FoxTalk

Июль 2003
№ 7 (73)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

VFP 8: новая версия для ваших заказчиков

Мартин Сальяс (Martin Salias)

Когда появилась версия Visual FoxPro 7.0, реализованные в ней новые возможности никого не оставили равнодушным. В самом деле, эта версия воспринималась как «мечта разработчиков». Она дала нам технологию Intellisense, окно Document View, привязку окон — Dockable Windows, сохранение информации обо всех выполняемых в командном окне действиях — Persistent Command Window, и много других хороших вещей, которые, несомненно, обеспечили нам более высокую производительность труда в нашей повседневной работе.

Однако, когда мы, в конце концов, приступили к поставке первых приложений, целиком и полностью созданных в версии VFP 7, даже несмотря на то, что в них использовались COM-технология или службы Web Services, в целом, эти приложения не произвели на наших заказчиков особенно сильного впечатления.

Да, у нас была возможность добавить в меню пиктограммы и обеспечить какие-то эффекты применительно к командным кнопкам и другим элементам управления, но по-настоящему новых функциональных возможностей оказалось немного.

Список ваших желаний — список их желаний

Никто не может отрицать, что работая над версией Visual FoxPro 8, группа разработчиков из фирмы Microsoft изучает список запросов, подготовленный сообществом пользователей продукта. Почти каждая новая функциональная возможность или внесенное исправление являются следствием какого-либо пункта из этого списка. А то немногое из нововведений, что не найти в списке, по сути являются дополнениями, предоставленными согласно принципу «сверх запроса».

Но на сей раз, уже имея хороший набор инструментов, предназначенных для разработчиков, мы переадресовали команде VFP многие из тех требований, которые выдвигают наши заказчики. И команда выполнила очень многие из этих запросов вдобавок к внушительному массиву «не визуальных» функциональных возможностей и возможностей интегрированной среды разработки.

Давайте рассмотрим краткий перечень того, что вы теперь можете продемонстрировать своим пользователям:

Июль 2003

материал имеет отношение к соответствующей версии

материал имеет отношение к соответствующей платформе

исходные тексты программ можно скачать из Интернета

- Обновленные варианты диалогов Open и Save.
- Новые возможности элемента управления Page Frame.
- Состоящие из нескольких строк подсказки Tooltip и наличествующее почти у всех элементов управления свойство StatusBarText.
- Усовершенствованный элемент управления Grid (с автоматическим определением размеров столбцов и фиксацией их положения, сохранением выделения выбранного элемента при потере фокуса объектом Grid, функционированием в стиле списков ListBox, реальным сокрытием столбцов, пиктограммами в заголовках столбцов, автоматическим размещением по центру ячейки элементов управления CheckBox и другими возможностями).
- Улучшенная обработка текста и графики при их совместном использовании в одном и том же элементе управления.
- Использование гиперссылок в элементах управления TextBox и EditBox.
- Свойство BackColor для элемента управления CommandButton.
- Наконец-то, скрытая строка состояния StatusBar.
- Поддержка интерфейса GDI+ для элементов управления Image (что означает анимированный формат gif, форматы tiff, png, поворот и переворот картинки и так далее).
- Поддержка оформительских схем Windows XP Themes.
- Полноцветные пиктограммы для ваших исполняемых модулей (эту возможность мы ждали со времени появления ОС Windows 95).
- Экспорт 65 535 строк в табличный процессор Excel (без использования автоматизации, разумеется).
- «Не объектно-ориентированные» отчеты. Извините, ребята, придется согласиться вот на что: более гибкое управление полосами и полями с подстройкой размеров (stretching-fields), непосредственное управление драйвером принтера, встроенный трафарет представления номеров страниц «page x of z», возможность последовательного размещения отчетов на одной и той же странице (при печати) и подавление диалога печати (print dialog).
- Масса хороших идей относительно реализации интерфейса, которые можно позаимствовать у дополнений, полученных интегрированной средой разработки IDE (например, панель Task Pane, инструментальный «ящик» Toolbox, инструментальное средство Code References и так далее).

Я сказал «краткий перечень»? Что ж, он действительно «краткий», если вы сравниваете его с полным списком новых возможностей, реализованных в версии Visual FoxPro 8.0.

Давайте посмотрим на то, как работают некоторые из этих вещей.

Обновленные варианты диалогов Open и Save

Это новшество позволяет вашим приложениям, работающим под управлением ОС Windows 2000 или XP, выглядеть должным образом. Диалоги Open и Save обладают теперь всеми стандартными функциональными возможностями, такими как панель Places Bar, различные типы отображений, пиктограммы thumbnails, изменение размеров и так далее. Просто взгляните на этот снимок с экрана:

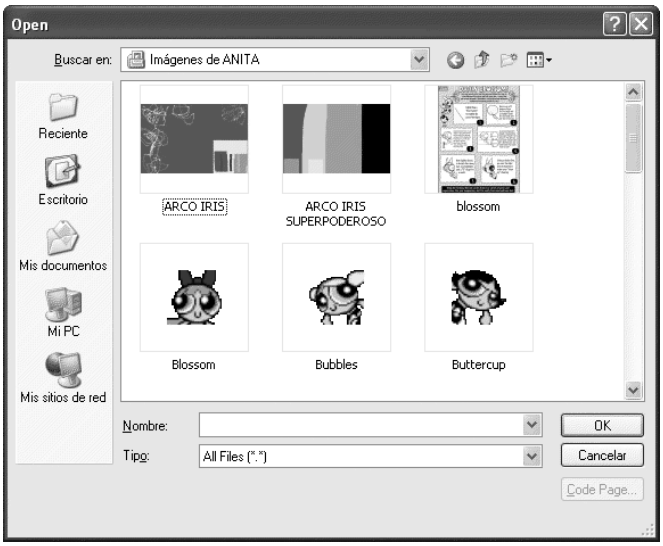

Рис. 1. Новые возможности для элементов управления Page Frame.

Прежде всего, теперь вы можете построить класс на основе объекта Page и поместить его на PageFrame, когда новые страницы добавляются путем изменения значения свойства PageCount. Такая возможность достигается использованием новых свойств MemberClassLibrary и MemberClass и применима также ко всем остальным элементам управления, для которых раньше образование подкласса было затруднено, например, к командным кнопкам CommandButton и объединенным в группы Group переключателям OptionButton, столбцам Column и заголовкам Header в элементах управления Grid.

Тем самым существенно облегчается работа со всеми этими элементами управления.

Но с точки зрения визуального восприятия на первый план выходит новое свойство TabOrientation, которое может получать значения 0, 1, 2 или 3 с тем, чтобы закладки tab размещались сверху (как обычно), внизу, слева или справа, соответственно. Смотрите в следующем разделе пример формы с элементом управления grid, имеющим весьма привлекательный внешний вид.

Усовершенствованные элементы управления grid

Объекты Grid — это одни из самых сложных и полезных интерфейсных элементов управления в VFP, и в своем новом воплощении они получили ряд стандартных функциональных возможностей, которых всегда не хватало прежде. Теперь вы, наконец, можете зафиксировать один или несколько столбцов, расположенных слева так, что при горизонтальной «прокрутке» эти столбцы постоянно будут видны на экране, что во многом аналогично тому, как вы можете поступить с электронной таблицей Excel.

Вы также можете автоматически изменить размеры столбца таким образом, чтобы в нем помещалось самое длинное из представленных значений, всего лишь щелкнув мышью по правой разделительной линии этого столбца. Если ваши пользователи хотят выполнить автоматическую подстройку размеров всех столбцов одновременно, просто научите их тому, что надо дважды щелкнуть мышью по маленькому квадратику, расположенному в верхнем левом углу элемента управления grid.

Одним из наиболее частых запросов, начиная с момента появления версии VFP 3.0, была просьба о возможности сохранять выделение текущей записи в элементе управления grid даже после того, как этот элемент управления терял фокус. Как-никак элементы управления list box имели такую возможность, начиная с версии FoxPro для DOS. Хуже того, различные версии VFP выставляли напоказ набор сбивающих с толку свойств, которые выдавались за средство достижения именно этой цели.

И наконец, вот оно: этот трюк выполняет свойство HighlightStyle. Даже более того, если вы хотите использовать элемент управления grid для выбора строк только целиком, а не отдельных ячеек, вы можете «выключить» свойство AllowCellSelection, чтобы заставить элемент управления grid функционировать аналогично списку list box.

Еще раздражение у разработчиков вызывал тот факт, что указание для свойства столбца visible значения .F. не приводило к исчезновению данного столбца с экрана. Это и заставляло нас изобретать множество извилистых обходных путей. Именно этот эффект достигается сейчас указанными действиями. Вы можете также использовать в заголовках столбцов пиктограммы и разместить элементы управления checkbox в столбце строго по центру, просто задав для нового свойства столбца Centered значение .T..

Рис. 2. Кнопка CommandButton и новое свойство PicturePosition.

На этой форме вы видите кнопку CommandButton, использующую новое свойство PicturePosition, которое может иметь одно из 14-ти возможных значений и указывет положение картинки, и как она выровнена относительно текста.

Вся соль в деталях

Еще один набор «миниатюрных» возможностей, которые в конечном итоге могут произвести на ваших пользователей неизгладимое впечатление, включает динамическое использование в элементах управления TextBox и EditBox гиперссылок на URL-локаторы, обновление меток (label) и фигур (shape) (для принудительной их перерисовки после выполнения некоторых операций) и способ, позволяющий отключить вызывающую раздражение строку состояния status bar, с тем чтобы не обременять своих пользователей знанием того, какая именно таблица открыта, и какая строка в этой таблице является текущей (если в этом нет необходимости, конечно).

Наконец, после долгих споров команда разработчиков добавила свойство BackColor для кнопок. По моему мнению, стандартная кнопка (не графическая) не должна иметь своего собственного цвета, а использовать тот, который определяется настройками Windows. Но поскольку я неоднократно отвечал

на вопросы по этому поводу во многих форумах, должен признать, я рад тому, что создатели версии VFP 8 предусмотрели упомянутое свойство.

Что и в самом деле доставляет мне удовольствие, так это возможность скрыть раздражающие сообщения строки состояния StatusBar о текущей рабочей области. Да! Теперь вы можете избежать этого противного сообщения «Record 5/598»! Вам только надо выполнить следующую команду:

```
SET NOTIFY CURSOR OFF
```

«Лисья шкурка» бывает разной

Поддержка изображений всегда была в VFP «слабым местом» в сравнении с другими средами разработки (в конце концов, предполагалось, что Fox — это инструмент для работы с базами данных). Рассматриваемая версия VFP возлагает решение большей части задач по обработке изображений на интерфейс GDI+. Интерфейс Graphics Device Interface+ представляет собой API, основанный на классах, предназначенных для использования в C/C++, и который эффективно поддерживает — посредством прямого обращения к драйверам — набор графических форматов, включающих среди прочих форматы ani, bmp, cur, dib, emf, gif, ico, jpg, png, tiff, wmf и даже анимированный формат gif. Поэтому теперь вы можете добавить элементарную анимацию к своим элементам управления Image, не прибегая к услугам объекта timer и сложному программированию.

Вы также можете использовать в свойствах picture большинства элементов управления, помимо обычных форматов .bmp и .dib, файлы с расширениями .gif, .jpg, .cur, .ani и .ico. Анимированные gif-файлы поддерживаются только в элементах управления Image, и, кроме того, интерфейс GDI+ предоставляет вам возможность выполнить некоторые базовые операции с графикой, например поворот и переворот (flip) изображений.

Еще одна, не функциональная, возможность, которая может придать вашим приложениям более привлекательный вид (без необходимости вносить в них какие-либо существенные изменения) — это поддержка оформительских наборов Windows XP Themes. Данная возможность позволяет VFP не отставать от последней моды на сменяемые оболочки.

Разумеется, темы — это нечто такое, что обычно применяется к самой Windows и всем тем приложениям, которые их поддерживают, однако, приятно иметь возможность не отставать от современного стандарта на внешний вид и представление и не подвергаться осуждению за использование старомодного интерфейса.

