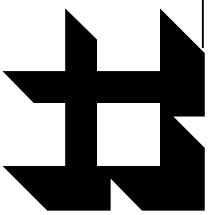

ГЛАВА 3

Проектирование приложений

Что представляет собой ваше будущее приложение? Это, как правило, набор файлов: главный (стартовый) файл программы, базы данных, таблицы, формы, отчеты, меню, файлы графических изображений, программные модули, классы и т. д.

Прежде чем приступить к созданию приложения Visual FoxPro, мы должны четко уяснить себе организацию самого приложения и входящих в него компонентов.

Менеджер проекта и определение структуры приложения

Все компоненты (файлы), участвующие в создании приложения, объединены в проект. Проект — это контейнер, в котором хранится все то, из чего состоит приложение. Сборкой всех файлов приложения в единый проект занимается Менеджер проекта — это компонент, отображающий файлы проекта и имеющий средства для манипуляции этими файлами и проектом в целом. В Менеджере проектов в виде иерархического дерева отображаются различные файлы, входящие в проект: базы данных, таблицы, формы, отчеты, библиотеки классов, программы, меню и т. д. Использование Менеджера проекта упрощает разработку приложения, т. к. используемые в нем элементы приложения размещаются по разделам. Кроме того, запоминается расположение каждого включенного в проект элемента. Добавленные в проект файлы можно редактировать, удалять, а те из них, которые предназначены для запуска (формы, программы, запросы, отчеты и т. д.), можно запустить на выполнение. После завершения создания отдельных файлов проекта строится само приложение — APP- или EXE-модуль. Также может быть создан файл DLL. Если в проекте имеется файл с классом `OLEPUBLIC`, то при генерации EXE-модуля созданные серверы (объекты, доступные клиентам автоматизации) отображаются в списке **Server Classes** вкладки **Servers** диалогового окна **Project Information**.

Проект сохраняется в виде таблицы VFP, имеющей расширение pjx (плюс дополнительный файл pjt для хранения memo-полей). Таблица проекта может быть открыта

командой USE, при этом расширение файла должно быть обязательно указано в команде.

При необходимости проект очищается (**Project** | **Clean Up Project**) или обновляется (<F5> или **Project** | **Refresh**).

В первом случае выполняется метод CleanUp проекта, который удаляет записи с пометкой удаления и упаковывает memo-поля, во втором — Refresh, который обновляет окно Менеджера проектов, приводя его в соответствие с таблицей проекта.

Работа Менеджера проектов зависит от того, как он настроен. Настройка производится в окне **Projects** (**Tools** | **Option** | **Projects**).

Вкладка *Projects* — проекты

Вкладка предназначена для настройки диспетчера проектов, т. е. для поддержки и компиляции приложений (рис. 3.1).

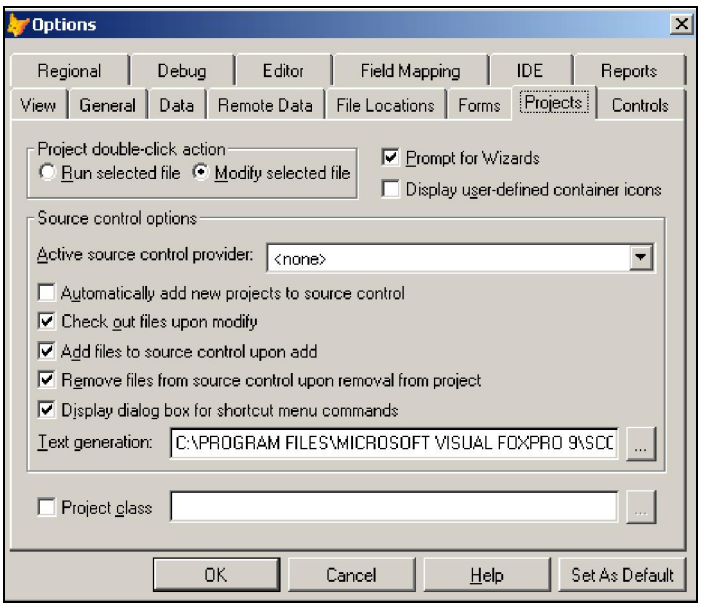

Рис. 3.1. Вкладка **Projects** диалогового окна **Options**

Опция **Project double-click action** определяет, какие действия будут производиться с проектом при двойном щелчке мыши на нем: если выбирается опция **Run selected file**, то проект запускается на выполнение. В случае выбора опции **Modify selected file** файл будет открыт для редактирования. Как вы понимаете, редактировать приходится гораздо чаще.

Если установлен флаг **Prompt for Wizards** (Предлагать услуги соответствующих мастеров), то VFP автоматически предлагает услуги мастера при создании нового файла из окна диспетчера проектов. Иначе запускается конструктор соответствующего объекта.

Если установить флаг **Display user-defined container icons**, VFP будет отображать в окне диспетчера проектов значки контейнеров, определенных пользователем.

Группа переключателей **Source control options** (Параметры управления исходными материалами) предназначена для пользователей пакета Microsoft Visual SourceSafe (VSS). Этот пакет позволяет синхронизировать работу группы разработчиков, не разрешая заменять фрагменты кода разными разработчиками, давая возможность восстанавливать более старые версии программ.

Флажок **Automatically add new projects to source control** (Автоматически добавлять новые проекты в систему управления исходными материалами) добавляет новые проекты в VSS автоматически. Разработчики редко пользуются этим флажком.

Как правило, они предпочитают добавлять данные о новом проекте из меню **Projects | Add Project to Source Control**.

Флажок **Check out files upon modify** (Отметить файлы при модификации) дает возможность автоматически вызывать Visual SourceSafe и помечать файл. Если опция отключена, то файл открывается только для чтения, и, чтобы иметь возможность его модифицировать, вам придется вручную помечать его перед попыткой открыть.

