FoxTalk

Сентябрь 2003
№ 9 (75)

русское издание

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

Переделайте свои курсоры в CursorAdapter

Дуг Хенниг (Doug Hennig)

Класс CursorAdapter является одной из наиболее важных новых функциональных возможностей, представленных в версии VFP 8, поскольку он обеспечивает простой в использовании унифицированный интерфейс для доступа к удаленным данным. В этом месяце Дуг Хенниг изучает класс CursorAdapter и то, как он работает. В следующем месяце мы рассмотрим некоторые возможности эффективного применения этого класса.

Все больше и больше VFP-разработчиков хранят свои данные не в VFP-таблицах, а где-то еще, например, в БД SQL Server или Oracle. Для этого имеется много причин, включая «хрупкость» VFP-таблиц (как воображаемую, так и реальную), безопасность, размер базы данных и корпоративные стандарты. С появлением каждой новой версии фирма Microsoft упрощает доступ к не-VFP-данным и даже поощряет использование таких данных, поставляя на одном компакт-диске с версией VFP 7 MSDE (Microsoft Data Engine, бесплатная версия SQL Server с ограниченными возможностями).

Впрочем, доступ к серверной базе данных никогда не был столь же простым делом, что и использование VFP-таблиц, и для его осуществления может быть задействовано пугающее разнообразие механизмов:

- Удаленные представления (remote views), которые базируются на ODBC-соединениях.
- Функции сквозных SQL-запросов (SQL Pass Through — SPT), такие как SQLCONNECT(), SQLEXEC() и SQLDISCONNECT(), которые также базируются на ODBC-соединениях.
- Объектная модель ActiveX Data Objects, или ADO, которая предоставляет объектно-ориентированного клиента (front end) для OLE DB-провайдеров различных СУБД.
- Спецификация XML, которая является простым, независимым от платформы, механизмом пересылки данных.

Если вы потратили некоторое время на работу с этими средствами обмена данными, то среди прочего, вероятно, обратили внимание на то, что каждый из них совершенно не похож на другие. Это означает, что каждый из них потребует от вас дополнительных усилий на его изучение, а перевод готового приложения с использова-

Сентябрь 2003

материал имеет отношение к соответствующей версии

материал имеет отношение к соответствующей платформе

исходные тексты программ можно скачать из Интернета

ния одного механизма на использование другого является нетривиальной задачей.

По сравнению с предыдущими версиями, версия 8 существенно облегчает доступ к удаленным данным, поскольку в ней имеется новый базовый класс: CursorAdapter. По-моему, класс CursorAdapter — это одна из наиболее значимых новых функциональных возможностей, реализованных в версии VFP 8. Я считаю, что эта возможность является столь замечательной по следующим причинам:

- Класс CursorAdapter облегчает использование технологий ODBC, ADO или XML, даже если вы не слишком хорошо с ними знакомы.
- Он обеспечивает унифицированный интерфейс для доступа к удаленным данным, независимо от выбранного вами механизма обмена данными.
- Он облегчает переход от одного механизма обмена данными к другому.

Вот пример для иллюстрации последней из перечисленных причин. Предположим, что у вас есть приложение, которое использует технологию ODBC совместно с классом CursorAdapter для организации доступа к данным, хранящимся на SQL Server, и по какой-то причине вы хотите внести изменение в это решение и использовать взамен ODBC объектную модель ADO. Вы должны всего лишь изменить значение свойства DataSourceType класса CursorAdapter и определить соединение с серверной базой данных, и ваша цель достигнута. Остальные компоненты этого приложения ничего не знают о внесенном изменении, и это их не волнует; они по-прежнему видят все тот же курсор, независимо от того, какой механизм был использован для обращения к данным.

Свойства, события, методы

Давайте начнем наше изучение класса CursorAdapter с того, что рассмотрим его свойства, события и методы (PEM). Мы не будем рассматривать здесь все PEM, но только самые важные. Список всех свойств, событий и методов ищите в документации для версии VFP 8.

DataSourceType

Это свойство является «важной персоной»: оно определяет функциональное поведение класса и то, какого рода значения получают некоторые другие свойства. Допустимыми вариантами являются “Native”, что указывает на использование встроенных таблиц, или “ODBC”, “ADO” или “XML”, что означает использование для обращения к данным соответствующего механизма.

DataSource

Это свойство определяет средство доступа к данным. VFP игнорирует это свойство, если свойство DataSourceType имеет значение “Native” или “XML”. Для использования технологии ODBC, укажите в качестве значения свойства DataSource допустимое значение ссылки на ODBC-соединения (подразумевается, что соединение вы должны организовать сами). В случае использования объектной модели ADO значением свойства DataSource должен быть набор записей RecordSet, чей объект ActiveConnection указывает на открытый объект Connection модели ADO (еще раз, вы сами должны организовать все это).

UseDEDataSource

Если это свойство имеет значение .T. (по умолчанию оно имеет значение .F.), свойства DataSourceType и DataSource вы можете оставить в покое, поскольку вместо них класс CursorAdapter будет использовать свойства класса DataEnvironment. (В версии VFP 8 класс DataEnvironment также получил новые свойства: DataSourceType и DataSource.) Примером того случая, когда вам следует указать для свойства UseDEDataSource значение .T., является ситуация, в которой необходимо, чтобы все объекты класса CursorAdapter в среде окружения DataEnvironment использовали одно и то же ODBC-соединение.

SelectCmd

Во всех случаях, кроме случая использования спецификации XML, значением этого свойства является команда SQL-SELECT, которая используется для выборки данных. Если используется XML, значением этого свойства может быть допустимая XML-строка, которая может быть преобразована в курсор (путем использования внутреннего обращения к функции XMLTOCURSOR()), или выражение (например, UDF-выражение), которое возвращает допустимую XML-строку.

CursorSchema

В этом свойстве хранится структура курсора в таком же точно формате, который вы использовали бы в команде CREATE CURSOR (имеется в виду все то, что заключено в круглые скобки в этой команде). Вот пример: CUST_ID C(6), COMPANY C(30), CONTACT C(30), CITY C(25). Хотя можно оставить это свойство пустым и отдать классу CursorAdapter распоряжение о том, чтобы он определил структуру при создании курсора, будет лучше, если свойство CursorSchema заполните вы. Во-первых,

если свойство CursorSchema является пустым или имеет некорректное значение, вы либо получите сообщения об ошибках при открытии среды DataEnvironment формы, либо не сможете выполнять применительно к полям из CursorAdapter операцию «перетащить и бросить» их в форму с целью создания элементов управления. К счастью, построитель CursorAdapter Builder, который поставляется в составе VFP, может автоматически заполнить это свойство вместо вас.

AllowDelete, AllowInsert, AllowUpdate и SendUpdates

Эти свойства, которые по умолчанию имеют значение .T., определяют, могут ли выполняться операции удаления, вставки и обновления, и пересылаются ли внесенные изменения в источник данных.

KeyFieldList, Tables, UpdatableFieldList и UpdateNameList

Эти свойства, которые служат той же самой цели, что и имеющие идентичные имена свойства представлений, получающие свои значения в результате применения функции CURSORSETPROP(), необходимы в том случае, если вы хотите, чтобы VFP автоматически обновляла источник данных, пересылая в него те изменения, которые были внесены в курсор. Свойство KeyFieldList — это разделенный запятыми список полей (без указания псевдонимов), которые формируют первичный ключ для курсора. Свойство Tables — это разделенный запятыми список таблиц. Свойство UpdatableFieldList — это разделенный запятыми список тех полей (без указания псевдонимов), которые могут обновляться. Свойство UpdateNameList — это разделенный запятыми список, в котором имена полей курсора соотносятся с именами полей таблицы. Свойство UpdateNameList имеет следующий формат: CURSORFIELDNAME1 TABLE.FIELDNAME1, CURSORFIELDNAME2 TABLE.FIELDNAME2 и так далее. Обратите внимание на то, что даже если свойство UpdatableFieldList не содержит имени первичного ключа таблицы (потому что вы не хотите, чтобы это поле обновлялось), это имя все-таки должно присутствовать в списке UpdateNameList, иначе обновления выполняться не будут.

**Cmd, *CmdDataSource и *CmdDataSourceType*

Если вы хотите с большей степенью детализации управлять тем, как VFP удаляет, вставляет или обновляет записи в источнике данных, вы можете присвоить соответствующие значения этому набору свойств — замените символ “*” в именах этих свойств на Delete, Insert и Update.

CursorFill (UseCursorSchema, NoData, Options, Source)

Этот метод создает курсор и заполняет его данными из источника данных (впрочем, вы можете передать значение .T. в качестве параметра NoData, чтобы создать пустой курсор). Передайте значение .T. в качестве первого параметра, чтобы использовать схему, определенную в свойстве CursorSchema, или значение .F., чтобы создать соответствующую структуру из источника данных (по-моему, такое функциональное поведение выглядит задом наперед). Должна быть включена (On) установка MULTILOCKS или же этот метод не будет работать. Если метод CursorFill по какой-то причине потерпит неудачу, то он не выдает сообщения об ошибке, но возвращает значение .F.; для выяснения того, что произошло, используйте функцию AERROR() (приготовьтесь, впрочем, к тому, что придется «вести раскопки», поскольку получаемые вами сообщения об ошибках зачастую недостаточно конкретны, чтобы поведать о том, в чем именно заключается проблема).

CursorRefresh()

Этот метод аналогичен функции REQUERY(): он обновляет содержимое курсора.

Before() и After*()*

Почти все методы и события имеют свои инициируемые «до» и «после» их наступления события-«перехватчики», позволяющие вам настроить по своему усмотрению функциональное поведение класса CursorAdapter. Например, в коде события AfterCursorFill вы можете создать для курсора индексы с тем, чтобы они всегда были доступны. В случае наступления событий Before вы можете вернуть значение .F., чтобы воспрепятствовать действию, которое предусмотрено при наступлении соответствующего события (аналогично событиям контейнера базы данных).

Пример

Вот пример (программа CursorAdapterExample.prg находится на дискете), который извлекает конкретные поля записей о покупателях из Бразилии из таблицы Customers базы данных Northwind, поставляемой вместе с SQL Server. Этот курсор является обновляемым, поэтому если вы внесли в него изменения, закройте его, а затем исполните программу еще раз, — вы увидите, что ваши изменения были сохранены на сервере.

```
local lcConnString, ;
   loCursor as CursorAdapter, ;
   laErrors[1]
```

```
lcConnString = 'driver=SQL Server;server=(local);' + ;
  'database=Northwind;uid=sa;pwd=;trusted_connection=no'
* измените пароль на тот, который используется
* для доступа к вашей базе данных
loCursor = createobject('CursorAdapter')
with loCursor
  .Alias               = 'Customers'
  .DataSourceType      = 'ODBC'
  .DataSource          = sqlstringconnect(lcConnString)
  .SelectCmd           = "select CUSTOMERID, " + ;
    "COMPANYNAME, CONTACTNAME from CUSTOMERS " + ;
    "where COUNTRY = 'Brazil'"
  .KeyFieldList        = 'CUSTOMERID'
  .Tables              = 'CUSTOMERS'
  .UpdatableFieldList = 'CUSTOMERID, COMPANYNAME, ' + ;
    'CONTACTNAME'
  .UpdateNameList      = ;
    'CUSTOMERID CUSTOMERS.CUSTOMERID, ' + ;
    'COMPANYNAME CUSTOMERS.COMPANYNAME, ' + ;
    'CONTACTNAME CUSTOMERS.CONTACTNAME'
  if .CursorFill()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif .CursorFill()
endwith
```

Изменения, внесенные в классы DataEnvironment и Form

Для обеспечения поддержки нового класса CursorAdapter некоторые изменения были внесены в классы DataEnvironment и Form и в их конструкторы.