Посмотрите на ту же самую тестовую форму, которую я демонстрировал прежде, однако, на сей раз она оформлена в стиле замысловатой XP-темы:

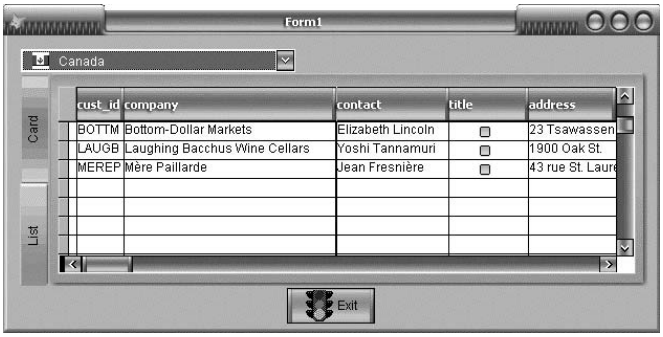

Рис. 3. Возможности использования тем для оформления форм.

Кроме того, теперь мы можем ассоциировать 256-цветные пиктограммы с нашими приложениями. Благодаря этим дополнениям наши приложения получат гораздо более современный вид и будут в большей степени соответствовать современным интерфейсам и средам.

«Вкусности» для экспорта и печати

Вы, вероятно, оказывались в такой ситуации, когда некоторые опытные пользователи Excel обнаруживали, что ваше приложение могло экспортировать максимум 16 384 строк. Для обхода этого ограничения были предложены несколько способов, большинство из которых использовали автоматизацию, дабы позволить заполнение более длинных таблиц, но это оплачивалось потерей производительности. Теперь команда export может сформировать до 65 535 строк, что является предельным значением для самого процессора Excel.

Если вы надеялись прочесть некую статью о замечательных объектно-ориентированных отчетах нового поколения, забудьте об этом. В версии VFP 8.0 таковых не предусмотрено. Однако, взамен команда разработчиков новой версии разрешила несколько проблем из числа тех, о которых «всегда просят».

Во-первых, больше нет необходимости «взламывать» FRX-файлы, чтобы очистить поля tag и tag2 в первой записи. Если вы не хотите хранить информацию о драйвере принтера, вы просто снимаете отметку о выборе такой возможности в меню Report (команда Printer environment).

У вас есть встроенный механизм для печати номеров страниц по трафарету «Page 3 of 7» с помо-

щью новой системной переменной _PageTotal. Если вы пользуетесь таким трафаретом, то механизм создания отчетов, в сущности, выполняет два прохода (так же, как поступало большинство из нас все это время), но неплохо иметь возможность сделать работу естественным путем.

Вы также можете частично решить проблему использования нескольких полос detail, образуя цепочку из заданий на печать и указывая VFP на то, что не надо «прогонять» страницу из принтера после ее печати. Эта возможность имеет некоторые ограничения, поскольку она не работает в режиме preview, но, по крайней мере, при этом вам вернули нечто, что мы привыкли делать еще в стародавние времена в версии FoxPro для DOS.

Еще одна вещь, которая досаждает некоторым пользователям и которую мы не в состоянии были предотвратить, — это диалоговое окно «Printing...». Его больше нет, если воспользоваться новым предложением NODIALOG команды REPORT.

...и некоторые идеи относительно интерфейса

И наконец, в интегрированной среде разработки IDE появилось несколько новых (и необычных) инструментальных средств, которые вы можете имитировать (в самом деле, мы, вероятно, получим доступ к исходному коду в финальной версии) в своих приложениях.

Я закончу эту статью демонстрацией некоторых из этих новых игрушек. Надеюсь, что вскоре более подробно напишу об этих и других замечательных возможностях версии Visual FoxPro 8.0.

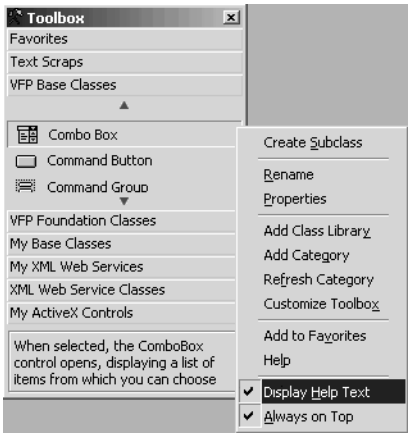

Рис. 4. Инструментальная панель Toolbox.

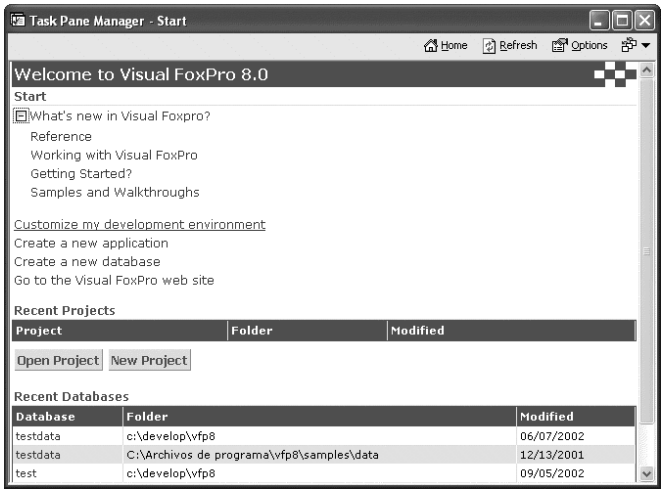

Рис. 5. Панель задач.

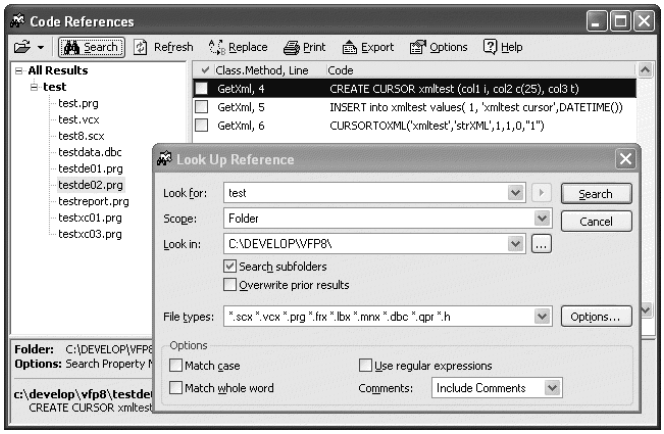

Рис. 6. Диалоговое окно Code References.

❑❑❑

Класс Timer для многократного использования в приложении

Прадип Ачарья (Pradip Acharya)

WIN 7.0 6.0 DOWNLOAD

Входящий в состав VFP базовый класс Timer подключается к конкретной форме, предлагает ограниченные функциональные возможности и выполняет только одну задачу. Разве плохо было бы иметь глобальный класс Timer, способный выполнить любую задачу, не привязаный к конкретной форме, с возможностью многократного использования? В этой статье Прадип Ачарья описывает простой в использовании производный класс xTimer с новыми функциональными возможностями.

Что, если мое приложение не имеет даже формы, а мне необходим таймер? Такое вполне может случиться во многих сценариях, таких, например, как запуск таймера на старте приложения, или в промежутке между исполнением двух форм, или в целях, вообще не связанных с формами. Во-вторых, почему я не могу иметь таймер, действие которого распространяется на все приложение, и который я могу использовать для того, чтобы выполнять разные задачи в разное время? Например, в многопользовательской среде я могу столкнуться с трудностями при открытии таблицы, даже если для этой таблицы предусмотрен режим совместного использования, и тогда мне может понадобиться выждать полсекунды и снова повторить попытку. Класс xTimer предоставляет объекты Timer, которые не привязаны к какой-то конкретной форме, могут многократно использоваться в нескольких задачах (это не одна только жестко привязанная процедура события Timer), обладают рядом других интересных функциональных возможностей. Например, этот класс позволяет вам организовать цикл из нескольких шагов, повторяющийся по истечению некоторого промежутка времени, обеспечивая возможность обрабатывать более сложные задачи. Вы получаете преимущество программирования по интерфейсу и сокращаете программирование, необходимое в каждом конкретном случае, внутри специализированных таймеров. Класс xTimer.VCX вместе с демонстрационной программой xTimer.PRG можно найти в файле-приложении на дискете.

Во-первых, пример

Давайте сначала протестируем демонстрационную программу, чтобы проиллюстрировать ту задачу, которую решает класс xTimer. Распакуйте файлы и поместите их где-нибудь в FoxPro-каталоге. В командном окне Command Window наберите:

```
xTimer()
```

Появится мерцающее сообщение, которое будет присутствовать на экране в течение одной секунды и исчезать за полсекунды. Уже сейчас мы применяем функциональную возможность, о которой говорилось ранее и которая обеспечивает пошаговое исполнение, используя цикл из трех шагов длительностью по 0.5 секунд каждый. Теперь щелкните где-нибудь мышью или нажмите клавишу Esc, чтобы остановить таймер. Прежде чем я продемонстрирую вам независимый от формы характер класса xTimer, обратите внимание на то, что на следующем шаге вы должны успеть щелкнуть мышью по телу сообщения или нажать клавишу Esc до того, пока сообщение остается видимым. Это объясняется тем, что если приложение невидимо, ОС Windows ограничит область ввода данных. Затем наберите:

```
xTimer(1)
```

Обратите внимание на то, что таймер функционирует при полном отсутствии какой-либо формы.

Как создать объекты xTimer

Чтобы он был доступен в пределах приложения, объект xTimer привязывается к объекту _SCREEN, принадлежащему VFP. Хотя в демонстрационном примере используется только объект Timer1, давайте создадим набор объектов, как показано в листинге 1. Этот код взят из программы xTimer.PRG.

Листинг 1. Программный код для создания объектов xTimer.

```
* Поместите следующий блок в загрузочный код StartUp
* своего приложения, чтобы создать таймер(ы)
*--------------------------------------------------
 set classlib to xTimer additive
 private MaxTimers
 MaxTimers = 6
 local i, TimerName, temp
 for i = 1 to m.MaxTimers
    _SCREEN.AddObject( ;
    "Timer" + ALLTRIM(STR(m.i, 10)), ;
    "xTimer")
 endfor
```

Как уничтожить объекты xTimer

Приведенный в листинге 2 код — это программа xTimer.PRG. Обратите внимание на следующее: в том случае, если конкретные переменные содержат ссылки, эти переменные получают значение .f., чтобы избавиться от еще оставшихся объектных ссылок, которые могут препятствовать прекращению работы таймера.

Листинг 2. Программный код для уничтожения объектов класса xTimer.

```
* Поместите следующий блок в программный код ShutDown
* своего приложения, чтобы уничтожить таймер(ы)
 local i, TimerName, temp
 on error *
 for i = 1 to m.MaxTimers
     TimerName = "Timer" + ALLTRIM(STR(m.i, 10))
     temp = "_SCREEN." +m.TimerName+ ".Enabled"
 &temp = .f.
 temp = "_SCREEN." +m.TimerName+ ".SaveValue1"
 &temp = .f.
 _SCREEN.RemoveObject(m.TimerName)
 endfor
 release ClassLib xTimer
 on error
```

Как использовать класс xTimer

Процедура Init класса xTimer имеет три параметра:

```
LPARAMETERS p1millisec, p2cCommu, p3nMaxTrip
```

В условиях рекомендованного и обычного использования, третий параметр либо опускается, либо получает нулевое значение. В этом режиме работы таймер сработает только однажды и блокирует (disable) сам себя перед исполнением команды, переданной в качестве второго параметра. Если вам это необходимо, вы должны сами перезапустить таймер по завершении командной процедуры. Такой режим работы таймера называется режимом плавающего цикла, поскольку таймер не перезапускается до тех пор, пока не завершится командная процедура. Режим плавающего цикла описывается позже. Вот примеры того, как привести таймер в действие:

```
_SCREEN.Timer1.Init(600, "MyCommu(.t.,2)")
_SCREEN.Timer1.Init(60000, "?TIME()", 1)
_SCREEN.Timer1.Init(60, "oForm2.TextBox3.SetFocus")
```

Шестьдесят миллисекунд — это один такт. Во втором примере таймер будет продолжать работать до тех пор, пока не будет остановлен умышленно, как это делается в демонстрационном примере, по счетчику, который обеспечивает третий параметр. Обратите внимание на обязательное использование круглых скобок (), если только второй параметр не является VFP-командой, как в последнем примере. Текст команды не может начинаться со слова RETURN (точнее, может, но будет удален). Это так, потому что слово RETURN, встретившееся в тексте команды, приведет к преждевременному прекращению исполнения программного кода события Timer. Если вы настаиваете на включении в текст команды слова RETURN, создайте переменную, объявленную как public, а затем передайте команду в следующем виде: MyResult = MyCommand(). Вы вправе исследовать весь программный код внутри класса xTimer с помощью утилиты Class Browser.