Флажок **Add files to source control upon add** (Добавлять файлы в систему управления исходными материалами при добавлении в проект) — новые файлы проекта автоматически помещаются в Visual SourceSafe.

Флажок **Remove files from source control upon removal from project** (Удалять файлы проекта из системы управления исходными материалами) — удаляет ссылки на файл из базы данных Visual SourceSafe, физически же файлы с диска не удаляются.

Флажок **Display dialog box for shortcut menu commands** (Отображать диалоговое окно для команд контекстного меню) позволяет выполнять команды VSS из контекстного меню проекта сразу для нескольких файлов.

Список **Text generation** (Воспроизведение текста) позволяет идентифицировать файл, который сохраняет целостность информации между VSS и VFP. В частности, этот файл создает программный код для экранов, меню, отчетов и т. д.

Установкой флажка **Project class** можно выбрать и указать класс, который будет использоваться по умолчанию при разработке нового проекта.

Меню Менеджера проектов

При работе с проектом в главном меню Visual FoxPro появляется дополнительный пункт меню **Project** (рис. 3.2, табл. 3.1).

Таблица 3.1. *Команды меню* ***Project***

Команды меню	Описание
New File (Новый файл)	Создает новый файл, который будет добавлен в проект
Add File (Добавить файл)	Добавляет в проект уже созданные файлы

Таблица 3.1 *(окончание)*

Команды меню	Описание
Modify File (Изменить файл)	Открывает указанный файл для редактирования
Browse File (Просмотр файла)	Открывает таблицу для просмотра
Preview File (Предварительный просмотр файлов)	Открывает файл в окне предварительного просмотра
Run File (Выполнить файл)	Запускает файл на выполнение
Remove File (Удалить файл)	Удаляет файл из проекта
Rename File (Переименовать файл)	Переименовывает файл
Exclude (Исключить)	Исключает файл из проекта
Include (Включить)	Включает файл в проект
Set Main (Установить как главный модуль)	Устанавливает файл в качестве основного модуля проекта
Edit Description (Правка описания)	Открывает окно редактирования описания файла
Project Info (Информация проекта)	Отображает информацию о проекте
Errors (Ошибки)	Показывает ошибки, возникшие при построении проекта
Build (Построить)	Строит проект заново
Refresh (Обновить)	Обновляет информацию в окне проекта
Clean Up Project (Очистить проект)	Очищает проект, исключая из него удаленные файлы

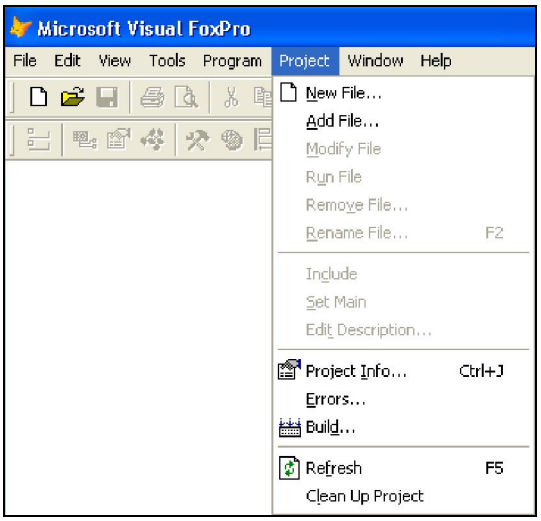

Рис. 3.2. Дополнительный пункт главного меню для работы с проектом

Компоненты приложения (типы файлов), их размещение

Файлы в проекте систематизированы по видам. Вкладка **All** отображает все файлы. Кроме того, список файлов выводится на вкладке **Files** диалогового окна **Project Information**, открываемого при нажатии клавиш <Ctrl>+<J> или <Alt>+<Enter> при выполнении команды меню **Project | Project Info**.

В верхней части окна проекта находятся вкладки, которые управляют отображением определенных типов файлов проекта (табл. 3.2).

Таблица 3.2. *Вкладки окна* ***Project Manager***

Вкладка	**Отображаемые файлы**
All (Все)	Все файлы
Data (Данные)	Базы данных, свободные таблицы, представления, хранимые процедуры, запросы
Documents (Документы)	Формы, отчеты, этикетки
Classes (Классы)	Классы
Code (Коды)	Программы и библиотеки
Other (Остальные)	Меню, текстовые и графические файлы

Немаловажный вопрос — где расположить файлы проекта. Крайне нежелательно размещать все файлы в одном каталоге, тем более в каталоге Microsoft Visual FoxPro.

Как правило, для каждого нового проекта создается свой каталог, а в нем — подкаталоги. Какие именно подкаталоги — зависит от решаемой задачи.

Нежелательно в именах каталогов, где расположен ваш проект, использовать пробелы, ибо они приведут к некоторому усложнению программирования (необходимы будут дополнительные кавычки во всех путях доступа). Будет замечательно, если название вашего каталога проекта уместится в 8 латинских символов, как это принято в DOS.

Теперь перейдем к подкаталогам. Общая практика такова: выделяется отдельный подкаталог для баз данных и таблиц, например, `DATA` (все названия — условные, вы можете дать любые свои), для форм и этикеток — `FORMS`, для программ — `PRG`, для классов и библиотек — `CLASS`, для файлов отчетов — `REPORTS`. Некоторые программисты создают отдельные подкаталоги для справочников или документов (если их много, то это имеет смысл). Главное — чтобы файлы не находились в корневом сегменте каталога. Очень важно выделить отдельный подкаталог для таблиц и баз данных, это позволит организовать сохранение данных. Также важно сохранять и формы, отчеты, программы и т. д. Вы же не хотите потерять все вами созданное? Итак, после размещения файлов по подкаталогам мы получим следующую структуру вашего каталога проекта:

- `C:\My_Project` — рабочий каталог проекта;
- `C:\My_Project\Class` — файлы VCT, VCX;
- `C:\My_Project\Data` — файлы DBC, DCT, DCX, DBF, FPT, CDX;
- `C:\My_Project\Forms` — файлы SCX, SCT;
- `C:\My_Project\Menu` — файлы MNT, MNX, MPR, MPX;
- `C:\My_Project\Prg` — файлы PRG, FXP;
- `C:\My_Project\Reports` — файлы FRT, FRX.