Прежде всего, как я упомянул ранее, класс DataEnvironment обладает теперь свойствами DataSource и DataSourceType. Он не использует эти свойства самостоятельно, но они используются всеми членами класса CursorAdapter, у которых свойство UseDEDataSource имеет значение .T.. Во-вторых, теперь вы можете визуально создавать подклассы класса DataEnvironment с помощью конструктора Class Designer (вот это да!).

Что касается форм, то теперь вы можете определить используемый подкласс класса DataEnvironment, указывая значения новых свойств DEClass и DEClassLibrary. Если вы поступаете таким образом, то все, что вами было создано в существующей среде DataEnvironment (курсоры, программный код, и так далее) будет потеряно, но, по крайней мере, вас сначала предупредят об этом. Со всем сказанным до некоторой степени связана прекрасная новая возможность, появившаяся у форм, в виде свойства BindControls — присваивание этому свойству значения .F. в окне Property означает, что VFP не будет пытаться выполнить связывание элементов управления с данными во время инициализации этих элементов управления, а сделает это только тогда, когда вы укажете для свойства BindControls значение .T.. Что в этом хорошего? А как часто вы мучились от того, что параметры передаются в метод Init, который срабатывает после того, как все элементы управления будут инициализированы и привязаны к своим источникам данных ControlSource? Что, если вы хотите передать в форму параметр, который сообщает ей о том, какую таблицу необходимо открыть, или иные сведения, влияющие на источники данных ControlSource? Упомянутое новое свойство мгновенно разрешает эту проблему.

Преимущества

Имеется множество серьезных причин для того, чтобы использовать вместо удаленных представлений, SPT-запросов, модели ADO или спецификации XML класс CursorAdapter:

- Каждый из перечисленных механизмов доступа к данным имеет свой собственный интерфейс. Используя удаленные представления, вы открываете их с помощью команды USE. Применяя SPT-запросы, вы используете для создания курсора функции SQLCONNECT() и SQLEXEC(). Используя модель ADO, вы создаете экземпляры ADO-объектов Connection и RecordSet (и возможно объект Command, в зависимости от того, как вы извлекаете данные) и обращаетесь к их методам Open. Используя технологию XML, вы сначала откуда-то получаете XML-строку (например, от COM-компонента в n-уровневом приложении или от SQL Server посредством SQLXML), затем используете функцию XMLTOCURSOR() для преобразования полученной строки в VFP-курсор. Возврат обновлений на сервер (back end) также имеет свои отличия для каждого из этих механизмов. Следовательно, по мере того как вы переходите от одного способа к другому, вы должны изучать новые методики, плюс вам, возможно, придется переписать массу готового программного кода.

  Класс CursorAdapter имеет унифицированный интерфейс независимо от того, какой механизм вы используете для извлечения данных. Вы определяете значения различных свойств объекта и обращаетесь к методу CursorFill для извлечения данных, работаете с курсором так, как вы работали бы с любым другим VFP-курсором, а затем обращаетесь к функции TABLEUPDATE(), чтобы переслать изменения обратно (или позволяете VFP обработать эту задачу в автоматическом режиме).
- В среде разработки вам часто необходимо открыть курсор в окне Command Window чтобы извлечь его содержимое. Это легко сделать для удаленных представлений (просто выполнить команду USE применительно к этому представлению), но требует гораздо больших усилий в случае использования технологий SPT, ADO и XML.

В зависимости от настроек (например, если он совершенно независим), открытие курсора из подкласса CursorAdapter может быть почти столь же легким делом, как и открытие удаленного представления: вы просто создаете экземпляр объекта этого подкласса и вызываете метод CursorFill. Вы даже могли бы вызвать этот метод из метода Init, чтобы сделать эту операцию одношаговой.

- Перенос готового приложения может оказаться сравнительно легким делом в случае удаленных представлений: вы заменяете имеющиеся объекты Cursor, которые указывают на локальные таблицы или представления в среде окружения DataEnvironment, на те, которые ссылаются на удаленные представления. Используя технологии SPT, ADO и XML, вы должны перестроить всю свою схему доступа к данным.

  Перевод готового приложения на использование класса CursorAdapter столь же прост, как и в случае использования удаленных представлений — вы просто заменяете объекты Cursor на объекты CursorAdapter.
- С удаленными представлениями легко работать в конструкторах Form и Report Designer. Вы можете добавить удаленное представление в среду окружения данных DataEnvironment и затем воспользоваться преимуществом визуальной поддержки, обеспечиваемой средой DataEnvironment: «перетаскивание и бросание» полей в автоматически создаваемые элементы управления, простое связывание элемента управления с полем путем их выбора из комбинированного списка в окне Properties и так далее. Технологии SPT, ADO и XML не имеют такой визуальной поддержки.

  Класс CursorAdapter имеет такую же поддержку в среде DataEnvironment, какую имеют удаленные представления.
- Удаленные представления легко создать, используя конструктор View Designer. Хотя в прошлом этот инструмент имел серьезные ограничения, особенно при работе с представлениями, объединяющими больше двух таблиц, в версии VFP 8 эти проблемы в основном решены и добавлена масса новых возможностей, например, двунаправленное редактирование: вы можете изменить код в окне SQL и увидеть отображение этих изменений в визуальной части конструктора View Designer. При использовании технологий SPT, ADO и XML объем работы возрастает, поскольку вы должны программировать абсолютно все: создание и закрытие соединения, исполняемые предложения SQL-SELECT и так далее.

  Версия VFP 8 включает построитель CursorAdapter Builder, который быстро справляется с настройкой свойств, необходимых для извлечения и обновления данных. Он даже включает построитель SelectCmd, который, используя элемент управления mover, хотя и не столь наглядно, как конструктор View Designer, позволяет вам выбрать те поля, которые должны быть извлечены из удаленных таблиц.
- Нетрудно обновить серверную базу данных, переслав в нее изменения, внесенные в удаленные представления, и наборы записей RecordSet модели ADO. Исходя из того, что свойства представления были определены надлежащим образом, вы просто вызываете функцию TABLEUPDATE(). В случае модели ADO вы обращаетесь к методам RecordSet.Update() или UpdateBatch(). Используя SPT-запросы и спецификацию XML, вы имеете массу ручной работы для того, чтобы вернуть обновления обратно в источник данных.

  Как мы видели ранее, выполнение обновлений в случае использования класса CursorAdapter может быть настолько же простым, насколько просто определить значения для нескольких свойств, чтобы заставить VFP проделать за вас всю работу, или же вы можете добиться большей гибкости, указывая, как обрабатываются все операции удаления, вставки и обновления.
- Поскольку результирующий набор, созданный удаленными представлениями и SPT-запросами, — это VFP-курсор, он может быть использован в VFP где угодно: в элементе управления grid, в отчете, обрабатываться в цикле SCAN и так далее. Данные, представленные средствами моделей ADO и XML, с другой стороны, должны быть преобразованы в курсоры прежде, чем их можно будет использовать таким же образом, что усложняет ваше приложение и требует времени на дополнительную обработку.

  Результирующий набор класса CursorAdapter — это VFP-курсор, поэтому он имеет те же самые преимущества, что и удаленные представления и SPT-запросы. Еще лучше, вы получаете VFP-курсор даже в том случае, если источником данных являются объекты моделей ADO и XML, потому что класс CursorAdapter автоматически заботится вместо вас о конвертации «в» и «из» курсора.
- Поскольку предложение SQL-SELECT для получения удаленного представления является предопределенным, вы не можете изменить его «на лету». Хотя это замечательно с точки зрения ти-

пичной формы ввода данных, с точки зрения запросов и отчетов это может оказаться проблемой. Вам, возможно, придется создать из одного и того же набора данных несколько представлений, каждое из которых отличается выбираемыми полями, структурой предложения WHERE и так далее. Это не является проблемой в случае использования технологий SPT, ADO и XML.

Класс CursorAdapter не страдает от упомянутой проблемы — вы можете легко изменить свойство SelectCmd, чтобы заменить данные и способ их выборки.

- Вы не можете обращаться к хранимой процедуре из удаленного представления, поскольку удаленное представление нуждается в прямом доступе к используемым таблицам. Это может явиться проблемой для администраторов базы данных вашего приложения — некоторые DBA полагают, что в целях обеспечения безопасности и по иным причинам доступ к данным должен осуществляться только через хранимые процедуры. Кроме того, поскольку они откомпилированы на сервере, хранимые процедуры часто выполняются значительно быстрее, чем предложения SQL-SELECT. Работая с технологиями SPT, ADO и XML, вы можете при необходимости обращаться к хранимым процедурам.

  Класс CursorAdapter в случае необходимости также может использовать хранимые процедуры, вы достигаете этого, настроив только свойство SelectCmd.
- Удаленные представления существуют в контейнере базы данных, следовательно, это еще один набор файлов, который вы должны поддерживать и инсталлировать в клиентскую систему. Кроме того, когда вы открываете представление, VFP пытается блокировать принадлежащие представлению записи в контейнере DBC, хотя бы и на краткое время. Это может вызвать конфликт в приложениях с большой интенсивностью использования, когда несколько пользователей могут попытаться открыть форму одновременно. Хотя существуют обходные пути решения этой проблемы (копирование контейнера DBC на локальную рабочую станцию и использование этой копии или, в версии VFP 7 и более поздних, использование команды SET REPROCESS SYSTEM для увеличения лимита времени, отводимого на разрешение конфликта блокировки), подобная ситуация представляет собой нечто такое, что вам придется запланировать. Еще одна дополнительная проблема: если вы используете представление SELECT * для извлечения всех полей из конкретной таблицы, а структура этой таблицы претерпевает изменения на сервере, представление оказывается неправильным и должно быть сформировано заново. Поскольку они не имеют ничего общего с контейнерами баз данных, ничто из сказанного не является вопросом для технологий SPT, ADO и XML.

  По причине того, что класс CursorAdapter не существует в контейнере базы данных, он также не имеет этих проблем.
- Поскольку удаленные представления и SPT-запросы работают только с технологией ODBC, они усердно трудятся в модели «клиент-сервер», использующей прямые подключения к данным. Модели ADO и XML являются предпочтительными механизмами для передачи данных между уровнями в n-уровневом приложении.

  Поскольку класс CursorAdapter может создавать VFP-курсоры из данных моделей ADO или XML, он идеально подходит для использования в слое пользовательского интерфейса n-уровневого приложения.

Заключение

Я считаю, что класс CursorAdapter является одним из самых важных и наиболее впечатляющих дополнений, реализованных в версии VFP 8, потому что он обеспечивает унифицированный и легкий в применении интерфейс для доступа к удаленным данным плюс, как мы увидим в будущих статьях, он позволяет нам создавать повторно используемые классы данных. В следующем месяце мы рассмотрим специфику доступа к встроенным и удаленным данным с использованием технологий ODBC, ADO и XML. Еще через месяц мы рассмотрим создание повторно используемых классов данных и обсудим, как использовать класс CursorAdapter в отчетах.

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

❑❑❑

Если бы вариант использования мог использовать вариант…

Нэнси Фолсом (Nancy Folsom)

В своей последней статье «Модель поведения» Нэнси Фолсом привела пример, в котором представила форму оплаты в виде диаграммы классов и UML-модели из статьи «Каким способом переданы данные?», опубликованной за месяц до этого. В этом месяце Нэнси рассмотрит моделирование вариантов использования (use case) с практической точки зрения: она объяснит, каким образом можно применить эту методику для решения более интересной задачи осуществления импорта данных в систему.

