**Структуры файлов в C++ Builder**

**Наталия Елманова, Центр Информационных Технологий**

**Структура файла проекта**

Для каждого приложения C++ Builder создается один файл проекта, один make-файл и один файл ресурсов. Файл проекта генерируется при выборе пункта меню File/New Application. Первоначально файлу проекта присваивается по умолчанию имя Project1.cpp. Если в процессе разработки приложения добавляются формы и модули, C++ Builder обновляет файл проекта.

Для просмотра файла проекта следует выбрать пункт меню View/Project Source. Эта операция выполнит загрузку исходного текста файла проекта в редактор кода (рис. 1).

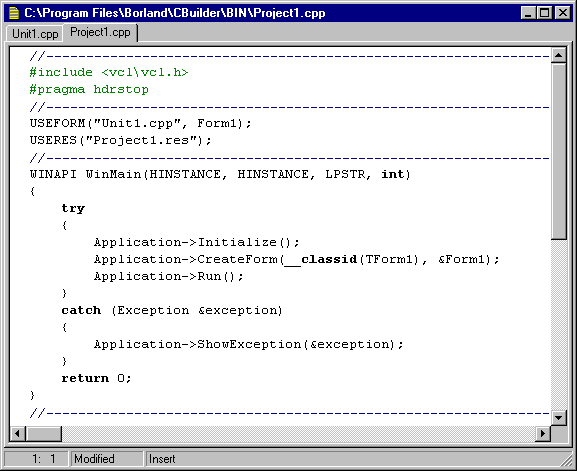


Рис. 1. Просмотр файла проекта в редакторе кода

Файл проекта имеет такую же структуру, как и файл модуля. Подобно файлу модуля, это файл исходного кода на языке C++, который компилируется с другими файлами при создании исполняемого файла.

В файле проекта имеется определенный набор ключевых элементов:

Директива препроцессора #include <vcl\vcl.h> предназначена для включения в текст проекта заголовочного файла, ссылающегося на описания классов библиотеки компонентов.

Директива препроцессора #pragma hrdstop предназначена для ограничения списка заголовочных файлов, доступных для предварительной компиляции.

Директива USEFORM сообщает, какие модули и формы используются в проекте.

директива USERES компилятора присоединяет файлы ресурсов к выполняемому файлу. При создании проекта автоматически создается файл ресурсов с расширением \*.res для хранения курсоров, пиктограммы приложения и др.

Application->Initialize() Это утверждение критично только в случае, если приложение является OLE automation-сервером. В остальных случаях оно фактически ничего не делает.

Application->CreateForm() Это утверждение создает форму приложения. По умолчанию, каждая форма в приложении имеет свое утверждение CreateForm.

Application->Run() Это утверждение запускает приложение (точнее, переводит его в состояние ожидания наступления одного из событий, на которое оно должно реагировать).

Конструкция try...catch используется для корректного завершения приложения в случае возникновения ошибки при инициализации, создании форм, запуске приложения.

**Структура make-файла**

Make-файл - это текстовый файл, содержащий макросы, директивы и инструкции по компиляции и сборке проекта для утилиты make.exe. Отметим, что make-файл по существу ничем не отличается от обычного знакомого пользователям С и С++ make-файла. Он генерируется автоматически при создании проекта, и его содержимое зависит от установок, указанных программистом в опциях проекта. Чтобы увидеть содержание make-файла, следует выбрать пункт меню View/Project Makefile (рис. 2).

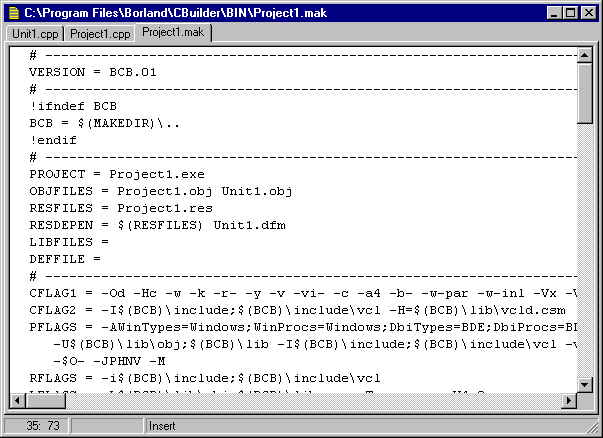


Рис. 2. Пример структуры make-файла

**Структура модуля**

Модули являются основой создания библиотек и приложений в C++ Builder. Модуль содержит исходный текст на языке C++ и первоначально представляет собой файл с расширением \*.CPP. В дальнейшем каждый такой файл компилируется в объектный файл с расширением \*.OBJ. Объектные файлы, в свою очередь, собираются компоновщиком в выполняемый файл с расширением \*.EXE.

При добавлении к проекту новой формы генерируется новый модуль. При добавлении модуля к проекту при помощи выбора пункта меню File/New Unit создается пустая структура модуля, в которой включены директивы: #include <vcl\vcl.h>; #pragma hdrstop; #include "Unit2.h"

Директива #include "Unit2.h" указывает на то, что в текст модуля должен быть включен соответствующий заголовочный файл.