Что же будет находиться в корневом сегменте рабочего каталога?

Там должны находиться следующие файлы:

- файл проекта (My_project.pjt, My_project.pjx);
- файл Config.fpw;
- файл ресурсов — FoxUser.dbf, FoxUser.fpt;
- готовый EXE-файл, т. е. уже построенный проект.

Важный вопрос: как обратиться к папке проекта, а также к файлам, расположенным в подкаталогах? Тут сыграет свою немаловажную роль строка `PATH` из файла Config.fpw. Если такая строка указана, то обращаться к файлам можно без указания путей, например:

```
MODI COMMAND Main.prg или
USE Table1.dbf
```

Программа и таблица будут открыты, несмотря на то, что программа находится в подкаталоге PRG, а таблица — в подкаталоге DATA.

Бывают, хотя и редко, ситуации, когда простого указания путей доступа оказывается недостаточно. Например, когда вы одновременно работаете с несколькими одноименными таблицами, принадлежащими разным базам данных. В таком случае пути придется прописать полностью.

Существует и другой взгляд на эту проблему. Некоторые программисты не используют строку PATH из файла Config.fp, указывая в файле конфигурации лишь кодовую страницу. А определение пути к файлам дается в главной процедуре проекта:

```
lcPath=JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
SET PATH TO FORMS, MENUS, DATA...
```

Но тем и хорош Visual FoxPro, что может предложить множество путей решения одной и той же проблемы.

По умолчанию в готовый EXE-файл включаются все файлы, необходимые для работы приложения — программы, формы, меню, библиотеки, классы и т. д., кроме файлов баз данных и таблиц (такие файлы можно узнать по перечеркнутому кружочку возле имени файла в Менеджере проекта). Все включенные в проект файлы после компиляции станут немодифицируемыми (доступными только для чтения). Понятно, что файлы таблиц (*.dbf) включать не стоит, поскольку они подвергаются постоянным операциям изменения-дополнения-исключения данных, но иногда бывают случаи, когда необходимо скрыть какие-либо статические данные от пользователя, но обеспечить доступ к ним самой программы. Для такого случая лучшего места, чем "внутренности" EXE-файла, просто нет.

Еще один немаловажный вопрос касается отчетов. Включать их в EXE-файл или не включать? Если отчетов много, и они подвергаются частым изменениям, есть смысл исключить эти отчеты из конечного файла и поставлять их вместе с приложением в отдельном подкаталоге. В любом случае решение о том, какие файлы исключить, а какие оставить, за вами.

Узлы Менеджера проектов

Для удобства работы с проектом разработчиками организована иерархическая модель Менеджера проектов (рис. 3.3).

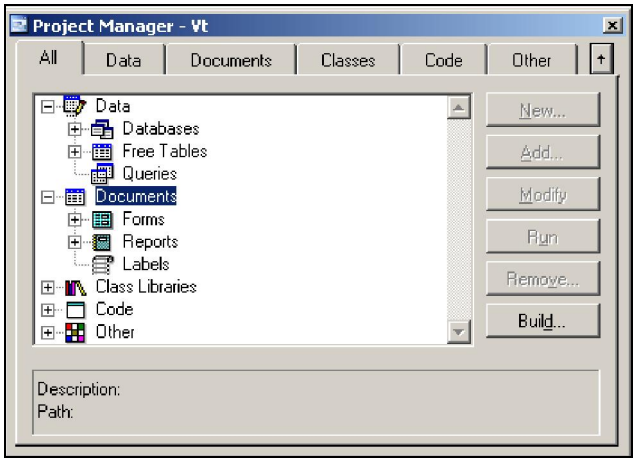

Рис. 3.3. Иерархическое дерево Менеджера проектов

В верхнем уровне иерархии отображаются узлы Менеджера: `Data`, `Documents`, `Classes`, `Code`, `Other`. Каждый уровень иерархии может находиться в свернутом и развернутом состоянии. Если узел свернут, слева от его названия располагается знак "+". Знак "+" означает, что узел может быть развернут. При раскрытии уровня знак "+" заменяется на "–". На следующем уровне располагаются файлы данной категории. Например, для узла `Data` узлами второго уровня являются `Databases`, `Free Tables` и `Queries`. Для того чтобы свернуть соответствующий узел, установите курсор мыши на знак "–" и нажмите кнопку мыши.

Узел *Data*

На вкладке **Data** (рис. 3.4) размещаются базы данных, свободные таблицы, запросы. При щелчке на иконке **Databases** разворачивается список баз данных. Если развернуть базу данных, то мы получим список включенных в нее таблиц, представлений, подключений ODBC и хранимых процедур.

