Глава 2

Начало

©Перевод сделан Тюрюмовым А.Н.

В этой главе рассматривается структура простой программы, написанной с использованием wxWidgets. Мы покажем запуск и процесс завершения wxWidgets-приложения, как показать главное окно и как обрабатывать команды от пользователя. На этом, следуя философии wxWidgets использовать только простые и красивые решения, мы главу и закончим. Для компиляции примеров вам возможно придется обратится к Приложению 1 «Установка wxWidgets».

2.1 Небольшой пример приложения на базе wxWidgets

Рис. 2.1 показывает, как наше приложение будет выглядеть в операционной системе Windows.

Минимальное приложение на wxWidgets показывает главное окно (класс `wxFrame`) со строками меню и статуса. Меню позволяет увидеть вам информацию «О программе» или выйти из программы. Это очень простое приложение, однако его вполне достаточно, чтобы проиллюстрировать некоторые базовые принципы построения приложений. Кроме того с накоплением опыта вы можете использовать этот пример, добавляя в него необходимую вам функциональность.

Рис. 2.1: Пример под ОС Windows

2.2 Класс приложения

Каждое приложение на wxWidgets определяет свой собственный класс приложения, являющийся потомком от `wxApp`. В программе существует единственный экземпляр данного класса, который представляет из себя представление данного приложения. Обычно этот класс объявляет функцию `OnInit`, которая вызывается, когда библиотека wxWidgets готова запустить ваш код (функция является эквивалентом функции `main` языка C или `WinMain` в Win32-приложениях).

Ниже дано минимально возможное объявление класса приложения:

```
// Объявляем класс приложения
class MyApp : public wxApp
{
public:
    // Вызывается при старте приложения
    virtual bool OnInit();
};
```

Реализация функции `OnInit` обычно создает по крайней мере одно окно, считывает параметры из командной строки, инициализирует нужные для работы структуры данных и выполняет другие действия, необходимые для запуска программы. Если функция возвращает `true`, то wxWidgets запускает петлю сообщений, которая обрабатывает ввод пользователя и запускает обработчик сообщений в случае необходимости. Если данная функция возвращает `false`, то wxWidgets корректно очищает свои внутренние структуры и завершает работу приложения.

Простейшая реализация функции `OnInit` может иметь следующий вид:

```
// Инициализируем приложение
bool MyApp::OnInit()
{
    // Создаем главное окно приложения
    MyFrame *frame = new MyFrame(wxT("Minimal wxWidgets App"));

    // Показываем его
    frame->Show(true);

    // Запускаем петлю сообщений
    return true;
}
```

Реализация создает экземпляр нового класса окна `MyFrame` (мы определим этот класс позднее), показывает окно на экране и возвращает true, чтобы запустить петлю сообщений. Главные окна (такие как фреймы и диалоги), в отличие от дочерних, должны быть показаны сразу после создания.

Заголовок главного окна передается конструктору, обернутым в макрос `wxT()`. В дальнейшем вы заметите, что этот макрос используется в большинстве примеров библиотеки wxWidgets и позволяет преобразовать строки и символы к правильному

типу, что позволяет приложению компилироваться в режиме Unicode. Данный макрос является синонимом макроса `_T()`. Также в программах вы возможно встретите макрос `_()`, который говорит библиотеке, что строку можно перевести на другой язык. Обратитесь к Главе 16 «Написание локализованных приложений» за дополнительной информацией.

Где же находится код, который создает экземпляр класса `MyApp`? wxWidgets делает это самостоятельно, но вы должны явно указать тип объекта, который необходимо создать. Поэтому вам необходимо добавить следующие строки в ваш файл с реализацией:

```
// Говорит wxWidgets, что надо создать объект MyApp
IMPLEMENT_APP(MyApp)
```

Без этого определения wxWidgets не сможет создать объект вашего приложения. Данный макрос также вставляет код, который проверяет, что объект является приложением и библиотека была скомпилирована с необходимыми опциями. Это позволяет wxWidgets сообщать о некоторых ошибках, которые впоследствии могли бы вызвать появление трудно выявляемых ошибок времени выполнения.

Когда wxWidgets создает объект типа MyApp, результат сохраняется в глобальной переменной `wxTheApp`. Можно использовать эту переменную в вашей программе, однако лучше явно не обращаться к ней в своем коде. Вместо этого, после объявления вашего класса приложения поместите макрос

```
// Реализация MyApp& GetApp()
DECLARE_APP(MyApp)
```

после чего вы сможете использовать в программе функцию `wxGetApp`, которая возвращает ссылку на объект приложения типа `MyApp`.