По прошествии времени я против своей воли пришла к убеждению, что достижение хороших результатов в объектно-ориентированном программировании зависит от моделирования. Причина моего нежелания согласиться с этим постулатом заключается просто-напросто в том, что применение именно и в особенности вариантов использования представляется чрезвычайно мучительным делом с точки зрения их описания. Хотя я по-прежнему считаю, что варианты использования — это просто пытка, их применение окупается там, где я этого меньше всего ожидала. Если до этого вам не приходилось описывать варианты использования, сделаю предупреждение: ощущение у вас будет такое, как будто вы пишете учебное пособие, предназначенное для «младшего школьного возраста» (посмотрите статью по адресу www.straightdope.com/classics/a980814a.html). Вам может показаться, что вы пишете совершенно очевидные вещи. Если я в состоянии убедить вас сделать попытку и действовать в указанном направлении, то полагаю, вы согласитесь с тем, что варианты использования заслуживают потраченных на них усилий; они не только ведут прямиком к техническому проекту, но также предоставляют материалы для написания документации, проведения тестирования и организации управления проектом (определение области применения (managing scope) и защита от возможной «деформации» функциональных возможностей).

Основные положения и требования

В прошедшие месяцы я создала в качестве примера, предназначенного для использования в последующих статьях, форму для ввода данных о платежах, обладающую потрясающе мощными функциональными возможностями (см. рис. 1). Мне хотелось бы рассмотреть новую задачу. Итак, необходимо импортировать платежи из некоторой «внешней» системы в «платежное» приложение. Осуществить такой импорт легко, если: 1) это одноразовая акция, и поэтому я могу написать программную «заплатку», которую никто не будет сопровождать или использовать повторно; 2) импорт осуществляется в строго определенном формате, который гарантированно никогда не будет изменен; 3) платежное приложение может «диктовать» формат импортируемых данных или 4) импортируются таблицы и столбцы, а не предметные сущности реального мира. Перечисленные ситуации четко определены и, следовательно, не представляют особого интереса с точки зрения данной статьи. С другой стороны, импорт оказывается трудной для решения задачей, если пользователям необходимо импортировать данные, представленные в различных форматах. В этом случае программа должна быть расширяемой или же реально импортируемые данные должны позволять разную последовательность действий.

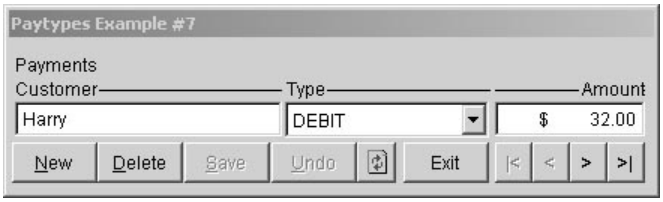

Рис. 1. Форма ввода платежей из предыдущих статей.

Оговорю системные требования, учитываемые в данном примере (см. рис. 2). В рассматриваемом приложении имеются таблицы для хранения данных о платежах, покупателях и типах оплаты. Пользователи располагают формами для ввода данных о платежах — что является их основной задачей — и формами для ввода данных о новых покупателях и новых типах оплаты. Пакетный ввод платежей из электронных источников — это та новая функцио-

1. Поддержка различных источников импорта. Прямо сейчас нам необходимо импортировать данные из табличного процессора Excel и СУБД Access, но мы хотим иметь возможность добавлять другие форматы в будущем.
2. К работе будут привлечены пользователи, не обладающие «техническими» знаниями. Они будут знать, как выполнить работу по вводу платежей, но им неизвестны имена таблиц или полей. Эта система не может требовать от пользователя знаний на уровне таблиц или полей.
3. Процедура импорта должна была бы исполняться с минимальным перепрограммированием, даже если вносятся изменения в табличные структуры.

Рис. 2. Наши новые требования.

нальность, для которой в данной статье представлен вариант использования, как пример сравнительно простого моделирования вариантов использования.

Разработка вариантов использования

Вариант использования аналогичен рецепту. Вариант использования представляет собой связный рассказ на обычном языке о том, как выполнить логически завершенный кусок работы. Язык этого рассказа должен быть обычным с точки зрения пользователя. Детализация (степень подробности) варианта использования является предметом для обсуждения. Поскольку мой опыт — это создание информационных приложений в области бизнеса, то я предпочитаю считать корректным тот уровень, в основе которого лежит некоторая задача. Задача, даже если она является составной частью системы большего размера, либо определенным образом связана с другими задачами, или же занимает определенное место в порядке исполнения всех задач — это нечто самостоятельное. Например, клиенты не могут размещать чеки для последующего инкассирования (deposit checks) прежде, чем они откроют счет. Следовательно, может существовать один вариант использования для процедуры «Customer makes deposit» («Клиент делает взнос») и другой — для процедуры «Open customer account» («Открыть для клиента счет»). В первом случае неявно присутствует вариант использования «Check Customer Status» («Проверить статус клиента»).

Чтобы ваша попытка оказалась более успешной, описывайте варианты использования итеративно. Если я слишком рано углубляюсь в детали, я имею тенденцию «увязнуть» в мелочах и упустить из виду более крупные цели. Сначала приступайте к верхнему уровню и выясните лишь список вариантов использования, который проистекает из предъявляемых требований. Потом представьте каждый вариант использования в кратком изложении и, наконец, затем «соберите» подробности. Легче заметить сходство вариантов использования в том случае, если вы до некоторого момента разрабатываете все варианты одновременно (насколько это возможно). Когда вы выявляете некий дополнительный вариант использования в процессе создания другого варианта, тогда легче «подтянуть» новый вариант использования до уровня остальных, если вы работаете в упомянутой манере.

Всегда придерживайтесь точки зрения пользователя. Вариант использования предназначен не для вас — проектировщика, разработчика или языка программирования. Эксперты утверждают, что варианты использования следовало бы писать пользователям. Откровенно говоря, я считаю, что пользователи не станут этого делать. Я буду просто в восторге, если они примут участие в выработке требований. (Нет, я не говорю им, что наше занятие носит иное название нежели «обсудим по телефону».) Идея замечательная, не спорю, и если ваш клиент готов это сделать, не отказывайтесь. В противном случае, если описанием вариантов использования занимаетесь вы, пишите так, как это делал бы Аль Пачино (Al Pacino): проникните в голову к пользователю и описывайте варианты использования теми словами, какие использовал бы пользователь.

На что похож вариант использования?

Формат для изложения вариантов использования варьируется от формального до произвольного. Возьмите вот этот документ и ознакомьтесь с некоторыми приведенными в нем примерами http://members.aol.com/acockburn/papers/uctempla.doc. Я предпочитаю использовать формальный шаблон, вероятно потому, что легко теряюсь в отсутствии ясного «путеводителя». Впрочем, формализация может сбить с толку. Если вы обнаружите, что строгая формализация является препятствием, используйте неформальный подход. В этой статье я пользуюсь неформальным шаблоном. Имеет смысл принять в качестве рабочего такой формат, который подходит вам и вашему рабочему коллективу и обеспечивает получение следующих сведений:

- В чем состоит суть сценария?
- С чего сценарий начинается и чем он заканчивается?
- Что должно послужить основанием для того, чтобы сценарий начал исполняться, и каково будет состояние системы после того, как сценарий завершит свою работу?

- Кто участвует в сценарии? «Кто» — это действующие субъекты, и ими могут быть люди или другие системы.
- Из каких шагов складывается сценарий?
- Какие действия выполняют действующие субъекты и система на каждом шаге?
- Каковы штатные действия, предусмотренные для штатных шагов?
- Какие исключения встречаются на каждом шаге? Как система обрабатывает эти исключения? Какие возможности для выбора будут предоставлены пользователю?
- Существуют ли другие варианты использования сильно или слабо связанные с данным вариантом использования?

Итак, содержание варианта использования включает не только сценарий, но и ссылки на связанную с ним информацию о системе.

Вопросы, на которые вы должны ответить

Несколько вопросов, которые должны получить ответ при написании варианта использования:

- Какова цель написания варианта использования? Цель заключается в том, чтобы понять: кто использует систему, как они это делают, и что делает система в ответ на действия пользователей.
- Что такое конечный продукт? Конечный продукт – это набор документов, состоящий из нескольких или множества текстовых документов: по одному документу для каждого варианта использования.
- Чем полезны варианты использования для проекта? Как было упомянуто ранее, планы тестирования и руководства пользователя можно написать или, по крайней мере, начать их составление, базируясь исключительно на вариантах использования, ибо система будет строиться в соответствии с вариантами использования.
- Как мне узнать, что я завершил работу? Вы завершили работу тогда, когда для всех требований будут предусмотрены варианты использования.
- Являюсь ли я подходящим человеком для выполнения этой задачи? Варианты использования и собранная в связи с этим информация — все это касается заказчика. Не все разработчики или проектировщики хотя бы похожи на пользователей. В этом нет ничего неправильного. Единственная неправильная вещь — это «избиение» пользователей за то, что они не являются компьютерными «спецами». Если бы они в этом разбирались, мы бы им не понадобились. Если вы не любите пользователей, если вы не можете прочувствовать их проблемы, если они разбазаривают ваше время... подумайте о том, чтобы объединиться в команду с коллегой, который в состоянии будет сделать все это.

Вариант использования №1: служащий импортирует пакет платежей в систему.

Служащий выбирает тип источника импорта и источник импорта, который содержит пакет платежей. Он сообщает системе о том, как именно импортируемые данные отображаются в системные платежи, а затем, когда служащий готов выполнить операцию, он отдает распоряжение системе начать процедуру импорта. Система импортирует все платежи и выдает служащему сообщение по завершении процесса.

Предположения

1. Импорт продолжается без ошибки.
2. Служащий знает, как определить и найти известный тип импорта.
3. Вся информация о платежах, содержащаяся в импортируемых данных, является полной и корректной.
4. Служащий будет знать, как отображать импортируемые данные в соответствующие платежные данные.
5. Человек инициирует и начинает операцию импорта. Что, если это автоматический процесс, исполняемый, скажем, в ночное время?

Рис. 3. Конспект сценария.

Приступим

Давайте шаг за шагом проделаем все вышеизложенное.

Список сценариев

Поскольку я всего лишь добавляю новые функциональные возможности, составить такой список просто. Слишком просто:

ВариантИспользования#1: служащий обращается к системе с запросом об импорте пакета платежей.

Конспект, описывающий штатное исполнение сценария

Составьте конспект сценария, состоящий из нескольких предложений (см. рис. 3). Если для этого вам требуется больше пяти предложений, возможно, вы имеете дело с несколькими вариантами использования; если у вас не набирается хотя бы трех предложений, вы, быть может, пытаетесь создать сценарий на слишком низком уровне. При написании сценария учитывайте следующие советы:

- Используйте обычный язык. Будьте ясны и последовательны в изложении.
- Фиксируйте начало и конец. Как сценарий стартует? Когда прекращается его исполнение?

Вариант использования №1: служащий импортирует пакет платежей в систему.
Некий служащий выбирает какой-то тип источника импортируемых данных и сам источник импорта, который содержит набор платежей. Служащий сообщает системе о том, как именно импортируемые данные отображаются в системные платежи, а затем, когда служащий готов выполнить операцию, он отдает системе распоряжение начать процедуру импорта. Система импортирует все платежи и сообщает служащему о завершении работы.

Предположения

1. Импорт продолжается без ошибки.
2. Служащий знает, как идентифицировать и найти известный тип импорта.
3. Вся информация о платежах, содержащаяся в импортируемых данных, является полной и корректной.
4. Служащий будет знать, как отобразить импортируемые данные в соответствующие платежные данные.
5. Человек инициирует и начинает процедуру импорта. Что, если это автоматический процесс, который исполняется, скажем, в ночное время?

Исключения

1. Информация о платежах может быть неполной. Какие сведения необходимы, а какие необязательны, если таковые существуют? Замечание: сбор требований должен был бы обеспечить эту информацию, но не существует совершенных процедур, и это одна из причин для применения вариантов использования: собрать те требования, которые были упущены на стадии сбора требований:
   a. Если данные о типе платежа или покупателе отсутствуют, или если количество меньше чем или равно нулю, тогда не импортировать платеж.
2. Покупатель не существует в системе.
   a. Импортировать покупателя автоматически.
3. Тип оплаты не существует в системе.
   a. Не импортировать платеж.

Рис. 4. Возможные исключения.