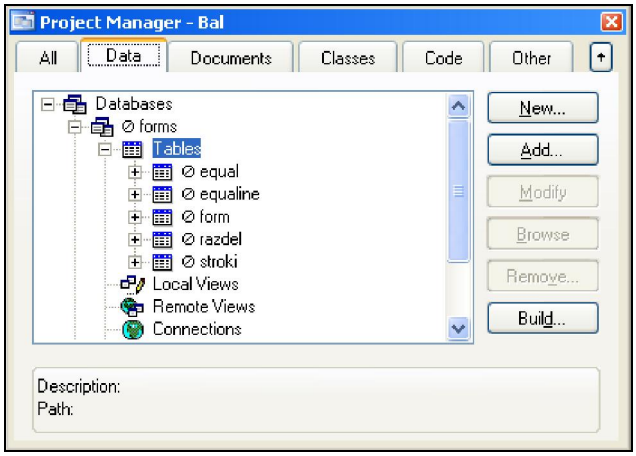

Рис. 3.4. Вкладка **Data** Менеджера проектов

Щелчок на имени какой-либо таблицы базы данных позволяет увидеть список ее полей и индексов. Если возле иконки с именем таблицы вы увидите значок в виде перечеркнутого круга (Exclude Icon), это означает, что эта таблица не может быть включена в компилированные файлы с расширением app, exe или dll. Чтобы установить или снять признак `Exclude Icon`, в главном меню имеются команды **Project | Include** и **Project | Exclude**.

Узел *Documents*

На вкладке **Documents** (рис. 3.5) располагаются формы, отчеты и этикетки.

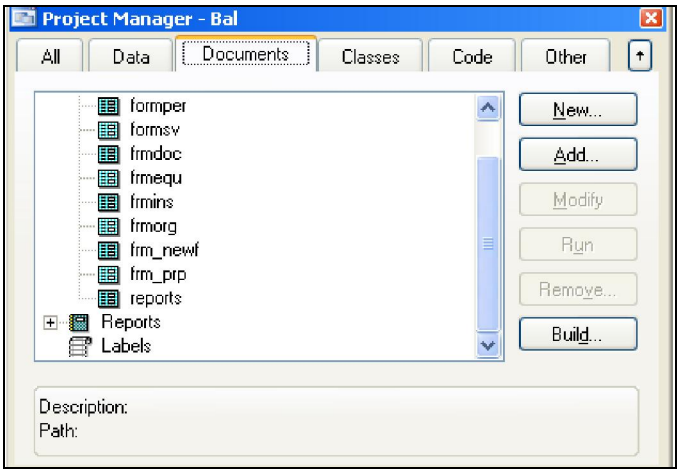

Рис. 3.5. Вкладка **Documents** Менеджера проектов

Узел *Class Libraries*

На этой вкладке (рис. 3.6) располагаются файлы библиотек классов. Если раскрыть библиотеку, можно увидеть список входящих в нее классов.

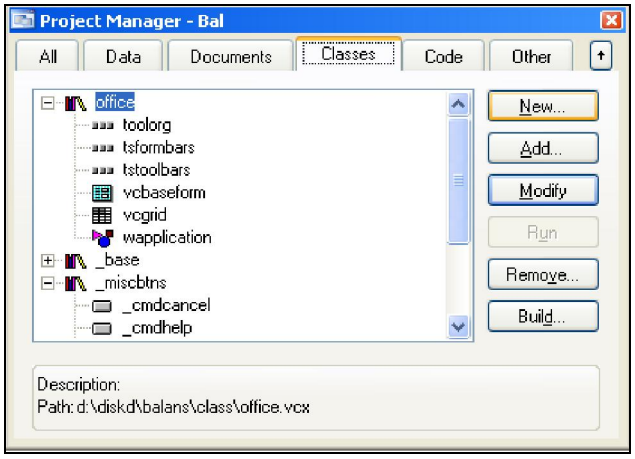

Рис. 3.6. Вкладка **Classes**

Узел *Code*

На вкладке **Code** (рис. 3.7) размещаются файлы программ, API-библиотек и приложений, скомпонованных в APP-файлы, которые могут быть включены в итоговый EXE-файл.

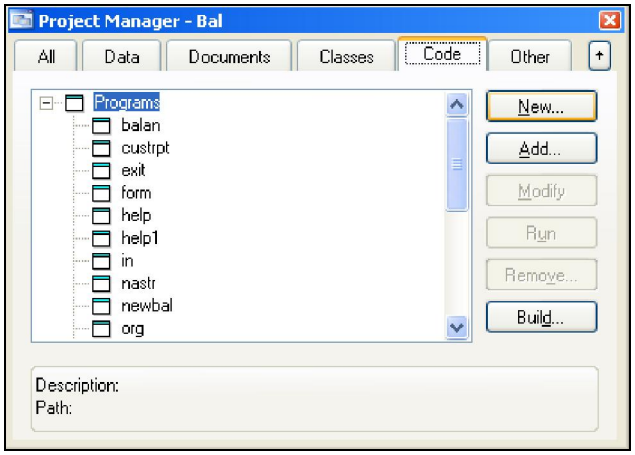

Рис. 3.7. Вкладка **Code**

Создание компонентов проекта в Менеджере проектов

После знакомства со вкладками Менеджера проектов становится ясно, что его функции не ограничиваются простым хранением файлов и представлением в удобном для просмотра виде. Менеджер проекта имеет 6 кнопок, которые позволяют добавлять, удалять, редактировать и даже запускать на выполнение указанные компоненты (табл. 3.3).

Таблица 3.3. *Функции кнопок Менеджера проектов*

Наименование	Описание
New	Позволяет создать и добавить в проект новый компонент того типа, который выделен в списке
Add	Позволяет добавить в проект уже имеющийся компонент
Modify	Позволяет редактировать выделенный компонент
Run	Позволяет запустить на выполнение указанный компонент
Remove	Позволяет исключить из проекта выделенный компонент
Build	Позволяет скомпилировать проект

Процесс создания файла приложения можно начать прямо из окна Менеджера проектов. Для этого достаточно установить курсор на нужный компонент и нажать кнопку **New**. Рассмотрим, например, как создать файл программы. Раскроем узел `Code` Менеджера проектов и установим курсор на уровень `Program`. Нажмем кнопку **New** — и увидим на экране окно редактора программного кода. Введем в этом окне простую команду

```
MessageBox('Hello, world!')
```

и сохраним ее либо используя команду главного меню **File** | **Save As**, либо нажав комбинацию клавиш <Ctrl>+<W>. Можно просто закрыть окно редактора или конструктора, в случае, если были произведены изменения, последует запрос на подтверждение сохранения. По умолчанию Менеджер проектов присваивает файлу имя `Program1`, но вы можете изменить его в диалоговом окне. Вы можете запустить свою первую программу, установив на нее курсор мыши и нажав кнопку **Run** Менеджера проектов.