При создании модуля используются следующие правила:

Имя должно быть уникальным. Два модуля с одним и тем же именем не могут использоваться одновременно в одном и том же проекте.

Если модуль связан с формой, то имя исходного файла модуля и файла формы (с расширением \*.dfm) должны быть одинаковыми. При создании обработчика событий в инспекторе объектов в тексте файла модуля генерируется шаблон функции, в который разработчик должен вводить код, выполняемый при наступлении обрабатываемого события.

**Структура h-файла**

h-файл генерируется при создании нового модуля. В нем содержится информация о данных и функциях, которые можно использовать в модуле. h-файл для модуля, связанного с формой, содержит описания интерфейсных элементов и других компонентов этой формы и обработчиков событий для них (то есть, в терминологии объектно-ориентированного программирования, описание класса формы). Такие описания автоматически добавляются в h-файл при внесении в форму новых компонентов или генерации новых обработчиков событий. Иными словами, в h-файле содержится интерфейс, а в самом модуле - реализация.

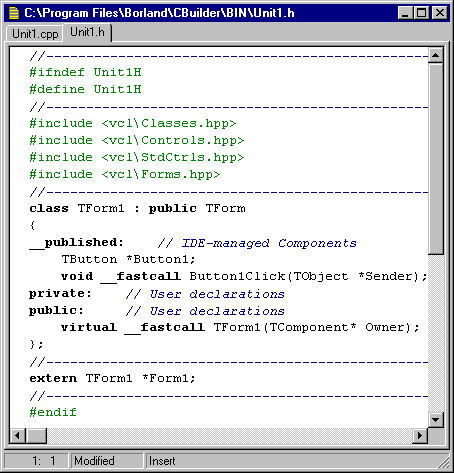


Рис. 3. Пример структуры h-файла

Примечание. При удалении из формы компонентов их описания удаляются из h-файла, а описания обработчиков событий сохраняются. При переименовании компонентов изменяются их описания в h-файле, а также имена и описания обработчиков событий, сгенерированные автоматически. Однако при этом не изменяются ссылки на эти компоненты и обработчики событий, используемые в других функциях, поэтому рекомендуется переименовывать компоненты и обработчики событий сразу же после их создания, пока на них не появились ссылки.

Не рекомендуется удалять из модуля пустые функции (например, случайно созданные шаблоны обработчиков событий). Они не увеличат размер исполняемого файла, но их удаление может привести к невозможности заново сгенерировать обработчик события, если в этом возникнет необходимость.

Отметим, что в модуле могут содержаться функции, не описанные в h-файле, однако видимость их в этом случае ограничивается данным модулем.

Внутри модуля функции могут быть определены и ссылаться друг на друга в произвольном порядке. Если данный модуль ссылается на другие формы и модули, следует с помощью директивы препроцессора #include включить в него соответствующий h-файл с помощью пункта меню File/Include Unit Hdr... . После этого интерфейсные элементы другого модуля будут доступны в данном модуле.

**Файл формы**

Форма является одним из важнейших элементов приложения C++ Builder. Процесс редактирования формы происходит при добавлении к форме компонентов, изменении их свойств, создании обработчиков событий.

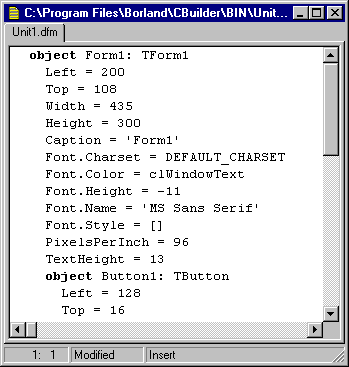


Рис. 4. Структура файла формы

Когда к проекту добавляется новая форма, создаются три отдельных файла:

Файл модуля (.cpp) - cодержит исходный код, связанный с формой.

h-файл(.h) - cодержит описание класса формы, то есть описания содержащихся на форме компонентов и обработчиков событий.

Файл формы (.dfm) - двоичный файл, содержащий сведения об опубликованных (то есть доступных в инспекторе объектов) свойствах компонентов, содержащихся в форме. Двоичный файл формы содержит информацию, используемую для конструирования формы из компонентов, расположенных на ней. При добавлении компонента к форме и заголовочный файл, и двоичный файл формы модифицируются. При редактировании свойств компонента в инспекторе объектов эти изменения сохраняются в двоичном файле формы.

Хотя в C++ Builder файл .dfm сохраняется в двоичном формате, его содержание можно просмотреть с помощью редактора кода. Для этого нужно нажать правую клавишу мыши над формой и из контекстного меню формы выбрать пункт View as Text.

Отметим, что при изъятии какого-либо компонента с формы в буфер обмена в последнем реально оказывается часть тестового представления файла формы, содержащая описание данного компонента. В этом можно убедиться, выполнив затем операцию вставки из буфера обмена в любом текстовом редакторе. Модули Delphi 2.0

C++ Builder создан на основе визуальной библиотеки компонентов Borland Delphi, ставшей за последние два года весьма популярной среди разработчиков. По этой причине этот продукт имеет общую с Delphi библиотеку классов, часть из которых осталась написанной на Object Pascal. Из этого следует, что в приложениях можно использовать компоненты, созданные для Delphi 2.0.