- Включайте в конспект относящиеся к делу подробности, но не увязните в них. Для этого есть масса возможностей в дальнейшем.
- Почаще задавайте вопрос: «Каковы мои предположения? Что согласно моему предположению произошло или что произойдет позже?» Этот вопрос может выявить исключения, дополнительные варианты использования и пропущенные требования.

Обращайте внимание на возможные исключения

Исходя из определенных вами в конспекте допущений, выберите те исключения, которые могут встретиться в процессе работы (см. рис. 4) и составьте перечень откликов системы на эти исключения.

Анализ варианта использования и выявление всех связанных с рассматриваемым вариантов использования, которые необходимы в данных обстоятельствах

Проследите за наличием всех тех дополнительных сценариев, которые могут выявиться как следствие анормального исполнения рассматриваемого сценария (см. рис. 5) или тех сценариев, которые не были замечены на стадии сбора требований.

Вариант использования №1: служащий импортирует пакет платежей в систему.
Некий служащий выбирает какой-то тип источника импортируемых данных и сам источник импорта, который содержит набор платежей. Служащий сообщает системе о том, как именно импортируемые данные отображаются в системные платежи, а затем, когда служащий готов выполнить операцию, он отдает системе распоряжение начать процедуру импорта. Система импортирует все платежи и сообщает служащему о завершении работы.

Предположения

1. Импорт продолжается без ошибки.
2. Служащий знает, как идентифицировать и найти известный тип импорта.
3. Вся информация о платежах, содержащаяся в импортируемых данных, является полной и корректной.
4. Служащий будет знать, как отобразить импортируемые данные в соответствующие платежные данные.
5. Человек инициирует и начинает процедуру импорта. Что, если это автоматический процесс, который исполняется, скажем, в ночное время?

Исключения

1. Информация о платежах может быть неполной. Какие сведения необходимы, а какие необязательны, если таковые существуют? Замечание: сбор требований должен был бы обеспечить эту информацию, но не существует совершенных процедур, и это одна из причин для применения вариантов использования: собрать те требования, которые были упущены на стадии сбора требований:
   a. Если данные о типе оплаты или покупателе отсутствуют, или если количество меньше чем или равно нулю, тогда не импортировать платеж.
2. Покупатель не существует в системе.
   a. Импортировать покупателя автоматически.
3. Тип оплаты не существует в системе.
   a. Не импортировать платеж.

Связанные с данным варианты использования (см. также)

Вариант использования №2: исключение встретилось при импорте платежа.
Вариант использования №3: импорт покупателя в систему

Рис. 5. Связанные с рассматриваемым варианты использования.

Наполнение сценария содержанием

Начните с ответа на вопрос: «Что произошло, чтобы сценарий начал исполняться?» Продолжайте отвечать на следующие вопросы: «Что происходит далее?» «Какова реакция системы?» «А что происходит потом?» Вот некоторые дополнительные советы, которым необходимо следовать при изложении подробностей:

- Используйте действительный залог («Мэри выбирает тип источника импорта», а не «выбирается тип источника импорта».) В противном случае, легко запутаться в том, что именно происходит и кто является «виновником» происходящего.

- Не используйте местоимения, допускающие двоякое толкование. Неясно к чему относится указание «это», будьте точны.
- Используйте настоящее время («Мэри выбирает...», а не «Мэри выберет...»).
- Используйте простые предложения.
- Не используйте программистский жаргон. «Джо находит импортируемый файл», а не «Джо открывает проводник файлов и находит файл, щелкает по нему мышью, затем щелкает мышью по кнопке «открыть». О специальных технических нюансах побеспокоится технический проект.
- Не переживайте слишком сильно по поводу реализации. С реализацией, опять-таки, разберется технический проект. Однако, используйте макеты экрана, чтобы проиллюстрировать для пользователя, как приблизительно все это будет выглядеть.
- Используйте конструкцию «действующий субъект делает что-то», которая в результате дает «некоторый отклик системы».
- Каждое предложение должно быть чем-то одним: «Мэри выбирает импортируемый файл». Если есть несколько вещей, которые пользователь может сделать перед тем, как на его действия должна отреагировать система (скажем, Мэри выбирает тип источника импорта и местонахождение источника импорта), то затем потребуется явное действие, которое предпринимает Мэри, чтобы сообщить системе о том, что система должна включиться в процесс. Тщательная проработка этих подробностей прямо сейчас — это достаточно нетривиальная задача (делает ли система что-либо в промежутке времени между тем, как Мэри выбирает тип оплаты и файл). Это именно того рода решение, которое следовало бы включить в вариант использования, поскольку оно определяет навыки пользователя и задачу программиста. Вот поэтому-то, кроме всего прочего, написание вариантов использования может оказаться утомительным занятием.

На рис. 6 представлен черновой вариант подробного описания. Обратите внимание на то, что мне пришлось добавить разделы для изложения предусловия и постусловия. Эти разделы появились в результате проверки моих предположений о том, что должно произойти до того, и какое влияние окажет сценарий после своего завершения на состояние системы. Заметьте также, что теперь имеется большее количество вариантов использования. Вот они:

- Вариант использования № 2: исключение встречается в процессе импорта платежа.
- Вариант использования № 3: импорт покупателя в систему.
- Вариант использования № 4: проверка информации об оплате на полноту.
- Вариант использования № 5: проверка информации об оплате на корректность.
- Вариант использования № 6: добавить платеж в систему.

Описывая вариант использования, я отслеживала такие места, где реакция системы является слишком сложной или действие могло бы быть повторно используемым. Например, если для данного платежа отсутствует покупатель (полнота), тогда процедура импорта не обязана нести ответственность за принятие решения о том, как обрабатывать такую ситуацию. Это решение принадлежит бизнес-правилу, и на самом деле, большая часть вариантов использования — это бизнес-логика. Процедура импорта просто должна знать о том, что система вообще может принять предложенный платеж.

Если вы не начали составлять глоссарий во время сбора требований, сделайте это. Если глоссарий у вас есть, не переставайте пополнять его по мере написания вариантов использования. Используйте этот глоссарий как руководство по языку и терминологии при написании вариантов использования.

В заключение

Следующий шаг — анализ варианта использования, проводимый вместе с заказчиком. Имеет ли сценарий смысл для заказчика? Может ли заказчик придумать какие-либо дополнительные исключения для определенных вами правил? Имеет ли смысл вариант использования для проектировщиков (или вы сами выступаете в роли программиста)? В состоянии ли проектировщики заметить какие-либо неоднозначности? Прежде чем продолжить дальнейшую работу над проектом, оцените его выполнимость. Разумно ли продолжать работу, принимая во внимание способ работы данной функциональной возможности и ее влияние на пользователей и другие системы? Гораздо лучше потратить часы и даже дни на такой вариант использования, который демонстрирует, что проект неосуществим, чем потратить месяцы на написание модулей, которые никогда не работают по-настоящему хорошо для пользователей и являются кошмаром при сопровождении.

Постусловие

Как только варианты использования готовы, вы можете приступить к разработке модели предметной области (domain model). Выбирайте такие существительные, которые могут стать классами или свойст-

Вариант использования №1: служащий импортирует пакет платежей в систему.
Некий служащий выбирает некоторый тип источника импорта и источник импорта, который содержит набор платежей. Он сообщает системе о том, как импортируемая информация отображается в системные платежи, а затем, когда служащий готов выполнить операцию, отдает распоряжение системе начать процедуру импорта. Система импортирует все платежи и сообщает служащему о завершении работы.

Предусловия:

Предполагается, что служащий знает, как идентифицировать и найти известный тип импорта, и что он знает, как отобразить импортированные платежи в системные платежи. Предполагается, что во время импорта только этот служащий использует базу данных приложения.

Постусловия:

Платежи, которые могли бы быть импортированы, находятся в системе и доступны другим пользователям. Система выдает на экран сообщение о том, сколько платежей импортировано из общего числа доступных. Система протоколирует все ошибочные ситуации. Информация об ошибке включает данные о платеже и причину ошибки.

Связанные с данным варианты использования (см. также)

Вариант использования № 2: во время импорта платежей встретилось исключение.
Вариант использования № 3: импорт в систему данных о покупателе.
Вариант использования № 4: проверка на полноту данных о платежах.
Вариант использования № 5: проверка корректности данных о платежах.
Вариант использования № 6: добавление платежа в систему.

Сценарий

Действующий субъект	*Система импорта*
Система импорта Мэри выбирает опцию импорта платежей.	Система открывает на экране окно мастера, начиная с процедуры навигации для обнаружения файлов и URL-локаторов. Система также выдает на экран список доступных типов импорта. Доступными типами импорта являются табличный процессор Excel, СУБД Access и служба Web service.
Мэри выбирает источник и тип импорта.	Система проверяет существование источника и его соответствие предполагаемому типу. Если источник не существует, см. вариант использования № 2.a: “Источник недоступен”. В противном случае, система открывает источник и выдает на экран информацию в простом (строка|столбец) формате. Система также выдает на экран вариант данных о платеже, который может быть импортирован (Имя покупателя| количество| тип платежа). Система открывает на экране окно инструмента визуального связывания.
Мэри использует инструмент визуального связывания, чтобы связать импортируемую колонку с некоторой частью платежной информации. Когда она это сделала, Мэри сообщает системе о том, что она готова начать импорт.	Система перебирает в цикле все импортируемые платежи, чтобы проверить полноту (см. вариант использования № 4) и корректность данных (см. вариант использования № 5). Если информация полна и корректна, добавить платеж в систему (см. вариант использования № 6). Когда система закончила работу, она выдает на экран результаты операции импорта. Представляются сведения о количестве платежей, которые были успешно импортированы, количестве неуспешных операций импорта, список платежей, которые не удалось импортировать, и причины неудачи.
Мэри анализирует представленные на экране результаты и может затем принять одно из двух решений: распечатать полученные результаты или закрыть окно.	Если Мэри принимает решение распечатать результаты, система делает это и затем закрывает мастера. В противном случае система просто закрывает мастера.

Рис. 6. Первый черновик варианта использования.

вами; прилагательные, которые могли бы стать свойствами; и глаголы, которые могут быть методами. Не используйте слишком сложные грамматические конструкции. Чем проще язык изложения, тем проще вариант использования.

Чем так замечательны варианты использования?

Варианты использования замечательны тем, что они полезны. Они образуют фундамент не только для технического проекта, но и для планов тестирования и руководств, а также не теряют своего значения для заказчика и управления проектом. Как часто на поздней стадии проектирования клиент задает вопрос: «Почему вы делаете это именно так?» Вы могли бы пожать плечами и ответить: «Не знаю. Есть ощущение, что это — удачная идея». Но вместо этого, можно было бы схватить вариант использования № стонадцатый и в точности продемонстрировать клиенту, почему именно все делается так, а не иначе. Кроме того, все — от клиента до разработчика — имеют четкое определение области действия, следовательно, проще избежать искажения функциональных возможностей. Думайте о варианте использования, как о повторно используемом модуле. Со временем варианты использования, точно так же как программный код, могут быть применены повторно.

❑❑❑

Научный подход к внешнему виду

Арт Бергквист (Art Bergquist)

В этой статье Арт Бергквист демонстрирует нам легко реализуемую методику, с помощью которой вы можете заставить элемент управления combo box обрести внешность метки — label (то есть заставить этот элемент управления работать в режиме «только для чтения» как в смысле внешнего вида, так и с точки зрения функциональности) и позже, при необходимости, снова привести его в то состояние, которое предусмотрено для стандартного (с возможностью работы в режиме «чтение-запись») комбинированного списка.