Точно так же происходит процесс создания новых форм, отчетов, меню и т. д. При этом на экран будет вызвано диалоговое окно, и вам будет предложено использовать мастер или конструктор. Созданные в Менеджере проектов файлы автоматически включаются в проект.

Главный (стартовый) файл приложения

Итак, мы имеем n-е количество различных файлов, из которых хотим построить свое приложение. Мы разложили их "по полочкам", т. е. включили их в проект. Теперь нужно определить главный файл приложения, с которого начнется выполнение программы. Такой главный файл еще называют точкой входа в приложение. Определить этот файл можно только в Менеджере проектов.

Главным (стартовым) файлом может быть:

- программный файл (PRG);
- форма (SCX);
- меню.

По умолчанию главным файлом проекта становится файл, включаемый в проект первым. Этот файл выделяется в проекте полужирным шрифтом.

Если вы по ошибке указали первым не главный файл проекта, то ошибку можно исправить, щелкнув правой кнопкой мыши по нужному файлу и выбрав в появившемся меню пункт **Set Main**. Выбранный файл станет главным, а ошибочно выбранный перестанет быть главным, и выделение с него будет снято. В пределах одного проекта может быть только один главный (стартовый) файл. Если главный файл не будет указан, это вызовет ошибку компиляции с сообщением `Cannot build without a main program`.

Итак, вы стоите перед выбором — меню, форма или программный файл? Меню можно сразу исключить из рассмотрения, потому что оно требует генерации. Многие начинающие программисты рассуждают так: у меня в проекте только одна форма, поэтому укажу-ка я в качестве главного файла форму. Это допустимо для маленьких проектов, не требующих настройки и доработки. Как правило, объем программ быстро растет, и многое из того, что можно сделать в главном файле — программе, нельзя выполнить в форме. Поэтому *всегда указывайте в качестве главного (стартового) файла программный файл*.

Главная процедура приложения и процедурные файлы

Мы указали в качестве главного (стартового) файла программный файл (иногда его называют главной процедурой приложения). Главная процедура приложения должна содержать команды настройки среды, объявление глобальных переменных, запуск главной формы или меню приложения, глобальный обработчик ошибок и выход из программы.

Команды настройки среды — это те самые SET-команды, о которых вы узнали из *главы 2*. Основная часть этих команд уже настроена так, как это удобно разработчику. Но при передаче готового проекта заказчику нужны совсем другие настройки. Например, команда `SET ESCAPE` по умолчанию установлена в `ON`, что позволяет остановить выполнение программы во время отладки. Но в готовом проекте такого быть

не должно. Можно установить различные настройки для режима отладки и режима выполнения, но для этого надо знать, в каком режиме работает проект. Самый простой способ это выяснить — использовать свойство `_VFP.StartMode`. Если `_VFP.StartMode=0`, то это режим отладки. Соответственно, в главной процедуре должна появиться конструкция `IF...ENDIF`:

```
IF _VFPStartMode=0
    SET ESCAPE ON
ELSE
   SET ESCAPE OFF
ENDIF
```

Кроме `SET ESCAPE`, важными являются еще некоторые настройки:

- `SET CENTURY` — определяет, будет ли показан век в датах;
- `SET DELETED` — определяет, будут ли доступны для просмотра и обработки записи, помеченные на удаление;
- `SET EXCLUSIVE` — определяет, в каком режиме будут открываться таблицы VFP — для монопольного использования или для совместного;
- `SET MULTILOCKS` — определяет, могут ли команды `LOCK()` и `RLOCK()` блокировать сразу несколько записей.

Существует еще одна проблема с настройками. Мало, оказывается, установить их в начале главной процедуры так, как этого требует приложение. Нужно еще выполнить один из основных законов программирования: "Не наследи!". То есть после окончания работы приложения нужно вернуть настройки в то состояние, в котором они находились до его запуска. Поэтому принцип работы с настройками должен быть таков:

1. Проверяем текущее состояние настройки.
2. Если текущее состояние настройки отличается от нужного нам значения, то сохраняем старое значение настройки и устанавливаем новое.
3. По окончании работы восстанавливаем настройки.

Это можно сделать, например, так:

```
LOCAL lcOldTalk
IF SET('TALK') = 'ON'
  SET TALK OFF
  lcOldTalk = 'ON'
ELSE
  lcOldTalk = 'OFF'
ENDIF
```

Объявление глобальных переменных — вторая важная функция главной процедуры. Глобальные переменные действуют не только в главной процедуре, но и во всех других процедурах, вызываемых из главной. Они не освобождаются автоматически по ее завершении. Обратите внимание на то, что в нашем примере объявлена как `LOCAL`.

Дело в том, что если не объявить переменные, то по умолчанию они получат область видимости PRIVATE. А для переменных главной процедуры это равнозначно объявлению их как PUBLIC, поскольку они будут видны во всех вызванных формах и процедурах.

Запуск главной формы или меню приложения тоже производится из главной процедуры. То есть для запуска главной формы вы должны указать в главной процедуре команду

```
DO FORM My_Form
```

Стоит сказать, что в Visual FoxPro существуют две идеологии построения приложений:

- на базе главного окна Visual FoxPro (SCREEN);
- на базе "As Top-Level" форм (т. е. на базе так называемой формы верхнего уровня).

