработки данных, которые я могла бы выбрать для последующей реализации. Впрочем, по опыту я знаю, что мне захочется реализовать этот метод, поэтому помещаю здесь «заглушку».

Теперь класс данных может загружать данные (и обратите внимание на то, что эта функциональность может быть тщательно протестирована перед тем, как перейти к следующей задаче по программированию). Сразу же после тестирования класса данных можно протестировать бизнес-класс с теми данными, которые теперь доступны. Бизнес-объект выполняет две обязанности: получение типов оплаты и принятие решения о том, какие из них необходимо выдать на экран. Первую обязанность бизнес-класс переложит на класс данных. Итак, самому бизнес-классу необходимо знать, какой класс данных будет решать данную задачу и как к нему следует обращаться. Потребность в создании объекта данных является общей, и поэтому обращение, позволяющее это сделать, может находиться в абстрактном классе.

```
DEFINE CLASS cpaytypesbizobj AS Custom
 dataclass = ""
 dataobject = .NULL.
 dataclasslib = ""

 PROCEDURE aBizObj.INIT
   THIS.dataobject = NEWOBJECT( THIS.dataclass, ;
       THIS.dataclasslib )
 ENDPROC
```

Дать ответ на вопрос о том, какой именно класс данных следует загружать — это обязанность конкретного бизнес-класса:

```
DEFINE CLASS cpaytypesbizobj AS abizobj

   dataclass = "cPayTypesDataObj"
   dataclasslib = "SPOOPClasses.VCX"
   Name = "cpaytypesbizobj"

 ENDDEFINE
```

Обратите внимание на то, что в данном случае единственным различием между абстрактным (aBizObj) и конкретным (cPayTypesBizObj) бизнес-классами являются те значения, которые получают свойства DataClass и DataClassLib. Бизнес-объект ничего не знает о том, какой именно класс потребуется (свойства DataClass и DataClassLib могли бы получить свои значения на этапе исполнения). Абстрактный бизнес-класс знает только то, что нужен будет класс данных, и он знает о том, как его создать. Всякий раз, когда бизнес-объекту необходимо нечто, связанное с данными, он будет обращаться с такими нуждами к своему объекту данных. Например, бизнес-объект мог бы вначале проверить, достаточно ли ин-

формации для создания нового покупателя — например, обязательно должно быть указано имя покупателя. Если все в порядке, бизнес-объект обратится к методу Save() объекта данных. Заметьте, что на уровне методов и свойств различные классы (уровни) обращаются к базовым правилам — именно здесь появляется интерфейс или согласованные свойства и методы. В данном случае, бизнес-объект рассчитывает на объект данных, имеющий метод Save(), точно так же, как компонент пользовательского интерфейса — форма или командная кнопка — будет предполагать нечто подобное. Например, командная кнопка Save в примере, рассматриваемом в этой статье, имеет следующий программный код в своем методе Click():

```
THISFORM.BizObj.SAVE()
```

Метод BizObj.Save() имеет следующий программный код в своем методе Save():

```
LOCAL loError
 loError = THIS.VALIDATE()
 IF ISNULL( loError )
   *!* Не можем ли мы обработать здесь все вопросы,
   *!* относящиеся к проверке допустимости?
   RETURN THIS.DataObject.Save()
 ENDIF
```

В настоящий момент форма должна сообщить, может ли она сохранить запись; затем «слово предоставляется» бизнес-объекту. Наконец, если объекту данных повезло с записью, он выполнит реальное сохранение данных. В примере реализуется только малая доля из обычного набора функциональных возможностей, предусмотренного для обработки данных, и эта реализация не включает функцию сохранения. Так что на данный момент эта реализация является, ну да, чем-то вроде враждебного пользователю шаблона программирования.

Как насчет обещанной формы для обслуживания типов оплаты? Как она использует бизнес-класс, который в свою очередь использует класс данных, который общается с данными? Очень простая форма показана на рис. 2.

Программный код, лежащий в основе этой формы, хранит и создает необходимый бизнес-класс (cPayTypesBizObj) и затем сообщает бизнес-объекту о том, что элемент управления TextBox надо ассоциировать с чем-то, что называется “Type”. Вот этот программный код:

```
*!* SPOOPPaytypesex5.Init()
 THIS.BizObj = NEWOBJECT(THIS.BizObjClass, ;
     THIS.BizObjClassLib)
 THIS.BizObj.Bind(THIS.cPayTypesText1, "type")
```

Возвращаясь на шаг назад к бизнес-классу, метод bind связывает элемент управления с источником данных. В данном случае я исхожу из специфического для встроенного в FoxPro элемента управления предположения относительно свойства ControlSource. В действительности, бизнес-класс не должен делать никаких предположений относительно того, чем является пользовательский интерфейс. Программный код следующий:

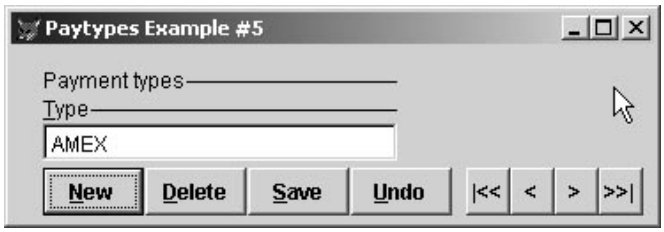

Рис. 2. Форма обслуживания типов оплаты.

```
*!* cPayTypesBizObj.Bind()
LPARAMETERS toObject, tcAttribute
toObject.CONTROLSOURCE = THIS.GET(tcAttribute)
```

Следующий шаг

Я обещала показать, каким образом типы оплаты могут оказаться «дороже денег», и как при этом проявляют себя преимущества n-уровневой архитектуры. В моем примере из реального мира заказчик, имея жестко запрограммированные типы оплаты, печатал отчеты о ежедневных поступлениях, упорядоченные по типу оплаты. Даже несмотря на то, что дебетовые карты (та песчинка, которая положила начало этой жемчужине) ведут себя в некоторых отношениях как наличные, в других случаях они подобны кредитным картам: особенно это проявляется в том, что они регистрируются. Банк равно принимает платежи по кредитной и дебетовой картам и, следовательно, денежные поступления необходимо группировать по этим категориям средств оплаты. Кроме того, не все кредитные карты устроены одинаковым образом, отсюда различные типы для карт AMEX, Novus и Visa/MC. Однако, они все-таки остаются кредитными картами. Хотя сейчас это не имеет значения, «уровневый» проект позволит нам приспособиться к возможному изменению сравнительно безболезненно. Я создала класс данных для нового типа оплаты, чтобы учесть атрибут категории. Программный код, представленный в листинге 1, меняется следующим образом:

```
LOCAL lcOrderBy, lcAlias, lcSource
 lcOrderBy = THIS.SortBy
 lcAlias = THIS.ALIAS
 lcSource = THIS.SOURCE

 SELECT TYPE, ID, ;
   PayTypeCats.Category AS Category, ;
```