Как отключить объект xTimer

Чтобы прервать работу объекта xTimer, наберите:

```
_SCREEN.Timer1.Off
```

что совершенно то же самое, что и исполнение предложения:

```
_SCREEN.Timer1.Enabled = .f.
```

Третий аргумент — фиксированный пошаговый цикл

Значение p3nMaxTrip > 0 означает, что свойство _SCREEN.Timer1.Trip будет циклически менять свое значение на один, два... и вплоть до MaxTrip при каждом переключении таймера. Когда этот аргумент опущен или равен 0, значение свойства Trip всегда остается равным 0, и объект xTimer сработает только один раз в режиме плавающего цикла. Если значение параметра p3nMaxTrip равно 1, свойство Trip будет менять свое значение между 1 и 0, а если параметр p3nMaxTrip имеет значение 2, свойство Trip будет менять свое значение между 1 и 2. В процессе работы фактически нет различия между настройками 1 и 2 параметра p3nMaxTrip. В демонстрационном примере используется значение 3, чтобы успеть дважды выдать на экран сообщение до тех пор, пока оно невидимо. Если вы замените значение 3 на 1 или 2 в первой строке программы xTimer.PRG и снова исполните демонстрационный пример, мерцание будет повторяться через равные промежутки времени (0.5 секунды).

Если значение параметра p3nMaxTrip больше 0, включается режим фиксированного цикла, означающий, что таймер не ждет завершения вашей командной процедуры перед тем, как начать следующий цикл. Если исполнение вашей команды требует больше времени, нежели один интервал таймера, вы пропустите такт. Однако, давайте не будем дурачить самих себя. В действительности, как бы там ни было, но таймеры не срабатывают в заданные интервалы времени. ОС Windows своевольно запускает таймеры всякий раз, когда сочтет нужным, в зависимости от того, с каким еще заданием ей необходимо справиться.

Использование класса xTimer с третьим аргументом, значение которого > 0, потребует проявления некоторой смекалки, поскольку вам придется предпринять различные действия в зависимости от состояния свойства Trip внутри вашей командной процедуры. В демонстрационной программе командная процедура объединяет обработку обоих режимов, простого режима плавающего цикла и более сложного режима фиксированного цикла, как показано в листинге 3.

Листинг 3. Программный код для обработки событий плавающего цикла и фиксированного цикла (из демонстрационной программы).

```
* Запуск теста
 _SCREEN.Closable = .t.
 on key label ESC clear events
 on key label LEFTMOUSE clear events
 _SCREEN.Timer1.Init(500, "Blink()", THIRD_ARG)
 Application.Visible = empty( m.p1hide )
 Blink("Init")
 read events

 on key label ESC
 on key label LEFTMOUSE
 wait clear
 set ClassLib to
 Application.Visible = .t.

 PROCEDURE Blink
 LPARAMETERS p1init
 local temp
 DO CASE
 && Переход в режим плавающего цикла
 case empty(_SCREEN.Timer1.MaxTrip )
     if empty(ON("KEY", "ESC"))
         return
     endif
     if empty( m.p1init )
         wait clear
         temp = "1"
     else
         temp = "0"
         wait window nowait noclear ;
           "......Hello World." +CRLF+ ;
           "......ESC to stop (while visible)."+CRLF+ ;
           "......Or, click on this"
     endif
    _SCREEN.Timer1.Init(500, "Blink("+m.temp+")")

 && Переход в режим фиксированного цикла
 case not empty( m.p1init ) or ;
 (_SCREEN.Timer1.ThisTrip != 1)
   wait window nowait noclear ;
     "......Hello World." +CRLF+ ;
     "......ESC to stop (while visible)."+CRLF+ ;
     "......Or, click on this"

 other
     wait clear

 ENDCASE
 return
```

Обратите внимание на то, что в режиме плавающего цикла таймер был перезапущен вручную.

Восстановление временных переменных

При реализации объекта xTimer в своем собственном приложении я столкнулся с некоторой проблемой, поскольку временные переменные успевали менять свое значение даже за то короткое время, которое проходило между запуском таймера и переключением события таймера. Например, спустя 60 миллисекунд мне потребовалось переместить фокус с помощью обращения NextObject.SetFocus. Однако, значение ссылки NextObject успело за это время измениться при исполнении какого-то кода какой-то процедуры обработки наступления какого-то события. В конце концов, когда таймер сработал, фокус оказался у неправильного объекта. Чтобы предотвратить такую ситуацию, не теряя универсальности класса xTimer, я должен был иметь возможность «заморозить» значение любой переменной, чтобы восстановить его перед выполнением командной процедуры. В наибольшей степени это применимо к передаче объектных ссылок, поскольку объектные ссылки не имеют никакого символьного представления. В противном случае, вы можете просто организовать передачу параметров внутри командных аргументов, как показано в листинге 3. По умолчанию класс xTimer позволяет сохранить одну такую переменную, но его возможности можно увеличить. Для этого используются два свойства: SaveVar1 и SaveValue1. Прежде чем запускать таймер путем обращения к его событию Init, присвойте этим двум свойствам следующие значения:

```
WITH _SCREEN.Timer1
  .SaveVar1 = "MyVarName"
  .SaveValue1 = m.MyVarName
ENDWITH
```

Перед исполнением командной процедуры для переменной MyVarName будет восстановлено то ее значение, которое она имела в момент запуска таймера. Как показано в листинге 2, мы гарантируем, что к моменту уничтожения объекта таймера не будет существовать ни одна объектная ссылка.

Заключение

Новый класс xTimer не привязан к какой-либо конкретной форме, доступен глобально, может исполнять при переключении любую команду и может быть многократно использован для достижения различных целей. Кроме того, класс xTimer предоставляет полезные функциональные возможности, например, циклическое исполнение набора шагов в некотором временном интервале и восстановление временных переменных перед их использованием. Программируемый по интерфейсу, класс xTimer обеспечивает более универсальное программирование функций таймера.

❑❑❑

Славный Grid и мудрый Search

Владимир Трухин (Vladimir Trukhin)

Организовать поиск данных в колонках объекта Grid, расположенного на форме, достаточно просто. Так же просто сделать это на второй и третьей формах, но в дальнейшем этот процесс начинает несколько утомлять своей монотонностью. Самое время подумать о том, чтобы создать некоторый класс, способный самостоятельно разобраться с объектами Grid и их колонками, и Владимир Трухин показывает, как этот сделать.

Однажды, устав выстругивать очередную форму с элементами поиска записей, я решил сформулировать требования к способу, предоставляющему возможность простого и эффективного метода поиска на всех моих формах. Получилось нечто следующее:

- Объект, обеспечивающий поиск, должен быть формой, плавающей над формой с искомыми данными.
- Вновь создаваемый объект поиска должен привязываться к активной форме, если таковая существует. Если активных форм нет, объект поиска не должен создаваться.
- Каждая форма с объектами Grid вправе иметь свой объект поиска (две открытые формы не делят между собой одну и ту же форму поиска).
- Закрытие формы должно приводить к уничтожению объекта поиска.
- Объект поиска должен самостоятельно найти на форме все объекты класса Grid, доступные для поиска.
- Он должен самостоятельно определить список колонок каждого объекта Grid, доступных для поиска.
- Он должен предоставить пользователю списки обнаруженных объектов Grid и их колонок.
- Он должен обеспечивать не только поиск, но и отбор записей. (Это было настоятельным желанием опытных заказчиков.) Установленный критерий отбора должен действовать даже в том случае, если объект поиска уничтожен, а форма остаётся загруженной.
- Привязка объекта поиска к форме с данными не должна требовать серьёзной доработки формы.
- Объект поиска должен легко привязываться к формам в любом приложении. (Далее в этой статье я покажу на примере проекта Tasmanian Tradders, как внедрить этот класс в существующее приложение.)

Большинство из перечисленных соображений очевидны, и их решение лежит на поверхности, но некоторые из них заслуживают обсуждения.

На этой сцене два актёра...

Разумеется, объект поиска в состоянии найти все объекты Grid на активной форме, но он должен отобрать только те из них, которые предназначены для поиска.

На этом этапе появляются два участника игры. Первый игрок — это Mr. Search, сканирующий форму, другой — Mr. Grid, усердно сигнализирующий ему. Как и что может сообщить о себе Mr. Grid?

Конечно, Mr. Grid может иметь для этой цели специальные свойства, но это не желательно, так как такой подход потребует создания некоторого подкласса, производного от класса Grid. Это приведёт к тому, что его реализация может потребовать замены объектов Grid на давно и неплохо работающих формах. Это неважная идея — вносить в них беспорядок. Попробуем найти другой способ. Как правило, свойство Comment объекта Grid не заполняет никто. Однако, это свойство может быть хорошим флагом, сигнализирующим о возможности совместного бизнеса с Mr. Search. Кроме того, его значение будет неплохо выглядеть в списке таблиц, доступных для поиска.

Теперь о колонках... Их количество может быть значительно, и нет большого смысла производить поиск в каждой из них. Так же, как и в предыдущем случае, я использую незадействованное свойство, чтобы обозначить колонку как возможную для поиска. Для объекта Column это будет свойство Tag. Оно может хранить символьное значение “Find”, чтобы указать на возможность поиска в колонке.

Использование существующих свойств в качестве флагов продиктовано требованием минимальной доработки формы.

«Где прячутся волшебники?»

Первоначально проблема обнаружения всех возможных объектов Grid на форме показалась мне простым делом. Я думал, что достаточно выполнить следующую последовательность операций, чтобы найти Mr. Grid:

- 1. Получить массив объектов, расположенных на форме.
- 2. Получить ссылки к объектам класса Grid.
- 3. Сохранить ссылки в массиве для дальнейшего использования.

Однако всё оказалось гораздо сложнее. Оказалось, что Mr. Grid не всегда любит сидеть на поверхности формы, а иногда предпочитает прятаться в глубине иерархии объектов.

Как следствие, для решения проблемы пришлось создать метод, рекурсивно вызывающий сам себя для погружения в иерархию объектов. Первоначально он принимает в качестве параметра ссылку на объект формы и находит все объекты, расположенные на этом уровне. Затем метод просматривает полученный список с целью найти в нём объекты Grid с непустым свойством Comment. Если элемент списка не является объектом класса Grid, то это повод предположить, что он может содержать такой объект. В этом случае достаточно вызвать этот метод рекурсивно и передать ему ссылку на объект.

Если сканированный объект является экземпляром класса Grid, то необходимо зарегистрировать его в массиве, представляющем список обнаруженных объектов этого класса. Каждая строка такого списка хранит два значения:

- Название объекта Grid, полученное из свойства Comment.
- Ссылку на объект Grid.

Код метода для обнаружения всех подходящих объектов выглядит примерно так:

```
PROCEDURE DetectGrids(loCurObject)
 LOCAL ARRAY laGrids(1)
 LOCAL lnElement, lcObjRef, loObjRef
 IF AMEMBERS(laGrids,loCurObject,2)>0
   FOR lnElement=1 TO ALEN(laGrids,1)
     lcObjRef='loCurObject.'+laGrids(lnElement)
     loObjRef=&lcObjRef
     IF UPPER(ALLT(loObjRef.BaseClass))=='GRID' ;
         and !EMPTY(loObjRef.Comment)
       IF THIS.NumberOfGrids=0
         THIS.NumberOfGrids=1
       ELSE
         THIS.NumberOfGrids=THIS.NumberOfGrids+1
         DIMENSION THIS.Grids(THIS.NumberOfGrids,2)
       ENDIF
       THIS.Grids(THIS.NumberOfGrids,1)=loObjRef.Comment
       THIS.Grids(THIS.NumberOfGrids,2)=loObjRef
       IF EMPTY(loObjRef.Tag)
         loObjRef.Tag='gc'+;
         ALLT(STR(INT(RAND(SECONDS())*1000000)))+;
         ALLT(STR(THIS.NumberOfGrids))
       ENDIF
     ELSE
       THIS.DetectGrids(loObjRef)
     ENDIF
   ENDFOR
 ENDIF
 ENDPROC
```

Каждому мячу — своя лунка…

Возможно, в предыдущем кусочке кода вызывает лёгкое недоумение манипуляция со свойством Tag. Она призвана решить следующую проблему...