Построение приложения на базе главного окна FoxPro (SCREEN) предполагает, что пользовательское приложение (все его формы, отчеты и пр.) будет показано в главном окне Visual FoxPro. Разумеется, при этом придется скрыть системное меню и панель инструментов и заменить их своими.

Построение приложения на базе "As Top-Level" форм предполагает, что в качестве главного окна будет выступать созданное разработчиком окно со свойством ShowWindow = 2 — "As Top-Level form".

В случае использования формы верхнего уровня главное окно Visual FoxPro должно быть закрыто. Самый простой способ это сделать — в файле Config.fpw указать строку SCREEN=OFF.

Что лучше использовать для построения приложения: главное окно Visual FoxPro или форму верхнего уровня — вам придется решить самостоятельно. Некоторые программисты используют главное окно, предварительно убрав главное меню и панель инструментов, другие — As Top-Level form, третьи используют главное окно "As is", т. е. в том виде, в котором оно существует. Мы будем исходить из того, что приложение строится на базе формы верхнего уровня.

Нужно учесть еще один важный момент. После команды DO FORM My_Form обязательно должна быть выполнена команда READ EVENTS, которая указывает Visual FoxPro, где ему остановиться и ожидать реакции пользователя. При отсутствии этой команды после запуска готового EXE-файла окно будет мелькать на экране и сразу же закрываться. Такого эффекта не наблюдается при отладке программы, потому что во время отладки открыта среда Visual FoxPro. Итак, запуск формы в вашей главной процедуре должен выглядеть так:

```
DO FORM My_Form
READ EVENTS
```

Специально заостряем на этом внимание новичков, потому что при практическом программировании это вызывает многочисленные вопросы.

Одновременно может быть активна только одна команда READ EVENTS, остальные игнорируются. Для отмены действия READ EVENTS нужно использовать команду CLEAR EVENTS. По этой команде будет отменено действие команды READ EVENTS и выполнение перейдет на команду, следующую за командой READ EVENTS. То есть, если вы дали команду CLEAR EVENTS в какой-либо процедуре, то все то, что стоит следом за этой командой, вообще никогда не будет выполнено.

Так, где же давать команду CLEAR EVENTS? Конечно же, в специальном пункте меню **Выход**.

Если вы создаете приложение на базе формы верхнего уровня, то команду CLEAR EVENTS надо давать в событии UNLOAD вашей главной формы.

В результате получается такая логика выполнения программы:

1. Запускается главный (стартовый) файл.
2. Активизируется основное меню.
3. По команде READ EVENTS организуется "точка останова" для ожидания действий пользователя.
4. Команда CLEAR EVENTS прекращает действие команды READ EVNETS, завершает выполнение главного (стартового) файла, что приводит к закрытию приложения Visual FoxPro.

О главном обработчике ошибок вы прочтете в *главе 14*.

Но это еще не все. Нужно грамотно организовать *выход из программы*. И не только в "штатной ситуации", когда пользователь завершает свою работу с программой, аккуратно нажимая кнопку **Выход**, но и при условии, когда пользователь панически нажимает для выхода все кнопки подряд, начиная с крестика в правом верхнем углу главного окна Visual FoxPro и заканчивая снятием приложения в Диспетчере задач Windows. А что — всего лишь догадаться нажать комбинацию клавиш <Ctrl>+ +<Shift>+<Esc>! Если до выхода пользователя была активна команда READ EVENTS, то на экране появится сообщение: Can't quit Visual FoxPro. Именно эта команда и вызывает такое сообщение об ошибке. Чтобы перехватить описанные события закрытия приложения, используется специальная настройка ON SHUTDOWN. Перед выходом из программы обычно выполняются некоторые действия: закрываются таблицы, базы данных, освобождаются глобальные переменные, восстанавливается системное меню и т. д. Для выполнения всего этого потребуется дополнительная процедура, например

```
ON SHUTDOWN DO CleanUp()
```

Бывают такие "уникальные" пользователи, которые способны 5 и более раз вызвать одно и то же приложение. Поэтому в главной процедуре хорошо бы поместить еще вызов процедуры, которая не позволяет вызвать приложение более одного раза.

Как видите, главная процедура начинает разрастаться. Число глобальных процедур в объектно-ориентированном программировании должно быть минимальным, в противном случае это считается плохим стилем программирования, поэтому часть процедур может быть размещена в дополнительных процедурных файлах. Они имеют

расширение prg, а к главному файлу они подключаются с помощью команды `SET PROCEDURE TO... <ADDITIVE>`. Например, `SET PROCEDURE TO My_Proc ADDITIVE`.

Ресурсы

Ресурсы — это одна из самых "толстых" частей проекта, поскольку включает в себя меню, текстовые файлы и графические изображения, которые, как правило, имеют большой размер. Располагаются эти файлы в узле `Other` Менеджера проектов.

Узел *Other*

На вкладке **Other** (рис. 3.8) расположены все те файлы, которые не относятся к описанным выше вкладкам: файлы графических изображений, меню, текстовые файлы. Сюда же можно включить файл Config.fpw, если вы не собираетесь его модифицировать у заказчика. Если вы его поместите в проект, пользователю в наборе файлов приложения он уже не будет нужен (но помните, что и внести корректировки на месте установки в него будет нельзя!). На вкладке **Other** удобно поместить BMP-файлы, которые вы использовали в своих экранных формах или в качестве иконок. Они после компиляции также окажутся в исполняемом модуле (отчего он изрядно "пополнеет"). А вот звуковые файлы WAV включать в проект нет смысла — они в EXE-файл не включаются и пользователю передаются в явном виде.