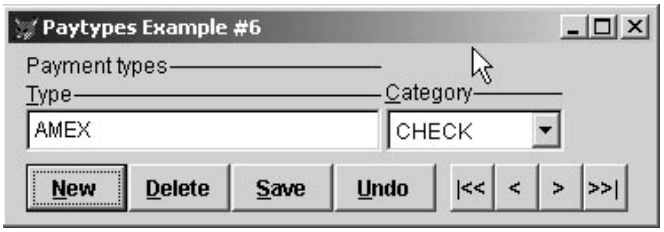

Рис. 3. Форма обслуживания типов оплаты с категориями.

```
   PayTypeCats.ID as CategoryID ;
   FROM &lcSource ;
   INTO CURSOR &lcAlias READWRITE ;
   ORDER BY &lcOrderBy

 INDEX ON ID TAG ID
 INDEX ON TYPE TAG TYPE
 INDEX ON Category TAG Category
 INDEX ON CategoryID TAG CategoryID

 THIS.MoveFirst()

 THIS.PostLoadData()
```

Исходный программный код добавляет дополнительный бизнес-класс, который специфицирует этот новый объект данных. Конечно, вместо того, чтобы добавлять новые бизнес-классы, можно было бы внести изменения в уже существующие. Однако, для лучшего понимания, я оставила весь код. Поскольку теперь типы оплаты распределены по категориям, я добавила еще один класс ComboBox, который выдает на экран категории типов оплаты. Не забыли об исходном объекте ComboBox, размещенном в форме payments и предназначенном для ввода типа оплаты? Новый класс ComboBox по существу является точно таким же — список значений для выбора. Источник значений для выбора — вот единственное, что отличает эти классы друг от друга. Форма показана на рис. 3, а программный код из метода Init() представлен здесь:

```
*!* SPOOPPaytypesex6.Init()
THIS.BizObj = NEWOBJECT(THIS.BizObjClass, ;
    THIS.BizObjClassLib)
THIS.BizObj.Bind(THIS.cPayTypesText1, "type")
THIS.BizObj.Bind(THIS.CboPayTypeCat1, "category")
```

Ужас связывания

Это та часть статьи, над которой я долго размышляла. Разумеется, хорошо заявлять о том, что элементы управления пользовательского интерфейса не общаются непосредственно с FoxPro-таблицами и будут вместо этого обращаться к бизнес-классам, которые уже общаются с данными... но в некоторый момент пользовательский интерфейс должен позволить бизнес-объекту знать, какой элемент управления какие именно данные собирает или отображает на экране. Метод Bind() является тем способом, с помощью которого я (нежестко) связала конкретный элемент управления пользовательского интерфейса (TextBox) с данными. Форма в этом примере в действительности даже не заботится о том, есть ли данные «в тылу» у какого-либо из ее элементов управления (хотя в этом было бы еще меньше интереса, чем в платежах только с плательщиком, типом и суммой). В данном случае бизнес-объект просто присваивает свойству ControlSource значение, хранящееся в поле курсора, как показано ниже:

```
LPARAMETERS toObject, tcAttribute
toObject.CONTROLSOURCE = THIS.GET(tcAttribute)
```

Желаете знать, чем это отличается от простого назначения Table.FieldName свойству ControlSource текстового поля TextBox в окне Properties конструктора Form Designer? Преимуществ, получаемых от возложения этой обязанности на бизнес-объект, несколько:

- Пользовательский интерфейс не должен использовать те же самые имена атрибутов, что и бизнес-объект.
- Если имена атрибутов (поля, например) меняются, пользовательский интерфейс не обязательно нуждается в изменении. Можно было бы «подставить» другой бизнес-объект, или метод Bind() мог бы транслировать старые имена в новые.
- Бизнес-объект использует курсор FoxPro, но ему ничего не надо знать о том, каким образом этот курсор получил свои данные. Он мог бы быть создан в результате исполнения сквозного SQL-запроса к SQL Server.

В следующий раз

Даже в простейшей форме payments встречаются любопытные трудности. Исходный код примера демонстрирует тот факт, что большая часть исходного «платежного» кода все еще встроена в форму, а поля привязаны к элементам управления. В следующий раз я покажу, как использовать те результаты, которые были получены при обработке типов оплаты, как фундамент для представления платежей (и всех других объектов реального мира).

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 (издательство EPS Publishing) и Virtual Fox User Group Newsletter и является автором книги Debugging Visual FoxPro Applications (издательство Hentzenwerke Publishing). nancy@mvps.org.

❑❑❑

Элемент управления Multi-Line ToolTip

Предраг Боснич (Predrag Bosnic)

WIN 7.0 DOWNLOAD

В этой статье Предраг Боснич использует широкие возможности версии VFP 7, чтобы продемонстрировать, каким образом вы можете использовать многострочный элемент управления Multi-Line ToolTip для разбиения на несколько строк текста подсказки максимально допустимого размера.

В Visual FoxPro есть встроенный элемент управления ToolTip, способный выдавать на экран одну текстовую строку, хранящуюся в свойстве ToolTip и имеющую максимальную длину в 127 символов (см. рис. 1). По определению предполагается, что для элемента управления ToolTip этого достаточно, но иногда на практике удачной оказывается идея продемонстрировать несколько строк текста, будучи в то же время уверенным в том, что элемент управления ToolTip не потеряет какие-либо символы из этого текста.

Анализ

На этапе проектирования я хочу, чтобы текст, используемый элементом управления ToolTip, хранился в свойстве типа string. Это означает, что максимальный размер такого текста будет равен 254 символам, поскольку это самое большее, что может содержать свойство типа string, и такой размер в два раза превышает максимальную длину текста, предусмотренную для элемента управления ToolTip, встроенного в Visual FoxPro.

