FoxTalk

Февраль 2003
№ 2 (68)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

Создание CHM-файлов из Word-документов, часть 1

Дуг Хенниг (Doug Hennig)

Некоторые инструментальные средства, например West Wind HTML Help Builder, могут сократить объем работы по созданию справочных Help-файлов, представленных в формате HTML (CHM). А что если вы начинаете эту работу не на пустом месте, а уже имея в своем распоряжении готовые Word-документы? В данной серии статей, состоящей из двух частей, Дуг Хенниг рассматривает основные моменты процесса создания CHM-файлов (в том числе те задачи, которые необходимо решить при формировании CHM-файлов из документов, представленных в формате текстового процессора Word) и знакомит вас с набором инструментов, предназначенным для автоматизации этого процесса.

материал имеет отношение к соответствующей версии

материал имеет отношение к соответствующей платформе

исходные тексты программ можно скачать из Интернета

В своей статье, опубликованной в июле 2000 года («Живая голая документация!»), я рассмотрел HTML Help Builder фирмы West Wind Technologies — инструмент, которым я обычно пользуюсь при создании для разрабатываемых мной приложений справочных Help-файлов, представленных в формате HTML (CHM). Построитель HTML Help Builder обеспечивает среду разработки, в которой предусмотрены все необходимые для создания CHM-файлов средства, включая возможности по редактированию и организации тематических разделов, генерации HTML-кода и CHM-компиляции (с использованием компилятора HTML Help фирмы Microsoft).

Вместе с тем, однажды я оказался в ситуации, когда использование HTML Help Builder, по сути дела, привело бы не к меньшим трудозатратам, а к необходимости проделать дополнительную работу: это был тот случай, когда документы-источники уже существовали в виде файлов текстового процессора Word. Да, HTML Help Builder поддерживает режим «перетащить-и-бросить» для текстового редактора Word, но если количество ваших документов исчисляется десятками или даже сотнями, использовать такой способ просто нереально. Дело обстояло следующим образом: недавно я (вместе с Тэймар Гранор (Tamar Granor), Деллой Мартин (Della Martin) и Тедом Роше (Ted Roche)) завершил для издательства Hentzenwerke Publishing работу над книгой «The Hacker's Guide to Visual FoxPro 7.0» (известной миллионам читателей под названием «HackFox»). Помимо авторства, я также отвечал за создание CHM-файла. Учитывая, что в качестве отправной точки в наличии имелось больше 900 документов в формате Word, и необходимо было

сформировать два CHM-файла (бета-версию и окончательный вариант), мне необходимо было автоматизировать этот процесс.

Даже если бы исходные документы еще не были оформлены в виде документов Word, то по целому ряду причин у вас могло возникнуть желание воспользоваться для создания CHM-файла теми средствами, которые обеспечивает текстовый процессор Word, а не возможностями HTML Help Builder. Во-первых, приложение Word предоставляет более удобную среду редактирования, поскольку является полнофункциональным текстовым процессором, а не просто элементарным редактором текста. Например, в текстовом процессоре Word реализованы проверка орфографии и грамматики и отслеживание вносимых в документ изменений, т. е. те функции, которые обычно необходимы при создании документов подобного типа. Во-вторых, для авторов документов использование Word, несомненно, является более привычным и удобным. В-третьих, вам может потребоваться как документация, используемая в режиме online, так и документация, представленная в печатном виде. Если вы создаете тематические разделы, используя HTML Help Builder, ваши возможности по созданию печатной документации весьма ограничены.

Вообще говоря, я и прежде сталкивался с этой проблемой. Несколько лет тому назад у меня был заказчик — объединение кредитных союзов (Union of Credit Unions) — который хотел распространить «образец» страхового справочника (policy handbook) среди входящих в это объединение кредитных союзов. Для использования подобного образца требовался способ, позволяющий таким образом его модифицировать, чтобы в результате получить свой собственный справочник, специфичный для данного кредитного союза. Мои заказчики хотели модифицировать образец средствами текстового процессора Word, однако, конечный результат необходимо было получить в виде CHM-файла — ради тех преимуществ, которые обеспечивает справочный Help-файл, представленный в формате HTML: один сжатый, легко распространяемый файл, а также автоматически создаваемая таблица содержания, индекс и возможности поиска (нечто такое, что было бы весьма затруднительно реализовать для набора Word- или HTML-документов). Процедура, которую я для них разработал, была аналогична процедуре, использованной мной в работе над книгой HackFox, хотя теперь в программном коде осталось гораздо меньше погрешностей.

При создании CHM-файла из набора Word-документов решаются две задачи. В данной статье внимание сфокусировано на первом шаге — преобразовании Word-документов в HTML-код. В следующем месяце я буду рассматривать второй шаг — формирование CHM-файла из HTML-документов.

Преобразование Word-документов в формат HTML: Фу-у!

Вы, вероятно, думаете: «Что ж, в этом месяце статья будет короткой. В конце концов, текстовый процессор Word может генерировать HTML-файлы по одному щелчку мыши». Разумеется, может. Давайте взглянем на HTML-код, сформированный процессором Word.

Вот начало HTML-файла, созданного текстовым процессором Word в результате преобразования документа, предназначенного для данной статьи:

```
<html xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">

<head>
<meta http-equiv=Content-Type content="text/html;
charset=windows-1252">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 9">
<meta name=Originator content="Microsoft Word 9">
<link rel=File-List href="./html_files/filelist.xml">
<link rel=Edit-Time-Data href="./html_files/editdata.
mso">
<title>Creating CHM Files From Word Documents-Part 1
</title>
<!--[if gte mso 9]><xml>
 <o:DocumentProperties>
  <o:Author>Doug Hennig</o:Author>
  <o:Template>ft951021.dot</o:Template>
  <o:LastAuthor>Doug Hennig</o:LastAuthor>
```

Придется просмотреть еще много-много страниц, заполненных такого рода вещами, прежде чем вы увидите текст самого документа. Фактически, по моим подсчетам, тексту в этом файле предшествует 48 788 символов! Конечно, размер такой «преамбулы» в разных документах разный; это зависит от сложности документа и от того, сколько стилей в нем используется.

Так, давайте посмотрим, на что похож сам текст. Вот как представлено первое предложение в следующем разделе:

```
<p class=bulletfirst><![if !supportLists]><span
style='font-family:Symbol'>p<span
style='font:7.0pt "Times New
Roman"'>       
</span></span><![endif]>Convert all the Word documents in
a specified directory
to HTML files in a different directory.
```

HTML-код поддерживает абзацные отступы, используя для этого тэги <UL> и <LI>. Но что здесь делают все эти конструкции <![IF>, <SPAN> и ?

Из-за того, что тексту предшествует HTML-код, а также из-за всех этих вставленных в текст тэгов,

для представления текстового файла размером 28K формируется HTML-файл размером 99K!

Не будем слишком суровы по отношению к Word. В конце концов, вы же не рассчитываете, открывая HTML-документ в Word, потерять при этом какую-то относящуюся к этому документу информацию: например, его свойства, стили и форматирование. Весь этот «лишний» HTML- и XML-код необходим для сохранения такого рода данных. Однако, с точки зрения того, что нам необходимо с этим документом сделать, подобный код — без всякой на то необходимости — увеличивает размер файлов.

Вот еще один вопрос: в HTML-представлении документа мне могут понадобиться стили, отличные от тех, которые были использованы при работе в Word. Мне, например, кажется, что если использовать фон светлых тонов, то тематический раздел в Help-справке будет иметь привлекательный вид и оставлять впечатление профессионально сделанной работы. Однако, мне не обязательно хотелось бы наблюдать такой же фон при редактировании данного документа в Word. Возможно также, что мне захочется редактировать документ, используя шрифт Times New Roman, но я предпочту использовать шрифты Tahoma или Verdana в CHM-файле. К сожалению, внести подобные изменения в готовый HTML-документ не так просто. Поэтому, помимо прочего, мне необходимо удалить из HTML-кода все стили, созданные процессором Word, и подключить свою собственную каскадную таблицу стилей (CSS).

Стриптиз: удовольствие и польза

Чтобы автоматизировать процесс удаления всех посторонних вещей из HTML-кода, сформированного Word, я создал набор классов, пару используемых для организации этой процедуры таблиц и программу, управляющую решением поставленной задачи. Давайте начнем с управляющей программы PROCESS.PRG.

Вот общая последовательная схема того, что делает эта программа:

- Преобразование всех хранящихся в указанном каталоге Word-документов в HTML-файлы, которые помещаются в другой каталог. Размещая файлы в различных каталогах, вы можете (в том случае, если процедуру необходимо повторить заново) быстро удалить все сформированные HTML-файлы, не беспокоясь при этом о том, как бы случайно не удалить исходные Word-документы. Кроме того, если необходимо выполнить «тонкую» настройку процедуры удаления, вы можете пропустить тот шаг, на котором формируется HTML-код (и который требует наибольших затрат времени), и просто повторно обработать уже имеющиеся HTML-файлы.
- Удаление из HTML-файлов нежелательных вещей в результате приводит к появлению новых файлов еще в одном каталоге. Вместо того, чтобы жестко программировать то, что вы собираетесь искать и удалять, такие данные хранятся в таблице. И опять-таки, создание новых файлов вместо перезаписи исходных Word-файлов позволяет выполнить процедуру повторно без необходимости заново формировать HTML-файлы средствами текстового процессора Word.
- Дополнительная, «настроечная», обработка HTML-файлов. После использования предлагаемой процедуры в нескольких проектах, я обнаружил, что различные наборы файлов выдвигают различные требования к обработке. Например, большинство тематических разделов в книге HackFox имеют подраздел «see also», который принадлежит другим темам и который необходимо связать с соответствующими HTML-файлами. Во время обработки моей статьи, предназначенной для журнала FoxTalk, мне необходимо было удалить размещенные в начале текста указания издателя. Следовательно, этим шагом управляют данные, для чего используется таблица, в которой определяется вся необходимая дополнительная обработка.

Замечаете сходство между этими двумя шагами? Возьмите набор файлов; сделайте что-нибудь с каждым из них; и пошлите выходные результаты в другой файл. По-видимому, для такого случая наилучшим образом подходит шаблон проектирования Iterator. У нас будет один объект, который будет последовательно перебирать файлы из указанного набора и обращаться к другому объекту, чтобы одинаковым образом обработать каждый файл.

Для определенного в визуальной библиотеке ASSEMBLE.VCX класса FileIterator в качестве базового используется класс Custom. У класса FileIterator есть свойства cDirectory, cFileSkeleton и aExclude, с помощью которых вы определяете исходный каталог, шаблон имен для файлов из обрабатываемого набора и расширения имен для тех файлов (если они есть), которые следует исключить из процесса обработки (например, свойство cFileSkeleton может иметь значение “*.*”, а свойство aExclude может иметь значение “GIF”, “JPG” and “TIF”, так что будут обработаны все файлы, за исключением графических). В свойстве cWriteDir хранится имя того каталога, в котором следует разместить выходные

файлы. Поскольку обработке подвергается большое количество файлов, я не хочу, чтобы эта работа прерывалась появлением диалоговых окон с сообщениями о возникших ошибочных ситуациях, поэтому свойство cLogFile обеспечивает возможность указать тот файл, в который следует записывать все подобные сообщения. Важно просмотреть содержимое этого файла по окончании процедуры, чтобы узнать, какие Word-документы необходимо откорректировать или какие процессы необходимо отрегулировать для устранения возникших непредвиденных затруднений. Метод GetFiles заполняет свойство-массив aFiles именами файлов, подлежащих обработке; он использует функцию ADIR() в совокупности со свойствами cDirectory и cFileSkeleton для составления исходного списка файлов, а затем удаляет из массива все те файлы, чьи расширения обнаружены в свойстве aExclude, и сортирует полученный в результате массив.