Недавно я попал в ситуацию, в которой были задействованы элементы управления combo box: мне необходимо было подвергнуть фильтрации список одного элемента управления combo box на основе значения, полученного из другого комбинированного списка. Однако, когда я это проделал, то оказалось, что в «отфильтрованном» списке осталось только одно значение для выбора. Не знаю как вы, но я рассматриваю элемент управления combo box, в списке которого есть только одно значение, как образец плохого (читайте: враждебного по отношению к пользователю) интерфейса. И вот почему: когда пользователь видит изображение стрелки в элементе управления combo box, он/она склонны щелкнуть по этой стрелке мышью, добиваясь тем самым появления (то есть открытия) комбинированного списка с целью посмотреть на другие возможные варианты выбора. Однако, если комбинированный список имеет только одно значение, вы просто потратите время (а следовательно, деньги) и энергию вашего пользователя на то, чтобы он убедился: других вариантов, которые можно было бы выбрать, в этом списке нет.

Данная статья демонстрирует легко реализуемую методику, с помощью которой вы можете так «преобразить» комбинированный список, что он приобретет вид элемента управления label (этот комбинированный список будет доступен только для чтения как с точки зрения внешнего вида, так и в смысле своей функциональности) и позже, в случае необходимости, вернуть его обратно в прежнее состояние. Как я сказал ранее, одной из причин, по которой вам это может потребоваться, является тот факт, что в некоторый момент времени комбинированный список имеет только одно значение для выбора. Или, возможно, ваш пользователь может находиться на таком уровне системы безопасности, на котором ему/ей позволено только видеть значение элемента управления combo box, но не изменять его (и, может быть, не видеть даже других элементов из этого комбинированного списка).

Первая попытка

Моей первой попыткой содрать шкуру с этой лисы стала следующая методика, которую я описал в статье, доступной по адресу: http://fox.wikis.com/wc.dll?Wiki~ComboBoxReadOnly~VFP

Я добился хороших визуальных результатов, следующим образом определяя значения для трех свойств элемента управления ComboBox:

```
WITH [ссылка на объект ComboBox]
  .Enabled = .F.
  .BorderStyle = 0 && None
   * Plain (элемент упр-ия появляется без 3-мерной рамки)
  .SpecialEffect = 1
ENDWITH
```

Чтобы вы сами могли это увидеть, без необходимости создавать вначале форму, наберите все ниже перечисленные команды в окне Command Window и наблюдайте результат на экране по мере ввода каждой из них:

```
CLEAR
 _Screen.BackColor = 12632256
 _Screen.AddObject('cboReadOnly', 'ComboBox')
 _Screen.cboReadOnly.Visible = .T.
 * На пол-экрана вниз
 _Screen.cboReadOnly.Top  = SYSMETRIC(2) / 2
 * На пол-экрана вправо
 _Screen.cboReadOnly.Left = SYSMETRIC(1) / 2
 _Screen.cboReadOnly.Enabled = .F.
 _Screen.cboReadOnly.BorderStyle = 0
 _Screen.cboReadOnly.SpecialEffect = 1
 _Screen.cboReadOnly.DisplayValue = 'ReadOnly Text'
```

После того, как вы ввели все эти команды в окне Command Window, вы должны были в итоге увидеть нечто, аналогичное тому, что изображено на рис. 1.

Конечно, если вы размещаете весь этот код в программе .PRG, вам следовало бы использовать конструкцию WITH ... ENDWITH, вот так:

```
CLEAR
 WITH _Screen
   .BackColor = 12632256
   .AddObject('cboReadOnly', 'ComboBox')
   .cboReadOnly.Visible = .T.
    * На пол-экрана вниз
   .cboReadOnly.Top  = SYSMETRIC(2) / 2
```

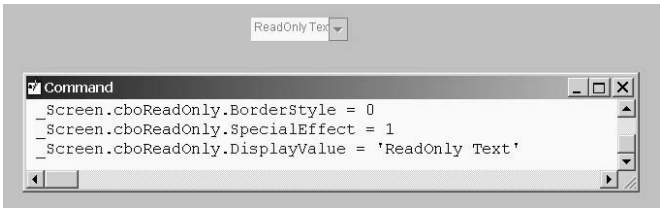

Рис. 1. Моя первая попытка создать combo box, открытый только на чтение.

```
    * На пол-экрана вправо
   .cboReadOnly.Left = SYSMETRIC(1) / 2
   .cboReadOnly.Enabled = .F.
   .cboReadOnly.BorderStyle = 0
   .cboReadOnly.SpecialEffect = 1
   .cboReadOnly.DisplayValue = 'ReadOnly Text'
ENDWITH
```

Если вы не преуспели с первого раза, не оставляйте попыток, попытайтесь снова

Сам по себе, мой первый «наскок» на эту проблему не был неудачным, но внешний вид элемента управления по-прежнему оставался несколько недружелюбным по отношению к пользователю, в том смысле, что пользователь видел все ту же (вводящую в заблуждение) «стрелку вниз», по которой он/она склонен был щелкнуть мышью, в ответ на что, разумеется, не последовало бы никаких действий: комбинированный список не появился бы, поскольку он был определен для использования в режиме «только для чтения».

Несколько месяцев тому назад я снова оказался в такой же ситуации, и, взглянув на нее по-новому (иногда время, как составная часть такого процесса, выступает в роли помошника), я «зачал» более удачное решение (после того, как моя жена родила нашего четвертого ребенка, мне нравится это слово).

Вот общая предпосылка. В методе события Init() создать объект shape и настроить его свойства .Height, .Top, .Width и .Left таким образом, чтобы в том случае, если этот объект является видимым, он полностью закрывал собой изображение «стрелки вниз» в элементе управления combo box. Затем вы определяете, когда надо сделать так, чтобы комбинированный список выглядел как элемент управления label, передавая в качестве единственного параметра для пользовательского метода Labelize() значение .T.. (Передача в метод Labelize() в качестве параметра значения .F. вернет элементу управления combo box тот внешний вид, который предусмотрен для него по умолчанию).

Давайте перейдем к программированию

Теперь давайте посмотрим, как я собственно реализовал вышеупомянутую функциональность. Прежде всего, разберем метод события Init() класса элемента управления combo box, который я создал для этой цели:

```
*  Cbo2Lbl.VCX/cboLabelEnabled::Init()
IF NOT DODEFAULT()
  RETURN .F.
ENDIF
LOCAL lcName, lnThisHeight, lnThisTop, ;
      lnThisRight, lcNameOfShape
WITH This
  lcName       = .Name
  lnThisHeight = .Height
  lnThisTop    = .Top
  lnThisRight  = .Left + .Width
  WITH .Parent
    * Уникальное имя для нового объекта shape
    lcNameOfShape = 'shp' + lcName
   .AddObject(lcNameOfShape, 'Shape')
    WITH .&lcNameOfShape.
        * Совпадает со значением свойства .Height
        * элемента управления combo box
      .Height = lnThisHeight
        * Совпадает со значением свойства .Top
        * элемента управления combo box
      .Top = lnThisTop
        * Полностью закрывает вертикальную стрелку
        * для раскрытия списка
        * в элементе управления combo box.
      .Width = SYSMETRIC(5)
      .Left  = lnThisRight - .Width
    ENDWITH
  ENDWITH
 .AddProperty('icShape', lcNameOfShape)
 .Refresh()
ENDWITH
```

Метод события Init() создает объект shape и настраивает его свойства .Height, .Top, .Width и .Left таким образом, чтобы в том случае, если этот объект является видимым, он полностью закрывал собой «стрелку вниз» в изображении элемента управления combo box.

```
*  Cbo2Lbl.VCX/cboLabelEnabled::Refresh()
DODEFAULT()
LOCAL lcComboboxArrowHidingShape, ;
      lnParentBackColor
WITH This
  lcComboboxArrowHidingShape = .icShape
  WITH .Parent
    lnParentBackColor = .BackColor
    WITH .&lcComboboxArrowHidingShape.
      .BorderColor = lnParentBackColor
       * Я обнаружил, что для некоторых сценариев
       * необходимы также следующие 2 строки
      .FillColor = lnParentBackColor
      .FillStyle = 0
    ENDWITH
  ENDWITH
ENDWITH
```

Метод события Refresh() гарантирует, что свойства .BorderColor и .FillColor объекта shape имеют те же значения, что и свойство .BackColor родительского объекта элемента управления combo box (этими родительскими объектами могли бы, например, быть форма, страница в страничном блоке, контейнер и

так далее). Идея заключается в том, что объект shape не должен привлекать внимания: пользователь не должен даже замечать его.

```
*--------------------------------------------
 * Cbo2Lbl.VCX/cboLabelEnabled::Labelize()
 *
 * Заставить этот комбинированный список
 * выглядеть как метка label.
 * Примером того, когда бы вам потребовалось проделать
 * это, служит ситуация, в которой у вас есть только одно
 * значение в комбинированном списке, и вы не хотите
 * сбить с толку своего пользователя, заставляя его/ее
 * думать, что он/она может изменить это значение
 * (если они откроют комбинированный список).
 *--------------------------------------------
 LPARAMETERS tlLabelize
 LOCAL llLabelize
 IF PCOUNT() < 1                  ;
     OR VARTYPE(tlLabelize) # 'L'
   llLabelize = .T.
 ELSE
   llLabelize =  tlLabelize
 ENDIF

 WITH This
   LOCAL lcComboboxArrowHidingShape
   lcComboboxArrowHidingShape = .icShape
   IF llLabelize
     * Сначала разберемся с самим
     * элементом управления combo box
    .Enabled     = .F.
    .BorderStyle =  0     && None
     * Теперь определим значения свойств элемента
     * управления shape, связанных с самим
     * комбинированным списком (перенесено в метод
     * события This.Init())
     WITH .Parent.&lcComboboxArrowHidingShape.
       .Visible = .T.
        * Поместить впереди в Z-порядке
        * (т. е. перед элементом управления combo box)
       .zOrder(0)
     ENDWITH
   ELSE

     * Сначала имеем дело с самим
     * элементом управления combo box
    .ResetToDefault('Enabled'    )
    .ResetToDefault('BorderStyle')

     * Теперь определим значения свойств элемента
     * управления shape, связанных с самим комбинированным
     * списком [перенесено в метод события This.Init()]
     WITH .Parent.&lcComboboxArrowHidingShape.
       .Visible = .F.
        * Поместить в конце Z-порядка
        * (т. е. позади элемента управления combo box)
       .ResetToDefault('zOrder')
     ENDWITH
   ENDIF
 ENDWITH
```

Пользовательский метод Labelize() позволяет вам сделать одно из двух, в зависимости от того, передаете ли вы значение .T. или .F. в качестве параметра. Если вы передаете значение .T., то тем самым заставляете комбинированный список combo box приобрести внешний вид метки label; если вы передаете значение .F., метод разворачивает процесс в обратном направлении (что вынуждает комбинированный список снова принять обличье элемента управления combo box): переданное в качестве параметра значение .F. возвращает элементу управления combo box внешний вид, определенный для него по умолчанию).

- Передача значения .T. не только присваивает свойству .Enabled значение .F., а свойству .BorderStyle значение 0 (то есть "None"), как я это делал во время своего первого «набега» на решение данной проблемы, но также делает объект shape, созданный в методе события Init(), видимым и размещает его впереди согласно Z-порядку (то есть передача значения .T. помещает объект shape поверх элемента управления combo box).
- Передача значения .F. разворачивает процесс в обратном направлении, возвращая с помощью метода ResetToDefault() значения свойствам .Enabled и .BorderStyle, а также делая скрывающий-стрелку-комбинированного-списка-объект-shape невидимым и помещая его в конец согласно Z-порядку (то есть передача значения .F. помещает объект shape позади элемента управления combo box).

Не хотите ли пример?

На дискете вы найдете форму Cbo2Lbl. Если вы выполните команду DO FORM Cbo2Lbl, то увидите довольно невинную форму (как показано на рис. 2).

Однако, если вы щелкните мышью по командной кнопке Labelize, комбинированный список будет заменен на то, что внешне выглядит как метка label (см. рис. 3).

Здорово, а? Щелчок мышью по командной кнопке Re-"Combo"-fy возвращает комбинированному списку внешний вид, принятый для него по умолчанию (комбинированный список в режиме «чтение-запись»), как показано на рис. 2.