Как видно на рис. 1, встроенный элемент управления ToolTip выглядит несколько странно при длине текста в 127 символов и легко может выйти за пределы экрана. (Версия 8.0 поддерживает до 4095 символов и автоматическое сворачивание текста в несколько строк — *прим. ред.*) Если я соглашусь с тем, что максимальная длина для элемента управления Multi-Line ToolTip будет равна 60 символам, тогда я могу рассчитать, что этот элемент управления будет иметь максимум пять строк.

На рис. 2 показан такой элемент управления ToolTip, каким я хочу его видеть, когда он демонстрирует текст подсказки максимально возможной длины.

Решение должно быть совместимым со всеми существующими приложениями, если они используют встроенный в Visual FoxPro элемент управления

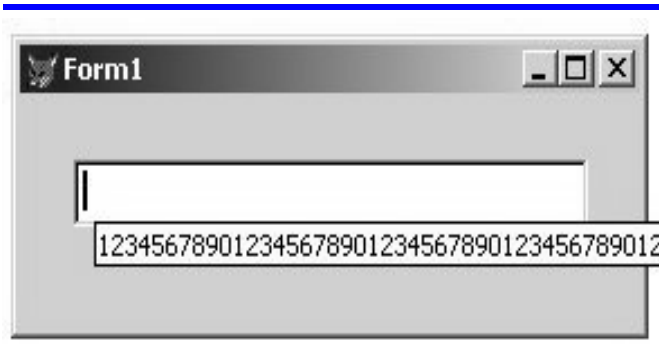

Рис. 1. Встроенный в Visual FoxPro элемент управления ToolTip.

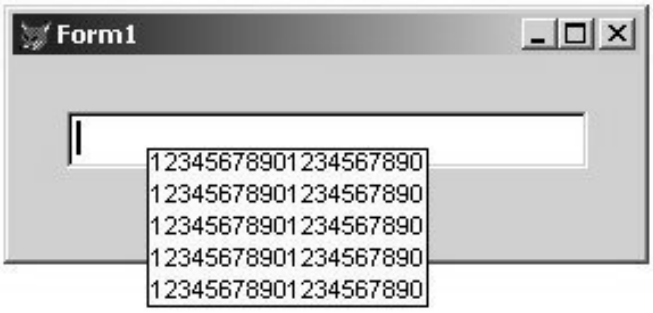

Рис. 2. Элемент управления Multi-Line ToolTip.

ToolTip; кроме того, я должен позволить разработчикам смешанное использование встроенного элемента управления и элемента управления Multi-Line ToolTip. С моей точки зрения это означает, что я не могу использовать существующие свойства ToolTipText, Tag или Comment. Текст подсказки должен быть записан в особое пользовательское свойство.

Активизация и деактивизация элемента управления ToolTip должна осуществляться на уровне формы и не должна быть связана со встроенным в Visual FoxPro свойством ShowTips.

В зависимости от положения элемента управления в форме и в зависимости от положения формы на рабочем столе поле подсказки может занимать четыре различных положения относительно основного элемента управления (см. рис. 3).

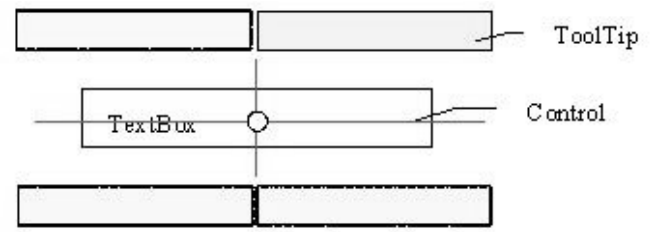

Рис. 3. Все четыре положения поля подсказки относительно основного элемента управления.

Проектные требования

Из всего того, что я выяснил при анализе, вытекают следующие проектные требования:

- Элемент управления Multi-Line ToolTip должен демонстрировать текст подсказки длиной до 254 символов.
- Максимальная длина строки должна быть 60 символов.
- Текст подсказки должен быть записан в специальном, определяемом пользователем свойстве (использование свойств ToolTipText, Tag и Comment запрещено).
- Элемент управления Multi-Line ToolTip должен активизироваться/деактивизироваться независимо от встроенного в Visual FoxPro элемента управления ToolTip.
- Этот элемент управления должен быть прост в использовании.
- Текст подсказки должен иметь свойства ForeColor и BackColor.
- Этот элемент управления должен быть совместим с существующими приложениями.

Проектирование и реализация

Давайте начнем с поля ToolTip. Это прямоугольная форма (rectangular shape), и она может появиться в любом месте на экране. По определению, поле ToolTip привязано к родительскому элементу управления и будет появляться в тот момент, когда курсор мыши указывает на этот родительский элемент управления, но при этом требуется решить проблему, связанную с определением точного местонахождения (координат) поля ToolTip.

Очень часто я размещаю элементы управления в форме близко к ее краю. Если элемент управления ToolTip активен, он может частично или полностью выйти за пределы текущей формы. Это означает, что я не могу использовать подсказку ToolTip для каких-либо элементов управления, размещенных в такой форме. Я должен использовать форму нового типа!

Давайте создадим библиотеку wbToolTip.vcx и новый класс на базе формы с именем ttScr1.

Определим следующие свойства:

```
TitleBar = 0 && (Off)
BorderStyle = 1 && (фиксированная одинарная линия)
AlwaysOnTop = .T.
```

Теперь у меня есть форма для использования элемента управления ToolTip, но как мне написать в этой форме текст подсказки? Похоже на то, что я могу использовать метод Paint и написать этот текст прямо на форме, но не забудьте, я хочу создать элемент управления Multi-Line ToolTip, поэтому взамен мне необходимо иметь возможность переносить текст на другую строку. Конечно, я не хочу это программировать, если только у меня нет другого выбора.

Для решения этой проблемы можно было бы привлечь любые элементы управления, реализованные в Visual FoxPro, которые обладают возможностью переносить текст на другую строку. Очевидные кандидаты — это элементы управления Label, TextBox и EditBox. Последний элемент управления из этого списка является наиболее приемлемым, поскольку не накладывает ограничений на размер текста, и если я в будущем приму решение увеличить размер текста подсказки, все по-прежнему будет работать.