Совет: Даже если вы не используете макрос `DECLARE_APP`, вы все равно можете использовать переменную `wxTheApp` для доступа к функциям `wxApp`. Это позволяет избежать необходимости включения заголовочных файлов вашего приложения. Также это может быть полезно внутри кода, который не знает о специфичных классах приложения (например, при создании библиотек), а также для уменьшения времени компиляции.

2.3 Класс фрейма

Рассмотрим класс фрейма `MyFrame`. Фрейм является основным окном, которое содержит все остальные окна и обычно имеет меню и строку состояния. Вот простейший пример объявления класса для фрейма, которое мы разместим после объявления `MyApp`:

```
// Объявляем класс главного окна
class MyFrame : public wxFrame
{
public:
    //Конструктор
```

```
    MyFrame(const wxString& title);

    // Обработчики сообщений
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

private:
    // Этот класс перехватывает сообщения
    DECLARE_EVENT_TABLE()
};
```

Наш класс для фрейма содержит конструктор, два обработчика сообщений (связывающих пункты меню с C++ кодом) и макрос, который сообщает, что класс содержит обработчики сообщений.

2.4 Обработчики сообщений

Как вы уже знаете, функции обработки сообщений в `MyFrame` не являются виртуальными и не должны быть виртуальными. Тогда как же они вызываются? Ответ расположен в следующей таблице сообщений:

```
// Таблица сообщений для MyFrame
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
    EVT_MENU(wxID_EXIT,  MyFrame::OnQuit)
END_EVENT_TABLE()
```

Таблица сообщений, помещенная в файл с реализацией класса, говорит wxWidgets каким образом связываются сообщения, поступающие от пользователя, и методы класса.

В указанной выше таблице нажатие кнопки мыши на пунктах меню с идентификаторами `wxID_EXIT` и `wxID_ABOUT` направляется функциям `MyFrame::OnQuit` и `MyFrame::OnAbout` соответственно. Макрос `EVT_MENU` является одним из многих возможных макросов таблицы сообщений, которые вы можете использовать, чтобы указать wxWidgets какие типы сообщений направлять в функцию. Идентификаторы, используемые в нашем примере, являются предопределенными, но чаще всего вы должны вводить свои собственные с помощью перечислений (enum), констант или директив препроцессора (`#define`).

Этот тип таблицы сообщений называется статическим, то есть не может быть изменен в процессе выполнения программы. В следующей главе мы расскажем, как создавать динамические обработчики сообщений.

Пока мы разбираемся с таблицами сообщений, давайте посмотрим на две функции, которые у нас используются для обработки сообщений.

```
void MyFrame::OnAbout(wxCommandEvent& event)
{
    wxString msg;
```

```
    msg.Printf(wxT("Hello and welcome to %s"),
               wxVERSION_STRING);

    wxMessageBox(msg, wxT("About Minimal"),
                 wxOK | wxICON_INFORMATION, this);
}

void MyFrame::OnQuit(wxCommandEvent& event)
{
    // Уничтожаем фрейм
    Close();
}
```

Метод `MyFrame::OnAbout` показывает небольшое сообщение пользователю, когда тот выбирает из меню пункт About[1]. Функция `wxMessageBox` принимает в качестве аргументов текст сообщения, заголовок окна, комбинацию стилей и указатель на родительское окно.

Как это указано в таблице сообщений, функция `MyFrame::OnQuit` вызывается, когда пользователь выбирает в меню пункт Quit[2]. Наш обработчик данного сообщения вызывает метод `Close` для уничтожения фрейма и таким образом завершает работу приложения, так как программа состоит всего из одного фрейма. На самом деле метод `Close` не уничтожает приложение, он просто генерирует сообщение `wxEVT_CLOSE_WINDOW`, обработчик которого по умолчанию уничтожает фрейм, используя для этого метод `wxWindow::Destroy`.

Существуют другие пути для завершения работы приложения — пользователь может щелкнуть на кнопке закрытия, расположенной на фрейме, или выбрать пункт Close из системного меню (или меню оконного менеджера среды запуска). Когда метод `OnQuit` вызывается в этих случаях? Ну если честно, то он не вызывается. В указанных случаях wxWidgets посылает фрейму сообщение `wxEVT_CLOSE_WINDOW` через вызов функции `Close` (подобно тому, как это делаем мы в методе `OnQuit`). wxWidgets по умолчанию перехватывает это сообщение и уничтожает окно. Ваше приложение может переопределить данное поведение и, например, запрашивать подтверждение у пользователя перед закрытием. Обратитесь к Главе 4 «Окна» за дополнительной информацией.