Это проблема отбора записей. Mr. Search и Mr. Grid должны решить, кто из них будет управлять переменной для выражения фильтра. Допустим, что это привилегия Mr. Search. Он заносит выражение фильтра в некоторую переменную, запускает фильтр и уходит... О-о-пс... Наши друзья имеют большие проблемы:

- Если имя переменной фиксировано, то Mr. Search не сможет организовать отбор на множественных формах. Может быть открыто несколько форм, и каждая из них может иметь свою форму поиска в то же самое время, что и другие.
- Конечно, Mr. Search при каждом своём появлении может создавать новую переменную со случайным именем, но даже страшно подумать сколько он их создаст, поскольку переменная должна оставаться в памяти всё время, пока действует фильтр, даже после исчезновения Mr. Grid.
- А что будет, если Mr. Grid захочет уйти, не дожидаясь, пока Mr. Search вернёт результат? (Форма поиска не модальная, а вида «always on top»). Правильно, он навсегда оставит все переменные фильтра в дебрях бесконечной памяти.

Совсем другое дело, если ответственность за переменную выражения фильтра возьмёт на себя Mr. Grid:

- Он может как угодно долго держать её у себя в кармане.
- Если Mr. Search понадобится это имя, чтобы установить фильтр, он может спросить об этом у Mr. Grid. Например, вот так:

```
PROCEDURE SelectedFilterVariable
     LOCAL loGridRef
     loGridRef=thisform.SelectedGridRef()
     return ALLT(loGridRef.Tag)
ENDPROC
```

- И наконец, уходя со цены, он обязательно уничтожит никому теперь не нужную переменную. Для этого ему нужно сделать совсем немного:

```
IF !EMPTY(this.Tag)
     LOCAL lcVarName
     lcVarName=this.Tag
     RELEASE &lcVarName
ENDIF
```

Свойство Tag использовано с целью упрощения модификации уже существующих форм.

Внедрение класса в существующий проект

Я не привожу детальное описание исходного кода, поскольку он полностью представлен в прилагаемом к статье файле и не содержит никаких хитростей. Более интересным и полезным будет описание внедрения этого класса в существующий проект. В качестве примера я выбрал проект Tasmanian Traders, который есть у каждого программиста Visual FoxPro. На следующем примере (см. рис. 1) представлен внешний вид класса поиска, работающего с формой Customers.

Внедрение класса требует выполнения нескольких простых шагов:

- 1. Расположить файл определения класса в структуре каталогов Tasmanian Traders.
- 2. Модифицировать форму CUSTOMERS.
- 3. Модифицировать главное меню приложения.

Определение класса и каталоги проекта Tasmanian Traders

Определение класса содержится в файле GRIDSEARC.PRG, который, в свою очередь, находится в Zip-файле. Достаточно извлечь его из архива и разместить в программном каталоге проекта. Это каталог C:\Program Files\Microsoft Visual FoxPro 7\Samples\Tastrade\Progs, который соответствует стандартной установке Visual FoxPro 7.0. Однако различные ПК могут иметь различные пути к этому каталогу. Если на вашем компьютере примеры установлены где-либо в другом месте, вы можете использовать вызов функции HOME(2), чтобы получить маршрут к каталогу примеров.

Модификация формы Customers

После того как описание класса размещено в подходящем месте, нужно сделать следующий шаг и модифицировать форму Customers и объект Grid, размещённый на ней.

В первую очередь необходимо создать свойство формы FinderRef и инициализировать его значением

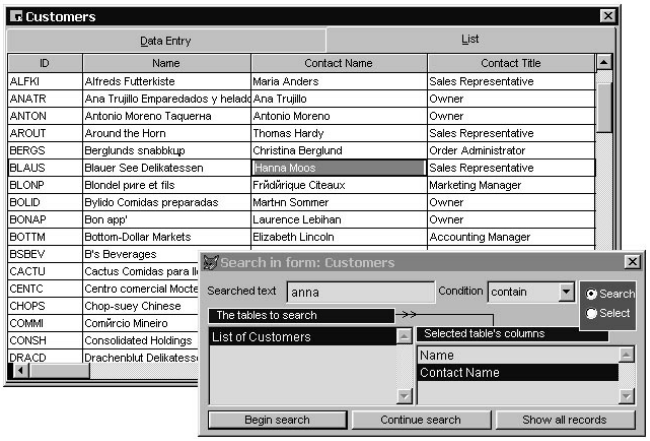

Рис. 1. Класс поиска, работающий с формой Customers.

NULL. Это свойство будет хранить ссылку на объект поиска.

```
FinderRef = NULL
```

Во-вторых, необходимо добавить в форму метод StartFinder(). Он будет создавать экземпляр класса поиска.

```
PROCEDURE StartFinder
   thisform.FinderRef=NEWOBJECT('GridSearch', ;
     'PROGS\GridSearch.PRG','',THIS)
   LOCAL loRef
   loRef=thisform.FinderRef
   loRef.Show()
 ENDPROC
```

Затем необходимо переопределить метод Destroy(). Он должен уничтожить объект поиска при закрытии формы.

```
PROCEDURE Destroy
   IF TYPE('thisform.FinderRef')='O' ;
       and !ISNULL(thisform.FinderRef)
     LOCAL loRef
     loRef=thisform.FinderRef
     loRef.RELEASE()
     thisform.FinderRef=NULL
   ENDIF

   DODEFAULT()
 ENDPROC
```

Следующее, что необходимо сделать, — это модифицировать объект grdList, расположенный на этой форме, следующим способом:

- 1. Переопределить метод Destroy() объекта Grid для уничтожения переменной фильтра:

```
PROCEDURE Destroy
  IF !EMPTY(this.Tag)
    LOCAL lcVarName
    lcVarName=this.Tag
    RELEASE &lcVarName
  ENDIF
  DODEFAULT()
ENDPROC
```

- 2. Определить объект Grid как используемый для поиска, присвоив свойству Comment объекта grdList значение “List of Customers”. Это значение, более понятное, чем действительное имя объекта на форме, пользователь увидит в объекте Listbox на форме поиска.

```
grdList.COMMENT="List of Customers"
```

- 3. Свойству Tag всех колонок, имеющих тип данных Character и необходимых для поиска, присвоить значение “Find”:

```
grdList.grcName.TAG="Find"
grdList.grcContactName.TAG="Find"
```

Модификация главного меню приложения

Главное меню проекта определено в файле MENUS\MAIN.MNX. Достаточно добавить новый пункт, например “Поиск в активной форме”, в меню Edit и написать процедуру для этого пункта.

```
LOCAL loActiveForm, loMrSearch
 IF TYPE('_screen.ActiveForm')!='O' ;
   OR ISNULL(_screen.ActiveForm)
   RETURN
 ENDIF

 loActiveForm=_screen.ActiveForm
 IF IsMember(loActiveForm,'StartFinder')
   loActiveForm.StartFinder()
 ENDIF
 RETURN


 FUNCTION IsMember(loObject,lcMember)
 LOCAL llPresent,lnNumberOfMembers, lnIndex
 llPresent=.F.
 LOCAL ARRAY laMembers(1)
 lcMember=UPPER(ALLT(lcMember))
 lnNumberOfMembers=AMEMBERS(laMembers,loObject,1)

 IF lnNumberOfMembers>0
   FOR lnIndex=1 TO lnNumberOfMembers
     IF UPPER(ALLT(laMembers(lnIndex,1))) == lcMember
       llPresent=.T.
       EXIT
     ENDIF
   ENDFOR
 ENDIF
 RETURN llPresent
 ENDFUNC
```

Итак, все изменения в проекте сделаны. Настало время ввести в окне команд магическую команду DO MAIN, чтобы запустить Tasmanian Traders и открыть форму Customers.

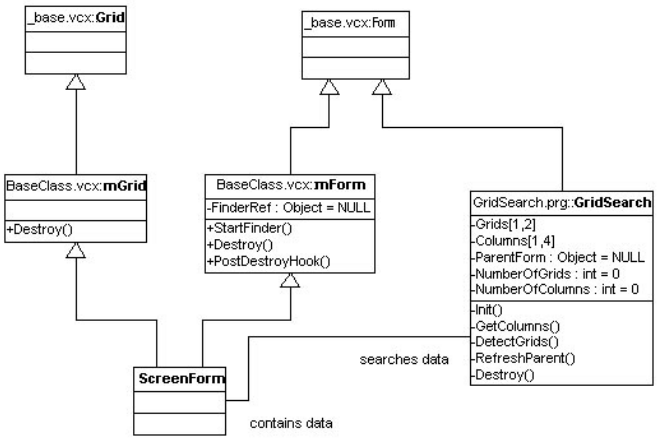

Рис. 2. Структура классов Form и Grid.

Заключение

Описанный класс не является панацеей от всех проблем, которые возникают в связи с необходимостью организации поиска данных, но он может быть успешно и легко применён в большинстве случаев. Всё, что необходимо сделать, — это создать классы Form и Grid (показаны на рис. 2), которые включают все необходимые свойства и методы, описанные ранее, чтобы облегчить работу по созданию ваших форм поиска.

Трухин Владимир Леонидович — ведущий инженер-программист отдела АСУ в ОАО «Воткинская ГЭС», г. Чайковский.
url: http://www.geocities.com/votges
email: vlt@votges.ru

❑❑❑

Подсказка в форме раскрывающегося списка

Предраг Боснич (Predrag Bosnic)

WIN 7.0 DOWNLOAD

В этой статье Предраг Боснич изучает возможность реализации элемента «подсказка в форме раскрывающегося списка» (ComboBox Item ToolTip) в Visual FoxPro 7 и попутно делится своим опытом.

Помню, в самом начале работы с Visual FoxPro 7, я хотел написать несколько строк кода для метода Init моей формы. Окно кода было открыто, но раскрывающийся список Procedure показывал только метод Load. Щелкнув этот раскрывающийся список, я впервые увидел элемент «подсказка в форме раскрывающегося списка», реализованный в Visual FoxPro 7. Хочу сказать, что был очень удивлен (приятно), поскольку считаю, что иногда этот тип помощи очень полезен.

Анализ

Из рисунка 1 следует, что этот тип элемента интерфейса построен на основе многострочной подсказки. Я уже имел предварительный опыт реализации элемента «многострочная подсказка», и у меня была уверенность, что я смогу также реализовать и другой ее вид.

Давайте посмотрим, как этот элемент работает в Visual FoxPro 7. Щелчок на раскрывающемся списке открывает его, но вы не увидите подсказки. Подсказка появляется, если вы переместите указатель мыши с выбранного элемента, и остается открытой до тех пор, пока вы не закроете список или не щелкните мышью вне его. Подсказка может содержать одну или несколько строк текста и располагаться по левую или по правую сторону от раскрывающегося списка. Повторюсь еще раз: подсказка не содержит элемента timer и остается открытой до тех пор, пока вы не выполните любое из упомянутых выше действий.

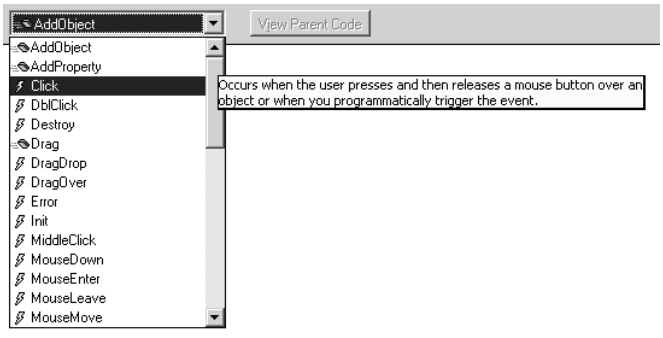

Рис. 1. Реализация подсказки для раскрывающегося списка в Visual FoxPro.

Трудно сказать, какое максимальное число символов Visual FoxPro может поместить в этот элемент, но моя подсказка может показать 254. Думаю, что для определения максимальной ширины хорошим выбором является строка в 60 символов. Это означает, что подсказка может содержать пять строк.