Программа Process последовательно перебирает перечисленные в массиве aFiles файлы, обращаясь к методу ProcessFile для каждого из них, чтобы выполнить необходимую обработку. Эта программа выдает на экран окно индикатора прогресса с командной кнопкой Cancel (создается экземпляр объекта класса SFProgressForm, определенного в визуальной библиотеке SFTHERM.VCX, а полученная объектная ссылка запоминается в свойстве oTherm), так что мы можем наблюдать за ходом процесса и прервать его в случае необходимости. Вот код из программы Process:

```
lparameters tcTitle
 local lcPath, ;
   llReturn, ;
   lnI, ;
   lcFile
 private plCancel
 with This

 * Используем для записи файлов тот же самый каталог, если
 * не был определен другой каталог.

   .cWriteDir = iif(empty(.cWriteDir), .cDirectory, ;
     .cWriteDir)

 * Создание объекта индикатора прогресса.

   .nFiles = alen(.aFiles, 1)
   lcPath  = sys(16)
   lcPath  = addbs(justpath(substr(lcPath, ;
     at(' ', lcPath, 2) + 1)))
   .oTherm = newobject('SFProgressForm', ;
     lcPath + 'SFTherm.vcx')
   .oTherm.SetMaximum(.nFiles)
   .oTherm.SetTitle(tcTitle)
   .oTherm.cCancelProperty = 'plCancel'

 * Обработка каждого файла. При возникновении ошибки,
 * записать информацию о ней
 * в журнальный log-файл и выставить флажок,
 * указывающий на то, что мы вернем
 * значение .F., но не прерывать обработку.
 * Если пользователь щелкнул мышью по
 * кнопке Cancel в окне индикатора прогресса,
 * прервать процесс.
   llReturn = .T.
   plCancel = .F.
   for lnI = 1 to .nFiles
     lcFile = .aFiles[lnI, 1]
     .cErrorMessage = ''
     do case
       case not .ProcessFile(lcFile)
         strtofile(.cErrorMessage, .cLogFile, .T.)
         llReturn = .F.
       case plCancel
         llReturn = .F.
         exit
     endcase
     .oTherm.Update(lnI, 'Processing ' + lcFile + '...')
   next lnI
 endwith
 return llReturn
```

Вызываемый программой Process метод ProcessFile обрабатывает каждый файл, обращаясь для этого к методу ProcessFile другого объекта, ссылка на который хранится в свойстве oProcess. Если что-то идет не так, значение свойства cErrorMessage выполняющего обработку объекта запоминается в нашем свойстве cErrorMessage вместе с именем файла и символами возврата каретки (carriage return) и перевода строки (line feed) (ccCRLF — константа, определенная в заголовочном файле ASSEMBLE.H). Этот метод возвращает признак успешного или неуспешного завершения процесса обработки.

```
lparameters tcFile
 local llReturn
 with This
   llReturn = .oProcess.ProcessFile(.cDirectory + ;
     tcFile, .cWriteDir + tcFile)
   if not llReturn
     .cErrorMessage = tcFile + ': ' + ;
       .oProcess.cErrorMessage + ccCRLF
   endif not llReturn
 endwith
 return llReturn
```

Вот как программа PROCESS.PRG использует объект FileIterator для преобразования Word-документов в HTML-файлы. Она «сотрудничает» с объектом GenerateHTML, который мы рассмотрим в дальнейшем. tlNoGenerateHTML — это параметр, передаваемый в программу PROCESS.PRG; передайте в качестве этого параметра значение .T. для отмены формирования HTML-файлов из Word-документов: например, в том случае, если вам необходимо просто повторить шаги по «чистке» HTML-кода. В переменных lcWordDocs, lcHTMLDir и lcLogFile хранятся соответственно каталог, предназначенный для документов процессора Word, каталог, в который должны быть записаны HTML-файлы, и имя журнального log-файла, куда должны быть записаны сообщения об ошибках.

```
if not tlNoGenerateHTML
   loIterator = createobject('FileIterator')
   with loIterator
     .oProcess   = createobject('GenerateHTML')
     .cDirectory = lcWordDocs
     .cWriteDir  = lcHTMLDir
     .cLogFile   = lcLogFile
```

```
    .GetFiles()
    .Process('Converting Word docs to HTML...')
  endwith
endif not tlNoGenerateHTML
```

Формирование HTML-файлов

Прежде чем обсуждать класс GenerateHTML, давайте рассмотрим его родительский класс — ProcessBaseClass, который является родительским классом для всех тех обрабатывающих объектов, которые мы будем использовать. У класса ProcessBaseClass есть метод ProcessFile, который получает в качестве параметров имена входного и выходного файлов, считывает содержимое входного файла в переменную, обращается к методу Process (абстрактный метод в данном классе) для того, чтобы выполнить обработку содержимого указанного файла, и записывает результаты в выходной файл. Этот метод возвращает значение .T., если процесс завершился успешно; если же нет — в свойстве cErrorMessage хранится причина неудачи.

```
lparameters tcInputFile, ;
   tcOutputFile
local lcFile, ;
  llReturn, ;
  lcStream, ;
  lcResult
with This

* Очистить сообщение об ошибке, чтобы не использовать
* сообщение, полученное для предыдущего файла.

  .cErrorMessage = ''

* Убедиться в том, что файл существует.
  .cFile   = tcInputFile
  lcFile   = justfname(tcInputFile)
  llReturn = file(tcInputFile)

* Считать содержимое файла и обработать его.
  if llReturn
    lcStream = filetostr(tcInputFile)
    lcResult = .Process(lcStream)
    strtofile(lcResult, tcOutputFile)

* Файл не найден, подготовить сообщение об ошибке.

  else
    .cErrorMessage = 'File not found'
  endif llReturn

* Учесть возникновение какой-либо ошибки
* в возвращаемом значении.

  llReturn = llReturn and not .lErrorOccurred
endwith
return llReturn
```

Вас, может быть, удивляет, почему метод ProcessFile сам считывает содержимое файла и передает его в программу Process, вместо того, чтобы просто передать имя файла и заставить программу Process заниматься чтением и записью. Причина заключается в том, что позже мы будем рассматривать использование этого метода для выполнения многократной обработки одного и того же файла, и вместо того, чтобы постоянно заниматься считыванием и записью на диск, у нас просто будет несколько объектов, обрабатывающих текстовый поток и записывающих на диск полученные результаты. Аналогично методу ProcessFile, для выполнения черновой работы метод ProcessStream обращается к программе Process, но он предполагает, что в качестве параметра будет передан текстовый поток, а не имена файлов, и возвращает обработанный поток.

```
lparameters tcInput
 local lcResult
 with This

 * Очистить сообщение об ошибке, чтобы не попалось
 * сообщение, полученное для предыдущего файла.

   .cErrorMessage = ''

 * Обработать входной поток и вернуть результат.

   lcResult = .Process(tcInput)
 endwith
 return lcResult
```

Как я уже упоминал, класс GenerateHTML является подклассом класса ProcessBaseClass. Он будет автоматизировать работу текстового процессора Word с целью генерации HTML-кода для указанного файла. Этот метод создает экземпляр Word и запоминает полученную объектную ссылку в свойстве oWord, а его метод Destroy «закрывает» Word. Поскольку мы не хотим, чтобы метод ProcessFile функционировал в своем обычном режиме (считывал содержимое Word-файла в переменную), этот метод переопределяется. Метод «приказывает» Word открыть указанный входной файл и сохранить его в виде HTML-кода в указанном выходном файле (wdFormatHTML — это константа, определенная в заголовочном файле ASSEMBLE.H, в которой хранится значение, предназначенное для передачи в метод SaveAs и определяющее сохранение данных в виде HTML-файла). Если указанный файл не является Word-файлом (возможно, класс FileIterator обработал все файлы, связанные с документами Word и находящиеся в одном каталоге, включая графические), мы просто скопировали бы этот файл в выходной каталог.

```
lparameters tcInputFile, ;
   tcOutputFile
 local llReturn, ;
   loDocument
 with This

 * Просто скопировать графические и HTML-файлы в выходной
 * каталог.

   if inlist(upper(justext(tcInputFile)), 'JPG', 'GIF', ;
     'BMP', 'TIF', 'HTM', 'HTML')
     copy file (tcInputFile) to (tcOutputFile)
     llReturn = .T.

 * Открыть файл, сохранить его как HTML-файл и закрыть.
```

Таблица 1. Некоторые записи из таблицы SAR.DBF, демонстрирующие, как надо обрабатывать сформированные процессором Word HTML-файлы.

Строка поиска	Строка замены	Комментарии
<html* >	<html>	Удалить стили в тэгах <HTML>.
<body* >	<body>	Удалить стили в тэгах <BODY>.
<![if* >		Удалить разделы IF.
<!*--><<chr(13) + chr(10)>>		Удалить комментарии.
<style>*</style><<chr(13) + chr(10)>>		Удалить определения стилей.
<p class=MsoNormal* > <p>		Удалить ненужные классы.
<o:p>		Удалить этот тэг.
<title>	<link rel="stylesheet" type="text/css" href="<<lcCSSFile>>"><<chr(13) + chr(10)>><title>	Добавить таблицу стилей.

```
  else
    loDocument = .oWord.Documents.Open(tcInputFile)
    llReturn   = not .lErrorOccurred
    if llReturn
      loDocument.AcceptAllRevisions()
      .cFile = lower(forceext(tcOutputFile, 'html'))
      loDocument.SaveAs(.cFile, wdFormatHTML)
      loDocument.Close()
      llReturn = not .lErrorOccurred
    endif llReturn
  endif inlist(upper(justext(tcInputFile)) ...
endwith
return llReturn
```

К слову о графических файлах, есть пара советов относительно создания Word-документов. Во-первых, не встраивайте графический файл в документ; вместо этого организуйте для него связь по ссылке. Сформированный для встроенной графики HTML-код ужасен и требует присутствия целого набора файлов в создаваемом Word подкаталоге. Если вы связываете два этих документа по ссылке, то Word формирует простой тэг <IMG> — почти так же, как поступили бы вы сами, если бы формировали HTML-код вручную. Во-вторых, чтобы свести к минимуму вопросы, связанные с определением путей доступа, прежде чем организовывать связь по ссылке, скопируйте графические файлы в тот же самый каталог, в котором находятся Word-документы. Word использует в ссылках относительные пути доступа к файлам с изображениями, и ссылки не будут работать, если только вы не используете точно такую же относительную структуру каталогов для HTML-файлов, что и для документов Word.

Удаление «мусора»

Как только Word-документы преобразованы в HTML-файлы, следующий шаг — это удаление из полученных файлов нежелательного HTML- и XML-кода. Для выполнения этой работы я снова воспользуюсь объектом FileIterator, но создам новый экземпляр объекта, поскольку на сей раз я работаю с другим каталогом и другим набором файлов. Выполняющий процедуру объект, ассоциированный с объектом Iterator — это объект ReplaceText. Его работа сводится к поиску в текстовом потоке некоторой строки и замене ее на другую строку. Собственно говоря, за один раз он может обрабатывать несколько наборов строк «найти-и-заменить»; метод AddSearchAndReplace получает в качестве параметра такую пару строк и добавляет эти строки в свойство-массив aReplace, а метод Process перебирает все пары, определенные в этом массиве. Мы сейчас рассмотрим метод ReplaceText.

Вместо того, чтобы жестко программировать перечень того, что необходимо «вычистить» в сформированном процессором Word HTML-документе, я решил управлять этим процессом с помощью данных. В таблице SAR.DBF определены четыре колонки: Order (в этом поле указывается последовательность обработки таблицы), Searchfor (искомая строка), Replace (строка замены) и Comment (комментарии о том, что именно достигается путем данной замены). Я не буду описывать все хранящиеся в таблице SAR.DBF записи, но некоторые из них приведены в таблице 1.

В таблице 1 вы заметите несколько интересных вещей. Во-первых, о чем свидетельствуют строки <HTML*> и <BODY*>, объект ReplaceText может обрабатывать шаблоны (wild cards). Строка <HTML*> будет соответствовать любой строке, которая начинается с тэга “<HTML” и заканчивается символом “>”. Во-вторых, иногда строка поиска заменяется на другую строку (например, строка <HTML*> заменятся на строку <HTML>), а иногда она заменяется пустой строкой (например, тэг <![IF*>), что означает удаление найденной строки. Третье, некоторые строки невозможно с легкостью определить в текстовом поле, например в поле SEARCHFOR, поэтому для таких строк я буду поддерживать вычисление значений. Например, для

представления символов возврата каретки и перехода на следующую строку используется строка <<CHR(13) + CHR(10)>>; такую запись намного проще ввести и выдать на экран, чем настоящие символы возврата каретки и перехода на следующую строку. Наконец, обратите внимание на то, как определяется таблица стилей: вдобавок к существующему тэгу <TITLE> используется тэг <LINK>. Не забудьте создать CSS-файл с определениями всех стилей, которые используются в HTML-файлах, или удалите эту запись из таблицы SAR.DBF.

