FoxTalk

Ноябрь 2003
№ 11 (77)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

Многократно используемые классы данных

WIN 8.0 DOWNLOAD

Дуг Хенниг (Doug Hennig)

VFP-разработчики всегда хотели многократно использовать классы данных. И хотя в прежних версиях существовали различные подходы к решению этой задачи, зачастую они представляли собой неловкие попытки обойти ограничения, присущие продукту. Теперь, в Visual FoxPro 8, у нас есть классы, которые можно действительно использовать многократно. В этой статье Дуг рассмотрит подклассы классов CursorAdapter и DataEnvironment, создаст некоторые многократно используемые классы данных из этих подклассов и покажет, как эти классы данных могут применяться как в формах, так и в отчетах.

8.0 7.0 6.0 5.0 3.0 2.x
материал имеет отношение к соответствующей версии

UNIX MAC DOS WIN
материал имеет отношение к соответствующей платформе

DOWNLOAD
исходные тексты программ можно скачать из Интернета

В предыдущих двух статьях мы рассмотрели новый базовый класс CursorAdapter, добавленный в VFP 8. По моему мнению, это одно из наиболее важных изменений в VFP 8, обеспечивающее нас удобным для использования, согласованным интерфейсом для доступа к внешним по отношению к VFP данным, например SQL Server. Вы увидите также, что он является основой для многократного использования классов данных.

Перед тем, как перейти к многократно используемым классам данных, давайте рассмотрим некоторые созданные мною классы, унаследованные от CursorAdapter и DataEnvironment, обеспечивающие дополнительное поведение, а также являющиеся отправной точкой для наших многократно используемых классов данных.

SFCursorAdapter

SFCursorAdapter (в SFDataClasses.vcx) является подклассом CursorAdapter, обладающим некоторыми дополнительными функциональными возможностями, например следующими:

- Он может автоматически обрабатывать параметризированные запросы: мы можем определить значения параметра как статические (константа) или динамические (выражение, например такое: =Thisform.txtName.Value, вычисляемое когда курсор открыт или обновлен).
- Он может автоматически создавать индексы на курсоре после его открытия.
- Он выполняет некоторые особые вещи для ADO, например, устанавливает свойство DataSource для ADO RecordSet, устанавливает свойство ActiveConnection объекта RecordSet для объек-

та ADO Connection, а также создает и пересылает объект ADO Command методу CursorFill при использовании параметризированного запроса.
- Он обеспечивает простую обработку ошибок (свойство cErrorMessage заполняется сообщением об ошибке).
- Он имеет методы Update и Release, отсутствующие в классе CursorAdapter.

Метод Init создает две коллекции (используя новый базовый класс Collection, поддерживающий создание коллекции), одну для параметров, которые могут потребоваться для свойства SelectCmd, и одну для тегов, автоматически создаваемых после открытия курсора. Он также присваивает MULTILOCKS значение ON, поскольку это необходимо для курсоров CursorAdapter.

```
This.oParameters = createobject('Collection')
This.oTags       = createobject('Collection')
set multilocks on
```

Метод AddParameter добавляет параметр в коллекцию параметров. Передаем этому методу имя параметра (оно должно совпадать с именем, появляющимся в свойстве SelectCmd) и, по выбору, значение этого параметра (если оно не передается сразу, его можно установить позднее с помощью метода GetParameter). Приведенный ниже код демонстрирует пару новых возможностей VFP 8: новый базовый класс Empty, не имеющий свойств, событий или методов, что делает его идеальным для создания легких объектов, и функцию ADDPROPERTY(), действующую как метод AddProperty для объектов, не имеющих этого метода.

```
lparameters tcName, ;
   tuValue
local loParameter
loParameter = createobject('Empty')
addproperty(loParameter, 'Name',  tcName)
addproperty(loParameter, 'Value', tuValue)
This.oParameters.Add(loParameter, tcName)
```

Используем метод GetParameter для возвращения определенного объекта — обычно это применяется в том случае, если нужно установить значение для этого параметра.

```
lparameters tcName
local loParameter
loParameter = This.oParameters.Item(tcName)
return loParameter
```

Метод SetConnection применяется для установки свойства DataSource нужного соединения. Если свойство DataSourceType имеет значение "ODBC", то пересылаем дескриптор соединения. Если "ADO", то свойство DataSource должно указывать на объект ADO RecordSet, у которого свойство ActiveConnection указывает на открытый объект ADO Connection. Итак, пересылаем объект Connection, а метод SetConnection создаст объект RecordSet и присвоит его свойству ActiveConnection значение пересланного объекта.

```
lparameters tuConnection
 with This
   do case
     case .DataSourceType = 'ODBC'
       .DataSource = tuConnection
     case .DataSourceType = 'ADO'
       .DataSource = createobject('ADODB.RecordSet')
       .DataSource.ActiveConnection = tuConnection
   endcase
 endwith
```

Для создания этого курсора вызываем метод GetData, а не CursorFill, поскольку он обрабатывает параметры и ошибки автоматически. Передаем .T. в GetData, если хотим, чтобы курсор был создан, но не был заполнен данными. Первое, что делает этот метод, создает переменные с областью видимости private, имеющие те же имена и значения, как у параметров, определенных в коллекции параметров (метод GetParameterValue, вызываемый отсюда, возвращает значение объекта непосредственно, либо вычислением, если выражение начинается со знака "="). Далее, если мы используем ADO и здесь присутствуют какие-либо параметры, то этот код создает объект ADO Command и устанавливает для его свойства ActiveConnection ссылку на объект Connection, а затем пересылает объект Command методу CursorFill — CursorAdapter требует этого для параметризированного ADO-запроса. Если мы не используем ADO или здесь нет каких-либо параметров, тогда код просто вызывает метод CursorFill для заполнения курсора. Обратите внимание, что .T. пересылается в метод CursorFill, указывая ему использовать свойство CursorSchema, если оно заполнено (хотелось бы, чтобы этим поведением обладал базовый класс). Если курсор создан, код вызывает метод CreateTags для создания нужных индексов на курсоре; если нет, он вызывает метод HandleError для обработки любых имевших место ошибок.

```
lparameters tlNoData
 local loParameter, ;
   lcName, ;
   luValue, ;
   llUseSchema, ;
   loCommand, ;
   llReturn
 with This

 * Если мы предполагаем заполнить этот курсор
 * (в отличие от создания пустого курсора),
 * то создаем переменные для хранения любых
 * параметров. Мы должны сделать это здесь,
 * а не в методе, поскольку хотим,
 * чтобы они были определены как private.

   if not tlNoData
     for each loParameter in .oParameters
       lcName  = loParameter.Name
       luValue = .GetParameterValue(loParameter)
```

```
      store luValue to (lcName)
    next loParameter
  endif not tlNoData

* Если мы используем ADO и существуют
* какие-либо параметры, то нам
* необходим объект Command для их обработки.

  llUseSchema = not empty(.CursorSchema)
  if '?' $ .SelectCmd and (.DataSourceType = 'ADO' or ;
    (.UseDEDataSource and .Parent.DataSourceType = 'ADO'))
    loCommand = createobject('ADODB.Command')
    loCommand.ActiveConnection = iif(.UseDEDataSource, ;
      .Parent.DataSource.ActiveConnection, ;
      .DataSource.ActiveConnection)
    llReturn = .CursorFill(llUseSchema, tlNoData, ;
      .nOptions, loCommand)
  else

* Попытаемся заполнить курсор

    llReturn = .CursorFill(llUseSchema, tlNoData, ;
      .nOptions)
  endif '?' $ .SelectCmd ...

* Если мы создали курсор, то создаем индексы,
* определенные для него. Если нет,
* обрабатываем эту ошибку.

  if llReturn
    .CreateTags()
  else
    .HandleError()
  endif llReturn
endwith
return llReturn
```

Есть и еще несколько методов, но мы не будем рассматривать их сегодня – проделайте это самостоятельно. HandleError использует AERROR(), чтобы определить что идет не так и поместить второй элемент массива ошибок в свойство cErrorMessage. Повторный запрос подобен GetData, за исключением того, что он обновляет данные в курсоре. Вызываем этот метод, а не CursorRefresh, поскольку, как и в случае с GetData, он обрабатывает параметры и ошибки. Обновление выполняется просто: вызывается TABLEUPDATE() для обновления исходного источника данных или HandleError, если произойдет сбой. AddTag добавляет информацию об индексе, создаваемом нами после создания курсора, в коллекцию тегов, в то время как CreateTags, вызываемый из GetData, использует информацию из этой коллекции в операторе INDEX ON.

Существует пример использования этого класса, взятый из TestCursorAdapter.prg (доступный на сопровождающей дискете). Он получает записи из таблицы Customers базы данных Northwind, поставляемой вместе с SQL Server. Также он использует параметризированный оператор для SelectCmd, показывая, как метод AddParameter позволяет обрабатывать параметры и как можно автоматически создавать теги для курсора, используя метод AddTag.

```
lnHandle = sqlstringconnect('driver=SQL Server;' + ;
   'server=(local);database=Northwind;uid=sa;pwd=;' + ;
   'trusted_connection=no')
loCursor = newobject('SFCursorAdapter', 'SFDataClasses')
with loCursor
  .DataSourceType = 'ODBC'
  .Alias          = 'Customers'
  .SelectCmd      = 'select * from customers ' + ;
    'where country = ?pcountry'
  .SetConnection(lnHandle)
  .AddParameter('pcountry', 'Brazil')
  .AddTag('CustomerID', 'CustomerID')
  .AddTag('Company',    'upper(CompanyName)')
  .AddTag('Contact',    'upper(ContactName)')
  if .GetData()
    messagebox('Brazilian customers in CustomerID order')
    set order to CustomerID
    go top
    browse
    messagebox('Brazilian customers in Contact order')
    set order to Contact
    go top
    browse
    messagebox('Canadian customers')
    loParameter = .GetParameter('pcountry')
    loParameter.Value = 'Canada'
    .Requery()
    browse
  else
    messagebox('Could not get the data. The error ' + ;
      'message was:' + chr(13) + chr(13) + .cErrorMessage)
  endif .GetData()
endwith
sqldisconnect(lnHandle)
```

SFDataEnvironment

Подкласс SFDataEnvironment (также из SFDataClasses.vcx) намного проще, чем SFCursorAdapter, но к нему также добавлены некоторые полезные функциональные возможности:

- Метод GetData класса SFDataEnvironment вызывает метод GetData всех членов класса SFCursorAdapter, так что не приходится вызывать их индивидуально. Подобным образом методы Requery и Update вызывают методы Requery и Update каждого члена класса SFCursorAdapter.
- Подобно классу SFCursorAdapter, метод SetConnection устанавливает для свойства DataSource значение ADO RecordSet, а для свойства ActiveConnection объекта RecordSet ссылку на объект ADO Connection. Однако он также вызывает метод SetConnection любого члена класса SFCursorAdapter, свойство UseDEDataSource которого имеет значение .F..
- Он обеспечивает простую обработку ошибок (свойство cErrorMessage заполняется сообщением об ошибке).
- Он обладает методом Release.

Мы рассмотрим только пару методов этого класса. Метод GetData очень прост: он только вызывает метод GetData любого объекта класса SFCursorAdapter, имеющего этот метод.

```
lparameters tlNoData
 local loCursor, ;
   llReturn
 for each loCursor in This.Objects
   if pemstatus(loCursor, 'GetData', 5)
```

```
    llReturn = loCursor.GetData(tlNoData)
    if not llReturn
      This.cErrorMessage = loCursor.cErrorMessage
      exit
    endif not llReturn
  endif pemstatus(loCursor, 'GetData', 5)
next loCursor
return llReturn
```

Метод SetConnection более сложен: он вызывает метод SetConnection любого объекта класса SFCursorAdapter, имеющего этот метод, при условии что свойство UseDEDataSource объекта имеет значение .F., а затем использует код, подобный тому, что находится в классе SFCursorAdapter, для установки свойства DataSource объекта, если свойство UseDEDataSource любого объета класса CursorAdapter имеет значение .T..

```
lparameters tuConnection
local llSetOurs, ;
  loCursor, ;
  llReturn
with This

* Вызывает метод SetConnection любого объекта
* CursorAdapter, не использующего
* существующее значение свойства DataSource.

  llSetOurs = .F.
  for each loCursor in .Objects
    do case
      case upper(loCursor.BaseClass) <> 'CURSORADAPTER'
      case loCursor.UseDEDataSource
        llSetOurs = .T.
      case pemstatus(loCursor, 'SetConnection', 5)
        loCursor.SetConnection(tuConnection)
    endcase
  next loCursor

* Если мы нашли какой-либо объект CursorAdapter,
* использующий существующее значение свойства DataSource,
* нам потребуется указать собственный источник.

  if llSetOurs
    do case
      case .DataSourceType = 'ODBC'
        .DataSource = tuConnection
      case .DataSourceType = 'ADO'
        .DataSource = createobject('ADODB.RecordSet')
        .DataSource.ActiveConnection = tuConnection
    endcase
  endif llSetOurs
endwith
```

TestDE.prg демонстрирует применение SFDataEnvironment в качестве контейнера для пары классов SFCursorAdapter. Поскольку этот пример использует ADO, каждому объекту класса SFCursorAdapter потребуется его собственное значение свойства DataSource, поэтому для свойства UseDEDataSource установлено значение .F. (значение по умолчанию). Обратите внимание, что единственный вызов метода SetConnection для DataEnvironment заботится об установке свойства DataSource для каждого объекта CursorAdapter.

```
loConn = createobject('ADODB.Connection')
loConn.ConnectionString = 'provider=SQLOLEDB.1;' + ;
  'data source=(local);database=Northwind;uid=sa;' + ;
  'pwd='
loConn.Open()
```

```
set classlib to SFDataClasses
loDE = createobject('SFDataEnvironment')

with loDE
  .AddObject('CustomersCursor', 'SFCursorAdapter')
  with .CustomersCursor
    .Alias          = 'Customers'
    .SelectCmd      = 'select * from customers'
    .DataSourceType = 'ADO'
  endwith
  .AddObject('OrdersCursor', 'SFCursorAdapter')
  with .OrdersCursor
    .Alias          = 'Orders'
    .SelectCmd      = 'select * from orders'
    .DataSourceType = 'ADO'
  endwith
  .SetConnection(loConn)
  if .GetData()
    select Customers
    browse nowait
    select Orders
    browse
  else
    messagebox('Could not get the data. The error ' + ;
      'message was:' + chr(13) + chr(13) + .cErrorMessage)
  endif .GetData()
endwith

loConn.Close()
```

Многократно используемые классы данных

Теперь, когда у нас для работы есть подклассы CursorAdapter и DataEnvironment, давайте поговорим о многократно используемых классах данных.