Нырнем в...

Давайте заглянем «внутрь» формы Cbo2Lbl. На уровне формы к существу дела имеет отношение только одна вещь: свойство .Caption получает значение "Combo-box-to-Label-and –back". В комбинированном списке (названном, кстати, cboComboBox) я создаю массив на уровне элемента управления

Рис. 2. Пример формы, показывающий нормальную (т. е. открытую как на чтение, так и на запись) combo box.

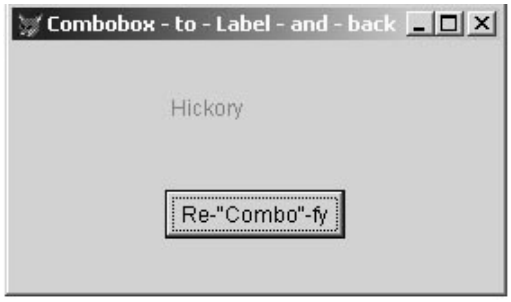

Рис. 3. Пример формы, показывающей объект, похожий на метку, но на самом деле немодифицируемый (read-only) combo box.

combo box (в методе события Init()), который содержит варианты выбора, доступные тому, кто исполняет этот пример формы:

```
IF NOT DODEFAULT()
   RETURN .F.
 ENDIF
 WITH This
   .AddProperty('aChoices[1]')
   .Requery()
 ENDWITH
```

Метод события Requery() просто запоминает три строки “Hickory”, “Oak” и “Pine” в массиве комбинированного списка.

```
DODEFAULT()
LOCAL lcOptionList
TEXT TO lcOptionList NOSHOW
Hickory
Oak
Pine
ENDTEXT
WITH This
   ALINES(.aChoices, lcOptionList, .T.)
  .Value        =  1
  .DisplayValue = .aChoices[1]
  .RowSource    = 'This.aChoices'
ENDWITH
```

Настоящей рабочей лошадью этой формы является командная кнопка cmdLabelize, чей метод события Click() переключает режимы комбинированного списка — стандартный и «только для чтения»:

```
WITH This
   DO CASE
     CASE .Caption = 'Labelize'
       .Parent.cboComboBox.Labelize(.T.)
       .Caption = [Re-"Combo"-fy]
     CASE .Caption = [Re-"Combo"-fy]
       .Parent.cboComboBox.Labelize(.F.)
       .Caption = [Labelize]
   ENDCASE
 ENDWITH
```

Если вы щелкаете мышью по командной кнопке Labelize, происходят две вещи: как упоминалось ранее, элемент управления combo box приобретает вид метки label; кроме того, свойство Caption командной кнопки Labelize получает значение Re-"Combo"-fy. Щелчок мышью по командной кнопке Re-"Combo"-fy отменяет (возвращает в прежнее состояние) то, что сделала командная кнопка Labelize.

Вот и все! Изучите пример формы и сопутствующий класс комбинированного списка, а затем смело внедряйте эту идею в свои собственные приложения.

Один разработчик, с которым я поделился этим решением, засвидетельствовал, что данная методика пригодилась бы ему в приложении, над которым он работал пару лет тому назад и где использовался каскад из четырех, или около того, комбинированных списков: пользователи должны были выбрать грузоотправителя (shipper), затем на основе этого, они должны были определить способы доставки (на следующий день, через день и так далее), затем на основе выбранного способа доставки они могли бы выбрать номера счетов (например, грузоотправителя, получателя или третьей стороны) и так далее. Практически та же ситуация, с которой столкнулся и я: в некоторых случаях имелось бы только одно значение для выбора. Такое положение дел довело этого разработчика до помешательства, особенно благодаря тому, что он должен был «выкапывать» данные из размещенной на SQL Server базы данных Goldmine, которая не только не была нормализована, но и не имела надлежащего разбиения на поля (грузоотправитель, предпочтительный способ доставки и тому подобное — все это хранилось в одном memo-поле на SQL Server, и потеря в производительности была невероятно большой).

Заключение

Коротко говоря, внедряя эту легко реализуемую методику, которую я описал в данной статье, вы можете заставить элемент управления combo box «выглядеть» как метка label (то есть существовать в режиме «только для чтения», как с точки зрения внешнего вида, так и в смысле функциональности) и позже, если необходимо, «переключиться» обратно на обычный (с возможностью чтения-записи) комбинированный список. Пользуйтесь научным подходом к внешнему виду комбинированного поля!

Хотя рассмотренный мной в этой статье программный код использует только те возможности, которые есть в версии VFP 7, вы можете адаптировать его так, чтобы он работал в предыдущих версиях Visual FoxPro.

Арт Бергквист (Art Bergquist) работает в фирме Vision Data Solutions, Inc. (VDSI), где проектирует и разрабатывает приложения в версии Visual FoxPro 7, эксплуатируя каркас приложений Visual MaxFrame Professional. Он имеет статус Microsoft Certified Professional (MCP). abergquist@visionds.com.

❑❑❑

Формы и поля редактирования с градиентной заливкой

Предраг Боснич (Predrag Bosnic)

WIN 7.0 DOWNLOAD

На этот раз Предраг Боснич рассказывает о том, как можно «оживить» Windows-формы, используя в качестве фона градиентную заливку цветом.

В течение многих лет графический Web-интерфейс (Web GUI) пытался «завладеть» графическим интерфейсом операционной системы Windows, но этого не произошло и его по-прежнему нелегко реализовать. За последние пять лет «паутина» адаптировала для своих нужд очень многое из того, чем располагает ОС Windows. Многие элементы управления, используемые в ОС Windows, работают теперь в среде Web. Технология Microsoft .NET выводит Web на передний край с точки зрения будущего разработки. Впрочем, как мне кажется, есть кое-что такое, что берет свое начало именно в «паутине» и что нам следует применить в Windows-приложениях: градиентная заливка цветом фона в формах. В ОС Windows формы были и пока еще остаются несколько «академичными» и в большей степени «техническими» с точки зрения их внешнего вида, тогда как Web-страницы живы и ярки. Как нам известно, ОС Windows «приспособила» градиентную заливку цветом для строки заголовка (title bar) формы, но средства разработки не обеспечивают явной поддержки этой возможности.

Как чаще всего бывает, и эта проблема не является исключением и имеет несколько решений. Я могу попытаться использовать замысловатые математические формулы и алгоритмы, чтобы добиться эффекта градиентной заливки в оперативном режиме. Могу обращаться к функциям языка программирования C++, чтобы проделать то же самое. Или я могу попытаться найти коммерческий элемент управления ActiveX, который обеспечивает такой же эффект, и если мне сопутствует удача, этот элемент управления будет работать в среде Visual FoxPro. Впрочем, в Visual FoxPro есть простое решение данной проблемы, и это решение попадает в категорию «советы». Я могу разместить в форме встроенный в Visual FoxPro элемент управления Image, и задать в его свойстве Picture значение, указывающее на «маленький» файл, в котором хранится картинка градиентной заливки. У элемента управления Image есть свойство Stretch, и я указываю для этого свойства значение 2 (stretch — растянуть). На этапе исполнения размеры элемента управления Image изменятся и будут «подогнаны» под размеры формы.

Класс Form

Первым делом я создаю библиотеку Gradient.vcx и класс, названный Grform, используя в качестве базового класс Form. Я добавляю в форму элемент управления Image и называю его ImageGR. Свойство Stretch элемента управления ImageGR должно иметь значение 2 (stretch — растянуть). В методе Init формы содержится следующий код:

```
DoDefault()
thisform.ImageGr.ZOrder(1)
thisform.Resize()
```

В методе Resize содержится вот такой код:

```
DoDefault()
this.ImageGr.Width  = this.Width
this.ImageGr.Height = this.Height
```

В событиях/методах: Click, DblClick, MouseDown, MouseUp, MouseMove и MiddleClick элемента управления ImageGR имеется следующий код:

```
This.parent.Click()
```

в методе Click, обращение this.Parent.DblClick() в методе DblClick и так далее. Тем самым достигается «прозрачность» элемента управления ImageGR для действий, выполняемых с мышью. Когда вы щелкаете мышью по форме, исполняется метод Click этой формы и так далее. Верьте или нет, но это все!!!

Класс EditBox

Для элемента управления edit box я использую ту же самую библиотеку (Gradient.vcx) и добавляю в нее новый класс GRedit. На этот раз базовым для создаваемого класса служит класс Container, поскольку в решении используются два элемента управления — поле редактирования Edit box и элемент управления Image. Код метода Init этого контейнера — это тот же самый код, который использовался для формы GRform. Однако, в методе Resize находится несколько иной программный код:

```
DoDefault()
 WITH this
   .image1.Width  = .Width
   .image1.Height = .height

   .edit1.Width  = .Width
   .edit1.Height = .height
 ENDWITH
```

Как видите, я должен был настроить размеры элементов управления Image и Edit box так, чтобы они совпадали с размерами контейнера. Свойство BackStyle элемента управления Edit box должно иметь значение 0 (transparent). И опять-таки, это все!

Примеры

Я использовал два примера, чтобы протестировать свое решение. Первый пример (см. рис. 1) является набором форм formset, состоящим из двух форм. В одной форме размещены восемь полей с различными определениями градиентной заливки. Когда вы щелкаете мышью по какому-либо из этих полей, меняется цвет фона второй формы. Если вы пытаетесь изменить размеры второй формы, вы можете видеть, как плавно распространяется по всей форме градиентная заливка фона цветом.

Второй пример (см. рис. 2) использует класс/элемент управления GREdit box. Важно отметить, что поле редактирования Edit box при получении фокуса автоматически закрывает собой фон с градиентной заливкой и снова демонстрирует его в случае потери фокуса. По-моему такое функциональное поведение является целесообразным.

Построитель

Класс (элемент управления) может быть более или менее сложен с точки зрения его использования, но если я могу упростить применение этого класса, снабдив его построителем, то, полагаю, стоит сделать это. В данном случае я могу использовать один и тот же построитель для класса GRform, класса Gredit, а также для еще одной пары классов. Создать новую форму исключительно просто, но принять решение относительно цвета градиентной заливки и выбрать этот цвет намного сложнее. В этом деле может помочь построитель, который демонстрирует список всех зарегистрированных «градиентных» файлов, обеспечив предварительный просмотр цвета градиентной заливки и предоставляя мне возможность зарегистрировать новый файл с градиентной заливкой цветом. Представьте себе следующую ситуацию: у меня есть очень полезный класс, но его очень трудно использовать. Только те разработчики, которым этот класс необходим «позарез», воспользуются им, и при этом они будут испытывать чувство

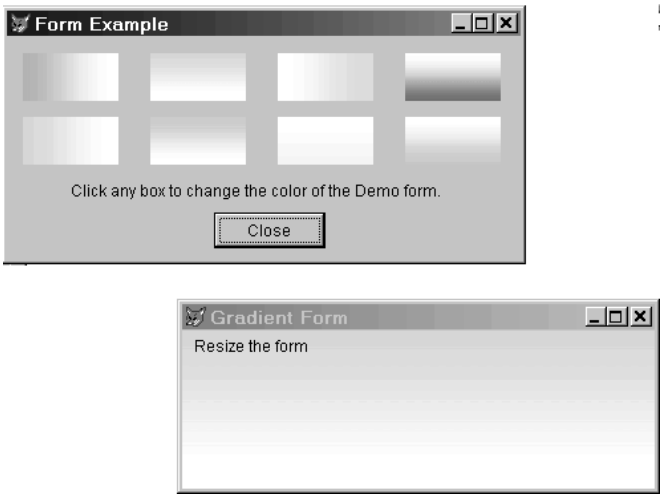

Рис. 1. Форма с градиентной заливкой фона.