Вот код из программы PROCESS.PRG, который создает экземпляры объектов FileIterator и ReplaceText, регистрирует все пары строк «найти-и-заменить» в таблице SAR.DBF с помощью объекта ReplaceText, а затем обращается к методу Process, чтобы обработать каждый файл. Обратите внимание, эта программа использует реализованную в версии VFP 7 функцию TEXTMERGE() применительно к обеим строкам, Searchfor и Replace, в том случае, если эти строки содержат выражения для вычисления значения. Кроме того, я решил использовать конструкцию SCAN FOR ORDER >= 0 вместо простого цикла SCAN; это позволяет мне исключить обработку определенных записей, не удаляя их из таблицы, путем указания в поле Order отрицательного значения.

```
loIterator = createobject('FileIterator')
 with loIterator
   .cDirectory    = lcHTMLDir
   .cWriteDir     = lcCHMDir
   .cFileSkeleton = '*.htm*'
   .cLogFile      = lcLogFile
   .GetFiles()
 endwith
 loProcess = createobject('ReplaceText')
 with loProcess
   use SAR order ORDER
   scan for ORDER >= 0
    .AddSearchAndReplace(textmerge(alltrim(SEARCHFOR)), ;
      textmerge(alltrim(REPLACE)))
   endscan for ORDER >= 0
   use
 endwith
 loIterator.oProcess = loProcess
 loIterator.Process('Cleaning up HTML docs...')
```

Как и остальные классы рассматриваемого процесса, ReplaceText — это подкласс класса ProcessBaseClass. Я не буду рассматривать его метод Process в полном объеме, а только те фрагменты, которые имеют непосредственное отношение к делу. Метод Process перебирает все пары строк, хранящиеся в массиве aReplace, и выясняет для каждой пары, что предполагается искать. Это не совсем прямолинейный способ из-за возможного использования символа шаблона "*": если такой символ присутствует, мы проходим по текстовому потоку (он хранится в переменной lcString), используя реализованную в версии VFP 7 функцию STREXTRACT(), для нахождения очередного включения в текст строки поиска до тех пор, пока не будут найдены все соответствующие строки. Обратите внимание на то, что этот код выполняет проверку LEN(lcSearch) > 0 вместо проверки NOT EMPTY(lcSearch), поскольку функция EMPTY() возвращает значение .T. для допустимых строк поиска, например, строк, состоящих из символов возврата каретки и перехода на следующую строку. Кроме того, обратите внимание на приведенные в коде комментарии относительно использования функции AT() в качестве способа разрешить проблему, связанную с ошибкой в работе реализованной в VFP функции ATC().

```
lcSearch      = .aReplace[lnI, 1]
 lnWildCardPos = at('*', lcSearch)
 if lnWildCardPos > 0
   lcBegin = left(lcSearch,   lnWildCardPos-1)
   lcEnd   = substr(lcSearch, lnWildCardPos + 1)
 endif lnWildCardPos > 0
 lnLastPos = 0
 do while len(lcSearch) > 0

 * Если у нас есть символ шаблона (wild card),
 * мы сначала найдем сам текст.

   if lnWildCardPos > 0
     lcSearch = strextract(lcString, lcBegin, lcEnd, 1, 1)
     if len(lcSearch) > 0
       lcSearch = lcBegin + lcSearch + lcEnd
     endif len(lcSearch) > 0
   endif lnWildCardPos > 0

 * Теперь, когда мы знаем, что необходимо искать,
 * попытаемся это найти,
 * используя поиск без учета регистра.
 * Обратите внимание на использование
 * функции AT()в том случае, когда функция ATC()
 * не сработала из-за ошибки:
 * если строка поиска слишком длинная,
 * функция ATC() возвращает значение 0,
 * но функция AT() работает правильно.

   if len(lcSearch) > 0
     lnPos   = atc(lcSearch, lcString)
     llFound = lnPos > 0
     if lnPos = 0
       lnPos   = at(upper(lcSearch), upper(lcString))
       llFound = lnPos > 0
     endif lnPos = 0
   else
     llFound = .F.
   endif len(lcSearch) > 0
```

В конце приведенного фрагмента программного кода переменная llFound получает значение .T., если строка поиска была найдена в текстовом потоке, а переменная lcSearch может быть пустой, если в текстовом потоке больше не встретился определенный с помощью шаблона искомый текст. Остальной код прост: в нем для выполнения замены без учета регистра строки поиска на строку замены используется функция STRTRAN() с применением нового параметра flags, появившегося в версии VFP 7:

```
lcString = strtran(lcString, lcSearch, lcReplace, -1, ;
   -1, 1)
```

Дополнительная обработка

Итак, к этому моменту мы преобразовали Word-документы в HTML-файлы и удалили из полученных файлов ненужный HTML- и XML-код. Последний шаг — это некоторая дополнительная обработка HTML-файлов. Поскольку такая обработка будет варьироваться от проекта к проекту, она управляется данными, хранящимися в таблице PROCESS.DBF. Из-за того, что существует возможность «серийной» обработки (multiple processes), мы будем использовать объект FileIteratorMultipleProcess для организации итеративной обработки файлов. Этот подкласс класса FileIterator использует для хранения принадлежащих процедуре обработки объектов массив aProcesses вместо единственного свойства oProcess. Кроме того, он обращается к методу ProcessStream объекта, представляющего процедуру, а не к методу ProcessFile, так что каждый файл считывается и записывается на диск только однажды, независимо от числа процедур, выполняемых с этим файлом.

В таблице PROCESS.DBF имеется шесть столбцов: Order (в котором определяется порядок обработки таблицы; как и SAR.DBF, отрицательное значение означает пропуск записи), Class и Library (где указываются имена класса процедуры обработки и VCX-файла, в котором хранится определение этого класса), Title (заголовок для индикатора прогресса), Code (любой VFP-код, который должен быть выполнен с помощью реализованной в версии VFP 7 функции EXECSCRIPT() как часть настройки данной процедуры) и Comment (комментарии о том, что делает данная процедура).

В состав тех процедур обработки, которые могут быть определены в таблице PROCESS.DBF, входят процедура обработки изображений (тэги IMG, формируемые процессором Word, содержат некоторые атрибуты, не являющиеся необходимыми, и могут не всегда правильно ссылаться на файлы изображений), процедура обработки таблиц («табличные» тэги, например тэги <TABLE> и <TD>, содержат информацию о стилях, более детально определенных в CSS-файле, а границы определяются не в тэге <TABLE>, используемом обычно для этих целей, а в тэгах <TD>), предшествующая форматированию обработка текста (эти разделы могут оказаться настоящим кошмаром при наличии HTML-кода, формируемого процессором Word по умолчанию, особенно после того, как все остальные вещи такого рода были удалены) и специальное форматирование абзаца (bullet) (как вы наблюдали ранее, текстовый процессор Word настаивает на использовании вместо тэгов <UL> и <LI> текста с отступами и символами, вставленными вручную).

Я создал набор классов для реализации наиболее распространенных процедур обработки, включая классы FixBullets, FixImages, FixPreformatted и FixTables. Чтобы создать собственные обрабатывающие классы, выясните, прежде всего, путем анализа HTML-файлов, полученных в результате всей предшествующей обработки, что вам необходимо сделать, примите решение о том, какие именно изменения необходимы и каким образом следует их осуществить, создайте подкласс на базе классов ProcessBaseClass или ReplaceText и внесите запись о классе этой процедуры в таблицу PROCESS.DBF. Что ж, это довольно туманное руководство, но в данном случае трудно быть более конкретным. Чтобы получить представление о том, как вы можете справиться с этой задачей, изучите классы, созданные мной.

Вот код из программы PROCESS.PRG, который создает экземпляры объектов классов процедур обработки, определенных в таблице PROCESS, регистрирует их с помощью объекта FileIteratorMultipleProcess, а затем обращается к методу Process, чтобы обработать все HTML-файлы:

```
loIterator = createobject('FileIteratorMultipleProcess')
 with loIterator
   .cDirectory    = lcCHMDir
   .cFileSkeleton = '*.htm*'
   .cLogFile      = lcLogFile
   .GetFiles()
   select 0
   use PROCESSES order ORDER
   count for ORDER >= 0 to lnProcesses
   dimension .aProcesses[lnProcesses]
   lnProcess = 0
   scan for ORDER >= 0
     lnProcess = lnProcess + 1
     loProcess = newobject(alltrim(CLASS), ;
       alltrim(LIBRARY))
     loProcess.cTitle = alltrim(TITLE)
     if not empty(CODE)
       execscript(CODE, loProcess)
     endif not empty(CODE)
     .aProcesses[lnProcess] = loProcess
   endscan for ORDER >= 0
   use
   .Process('Additional processing...')
 endwith
```

В заключение

Фу-у! Было о чем поговорить. Всего лишь несколько классов для выполнения всей работы по преобразованию Word-документов в приемлемые (как по внешнему виду, так и по размеру) HTML-документы. Итак, чтобы вы могли испробовать их в деле, в файл-приложение, который вы найдете на дискете, сопровождающей журнал, включены исходные Word-документы для всех моих статей, опубликованных в журнале FoxTalk, начиная с 2000 года, CSS-файл для украшения HTML-представления, настроенные таблицы SAR.DBF и PROCESS.DBF и несколько классов, определенных в библиотеке Foxtalk.VCX,

предназначенных для дополнительной обработки (например, изображения в исходных документах не являются ни встроенными, ни связанными, а вместо этого на них ссылаются директивы сервера публикаций, поэтому класс FixFoxTalkImages заменяет эти директивы на соответствующие тэги <IMG>).

Чтобы запустить процесс, распакуйте исходный код так, чтобы сохранилась предусмотренная структура каталогов, и выполните команду DO FOXTALKMAIN.PRG из каталога FoxTalk (эта программа просто обращается к программе PROCESS.PRG из каталога Process, передавая ей соответствующие параметры). Программа PROCESS.PRG будет обрабатывать Word-документы, находящиеся в подкаталоге WordDocs, и когда обработка будет полностью завершена, вы обнаружите новые подкаталоги HTMLDocs и HTMLHelp. В подкаталоге HTMLDocs находятся исходные файлы, сформированные процессором Word (откройте их в блокноте Notepad, чтобы увидеть все то, что в них вставил процессор Word), а в подкаталоге HTMLHelp хранятся файлы, полученные в результате дополнительной обработки (включая изображения и CSS-файлы). Сравните размеры файлов из подкаталога HTMLDocs с размерами файлов в подкаталоге HTMLHelp, а также откройте эти файлы в браузере, чтобы посмотреть, как изменился их внешний вид.

В следующем месяце я возьму HTML-файлы, которые были получены в результате рассмотренного в данной статье процесса, и сформирую из этих файлов CHM-файл.

Дуг Хенниг — один из партнеров в канадской фирме Stonefield Systems Group, Inc. Он является автором набора инструментальных средств для FoxPro-разработчиков Stonefield Database Toolkit for Visual FoxPro, а также соавтором книг «What's New in Visual FoxPro 7.0» и «The Hacker's Guide to Visual FoxPro 7.0», вышедших в издательстве Hentzenwerke Publishing, и автором книги «The Visual FoxPro Data Dictionary», вышедшей в издательстве Pinnacle Publishing. Дуг являлся техническим редактором книг «The Hacker's Guide to Visual FoxPro 6.0» и «The Fundamentals», вышедших в издательстве Hentzenwerke Publishing. Дуг в качестве докладчика участвовал в работе всех конференций Microsoft FoxPro Developers Conference (DevCon), и, кроме того, он выступает перед группами пользователей и на конференциях разработчиков, проводимых по всей Северной Америке. Профессионализм Дуга подтверждается сертификатами Microsoft Most Valuable Professional (MVP) и Certified Professional (MCP).
Его адрес: www.stonefield.com, dhennig@stonefield.com

❑❑❑

Поиск аномалий в данных

Джон М. Миллер (John M. Miller)

WIN 7.0 6.0 5.0 DOWNLOAD

В этой статье Джон Миллер обсуждает вопросы обеспечения качества данных и представляет методику обнаружения ошибок в данных. Он представляет метод общего назначения для анализа больших наборов данных и поясняет программный код для его реализации.

Каждый, кто хоть какое-то время работал с данными, особенно с большими объемами данных, пришел к пониманию того, насколько трудно поддерживать высокий уровень качества данных. Даже если данные постоянно находятся под нашим непосредственным контролем, могут потребоваться значительные усилия для обеспечения гарантии того, что они сохраняют свою корректность и достоверность.

Однако, сегодня обработка данных заключается, главным образом, в подготовке к пересылке куда-то еще и в получении информации из других источников. Иными словами, большая часть данных находится вне пределов нашего непосредственного контроля.

Качество данных может вызывать сомнения с разных точек зрения. Медленная обработка, устаревание, ошибки ввода постепенно ухудшают качество данных. Если данные импортируются из другого источника, сомнению может быть подвергнуто качество исходных данных. И, как бы неприятно не было нам признавать это, подчас мы сами неумышленно рискуем ухудшить качество либо тем, что исполняем применительно к рабочим данным программный код, вносящий ошибки, либо непосредственно манипулируя данными и ошибаясь при этом.

В VFP для поддержания целостности данных есть несколько способов: вы можете определить уникальные индексные значения, сформулировать правила для полей и таблиц и использовать триггеры

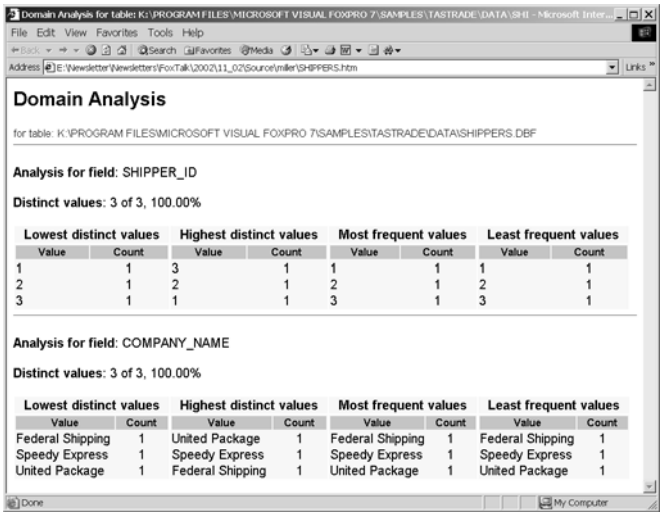
для обеспечения ссылочной целостности. Однако, не всегда возможно или желательно накладывать ограничения на все те отклонения, которые могут встретиться в данных. Кроме того, если данные поступают из внешнего источника, вам необходимо их проанализировать перед тем как добавлять в свою базу данных.