Одной из функциональных возможностей, которую разработчики долгое время просили Microsoft добавить в VFP, является многократно используемые среды данных. Например, у вас может быть некоторая форма и отчет, имеющие абсолютно одинаковые установки данных, но вам приходится вручную заполнять объект DataEnvironment для каждого из этих объектов, потому что DataEnvironment не является многократно используемым классом. Некоторые разработчики (и почти все производители каркасов приложений) облегчили создание многократно используемого класса DataEnvironment, создавая классы в самом коде (дочерние классы невозможно создать в графической среде разработки) и используя объект loader на форме для инициализации подкласса DataEnvironment. Однако это было очередной неуклюжей попыткой сделать продукт лучше, чем он есть на самом деле, и не помогало в случае с отчетами.

Теперь, в VFP 8, у нас есть возможность создавать как многократно используемые классы данных, способные предоставлять курсоры от любого источника данных любому потребителю, который в них нуждается, так и многократно используемую среду окружения, в которой можно размещать классы данных. На данный момент вы не можете использовать подклассы CursorAdapter или DataEnvironment в отчете непосредственно, но можете программно добавить подкласс CursorAdapter (например, в метод Init

Таблица 1. Свойства CustomersCursor.

Свойство	Значение
Alias	Customers
CursorSchema	CUSTOMERID C(5), COMPANYNAME C(40), CONTACTNAME C(30), CONTACTTITLE C(30), ADDRESS C(60), CITY C(15), REGION C(15), POSTALCODE C(10), COUNTRY C(15), PHONE C(24), FAX C(24)
KeyFieldList	CUSTOMERID
SelectCmd	select * from customers
Tables	CUSTOMERS
UpdatableFieldList	CUSTOMERID, COMPANYNAME, CONTACTNAME, CONTACTTITLE, ADDRESS, CITY, REGION, POSTALCODE, COUNTRY, PHONE, FAX
UpdateNameList	CUSTOMERID CUSTOMERS.CUSTOMERID, COMPANYNAME CUSTOMERS.COMPANYNAME, CONTACTNAME CUSTOMERS.CONTACTNAME, CONTACTTITLE CUSTOMERS.CONTACTTITLE, ADDRESS CUSTOMERS.ADDRESS, CITY CUSTOMERS.CITY, REGION CUSTOMERS.REGION, POSTALCODE CUSTOMERS.POSTALCODE, COUNTRY CUSTOMERS.COUNTRY, PHONE CUSTOMERS.PHONE, FAX, CUSTOMERS.FAX

DataEnvironment) для использования преимуществ многократного использования.

Давайте создадим классы данных для таблиц Customers и Orders БД Northwind. CustomersCursor (в NorthwindDataClasses.vcx) является подклассом SFCursorAdapter и имеет свойства, описанные в таблице 1.

Вряд ли вы подумали, что я печатал значения для всех этих свойств в окне свойств (property window), не так ли? Конечно, нет! Для выполнения этой работы я использовал конструктор CursorAdapter. Хитрость заключается в том, чтобы активировать опцию Use connection settings in builder only, ввести информацию о соединении (так, чтобы у вас было «живое» соединение), а затем — SelectCmd и использовать этот конструктор для добавления остальных свойств.

Теперь, если потребуются записи из таблицы Customers БД Northwind, можно просто использовать класс CustomersCursor. Конечно, мы не определили какой-либо информации о соединении, но это на самом деле неплохо, поскольку этот класс не должен заботиться о том, как получить данные (ODBC, ADO или XML) или какое ядро баз данных используется (существуют БД Northwind для SQL Server, Access и новая для VFP 8).

Однако обратите внимание, что этот курсор имеет дело со всеми записями таблицы Customers. Но иногда вам нужен только определенный заказчик. Итак, CustomerByIDCursor является подклассом CustomersCursor со свойством SelectCmd, замененным на select * from customers where customerid = ?pcustomerid, и следующим кодом в методе Init:

```
lparameters tcCustomerID
dodefault()
This.AddParameter('pCustomerID', tcCustomerID)
```

Это создает параметр, называемый pCustomerID (имеющий то же имя, что было определено в SelectCmd), и устанавливает его в любое переданное значение. Если ни одного значения не передано, используем GetParameter для возвращения объекта для этого параметра и установки его свойства Value перед вызовом GetData.

Класс OrdersCursor подобен CustomersCursor за исключением того, что получает все записи из таблицы Orders, а OrdersForCustomerCursor является подклассом, получающим эти записи только для определенного заказчика.

Чтобы проверить как это работает, запустите TestCustomersCursor.prg. Программа получит единственного заказчика из таблицы Customers в версии БД Northwind для SQL Server, а затем проделает то же самое для Access-версии. Это показывает, что отсутствие определенной информации о соединении в самом классе делает его более гибким и, следовательно, многократно используемым.

Пример: форма

Теперь, когда у нас есть некоторые многократно используемые классы данных, давайте попробуем применить их. Сначала создадим подкласс класса SFDataEnvironment, называемый CustomersAndOrdersDataEnvironment, содержащий классы CustomerByIDCursor и OrdersForCustomerCursor. Его свойство AutoOpenTables имеет значение .F., поскольку информацию о соединении нужно установить до того, как таблица может быть открыта. Свойство UseDEDataSource для обоих объектов CursorAdapter имеет значение .T.. Теперь объект DataEnvironment может использоваться на форме для отображения информации об определенном заказчике, включая и его заказы.

Давайте создадим подобную форму. CustomerOrders.scx имеет свойства DEClass и DEClassLibrary, установленные в значения “CustomersAndOrdersDataEnvironment” и “NorthwindDataClasses.vcx” соот-

ветственно, поэтому используем наш многократно используемый объект DataEnvironment. Он содержит значительный кусок кода в методе Load, потому что поддерживает источники данных ADO, ODBC и XML и создает свои собственные соединения, так что большая часть этого кода относится к обработке этих проблем. Если бы он поддерживал только один тип, например ODBC, и другой объект отвечал бы за управление соединениями (как это может быть в реальном приложении), код мог быть также прост, как приведенный ниже:

```
with This.CustomersAndOrdersDataEnvironment

 * Получаем соединение.

   lnHandle = oApp.oConnectionMgr.GetConnectionHandle()
   .SetConnection(lnHandle)

 * Определяем, что значение для параметров курсора будет
 * заполнено из поля ввода CustomerID.

   loParameter       = ;
     .CustomerByIDCursor.GetParameter('pCustomerID')
   loParameter.Value = '=Thisform.txtCustomerID.Value'
   loParameter       = ;
     .OrdersForCustomerCursor.GetParameter('pCustomerID')
   loParameter.Value = '=Thisform.txtCustomerID.Value'

 * Создаем пустые курсоры
 * и выводим сообщение об ошибке, если
 * это не будет выполнено.

   if not .GetData(.T.)
     messagebox(.cErrorMessage)
     return .F.
   endif not .GetData(.T.)
 endwith
```

Этот код использует метод GetParameter для обоих объектов CursorAdapter и присваивает свойству Value параметра pCustomerID текст из поля ввода на форме. Обратите внимание на применение "=", означающее, что свойство Value вычисляется каждый раз, когда оно требуется, поэтому это, по существу, динамический параметр (при заполнении пользователем текстового поля необходимо изменять параметр в текущее значение). Метод GetData вызывается для создания пустых курсоров, поэтому привязка данных этих элементов интерфейса будет работать.

Поле ввода txtCustomerID ни к чему не привязано. Оно вызывает метод Requery объекта DataEnvironment, следующий за методом Refresh самой формы. Это приводит к тому, что при введении идентификатора (ID) заказчика курсоры повторно запрашиваются, а элементы интерфейса обновляются. Другие поля ввода на этой форме связаны с полями в курсоре Customers, созданном объектом CustomersByIDCursor. Сетка (grid) привязана к курсору Orders, созданному объектом OrdersForCustomerCursor.

Запустим форму и введем "ALFKI" в качестве идентификатора заказчика (см. рис. 1). Если перейти с помощью табуляции с текстового поля, то появится информация об адресе заказчика и его заказах. Попытаемся изменить что-либо в этой информации о заказчике или заказе и затем закроем форму. Запустим ее снова, введем еще раз "ALFKI". Как видите, изменения, сделанные нами, были записаны в серверную часть БД без каких-либо усилий с нашей стороны.

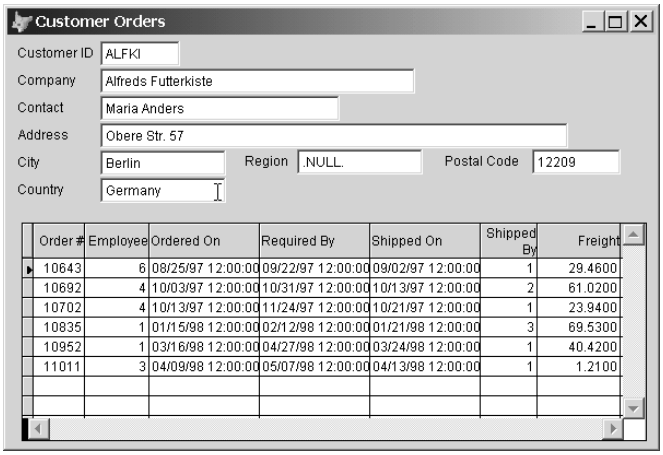

Рис. 1. Форма CustomerOrders.scx показывает как многократно используемый объект DataEnvironment может применяться на форме.

Неплохо, неправда ли? И не намного больше работы, чем при создании формы, основанной на локальных таблицах или представлениях. И что еще важнее, если изменить константу ccDATASOURCETYPE, определенную в методе Load, на "ADO" или "XML", то эта форма будет выглядеть и работать точно так же. (Если вы хотите использовать XML, то обратите внимание, что придется установить виртуальный каталог SQLXML для БД Northwind, как это описано в моей предыдущей статье, и скопировать в него файлы шаблонов XML, доступные на дискете).

Пример: отчет

Теперь давайте попробуем создать отчет. Наибольшая проблема заключается в том, что, в отличие от формы, мы не можем приказать отчету использовать подкласс DataEnvironment, а также выбрать подкласс CursorAdapter в среде окружения (DataEnvironment). Итак, мы должны поместить некоторый код в этот отчет, чтобы добавить подкласс CursorAdapter в DataEnvironment. Выглядит логичным поместить этот код в событие BeforeOpenTables среды окружения (DataEnvironment) отчета, но, на

самом деле, это не будет работать, поскольку при просмотре отчета BeforeOpenTables срабатывает на каждой странице (по причине, которую я не понимаю). Итак, поместим этот код в метод Init. Как и в случае с CustomerOrders.scx, CustomerOrders.frx имеет более сложный код, чем требуется для демонстрационных нужд. Если не учитывать этих требований, он будет также прост, как приведенный ниже:

```
with This
   set safety off

 * Получаем соединение.
   .DataSource = oApp.oConnectionMgr.GetConnectionHandle()

 * Создаем объекты CursorAdapter
 * для Customers и Orders.
   .NewObject('CustomersCursor', 'CustomersCursor', ;
     'NorthwindDataClasses')
   .CustomersCursor.AddTag('CustomerID', 'CustomerID')
   .CustomersCursor.UseDEDataSource = .T.
   .NewObject('OrdersCursor', 'OrdersCursor', ;
     'NorthwindDataClasses')
   .OrdersCursor.AddTag('CustomerID', 'CustomerID')
   .OrdersCursor.UseDEDataSource = .T.

 * Получаем данные и выводим сообщение
 * об ошибке в случае сбоя.

   if not .CustomersCursor.GetData()
     messagebox(.CustomersCursor.cErrorMessage)
     return .F.
   endif not .CustomersCursor.GetData()

   if not .OrdersCursor.GetData()
     messagebox(.OrdersCursor.cErrorMessage)
     return .F.
   endif not .OrdersCursor.GetData()

 * Устанавливаем связь от Customers к Orders.
   set relation to CustomerID into Customers
 endwith
```

Этот код больше, чем для формы, потому что мы не можем использовать подкласс DataEnvironment и должны отдельно кодировать это поведение.