Добавим элемент управления EditBox в форму и зададим следующие свойства:

```
BackColor = rgb(239,241,194)
DisabledBackColor = rgb(239,241,194)
DisabledForeColor = rgb(0,0,0)
FontName = Tahoma
FontSize = 9
ScrollBars = 0 (None)
SpecialEffect = 1 (Plain)
Left = -1
Top = -1
```

Значения, присвоенные двум последним свойствам, обеспечат эффект затенения (shadow), потому что эта форма ограничена одинарной линией (single line border).

Метод Init ToolTip-формы содержит следующий код:

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

 * toParentForm – ссылка на родительскую форму
 * tcToolTipText - ToolTip-текст
 * tnLeft – координата курсора мыши (xCoord)
 * tnTop – координата курсора мыши (yCoord)
 * tnForeColor – основной цвет текста
 * tnBackColor – цвет фона

 WITH thisform
   .oParentForm = toParentForm
   .nObjCenterX = tnLeft
   .nObjCenterY = tnTop
```

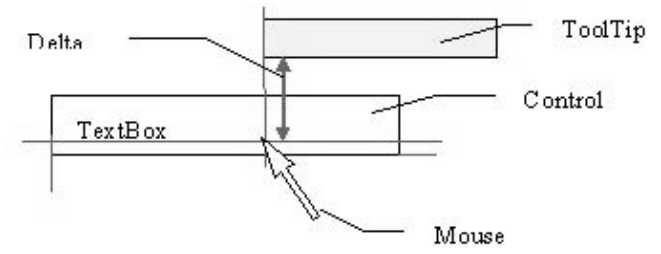

Рис. 4. Вот какое положение согласно моим требованиям должно занимать поле ToolTip относительно основного элемента управления.

```
   .comment = ALLTRIM(tcToolTipText)
   .Edit1.Value = ALLTRIM(this.comment)
   IF ! EMPTY(tnForeColor)
     .Edit1.DisabledForeColor = tnForeColor
   ENDIF
   IF !EMPTY(tnBackColor)
     .Edit1.DisabledBackColor = tnBackColor
   endif
 ENDWITH
 *
 WITH thisform
   .Left = tnLeft
   .Top  = tnTop
 ENDWITH
 *
 this.CalculatePosition()
 this.TitleBar = 0
```

Форма для демонстрации подсказки получает несколько параметров. Параметр toParentForm — это ссылка на вызывающую форму, в которой существует основной элемент управления. Я использую эту ссылку в событии Activate ToolTip-формы, чтобы передать фокус вызывающей форме. ToolTip-форма не должна сохранять фокус.

Текст подсказки будет храниться в свойстве Comment ToolTip-формы, а параметры tnLeft и tnTop запоминаются в свойствах nObjCenterX и nObjCenterY, соответственно. Параметры tnForeColor и tnBackColor используются для задания основного цвета и цвета фона ToolTip-формы, но эти установки вы, вероятно, определите только один раз — на уровне приложения.

Последняя строка в этом программном коде вызывает метод CalculatePosition (см. листинг 1), который вычисляет размер и координаты элемента управления ToolTip. На рисунке 4 показано то положение поля ToolTip, которое это поле должно занять относительно основного элемента управления.

Листинг 1. Метод CalculatePosition.

```
* CalculatePosition()
 LOCAL W, H, NoOfLines, jaDots, EH, K, strDots, ;
   Delta
 *
 NoOfLines = Int(LEN(thisForm.Comment)/60) + 1
 EH = 21 + 15*(NoOfLines-1)
 Delta = 25
 WITH thisform
   .Edit1.Height = EH
   .Height = EH -1
   IF NoOfLines = 1
    * Определить ширину формы, если
    * требуется продемонстрировать только одну строку.
    * Пока это фиксированное значение.
    * Значение jnAverageCharWidth для шрифта
    * Tahoma,9 равно 5 пикселям
     *.width = LEN(thisForm.Comment) * 5 + 15
     *
     .width = thisform.textwidth(thisform.Comment);
        + 15
     *
     .edit1.width = .width
   ENDIF
   W = .Edit1.Width
   H = .Edit1.Height
 ENDWITH
 *
 LOCAL jnX, jnY, Xmin,Xmax,Ymin,Ymax
 jnX = thisForm.nObjCenterX
 jnY = thisForm.nObjCenterY

 Xmin = W
 Xmax = _screen.Width-W
 Ymin = H
 Ymax = _screen.Height-H
 *
 Xscr = _screen.Width
 Yscr = _screen.Height
 *
 DO case
   CASE jnX > Xmax and jnY < Yscr/2
     K=3
     WITH thisForm
       .Left = jnX-W
       .Top  = jnY + Delta
     ENDWITH

   CASE jnX > Xscr/2 and jnX < Xmax and jnY < Ymin
     K=3
     WITH thisForm
       .Left = jnX-W
       .Top  = jnY + Delta
     ENDWITH

   CASE jnX > Xmin and jnX < Xscr/2 and jnY < Ymin
     K=4
     WITH thisForm
       .Left = jnX
       .Top  = jnY + Delta
     ENDWITH

   CASE jnX < Xmin and jnY < Yscr/2
     K=4
     WITH thisForm
       .Left = jnX
       .Top  = jnY + Delta
     ENDWITH

   CASE jnX < Xmin and jnY > Yscr/2
     K=1
     WITH thisForm
       .Left = jnX
       .Top  = jnY-Delta-H
     ENDWITH

   CASE jnX > Xmin and jnX < Xscr/2 and jnY > Ymax
     K=1
     WITH thisForm
       .Left = jnX
       .Top  = jnY-Delta-H
     ENDWITH

   CASE jnX > Xscr/2 and jnX < Ymax and jnY > Ymax
     K=2
     thisForm.Left = jnX-W
     thisForm.Top  = jnY-Delta-H

   CASE jnX > Xmax and jnY > Yscr/2
     K=2
     thisForm.Left = jnX-W
     thisForm.Top  = jnY-Delta-H
```

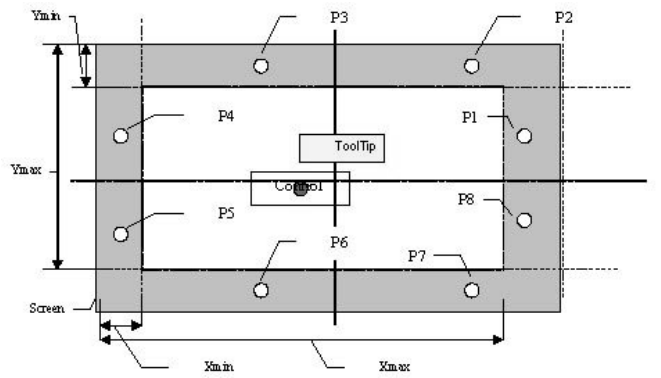

Рис. 5. P1-P8 — это типовые положения указателя мыши, использованные в методе CalculatePosition.