Есть несколько методик проведения анализа данных с целью выявления в них различных аномалий; чаще всего применяется «лобовое» решение — просмотр таблиц с помощью команды BROWSE и визуальное инспектирование данных. Мы все занимались этим делом, и вы многократно находили ошибки; но если в таблице насчитывается несколько сотен и более записей, такая работа быстро переходит в разряд утомительной и неосуществимой. Таким образом, нам остается организовать до некоторой степени автоматизированный процесс.

В зависимости от природы данных, вы можете написать программный код для анализа и поиска проблемных мест. Однако, число проблем, присутствие которых необходимо выявить в ходе тестирования, может быстро возрастать и потребовать больших затрат времени на выполнение всей процедуры. Хотя некоторые проблемы могут быть выявлены только с помощью программного кода, специально предназначенного для обнаружения данной аномалии, многие другие отклонения могут быть вскрыты при использовании более общего метода. В данной статье рассматривается программа, которую я называю SCRUB (scrub в переводе с английского — чистить) и использую для обнаружения «неправильностей» в данных применительно к любой таблице. В предлагаемой методике внимание фокусируется на обнаружении непреднамеренных и механических ошибок в данных.

Алгоритм для программы SCRUB был представлен на встрече Los Angeles DAMA Chapter, проходившей летом 1999 года. В основе своей представленная идея заключалась в том, что если вы исполняете некоторые простые запросы к таблице, вы можете идентифицировать множество аномалий в данных. Я вчерне набросал эти запросы на салфетке и запрограммировал их сразу же, как только вернулся в офис.

Минимальные требования программы SCRUB — это имя таблицы и выходного файла. Программа SCRUB исходит из предположения, что таблица находится в выбранной рабочей области и меняет расширение файла DBF на расширение HTM для создания выходного файла. Программа SCRUB не требует параметров и будет работать, если в командном окне ввести команду DO SCRUB.PRG.



Domain Analysis for table: K:\PROGRAM FILES\MICROSOFT VISUAL FOXPRO 7\SAMPLES\TASTRADE\DATA\SHI - Microsoft Inter...

File Edit View Favorites Tools Help

Address E:\Newsletter\Newsletters\FoxTalk\2002\11_02\Source\miller\SHIPPERS.htm

Domain Analysis

for table: K:\PROGRAM FILES\MICROSOFT VISUAL FOXPRO 7\SAMPLES\TASTRADE\DATA\SHIPPERS.DBF

Analysis for field: SHIPPER_ID

Distinct values: 3 of 3, 100.00%

Lowest distinct values		Highest distinct values		Most frequent values		Least frequent values	
Value	Count	Value	Count	Value	Count	Value	Count
1	1	3	1	1	1	1	1
2	1	2	1	2	1	2	1
3	1	1	1	3	1	3	1

Analysis for field: COMPANY_NAME

Distinct values: 3 of 3, 100.00%

Lowest distinct values		Highest distinct values		Most frequent values		Least frequent values	
Value	Count	Value	Count	Value	Count	Value	Count
Federal Shipping	1	United Package	1	Federal Shipping	1	Federal Shipping	1
Speedy Express	1	Speedy Express	1	Speedy Express	1	Speedy Express	1
United Package	1	Federal Shipping	1	United Package	1	United Package	1

Done My Computer

Рис. 1. Пример выходных данных программы SCRUB.

Эта программа создает HTML-файл с результатами анализа. Как только анализ будет закончен, необязательный булевский параметр Boolean передаст выходной HTML-файл в браузер IE. Браузер IE необходим только для реализации данной возможности, и программа легко может быть модифицирована для использования другого браузера. На рис. 1 представлен пример выходных данных программы SCRUB.

Для создания выходных данных программа исполняет по пять запросов для каждого поля таблицы. Будьте осторожны и не исполняйте программу SCRUB применительно к большим рабочим таблицам в рабочее время, поскольку каждый из этих запросов должен последовательно перебрать все записи в таблице. Поскольку в запросах используется предложение DISTINCT языка запросов SQL, их нельзя оптимизировать и, следовательно, для их выполнения может потребоваться много времени. Для каждого поля программа вычисляет количество значений, отличающихся от значений, хранящихся в этом поле, 10 наименьших из отличающихся значений, 10 наибольших из отличающихся значений, 10 самых редко встречающихся отличающихся значений и 10 наиболее часто встречающихся отличающихся значений.

Эта информация полезна, потому что отсортировать встретившиеся аномалии вручную трудно, а такой отчет легко может указать на проблемы в данных. Например, если количество отличающихся значений для данного поля неожиданно изменилось, такой факт мог бы указать на проблему, связанную с работой приложения, или ошибку ввода данных.

Подобно тому, как изменения в списке наиболее часто встречающихся отличающихся значений также могли бы означать наличие затруднений. Самые интересные и полезные — это отличающиеся значения, которые встречаются реже всего, а также наименьшие и наибольшие из отличающихся значений. Среди таких значений вы обнаружите орфографические ошибки, ошибки пользовательского интерфейса UI, проблемы связывания данных (data-binding problems) и многие другие разновидности проблем, влияющих на качество данных.

Исходный код программы SCRUB замечателен не только тем, что он делает; он также демонстрирует некоторые хорошие приемы программирования. Например, этот код в высшей степени «читабелен» даже для тех программистов, кто не знаком с синтаксисом используемого языка программирования. Этот код разрежен, а принятые соглашения написания кода ясны и неукоснительно соблюдаются. Программа хорошо структурирована и в ней использованы имена, которые истолковываются однозначно. Кроме того, будучи генератором HTML-кода, программа SCRUB, помимо прочего, решает еще две задачи. Вырабатываемый этой программой HTML ясен, хорошо организован и легок для понимания. Наконец, необходимо, по мере возможности, упростить внесение изменений в сформированный HTML, чтобы облегчить чтение HTML в контексте программы. По одним только перечисленным причинам данный код заслуживает изучения.

```
* SCRUB.prg, FoxPro Command File
* (c) jMM [2002.02.28 01:45:29 PM PST]

lparameter tlShow

* Эта подпрограмма анализирует любой DBF-файл
* для выявления аномалий в данных.
* Подразумевается, что анализируемая таблица открыта
* в текущей рабочей области.
* Подпрограмма создает файл с тем же именем, что и алиас,
* выбранной в данный момент рабочей области,
* с расширением HTM.

* ПРЕДУПРЕЖДЕНИЕ!
*     Исполнение этой программы может потребовать
*     исключительно много времени,
*     если в таблице хранится очень много записей
*     и/или в этой таблице определено очень много
*     полей.

 local lcHTMName

 lcHTMName = forceext( alias() , 'htm' )

 set textmerge on noshow
 set textmerge to ( lcHTMName )

 set nulldisplay to "NULL"

 #define CR chr(13)

 private lnStart, lnStart2
 local lnSecs , lnET , lnTR, lcMsg, i

 lnStart = seconds()
```

```
\<html>
\ <head>
\  <title>Domain Analysis for table: << dbf() >>
\  </title>
\  <style>
\   body {font-family: arial, sans-serif}
\   table.main th {
\    padding-left:15px;
\    padding-right:15px;
\    background-color:whitesmoke}
\   table.main td {
\    padding:0px;
\    vertical-align:top;
\    text-align:center}
\   table.results {
\    background-color:whitesmoke;
\    width:100%}
\   table.results td.value {
\    padding:0px;
\    text-align:left}
\   table.results td.count {
\    padding:0px;
\    text-align:center}
\   table.results th {
\    background-color:silver;
\    font-size:x-small}
\  </style>
\ </head>
\ <body>
\  <h2>Domain Analysis</h2>
\  <small> for table: << dbf() >></small>
```

Этот первый раздел программы получает параметр, который управляет передачей выходных данных браузеру IE, а затем формирует выходной HTML-файл. Данный фрагмент кода открывает выходной файл, используя команду textmerge языка программирования VFP, и формирует начало HTML-документа.

```
for i = 1 to fcount()

   if not inlist( type( field( i ) ) , "G" , "M" )

     lcMsg = allt( str( i ) ) + ' of ' + ;
       allt( str( fcount() ) )

     lnStart2 = seconds()

     \  <hr>
     \  <p><b>Analysis for field</b>: <<field(i)>></p>

     DomainAnalysis( field(i) , lcMsg )

     lnSecs = seconds()-lnStart
     lnET = ntot( ( ( lnSecs ) / i ) * ( fcount() ) )
     lnTR = ntot( ( ( lnSecs ) / i ) * ( fcount() -i ))

     wait window nowait ;
       "Estimated time: " + lnET + CR + ;
       "Estimated time remaining: " + lnTR + CR + ;
       "Total elapsed time: " + ntot( lnSecs ) + CR + ;
       "Elapsed time for field " + field( i ) + ": " +;
       ntot( seconds()-lnStart2 )

   endif

 endfor
```

Главный цикл программы перебирает все поля таблицы, исключая поля типа Memo и General, и формирует в HTML-файле заголовок для каждого поля, а также создает текстовую строку, которая используется для выдачи на экран информации о состоянии программы в процессе ее исполнения. Внутри цикла вызывается процедура DomainAnalysis, которой пере-

дается в качестве параметра имя анализируемого поля.

```
procedure DomainAnalysis( tcFN , tcMsg )

 local lcAlias

 lcAlias = alias()

 DomainAnalysis1( lcAlias , tcFN , tcMsg )

 \  <table class="main">
 \   <tr>
 \    <th>Lowest distinct values</th>
 \    <th>Highest distinct values</th>
 \    <th>Most frequent values</th>
 \    <th>Least frequent values</th>
 \   </tr>
 \   <tr>
 \    <td>
 \     <table class="results">

 tableHeader()
 DomainAnalysis2( lcAlias , tcFN , tcMsg )

 \     </table>
 \    </td>
 \    <td>
 \     <table class="results">

 tableHeader()
 DomainAnalysis3( lcAlias , tcFN , tcMsg )

 \     </table>
 \    </td>
 \    <td>
 \     <table class="results">

 tableHeader()
 DomainAnalysis4( lcAlias , tcFN , tcMsg )

 \     </table>
 \    </td>
 \    <td>
 \     <table class="results">

 tableHeader()
 DomainAnalysis5( lcAlias , tcFN , tcMsg )

 \     </table>
 \    </td>
 \   </tr>
 \  </table>

 select ( lcAlias )

 set message to
```

Процедура DomainAnalysis определяет структуру выходного файла, формируя набор HTML-таблиц для хранения результатов анализа, и вызывает те процедуры, которые выполняют анализ и которым с большой выдумкой присвоены имена от DomainAnalysis1 до DomainAnalysis5 включительно.

```
procedure tableHeader

 \      <tr>
 \       <th>Value</th>
 \       <th>Count</th>
 \      </tr>
```

Процедура tableHeader формирует общие для каждого набора результатов ячейки HTML-заголовка.

Исследование данных

Первое исследование, которое проводится для каждого поля — это простой подсчет количества уникальных значений. Полученный результат выдается в сравнении с количеством записей в таблице. Число и доля уникальных значений в процентном отношении обеспечивает такую систему показателей, которая позволяет оценивать содержимое поля без необходимости понимать смысловое значение конкретных элементов данных. Некоторые значения поля всегда уникальны, например, поля первичных ключей. Некоторые хранящиеся в поле значения имеют определенное количество или определенный процент уникальных значений, например, поля внешних ключей — они зависят от набора значений, хранящихся в другой таблице. Поле с единственным уникальным значением, разумеется, является причиной для беспокойства, если в таком поле для каждой записи хранится одно и то же значение, то что вообще здесь делает такой «выскочка»? Поле, в котором уникальные значения составляют чуть меньше 100 процентов, также могло бы оказаться проблемным. Наконец, если количество или доля уникальных значений в процентном отношении изменяются по прошествии времени, это также могло бы оказаться причиной для будущих расследований.

```
procedure DomainAnalysis1( tcAlias, tcFN, tcMsg )

 local lnPct

 set message to ;
   "Calculating distinct values for field " + ;
   tcMsg + " " + proper( tcFN )

 select distinct &tcFN ;
       from ( tcAlias ) ;
       where not deleted() ;
       into cursor temp1

 select count( * ) as cnt ;
       from ( tcAlias ) ;
       where not deleted() ;
       into cursor temp2

 \  <p><b>Distinct values</b>: << reccount( 'temp1' )>>
 \\of << temp2.cnt >>

 if temp2.cnt > 0
   lnPct = ( reccount('temp1') / temp2.cnt ) * 100
   \\, << round( lnPct , 2 ) >>%
 endif

 \\</p>

 use in temp1
 use in temp2
```

Процедура DomainAnalysis1 выполняет два SQL-запроса и с помощью функций textmerge помещает результаты этих запросов в HTML-файл. По первому запросу будут получены уникальные значения, хранящиеся в данном поле. Второй запрос подсчитывает количество неудаленных записей в таблице. Вы-

ходные данные состоят из количества уникальных значений, количества всех значений и доли уникальных значений в процентном отношении.