Как без особых усилий поместить эти поля в отчет? Поскольку на этапе разработки в среде окружения отсутствуют объекты CursorAdapter, мы не можем просто перетащить поля из них в отчет. Вот подсказка: создадим программу, создающую курсоры и оставляющую их в области видимости (либо с помощью приостановки, либо делая объекты CursorAdapter с областью видимости public), а затем используем функцию Quick Report для помещения полей с нужными размерами в отчет.

Рисунок 2 показывает, что этот отчет выглядит так же, как при предварительном просмотре. Как и в случае с формой, можно попробовать изменить оператор #DEFINE в методе DataEnvironment.Init, чтобы проверить, как он работает с другими типами источников данных.

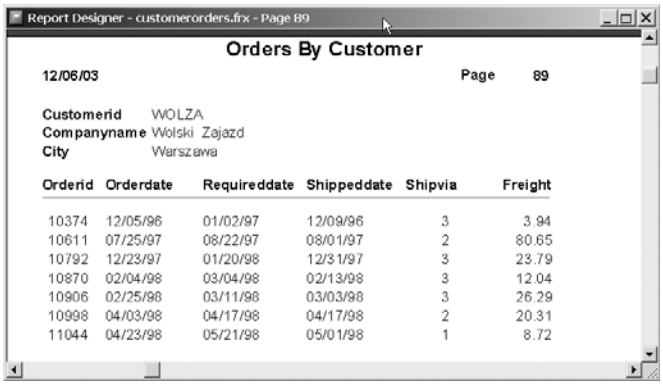

Рис. 2. Настройка отчетов требует больше усилий, но и они могут реализовать преимущества многократно используемых классов данных.

Заключение

На этом мы заканчиваем изучение нового базового класса CursorAdapter. Его появление очень взволновало меня, и я планирую исправить компоненты для обработки данных моего каркаса приложения, чтобы полностью использовать его преимущества.

В следующей статье мы узнаем, что обработка ошибок в VFP 8 была значительно улучшена с помощью новых структурированных команд обработки ошибок TRY ... CATCH ... FINALLY ... ENDTRY и нового базового класса Exception.

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

❑❑❑

Проблемы адресации

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

Одна из вечных дилемм, с которой приходится сталкиваться разработчикам баз данных, это вопрос о том, как лучше сохранять имена и адреса. В данной статье Энди Крамек и Марсиа Акинс изучат эту проблему и попытаются разобраться со всеми трудностями, связанными с логическим проектированием данных.

Марсиа: В последнее время я проектирую структуру данных для приложения и столкнулась с затруднением, связанным с тем, как наилучшим образом сохранять информацию об именах и адресах различных сущностей, с которыми это приложение работает. Главная проблема заключается в том, что у нас есть «клиенты», которые могут быть как организациями, так и частными лицами и, в то же время, частные лица могут принадлежать одной или нескольким организациям. Эти требования еще более усложняются тем фактом, что организация может иметь несколько связанных с нею разных адресов. Это могут быть как географические (например, региональные или головные офисы), так и функциональные адреса (скажем, рассылки (Mailing), отправки (Ship To), получения счетов (Billing)).

Энди: Полагаю, что частные лица также могут иметь несколько адресов, один или более из которых ассоциируются с различными организациями, к которым это частное лицо принадлежит.

Марсиа: Совершенно верно! Попытка представить все это в совокупности — моя головная боль. Полагаю, ты являешься экспертом в этом вопросе. Как ты справляешься с этим?

Энди: Проектирование модели данных не отличается от проектирования любого другого элемента программного обеспечения. Просто нужно знать в точности, какие задачи она должна выполнять и приспособить подходящие парадигмы.

Марсиа: А вот это одно из наиболее затертых слов, известных мне! Похоже на то, что мир остановится, если я не буду слышать каждый день о «смене парадигм». Но в этом случае ты, по крайней мере, используешь его корректно, поскольку оно означает «пример или шаблон, в особенности тот, что лежит в основе теории или методологии». Итак, что является наилучшей парадигмой для сохранения имен и адресов?

Энди: Давай начнем с того, что точно решим, о чем идет речь. Обычно я использую другой метод обсуждения, но в случае данных я предпочитаю начать с пары конкретных примеров, а затем абстрагировать требуемую обобщенную схему из них.

Марсиа: Хорошо, познакомься с г-жой Анитой Дринк. Она — судебный секретарь, работающий для хорошо известной адвокатской конторы «Dewey Cheetem and Howe», владельцем которой является знаменитый адвокат Лес Сеймур-Кэш. Эта фирма расположена в Кливленде, штат Огайо, с филиалом в Акроне, где и работает Анита (проживает она также в Акроне).

Энди: Итак, г-жа Дринк является частным лицом и может быть связана, по крайней мере, с двумя адресами: домашним и рабочим.

Марсиа: Да, и это означает, что существует связь «один ко многим» между Анитой и ее адресами, поэтому мы явно не сможем сохранить эту информацию в одной таблице. Нам необходимо две: одна для имени и связанная с ней таблица для адресов (см. рис. 1).

Энди: С точки зрения г-жи Дринк, это правильно. Но, конечно, оба эти адреса могут быть адресами также и других людей. Все сотрудники ее фирмы используют такой же рабочий адрес, и любой из членов ее семьи имеет такой же домашний адрес. Следовательно, правильной моделью будет связь «многие ко многим». Хорошо известно, что связь «многие ко многим» нельзя смоделировать напрямую – необходима «соединяющая» таблица (иногда называемая «распределяющей» таблицей) для того, чтобы разбить ее на две связи «один ко многим» (см. рис. 2).

Марсиа: Но как мы узнаем, какой адрес кому принадлежит?

Энди: Ты права. Для того чтобы построить правильную модель, нам необходимо включить некоторый индикатор, определяющий связь, представленную каждой соединяющей записью.

Марсиа: Опять ты используешь красиво звучащие слова, смысл которых не ясен. Наверное, ты имеешь

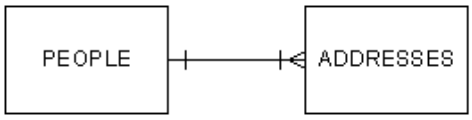

Рис. 1. Обработка простых имен и связей адресов.

Рис. 2. Обработка сложного имени и связей адресов.

в виду, что когда мы создаем запись в соединяющей таблице, нам необходимо иметь возможность определить тип адреса. Я полагаю, что это может быть сделано с помощью ключа к записи в отдельной справочной таблице для типа адреса. Единственный вопрос заключается в том, должен ли этот ключ храниться в таблице Addresses или в связывающей таблице?

Энди: Если мы сохраним его в таблице Addresses, то ограничим каждый адрес только одной функцией. Поэтому, если один адрес имеет две или более функций, то его придется вводить несколько раз. Однако если мы сохраним адрес в связывающей таблице, то сможем связывать любое число типов с одним физическим адресом (см. рис. 3).

Марсиа: Действительно ли нам требуется такая возможность?

Энди: Да. Ты упомянула в своем примере, что фирма может иметь различные адреса для почты, получения счетов и т. д. Однако один и тот же адрес можно использовать для всех этих функций. Поэтому нам необходима возможность иметь множество указателей на одну адресную запись. Кроме того, у нас по-прежнему остается одна неразрешенная проблема.

Марсиа: Я знаю! Как мы определим на уровне представления (интерфейса пользователя), что означает изменение в домашнем адресе Аниты? Вся семья переехала (в этом случае нам необходимо редактировать существующую адресную запись)? Или Анита разошлась со своим мужем и съехала (в этом случае нам необходимо добавить новую адресную запись и

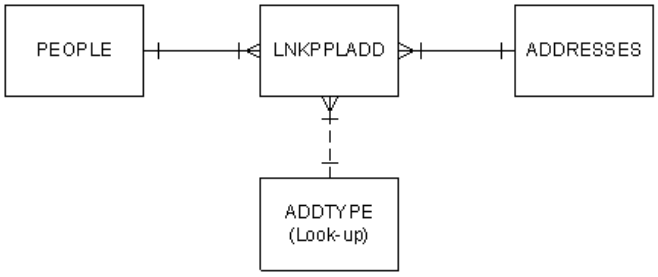

Рис. 3. Связывание типа с именем и адресом.

редактировать ссылку для Аниты)? Если мы ошибемся, то получится, что если Анита изменила место работы, то вся фирма переедет вместе с ней.

Энди: Это не так. Говорю еще раз: пока это может быть проблемой, не важно, как ты делаешь это. Это детали реализации и они не имеют отношения к проектированию модели данных. Придерживайся сути, пожалуйста! Проблема, над которой мне пришлось потрудиться, — что делать с телефонными номерами? Как много телефонных номеров может быть у Аниты и где мы должны хранить их?

Марсиа: Не уверена, что понимаю тебя.

Энди: Ну подумай сама. На самом деле, большая часть телефонных номеров связана с адресом. Поэтому Анита будет иметь один (или более) номеров, связанных с ее домом, и по крайней мере один, связанный с ее местом работы. Если Анита сменит работу, телефонный номер в фирме, используемый ею для работы, не изменится, изменится ее связь с ним. Это наводит на мысль, что мы, возможно, должны связать телефонный номер с адресом.

Марсиа: Но это не всегда верно! Анита может иметь телефонный номер, связанный только с ней, например, номер сотового телефона. И, раз уж мы заговорили об этом, речь может идти не только о телефонных номерах. Как насчет номеров факсов и пейджеров?

Энди: Тогда, полагаю, мы также должны учесть адреса электронной почты и Web-сайтов. Эти адреса также могут быть либо персональными, либо связанными с местом работы. Подобная проблема существует и с этим видом данных. Иногда они связаны с некоторым адресом, а иногда относятся непосредственно к некоторому лицу.

Марсиа: И в чем же заключается решение?

Рис. 4. Иерархическая структура адреса.

Энди: Со строго логической точки зрения, можно просто рассматривать все их как некоторый тип адреса. Единственное различие заключается в том, что все эти элементы являются единичными элементарными кусочками данных, в то время как почтовый адрес состоит из нескольких составляющих. Поэтому, если мы просто сохраняем любой адрес в поле с типом данных memo, то решим проблему.

Марсиа: Это очевидная глупость! Они отличаются функционально, поскольку почтовый адрес не просто один однородный кусочек информации. Он является иерархией, в которой каждый элемент значимо независим. Можно, если это действительно необходимо, сохранить каждый элемент адреса в различных таблицах подобно тому, как это изображено на рис. 4.

Энди: Я вижу, что ты все поняла. Мы сохранили номер улицы или телефонный номер (и так далее) в качестве деталей адреса, но только почтовый адрес будет иметь дополнительные ключи, связанные с таблицами почтовых кодов, городов, штатов и стран (postal code, city, state, country). Да, подобная структура сможет работать, но она будет очень громоздкой. На твое усмотрение может оказаться лучшим вариантом выделить телефонные номера (и связанные элементы) в их собственную таблицу.

Марсиа: Это имеет смысл. Если этого не сделать и потребуется связать телефонный номер и адрес, то не обойтись без рефлексивного соединения (reflexive join), а это может оказаться настоящей бедой, если таблицы будут большими. Кроме того, это имеет смысл с точки зрения самого бизнеса. Структура на рис. 4 будет удачной, если у вас есть полные адреса, но для того, чтобы приспособить ее к неполным адресам (без почтового кода или с отсутствующим штатом, например), потребуется иметь все внешние ключи высшего уровня в каждой таблице. Ясно, что мы должны сохранить почтовые адреса как единую сущность, а не множество сущностей.

Энди: Да, согласен с тобой. Мы сохраним почтовые адреса в таблице PAddresses, а номера телефонов, факсов и пейджеров, адреса Web-страниц и электронной почты в таблице электронных адресов EAddresses (см. рис. 5). Это означает, что соединяющая таблица теперь имеет три ключа, по одному для person, paddress и eaddress. Нам необходимо убедиться, что база данных выполняет правило, согласно которому, по меньшей мере, должны присутствовать два ключа.

Марсиа: С точки зрения данных это, может быть, и правильно. С точки зрения бизнеса нам, на самом деле, необходимо убедиться, что всегда существует ключи для name и type, а также ключ для eaddress либо для paddress (или оба сразу).

Энди: Единственной остающейся проблемой является то, что мы не можем напрямую связать eaddress с paddress.

Марсиа: А зачем тебе это нужно? Возьмем для примера номер домашнего телефона Аниты. Пока она живет в этом месте, он связан с ее домашним адресом. Если она переезжает, произойдет одна из трех вещей: она может перевести свой номер телефона на новый адрес; она может отключить телефон; номер может быть переоформлен на нового собственника жилья. Как ни крути, а номер телефона не имеет независимой связи с адресом, только через лицо, проживающее там.

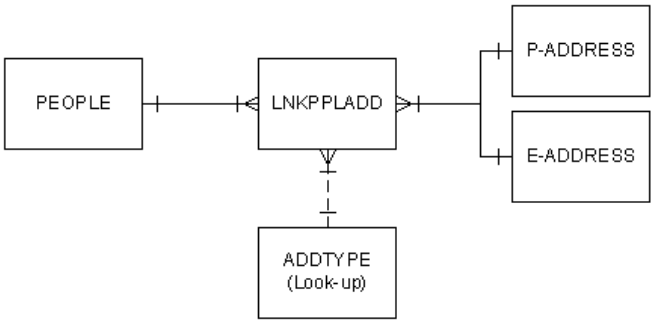

Рис. 5. Обработка имен, почтовых и электронных адресов.