```
   OTHERWISE
     K=1
     WITH thisForm
       .Left = jnX
       .Top  = jnY-Delta-H
     ENDWITH
 ENDCASE
 RETURN
```

Приведенный в листинге 1 программный код отвечает за вычисление координат поля ToolTip, и это вычисление предусматривает применение нормального начертания шрифта Tahoma 9. Мне необходимо учитывать только четыре различных случая в структуре DO CASE, но для более ясного представления в приведенном программном коде учтены восемь случаев. На рис. 5 показаны положения указателя мыши, использованные в методе CalculatePosition.

Событие Activate элемента управления ToolTip содержит следующий программный код:

```
thisform.oParentform.Show()
RETURN .t.
```

Этот код вернет фокус в вызывающую форму и оставит ToolTip-форму видимой.

Давайте создадим подклассы элементов управления, встроенных в Visual FoxPro, и сохраним эти подклассы в библиотеке wb__base.vcx. Каждый элемент управления должен иметь свойство wb_cToolTipText. Метод MouseEnter каждого элемента управления содержит следующий код:

```
IF PemStatus(thisform,'wb_LshowTips',5)
  thisform.StartToolTip(this)
endif
```

Как вы можете видеть, этот программный код определяет, есть ли у формы свойство wb_lShowTips. Если у формы есть такое свойство, тогда элемент управления знает, что у такой формы есть метод StartToolTip и обращается к нему. Такая схема позволяет мне использовать размещенные в формах элементы управления, которые не поддерживают элемент управления Multi-Line ToolTip. Более надежный фрагмент программного кода выглядит вот так:

```
IF PemStatus(thisform,'wb_LshowTips',5) ;
   and PemStatus(thisform,'StartToolTip',5)
   thisform.StartToolTip(this)
endif
```

Аналогично, метод MouseLeave содержит следующий код:

```
IF PemStatus(thisform,'wb_LshowTips',5)
   thisform.KillToolTip()

 endif
```

или:

```
IF PemStatus(thisform,'wb_LshowTips',5) ;
   and PemStatus(thisform,'KillToolTip',5)
   thisform.StartToolTip(this)
 endif
```

На этот раз элемент управления ToolTip деактивизируется.

События KeyPress и MouseDown должны иметь тот же самый программный код, что и событие MouseLeave, потому что элемент управления ToolTip должен быть деактивизирован в том случае, если пользователь приложения нажимает какую-либо клавишу на клавиатуре или щелкает кнопкой мыши.

Класс wbToolTip использует в качестве базового элемент управления Timer и имеет следующие свойства:

- objRef — ссылка на объект (например, кнопка command button, список listbox и так далее).
- ToolTipForeColor — основной цвет текста подсказки.
- ToolTipBackColor — цвет фона для текста подсказки.

Листинг 2 демонстрирует программный код события Timer.

Листинг 2. Программный код события Timer.

```
* 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
endif
*
joObject = This.oObjRef
jcToolTip = Alltrim(joObject.wb_cToolTipText)

If Empty(jcToolTip)
  This.Enabled = .F.
  Return
Endif
thisform.wb_LIsToolTipActive = .t.
*
jnLeft = Mcol('',3)
jnTop  = Mrow('',3)

*--- Активизировать экран ToolTip ------------------------
Thisform.wb_oToolTip = Createobject(;
  'ttscr1',Thisform,jcToolTip,jnLeft,jnTop,;
  This.nToolTipForeColor,This.nToolTipBackColor)
Thisform.wb_oToolTip.Visible = .T.
Thisform.wb_oToolTip.Show()

*** Не "убивайте" timer в этом месте.
*** Timer должен быть активен во время сессии ToolTip!
```

Таймер настроен на задержку 0.8 секунды перед появлением подсказки. В это время пользователь может переместить мышь к следующему элементу управления или за пределы формы. Чтобы предотвратить появление ToolTip-формы в том случае, если такое произойдет, программный код использует функцию SYS(1270) для определения того, на какой объект указывает в данный момент курсор мыши.

При вычислении координат меня не волнует размер границ формы, хотя аппроксимация достаточно хорошая.

В библиотеке Wb__base.vcx определен класс формы _formTT, у которого есть все свойства и методы, необходимые для поддержки базовых элементов управления, использующих элемент управления Multi Line ToolTip. Вот эти свойства:

```
Wb_nToolTipType = 0
Wb_IsToolTipActive = .T./.F.
Wb_LShowToolTip = .T./.F., главный флажок MultiLine ToolTip
Wb_oToolTip = ссылка на элемент управления ToolTip
```

Этот класс имеет также методы StartToolTip и KillToolTip. Вот метод StartToolTip:

```
Lparameters toControlReference
 Do Case
   Case Thisform.wb_nToolTipType = 0  && multi-line tooltip
     If Thisform.wb_lShowTips = .T.
       With Thisform.wbToolTip
         .oObjref = toControlReference
         .Enabled = .T.
       Endwith
     Endif

   Otherwise
     Return
 Endcase
```

Метод StartToolTip активизирует элемент управления Timer, размещенный в форме и названный wbToolTip. Метод Timer сделает все остальное. Обратите внимание на предложение CASE. В нем учтено только одно условие ToolTipType = 0. Это позволяет мне реализовать другие типы элементов управления ToolTip путем дополнения конструкции CASE.

Метод KillToolTip содержит следующий программный код:

```
If Thisform.wb_lShowTips = .T.
   If Type('thisform.wb_oToolTip')='O' ;
     And !Isnull(Thisform.wb_oToolTip)
     Thisform.wb_oToolTip.Release()
   Else
     Thisform.wb_oToolTip = .F.
   Endif
 Endif
```

Собрав все это воедино, нетрудно воспользоваться элементом управления Multi-Line ToolTip. Ваша форма должна использовать в качестве базового класс _FormTT. Этот класс уже содержит элемент управления wbToolTip. Поместите в форму любые необходимые вам элементы управления, взяв их из библиотеки классов wb_base.vcx. Для каждого размещенного в форме элемента управления вы должны присвоить значение свойству wb_ctooltiptext и, наконец, присвоить свойству wb_LShowTips (формы) значение истины True.

Когда вы подводите курсор мыши к элементу управления, в свойстве cToolTipText которого хранится некоторое значение, событие MouseEnter инициирует элемент управления ToolTip, чей метод Timer выдает на экран подсказку, если только внутренний флажок этого элемента управления не сигнализирует о том, что он уже активен. После задержки (по умолчанию она составляет 0.8 секунды, но это значение может быть определено программистом) на экран выводится текст подсказки, и при этом запускается таймер, размещенный в форме ToolTip. Если пользователь не перемещает указатель мыши в течение четырех секунд, этот таймер «убьет» элемент управления ToolTip.

Когда курсор мыши «покидает» данный элемент управления, событие MouseLeave элемента управления вызовет метод KillToolTip формы. Этот метод уничтожит элемент управления ToolTip.

В кратком изложении точная последовательность событий такова:

- 1. Пользователь подводит курсор мыши к какому-то элементу управления.
- 2. Метод Control.MouseEnter() обращается к методу Form.StartToolTip(this).
- 3. Метод Form.StartToolTip() делает доступным элемент управления Form.wbToolTip.
- 4. Метод Form.wbToolTip.Timer() тестирует состояние флажка и создает окно подсказки.
- 5. Пользователь читает текст подсказки.
- 6. Пользователь убирает курсор мыши с этого элемента управления.

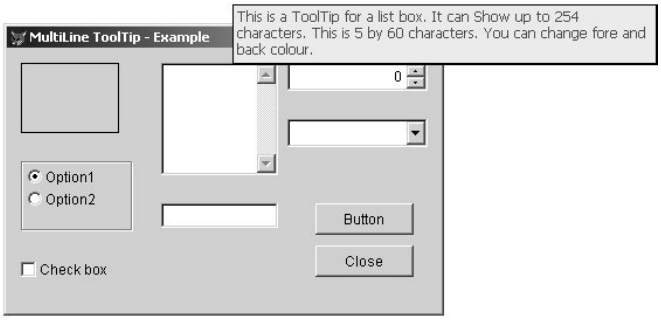

Рис. 6. Элемент управления Multi-Line ToolTip, основной пример.

- 7. Событие MouseLeave() вызывает метод Form.KillToolTip().
- 8. Метод Form.KillToolTip() закрывает ToolTip-форму.

Рисунок 6 демонстрирует работающий элемент управления.

Библиотека wbToolTip.vcx

Эта библиотека содержит два класса, wbToolTip и ttScr1. Класс wbToolTip должен быть добавлен в форму. Все элементы управления, размещенные в такой форме, поддерживают элемент управления Multi-Line ToolTip.

Класс wbToolTip

В таблице 1 дано описание свойств класса wbToolTip.

Таблица 1. Свойства класса wbToolTip.

Имя	Тип	Описание
oObjRef	Obj	Ссылка на объект (например, кнопка, список list box и так далее).
nToolTipBackColor	Int	Цвет фона.
nToolTipForeColor	Int	Основной цвет.

Класс ttScr1

В таблицах 2 и 3 описываются свойства и метод класса ttScr1.

Таблица 2. Свойства класса ttScr1.

Имя	Тип	Описание
nObjCenterX	Int	Для внутреннего использования. Положение указателя мыши, координата X.
nObjCenterY	Int	Для внутреннего использования. Положение указателя мыши, координата Y.
oParentForm	Int	Ссылка на родительский объект формы.

Таблица 3. Метод класса ttScr1.

Имя	Тип	Описание
CalculatePosition	Private	Для внутреннего использования. Вычисляет положение и размер окна ToolTip.

Библиотека Wb_base.vcx

Эта библиотека содержит базовые элементы управления, реализованные в версии Visual FoxPro 7. Все визуальные элементы управления могут использовать элемент управления Multi-Line ToolTip. _FormTT — это класс формы, и в нем определены все свойства и методы, необходимые для поддержки элемента управления Multi-Line ToolTip.

Класс ttForm

Методы и свойства класса _ttForm описаны в таблице 4 и 5.

Таблица 4. Свойства класса _ttForm.

Имя	Тип	Описание
Wb_nToolTipType	Int	Используемый тип ToolTip: 0 (по умолчанию).
Wb_LShowTips	Int	Главный флажок ToolTip. Если имеет значение .T., функциональная возможность ToolTip активизирована.
Wb_oToolTip	Obj	Ссылка на объект ToolTip.

Таблица 5. Методы класса _ttForm.

Имя	Тип	Описание
StartToolTip	Private	Активизирует ToolTip.
KillToolTip	Private	Деактивизирует ToolTip.

Заключение

Этот пример неплох для начала, но рассмотрите следующие возможные направления его усовершенствования: более элегантный и ясный программный код, увеличение размеров окна ToolTip и текста, использование технологии IntelliSense.

В следующий раз я представлю функции интерфейса Windows API и изменю форму элемента управления Multi-Line 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. misobosnic@aol.com.

❑❑❑

Сохраняй терпение

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

WIN 7.0 DOWNLOAD

В этом месяце Энди Крамек и Марсиа Акинз завершат свое исследование модели событий VFP 7.0, начатое ими несколько месяцев назад с рассмотрения форм. На этот раз они изучат классы контейнеров, начиная со страничных блоков и страниц, а затем займутся и сетками.

Марсиа: Несомненно, в процессе написания этой серии статей я узнала много. По-видимому, и порядок срабатывания событий не настолько однозначен, как мы думали прежде. Хотя набор событий всегда один и тот же для любого элемента управления, их последовательность зависит от того, используется ли для перемещения между элементами управления клавиатура или мышь.

Энди: Я согласен. Теперь я понимаю, почему людям доставляют столько хлопот такие элементы управления, как раскрывающиеся и обычные списки. Их поведение, по крайней мере, может сбить с толку.

Марсиа: Я надеюсь, что наше сегодняшнее изучение контейнеров принесет еще более поразительные результаты, так что давай продолжим.

Энди: Да, дорогая. Я полагаю, что ты хочешь начать с объектов grid?

Марсиа: Как это ни странно, нет. Думаю, что было бы логичнее начать со страничных блоков и страниц. А уже потом переходить к grid.

Энди: Обсуждая применение модели событий к страницам и страничным блокам, будет правильным начать с события UIEnable, поскольку оно срабатывает для каждого элемента управления, содержащегося на странице, независимо от того, активирована страница или нет.

Марсиа: Хорошо, но почему страничный блок содержит событие UIEnable? Насколько я могу судить, это событие не запускается никогда и ни при каких условиях!

Энди: Это хороший вопрос. Он настолько хорош, что я даже не способен ответить на него!

Марсиа: В самом деле? Тогда давай посмотрим, сможешь ли ты ответить на следующий. Ты говорил, что событие UIEnable для каждого элемента управления срабатывает независимо от того, активирована страница или нет. Это не совсем точно. Если на странице (неважно, активирована она или нет) есть grid, событие UIEnable срабатывает только для этого объекта. Это событие никогда не срабатывает для элементов управления, содержащихся в столбцах grid.

Энди: Кхм... Возможно, это происходит потому, что элементы управления, содержащиеся в grid, принадлежат ей, а не странице, на которой находится этот объект. Но будет ли то же верно для других элементов управления, находящихся вне контейнеров? Возможно, UIEnable срабатывает только для самого внешнего слоя контейнера?

Марсиа: Нет, дело не в этом. События UIEnable для всех элементов управления в любом другом контейнере срабатывают до события UIEnable для самого контейнера. (Точно также как для события Init! Сначала для элементов управления, а потом для контейнера). Элементы grid точно также принадлежат этому объекту, как элементы в любом другом контейнере принадлежат этому контейнеру. Иерархия объектов не отличается.

Энди: Это на самом деле странно! Но сетка не обычный класс контейнера, поэтому я предлагаю взять на заметку это обстоятельство, как еще одно отличительное свойство сетки. Но на этом месте у нас возникает еще одна тема. Дело в том, что UIEnable является одним из тех часто забываемых событий, которые могут быть использованы для значительного улучшения производительности в приложении.

Марсиа: Да?! Как же это?

Энди: Представь себе, что первая страница в страничном блоке используется для ввода критериев фильтрации, применяемых при выборке некоторых данных из базы данных и отображения их в Grid на второй странице. Если мы вызовем метод, повторно запрашивающий данные из метода Activate() второй страницы, тогда этот код будет запускаться каждый раз при активации страницы. Даже если мы переместили фокус куда-нибудь еще (например, чтобы показать модальную форму) и мгновенно вернулись на ту же страницу. Это может создавать значительные издержки.