Однако совместимость с Delphi этим не исчерпывается. В проектах C++ Builder можно использовать не только библиотеку компонентов Delphi, но и код, написанный на Object Pascal, а также формы и модули Delphi. Эти возможности появились благодаря включению в С++ Builder обоих компиляторов -- С++ и Object Pascal.

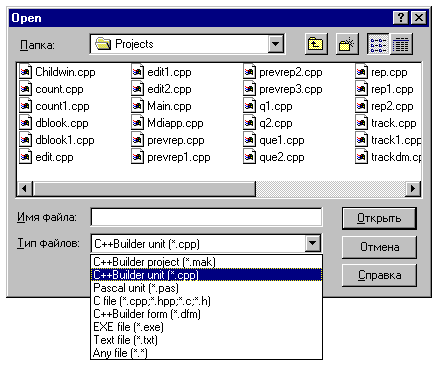


Рис. 5. Типы файлов, используемые в проектах С++ Builder

В соответствии с этим в качестве частей проекта могут быть использованы модули, написанные на Object Pascal. В этом можно убедиться, взяв формы какого-нибудь из примеров, созданных в Delphi 2.0 и включив их в проект С++ Builder. Возьмем в качестве такого примера приложение Graphex из набора примеров Delphi 2.0. Создадим новый проект, удалим из него созданную по умолчанию форму и добавим два модуля из приложения Graphex - Graphwin.pas и Bmpdlg.pas.

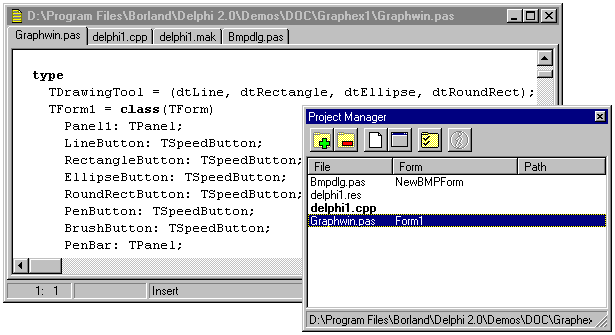


Рис. 6. Добавление модулей Delphi к проекту.

Скомпилируем проект. Можно убедиться в работоспособности полученного приложения, запустив и протестировав его.

В случае использования форм Delphi к проекту добавляются два файла - файл формы с расширением \*.dfm и файл модуля с расширением \*.pas. Описание класса формы содержится в самом модуле, следовательно, для этого не требуется отдельный файл заголовка.

Рассмотрим более подробно структуру модуля Delphi. В качестве простейшего примера приведем структуру модуля, связанного с формой, содержащей единственный интерфейсный элемент - кнопку закрытия окна: unit Unit1; interface uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs; type TForm1 = class(TForm) Button1: TButton; procedure Button1Click(Sender: TObject); private { Private declarations } public { Public declarations } end; var Form1: TForm1; implementation {$R \*.DFM} procedure TForm1.Button1Click(Sender: TObject); begin close; end; end.

Основными элементами модуля являются:

Заголовок модуля - предназначен для идентификации модуля и должен быть уникальным для данного проекта. Совпадает с именем файла с расширением \*.pas.

Секция интерфейса (начинается зарезервированным словом interface). Секция интерфейса является местом, где объявляются константы, типы, переменные, процедуры\* и функции, которые доступны для использования другими программными модулями. При объявлении процедур и функций в секции интерфейса необходимыми являются только их заголовки. Тела, или реализации, этих процедур и функций располагаются в секции реализации.

Секция реализации (начинается зарезервированным словом implementation) содержит процедуры и функции, включая обработчики событий. Объявления обработчиков событий должны присутствовать в секции интерфейса. Помимо этого, секция реализации может иметь дополнительные процедуры, функции и объявления констант, типов и переменных, упоминания о которых нет в секции интерфейса, но эти объявления видимы только в пределах данного модуля.

Секции инициализации и завершения (начинаются с зарезервированных слов initialization и finalization) - необязательные секции для размещения операторов, выполняемых соответственно при запуске и завершении приложения. Предложение uses (необязательное) содержит имена других модулей, используемых данным модулем и может располагаться непосредственно за ключевым словом interface и за ключевым словом implementation. Предложение uses делает секцию интерфейса других модулей доступной в данном модуле.

Отметим, однако, что для создания новых модулей Delphi требуется наличие Delphi. В С++ Builder их можно только использовать. Возможности редактирования форм и модулей Delphi в С++ Builder существенно ограничены - допустимы только операции, не затрагивающие секцию интерфейса модуля. Например, возможно перемещение интерфейсных элементов в пределах содержащих их компонентов-контейнеров, изменение размеров, копирование в буфер обмена, но запрещено их добавление или удаление. Возможны изменения внутри кода процедур и функций, но запрещено добавление новых обработчиков событий.

\*В терминологии Pascal процедурами обычно называют функции, не возвращающие значений (в частности, обработчики событий). Елманова Н.З., Кошель С.П. Введение в Borland С++ Builder