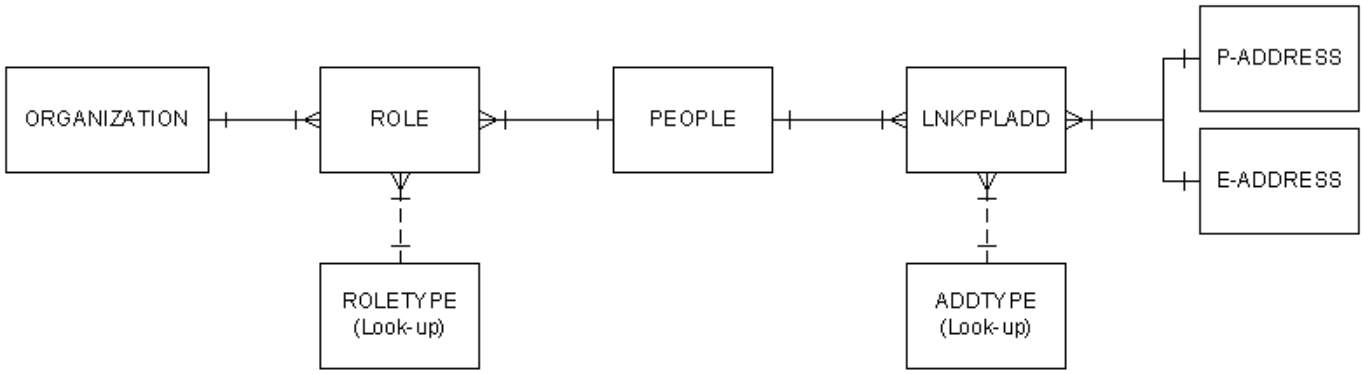

Рис. 6. Добавление организаций и должностных обязанностей.

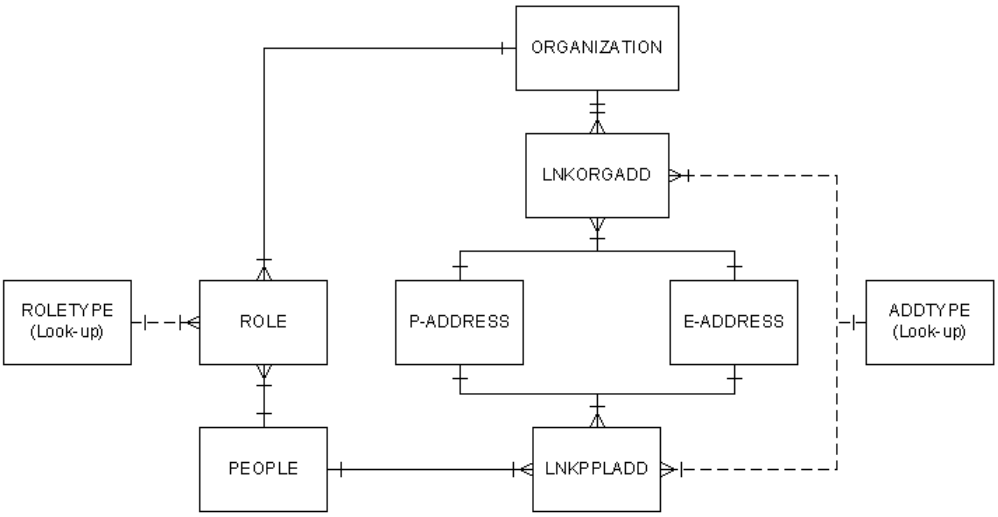

Рис. 7. Добавление адресов организаций.

Энди: Это верно для частных лиц, вроде Аниты, но что можно сказать о ее бизнесе? С фирмой “Dewey Cheetem and Howe”, по-видимому, можно будет всегда связаться по ее почтовым и телефонным адресам, независимо от того, кто работает в ней (до тех пор, пока они не обанкротятся, конечно).

Марсиа: Это отдельная проблема. Я вижу выход в том, что нам необходимо сохранять организации в их собственных таблицах и связывать людей с организациями через их должностные обязанности.

Энди: Эта связь (между организацией и ее сотрудниками) также является связью типа «многие ко многим». В организации имеется множество должностных обязанностей, а отдельный сотрудник не только может выполнять множество обязанностей в одной организации, но также работать в нескольких организациях.

Марсиа: Поэтому нам необходимо добавить другую таблицу соответствия для типов должностных обязанностей и соединяющую таблицу для того, чтобы разбить связь «многие ко многим». Таким способом можно соединить отдельное лицо с его организацией(-ями) (см. рис. 6).

Энди: Все это выглядит прекрасно, но у меня есть один вопрос. Как же мы теперь будем сохранять адресную информацию (как paddress, так и eaddress) для организации?

Марсиа: Почему бы не делать это точно так же, как для личных адресов? Просто добавь необходимую соединяющую таблицу между таблицей organization и существующими таблицами адресов (см. рис. 7).

Энди: Это выглядит хорошо только на первый взгляд, но при дальнейшей проверке не выдержива-

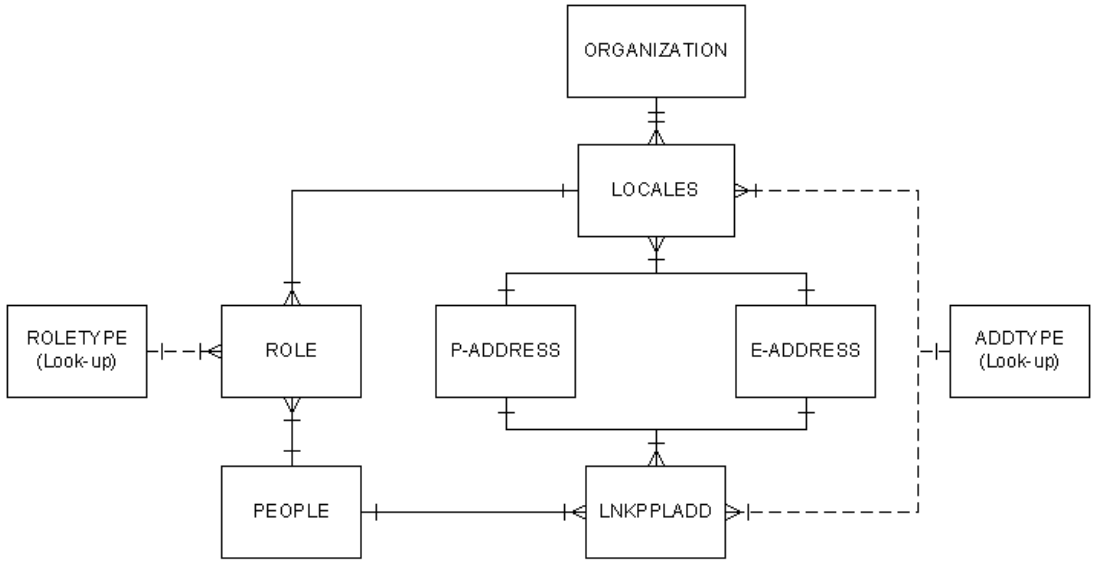

Рис. 8. Связывание должностных обязанностей с расположением организаций.

ет критики. Как мы отмечали ранее, организации могут иметь несколько почтовых адресов. Более того, каждое физическое расположение, возможно, связано с несколькими телефонными номерами, адресами электронной почты и так далее. Исходя из этого набора условий, структура, предлагаемая тобою, делает невозможным связать Аниту ни с ее рабочим адресом, ни с номером ее рабочего телефона.

Марсиа: И как же нам поступить теперь?

Энди: Проблема заключается в том, что мы связываем роль напрямую с организацией. В отличие от таблицы people, таблица organization, на самом деле, представляет абстрактные сущности, чьи конкретные проявления определяются их адресами или «географическими положениями». Различные географические расположения представляются в модели данных с помощью соединяющей таблицы между организациями и их адресами. (Обратите внимание на то, что мы не нуждаемся в этом дополнительном уровне для сотрудников, поскольку не существует такой вещи, как абстрактный сотрудник).

Марсиа: Ага! Теперь я вижу ответ. На самом деле нам необходимо связать должностные обязанности с организациями через их географическое расположение (см. рис. 8).

Энди: Очень близко, но это еще не все. На самом деле, здесь существует другая связь «многие ко многим». Один сотрудник может иметь много адресов в заданной географической области, но также есть множество других людей, имеющих те же адреса в этой области.

Марсиа: О да, каждый новый сотрудник, добавляемый нами, требует своей собственной записи в таблице locale, пусть даже они дублируют уже существующие записи. Это очень быстро может привести к значительному увеличению таблицы.

Энди: Именно так. Нам необходимо разбить эту связь «многие ко многим» с помощью другой соединяющей таблицы, как это показано на рис. 9.

Марсиа: Думаю, мы достигли желаемого – теперь мы можем моделировать любые связи. Кроме того, эта модель эстетически приятна – не имеет пересекающихся линий и очень симметрична.

Энди: Да, я согласен с тобой, хотя, на самом деле, это не являлось целями нашей логической разработки. Дело в том, что если имеешь дело с данными, являющимися симметричными (как в данном случае), и результирующая модель выглядит симметрично и не заполнена пересекающимися связями, тогда, вероятно, ты на правильном пути.

Марсиа: Да, но серьезным испытанием является то, насколько хорошо она может быть приспособлена к нашему сценарию.

Энди: Похоже, что модель работает достаточно хорошо. Таблица 1 показывает, как данные могут сохра-

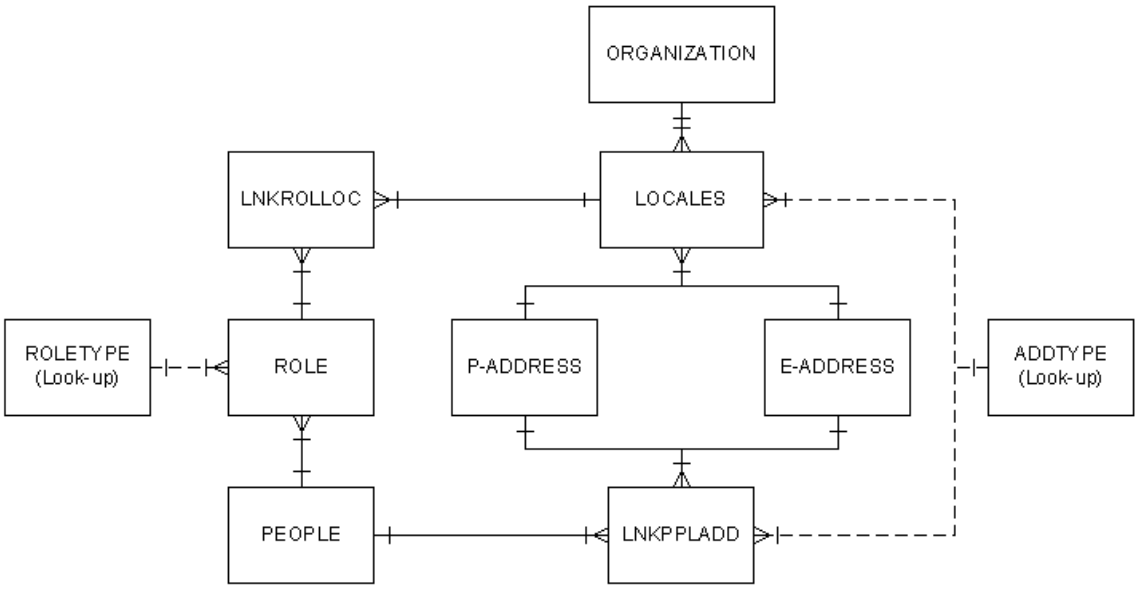

Рис. 9. Окончательная логическая модель данных.

Таблица 1. Проверка модели данных.

Сотрудник	Обязанности	Организация и расположение	PAddress	EAddress
Анита Дринк	Служащая	"Dewey Cheetem and Howe" филиал в Акроне	12333 Cross Street Акрон, Огайо, 44308	Тел.: 330 123 4566 Сотовый: 330 849 4922 Эл. адрес: adrink@aol.com
Лес Сеймур-Кэш	Владелец	"Dewey Cheetem and Howe" Головной офис	21499 Letsby Avenue Кливленд, Огайо, 44118	Тел.: 216 691 8433 Сотовый: 216 849 1122 Эл. адрес: boss@dch.com
		"Dewey Cheetem and Howe" Головной офис	Standard Building Ontario Street Кливленд, Огайо, 44101	Тел.: 216 381 1200 факс: 216 381 5522 URL: www.dch.com
		"Dewey Cheetem and Howe" Филиал в Акроне	85122 Main Street Акрон, Огайо, 44309	Тел.: 330 432 8524 факс: 330 432 8462

няться в главной таблице. Оставим в качестве упражнения для наших читателей заполнение необходимых связывающих таблиц.

Марсиа: Но я думала, что Анита является судебным секретарем, а ты показал ее как сотрудницу.