Марсиа: Хорошо, но ты можешь достаточно просто обойти это обстоятельство, включив в метод Activate() код, проверяющий наличие совпадений текущих данных с записью, определенной свойством RecordSource сетки.

Энди: Да, но ты можешь полностью избежать этой проблемы, если используешь событие UIEnable() сетки вместо метода Activate() страницы.

Марсиа: Я полагаю, что сейчас стоит упомянуть о том, что этот метод получает единственный параметр, lEnable, имеющий значение True, когда страница активирована, и False, когда она деактивирована.

Энди: Хорошее замечание. Мы должны также подчеркнуть, что UIEnable не сработает до тех пор, пока активная страница не изменится. Например, при инициализации формы со страничным блоком сработает событие Activate для первой страницы, но UIEnable для содержащихся в ней элементов не сработает, поскольку активная страница не изменилась.

Марсиа: Рада, что ты упомянул это обстоятельство, потому что оно может сбить с толку. Итак, давай пойдем дальше и посмотрим, что случится, когда мы переходим на новую страницу.

Энди: Я полагаю, что первым сработает событие Valid для текущего элемента на текущей странице (если этот элемент имеет метод Valid), а за ним следует его событие LostFocus.

Марсиа: Да, это достаточно просто. Далее сработает событие Deactivate текущей страницы, а следом UIEnable для всех содержащихся в ней элементов.

Энди: Это именно то, что я и ожидал.

Марсиа: В заключение сработает Activate новой страницы, а следом события UIEnable содержащихся на ней элементов. Похоже, здесь нет никаких сюрпризов. Можно теперь переходить к контейнеру?

Энди: Хорошо, мы так и сделаем. Итак, когда мы используем для перемещения клавиатуру, последовательность событий в контейнере будет следующей:

- 1. Событие When для первого содержащегося в нем элемента.
- 2. Событие GotFocus для контейнера.
- 3. Событие GotFocus для первого содержащегося в нем элемента.

Марсиа: Это не очень ясно. Почему ты полагаешь, что событие When первого элемента сработает прежде, чем GotFocus контейнера?

Энди: На самом деле, это совершенно ясно. Поскольку контейнер не содержит метода When, нет никакого способа узнать, получит ли контейнер когда-либо фокус. Если все содержащиеся в нем элементы отключены или если они возвращают значение false из своего метода When, тогда GotFocus для контейнера никогда не сработает.

Марсиа: Ну конечно! Я также обратила внимание, что когда я перехожу, используя табуляцию, с последнего элемента контейнера, событие When следующего элемента по порядку перехода срабатывает прежде, чем LostFocus контейнера. Это не совпадает с порядком, который мы наблюдаем для других элементов управления. Когда мы используем клавиатуру для навигации между ними, LostFocus текущего элемента срабатывает прежде, чем When элемента, к которому мы переходим. Можешь ли ты предположить, почему контейнер ведет себя по-другому, чем, скажем, окно для ввода текста?

Энди: Я могу только предположить, что это происходит потому, что когда мы говорим о контейнерах, содержащихся в формах, мы говорим более чем о простом переходе при помощи табуляции. Пока контейнер находится в фокусе, Visual FoxPro использует этот порядок перехода внутри контейнера. Но когда последний по порядку табуляции элемент управления в контейнере теряет фокус, VFP выполняет дополнительную работу по восстановлению порядка табуляции для контейнера, содержащего этот контейнер.

Марсиа: Кхм... Я думаю, это несколько расплывчато, но, поскольку я не встречала лучшего объяснения, я думаю, что приму его. Может быть, мы сможем спросить у команды Fox, есть ли другие причины для такого поведения.

Энди: Итак, есть ли какая-либо разница, если для перехода «в» и «из» контейнера используется мышь?

Марсиа: Конечно! Но на этот раз этот порядок согласуется с нашим заключением для других элементов управления. Когда мы щелкаем мышью элемент в контейнере, события срабатывают в следующей последовательности:

- 1. Событие When для элемента в контейнере, который щелкнули мышью.
- 2. Событие LostFocus для элемента, который ранее был в фокусе.
- 3. Событие GotFocus для контейнера.
- 4. Событие GotFocus для заключенных элементов.

Энди: Обошлось без сюрпризов. Я полагаю, что когда мышь используется для перемещения из контейнера, мы столкнемся с подобным развитием событий:

- 1. Событие When для элемента, который щелкнули мышью.
- 2. Событие LostFocus для элемента в контейнере, который ранее был в фокусе.
- 3. Событие LostFocus для самого контейнера.

Марсиа: Думаю, в этом ты прав! Действительно нет ничего удивительного в порядке срабатывания событий для контейнеров. Я бы удивилась, если бы мы увидели столь предсказуемое поведение для сеток.

Энди: Единственная вещь, которую я могу предсказать для сеток, это то, что они непредсказуемы!

Марсиа: Это только потому, что ты не используешь их. На самом деле, порядок срабатывания событий для сеток очень даже предсказуемый! Например, вот что происходит, когда сетка первый раз получает фокус:

- 1. Срабатывает событие When для сетки.
- 2. Срабатывает событие When для свойства CurrentControl в первом столбце.
- 3. Срабатывает событие GotFocus для CurrentControl в ActiveColumn.
- 4. Срабатывает событие сетки AfterRowColChange.

Энди: Давай остановимся здесь! Я вижу, что событие AfterRowColChange сетки срабатывает независимо от того, получает ли сетка фокус. Это означает, что любой код метода, вызванный из события AfterRowColChange сетки, будет выполняться не только когда пользователь переходит на новую строку или столбец, но также и при других обстоятельствах.

Марсиа: Именно так. Но это абсолютно понятно для меня, потому что свойства ActiveRow и ActiveColumn сетки значимы только тогда, когда сетка находится в фокусе. Если мы устанавливаем фокус на сетку, у нас еще нет текущей строки, поэтому событие BeforeRowColChange не может сработать. Но когда мы, в конце концов, устанавливаем на ней фокус, у нас появляется текущая строка. Следовательно, по определению, мы изменяем строки. Что и требовалось доказать!

Энди: Итак, кому-то еще не ясно? То есть, ты говоришь, что событие AfterRowColChange сетки срабатывает потому, что была изменена строка. А как же насчет столбцов?

Марсиа: Когда сетка получает фокус, ее свойство ActiveColumn стремится перейти к тому столбцу, который был активен, когда сетка ранее фокус потеряла. Когда вы первый раз переходите при помощи клавиши табуляции на сетку, свойство CurrentControl первого столбца получает в соответствие значение, возвращаемое ActiveCell. Представим, что потом вы переходите на третий столбец сетки, прежде чем щелкните другой объект формы. Когда в следующий раз сетка получит фокус, свойство ActiveColumn, по умолчанию, будет указывать на третий столбец.

Энди: Это согласуется с тем, что я видел. И вот что происходит, когда сетка теряет фокус:

- 1. Срабатывает событие Valid для сетки.
- 2. Срабатывает событие BeforeRowColChange для сетки.
- 3. Срабатывает событие Valid для CurrentControl в ActiveColumn.
- 4. Срабатывает событие LostFocus для активной ячейки в сетке.

Марсиа: Обрати внимание на то, что когда перемещаешься с сетки на другой объект той же формы, событие Valid для сетки срабатывает прежде, чем событие Valid для активированной ячейки. Это явная ошибка, которая может сбить с толку, когда фокус перемещается между сеткой и другим объектом. Как ты и предполагал, именно метод Valid определяет, может ли сетка потерять фокус. Однако, так как событие Valid для сетки вызывается прежде, чем для любого другого содержащегося в ней элемента управления, этот элемент не может предохранить сетку от потери фокуса, даже если его собственное подтверждение провалилось.

Энди: Это на самом деле неприятно, поскольку позволяет ввести в сетку, а значит и в вашу базу данных, неправильные данные. Единственный путь избежать этого состоит в том, чтобы убедиться, что вы явно вызвали метод Valid для любого элемента управления, который принадлежит событию Valid сетки. Это позволит вам проконтролировать результаты проверки достоверности размещенных элементов и действовать соответственно. Таким образом, если элемент в первом столбце сетки называется “Text1” и содержит код в своем Valid, следующая строка в методе Valid сетки позаботится об этой проблеме:

```
RETURN This.Column1.Text1.Valid()
```

Марсиа: Все это означает для нас необходимость исследовать порядок срабатывания событий для объектов, содержащихся в сетке, когда сетка находится в фокусе. Вы можете быть удивлены, увидев, что она абсолютно предсказуема и выглядит именно так, как мы и предполагали. Вот как все происходит, когда мы используем клавиатуру:

- 1. Срабатывает событие KeyPress для CurrentControl в ActiveColumn.
- 2. Срабатывает событие BeforeRowColChange сетки.
- 3. Срабатывает событие Valid для CurrentControl в ActiveColumn.
- 4. Срабатывает событие LostFocus для активной ячейки.
- 5. Срабатывает событие When для CurrentControl в столбце, на который мы переместились.
- 6. Срабатывает событие GotFocus для новой активной ячейки.
- 7. Срабатывает событие AfterRowColChange сетки.

Энди: Похоже, что использование мыши также не приводит к появлению отличий. Когда вы применяете мышь для перемещения между ячейками в сетке, очевидно, что событие KeyPress не срабатывает. Остальные события срабатывают абсолютно в том же порядке, что и при использовании клавиатуры. Однако, когда используешь мышь, событие Click вновь активированной ячейки срабатывает немедленно после события AfterRowColChange сетки. Это сюрприз! В некотором отношении порядок срабатывания событий более предсказуем для сеток, чем для других элементов!

Марсиа: Могу поспорить, ты никогда бы не поверил в это, если бы не увидел собственными глазами. Конечно, мы не пытались добавить контейнер к столбцу сетки или разместить сетку внутри сетки. Определенно, при этом вещи будут выглядеть более сложно, поскольку в дело вовлечены дополнительные уровни. Используя только три столбца, мы сумели показать лишь основы. Более сложные случаи мы оставляем в качестве упражнений для наших читателей.

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

❑❑❑

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 05/06/03. Формат 60x90 1/8. Тираж 330 экз.