В ходе второго исследования выявляются 10 наименьших из хранящихся в поле значений и организуется счетчик, который определяет, сколько раз встречается в данном поле в таблице каждое такое значение. Тем самым определяется самая нижняя граница для значений, хранящихся в данном поле. В ходе такого исследования выявляются пустые поля, а также поля, значения в которых начинаются с непечатаемых символов. Счетчик дает представление о том, насколько распространенным является данное значение. Если конкретное недопустимое значение встретилось только однажды, тогда это, вероятно, опечатка. Если это значение встречается неоднократно, тогда это могла бы быть программная ошибка или вопрос обучения пользователей.

```
procedure DomainAnalysis2( tcAlias, tcFN, tcMsg )

 set message to ;
   "Calculating lowest distinct values for field " + ;
   tcMsg + " " + proper( tcFN )

 select ;
       top 10 ;
            &tcFN as fld , ;
            count(  &tcFN ) as cnt ;
       group by 1 ;
       from ( tcAlias ) ;
       order by 1 ;
       where not deleted() ;
       into cursor temp1
 dumpdata()

 use in temp1
```

По запросу DomainAnalysis2 выбираются наименьшие отличающиеся значения путем группировки хранящихся в поле значений. Создаваемый для поля список включает значение, хранящееся в данном поле, и количество записей, в которых содержится каждое значение. Этот запрос упорядочен по хранящемуся в поле значению, и выбор первой десятки значений дает 10 наименьших значений, отсортированных в порядке возрастания.

Данная подпрограмма, также как и подпрограммы, начиная с DomainAnalysis3 по DomainAnalysis5 включительно, формирует курсор с общей структурой, который позволяет главной процедуре dumpData представить полученные результаты в виде выходных HTML-данных.

Третье исследование — это еще один анализ границ, и он является противоположностью второго исследования. Вместо наименьших из отличающихся значений вычисляются наибольшие из отличающихся значений. Это может быть полезно при «вылавливании» значений, представленных в нижнем регистре, и тех значений, которые начинаются с цифры.

Процедуры DomainAnalysis2 и DomainAnalysis3 исходят из предположения, что многие ошибки в данных встречаются на границах обычного диапазона значений данных, и если исследовать эти границы, можно обнаружить и исправить подобные ошибки.

```
procedure DomainAnalysis3( tcAlias, tcFN, tcMsg )

 set message to ;
   "Calculating highest distinct values for field " + ;
   tcMsg + " " + proper( tcFN )

 select ;
       top 10 ;
            &tcFN as fld , ;
            count(  &tcFN ) as cnt ;
       group by 1 ;
       from ( tcAlias ) ;
       order by 1 descending ;
       where not deleted() ;
       into cursor temp1

 dumpdata()

 use in temp1
```

Единственное отличие процедуры DomainAnalysis2 от процедуры DomainAnalysis3 заключается в том, что в процедуре DomainAnalysis3 в состав предложений, составляющих SQL-запрос, включена опция descending. Этот запрос возвращает 10 наибольших значений.

Две следующие анализирующие процедуры концентрируются на вычислении числа, которое определяет, сколько раз встречаются значения. Первая из этих процедур, DomainAnalysis4, подсчитывает количество тех значений, которые встречаются чаще всего. Эти значения являются наиболее употребительными, и они имеют дело с основным массивом данных, а не с его границами. Тут вы вскроете большие проблемы. Самая распространенная проблема — пустые значения. Если вы повторяете этот анализ периодически, вы можете выявить характер отклонений в данных, и, если он внезапно меняется, значит произошло значительное изменение в данных — еще один признак возможных затруднений.

```
procedure DomainAnalysis4( tcAlias, tcFN, tcMsg )

 set message to ;
   "Calculating most frequent distinct values " + ;
   "for field " + tcMsg + " " + proper( tcFN )

 select ;
       top 10 ;
            &tcFN as fld , ;
            count(  &tcFN ) as cnt ;
       group by 1 ;
       from ( tcAlias ) ;
       order by 2 descending ;
       where not deleted() ;
       into cursor temp1

 dumpdata()

 use in temp1
```

Этот запрос аналогичен остальным запросам. В данном случае результатом запроса является не список значений, а количество появлений значения. Окончательный анализ выявляет значения, которые встречаются реже всего. При этом идентифицируются мелкие ошибки различных типов, орфографические и случайные ошибки.

```
procedure DomainAnalysis5( tcAlias, tcFN, tcMsg )

 set message to ;
   "Calculating least frequent distinct values " + ;
   "for field " + tcMsg + " " + proper( tcFN )

 select ;
       top 10 ;
            &tcFN as fld , ;
            count(  &tcFN ) as cnt ;
       group by 1 ;
       from ( tcAlias ) ;
       order by 2 ;
       where not deleted() ;
       into cursor temp1

 dumpdata()

 use in temp1
```

Опять-таки, единственное отличие процедуры DomainAnalysis4 от процедуры DomainAnalysis5 состоит в том, что в последней из них в состав предложений, составляющих SQL-запрос, не включена опция descending.

Остальной программный код состоит из набора вспомогательных подпрограмм. Первая из них — это процедура dumpData, которая вызывается в конце работы каждой анализирующей подпрограммы для вывода результатов.

```
procedure dumpData

 scan next 10
   \       <tr>
   \        <td class="value">
   \         << formatdata( fld ) >>
   \        </td>
   \        <td class="count">
   \         << formatdata( cnt ) >>
   \        </td>
   \       </tr>
 endscan
```

Процедура dumpData последовательно просматривает текущий курсор (как правило, это курсор temp1) и выдает ячейки HTML-таблицы, предназначенные для представления значения и количества его появлений в поле таблицы.

Процедура dumpData обращается к функции formatData, которая возвращает неразделяющий HTML-символ пробела (HTML non-breaking space character), если передано пустое значение.

```
function formatData( txValue )
return iif( empty( txValue ) , " " , txValue )
```

Последняя вспомогательная функция, ntot, преобразует целое значение в символьную строку в формате представления времени hh:mm:ss. Эта функция используется для преобразования полученного с помощью VFP-функции seconds() значения истекшего времени в форму, понятную пользователям.

```
function ntot( tnValue )

 #define secondsPerHour 3600
 #define secondsPerMinute 60

 local lcHours , lcMinutes , lcSeconds
 local lnHours , lnMinutes , lnSeconds

 lnHours = int( tnValue / secondsPerHour )
 lnMinutes = tnValue % secondsPerHour
 lnMinutes = int( lnMinutes / secondsPerMinute )
 lnSeconds = int( tnValue % secondsPerMinute )

 lcHours  = padl( lnHours , 2 , '0' )
 lcMinutes = padl( lnMinutes , 2 , '0' )
 lcSeconds = padl( lnSeconds , 2 , '0' )

 return lcHours + ":" + lcMinutes + ":" + lcSeconds
```

После того как процедура DomainAnalysis завершила свою работу, остается только выполнить небольшую подчистку, завершив формирование HTML-документа и закрыв HTML-файл. Последняя задача, если это требуется согласно переданному параметру, — запустить браузер Internet Explore и открыть в нем созданный HTML-документ.

```
\ </body>
\</html>


 set textmerge to

 if tlShow

    ox=createobject( 'internetexplorer.application' )
    ox.navigate( 'file:///'+ fullpath( lcHTMName ) )
    ox.visible=.t.
    release ox

 endif
```

Эта задача решается путем создания экземпляра объекта браузера IE и обращения к методу navigate объекта IE с передачей ему URL-локатора для созданного файла.

Чистка базы данных TazTrade

Посмотрите в выходных файлах окончательные результаты исполнения программы SCRUB применительно к таблицам из набора данных поставляемого в составе VFP примера TazTrade. Хотя данные из примера TazTrade находятся в хорошем состоянии, проведенный анализ дал некоторые интересные результаты.

В таблице 1 приведены наибольшие из отличающихся значений, полученные в результате анализа поля orditems.quantity. Оказалось, что объем одного из заказов составил 100 000 единиц. Следующее наибольшее значение не превышает 1 000 единиц, а

подавляющее большинство заказов имеет объем менее 150 единиц. Это один вопрос — два заказа с большими значениями, и он определенно служит основанием для более пристального рассмотрения ситуации.

Таблица 1. Наибольшие из отличающихся значений.

Значение	Счетчик
100000.000	1
998.000	1
130.000	2
120.000	8
110.000	5
100.000	11
91.000	1
90.000	6
84.000	2
80.000	19

В таблице 2 приведены наиболее часто встречающиеся значения для поля orditems.ord_id из таблицы orditems. Выяснилось, что объем одного из заказов значительно превышает объемы других заказов. Хотя такая ситуация может быть вполне нормальным явлением, разумно было бы провести расследование по этому поводу и посмотреть, что еще могло бы отличать этот заказ от остальных.

Таблица 2. Наиболее часто встречающиеся значения.

Значение	Счетчик
1078	26
22	6
99	6
658	6
848	6
980	6
16	5
59	5
95	5
159	5

В таблице 3 приведены наименьшие из отличающихся значений поля customer.contact_title. Эта таблица свидетельствует о том, что один из клиентов не имеет данных о своей должности. Возможно, это нормальное явление, но этот факт не является обычным, потому что данные о должности пропущены только для одного из клиентов. Если бы это было обычным явлением, тогда вы могли ожидать, что таких пропусков было бы больше.

Таблица 3. Наименьшие из отличающихся значений.

Значение	Счетчик
(blank)	1
Accounting Manager	10
Assistant Sales Agent	2
Assistant Sales Representative	1
Marketing Assistant	6
Marketing Manager	12
Order Administrator	2
Owner	17
Owner/Marketing Assistant	1
Sales Agent	5

Интересное дополнение, на реализацию которого у меня не нашлось времени, заключается в сохранении вычисленных значений и последующем сравнении текущего отчета со всеми или с любым из предыдущих отчетов. Это облегчит выявление дополнительных аномалий, появившихся по прошествии времени.

Из-за того, что проводимый анализ может потребовать много времени, вам следовало бы исполнять программу SCRUB в нерабочее время или применительно к резервной копии рабочих данных. Исполняйте эту программу так часто, как это возможно, и сравнивайте полученные результаты с результатами предыдущего анализа, чтобы получить более полное представление о том, как меняются ваши данные. Значение программы SCRUB состоит в том, что результаты ее работы могут дать вам хорошее представление о хранящихся в большой таблице данных в ясном и кратком формате. Чем больше вы используете эту программу, тем больше возрастает ее цена.

Джон М. Миллер (John M. Miller) — признанный авторитет в области спецификации, проектирования и реализации программного обеспечения. Он занимается проектированием и реализацией программных приложений начиная с 1984 года и внес свой вклад в создание приложений для различных общественных и частных организаций. Джон часто выступает по вопросам проектирования и разработки приложений на местных, общенациональных и международных технических конференциях и публикует статьи в ведущих технических изданиях. Кроме того, его квалификация востребована производителями инструментальных средств, используемых для проектирования и разработки приложений, в целях оценки и тестирования создаваемых ими продуктов.

Его адрес: jMiller@pdata.com

❑❑❑

Интеграция браузера IE в VFP-приложения

Реми Карон (Remi Caron)

DOWNLOAD

Ну да, каждому из нас известно, что браузер — это неотъемлемая часть операционной системы. Тут все ясно. Но вы можете не знать, что этот браузер можно сделать неотъемлемой частью приложения, созданного средствами Visual FoxPro. Как добиться этого объясняет Реми Карон.

В данной статье я покажу вам, как следует обращаться с браузером Internet Explorer. Большинство из вас, вероятно, пользуется этим прекрасным элементом управления для «путешествий» по просторам Internet. В данной статье все внимание уделяется тому, чего вы можете добиться от браузера при разработке своих собственных приложений. Приведенные в этой статье примеры создавались с использованием языка программирования Visual FoxPro, но это вовсе не является причиной для того, чтобы немедленно прекратить чтение — эти примеры «работают» для всех, независимо от того, какой язык программирования использует читатель. Тщательно изучите рассматриваемый программный код, почувствуйте мощь предлагаемых возможностей и воспользуйтесь этой силой в своей собственной разработке.