Энди: Да, но «секретарь в суде» является названием должности, и это можно сохранить в таблице role явно или в качестве соответствия в списке стандартных должностей. Функцией этого типа является описание связи между сущностями. В данном случае Анита просто сотрудница фирмы. Кстати, хотя мы, фактически, не пытались определить такие поля в этих таблицах, ты, возможно, захочешь иметь подобное описательное поле и в таблице Locales, и в таблице Address (кроме того, если типом является «телефонный номер», здесь могут быть много различных «подтипов»). Но опять все это вопросы реализации. Это не меняет логическую модель.

Марсиа: Уф! Не удивительно, что я заработала головную боль, пытаясь представить все это, ведь модель получилось достаточно сложной, не так ли?

Энди: На самом деле – нет. Как мы уже говорили в самом начале, все это просто вопрос видения вещей в правильной перспективе: вы отвлекаетесь от деталей и определяете общие требования. После того, как это сделано, разработка продвигается сама собой.

Марсиа: Конечно, на нас по-прежнему остается решение по конкретной структуре таблиц, но теперь, когда у нас есть законченная логическая модель, это достаточно просто. Осталось только решить, где определенные элементы данных будут располагаться.

❑❑❑

Элемент интерфейса ComboTree

Предраг Боснич (Predrag Bosnic)

WIN 7.0 DOWNLOAD

На этот раз Предраг Боснич объединяет элементы интерфейса Visual FoxPro и управляющие элементы ActiveX для создания очень мощного и интересного элемента.

Раскрывающийся список (combo box) используется столь часто, что впору говорить, что форма, не содержащая раскрывающегося списка, уже не является таковой. Обычно раскрывающиеся списки применяются, если пользователям необходимо выбрать некоторое значение из предопределенного набора вариантов и разработчик не хочет использовать список (list box) ввиду того, что он занимает намного больше пространства формы. Это вполне приемлемо, но что делать, если некоторые элементы этого списка имеют свои подэлементы? В этом случае стандартный раскрывающийся список помочь не может. Возможное решение заключается в использовании второго раскрывающегося списка для отображения элементов, связанных с элементом, выбранным из первого раскрывающегося списка. Примерами могут служить такие структуры, как страна/города или таблица/поля и т. д. Впрочем, вы можете решить, что раскрывающийся список не подходит для отображения структурированных элементов.

За последние несколько лет, благодаря тому, что практически все современные приложения имеют интерфейс в «стиле Explorer», TreeView стал общепринятым элементом. В отличие от раскрывающегося списка, TreeView может отображать структурированные элементы и, казалось бы, идеально подходит для решения нашей проблемы. Единственным недостатком TreeView является его размер – этот элемент не может быть закрыт или открыт и занимает значительное пространство. Поскольку довольно часто пространство формы ограничено, элемент, сочетающий в себе характеристики раскрывающегося списка и TreeView, оказался бы идеальным решением. Существует ли нечто подобное на рынке? Я не смог найти похожего элемента OCX, но несколько фирм-разработчиков используют элементы этого типа в своих программных продуктах. Несомненно, Microsoft является одной из таких компаний, и в Windows есть диалоговое окно Open с элементом интерфейса этого типа. Вы могли обратить внимание, что приложение Analysis Manager компании Microsoft также использует этот элемент (см. рис. 1 и 2).

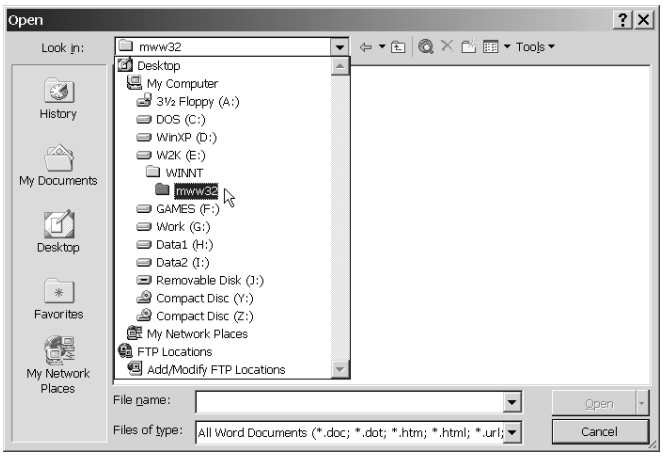

Рис. 1. Диалоговое окно Open использует элемент combo tree.

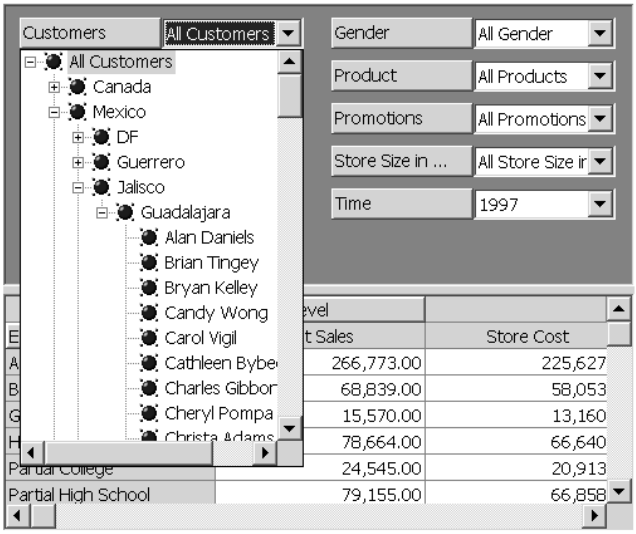

Рис. 2. Analysis Manager (Cube Editor) также использует элемент combo tree.

Анализ

По определению раскрывающееся дерево (combo tree) является модальным элементом. Все мы знаем о модальных формах, но что этот термин означает на самом деле? Давайте разберемся. Стандартный

раскрывающийся список ведет себя следующим образом. Если вы открываете раскрывающийся список, то либо выбираете в нем некоторый элемент, либо отменяете действие. Посмотрим на диалоговое окно Open в Windows. Когда вы щелкаете мышью, список открывается и показывает некоторый вариант TreeView. Щелчок любого элемента закрывает список (то же произойдет, если вы щелкните вне TreeView). Подобное поведение можно наблюдать и в Cube Editor (см. рис. 2), но на этот раз вы можете раскрывать и сворачивать его узлы. Если щелкнуть раскрывающийся список еще раз, дерево закроется, и выбранный элемент появится в поле ввода раскрывающегося списка. Вдобавок к этому посмотрите на рис. 3 – сводная таблица Microsoft Excel также использует раскрывающееся дерево. На этот раз данный элемент содержит TreeView и две командные кнопки. Можно щелкать где угодно в TreeView, раскрывать и сворачивать узлы, выбрать узел, а затем, по своему усмотрению, щелкнуть кнопку OK или Cancel. Тем самым вы сделаете выбор и этот элемент закроется.

Подобная версия с двумя кнопками выглядит как наиболее подходящее решение. Она дает возможность полностью раскрывать и сворачивать TreeView, а наличие кнопок обеспечивает простоту использования.

Решение

Первая идея является почти автоматической реакцией на эту проблему. Щелчок раскрывающегося списка не должен приводить к его раскрытию, напротив, он будет выводить форму с TreeView и двумя кнопками, расположенными точно под раскрывающимся списком (см. рис. 4).

Вторая идея основывается на использовании составного элемента. Контейнер содержит раскрывающийся список TreeView, а также две кнопки. Когда этот элемент закрыт, его высота уменьшается, чтобы отображать только раскрывающийся список. При щелчке на раскрывающемся списке высота контейнера изменится до его полного размера, показывая TreeView и кнопки. И, разумеется, после нажатия кнопки OK или Cancel высота этого элемента опять уменьшается до размера раскрывающегося списка.

Обычно я говорю, что каждая проблема имеет, по крайней мере, два решения, но на этот раз у меня действительно два совершенно правомерных решения и вопрос заключается в том, какое из них предпочтительней? На первый взгляд кажется, что ни одно из них не скрывает существенных недостатков. Однако я выбрал решение, использующее форму, поскольку решение с контейнером предполагает слу-

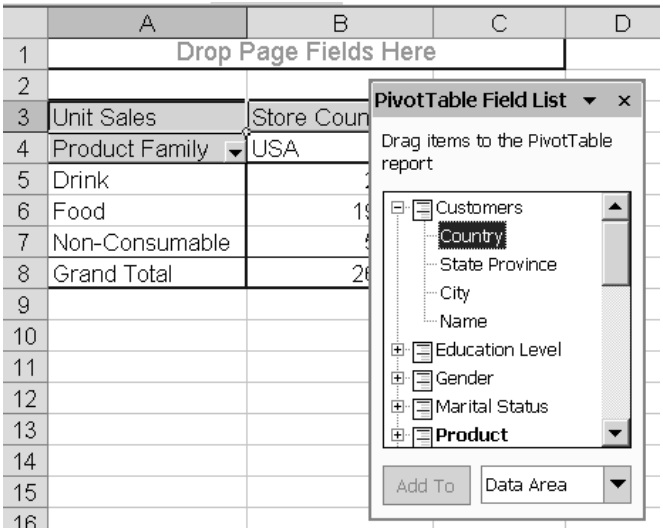

Рис. 3. Сводная таблица Excel использует элемент combo tree.

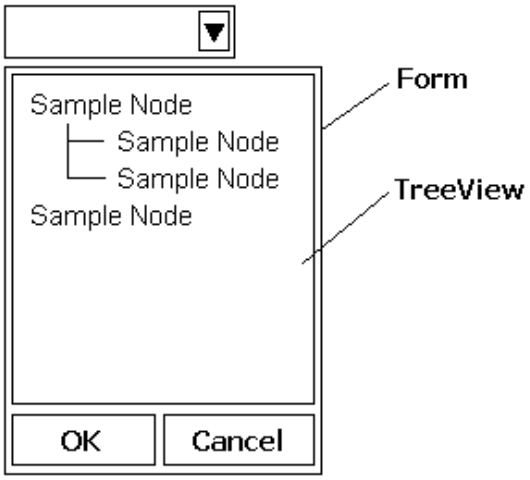

Рис. 4. Это элемент ComboTree, который я хочу создать.

чай, вызывающий сбой. Представьте следующую ситуацию: если расположить раскрывающийся список в нижней части формы, то при его открытии (щелчком мыши) вы не сможете увидеть раскрывающийся список полностью.

Моим первым шагом является создание библиотеки класса wbCboTree и некоторого класса, основанного на элементе combo box. Я назову его просто: “Cbo”. Конечно, наиболее важной частью является прерывание события Open объекта combo box. Оно срабатывает при щелчке раскрывающегося списка, и событие MouseDown отвечает за действие “open”.

Именно здесь я могу разместить мой код и отменить стандартное поведение с помощью ключевого слова NODEFAULT. Метод MouseDown содержит следующий код:

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord

 LOCAL jnLeft as Integer, jnTop as Integer
 jnLeft = 0
 jnTop  = 0
 = this.ABS_pos(this,@jnLeft,@jnTop)

 this.oCboForm = CreateObject('CboForm',this)
 this.oCboForm.left = jnLeft
 *
 IF (jnTop + this.Height+1+this.oCboForm.height);
    > Sysmetric(2)
   this.oCboForm.top=jnTop-this.oCboForm.height-1
 ELSE
   this.oCboForm.top  = jnTop + this.Height + 1
 ENDIF
 **********************************
 this.Fill_Tree(this.oCboForm.Tree)
 **********************************
 this.cStatus = 'OPEN'
 this.oCboForm.show()
 NODEFAULT
```

Прежде всего необходимо вычислить положение формы, которую я хочу показать. Для этого я использую метод ABS_POS. Посмотрите листинг 1, содержащий код метода ABS_POS. Этот метод будет вычислять левое и правое положение раскрывающегося списка по отношению к физическому экрану. После чего код создает форму и вычисляет ее итоговое положение. Перед тем, как форма будет показана, этот код вызывает метод Fill_Tree, передавая ссылку на элемент TreeView. А уж заполнение TreeView является обязанностью пользователя.

Листинг 1. Метод ABS_POS.