Требования, предъявляемые к разработке

Требования к нашей разработке, вытекающие из проведенного мною анализа, выглядят так:

- Подсказка в форме раскрывающегося списка должна вмещать текст до 254 символов. Такое же значение было определено для многострочной подсказки.
- Максимальная длина строки должна составлять 60 символов. Такое же значение было определено для многострочной подсказки.
- Подсказка в форме раскрывающегося списка должна активироваться/деактивироваться независимо от собственных подсказок Visual FoxPro.
- Этот элемент должен быть прост в использовании. Если потребуется, необходимо создать соответствующий конструктор (builder).
- Подсказка должна иметь свойства ForeColor/BackColor.
- Этот элемент должен быть совместим с существующими приложениями.

Разработка и реализация

Каждая строка раскрывающегося списка имеет связанный с ней текст. Учитывая это обстоятельство, я не вижу решения поставленной задачи в рамках собственного элемента Visual FoxPro 7 — ComboBox. Если еще принять во внимание тот факт, что нам нужно вычислить строку, находящуюся под указателем мыши, то идея использования собственного раскрывающегося списка Visual FoxPro 7 мне совсем не

нравится. Где-то рядом должно быть другое решение для ленивых программистов вроде меня! Похоже, что проблема заключается в том, как определить строку, находящуюся под указателем мыши. Если я смогу определить ее, то останется лишь найти текст, связанный с этой строкой, и показать его в окне подсказки – что может быть проще!

Я вспомнил довольно интересное обстоятельство: элемент MS TreeView имеет метод HitTest, позволяющий программисту знать, какой узел находится под указателем мыши. Именно это свойство использует операция Drag-n-Drop. Если бы собственные элементы Visual FoxPro 7 ComboBox или ListBox могли иметь нечто подобное – это было бы идеально. К сожалению, ничего подобного для них я не нашел.

Затем меня осенила еще одна мысль: когда я работал с бета-версией Visual FoxPro 7, то читал об элементе Grid и его методе GridHitTest. Действительно, элемент Grid имеет этот метод и может возвращать положение ячейки, расположенной под указателем мыши. Эврика! Это могло быть решением. Если вы пока не догадались, каким образом, то представьте контейнер с раскрывающимся списком, расположенным над сеткой. Сетка имеет только один столбец, причем заголовок, горизонтальная полоса прокрутки, маркер записи и горизонтальные линии являются скрытыми. Рисунок 2 иллюстрирует эту идею.

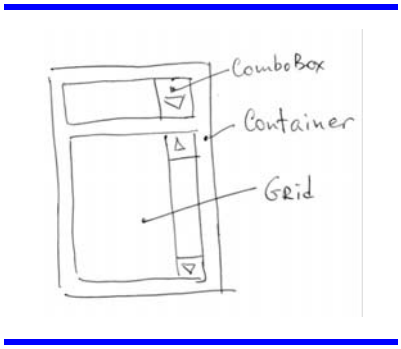

Рис. 2. Идея для подсказки в форме раскрывающегося списка.

Как работает этот элемент? Контейнер показывает только раскрывающийся список. Когда я щелкаю этот список, он не раскрывается, вместо этого меняется размер контейнера (высота) и он показывает раскрывающийся список и сетку. Когда я перемещаю указатель мыши над сеткой, метод GridHitTest возвращает положение ячейки. Я могу выделить эту ячейку, найти текст подсказки и показать ее. И, наконец, когда я щелкаю любую строку сетки для выбора опции, я деактивирую подсказку, опять меняю размер контейнера (высоту) и показываю только раскрывающийся список. Как вы знаете, элемент Grid основан на таблице, и я помещаю мой раскрывающийся список в эту таблицу. Это не так плохо, потому что на стадии разработки я могу создать таблицу со списком в первом столбце и затем добавить второй столбец с текстом подсказки и третий столбец с номерами для поддержания порядка элементов. Таблица 1 иллюстрирует эту идею.

Таблица 1. Метаданные подсказки в форме раскрывающегося списка.

xName	xOrder	xDesc
Option-1	1	Это текст подсказки для Option-1.
Option-2	2	Это текст подсказки для Option-2.
Option-3	3	Это текст подсказки для Option-3.

Подобным образом я мог бы использовать элемент ListBox и вычислить положение под указателем мыши. Элемент ListBox не обладает методом HitTest, но обладает свойством TopIndex. Используя это свойство, специальный шрифт для ListBox и событие MouseMove для ListBox, я могу вычислить, какой элемент находится под указателем мыши. Однако в качестве решения я использую элемент Grid, поскольку мне необходима таблица для хранения текста моей подсказки в форме раскрывающегося списка.

Давайте создадим библиотеку wbComboToolTip.vcx и новый класс ttScr1, основанный на форме. Установим следующие свойства:

```
TitleBar = 0
BorderStyle = 1
AlwaysOnTop = .T.
```

Добавим элемент EditBox к форме и установим следующие свойства:

```
BackColor = rgb(239,241,194)
DisabledBackColor = rgb(239,241,194)
DisabledForeColor = rgb(0,0,0)
FontName = Tahoma
FontSize = 9
ScrollBars = 0
SpecialEffect = 1
Left = -1
Top = -1
```

Последние два свойства будут создавать эффект тени, поскольку граница формы состоит из одной линии. Метод Init формы ToolTip содержит следующий код:

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

* toParentCbo - ссылка на родительский раскр-ся список
* tcToolTipText - текст подсказки
* tnLeft - xCoord для объекта "подсказка"
* tnTop - yCoord для объекта "подсказка"
* tnForeColor - основной цвет
* tnBackColor - цвет фона

 WITH thisform
   .oParentCbo = toParentCbo
   .comment = ALLTRIM(tcToolTipText)
```

```
   .Edit1.Value = ALLTRIM(this.comment)
   IF ! EMPTY(tnForeColor)
     .Edit1.DisabledForeColor = tnForeColor
   ENDIF
   IF !EMPTY(tnBackColor)
     .Edit1.DisabledBackColor = tnBackColor
   Endif
 ENDWITH

 LOCAL jnTTwidth as Integer
 jnTTwidth = thisform.CalcSize()
 IF (tnLeft+toParentCbo.Width+jnTwidth)> ;
   _screen.Width
   this.Left = tnLeft - jnTTwidth
 ELSE
   this.Left= tnLeft + this.width
 ENDIF
 thisform.Top  = tnTop
```

Как видите, метод Init определяет положение левого края для подсказки, поскольку оно зависит от ширины подсказки. Метод CalcSize вычисляет ширину. Вызываемый код вычисляет положение верхнего края для подсказки, поскольку имеет всю необходимую для этого вычисления информацию.

Теперь я могу создавать элемент ComboBox. Как я уже говорил, я использую контейнер, собственный ComboBox и элемент Grid. На рисунке 3 показано, как это будет выглядеть.

Я использовал ту же библиотеку — wbComboToolTip.vcx, но на этот раз я создал новый класс, основанный на классе контейнеров, с тем же именем — wbComboToolTip. Прежде чем я начну кодировать, позвольте мне добавить несколько необходимых свойств.

- cTable — имя таблицы, содержащей все элементы строки списка и текст подсказки.
- lEscape — пользователь покидает элемент без изменения выбора.
- Wb_ComboToolTipBackColor — цвет фона подсказки.
- Wb_ComboToolTipForeColor — основной цвет подсказки.
- Wb_LcboTTactive — подсказка уже активна.
- Wb_oCboToolTip — ссылка объекта на элемент подсказка.
- xStyle — 2D/3D стиль.

Теперь я добавляю раскрывающийся список в контейнер и устанавливаю некоторые свойства.

```
Name = wbcTT_combobox1
DisabledBackColor = 255,255,255
Left = 0
Top = 0
```

Далее для метода ComboBox.MouseDown:

```
SELECT (this.Parent.cTable)
 with this.parent
   .height = .nDesignHeight
   .Timer1.enabled = .t.
   .wbcTT_grdList.setFocus()
 endwith
 this.Enabled = .f.
```

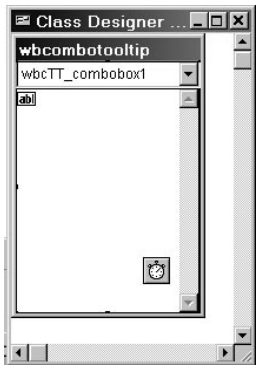

Рис. 3. Элемент wbComboToolTip в Class Designer.

Как видите, элемент Container изменяет высоту; таймер становится активным; раскрывающийся список отключен; и сетка получает фокус.

Следующим шагом мы добавим элемент Grid в контейнер. Подгоним ширину сетки под ширину раскрывающегося списка и установим следующие свойства:

```
Name = wbcTT_grdList
 AllowHeaderSizing = .F.
 AllowRowSizing = .F.
 ColumnCount = 1
 DeleteMark = .F.
 GridLines = 0
 HeaderHeight = 0
 HiglihtRowLineWidth = 0
 RecordMark = 0
 ScrollBars = 2 (Vertical)
 SplitBar = .F.
 Left = 0
 Top = 24
 RecordSourceType = 1
 RecordSource = (None)
```

Конечно, наиболее значимым методом является Column.MouseMove, и вот его код:

```
* Column.MouseMove
 LOCAL jnWhere_Out as Integer
 Local jnRelRow_Out as integer, joParent as Object
 joParent = .F.
 jnWhere_Out = 0
 jnRelRow_Out= 0
 this.parent.GridHitTest(nXCoord, nYCoord, ;
   @jnWhere_Out, @jnRelRow_Out)
 this.parent.ActivateCell(jnRelRow_Out, 1)
 joParent = this.Parent.parent

 SELECT (joParent.cTable)
 LOCATE FOR xname = Alltrim(this.wbcTT_text1.value)
 jcToolTipText = Alltrim(xDesc)

 LOCAL jnLeft, jnTop, jlCreateToolTip
 jnLeft = 0
 jnTop  = 0
 joParent.RetPosition(joParent, @jnLeft, @jnTop)
 jnTop  = jnTop  + joParent.wbcTT_combobox1.Height + ;
          (jnRelRow_out - 1)* this.parent.RowHeight
 *-----------------------------------------------------
 jlCreateToolTip = !joForm.wb_lCboTTactive
 joParent.showtooltip(jnLeft, jnTop, jcToolTipText, ;
   jlCreateToolTip)
 joParent.timer1.enabled = .t.
```

Как я уже говорил, наиболее важной частью этого подхода является определение строки, находящейся под указателем мыши. Переменная jnRelRow_Out содержит это значение. Определив его, я нахожу нужную строку в сетке, запись в таблице и могу по-

лучить текст подсказки. Метод RetPosition вычисляет положение элемента «подсказка» и возвращает значение в переменные jnLeft и jnTop.