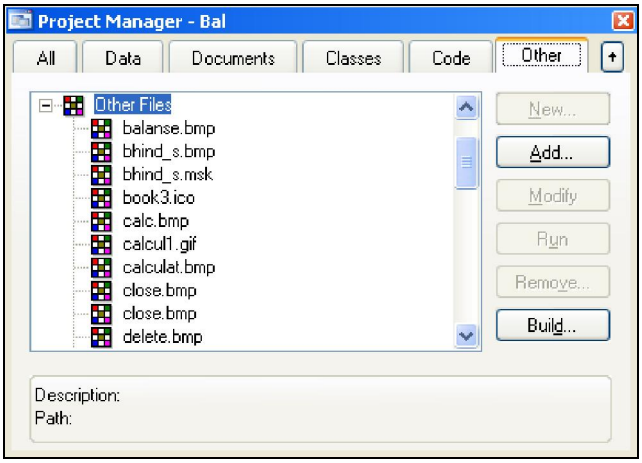

Рис. 3.8. Вкладка **Other**

Исполняемые файлы

Для того чтобы построить приложение, недостаточно включить в проект все файлы, его еще нужно скомпилировать. Это значит, что нужно щелкнуть на кнопке **Build** в

правой части Менеджера проектов. На экране появится диалоговое окно с несколькими опциями.

Из группы опций **Build Action** (рис. 3.9) необходимо выбрать одну:

- **Rebuild project** — файлы проекта просматриваются на предмет наличия синтаксических ошибок, нужные файлы компилируются (нужные — это те, у которых дата создания FXP-файла более ранняя, чем дата создания PRG-файла).
- **Application (app)** — строится проект, к нему привязываются все файлы, помеченные для включения в APP-файл. Для работы APP-файла на компьютере пользователя необходима установка Visual FoxPro. При построении APP компилятор VFP собирает приложение, откомпилированное в p-код, за счет чего приложение получается сравнительно небольшим в размере.
- **Win32 executable / COM server (exe)** — выполняются те же операции, что и при построении APP-файла, а затем дополнительно включается загрузчик времени выполнения (runtime loader) и заголовок выполняемого файла. Для работы EXE-файла на компьютере пользователя необходимо наличие нескольких библиотек Visual FoxPro. Каких именно?

VFP 9.0	msvcr71.dll, vfp9r.dll, vfp9rrus.dll, gdiplus.dll
VFP 8.0	msvcr70.dll, vfp8r.dll, vfp8rrus.dll, gdiplus.dll
VFP 7.0	msvcr70.dll, vfp7r.dll, vfp7rrus.dll
VFP 6.0	vfp6r.dll, vfp6renu.dll , vfp6rrus.dll
VFP 5.0	vfpole50.dll, vfpodbc.dll, vfp500.dll, vfp5rus.dll, Foxpro.int

Библиотеки времени выполнения (run-time library) должны располагаться в каталоге

```
..\Program Files\Common Files\Microsoft Shared\VFP.
```

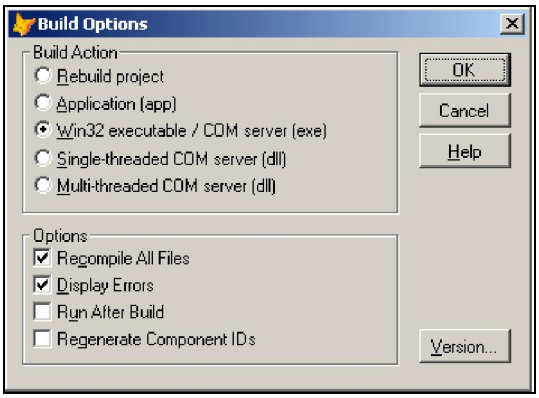

Рис. 3.9. Диалоговое окно компиляции проекта

- **Single-threaded COM server (dll)** (Однопоточный COM-сервер) и **Multi-threaded COM server (dll)** (Многопоточный COM-сервер) — создается файл динамически

связанной библиотеки (DLL). Другие приложения смогут использовать созданную DLL в качестве COM-сервера. DLL — это создание собственного приложения, оформленного в виде OLE-сервера. Такое приложение может предоставлять свои функции и данные другим приложениям, написанным на самых разных языках программирования, поддерживающих технологии OLE. Более подробно об этом вы можете прочесть в *главе 16*.

Проект может быть создан интерактивно и изменен в диспетчере проекта. Для этого кроме команд интерфейса **File** | **New** | **Project** и **File** | **Open** существуют команды `CREATE PROJECT` и `MODIFY PROJECT`. Программно проект можно создать, применив команду `BUILD PROJECT`.

Команда `CREATE PROJECT [<Имя файла> | ?] [NOWAIT] [SAVE]`
`[WINDOW <Имя окна1>] [IN [WINDOW] <Имя окна2> | IN SCREEN]`
`[NOSHOW][NOPROJECTHOOK]`
создает новый проект.

Опции и параметры:

Имя файла — имя файла с путем или без пути. При отсутствии расширения принимается расширение pjx. Одновременно с созданием pjx создается файл с расширением pjt, содержащий значения memo-полей PJX-файла.

`NOSHOW` — указывает, что проект невидим `(Visible=.F.)`.

`NOPROJECTHOOK` — указывает, что при выполнении команды не создается объект `ProjectHook`. Такой объект может быть создан и позднее функцией `CREATEOBJECT()` или `NEWOBJECT()`.

Команда `MODIFY PROJECT [Имя файла | ?] [IN SCREEN] [NOWAIT] [SAVE] [NOSHOW] [NOPROJECTHOOK]` служит для корректировки уже созданного проекта.

Команда `BUILD PROJECT ProjectFileName [RECOMPILE] [FROM ProgramName1 | MenuName1 ReportName1 | LabelName1 | FormName1 | LibraryName1...]`

Опции и параметры:

`ProjectFileName1` — имя создаваемого проекта.