```
LPARAMETERS toObject, tnLeft, tnTop
 *
 * toObject - ссылка на объект
 * tnLeft - значение 'Left' для объекта
 * tnTop - значение 'Top' для объекта
 *
 * Возвращает: Значения Left и Top для объекта в
 *          tnLeft и tnTop
 *-----------------------------------------------

 IF type('toObject') <> 'O' OR ISNULL(toObject)
    RETURN ''
 ENDIF

 LOCAL jcH as string, jn1 as integer, ;
      jcXobj as string, joForm as object

 joForm = _Screen.Activeform

 * Sys(1272) возвратит:
 * 'form1.pageframe1.page4.command2'
 *
 jcH = SYS(1272,toObject)
 jcXobj = ''
 jnObjects = GETWORDCOUNT(jcH,'.')
 FOR jn1=1 TO jnObjects
   jcXobj = IIF(EMPTY(jcXobj),jcXobj+;
     GETWORDNUM(jcH,jn1,'.'), jcXobj+'.'+;
     GETWORDNUM(jcH,jn1,'.'))
   IF UPPER(LEFT(jcXobj,4)) = 'FORM' AND jn1 = 1
     jcXobj = 'joForm'
   ENDIF
   jcBaseClass = UPPER(&jcXobj..BaseClass)
   DO case
     CASE jcBaseClass = 'FORM'
       tnLeft=tnLeft+&jcXobj..Left+SYSMETRIC(3)
       IF &jcXobj..TitleBar = 1
         tnTop=tnTop+&jcXobj..Top+SYSMETRIC(4)+;
         SYSMETRIC(9)
       ELSE
         tnTop=tnTop+&jcXobj..Top+SYSMETRIC(4)
       ENDIF
       CASE jcBaseClass = 'PAGEFRAME'
         tnLeft = tnLeft + &jcXobj..Left
         * && высота вкладок PageFrame равна 26
         IF &jcXobj..Tabs = .t.
           tnTop  = tnTop  + &jcXobj..Top + 26
         ELSE
           tnTop  = tnTop  + &jcXobj..Top
         ENDIF

       CASE jcBaseClass = 'PAGE'
         *

       CASE jcBaseClass = 'CONTAINER'
         tnLeft = tnLeft + &jcXobj..Left
         tnTop  = tnTop  + &jcXobj..Top

       OTHERWISE
         exit
   ENDCASE
 NEXT
 tnLeft = tnLeft + toObject.Left
 tnTop  = tnTop  + toObject.Top
 RETURN
```

Этот класс имеет три пользовательских свойства:

- cStatus — статус (OPEN/CLOSE);
- nStyle — внешний вид, где 0=3D, 1=2D;
- oCboForm — ссылка на объект oCboForm.

Второй класс данной библиотеки классов – это класс command button, называемый cTopOK. Я буду использовать его для двух кнопок – OK и Cancel. Код для этих кнопок не слишком велик, но будет полезным иметь некоторый класс на тот случай, если в будущем по какой-либо причине захочется улучшить любую из них. Метод Click содержит следующий код:

```
LOCAL joPapa as Object

 joPapa = this.Parent
 with joPapa
   if this.caption = 'OK'
     IF isNull(.Tree.SelectedItem)
       return
     endif
     .oCboPapa.RowSource = .Tree.SelectedItem.text
     .oCboPapa.value = .Tree.SelectedItem.text
     .oCboPapa.refresh()
     .refresh()
     .oCboPapa.cStatus = 'CLOSE'
   ENDIF
   thisform.Release()
 endwith
```

Как видите, этот код присваивает некоторое значение раскрывающемуся списку и устанавливает свойство cStatus.

Третий класс, который я хочу создать, – это класс form, названный мною CboForm. На эту форму я добавляю TreeView, а внизу помещаю две

кнопки, основанные на классе cTopOK, только что созданном мною (см. рис. 5).

Установим следующие свойства:

```
AlwaysOnTop = .T.
BorderStyle = 1 - одиночная линия
ControlBox = .F.
TitleBar = 0 - Off
```

У этой формы есть пользовательское свойство oCboPapa, являющееся ссылкой на родительский раскрывающийся список. Метод Init формы принимает ссылку на родительский раскрывающийся список и сохраняет ее в свойстве oCboPapa.

```
LPARAMETERS toCboPapa
this.oCboPapa = toCboPapa
```

Метод LostFocus содержит только одну строку кода:

```
ThisForm.Release()
```

Чтобы обеспечить поведение стандартного раскрывающегося списка, при потере фокуса моя форма будет уничтожаться.

Пример использования

Использовать CboTree очень просто. Как и для других визуальных элементов интерфейса, добавьте его на форму и подберите размер. Используя метод Fill_Tree, вы можете написать код, заполняющий TreeView (см. рис. 6).

Усовершенствования

Кроме оптимизации кода, на ум приходят следующие возможные улучшения:

- Обработчик изменения размеров – было бы хорошо иметь возможность изменять размеры CboForm, если она отображает объект TreeView большого размера.
- Было бы хорошо сохранять статус элемента TreeView. Это значит, что при следующем открытии раскрывающегося списка, TreeView будет открываться в том же состоянии, в котором был оставлен в предыдущий раз.
- На этапе разработки я не могу менять ширину CboForm (но это можно сделать модификацией класса!). Создание свойства для поддержки требуемой ширины может решить эту проблему.

Заключение

На этот раз я объединил элементы интерфейса Visual FoxPro и управляющего элемента ActiveX для создания очень мощного и интересного элемента. Я надеюсь, он найдет применение во многих ваших формах. Я действительно считаю, что все усовершенствования, упомянутые мною выше, являются желательными. Пожалуйста, загрузите исходный код и используйте его, приспосабливая к собственным нуждам. До следующей встречи, наилучших пожеланий.

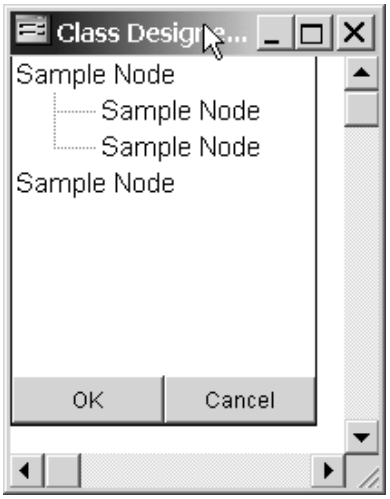

Рис. 5. Класс CboForm в окне Class Designer.

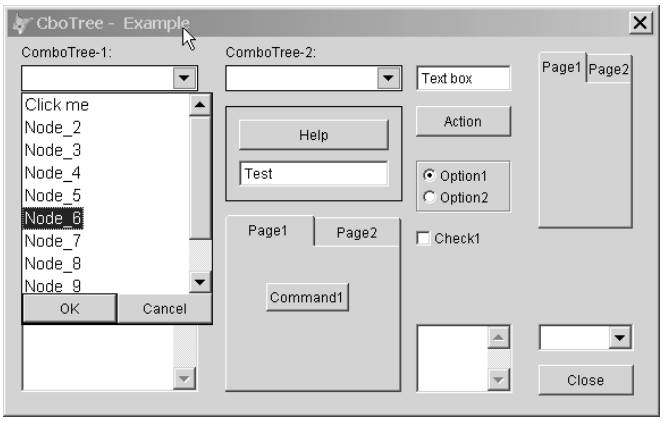

Рис. 6. Элемент CboForm ComboTree на форме.

Предраг Боснич (Predrag Bosnic) начал свое путешествие в области ИТ в 1979, используя компьютер UNIVAC 1100, язык программирования Fortran и Mapper. В течение последующих 20 лет он побывал в мире ПК, dBase, Clipper и Fox. На днях он обнаружил себя в Лондоне, работающим в фирме с названием Westwood Forster Ltd, где он является старшим разработчиком (senior developer) и вытворяет разные необыкновенные вещи с помощью Visual FoxPro и SQL Server. mbosnic@westwood-forster.co.uk.

❑❑❑

Стань плодотворнее с VFP, часть 2

Ричард Шуммер (Richard Schummer)

WIN 8.0 7.0 6.0

Visual FoxPro является исключительно эффективным инструментальным средством разработки. В этом месяце Рик Шуммер продолжает свою серию статей публикацией разнообразных советов и практических приемов, позволяющих VFP-разработчику добиться большей эффективности в работе, применяя утилиту Class Browser и новый инструмент Task Pane.

Утилита Class Browser — это мощный инструмент, который помогает разработчикам, использующим Visual FoxPro, управлять классами и библиотеками классов. Утилита Task Pane — это новшество, появившееся в версии VFP 8 и обеспечивающее массу способов повысить производительность труда. Я не буду рассматривать «интимные» подробности того, как пользоваться каким-либо из этих инструментов, но расскажу о том, как «выжать» из них дополнительные возможности, позволяющие сделать ваш труд более эффективным, что в итоге может сэкономить значительный объем времени, если вы постоянно работаете с упомянутыми утилитами.

Class Browser

Утилита Class Browser — это инструментальное средство, предназначенное для повышения производительности труда, которое позволяет разработчику создавать, модифицировать и удалять классы, а также образовывать подклассы на базе уже имеющихся классов. С помощью этой утилиты вы можете переопределить суперкласс класса, изменить пиктограмму, которая выдается на экран в окнах диспетчера Project Manager и утилиты Class Browser, а также выполнять такие задачи, как копирование или перемещение класса из одной библиотеки классов в другую. Досконально изучение возможностей, которыми обладает утилита Class Browser, в результате может обеспечить существенные преимущества разработчику, использующему Visual FoxPro.

Настройка файла, открываемого по умолчанию при запуске Class Browser

Случалось ли вам переживать один из таких периодов, когда вы день-деньской напролет занимаетесь рефакторингом одного-единственного класса или формируете набор классов для одной и той же библиотеки классов? Вы снова и снова запускаете утилиту Class Browser и открываете в ней все одну и ту же библиотеку классов. Не возникало ли у вас мысли о том, что неплохо было бы иметь возможность выполнить некоторую настройку так, чтобы когда вы запускаете утилиту Class Browser, она при этом открывала бы указанную библиотеку классов? Что ж, вы можете добиться такого эффекта, если отдадите соответствующее распоряжение утилите Class Browser.

Во-первых, запустите Class Browser и откройте ту библиотеку классов, которую вам необходимо открывать «по умолчанию» при запуске этой утилиты. В командном окне наберите строку:

```
_oBrowser.SetDefaultFile()
```

Каждый раз, когда будет запускаться утилита Class Browser, она будет открывать ту же самую библиотеку классов. Внутренний механизм работает таким образом, что утилита Class Browser обновляет файл Browser.dbf, меняя на .T. значение, хранящееся в столбце Default той записи, которая относится к данной библиотеке классов (см. значение, хранящееся в столбце Name, чтобы найти необходимую библиотеку классов и полный путь доступа к ней). Указанный способ прекрасен в качестве временного средства, но было бы досадно, если бы вы не могли вернуть эту настройку обратно, в исходное состояние, когда не открыта никакая библиотека классов. Не волнуйтесь, если используемую по умолчанию библиотеку классов необходимо «вычистить», выполните следующее предложение после запуска утилиты Class Browser:

```
_oBrowser.ResetDefaultFile()
```

Теперь утилита Class Browser запускается, но библиотека классов при этом не открывается, как это и предусмотрено в штатном режиме работы.

Открытие конкретного класса при запуске утилиты Class Browser

Запуская утилиту Class Browser из программы, вы можете открыть указанный класс и даже заранее выбрать такой класс, а также свойство или метод выбранного класса.

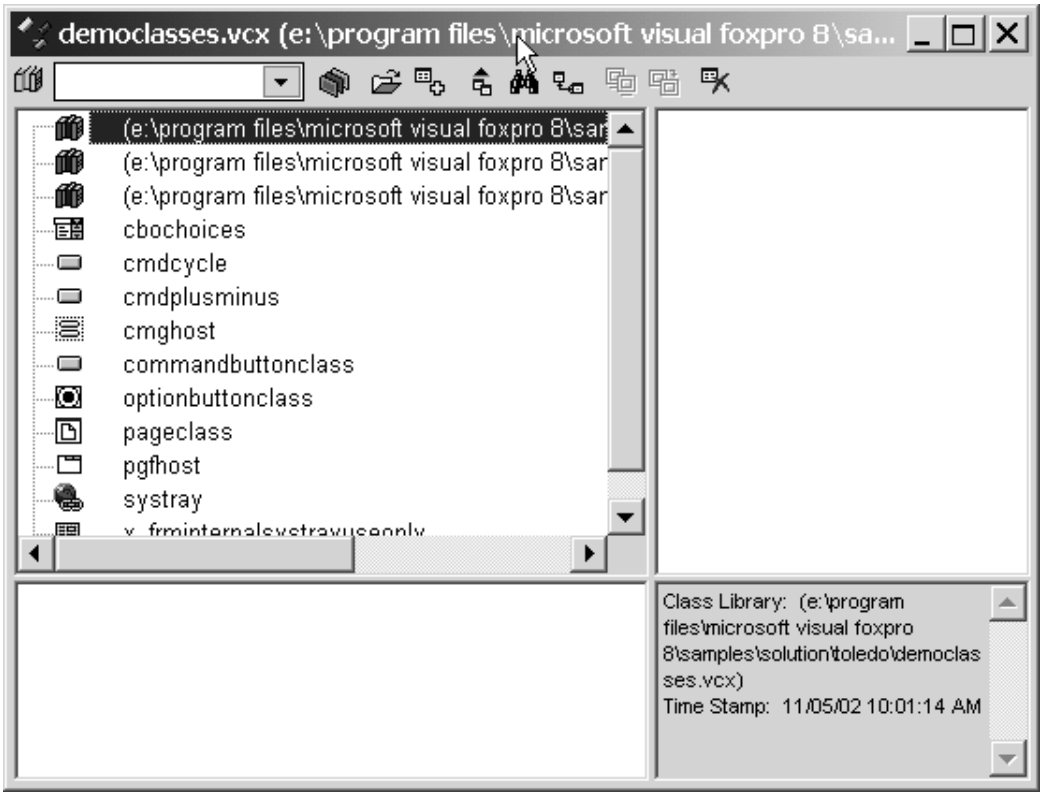

Рис. 1. Утилиту Class Browser можно запустить, открыв при этом несколько классов одновременно (это можно сделать программно, а также с помощью интерфейса данного инструментального средства).

Первый параметр, который передается утилите Class Browser, это библиотека классов. Чтобы библиотека классов успешно открылась, она должна находиться в том каталоге, куда была инсталлирована Visual FoxPro, или должен быть указан полный путь доступа к этой библиотеке. Второй параметр — это имя класса, за которым следует точка, а затем имя соответствующего свойства или метода. Если вы передаете в качестве второго параметра одно только имя класса, библиотека классов будет открыта, но никакой конкретный класс выбран не будет. Чтобы данный класс был «подсвечен» в левой панели (TreeView) на экране, необходимо передать имя свойства или метода, «приписанное» к имени класса. Вот пример обращения к утилите Class Browser, который при ее запуске открывает класс phkDevelopment из библиотеки классов CPhkBase2 и свойство lCopyAppToTestDirectory этого класса.