В нашем примере этого делать не требовалось, но большая часть приложений реализует функцию `OnExit` в классе приложения для корректной очистки структур данных перед выходом. Обратите внимание, что эта функция вызывается только если `OnInit` возвратила `true`.

2.5 Создание фрейма

Наконец, у нас есть конструктор для фрейма, который инициализирует иконку для приложения, меню и строку состояния.

[1] О программе (англ.)
[2] Выход (англ.)

```
#include "mondrian.xpm"

MyFrame::MyFrame(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title)
{
    // Устанавливаем иконку для фрейма
    SetIcon(wxIcon(mondrian_xpm));

    // Создаем меню
    wxMenu *fileMenu = new wxMenu;

    // Добавляем пункт "About" (о приложении), который
    // должен показывать маленькую помощь
    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(wxID_ABOUT, wxT("&About...\tF1"),
                     wxT("Show about dialog"));

    fileMenu->Append(wxID_EXIT, wxT("E&xit\tAlt-X"),
                     wxT("Quit this program"));

    // Теперь добавляем созданное меню в строку меню...
    wxMenuBar *menuBar = new wxMenuBar();
    menuBar->Append(fileMenu, wxT("&File"));
    menuBar->Append(helpMenu, wxT("&Help"));

    // ... и присоединяем к фрейму
    SetMenuBar(menuBar);

    // Создаем строку состояния для красоты
    CreateStatusBar(2);
    SetStatusText(wxT("Welcome to wxWidgets!"));
}
```

Данный конструктор вызывает конструктор родительского класса с указанием родительского окна (NULL означает его отсутствие), идентификатором этого окна и заголовком. Вместо идентификатора в нашем случае передается значение `wxID_ANY`, которое говорит wxWidgets, что идентификатор должен быть создан библиотекой самостоятельно. Таким образом, конструктор создает настоящее окно с помощью перенаправления запроса на создание к родительскому классу.

Маленькие изображения и иконки могут быть помещены в программу с помощью мультиплатформенного формата XPM. Файл XPM представляет из себя обычный C++ код, а потому может быть включен так, как показано в нашем примере. Строка с `SetIcon` создает иконку в стеке, используя C++ переменную `mondrian_xpm` (определенную в файле mondrian.xpm) и связывает полученную иконку с фреймом.

Далее создается меню. При добавлении элементов меню указывается их идентификатор (например, стандартный идентификатор `wxID_ABOUT`, как в нашем случае), текстовую метку для элемента меню и строка помощи, которая может быть показана

в строке состояния. Внутри каждой метки можно поместить мнемоники (указатели для вызова пункта с клавиатуры) с помощью предшествующего букве амперсанда (&), а также указать акселератор, отделяя его от сообщения с помощью символа табуляции (\t). Мнемоники позволяют пользователю выбрать пункт меню с помощью нажатия соответствующей клавиши во время навигации по меню. Акселератор — это комбинация клавиш (например, Alt+X), которая может быть использована для выполнения действия вообще без вызова меню.

В конце конструктор создает строку статуса, состоящую из двух полей и расположенную снизу фрейма. Для первого поля устанавливается значение «Welcome to wxWidgets!».[3]

2.6 Текст программы

Объединим все кусочки вместе, чтобы увидеть как будет выглядеть программа целиком. Обычно я отделяю заголовочные файлы и файлы реализации, но для данного крошечного примера я объединил их в одином файле.

```
// Имя:         minimal.cpp
// Цель:        Пример минимального приложения на wxWidgets
// Автор:       Julian Smart

#include "wx/wx.h"

// Объявляем класс приложения
class MyApp : public wxApp
{
public:
    // Вызывается при старте приложения
    virtual bool OnInit();
};

// Объявляем класс главного окна
class MyFrame : public wxFrame
{
public:
    //Конструктор
    MyFrame(const wxString& title);

    // Обработчики сообщений
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

private:
    // Этот класс перехватывает сообщения
    DECLARE_EVENT_TABLE()
```