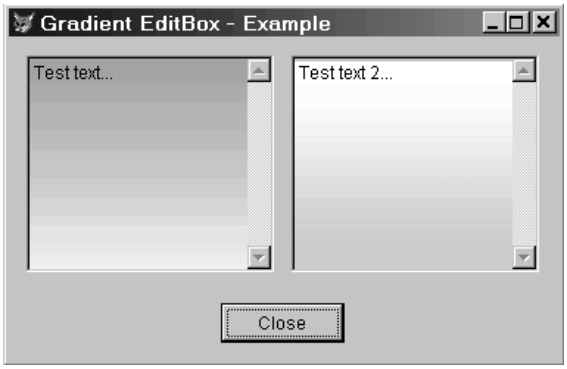

Рис. 2. Поле ввода Edit Box с градиентной заливкой фона.

«ненависти» ко мне. С другой стороны, если тот же самый класс легко использовать, или для него предусмотрен построитель, разработчики будут пользоваться им чаще, и меня никто не будет «ненавидеть».

Если я рассмотрю сложившуюся ситуацию, становится ясно, что для определения цвета градиентной заливки фона формы мне необходимы только имя JPG-файла, полный путь доступа к этому файлу и пара флажков. Я создал таблицу GradColor для хранения в ней всех метаданных. Эта таблица имеет следующую структуру:

```
Id (integer)
xName (char 15)
xPicture (char 100)
Fo (L)
Ed (L)
```

Главная форма построителя показана на рис. 3.

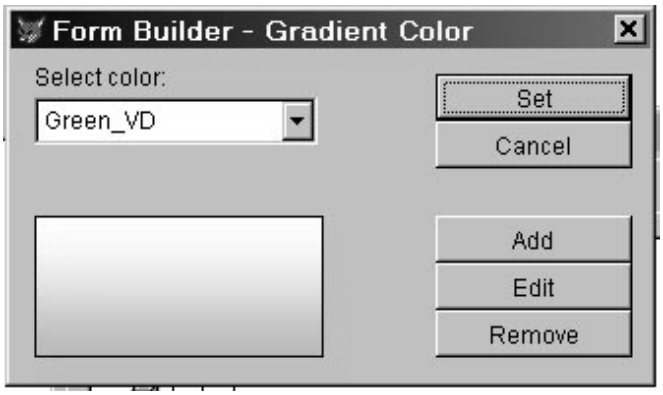

Рис. 3. Форма с градиентной заливкой — построитель.

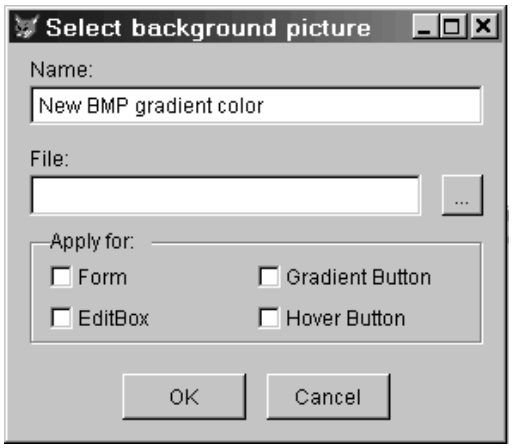

Рис. 4. Построитель позволяет добавить новое определение.

Как видите, пользоваться построителем очень просто. В комбинированном списке Select Color хранятся все известные определения градиентной заливки (этот список привязан к файлу GradColor). Когда я выбираю одно из значений, в окне предварительного просмотра будет показан цвет градиентной заливки. Если я удовлетворен этим цветом, то я нажимаю на кнопку Set. Если мне необходимо добавить новый файл с определением в таблицу GradColor, я щелкну мышью по кнопке Add. На рис. 4 показана форма Add. Мне необходимо только вписать имя определения цвета градиентной заливки и указать тот файл, в котором это определение хранится.

Групповое поле Apply For содержит имена тех элементов управления, для которых используется данное определение. Как вы можете видеть, помимо формы и элемента управления edit box, у меня есть еще пара элементов управления, но пока не обращайте на них внимания. До сих пор все было очень просто, но возникает вопрос, как вам создать файл с определением градиентной заливки цветом?

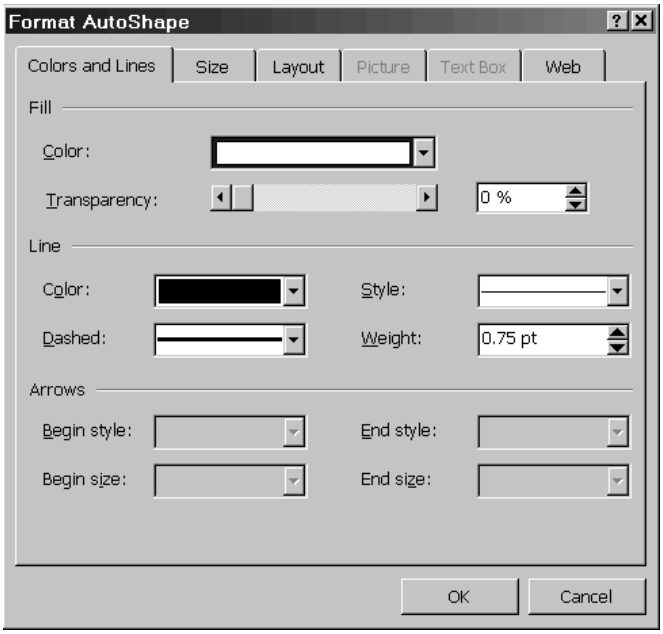

Рис. 5. Диалоговое окно Format AutoShape.

Определение градиентной заливки цветом

Определение градиентной заливки цветом — это графический файл. Для картинки я выбрал размер 144 x 59 пикселей, что дает мне файл размером менее 1K (я использую разрешение 1024 x 768). И опять-таки, все это прекрасно, но как создать такой файл? Что ж, вы можете воспользоваться любым коммерческим программным обеспечением, способным создавать градиентную заливку цветом, но я использую текстовый редактор MS Word XP. Чтобы создать файл с градиентной заливкой цветом, выполните следующие шаги:

- 1. Активизируйте текстовый редактор MS Word.
- 2. Начните новый рисунок (меню: Insert | Picture | New Drawing).
- 3. Добавьте в рисунок прямоугольник (не используйте рамку).
- 4. Задайте для этого прямоугольника размеры 144 x 59 пикселей.
- 5. Щелкните правой кнопкой мыши по прямоугольнику и выберите в меню пункт Format AutoShape (см. рис. 5).
- 6. Перейдите на вкладку Colors and Lines.

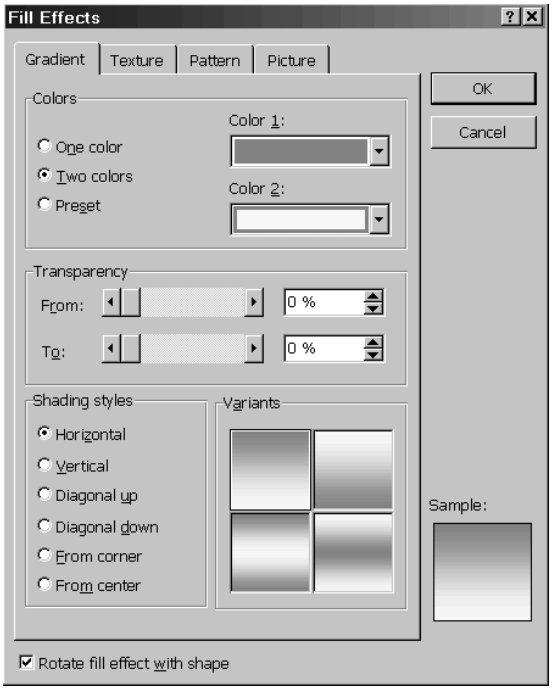

Рис. 6. Диалоговое окно Fill Effects.

- 7. Щелкните мышью по комбинированному списку Color и выберите опцию Fill Effects (см. рис. 6).
- 8. Выберите цвет и эффект по своему усмотрению.
- 9. Щелкните мышью по кнопке OK в форме Fill Effects.
- 10. Щелкните мышью по кнопке OK в форме Format AutoShape. (Итак, ваш прямоугольник получил градиентную заливку цветом).
- 11. Скопируйте этот объект с изображением прямоугольника в буфер обмена Clipboard (Ctrl-C).
- 12. Активизируйте приложение MS Paint.
- 13. Выполните вставку из буфера обмена (Ctrl-V).
- 14. Сохраните результат как JPG-файл.

Хорошая мысль — хранить все ваши файлы с определениями градиентных заливок в одном каталоге: лично у меня есть подкаталог GradColor, расположенный в каталоге C:\Program Files\Microsoft Visual FoxPro 7\Wizards. (Для нормальной работы примеров, которые вы найдете на дискете, все файлы с изображениями нужно поместить в каталог =HOME()+"Wizards\GradColor\", именно туда указывает значение свойства Picture — *прим. ред.*)

GoTo...

Избегаемая большинством программистов команда GoTo почти забыта. В разговоре это звучит несколько жестче или выразительнее. Однако, мне она нравится. Мне нравится, когда кто-нибудь говорит мне: “Go to www.ABC.com” или “Go to www.XYZ.com” и потом оказывается, что это очень хороший и интересный сайт. Если вас интересуют вопросы проектирования пользовательского интерфейса, тогда GoTo:

- www.AskTog.com — Брюс Тогнацци (Bruce Tognazzi) является, вероятно, одним из ведущих авторитетов в области Проектирования Программного Обеспечения. Он работает вместе с Якобом Нильсеном (Jacob Nielsen) — гуру в вопросах обеспечения удобств для общения с Web. На этом сайте вы можете найти ежемесячные постоянные рубрики, в которых Брюс рассматривает конкретные вопросы проектирования, и очень полезный раздел, который называется “First Principles”.
- http://msdn.microsoft.com/ui — сайт фирмы Microsoft, все внимание которого сосредоточено на Windows-технологиях. Вы можете обнаружить там руководство по проектированию приложений для ОС Windows XP, которое можно «скачать», и тому подобные вещи.
- http://research.microsoft.com.ui — сайт фирмы Microsoft, посвященный тем интерфейсам, которые должны появиться в будущем... Они работают над трехмерным «рабочим столом» (3D desktop)!
- www.iarchitect.com — место, где вы можете найти примеры хорошего и плохого проектирования интерфейса.
- www.Ulweb.com — сайт, посвященный общим вопросам проектирования пользовательского интерфейса со списком книг по этой теме.

Заключение

В начале своего пути я предвидел, что самой трудной задачей окажется именно эта. Однако, к тому времени, как я завершил свой труд, решение оказалось сравнительно простым, практически не содержащим программного кода. Это именно то, что я называю «мощь Visual FoxPro». В следующем месяце я продемонстрирую вам, как создаются командные кнопки с эффектом градиентной заливки цветом их фона.

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.

❑❑❑

Новый взгляд на Outlook

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

WIN 7.0 DOWNLOAD

Хотя в составе персонального информационного менеджера Outlook есть мастер экспорта (export wizard), он не предоставляет доступ ко всей той информации о получаемом сообщении электронной почты, которой в действительности обладает данное приложение. Чтобы по-настоящему овладеть процессом извлечения данных из приложения Outlook, вам необходимо использовать технологию Automation. Энди Крамек и Марсиа Акинз обещают вам, что как только вы во всем этом разберетесь, получить данные будет исключительно просто.

Энди: Я знаю, мы, как правило, не отвечаем «на заявки», но эта заявка поступила от того, кому очень трудно сказать «нет».

Марсиа: От кого? Скажи мне, кто это?

Энди: Я, наверное, не могу нарушать оказанное мне доверие, поэтому давай на минуту сделаем вид, что это был наш уважаемый редактор. Итак, он (извините, тут следовало бы сказать «та анонимная личность») хочет сделать следующее: извлечь сообщения электронной почты из папки InBox приложения Outlook и поместить их в таблицу Visual FoxPro.