```
DO (_browser) WITH "CPhkBase2", ;
     "phkDevelopment.lCopyAppToTestDirectory"
```

Такая возможность может оказаться полезной, если вы пытаетесь сэкономить время на постоянно повторяющейся операции по открытию одной и той же библиотеки классов в ходе одного сеанса разработки. Еще одно «сокращение» процедуры запуска утилиты Class Browser сводится к тому, что надо щелкнуть правой клавишей мыши по командной кнопке Open и выбрать библиотеку классов в открывшемся списке тех библиотек классов, которые использовались последними.

При запуске утилиты Class Browser можно также открыть несколько библиотек классов одновременно. Это достигается путем передачи в качестве первого параметра нескольких имен библиотек классов, разделенных запятыми:

```
DO (_browser)
  WITH "G2Metadata,;
       g2Utils, ;
       G2CustomControls"
```

На рис. 1 показано, как будет выглядеть окно утилиты Class Browser при исполнении вышеприведенной команды в окне Command.

Переименование методов и свойств без открытия класса

Вы можете переименовать методы и свойства класса, щелкнув правой кнопкой мыши по имени этого свойства или метода в правой панели утилиты Class Browser и выбрав в меню быстрого вызова пункт Rename... Эти действия приводят к открытию диалогового окна, которое позволяет вам переименовать выбранное свойство или метод. Пожалуйста, не забудьте, что при изменении имени метода или свойства ссылки на эти метод или свойство в программном коде не меняются как по волшебству. Вам придется вручную найти

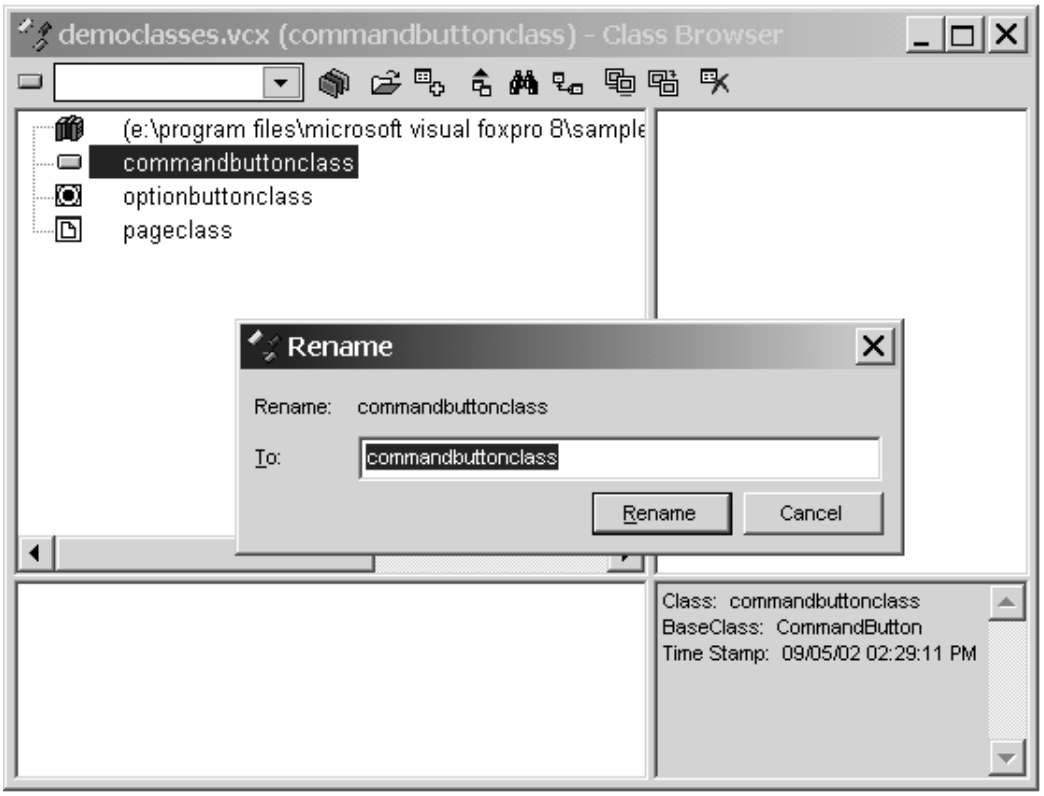

Рис. 2. Утилита Class Browser обеспечивает возможность переименовать метод и свойство, не открывая соответствующий класс.

и заменить все измененные ссылки (есть единственный случай, который обсуждается далее в этой статье, когда в ручной замене ссылок нет необходимости). Кроме того, вы можете использовать новый инструмент Code Reference, который входит в состав версии VFP 8, чтобы найти прежнее имя свойства/метода и заменить его на новое (см. рис. 2).

Это намного быстрее, чем открывать класс, добираться до диалогового окна Edit Property/Method, искать по списку необходимые свойство или метод, а затем изменять его имя.

Безопасное изменение имени класса без разрушения ссылок на подклассы

Похоже, что так всегда и бывает. Вы приступаете к разработке этого отличного нового класса, и вот уже его определение находится у вас в библиотеке классов, что в 3:00 часа утра представляется вполне логичным. Вы продолжаете работать над этим классом, «вылизываете» его и после серьезного тестирования применяете его в нескольких формах и других классах проекта. На следующее утро, после необходимого минимума сна и самой первой банки колы, вы понимаете, что наиболее подходящим местом для вашего замечательного класса является совсем другая библиотека классов. Перемещение или переименование класса или библиотеки классов может повлечь за собой возникновение всевозможных проблем для разработчиков, использующих Visual FoxPro. Если вы переименовываете или перемещаете класс, все другие классы и формы, которые используют этот класс, будут «приставать» к вам с просьбой указать местонахождение данного класса всякий раз, как вы будете открывать их в соответствующих конструкторах. Это может обернуться сущим наказанием, особенно в том случае, если подвергнутые переименованию классы являются составной частью каркаса (framework). Итак, как же нам избежать подобной напасти?

Если существуют подклассы или экземпляры объекта измененного класса, входящая в состав Visual FoxPro утилита Class Browser автоматически выполнит «настройку» имен и/или «адресов» для всех файлов библиотек классов и форм. Вы также можете открыть файлы проекта Visual FoxPro, которые будут загружаться во всех файлах библиотек классов и форм для данного проекта. Следовательно, если вы переименовываете какой-то класс, рекомендуется, чтобы в утилите Class Browser были загружены все файлы библиотек классов и форм, которые используют изменяемый класс или являются его подклассами, дабы все ссылки на изменяемый класс были обновлены автоматически.

Единственное, что не будет обновлено автоматически, — это программный код, в котором имеются ссылки на изменяемый класс, например:

```
thisform.oRegistry = NEWOBJECT("cusRegistry", ;
    "CFramework")
this.oBusiness     = CREATEOBJECT("cusInvoiceBO")
```

Утилита Class Browser достаточно разумна лишь для того, чтобы внести исправления в наследственную иерархию объектов, но не в те ссылки на классы, которые имеются в программном коде. Это нечто такое, что вам придется проделать вручную. Но это намного лучше, чем разбираться с программным кодом и объектами, которые оказались «сломанными» в результате изменения имени некоторого класса. Если изменяемый класс определен в общей библиотеке и он используется в нескольких проектах, вы должны позаботиться о том, чтобы были открыты все такие проекты или же подвергнуть пересмотру свое решение о переименовании данного класса.

Task Pane

Утилита Task Pane — это новое инструментальное средство, которое входит в состав версии VFP 8. Task Pane — это интерфейс, в основе которого лежит браузер, работающий «внутри» VFP-формы. Утилита Task Pane представляет собой коллекцию «сокращений» для тех задач, которые важны для большинства VFP-разработчиков. Некоторые панели этой утилиты содержат статичную информацию, тогда как содержание других динамично меняется в соответствии с вашими предпочтениями, указанными для данного инструмента (см. рис. 3).

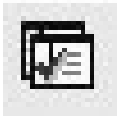

Рис. 3. Пиктограмма Task Pane находится на панели инструментов Standard.

Добавление возможности быстрого доступа к инструментальным средствам разработки

Используемая по умолчанию страница Start утилиты Task Pane имеет несколько разделов. Второй раздел позволяет вам организовать быстрый доступ к инструментальным средствам разработки. Этими инструментальными средствами разработки могут быть утилиты, которые создали вы сами, другие разработчики или такие, которые входят в состав VFP (см. рис. 4).

Можно организовать быстрый доступ к исполняемым модулям VFP (APP или EXE), формам или отчетам. Если это не то, что вы ищете, можете написать сценарий (script) (сюрприз, сюрприз: сценарий — это программный код, написанный на языке Visual FoxPro), чтобы проделать все то, что вы можете сделать средствами Visual FoxPro. Тривиальным примером такого подхода могло бы служить помещение имени файла в буфер обмена Clipboard ОС Windows:

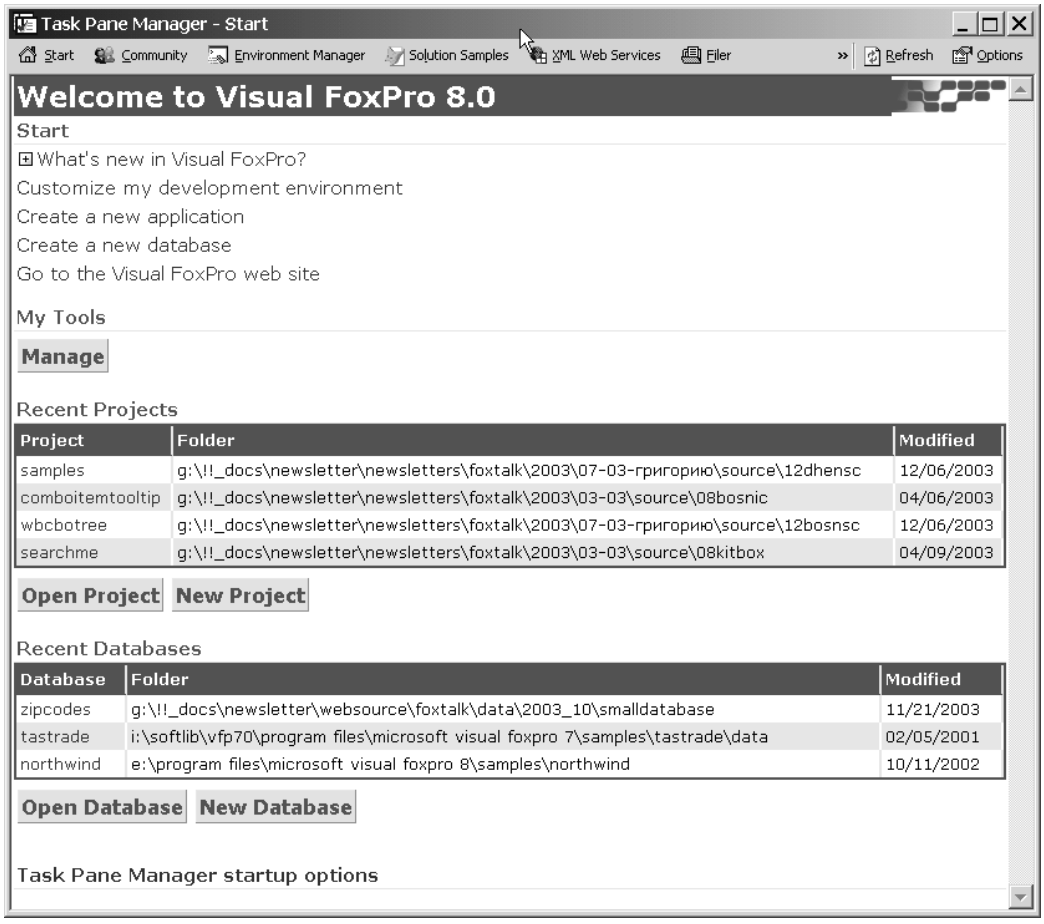

Рис. 4. Входящая в состав Visual FoxPro утилита Task Pane позволяет разработчикам подключаться к инструментальным средствам разработки.

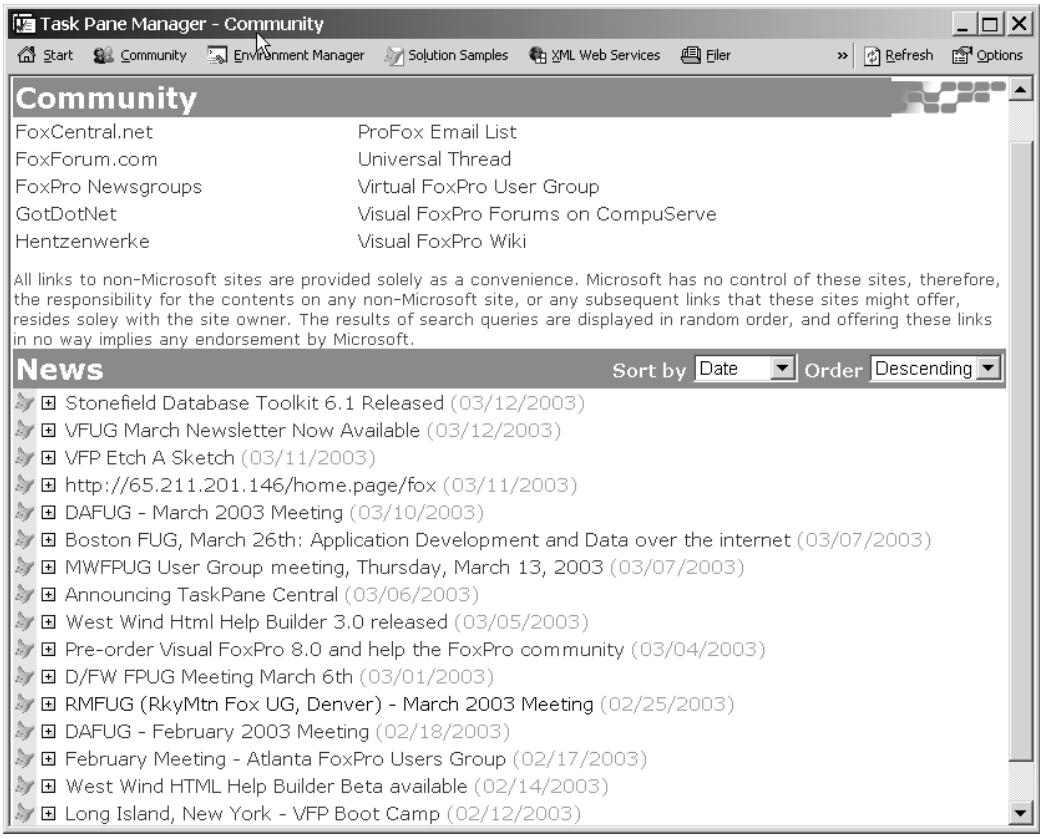

Рис. 5. Входящая в состав Visual FoxPro утилита Task Pane предоставляет ссылки на новостные сообщения с сайта FoxCentral, обновленные элементы сайта Fox Wiki и новости на сайте Universal Thread.

Быстрый доступ к основным web-сайтам с новостями о VFP

В своей деятельности фирма Microsoft и особенно Fox-команда ориентируются, главным образом, на интересы и нужды сообщества разработчиков. Именно поэтому они обеспечивают для вас связь с несколькими популярными Web-сайтами, в центре внимания которых находится VFP: это два основных новостных сайта и сайт форума Fox Wiki. Вы можете конкретизировать с каких сайтов и какой давности новости вы хотите получать (см. рис. 5).

В настоящее время верхняя часть панели раздела Community, в которой перечислены Web-сайты, не конфигурируется. Я обнаружил, что использование именно этой панели обеспечивает ежедневно изрядную экономию времени. Вместо того, чтобы запускать три экземпляра браузера, дабы получить новости с сайтов FoxCentral.net или UniversalThread.com и обновленные темы с сайта Fox Wiki, я должен всего лишь запустить утилиту Task Pane и позволить ей обновить список новостных элементов и тех тем, в которых появилось что-то новое. Если вы щелкните мышью по пиктограмме детализации (знак плюс) рядом с любым из новостных элементов, то на экране появится текст данного выпуска новостей. Нажатие на сам элемент, снабженный гиперссылкой, доставит вас непосредственно на этот web-сайт.