[3] Добро пожаловать в wxWidgets (англ.)

```
};

// Реализация MyApp& GetApp()
DECLARE_APP(MyApp)

// Говорит wxWidgets, что надо создать объект MyApp
IMPLEMENT_APP(MyApp)

// Инициализируем приложение
bool MyApp::OnInit()
{
    // Создаем главное окно приложения
    MyFrame *frame = new MyFrame(wxT("Minimal wxWidgets App"));

    // Показываем его
    frame->Show(true);

    // Запускаем петлю сообщений
    return true;
}

// Таблица сообщений для MyFrame
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
    EVT_MENU(wxID_EXIT,  MyFrame::OnQuit)
END_EVENT_TABLE()

void MyFrame::OnAbout(wxCommandEvent& event)
{
    wxString msg;
    msg.Printf(wxT("Hello and welcome to %s"),
               wxVERSION_STRING);


    wxMessageBox(msg, wxT("About Minimal"),
                 wxOK | wxICON_INFORMATION, this);
}

void MyFrame::OnQuit(wxCommandEvent& event)
{
    // Уничтожаем фрейм
    Close();
}

#include "mondrian.xpm"

MyFrame::MyFrame(const wxString& title)
```

```
    : wxFrame(NULL, wxID_ANY, title)
{
    // Устанавливаем иконку для фрейма
    SetIcon(wxIcon(mondrian_xpm));

    // Создаем меню
    wxMenu *fileMenu = new wxMenu;

    // Добавляем пункт "About" (о приложении), который
    // должен показывать маленькую помощь
    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(wxID_ABOUT, wxT("&About...\tF1"),
                     wxT("Show about dialog"));

    fileMenu->Append(wxID_EXIT, wxT("E&xit\tAlt-X"),
                     wxT("Quit this program"));

    // Теперь добавляем созданное меню в строку меню...
    wxMenuBar *menuBar = new wxMenuBar();
    menuBar->Append(fileMenu, wxT("&File"));
    menuBar->Append(helpMenu, wxT("&Help"));

    // ... и присоединяем к фрейму
    SetMenuBar(menuBar);

    // Создаем строку состояния для красоты
    CreateStatusBar(2);
    SetStatusText(wxT("Welcome to wxWidgets!"));
}
```

2.7 Компиляция и запуск программы

Данный пример можно найти на прилагаемом CD-ROM в папке examples/chap02. Для компиляции его необходимо скопировать на ваш жесткий диск. В связи с тем, что невозможно создать makefile, который будет работать «из коробки» с любой средой программирования, которая возможно установлена у читателя, мы сделали проект для среды DialogBlocks с конфигурацией, подходящей для большинства платформ и компиляторов. Обратитесь к Приложению 3 «Создание приложений с помощью DialogBlocks», чтобы настроить программу DialogBlocks для ее использования на вашем компьютере. Детали компиляции wxWidgets-приложений рассматриваются в Приложении 2 «Сборка ваших собственных приложений на wxWidgets».

Установите wxWidgets и DialogBlocks с предлагаемого CD. В системе Windows вам также необходимо установить подходящий компилятор (один из них присутствует на CD), если он у вас еще не установлен. После этого пропишите пути до библиотеки wxWidgets и вашего компилятора в настройках DialogBlocks на вкладке Paths и откройте файл examples/chap02/minimal.pjd. Выберете подходящую конфигура-

цию для вашего компилятора и платформы, например «MinGW Debug» или «VC++ Debug» (Windows), «GCC Debug GTK+» (Linux) или «GCC Debug Mac» (Mac OS X). Далее нажмите кнопку «Build and Run Project». Возможно среда спросит не хотите ли вы собрать библиотеку wxWidgets, если вы еще этого не сделали.

Вы можете найти похожий пример в папке samples/minimal в дистрибутиве wxWidgets. Если вы не хотите использовать DialogBlocks, то вы можете просто скомпилировать указанный пример вместо нашего. Обратитесь к Приложению 1 «Установка wxWidgets», чтобы узнать как собрать стандартные примеры wxWidgets.

2.8 Поток выполнения

Действия, происходящие при старте программы:

1. В зависимости от платформы запускается функция `main`, `WinMain` или эквивалентная ей (эта функция предоставляется библиотекой wxWidgets, а не вашим приложением). wxWidgets инициализирует внутренние структуры данных и создает экземпляр класса `MyApp`.

2. wxWidgets вызывает метод `MyApp::OnInit`, который создает экземпляр класса `MyFrame`.

3. Конструктор класса `MyFrame` создает окно через конструктор `wxFrame` и добавляет иконку, меню и строку состояния.

4. `MyApp::OnInit` показывает фрейм и возвращает `true`.

5. wxWidgets запускает петлю сообщений, ждет сообщения и вызывает соответствующий обработчик сообщений.

Как уже было указано, приложение завершается, когда закрывается основной фрейм, что можно сделать через соответствующий пункт меню, через стандартные кнопки или стандартные меню (это зависит от платформы).

2.9 Итоги

Глава дает представление о том, как работает простейшее приложение на wxWidgets. Мы коснулись темы обработки сообщений, использования класса `wxFrame`, инициализации приложения, создания меню и строки состояния. Однако, несмотря на сложность вашего реального приложения, базовые принципы его функционирования останутся теми же самыми, как и у нашего маленького примера. В следующей главе мы подробнее рассмотрим сообщения и методы их обработки.