Браузер в роли рабочего стола для VFP

Давайте начнем с чего-нибудь простого — используем браузер в качестве активного рабочего стола в интерактивной среде разработки IDE (см. рис. 1). Ваша любимая группа новостей или Web-сайт могли бы послужить вам фоном во время работы. (Замечание: ссылки будут работать точно так же, как они работают в браузере.)

Давайте рассмотрим тот гигантский объем работы, который необходимо проделать, чтобы добиться поставленной цели. Во-первых, создайте форму, в которой размещены следующие элементы управления:

- Текстовое поле text box для ввода URL-локатора.
- Web-браузер в виде элемента управления ActiveX, как показано на рис. 2.
- Командная кнопка Go для того, чтобы начать собственно навигацию с помощью элемента управления Web-браузер.

Если рассматриваемый элемент управления ActiveX не установлен в вашей системе, вы можете бесплатно загрузить его по адресу www.microsoft.com/ie.

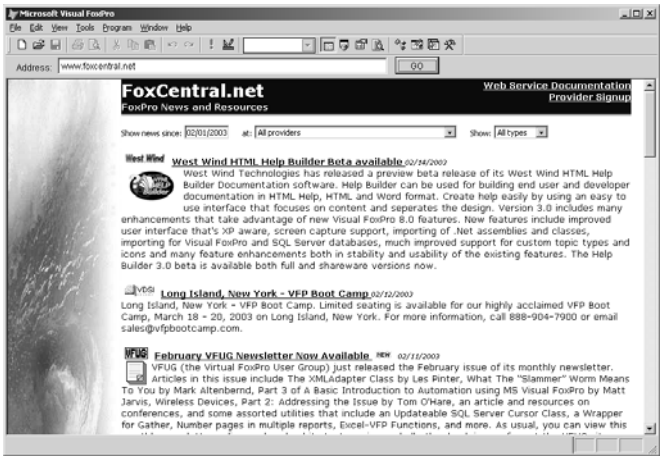


Рис. 1. Internet Explorer, встроенный в интегрированную среду разработки.

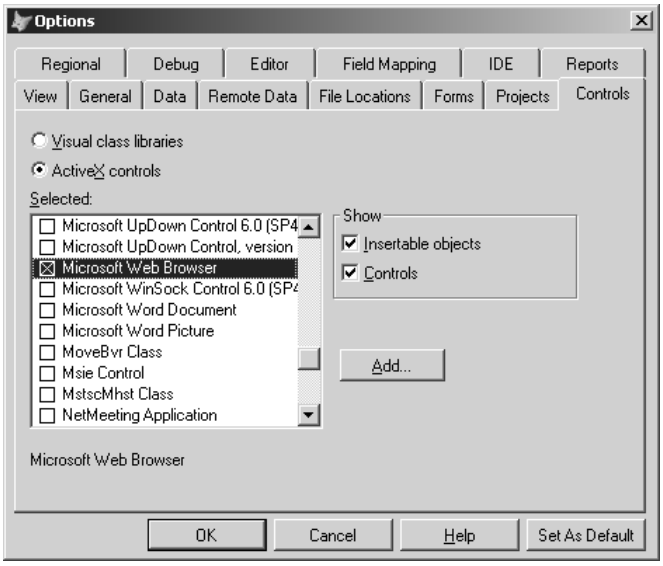


Рис. 2. Выбор элемента управления Microsoft Web browser в диалоговом окне Options в FoxPro.

Итак, создайте форму и разместите в ней перечисленные элементы управления так, как показано на рис. 3.

Далее присвойте свойствам значения, как показано в окне Properties. Как вы можете видеть в окне

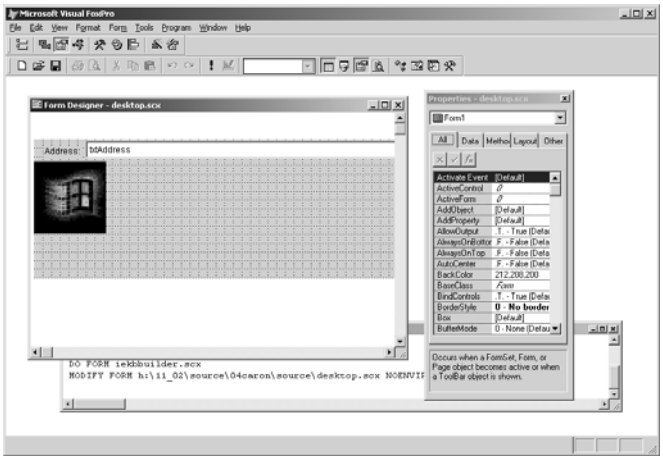


Рис. 3. Форма для рабочего стола с элементом управления ActiveX Internet Explorer.

Properties, у этой формы есть три метода, в которые помещен код данного примера. Я сейчас расскажу об этих методах:

```
* form: Init Event
* Определение инициализирующих значений
* для формы и элементов управления

WITH This
   * Указать адрес встраиваемой web-страницы
   .txtAddress.Value = "About:blank"
   * Навигация: вызвать событие click навигационной кнопки
   .Navigate()
   * Переместить элемент управления browser
   * в левую часть формы
   .oBrowser.Left    = 0
   .Visible = .T.
   * Вызвать метод resize для изменения размеров
   * элемента управления activex
   .Resize()
ENDWITH
```

Этот метод просто выполняет переход на страницу, принадлежащую самому браузеру, задает для свойства формы visible значение true и обращается к событию resize с тем, чтобы окно браузера соответствовало размерам рамки IDE. Таким образом, окно браузера заполняет собой рабочий стол.

```
* form: Resize Event
* Меняет размеры браузера
* при каждом изменении размеров формы.

WITH This.oBrowser
    * Настройка высоты браузера для того,
    * чтобы адресное поле оставалось видимым
    .Height = Thisform.Height-.Top
    * Настройка ширины браузера
    .Width  = Thisform.Width
ENDWITH

* form: Navigate
* Обращение к методу navigate
* из элемента управления ActiveX

Thisform.oBrowser.Navigate(ALLTRIM( ;
   Thisform.txtAddress.value))
```

Кроме того, по нажатию на расположенную на форме кнопку Go будет инициирован метод navigate.

```
* Кнопка Go: событие Click
*--------------------
Thisform.Navigate()
```

И наконец, есть еще одна «хитрость», о которой вы должны знать для того, чтобы добиться надлежащего функционирования от этой формы. Чтобы вы могли исполнить предлагаемую форму, обязательно надо сделать одну вещь (спасибо Кену Леви (Ken Levy) за этот совет). Необходимо добавить в метод Refresh() элемента управления браузера команду NODEFAULT. Если вы об этом забудете, то при каждом обновлении (refresh) элемента управления браузера будете получать сообщение об ошибке «OLE error code 0x80004005: Unspecified error». (У меня установлена бета-версия Windows 2003 server build 3718 и IE версии 6.0.3718. В этой конфигурации наличие или отсутствие Nodefault роли не играет, ошибка не появляется — *прим. ред.*).

После того как эта форма создана в соответствии с данными указаниями, вы можете просто исполнить ее из окна Command с помощью следующей команды: DO FORM desktop. Готово! Вот и все, что необходимо сделать для того, чтобы рассматриваемая форма заработала. Эту форму вы найдете на сопровождающей дискете. (Сам элемент управления Web browser не может быть включен в этот файл.)

Использование просматриваемой информации

Как вам, вероятно, известно, во «всемирной паутине» в огромном количестве имеется информация, которую вы могли бы использовать в своей профессиональной деятельности и которая могла бы помочь вам в работе — например, группы новостей и базы знаний, — но, на самом деле, этой информации там слишком уж много. База знаний Microsoft Knowledge Base, например, перегружена ненужными вам сведениями. Как правило, вы работаете с двумя или тремя языками программирования, используете еще какие-то инструментальные средства общего назначения и платформу Windows X.

Поиск информации в этих огромных базах знаний может оказаться утомительным занятием. После того, как вы нашли то, что искали, кому-то из ваших коллег вскоре после этого, возможно, потребуется разыскать те же самые сведения. Зачем заставлять их вести такие же утомительные поиски? Чтобы разрешить эту «проблему», я высказал идею о формировании подмножества базы знаний Microsoft Knowledge Base для нашей фирмы. В такой

базе знаний хранятся только те сведения, которые представляют интерес для меня и моих коллег. Хотя в данной статье я лишь поверхностно раскрываю свой замысел, вы подхватываете мою идею и можете ее развить дальше.

Для хранения информации нам необходима таблица. Структура этой таблицы представлена ниже.

Таблица 1. Структура таблицы.

Поле	Тип	Ширина	Decimals
Art_id	Character	10	0
Art_categ	Character	30	0
Article	Memo		
aaUrl	Memo		

Далее нам необходимо иметь FoxPro-форму со ссылкой на экземпляр объекта браузера Internet Explorer. Вид этой формы показан на рис. 4. Поскольку она включена на дискету, сопровождающую журнал, я не стану разбирать здесь весь программный код для этой формы, но рассмотрю те фрагменты, которые являются самыми важными.

На снимке с экрана вы можете видеть, что выбранный в окне Web-браузера текст помещен также на форму, предназначенную для работы с базой знаний. Это достигается путем выделения нужного текста в окне Web-браузера с последующим нажатием кнопки New, размещенной на форме базы знаний. Вслед за этим пользователю предложат выбрать соответствующую категорию для интересующей его информации; как только категория определена, выбранный текст действительно будет добавлен в «индивидуализированную базу знаний».

Самое главное, что вы должны усвоить из этого примера — это как получить доступ к информации, предоставленной «внутри» браузера (в действительности, вы хотите иметь доступ к загруженному документу). Получить доступ к данным, являющимся частью документа, достаточно просто. Браузер позволяет вам обращаться к загруженному документу как к объекту с помощью объектной модели DOM. Модель Document Object Model — это тот справочный материал, который вам следует тщательно проработать. Как только вы это сделаете, вам откроется вся мощь предоставляемых этой моделью возможностей, и вы узнаете, каким образом можно использовать эту мощь для своих нужд. В форме из рассматриваемого примера я организовал доступ к информации, выбранной в окне экземпляра объекта браузера Internet Explorer, который также показан на снимке с экрана. Чтобы предлагаемая схема работала, нам необходимо иметь возможность обращаться по ссылке

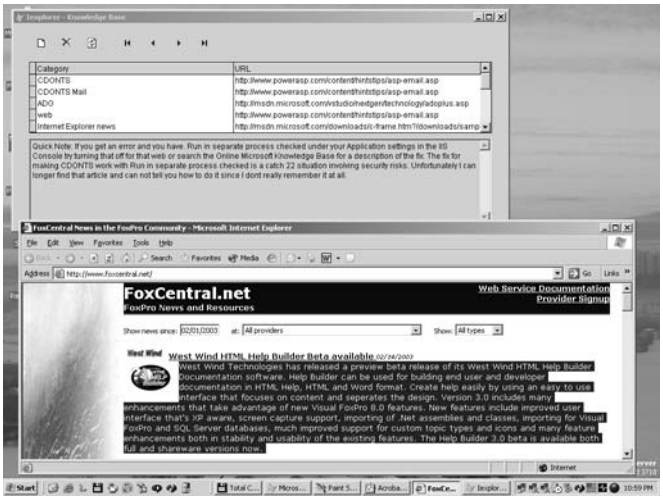


Рис. 4. Формирование вашей личной базы знаний.

из формы к экземпляру объекта браузера IE. Это делается в методе Init формы, предназначенной для работы с базой знаний:

```
* Метод Init
* создает ссылку на браузер IE и переходит на ваш
* излюбленный web-сайт, делая браузер IE видимым

WITH Thisform
  .AddProperty("oBrowser", CREATEOBJECT( ;
    "internetexplorer.application"))
  .oBrowser.Navigate("http://www.foxcentral.net")
  .oBrowser.Visible = .T.
ENDWith
```

Теперь, когда у нас есть ссылка на браузер IE, можно приступить к просмотру той информации, которую мы, возможно, захотели бы сохранить в своей собственной частной базе знаний. Я выбрал некоторый текст в окне браузера IE и нажал кнопку New, находящуюся на форме базы знаний. Код, выполняемый при нажатии на кнопку New, осуществил некоторую проверку и добавил в таблицу запись с выбранным текстом. FoxPro-код, обеспечивающий добавление записи и проверку, в данной статье не представлен, но его можно найти на сопровождающей дискете. Давайте проанализируем ту строку кода, которая получает выбранный текст из браузера IE. Происходит следующее:

```
* cmdNew: Click
lcArticle = ThisForm.oBrowser.Document. ;
   selection.createrange.htmltext

* включить URL-локатор страницы
lcAddress = ThisForm.oBrowser.LocationUrl
```

Строка кода, которая присваивает значение переменной lcArticle, достаточно длинна. Позвольте мне разбить ее на отдельные фрагменты:

- *Thisform.oBrowser* — это обращение к браузеру IE через форму.
- *Thisform.oBrowser.Document* — идем дальше на уровень документа.
- *Thisform.oBrowser.Document.Selection* — этот фрагмент строки предоставляет нам доступ к выбранному в документе тексту.
- *Thisform.oBrowser.Document.Selection.CreateRange* — создаем объект, определяющий диапазон выбора.
- *Thisform.oBrowser.Document.Selection.CreateRange.HTMLText* — возвращаем текст, связанный с объектом диапазона выбора.