```
_cliptext = GETFILE()
```

Если вам понадобилось обеспечить возможность быстрой смены атрибутов шрифта, используемого для «рабочего стола» в VFP, вы могли бы создать сценарий со следующим программным кодом:

```
lcFontStyle = SPACE(0)
 lcFontStyle = lcFontStyle + ;
               IIF(_screen.FontBold, "B", SPACE(0))
 lcFontStyle = lcFontStyle + ;
               IIF(_screen.FontItalic, "I", SPACE(0))

 lcFontAttrib = GETFONT(_screen.FontName, ;
                        _screen.FontSize, ;
                        lcFontStyle, ;
                        _screen.FontCharSet)

 IF NOT EMPTY(lcFontAttrib)
    lnFontAttrib = ALINES(laFontAttrib, ;
                          lcFontAttrib, .T., ",")
    _screen.FontName    = laFontAttrib[1]
    _screen.FontSize    = VAL(laFontAttrib[2])
    _screen.FontBold    = "B" $ laFontAttrib[3]
    _screen.FontItalic = "I" $ laFontAttrib[3]
    IF lnFontAttrib > 3
       _screen.FontCharSet = VAL(laFontAttrib[4])
    ENDIF
 ENDIF
 RETURN
```

Управление средой окружения VFP

Есть множество настроек среды окружения, которые разработчики, как правило, определяют в программе запуска, исполняемой при загрузке Visual FoxPro. Некоторые разработчики создают сложные надстройки к проекту, которые каждый раз, когда проект от-

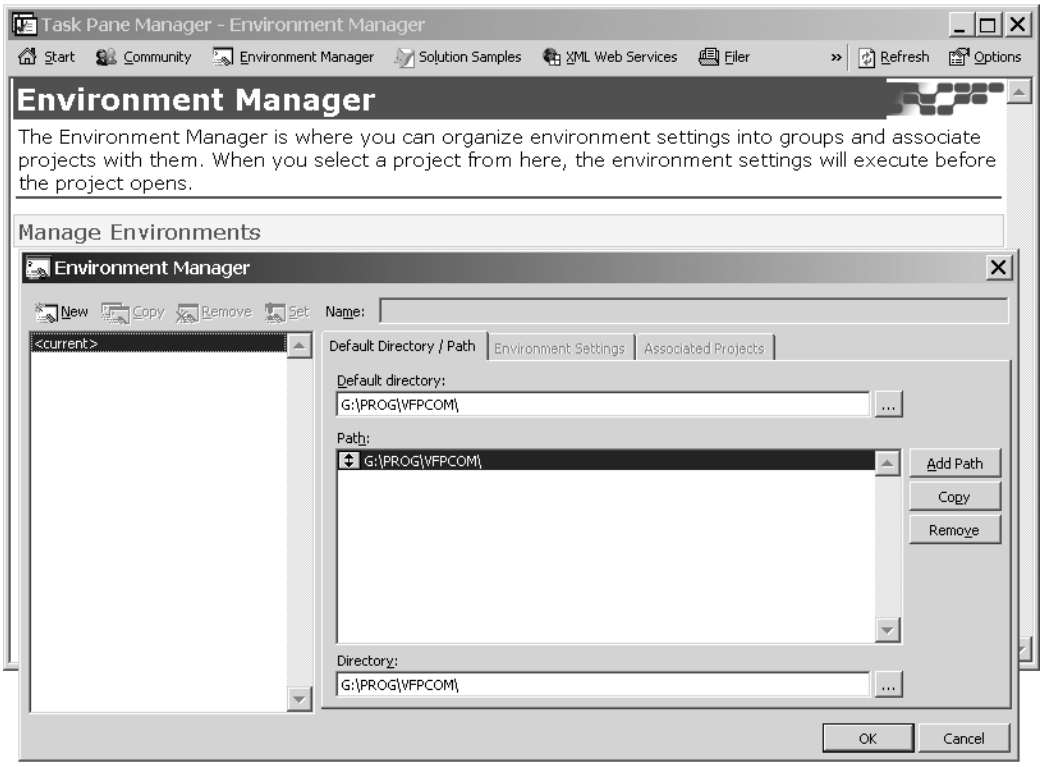

Рис. 6. Входящая в состав Visual FoxPro утилита Task Pane предоставляет возможность иметь несколько конфигураций среды окружения, которые вы можете переключать, работая как с различными проектами, так и в рамках одного и того же проекта.

крывается или активизируется, обеспечивают конфигурацию параметров среды окружения на этапе разработки (например, к таким параметрам относится множество доступных команд SET: Talk, Path, Deleted, Exclusive и так далее). Утилита Task Pane предоставляет возможность определить состояние среды окружения, которое запоминается и может быть загружено, с тем чтобы те разработчики, которые не используют для этого иных способов, могли работать с разными конфигурациями среды окружения (см. рис. 6).

Если вы не используете свой собственный самодельный механизм для контроля за средой окружения VFP, или если вам просто необходимо быстро внести в нее изменения, вам определенно следует присмотреться к данной панели.

Изучение служб Web Services

Начиная с версии 7, Visual FoxPro может работать со службами Web Services. Службы Web Services регистрируются с помощью менеджера IntelliSense Manager. После того, как служба Web Service зарегистрирована, у Visual FoxPro есть несколько способов воспользоваться теми данными, которые были зарегистрированы. Например, VFP может привлечь механизм IntelliSense в качестве «помощника» при написании кода для использования службы Web Service. Утилита Task Pane позволит взглянуть на службу Web Service «изнутри»: сразу же после получения ссылки на данную службу утилита Task Pane продемонстрирует публичные методы службы и предоставит примеры использования этих методов в виде программного кода (см. рис. 7).

Таким образом, вы сэкономите бесчисленные часы, затраченные на поиск утерянной документации о какой-либо службе Web Service или на постоянно повторяющееся рысканье по Web-просторам с целью добыть эту самую документацию.

Заключение

В этом месяце я дал несколько советов относительного того, как повысить производительность труда с помощью утилиты Class Browser и нового инструментального средства из состава версии VFP 8 — утилиты Task Pane. В следующем месяце мы будем искать способы повысить эффективность своего труда путем умелого использования таких инструментов, как диспетчер Project Manager, технология IntelliSense и впервые появившаяся в версии VFP 8 утилита Code Reference.

Рик Шуммер (Rick Schummer) является партнером в компании Geeks and Gurus, Inc. После работы он наслаждается написанием инструментов разработки и время от времени пишет статьи для своих любимых периодических изданий по Fox. Рик является членом-основателем и секретарем организации Detroit Area Fox User Group (DAFUG). Кроме того, он регулярный ведущий пользовательских групп в Северной Америке. www.geeksandgurus.com, www.rickschummer.com, raschummer@geeksandgurus.com.

❑❑❑

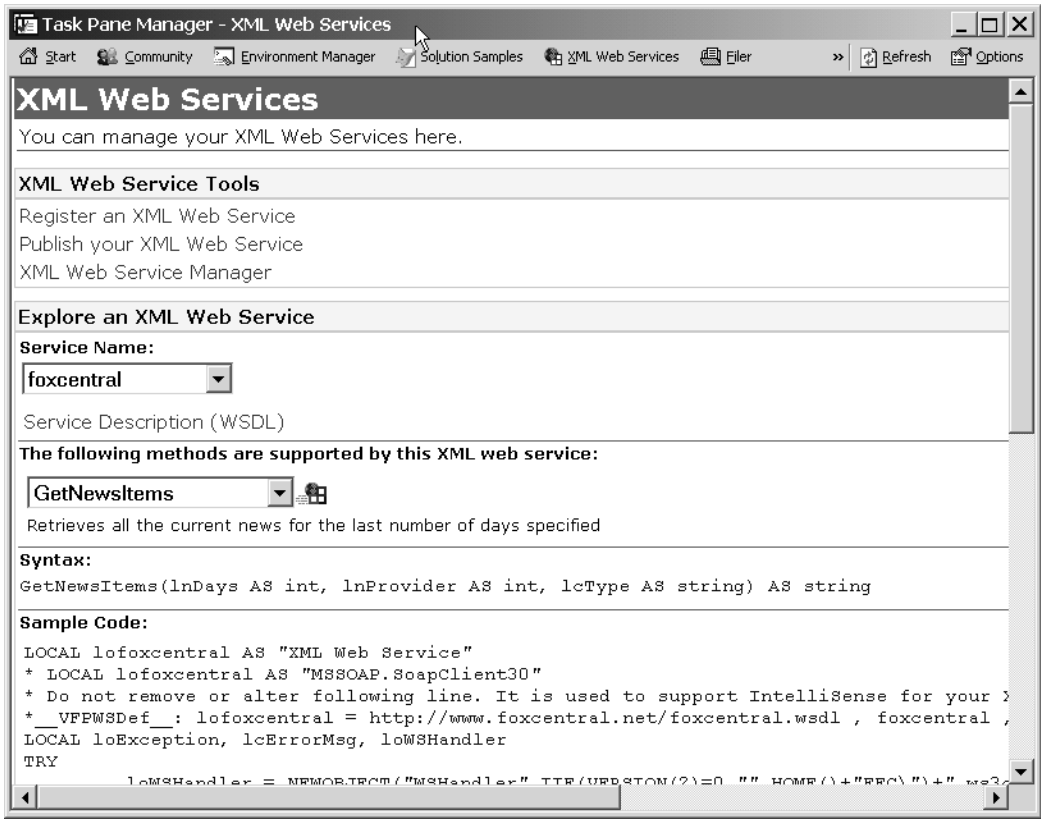

Рис. 7. Входящая в состав Visual FoxPro утилита Task Pane предоставляет документацию о службе Web Service.

FoxTalk *русское издание*

Печатается ежемесячно

Учредитель и издатель:

ООО Эдэль.

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

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

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

Журнал зарегистрирован комитетом
Российской Федерации по печати.

Регистрационное свидетельство
№ 015520 от 17.12.1996

Подписано в печать 05/12/03. Формат 60x90 1/8. Тираж 280 экз.