Марсиа: Что ж, сделать это должно быть достаточно просто, поскольку в приложении Outlook есть мастер экспорта (хотя он и не инсталлируется по умолчанию), чьи возможности включают экспорт сообщений электронной почты в FoxPro-таблицу. Чем не устраивает такая возможность?

Энди: Ах да, это так, но, к сожалению, ты не можешь с помощью этого мастера добраться до всех данных. Некоторые «жизненно» важные элементы информации (включая дату и любые включения) просто недоступны.

Марсиа: Хорошо, если тебе необходимо всего лишь импортировать всю твою папку InBox в набор таблиц Visual FoxPro, тогда задача, безусловно, решается достаточно просто путем использования технологии Outlook Automation. Но прежде чем мы сможем начать разговор о технологии Automation, нам необходимо принять решение относительно структуры таблиц в Visual FoxPro.

Энди: Но мы не можем определить эту структуру до тех пор, пока нам неизвестно, какие именно данные мы должны хранить в FoxPro-таблицах.

Марсиа: Это мы уже знаем. Нам необходимы поля для даты получения сообщения, адреса электронной почты отправителя, темы, содержимого сообщения и всех вложений.

Энди: Это все? Но я предполагаю, что для вложений мы должны добавить дочернюю таблицу (в конце концов, одно сообщение может иметь несколько вложений), а это означает добавление поля ключа для привязки вложения к его родительскому сообщению. Все это должна обеспечить структура, представленная на рис. 1, как ты считаешь?

Марсиа: Да, предложенная структура должна прекрасно справиться с поставленной задачей. Итак, осталось только написать программный код для извлечения информации из приложения Outlook.

Энди: Признаюсь, что я не очень понимаю, с чего начать.

Марсиа: Первым делом надо создать экземпляр объекта приложения Outlook, и для этого — так же как и в случае со всеми остальными серверами Office Automation — ты можешь просто использовать идентификатор .Application во встроенной в Visual FoxPro функции CreateObject():

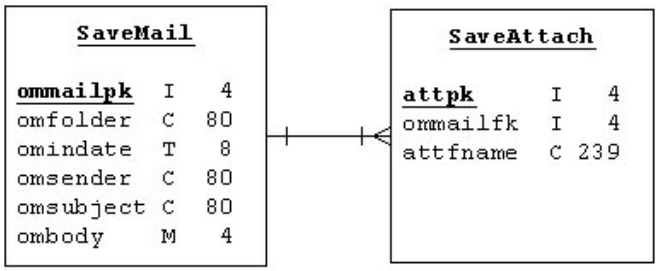

Рис. 1. Структура таблицы для хранения почтовых сообщений Outlook.

```
oOutlook = CREATEOBJECT( «outlook.application» )
```

Энди: Ага! Это достаточно просто. Так, а как мы добираемся до данных? Позволь, я выскажу предположение: мы это делаем, используя коллекцию Items объекта приложения и выполняя поиск элементов «Mail» в этой коллекции?

Марсиа: Можно было бы простить тебе такой ход мыслей, но приложение Outlook несколько отличается в этом смысле от остальных серверов Office. На самом деле, ты не можешь получить непосредственный доступ к данным и должен сначала создать объект NameSpace, который действует как шлюз. Объект NameSpace обеспечивает методы для входа «в» и выхода «из» источника данных, а также некоторые дополнительные специфичные для указанного источника данных методы. В настоящее время приложение Outlook поддерживает только один источник данных — «MAPI», и объект NameSpace для этого источника предоставляет, помимо прочих, методы для доступа непосредственно к личным папкам приложения Outlook.

Энди: Выбор из одного возможного варианта выглядит несколько глупо, но полагаю это сделано для того, чтобы обеспечить возможность дальнейшей разработки (хотя, учитывая среднюю продолжительность жизни любой конкретной технологии, эти намерения представляются как-то слишком уж оптимистичными). Следовательно, затем мы должны создать объект NameSpace для источника данных MAPI:

```
oNameSpace = oOutlook.GetNameSpace( «MAPI» )
```

Марсиа: Совершенно верно. Теперь мы можем получить ссылку на любую из «личных папок» приложения Outlook.

Энди: Я думал, нас интересовала электронная почта, а не личные сведения?

Марсиа: Папки в приложении Outlook организованы иерархически, и корневая вершина этой иерархии называется «Personal Folders» (полагаю, такое название соответствует шаблону, согласно которому твои каталоги, создаваемые по умолчанию в ОС Windows, называются «My Documents»). Если ты взглянешь на собственный экземпляр приложения Outlook, то можешь видеть, что эта вершина является корневой для всей структуры папок.

Энди: Ладно, и поскольку нас интересует электронная почта из папки InBox, я допускаю, что InBox — это одна из вложенных папок внутри иерархии с указанным корнем. Есть ли там еще что-нибудь?

Марсиа: Это зависит от того, добавлял ли ты какие-либо пользовательские папки или нет, но по умолчанию приложение Outlook создаст папки Deleted Items, OutBox, Sent Mail, InBox, Calendar, Contacts, Journal, Notes, Tasks и Drafts.

Энди: По-видимому, как только мы обзавелись ссылкой на объект NameSpace, мы можем одинаковым образом работать с любой из перечисленных папок. Следовательно, способ, с помощью которого мы добираемся до электронной почты, хранящейся в папке InBox, будет, в принципе, применим к содержимому любой другой папки (будучи модифицирован, разумеется, для обработки содержимого различных типов).

Марсиа: И да, и нет! На самом высоком уровне это утверждение справедливо: у всех папок есть коллекция Items. Но сами по себе свойства отдельных элементов из этих коллекций зависят от папки. Так, элемент Mail Item имеет свойство SenderName, а у элемента Contact Item его нет.

Энди: Могу я воспользоваться утилитой Object Browser, чтобы изучить эти вещи? А если нет, то что тогда делать?

Марсиа: Что ж, ты мог бы воспользоваться этой утилитой. Однако, когда имеешь дело с такими иерархическими структурами, проще, я считаю, воспользоваться технологией IntelliSense для выяснения того, как получить нужную мне информацию. Например, чтобы добраться до папки InBox (как только у нас появляется ссылка на объект NameSpace), мы можем начать перебирать объекты до тех пор, пока не найдем тот, который носит имя InBox. Сначала мы получаем ссылку на корневую папку (Personal Folders) и определяем, сколько в ней есть вложенных папок (по умолчанию там имеется 10 папок, которые мы перечислили ранее):

```
oRootFolder = oNameSpace.Folders(1)
? oRootFolder.Folders.Count
```

Энди: Это представляется достаточно разумным, но как мы узнаем, что есть что? Перебирать все объекты и проверять их имена? Полагаю, мы могли бы использовать вот такой программный код:

```
FOR lnCnt = 1 TO oRootFolder.Folders.Count
   IF LOWER( oRootFolder.Folders[lnCnt].Name ) = «inbox»
     OInBox = oRootFolder.Folders[lnCnt]
     EXIT
   ENDIF
 NEXT
```

Марсиа: Этот код работал бы, но есть более короткий вариант. Если ты откроешь с помощью утилиты Object Browser объектную библиотеку типов для приложения Outlook, то обнаружишь в разделе Enums запись с именем olDefaultFolders. Эта запись определяет для каждой из предусмотренных по умолчанию папок константы, которые могут быть переданы непосредственно в метод GetDefaultFolder() объекта NameSpace. Итак, чтобы получить объектную ссылку на папку InBox, нам, по существу, необходимо проделать только следующее:

```
oInBox = oNameSpace.GetDefaultFolder( 6 )
```

Энди: Вот так намного лучше! Конечно, этот способ не будет работать применительно к пользовательским папкам (тебе действительно придется все их перебрать). Ладно, итак мы (наконец) очутились в папке InBox. Что теперь?

Марсиа: Итак, как мы сказали ранее, у объекта InBox есть коллекция Items. Каждый элемент в этой коллекции — это «объект сообщения» (более точно, каждый из этих элементов — это экземпляр объекта класса MailItem), и в данной коллекции имеется по одному элементу для каждого сообщения электронной почты, хранящегося в папке InBox. Поэтому, чтобы извлечь данные, мы просто организуем циклический просмотр указанной коллекции и получим значения соответствующих свойств каждого объекта сообщения.

Энди: А имена свойств мы можем получить либо «захватив» ссылку на объект сообщения и воспользовавшись технологией IntelliSense для исследования этого объекта, либо используя утилиту Object Browser для непосредственного просмотра свойств этого класса. Однако, в таком случае использование утилиты Object Browser представляется лучшим выбором, поскольку при этом можно видеть все свойства сразу.

Марсиа: Да, и по сути нам необходимы лишь несколько ключевых свойств, как показано в таблице 1.

Таблица 1. Свойства элемента E-mail, которые необходимо получить.

Свойство	Содержит	Тип данных
SenderName	e-mail-адрес отправителя	Character
ReceivedTime	Дата/время получения сообщения электронной почты	DateTime
Subject	Строка Subject (тема) сообщения электронной почты	Character
Body	Содержание сообщения	Character

Энди: Подожди минутку. Как насчет вложений? Я не вижу свойства, предназначенного для вложений.

Марсиа: Объект message имеет свойство Attachments, но в действительности это свойство является коллекцией, поэтому мы не можем обращаться с ним так же, как обращаемся с другими свойствами объекта message. Тут ничего не сократишь. Нам необходимо проверить свойство Count коллекции Attachments для каждого объекта message. Если его значение больше нуля, мы итеративно перебираем все объекты attachment из этой коллекции.

Энди: Тогда для вложений нам необходимо запомнить только имя файла, верно?

Марсиа: И опять-таки, ответом является «и да, и нет»! Да, нам необходимо хранить имя файла, но нет, потому что до тех пор, пока мы на самом деле не «сохраним» вложение как конкретный файл, это вложение, по сути дела, не существует как независимая сущность. Объект Attachment имеет метод SaveAsFile(), к которому мы обращаемся, передавая ему в качестве параметра полный путь доступа к этому файлу. Затем мы можем добавить запись в нашу таблицу вложений, чтобы сохранить как имя файла, так и внешний ключ к родительскому сообщению.

Энди: А, я понимаю. Осталось только заключить все это в оболочку в виде единственного класса, который будет извлекать все необходимые элементы.

Марсиа: Да, и этот программный код находится на дискете. Тебе надо только создать объект класса sesOutlook и обратиться к его методу ReadMail(), чтобы сохранить вложения и заполнить таблицы. Впрочем, поскольку мы рассмотрели все основные принципы того, как этот класс работает, нетрудно было бы дополнить его таким образом, чтобы он мог работать более избирательно.

Энди: О да, могу представить, как бы я мог его улучшить так, чтобы иметь возможность сформировать список доступных папок и выбрать содержимое только указанной папки. Полагаю, мы могли бы также расширить функции этого класса, обеспечив также возможность извлекать только определенные сообщения электронной почты. Может быть, только те сообщения, которые были получены после определенной даты, от конкретного человека, или те, которые имеют указанную тему.

Марсиа: Но все эти тонкости, я думаю, можно без опаски оставить на усмотрение наших высокоинтеллектуальных и в высшей степени способных читателей. Ты полагаешь, твой анонимный корреспондент будет доволен?

Энди: Уже по одному тому факту, что мы читаем эту статью, можно, я думаю, предположить: он согласен с нашим взглядом на приложение Outlook.

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

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

FoxTalk *русское издание*

Печатается ежемесячно

Учредитель и издатель:

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

FoxTalk (русское издание) индекс 72495

Объединенный каталог индекс 45007

Журнал для FoxPro-программистов.

FoxTalk (русское издание) индекс 72496

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

FoxTalk (русское издание) индекс 72497

Подписка на старые номера журнала FoxTalk.

Библиотека программиста индексы 72769,
72490, 72491, 47771, 80375, 82841

Книги компьютерной тематики по последним версиям
популярных программных продуктов.

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

Подписано в печать 05/10/03. Формат 60x90 1/8. Тираж 300 экз.