```
* wbToolTip.RetPosition
 LPARAMETERS toObject, tnLeft, TnTop
 * toObject - ссылка на объект
 * tnLeft - возвращает положение левого края
 * tnTop - возвращает положение верхнего края

 IF Upper(toObject.Parent.BaseClass) = 'FORM'
   tnLeft = tnLeft + toObject.Left
   TnTop  = tnTop  + toObject.Top
   tnLeft = tnLeft + Sysmetric(3) + ;
     toObject.Parent.Left
   tnTop  = tnTop  + Sysmetric(9) + Sysmetric(4) + ;
     toObject.Parent.Top
   IF Upper(toObject.BaseClass) = 'PAGEFRAME'
     tnTop = tnTop + (toObject.Height - ;
     toObject.PageHeight)
   ENDIF
   RETURN .t.
 ELSE
   IF PemStatus(toObject,'Left',5)
     tnLeft = tnLeft + toObject.Left
     TnTop  = tnTop  + toObject.Top
   ENDIF
   IF Upper(toObject.BaseClass) = 'PAGEFRAME'
     tnTop = tnTop + (toObject.Height - ;
     toObject.PageHeight)
   ENDIF
   joObject = toObject.Parent
   RETURN this.RetPosition(joObject, ;
     @tnLeft, @TnTop)
 ENDIF
```

Имея все необходимые значения, я вызываю метод ShowToolTip:

```
* wbToolTip.ShowToolTip
 LPARAMETERS tnLeft, tnTop, tcToolTipText, ;
   tlCreateToolTip

 * tnLeft - положение левого края подсказки
 * tnTop    - положение верхнего края подсказки
 * tcToolTipText - текст подсказки для отображения
 * tlCreateToolTip - флажок, показывающий
 *                   создана подсказка или нет

 LOCAL joObject, jnLeft, jnTop, jcToolTip, ;
   jnHeightBalonForme, jnCorection, jnTTwidth

 IF EMPTY(tcToolTipText)
   return
 ENDIF
 *--- Активирует окно подсказки ---
 IF tlCreateToolTip = .t.
   this.wb_oCboToolTip =CREATEOBJECT('ttscr1',;
     this,tcToolTipText,tnLeft,tnTop, ;
     this.wb_ComboToolTipForeColor,;
     this.wb_ComboToolTipBackColor)
   WITH this.wb_oCboToolTip
     .visible = .t.
 .Show()
   ENDWITH
   this.wb_lcbottactive = .t.
 ELSE
   WITH this.wb_oCboToolTip
     .Top = tnTop
     .Comment = tcToolTipText
     .Edit1.Value = tcToolTipText
     jnTTwidth = .CalcSize()
     IF (tnLeft + this.Width + jnTTwidth) > ;
       _screen.Width
       .Left = tnLeft - jnTTwidth
     ELSE
       .Left= tnLeft + this.width
     ENDIF
   ENDWITH
 ENDIF
```

Используя флажки wb_lCboActive и tlCreateToolTip, я могу контролировать поведение подсказки. Она создается только один раз, и движение указателя мыши будет перемещать подсказку и показывать соответствующий текст.

Вернемся назад к объекту Column элемента Grid. Когда указатель мыши покидает столбец, срабатывает событие MouseLeave.

```
* Column.MouseLeave
 LOCAL ox as Object
 ox = Sys(1270)
 IF Type('oX') = 'O' AND !IsNull(oX)
   IF Upper(Left(oX.name,6)) # 'WBCTT_'
         this.parent.parent.lEscape = .t.
         this.wbcTT_Text1.Click()
   ENDIF
 ENDIF
```

На самом деле, можно переместить указатель мыши очень быстро, и тогда это событие не сработает. По этой причине я добавил в контейнер элемент Timer. Метод Timer будет проверять, находится ли указатель мыши над элементом или нет.

Метод Activate формы ToolTip содержит следующий код:

```
* Form.Activate
 joX = this.RetTopForm(thisform.oParentCbo)
 joX.Show()
```

Я не могу разрешить форме подсказки получить фокус. Чтобы избежать этого, форма ToolTip должна активировать главную форму (форму, содержащую наш элемент ComboBox). Для того чтобы узнать ссылку на эту форму, я вызову метод RetTopForm.

```
* ToolTipForm.RetTopForm
 LPARAMETERS toObject

 IF Upper(toObject.Parent.BaseClass) = 'FORM'
   RETURN toObject.Parent
 ELSE
   joObject = toObject.Parent
   RETURN this.RetTopForm(joObject)
 ENDIF
```

Кроме того, я не могу позволить указателю мыши входить в область ToolTip. Чтобы добиться этого, используется событие MouseEnter на поле редактирования элемента «подсказка» и код, приведенный ниже:

```
thisform.oParentCbo.lEscape = .t.
thisform.oParentCbo.wbcTT_Text1.Click()
```

Чтобы реализовать эту идею, мне необходимо создать таблицу со следующей структурой:

- xName, char (25) — имя строки.
- xOrder, I — порядковый номер (позволяющий нам сортировать элементы).
- xDesc, char (254) — текст подсказки.

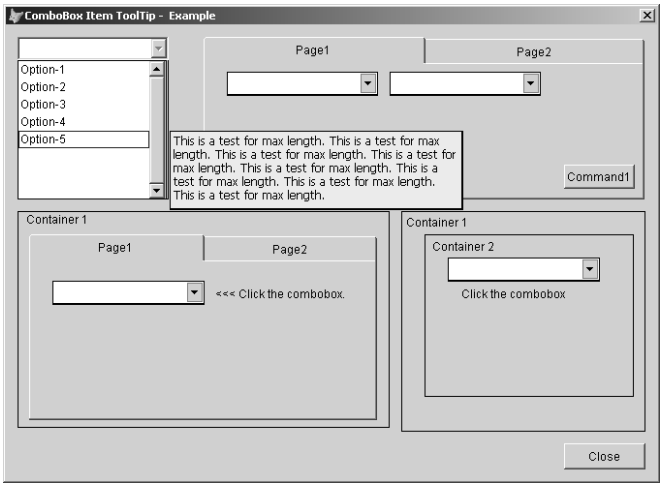

Рис. 4. Пример подсказки в форме раскрывающегося списка в действии.

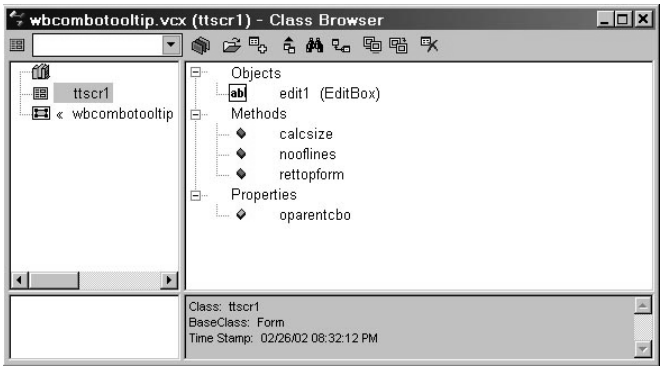

Рис. 5. Класс ttScr1, его методы и свойства.

Теперь я заполню эту таблицу несколькими записями и, похоже, пришло время протестировать мой элемент интерфейса. Все, что нужно сделать для этого, – создать форму, добавить на нее мой элемент ComboBox и установить свойство cTable для имени таблицы. Рисунок 4 показывает более сложный пример, использовавшийся мною для проверки этого элемента. Как видите, элемент находится в форме, но также в контейнере двухуровневой глубины и на вкладке элемента PageFrame. Для того, чтобы хранить все необходимые детали, каждый элемент имеет собственную таблицу.

Библиотека wbComboToolTip

Эта библиотека содержит два класса: ttScr1 (см. рис. 5, таблицы 2 и 3) и wbComboToolTip (см. рис. 6, таблицы 4 и 5).

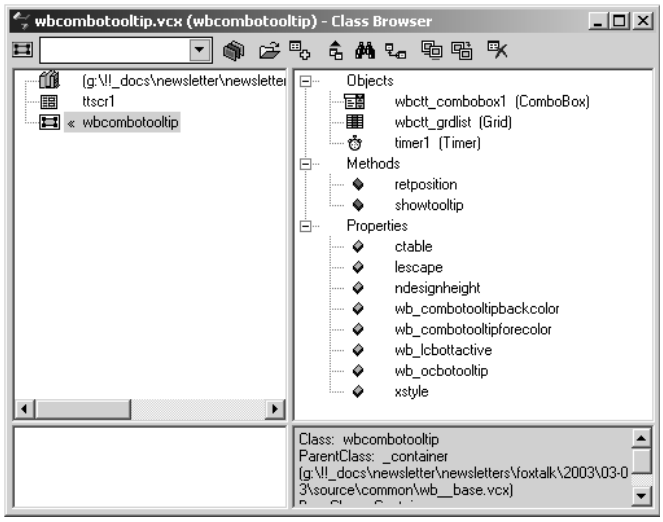

Рис. 6. Класс wbComboToolTip, его методы и свойства.

Таблица 2. Свойства (ttScr1).

Имя	Тип	Описание
oParentCbo	Obj	Ссылка на родительскую форму.

Таблица 3. Методы (ttScr1).

Имя	Тип	Описание
CalcSize	Private	Используется внутри класса. Вычисляет размер окна подсказки.
RetTopForm	Private	Используется внутри класса. Возвращает ссылку на форму.
NoOfLines	Private	Используется внутри класса. Возвращает число строк, используемых в подсказке.

Таблица 4. Свойства (wbComboToolTip).

Имя	Тип	Описание
cTable	Char	Имя таблицы, хранящее определение подсказки в форме раскрывающегося списка.
lEscape	L	Используется внутри класса. Пользователь оставляет элемент без изменения выбора.
nDesignHeight	Int	Высота элемента ComboBox (раскрывающийся список открыт). Пользователь определяет ее на стадии разработки.
Wb_ComboToolTipBackColor	Int	Цвет заливки.
Wb_ComboToolTipForeColor	Int	Основной цвет.
Wb_LcboTTactive	L	Флажок используется внутри класса, показывает, активна ли подсказка.
Wb_oCboToolTip	L	Используется внутри класса. Ссылка на подсказку.
xStyle	Int	Стиль подсказки. По умолчанию = 1 (плоский), 0 = 3D.

Таблица 5. Методы (wbComboToolTip).

Имя	Тип	Описание
RetPosition	Private	Используется внутри класса. Возвращает положение элемента «подсказка».
ShowToolTip	Private	Используется внутри класса. Показывает подсказку.

Конструктор

Использование элемента «подсказка в форме раскрывающегося списка» совсем несложно, но, в соответствии с одним из базовых принципов, «элемент интерфейса должен быть прост в использовании». Поэтому необходимо создать конструктор, помогающий нам на стадии разработки.

Как я уже говорил, все, что необходимо сделать, — это установить одно свойство (cTable) и ввести в таблицу данные о строках и тексте подсказки. Это именно те действия, которые я жду от конструктора. Рисунок 7 показывает главную форму конструктора и позволяет мне определить имя таблицы, ввести тексты для Item и ToolTip, переставлять строки и видеть эту подсказку. Ширина поля редактирования та же, что и ширина элемента «подсказка», и это дает вам в процессе разработки представление о том, как она будет выглядеть.

Кроме того, инсталлятор конструктора, снабженный исходным кодом (доступен в файле для загрузки), поможет вам с его установкой.

Усовершенствования

Этот класс выглядит прекрасно. Но его можно усовершенствовать следующим образом:

- Улучшить пошаговый поиск, поскольку он не работает.
- Каждый раскрывающийся список должен иметь таблицу, хранящую метаданные. Сделать возможным объединять два и более определений списков в одной таблице.
- Когда подсказка в форме раскрывающегося списка активна, используется таблица, содержащая метаданные для нее. После деактивации подсказки эта таблица остается выбранной. Было бы хорошо управлять этим из класса ComboBox Item ToolTip.

Реализовать эти усовершенствования – вот задача для вас (и для меня).

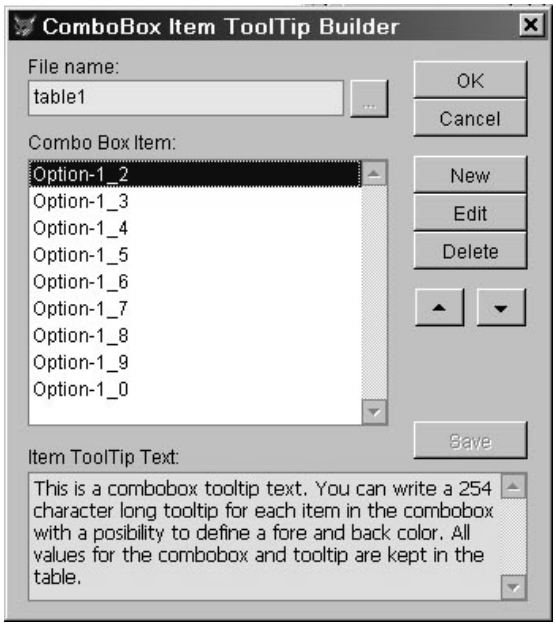

Рис. 7. Главная форма конструктора.

Заключение

Элемент «подсказка в форме раскрывающегося списка» не часто увидишь в приложениях, также не встречается он и в языках программирования. Этот элемент не стоит помещать в каждую вашу форму, но когда вы найдете подходящее приложение для его использования, ваши пользователи будут приятно удивлены.

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

❑❑❑

Что заключено в имени?

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

В этом месяце Энди Крамек и Марсиа Акинс проникают в мир шаблонов проектирования (design pattern), пытаясь разрешить сбивающую с толку загадку присвоения имен. В книге «Design Patterns» (Gamma, Helm, Johnson, and Vlissides, Addison Wesley, 1995), имя «Wrapper» является синонимом для шаблонов «Adapter» и «Decorator». Однако, поскольку совершенно ясно, что это различные шаблоны, логично предположить, что оба они не могут называться «Wrapper».

Марсиа: Я недавно просматривала шаблон Decorator в книге «Design Patterns: Elements of Reusable Object-Oriented Software» и нашла нечто весьма странное, чего не замечала ранее. Имя «Wrapper» приводится в ней как альтернатива для «Decorator».

Энди: Да, мне это кажется несколько непродуманным, хотя я не уверен, что соглашусь с тем, что они имеют в виду именно этот шаблон. Что же показалось тебе странным?

Марсиа: Но ведь они также приводят «Wrapper» как альтернативное имя для «Adapter».

Энди: В самом деле?! Да, действительно. И, насколько я вижу, строчка алфавитного указателя тоже говорит: Wrapper, see adapter, decorator.

Марсиа: Итак, маловероятно, что это ошибка. Тем не менее, несомненно, это бессмыслица, поскольку одной из главных причин для присвоения имен шаблонам проектирования являлось создание однозначного словаря для разработчиков, который позволил бы им проще общаться. Этот случай — очевидное нарушение этого принципа! А ведь в книге он излагается как первый из четырех основных элементов для шаблона проектирования в следующем виде:

1. Имя шаблона является помощником, используемым нами для описания проблемы проектирования, ее решений и заключается в слове или двух. Присвоение имен шаблонам сразу увеличивает наш проектировочный словарь. Он позволяет нам работать на более высоком уровне абстракции. Имея словарь для шаблонов, мы можем обсуждать их с нашими коллегами, в наших документах и даже с собой. Это упрощает обдумывание проектов и передачу их другим. Нахождение подходящего имени является одной из труднейших задач разработки нашего каталога.

Энди: Да, трудно не согласиться с заключительной частью этого параграфа! Как я упоминал ранее, я не обязательно согласен с тем, что «Wrapper» на самом деле является синонимом для чего-либо еще.

Марсиа: На чем ты основываешься?

Энди: А вот на чем. Все шаблоны для чего-то предназначены. Например, назначением шаблона Adapter является преобразование интерфейса одного класса в другой интерфейс, ожидаемый клиентом. Это позволяет классам работать вместе, что невозможно сделать другим способом, поскольку эти интерфейсы несовместимы. Сильно сказано, не так ли?

Марсиа: Может быть, если бы я знала, что это означает. Ты намекаешь, что Adapter описывает решение проблемы совместимости двух различных интерфейсов, выступая между ними в качестве механизма трансляции? Другими словами, Adapter устанавливает соответствие открытого интерфейса одного класса с открытым интерфейсом другого?

Энди: Да, это именно то, что я сказал.

Марсиа: Возможно, это то, что ты имел в виду, но это не то, что ты сказал! Но все равно, зачем мне это нужно? Только для того, чтобы избежать изменения существующего кода?

Энди: Это очень важная причина. Например, рефакторинг часто может приводить к значительным изменениям в определении интерфейса класса. Это, в свою очередь, может повлечь за собой значительные изменения кода приложения, обращающегося к службам такого класса. Подобные изменения всегда связаны с большим риском и, если это возможно, их нужно избегать.

Марсиа: Все обстоит хорошо, если исходный код доступен для вас. Вы можете просто сохранить эти методы в исходном интерфейсе этого класса, но удалить эту реализацию в новых методах. Тогда старые методы будут выступать в роли «контролирующих» методов и просто пересылать обращения к методам, выполняющим фактическую работу. Однако, необходимость изменения существующего кода остается. Чтобы избежать подобной модификации, просто со-

здайте новый класс, повторяющий исходный интерфейс. Теперь все, что вам необходимо, — это вызвать соответствующие методы объекта, основанном на этом новом классе.

Энди: Совершенно верно! И когда вы работаете с компонентами, для которых у вас нет исходного кода, это единственный способ работы с изменениями, без модификации вашего собственного кода.

Марсиа: Дай-ка я посмотрю, правильно ли я это поняла. Представим, что у меня есть класс управления данными, инициализирующийся формой, имеющей метод DoSave(). В качестве параметра метод принимает имя целевой таблицы. В моей форме может быть код, подобный приведенному ниже:

```
IF VARTYPE( ThisForm.oDataMgr ) # "O"
   ThisForm.oDataMgr = oFactory.New("DataManager" )
ENDIF
ThisForm.oDataMgr.DoSave( 'customers' )
```

Энди: Это выглядит убедительно. Теперь представь, что мы изменяем Data Manager так, что вместо метода DoSave() он генерирует метод DO, получающий на вход имя операции и один объект с параметрами. Тебе потребуется изменить код всех твоих форм следующим образом:

```
IF VARTYPE( ThisForm.oDataMgr ) # "O"
   ThisForm.oDataMgr = oFactory.New("DataManager" )
ENDIF
loParam = CREATEOBJECT("LINE" )
loParam.AddProperty('cTable', 'customers' )
llOK = ThisForm.oDatMgr.Do("SAVE", loParam )
```

Марсиа: Ах! Это не так уж весело, особенно, если у нас есть несколько методов, имена которых выглядят как DoXxxx(). Я могу оценить, где проще создать новый объект и позволить ему выполнить работу. Поскольку мы используем factory для создания наших объектов, мы можем просто изменить имя этого инициализируемого класса в classes.dbf (см. статью «Настоящая фабрика», FoxTalk, январь 2003 года). Вместо Data Manager он создает экземпляр нашего класса Adapter, где присутствует ожидаемый метод DoSave(), но этот метод теперь выглядит как показано ниже:

```
FUNCTION DoSave( tcTable )
IF VARTYPE( This.oDataManager ) # "O"
     This.oDataHandler = oFactory.New("DataHandler" )
ENDIF
loParam = CREATEOBJECT("LINE" )
loParam.AddProperty('cTable', 'customers' )
RETURN This.oDataHandler.Do("SAVE", loParam )
```

Энди: Совершенно верно. Конечно, в средах разработки, поддерживающих множественное наследование, Adapter выглядит намного проще. Вы создаете новый дочерний класс исходного класса и даете ему возможность наследовать интерфейс этого нового класса. В Visual FoxPro мы можем делать это только в том случае, если имеем дело с COM-объектами (для этого необходима библиотека типов), и даже тогда мы получим только интерфейс, а не реализацию.

Марсиа: Хорошо. Теперь, когда мы разобрались с Adapter, давай вернемся к Decorator (с которого я так или иначе начала). Назначением Decorator, как я себе это представляю, является добавление функциональных возможностей объекту, без фактического изменения какого-либо кода в этом объекте.

Энди: Это мне подходит. Гамма, Хельм и другие определили это как возможность «динамически добавлять дополнительные обязанности». А для «расширения функциональности Decorator предоставляет гибкую альтернативу построению дочерних классов».

Марсиа: Хорошо, теперь нам определенно ясна разница между ними. По определению, Adapter не может изменять функциональные возможности, он имеет отношение только к интерфейсам. Итак, раз уж мы начали заниматься изменением функциональности, мы преобразуем Decorator в новый шаблон.

Энди: Я могу представить себе два сценария, для которых подходит Decorator. В первом из них исходный код для данного объекта просто недоступен (возможно, объект является элементом ActiveX или используется класс библиотеки стороннего производителя). Во втором данный класс широко применяется, но расширенные функциональные возможности являются исключением, и крайне нецелесообразно добавлять код в исходный класс. Вместо того, чтобы создавать специализированный дочерний класс, вы можете использовать Decorator в этих особых случаях.

Марсиа: Давай обратимся к тому же примеру Data Manager, что мы использовали ранее. В моих существующих формах содержится следующий код:

```
IF VARTYPE( ThisForm.oDataMgr ) # "O"
   ThisForm.oDataMgr = oFactory.New("DataManager" )
ENDIF
ThisForm.oDataMgr.DoSave('customers' )
```

Но сейчас мне требуется новая форма, в которой будет выполняться некоторая проверка достоверности и, если она успешна, всего лишь вызываться метод DoSave(). Я могу добавить необходимый код прямо в эту форму, но она, на самом деле, не принадлежит уровню пользовательского интерфейса. В качестве альтернативы я могу модифицировать этот Data Manager и добавить метод BeforeSave(), а также вызывать его явно или он будет автоматичес-

ки вызываться из метода DoSave(). Однако, на самом деле, я не хочу модифицировать существующий производственный код, поскольку риск внесения ошибок очень высок. Итак, я должна использовать Decorator?

Энди: Да. Просто создай новый класс, который имеет метод DoSave(), вызывающий метод BeforeSave() и, при необходимости, передающий этот вызов Data Manager. Твоя новая форма просто инициализирует этот класс вместо Data Manager и вызывает его метод DoSave() следующим образом:

```
IF VARTYPE( ThisForm.oDataMgr ) # "O"
   ThisForm.oDataMgr = oFactory.New("DataDecorator" )
ENDIF

ThisForm.oDataMgr.DoSave('customers' )
```

Теперь Decorator предоставляет улучшенный метод DoSave(), а также сохраняет ссылку на Data Manager:

```
FUNCTION DoSave( tcTable )
 IF This.BeforeSave( tcTable )
     llOk = This.oDataMgr.DoSave( tcTable )
 ELSE
     llOk = .F.
 ENDIF
 RETURN llOk
```

Марсиа: Это все проясняет. Adapter и Decorator, несмотря на то, что реализованы сходным способом, совершенно различны по назначению. Итак, сейчас мы возвращаемся к вопросу об имени Wrapper. Ясно, что он не может служить синонимом для обоих шаблонов, поскольку они не являются тождественными. Является ли он простым шаблоном или шаблоном более высокого уровня, для которого и Adapter, и Decorator просто являются примерами?

Энди: Должен сказать, что этот шаблон определенно не является шаблоном более высокого уровня. Причина заключается в том, что слово «wrapper» (обертка) предполагает покрытие, поэтому описывать Decorator и Adapter, как «Wrappers», ошибочно даже лингвистически. Оба эти шаблона рассчитаны на «обман» клиентов, заставляя их поверить, что добавленный объект является желаемой целью, делая его абсолютно похожим на целевой объект. Следовательно, они могут быть названы либо «mimic» (подражатель), либо «impersonator» (самозванец), но никак ни Wrapper.

Марсиа: Я согласна. Итак, можем ли мы считать его полноправным шаблоном? Что, если мы будем придерживаться мнения (но это чисто умозрительное заключение), что целью шаблона Wrapper является управление его содержанием? И как часть этой общей функциональности он может (но не обязательно) включать функции, предоставляемые вместе или по отдельности шаблонами Decorator или Adapter. То, о чем я думаю, это сценарий, интегрирующий сущность, не являющуюся объектом в объектно-ориентированной среде, и дающий возможность обращаться к ней, как если бы она на самом деле была объектом.

Энди: Возможно ты права, но я не уверен, что понимаю тебя. Пожалуйста, приведи какой-нибудь пример.

Марсиа: Представь себе, что ты хочешь получить доступ к некоторой функциональной возможности, предоставляемой в виде библиотеки .DLL. Ты можешь создать некоторый класс, предоставляющий основные управляющие функции (такие, как проверка того, что эта .DLL доступна и надлежащим образом зарегистрирована), а также методы, имеющие доступ к реальным функциям .DLL. Это, я думаю, и будет «Wrapper».

Энди: Интересно! В этом я согласен с тобой.

Марсиа: Так и запишем!

Энди: Ключом для понимания, как всегда, является назначение. Для шаблона Wrapper идея «управления» объектом, который «обертывается», является наиболее значимым фактором. По сути, как мне кажется, большинство из классов, рассматриваемых нами и упоминаемых как «менеджер», на самом деле являются оболочкой. Например, Form Manager, создающий и удаляющий формы, управляющий наборами форм и обеспечивающий интерфейс для доступа к запущенным формам, на самом деле является оболочкой.

Марсиа: Ты имеешь в виду, что когда он заключает в оболочку объекты, он является «manager», а когда нет — «wrapper»?

Энди: Совершенно верно! И в заключение, давайте посмотрим, что же заключено в имени (см. таблицу 1 для сопоставления этих терминов).

Таблица 1. Сравнение Adapter, Decorator и Wrapper.

Шаблон	Назначение
Adapter	Модифицирует интерфейс без изменения существующего поведения.
Decorator	Модифицирует поведение без изменения существующего интерфейса.
Wrapper	Обеспечивает интерфейс и службы для поведения, определенного где-то в другом месте.

❑❑❑

Что делает пробел в этом имени?

Эдвард МакДермотт (Edward McDermott)

Использование пробелов в именах таблиц не было привычным для Эдварда Макдермотта. Однако он приспособился и научился работать с ними. В этой статье он показывает, как с этим справился.

Некоторые используют пробел на пробеле в именах таблиц! Признаюсь, что я консервативен, потому до недавнего времени не использовал пробелов в именах таблиц. Я использовал контейнеры баз данных, Windows 95/98/ME/2000 и другие возможности, но таблицы с пробелами в имени — никогда.

Однако, чтобы писать универсальные процедуры, мне необходим способ, работающий во всех возможных ситуациях, поэтому мне пришлось разобраться, как работать с раздражающими меня пробелами. Сегодня проблема заключается не в длине имени, а в наличии в нем пробелов. После того как Microsoft ввел длинные имена файлов, имена могут включать в себя и пробелы. Это означает, что я могу встретиться с пробелами в: 1) указаниях пути; 2) именах контейнеров баз данных, файлов и представлений.

Я не найду пробелы ни в имени поля, ни в индексном тэге, поэтому я не беспокоюсь о полях и тэгах. Однако я буду говорить о командах, предназначенных для того, чтобы открывать, выбирать и модифицировать контейнеры баз данных, таблицы, а также SQL-операторы. Кроме того, я должен рассмотреть ситуации, в которых могу делать как явные, так и косвенные ссылки.

Открытие явно указанной базы данных

Предположим, что вы хотите открыть базу данных Contacts.DBC. Вы просто пишите следующий код:

```
OPEN DATABASE C:\foxstuff\usage\Data\contact.dbc
```

Однако, если имя базы данных: “Students and classes.dbc”, то вы должны написать оператор, беря имя базы данных в кавычки, так, чтобы FoxPro воспринимала его как единую строку, включающую пробелы и символы после пробелов как часть этого имени. Это также справедливо в том случае, если в имени базы данных нет пробела, но он есть в имени какого-либо каталога в этом пути.

```
OPEN DATABASE ;
   "C:\foxstuff\usage\Data\Students and Classes.dbc"
```

Если вы использовали команду DBC(), чтобы увидеть имя текущей открытой базы данных, то увидите “C:\foxstuff\usage\Data\Students and Classes.dbc.” Вы обратили внимание на пробелы в строке, возвращенной этой функцией?

Открытие явно указанной таблицы

Предположим, что вы хотите открыть таблицу CLASSES.DBF. Вы просто пишите код:

```
USE C:\foxstuff\usage\Data\classes.dbf IN 0 SHARED
```

Однако, когда имя таблицы следующее: “test blank table.dbf”, то в этот путь снова входят пробелы. Вам опять необходимо заключить в кавычки имя таблицы, так, чтобы FoxPro воспринимала всю строку как имя. Результат будет выглядеть так:

```
USE "C:\foxstuff\usage\data\test blank table.dbf" ;
   IN 0 SHARED
```

Как будто все просто. Однако посмотрите на ваш сеанс данных. Имя, данное FoxPro таблице (на самом деле являющееся псевдонимом), не содержит никаких пробелов. Имена файла и псевдонима для таблицы, чье имя содержит пробелы, не совпадают. Пробелы были заменены на символы подчеркивания.

Построение псевдонима для файла из его имени

Я всегда использую псевдоним для файла, поэтому мне нужен способ создания псевдонима из имени файла. Следующий оператор будет выполнять требуемое преобразование:

```
Alias = STRTRAN(JUSTSTEM( ;
   "C:\foxstuff\usage\data\test blank table.dbf"), ;
   " ","_"))
```

Команда STRTRAN заменяет все пробелы символами подчеркивания. JUSTSTEM возвращает только заключительную часть имени файла, убирая путь и суффикс. Да, я могу назначить свой собственный псевдоним при открытии файла, но тогда эту информацию пришлось бы держать в голове. Предыдущее преобразование работает для всех таблиц, независимо от того, имеют ли они в своем имени пробелы или нет.

Открытие базы данных через косвенное определение

Представим, что вы хотите открыть базу данных, чье имя хранится в строке tcDBCfile. Обычно вы можете использовать именованное выражение или макроподстановку. Вы просто пишите следующий код:

```
tcdbcfile = "C:\foxstuff\usage\Data\contact.dbc"
OPEN DATABASE &tcdbcfile
OPEN DATABASE (tcdbcfile)
```

Оба оператора OPEN работают. Однако, стоит вам поменять это имя на другое, содержащее пробелы, как будет работать только именованное выражение. Приведенный ниже фрагмент кода иллюстрирует это:

```
tcdbcfile = ;
   "C:\foxstuff\usage\Data\Students and Classes.dbc"
 OPEN DATABASE &tcdbcfile     && Не работает
 OPEN DATABASE (tcdbcfile)    && Работает
```

Можно попытаться найти некоторый способ заставить работать и макроподстановку, но именованное выражение работает значительно быстрее.

Открытие таблицы через косвенное определение

То же происходит и при открытии таблицы через косвенное определение имени файла:

```
tctestfile = "C:\foxstuff\usage\Data\test blank table.dbf"
OPEN DATABASE &tctestfile    && Не работает
OPEN DATABASE (tctestfile)   && Работает
```

Однако жизнь становится гораздо интереснее при закрытии файла:

```
USE IN (tctestfile)                 && Не работает
USE IN (STRTRAN(JUSTSTEM(tctestfile)," ","_")) && Работает
```

Запомните, что опция IN требует рабочей области или псевдонима. Псевдонимом для “test blank table” будет “test_blank_table”, поэтому вы должны преобразовать пробелы в символы подчеркивания.

Три простых правила

Я вывел из всей приведенной выше информации три простых правила: 1) Всегда заключайте прямые ссылки на контейнеры базы данных и таблицы в разграничители. 2) Всегда используйте именованные выражения вместо макроподстановки для открытия контейнеров базы данных или таблиц. 3) Всегда используйте (STRTRAN(JUSTSTEM(filename)," ","_")) чтобы получить псевдоним таблицы из ее имени.

SQL и имена с пробелами

Другая половина проблемы «пробелов» относится к SQL-операторам. Используете ли вы SQL-операторы в виде сохраненных запросов, представлений или SQL-операторов, прописанных непосредственно в вашей программе, вы обнаружите, что пробелы в имени базы данных или таблицы оказывают сильное влияние на их работу.

Конструктор запросов (query builder) в FoxPro обрабатывает пробелы в контейнерах базы данных и таблицах корректно, добавляя строковые разграничители там, где требуется. Если вы строите ваши SQL-операторы при помощи конструктора запросов, то, возможно, даже не заметите, что иногда он добавляет кавычки, а иногда нет. Давайте проверим несколько SQL-операторов. После того как я открыл базу данных, я попытался сделать следующее:

```
* работающая абсолютная ссылка
 SELECT * from 'test blank table'

 * косвенная ссылка на имя таблицы
 lcfile = 'test blank table'
 SELECT * from (lcfile)              && Работает
 SELECT * from &lcfile               && Не работает
```

SQL и имена псевдонимов таблицы

В предыдущем примере SQL-операторы ведут себя так же, как VFP-операторы. Однако все эти примеры гораздо проще, чем большинство SQL-операторов, конструируемых и используемых вами. Обычно SQL-оператор в запросе или представлении включает в себя критерий выбора и условия соединения. В обоих случаях операторы могут требовать, чтобы вы ссылались на некоторое поле по его имени. Если в вашем SQL-операторе больше одной таблицы, и такое же имя поля используется не в одной таблице, вы должны делать абсолютную ссылку на это поле. Абсолютная ссылка на поле представляет собой псевдоним таблицы, за которым следует точка (“.”), после которой идет имя поля. Должны ли вы использовать строковые разграничители вокруг имени таблицы или вокруг имени поля? Ни в коем случае. (Для таблиц, включенных в БД, вы можете присваивать полям альтернативные имена (Caption), которые могут быть с пробелами. В дальнейшем можно использовать Caption в составе команды SELECT, и вот тут вам понадобятся кавычки — *прим. ред.*)

Вам необходимо определить псевдоним для имени таблицы, используя ключевое слово AS, для образования локального псевдонима таблицы, применяемого ко всему SQL-оператору. Например, если вы хотите сделать ссылку на поле first в таблице “test blank table”, вы можете использовать следующий оператор:

```
SELECT l_alias.first from 'test blank table' AS l_alias
```

Строка “l_alias”, следующая за ключевым словом AS, является локальным псевдонимом. Очевидно,

что вы не должны включать никаких пробелов в этот локальный псевдоним. Это заманит вас в ловушку, которую вы только что избежали. Вы можете использовать любую строку для локального псевдонима.

Построение сложных SQL-операторов

Часто я хочу выполнить более сложный SQL-оператор, динамически приспосабливаемый к определенным ситуациям. Для того, чтобы обеспечить себе полный контроль, я встраиваю SQL-оператор в строковую переменную, а затем выполняю эту строку:

```
sqlstr = "select * from 'test blank table'"
&sqlstr
```

SQL-оператор, встроенный в строку, является обычным SQL-оператором, следующим всем правилам, уже описанным мною. Единственная проблема, с которой я столкнулся, это поддержка моих строковых разграничителей. Иногда я также использую для разграничения имен таблиц квадратные скобки, чтобы сделать оператор более читаемым. Вот результат:

```
sqlstr = "select * from [test blank table]"
&sqlstr
```

Функция для упаковки всех таблиц в базе данных

Моя мама всегда говорила мне, что рецепт проверяется во время обеда. Применение этого утверждения для проблем, затронутых в этой статье, означает, что я должен продемонстрировать сказанное на примере некой универсальной функции. Ниже приведен код для функции, открывающей базу данных (определяемую переданным параметром), а затем проходящей через все таблицы, упаковывая каждую из них.

```
* Процедура открывает базу данных, определенную параметром,
* а затем сжимает каждый из ее файлов
 PROCEDURE pack_dbc
 LPARAMETERS tcdbcfile
 LOCAL lnItem, lnTables, latable(1), lcName, lcAlias
 LOCAL lcsaveexclusive, lcdatabase
 * Сохраняет настройку Exclusive и устанавливает
 * монопольный доступ к таблицам
 lcsaveexclusive = SET("EXCLUSIVE")
 SET EXCLUSIVE ON
 * Сохраняет текущую базу данных для восстановления
 lcdatabase = DBC()
 CLOSE DATABASES ALL
 * Открывает базу данных для сжатия
 OPEN DATABASE (tcdbcfile) EXCLUSIVE
 * Получает список таблиц базы данных
 lnTables = ADBOBJECTS(latable,"TABLE")
 * Открывает, сжимает и закрывает каждую таблицу
 FOR lnItem = 1 TO lnTables
     STORE latable[lnItem] TO lcName, lcAlias
     * Необходимо для таблиц, содержащих пробелы в именах
     lcAlias = STRTRAN(JUSTSTEM(lcAlias)," ","_")
     IF NOT USED(lcAlias)
         USE (lcName) IN 0 ALIAS (lcAlias)
         SELECT (lcAlias)
         WAIT WINDOW NOWAIT ;
           "Packing the " + lcName + " table"
         PACK
         USE IN (lcAlias)
     ENDIF
 NEXT lnItem
  * Восстанавливает настройки, включая текущую базу данных
 WAIT CLEAR
 SET EXCLUSIVE &lcsaveexclusive
 IF !EMPTY(lcdatabase)
     IF DBC() <> lcdatabase
         CLOSE DATABASE
         SET DATABASE TO (lcdatabase)
     ENDIF
 ELSE
     CLOSE DATABASE
 ENDIF
 WAIT WINDOW "Packing completed" TIMEOUT 5
 RETURN
```

❑❑❑

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 20/07/03. Формат 60x90 1/8. Тираж 300 экз.