Теперь несколько слов относительно той строки, в которой присваивается значение переменной lcAddress — чтобы предоставить своим коллегам возможность читать не только текст, выбранный на данной странице, я также поместил в таблицу URL-локатор той страницы, которую они могут посетить, щелкнув мышью по размещенной на форме кнопке Refresh.

Вот и все, что необходимо для создания очень простой системы, обеспечивающей формирование личной базы знаний. Это, разумеется, простой пример, но проникнитесь заложенной в нем идеей и используйте мощные возможности браузера IE в своих разработках, предназначенных для вас лично или для ваших заказчиков.

Справочная Help-система, настраиваемая пользователем

В первом примере было продемонстрировано, как можно извлечь данные из документа, загруженного в браузер IE. Однако, если возможно получение текста из документа, значит должна также существовать и возможность поместить текст в документ, верно? Но с какой целью? Ну... как насчет справочной Help-системы, которую может дополнять пользователь? Я начал с простой формы, показанной на рис. 5. Как вы можете видеть, в этой форме размещено несколько «обычных» элементов управления FoxPro, однако, «темной лошадкой» является браузер Internet Explorer, представленный в виде элемента управления ActiveX.

В списке слева будут перечислены тематические разделы Help-справки, доступные пользователю. В браузере эта информация будет представлена в HTML-виде. Кнопки, размещенные на панели инструментов, говорят сами за себя.

Вид демонстрируемой информации может быть изменен с помощью любого HTML-редактора, позволяющего модифицировать HTML-страницу. Предлагаемая HTML-страница очень проста и ее код показан ниже:

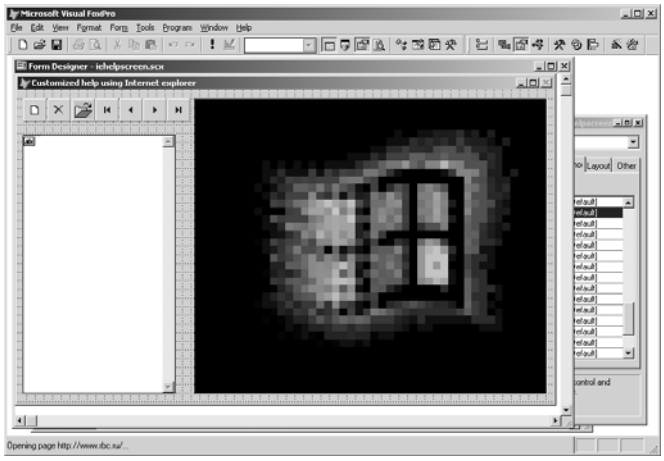


Рис. 5. Вид формы справочной Help-системы в режиме проектирования.

```
<HTML>
  <HEAD>
     <meta http-equiv="Content-Language" content="nl">
     <meta http-equiv="Content-Type" content="text/html;
➥          charset=windows-1252">
     <meta name="GENERATOR" content="Microsoft FrontPage
➥          4.0">
     <meta name="ProgId"
➥          content="FrontPage.Editor.Document">
     <title>New Page 1</title>
  </HEAD>

<BODY>
<P>
  <DIV id="Helptopic" align="center" style="font-size:16pt;
   color:red;">
     Help topic
  </DIV>
</P>

<P>
<DIV id="helptext" align="left" style="font-size:10pt;
   font-family:verdana;">
     Helptext starts here
  </DIV>
</P>
</BODY>
</HTML>
```

В этом HTML-коде добавлены два идентификатора: HelpTopic и Helptext. Эти идентификаторы позже будут использованы в нашем программном коде для выдачи на экран содержимого Help-справки взамен того текста, который виден на экране в данный момент. Когда этот HTML-код будет интерпретирован каким-либо визуальным инструментальным средством, вы увидите что-то вроде той картинки, которая показана на рис. 6. Можете воспользоваться этой страницей, чтобы модифицировать ее или дополнить ее возможности с помощью приложения FrontPage или иного редактора HTML-кода.

Большое дело, верно? Ситуация, собственно, обусловливается тем, что просмотр пользователями всей Help-справки организуется путем добавления запра-

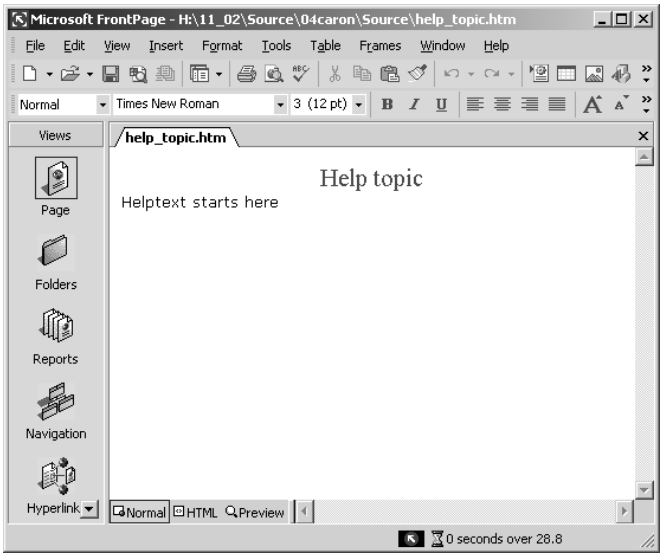


Рис. 6. Представление Help-справки в виде HTML-кода в приложении FrontPage.

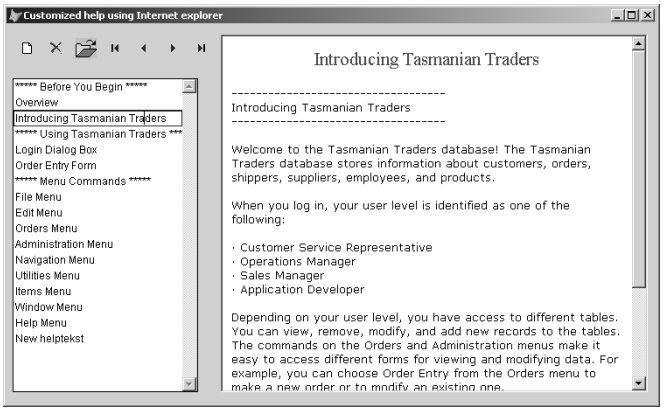


Рис. 7. Вид работающей Help-справки на экране.

шиваемой согласно выбранной теме информации к этой единственной странице. Работающая форма будет выглядеть примерно так, как показано на рис. 7.

Совершенно не отформатированные данные теперь выдаются на экран в виде HTML-страницы, которая может быть модифицирована таким образом, чтобы иметь более привлекательный вид для ее пользователей. Давайте посмотрим правде в глаза: чаще всего, разработчик — это не тот человек, который должен писать текст для справочной Help-системы. Так почему бы не предоставить возможность вносить изменения в справочную систему самим пользователям? Ведь возможность самим модифицировать наш инструмент является той отличительной чертой FoxPro, которая нам так нравится: вспомните, например, о технологии IntelliSense. И опять-таки, я демонстрирую только те отрывки кода, в которых собственно и заключен весь смысл — остальное вы можете исследовать, открыв форму, предоставленную на сопровождающей дискете.

Метод refresh браузера по-прежнему получает команду NODEFAULT по причинам, которые были объяснены в этой статье ранее. В данном конкретном случае я добавил небольшой цикл, чтобы обеспечить браузеру IE время на инициализацию и загрузку HTML-страницы надлежащим образом.

```
* Form: Init
This.oBrowser.Navigate(CURDIR() + "help_topic.htm")
DO WHILE .T.
* 4=READYSTATE_COMPLETE
  IF This.oBrowser.readyState = 4
       EXIT
  ENDIF
ENDDO
ThisForm.grdttrade.SetFocus()
```

Взаимодействие между данными и HTML-формой осуществляется следующим образом: я использую объектную модель DOM для того, чтобы через браузер добраться до документа. Вот программный код, заставляющий HTML-документ демонстрировать хранящуюся в таблице информацию для выбранного тематического раздела. Как я упомянул ранее, в этом методе я ссылаюсь на идентификаторы, которые разместил в тэгах в HTML-странице.

```
* Метод AfterRowColChange
WITH Thisform.oBrowser.Document.all
  .Helptopic.Innertext = ALLTRIM(ttrade.topic)
  .Helptext.Innertext = ALLTRIM(ttrade.details)
ENDWITH
```

Это решение — простое решение, но оно универсально и его можно развить и дополнить, а исходный код доступен на сопровождающей дискете, так что вы можете «поиграть» с ним и, возможно, воспользоваться и дополнить его так, чтобы он отвечал вашим потребностям. Если вы хотите передать этот код в эксплуатацию ряду заказчиков или использовать в каких-либо проектах, единственное, что вам необходимо сделать, это настроить соответствующим образом внешнее представление и работу HTML-файла, вот и все.

В следующий раз я покажу вам, как использовать Internet Explorer в качестве инструмента для организации управляемого данными Web-содержимого.

Реми Г. Дж. Карон (Remi G.J. Caron) является руководителем технического отдела и соучредителем фирмы BizzView B.V., Нидерланды. Он программирует на языке FoxPro начиная с версии Foxbase 1.02. Реми также является президентом секции Microsoft в группе разработчиков Software Developers Group Netherlands. Его адрес: Remi.Caron@Bizzview.com

❑❑❑

Ваш приказ — мое желание

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

WIN 7.0 DOWNLOAD

В этом месяце Энди Крамек и Марсиа Акинз рассуждают о том, как создать универсальный класс командной кнопки. Как они быстро выясняют, ключ к решению этой задачи, также как и при создании любых других классов, это правильно определить, что собственно входит в обязанности создаваемого класса, и не перегружать программный код решением других проблем. В файл-приложение к этой статье включены все необходимые классы.

Марсиа: Недавно я просматривала некоторые классы в библиотеках FoxPro Foundation Class и меня поразило, сколько же различных определений имеется там для классов командных кнопок. Более того, почти у каждого из этих классов есть программный код, размещенный прямо в событии Click() данной кнопки.

Энди: Что ж, мне известно, что ты — также как и я сам — всегда отстаивала размещение программного кода в методах, а не в событиях, но так ли уж это плохо, помещать код прямо в событие Click()? Мы в данном случае говорим о классе, следовательно, эта кнопка предназначена, по-видимому, для повторного использования.

Марсиа: Но в том-то и дело. Даже самый безопасный код может создать проблемы, если он размещен в неподобающем месте! Вот очень простой пример из библиотеки классов _MiscButtons.vcx. В этой библиотеке определен класс _cmdOK для кнопки, которая имеет заголовок “OK”, и в ее методе Click() присутствует следующий код:

```
IF TYPE("THISFORM.PARENT") = 'O'
  THISFORMSET.Release
ELSE
  THISFORM entire.Release
ENDIF
```

Энди: На первый взгляд, этот код кажется мне достаточно корректным. Хотя, поразмыслив, должен признать, я не уверен в том, что непременно ждал бы от кнопки OK деспотичного закрытия той формы, в которой она располагается, не говоря уже о закрытии набора форм. Теперь, когда я об этом размышляю, мне кажется, что такой фрагмент кода весьма опасен, поскольку предполагается, что все остальное тоже будет «как-нибудь» обработано.

Марсиа: Совершенно верно! В обязанности командной кнопки не входит решать, должна ли быть закрыта форма (или набор форм); задача командной кнопки — где-то кого-то уведомить о том, что она была нажата.

Энди: Ну хорошо, так, может быть, эта конкретная кнопка повторно используется только в определенных конкретных ситуациях. Как бы ты ее усовершенствовала?

Марсиа: Прежде, чем мы обсудим это, вернемся на шаг назад и давай беспристрастно рассмотрим вопрос о том, почему нам не нравится, когда программный код размещается в событиях. Могу назвать три причины своей неприязни, хотя полагаю, я должна признать: один только факт, что мне не нравится такой подход, не означает, что он обязательно плох. Первая причина заключается в следующем: если необходимо неоднократно обращаться к этому коду из разных мест, такое обращение, в конце концов, обретет примерно вот такой вид:

```
ThisForm.PageFrame1.Page2.cmdOK.Click()
```

Энди: Что ж, придется согласиться с тем, что данная конструкция не слишком привлекательна, и такое обращение, разумеется, не дает никакой подсказки относительно того, что должно произойти (ты всегда могла бы добавить комментарий, чтобы объяснить эту строку), но не так уж оно и плохо, верно?