`RECOMPILE` — компилирует все файлы, включаемые в проект.

`FROM ProgramName1 | MenuName1 ReportName1 | LabelName1 | FormName1 | LibraryName1` — задает имена включаемых в проект файлов. По умолчанию первая программа или файл меню становится главной процедурой проекта.

Для обновления файла проекта команда `BUILD PROJECT` указывается без опции `FROM`. Все измененные файлы будут откомпилированы заново.

Об обнаруженных ошибках выдается сообщение, которое сохраняется в файле ProjectFileName.ERR. Однако ошибки не препятствуют построению файла проекта.

Создание простого исполняемого модуля

Давайте попробуем применить прочитанное выше на практике. Мы создадим простой исполняемый модуль, который будет содержать форму с расположенной на ней кнопкой. Из предыдущего материала понятно, что такой модуль должен содержать главную процедуру, форму с кнопкой и форму верхнего уровня, файл Config.fpw.

Порядок действий должен быть такой:

1. Создаем проект `Knopka`. Используем для этого либо главное меню (**File | New | Project**), либо вводим команду в командном окне **CREATE PROJECT**.
2. Создаем форму `Form0` как форму верхнего уровня (As Top-Level form). В проекте выбираем узел `Documents`, раскрываем его и переходим на `Forms`.

   Нажимаем кнопку **New**, в проекте появляется модель формы. Созданная форма имеет свои свойства (Property), которые мы должны настроить. Чтобы изменить свойства формы, нажимаем, находясь на форме, правую кнопку мыши и в появившемся меню выбираем пункт **Property** (Свойства). Устанавливаем свойство формы `ShowWindow = 2 (As Top-Level Form)`. Поскольку форма `Form0` является "декорацией", на фоне которой происходят все действия программы, есть смысл развернуть ее во весь экран.

   Поэтому свойство формы `WindowState` устанавливаем равным 2 (`Maximized`). Если хотите, можете изменить цвет формы, используя ее свойство `BackColor`.
3. Создаем форму `Form1` с кнопкой. Ваши действия подобны выполненным в п. 2, однако свойства этой формы будут другими. `ShowWindow = 1 (In Top-Level Form)`, т. е. форма выполняется на фоне формы верхнего уровня, свойство `WindowState = 0 (Normal)`, это значит, что размер формы не изменяется. Какой мы ее создадим, такой она и появится на экране. В этой форме мы расположим кнопку. Чтобы кнопка появилась на экране, достаточно выбрать ее на панели инструментов **Form Controls Toolbar** и перетащить мышью в форму. Кроме **Form Controls Toolbar** можно использовать **Toolbox**  . В свойствах кнопки укажем заголовок (Caption), назовем кнопку **Выход**. Чтобы выход можно было осуществить, запишем следующий код в событие `Click` кнопки:

   `ThisForm.release()` — закроется форма `Form1`;

   `Form0.release()` — закроется форма `Form0`.
4. Необходимо создать главную процедуру проекта. В проекте перейдем к узлу `Code` и далее к `Program`. Создаем новую программу, нажав на кнопку **New** в правой части Менеджера проектов. Ниже приводится листинг главной процедуры. Обычно ее называют Main.prg или start.prg. Не будем отступать от стандарта, так ее и назовем — Main.prg.

```
SET DELETED ON
```

```
SET DATE GERMAN
SET EXCLUSIVE OFF
IF SET('TALK') = 'ON'
  SET TALK OFF
  PUBLIC gcOldTalk
  gcOldTalk = 'ON'
ELSE
        PUBLIC gcOldTalk
  gcOldTalk = 'OFF'
ENDIF
gcOldDir = FULLPATH(CURDIR())
gcOldPath= SET('PATH')
lcOnShutdown="Shutdown()"
ON SHUTDOWN &lcOnShutdown
ON ERROR ErrorHandler(ERROR(),PROGRAM(),LINENO())
Do Form Form0
Do form form1
Read events
CLEAR DLLS
RELEASE ALL EXTENDED
CLEAR ALL
SET PATH TO
SET PROCEDURE TO
CLEAR           && the screen


FUNCTION ErrorHandler(nError,cMethod,nLine)
LOCAL lcErrorMsg,lcCodeLineMsg
WAIT CLEAR
lcErrorMsg=MESSAGE()+CHR(13)+CHR(13)
lcErrorMsg=lcErrorMsg+"Method:    "+cMethod
lcCodeLineMsg=MESSAGE(1)
IF BETWEEN(nLine,1,10000) AND NOT lcCodeLineMsg="..."
        lcErrorMsg=lcErrorMsg+CHR(13)+"Line:         "+ALLTRIM(STR(nLine))
        IF NOT EMPTY(lcCodeLineMsg)
                lcErrorMsg=lcErrorMsg+CHR(13)+CHR(13)+lcCodeLineMsg
        ENDIF
ENDIF
IF MESSAGEBOX(lcErrorMsg,17,_screen.Caption)#1
        ON ERROR
        RETURN .F.
ENDIF
ENDFUNC

FUNCTION ShutDown
Cleanup()
QUIT
```

```
ENDFUNC

FUNCTION Cleanup
 ON ERROR
 ON SHUTDOWN
 SET CLASSLIB TO
 SET PATH TO
 CLOSE ALL
 RETURN
```

5. При запуске нашего приложения главное окно Visual FoxPro должно быть погашено. Самый простой способ добиться желаемого — указать команду в файле конфигурации:

   ```
   Screen=OFF
   ```

6. Теперь мы должны построить (скомпилировать) наше приложение. Для компиляции нажимаем кнопку **Build** в правой части Менеджера проектов, указываем имя приложения, например, `Knopka`. Вот и все — наше приложение готово. Давайте теперь его запустим с целью тестирования. Построенное приложение находится на CD (CHAR3\primery\2).