Марсиа: О, не плохо? Откуда, по-твоему, ты узнаешь, редактируя код в методе Click() кнопки OK, что этот метод вызывается также откуда-то еще? Ты легко мог бы внести изменение в функциональность, что было бы вполне разумно, если бы вызов метода осуществлялся в результате явного нажатия на эту кнопку, но что было бы совершенно неуместно при обращении к этому методу откуда-то еще.

Энди: Но наверняка это справедливо для любого метода?

Марсиа: Да, но вероятность внесения изменений в базовую функциональность метода ReleaseFormSet() гораздо меньше нежели вероятность внесения изменений в то, что делает «кнопочный» метод Click(), а этот факт приводит нас ко второй причине, препят-

ствующей размещению кода в событии. Такой код не документирует сам себя, и поэтому его трудно сопровождать.

Энди: С этим я не могу спорить. Я с гораздо большим удовольствием увидел бы обращение к методу, чье имя указывает на то, что должно произойти, нежели вынужден был бы разбираться с кодом, размещенным прямо в кнопке. Какова третья причина?

Марсиа: Третья причина заключается всего лишь в том, что тебе не известно, где находится этот код. Я имею в виду, что если у тебя в форме есть метод GetNextItem(), вполне безопасно сделать ставку на то, что в этом методе нет кода, который закрывает данную форму. А что находится в методе Click() кнопки с заголовком "OK"? Как мы видели, там могло бы быть все что угодно!

Энди: Я определенно не стал бы возражать против любой из этих причин. Однако, соглашения и стандарты, касающиеся программирования, полезны только до тех пор, пока им следуют. Сами по себе они не могут служить решением проблемы неуместного или ошибочно размещенного кода. Но, поскольку мы все-таки заключаем соглашение, давай вернемся назад к рассматриваемой нами задаче о том, как бы мы могли усовершенствовать код в упомянутой кнопке OK.

Марсиа: Я не хочу усовершенствовать этот код; я хочу переместить его!

Энди: Пардон?

Марсиа: Давай сейчас вернемся на минуту к основам, не возражаешь? Что является функцией командной кнопки?

Энди: Ты уже сказала, где-то кого-то информировать о том, что она была нажата.

Марсиа: Совершенно верно. Кнопка должна откликаться на нажатие. Это подразумевает рассылку уведомления в двух направлениях. Во-первых, об этом следует сообщить приложению-владельцу: «Меня нажали». Но, что не менее важно, кнопка должна также информировать пользователя о том, что она осознала произведенное им действие. Как часто доводилось тебе видеть пользователей, которые продолжают нажимать на кнопку, потому что приложение не обеспечивает надлежащей обратной связи?

Энди: Ага, похоже, они думают, что в первый раз их не услышали или что-то в этом роде. Вот поэтому мой корневой класс для командной кнопки имеет пользовательский метод OnClick(), куда я собственно и поместил код. В методе Click() есть код, который гарантирует, что сразу же, как только пользователь нажал на эту кнопку, она блокирует сама себя до тех пор, пока не выполнятся действия, определенные в пользовательском методе. Вот так:

```
WITH This
  .Enabled = .F.
  .OnClick()
  .Enabled = .T.
ENDWITH
```

Марсиа: Не лучше ли было бы обеспечить возможность конфигурации такого функционального поведения? Я бы добавила пользовательское свойство lDisableOnClick, по умолчанию присвоила ему значение True и модифицировала твой код следующим образом:

```
WITH This
  IF .lDisableOnClick
    .Enabled = .F.
  ENDIF
  .OnClick()
  IF .lDisableOnClick
    .Enabled = .T.
  ENDIF
ENDWITH
```

Затем, если по какой-то причине блокировать кнопку не надо, нам не придется переписывать какой-либо код в этом экземпляре объекта; мы просто изменим значение свойства.

Энди: Такой подход намного лучше. Итак, теперь у нас есть класс, которому вряд ли когда-либо потребуется код в методе OnClick(), а ведь мы всего лишь переместили код в другое место. Такое имя метода говорит мне не больше, чем имя Click() о том, что собственно делает этот код.

Марсиа: Я не сказала о том, что весь код должен остаться в методе OnClick(). Это всего лишь метод-перехватчик, исполняемый «после-нажатия», поскольку он вызывается вслед за наступлением события Click(). Что меня сейчас интересует, так это вопрос: не можем ли мы и в самом деле добавить код в этот метод, чтобы сделать всю эту конструкцию совсем универсальной.

Энди: Вот так, сразу, не могу сказать. Каким образом ты могла бы написать универсальный код для элемента управления command button?

Марсиа: Что ж, подумаем над этим. Командная кнопка не может существовать независимо. Она всегда должна быть частью какого-то более крупного контейнера. Это может быть сама форма, или инструментальная панель, или часть составного элемента управления, например, навигационная панель.

Энди: А! Понимаю, что ты имеешь в виду. Кнопка всегда должна иметь некоторого рода «родителя». Одним из больших преимуществ реализации ООП в Visual FoxPro является контейнерная модель, которая позволяет нам иметь доступ к родителю объекта через открытое свойство Parent вместо того, чтобы поддерживать указатели.

Марсиа: Верно. И, как мы уже говорили, обязанностью командной кнопки является уведомление кого-то где-то в приложении о том, что она нажата. На самом базовом уровне эту функцию можно было бы реализовать как метод, обращающийся к своему родительскому контейнеру.

Энди: Так куда ты клонишь?

Марсиа: Мне кажется, что мы просто могли бы определить для командной кнопки свойство, в котором хранилось бы имя того метода, к которому необходимо обратиться. Допустим, мы назовем это свойство cAction. Тогда мы могли бы разместить в методе OnClick() создаваемого класса код для обращения к тому методу, чье имя указано в этом свойстве, вот так:

```
WITH This
  IF NOT EMPTY( .cAction )
    *** Метод указан
    IF PEMSTATUS( .Parent, .cAction, 5 )
      *** У родительского контейнера есть указанный метод,
      *** поэтому вызываем его
      lcMethod = "This.Parent." + .cAction + "()"
      &lcMethod
    ELSE
      *** У родительского контейнера нет указанного метода
      *** Здесь нам необходимо обработать эту ситуацию
    ENDIF
  ENDIF
ENDWITH
```

Энди: Я вижу, ты здесь не предусмотрела никакой возможности для возврата значения.

Марсиа: Нет, зачем бы оно нам понадобилось? Предусматривается обращение к методу родительского контейнера, в котором собственно и будет выполняться необходимая работа. Следовательно, этот метод и должен разбираться со всеми вопросами, включая возврат значения. Что стала бы делать командная кнопка с возвращенным значением? Осталось только — теперь, когда мы решили проблему обратной связи с пользователем путем блокировки кнопки — сообщить ее родительскому объекту о том, что она была нажата.

Энди: Полагаю, ты идешь по правильному пути, но с этим кодом связаны две проблемы.

Марсиа: Позволь мне высказать предположение. Этот код размещен в неподобающем месте?

Энди: Возможно. Первая проблема заключается в том, что ты исходишь из следующего предположения: у контейнера есть пользовательский метод. Что происходит, если родительский контейнер окажется страницей в страничном блоке? Вспомни, мы не можем определять дополнительные методы для страниц в том случае, если эти страницы созданы визуально. Вторая проблема: раз мы явно выполняем проверку существования конкретного метода, мы должны предусмотреть обработку ошибок в классе командной кнопки. Было бы лучше, если бы кнопка обращалась к стандартному методу, определенному для всех контейнеров, и позволила этому методу разбираться с проблемами реализации запроса.

Марсиа: Итак, ты утверждаешь, что рассматриваемый фрагмент кода находится в неподходящем месте. Что, если мы включим метод DoAction как элемент определения интерфейса для всех наших контейнерных классов? Тогда мы могли бы модифицировать класс командной кнопки таким образом, чтобы она только делегировала необходимое действие методу DoAction() своего родительского контейнера. Этим решаются обе проблемы, поскольку, если у контейнера командной кнопки нет метода DoAction(), эта кнопка все равно ничего не может сделать. Модифицированный код выглядит вот так:

```
WITH This
  IF NOT EMPTY( .cAction )
    *** Метод указан
    IF PEMSTATUS( .Parent, "DoAction", 5 )
      *** У родительского контейнера есть метод,
      *** поэтому обращаемся к нему
      .Parent.DoAction( .cAction)
    ELSE
      *** Родительский контейнер не имеет метода DoAction
      lcMsg = "Coming soon to a computer near you"
      MESSAGEBOX( lcMsg, 16, "Major Waaah!" )
    ENDIF
  ENDIF
ENDWITH
```

Энди: Но проблема с командными кнопками, размещенными на страницах, по-прежнему не решена — эти кнопки никогда ничего не делают. Я думаю, тебе необходимо в качестве «крайнего средства» прибегнуть к обращению вместо того, чтобы выдавать сообщение об ошибке. Кнопка не должна пытаться обработать ошибку, она просто должна обратиться куда-то еще! Очевидный кандидат — обращение прямо к методу DoAction() на уровне формы. Единственной загвоздкой могло бы оказаться размещение кнопки в инструментальной панели Toolbar, но я полагаю, это мы могли бы отслеживать.

Марсиа: Подожди минуту, в этом нет необходимости! Учитывая, что обращение делается с использованием указателя ThisForm, такой подход будет просто прекрасно работать в случае использования ин-

струментальной панели. Не требуется никакой особой проверки. Сейчас нам необходимо только переместить это сообщение об ошибке в метод DoAction() формы и модифицировать код в методе OnClick() следующим образом:

```
WITH This
  IF NOT EMPTY( .cAction )
    *** Метод указан
    IF PEMSTATUS( .Parent, "DoAction", 5 )
      *** У родительского контейнера есть метод,
      *** поэтому вызываем его
      .Parent.DoAction( .cAction )
    ELSE
      *** У родителя нет метода DoAction,
      *** переходим к форме Form
      ThisForm.DoAction( .cAction )
    ENDIF
  ENDIF
ENDWITH
```

Энди: Совершенно верно. И теперь у всех контейнерных классов есть один и тот же общий метод DoAction(), который выглядит вот так:

```
LPARAMETERS tcAction

WITH This
  IF PEMSTATUS( This, tcAction, 5 )
    *** У нас есть метод, поэтому вызываем его
    lcMethod = "This." + tcAction + "()"
    &lcMethod
  ELSE
    *** У нас нет метода, пытаемся обратиться
    *** к методу родителя Parent
    IF PEMSTATUS(.Parent, "DoAction", 5 )
      .Parent.DoAction( tcAction )
    ELSE
      *** У родительского объекта нет метода DoAction,
      *** переходим к форме Form
      ThisForm.DoAction( tcAction )
    ENDIF
  ENDIF
ENDWITH
```

Марсиа: На мой взгляд, этот код вполне приемлем, а единственное место, где нам необходима обработка ошибок — это класс формы. Но мы не рассмотрели каким образом передавать параметры, используя эту методологию. Впрочем, поразмыслив, я полагаю, мы могли бы довольно легко справиться с этой задачей, добавив в класс командной кнопки свойство cParameters и модифицировав код для обработки этого свойства. Довольно просто.

Энди: Итак, в каком состоянии оставляют наш исходный класс кнопки OK все эти рассуждения? Если кнопка использует предлагаемый класс, ответственность за закрытие формы должна быть возложена на форму (в которой кнопка размещена), а закрытие набора форм могло бы осуществляться в наборе форм вместо того, чтобы проделывать все это непосредственно в методе Click() командной кнопки OK. Намного лучше!

Марсиа: Да, и мы также находимся в отличной форме! У нас теперь есть полностью универсальный класс командной кнопки, чья функциональность определяется значением единственного свойства cAction. Более того, этот класс не будет разрушен, даже если указанный метод не существует. Для этого класса необходимо только, чтобы для той формы или той инструментальной панели, которые являются родителями верхнего уровня, был определен метод DoAction(). Никакого иного кода и не требуется, и в других контейнерных классах тоже. Мне это нравится!

❑❑❑

FoxTalk *русское издание*

Печатается ежемесячно

Учредитель и издатель:

ООО Эдэль. Copyright © 1992-2003. Все права защищены.

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

FoxBASE+, FoxPro® и Visual FoxPro® являются зарегистрированными товарными знаками Microsoft Corporation.

FoxTalk (русское издание) индекс 72495

Объединенный каталог индекс 45007

Журнал для FoxPro-программистов.

FoxTalk (русское издание) индекс 72496

Журнал для FoxPro-программистов вместе
с дискетой с исходными текстами программ.

FoxTalk (русское издание) индекс 72497

Подписка на старые номера журнала FoxTalk.

**Библиотека программиста индексы 72769,
72490, 72491, 47771, 80375**

Книги компьютерной тематики по последним версиям
популярных программных продуктов.

Подписка в любом почтовом отделении связи по каталогу «Газеты. Журналы» Агентства Роспечать и «Объединенному каталогу».

Подписано в печать 20/02/03. Формат 60x90 1/8. Тираж 330 экз.