Глава 8

Использование стандартных диалогов

©Перевод сделан Тюрюмовым А.Н.

Глава описывает набор стандартных диалогов, которые реализованы в wxWidgets для отображения информации или получения данных от пользователя. Использование большинства стандартных диалогов потребует всего несколько строк кода. Знакомство с доступными стандартными диалогами позволит вам сэкономить много времени, а также сделает вид вашего приложения более профессиональным. Там, где это возможно wxWidgets использует существующую в ОС (нативную) реализацию, но для некоторых (таких как `wxTextEntryDialog`) в библиотеке существует своя собственная реализация. Последний тип диалогов мы будем называть «универсальными» (generic). В данной главе мы также покажем, как выглядят диалоги в различных системах, особенно если они сильно визуально различаются.

Чтобы ввести некоторую структуру мы разделим все диалоги на категории: информационные, файловые или каталожные, а также диалоги выбора и диалоги ввода.

8.1 Информационные диалоги

В данном разделе описаны диалоги для отображения информации: `wxMessageDialog`, `wxProgressDialog`, `wxBusyInfo` и `wxShowTip`.

8.1.1 wxMessageDialog

Рис. 8.1: wxMessageDialog в системе Windows

Рис. 8.2: wxMessageDialog в системе GTK+

Рис. 8.3: wxMessageDialog в системе Mac OS X

Этот диалог показывает сообщение и несколько кнопок, которые могут быть «OK», «Отмена», «Да» и «Нет». Дополнительно можно показать иконку, такую как знак восклицания или вопроса. Текст сообщения может содержать символы перевода строки («\n»).

Возвращаемое из `wxMessageDialog::ShowModal` значение содержит информацию о том, какую из кнопок нажал пользователь.

Рисунок 8.1 показывает вид диалога в системе Windows, рисунок 8.2 показывает его же под GTK+, а рисунок 8.3 — в Mac OS X.

Для создания диалога передайте ему родительское окно и сообщение. Можно также указать необязательные параметры — заголовок, стиль и положение. Далее необходимо вызвать `ShowModal`, чтобы показать окно на экране, а потом обработать возвращаемое значение.

Стиль является битовым полем со значениями, приведенными в таблице 8.1.

Таблица 8.1: Стили wxMessageDialog

Тип	Описание
`wxOK`	Показать кнопку «OK».
`wxCANCEL`	Показать кнопку «Отмена».
`wxYES_NO`	Показать кнопки «Да» и «Нет».
`wxYES_DEFAULT`	Установить кнопку «Да» в качестве выбора по умолчанию. Используется с флагом `wxYES_NO`. Данный флаг установлен по умолчанию для `wxYES_NO`.
`wxNO_DEFAULT`	Установить кнопку «Нет» как выбор по умолчанию. Используется с флагом `wxYES_NO`.
`wxICON_EXCLAMATION`	Показать восклицательный знак.
`wxICON_ERROR`	Показать иконку для ошибки.

Таблица 8.1: Стили wxMessageDialog (продолжение)

Тип	Описание
wxICON_HAND	Показать иконку для ошибки (аналогичен wxICON_ERROR).
wxICON_QUESTION	Показать знак вопроса.
wxICON_INFORMATION	Показать знак информирования.
wxSTAY_ON_TOP	В системе Windows диалог будет показываться поверх всех окон, даже если они не принадлежат данному приложению.

Пример использования wxMessageDialog

Пример кода, использующего wxMessageDialog:

```
#include "wx/msgdlg.h"

wxMessageDialog dialog( NULL, wxT("Message box caption"),
      wxT("Message box text"),
      wxNO_DEFAULT|wxYES_NO|wxCANCEL|wxICON_INFORMATION);

switch ( dialog.ShowModal() )
{
  case wxID_YES:
      wxLogStatus(wxT("You pressed \"Yes\""));
      break;

  case wxID_NO:
      wxLogStatus(wxT("You pressed \"No\""));
      break;

  case wxID_CANCEL:
      wxLogStatus(wxT("You pressed \"Cancel\""));
      break;

  default:
      wxLogError(wxT("Unexpected wxMessageDialog return code!"));
}
```

8.1.2 wxMessageBox

Вместо создания диалога можно воспользоваться удобной функцией wxMessageBox, которая принимает на вход строку с сообщением и заголовком, стиль и родительское окно. Например:

```
if (wxYES == wxMessageBox(wxT("Message box text"),
    wxT("Message box caption"),
    wxNO_DEFAULT|wxYES_NO|wxCANCEL|wxICON_INFORMATION,
```

```
    parent))
{
    return true;
}
```

Обратите внимание, что `wxMessageBox` возвращает значения, которые отличаются от возвращаемых значений метода `wxMessageDialog::ShowModal`. `wxMessageBox` возвращает `wxOK`, `wxCANCEL`, `wxYES` или `wxNO` в случаях, когда `wxMessageDialog::ShowModal` возвращает `wxID_OK`, `wxID_CANCEL`, `wxID_YES` и `wxID_NO` соответственно.

8.1.3 wxProgressDialog

`wxProgressDialog` показывает короткое сообщение и индикатор выполнения, отражающий как долго пользователю необходимо будет ждать завершения текущей операции. Кроме того диалог может содержать кнопку «Отмена» для отмены выполняемой операции, а также показывать прошедшее время, предполагаемое время и оставшееся время выполнения операции. Данный диалог реализован средствами wxWidgets для всех платформ. Рисунок 8.4 показывает вид `wxProgressDialog` в системе Windows.

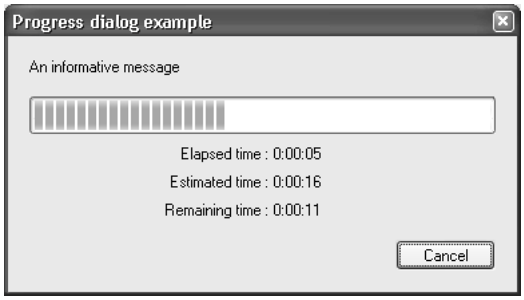

Рис. 8.4: wxProgressDialog в системе Windows

Объект диалога можно создать на стеке или динамически. При создании диалогу передаются следующие параметры: строка с заголовком, строка с сообщением (которое отображается на индикатором выполнения), максимальное значение для индикатора, родительское окно и стиль.

Стиль является битовым полем со значениями, приведенными в таблице 8.2.

Таблица 8.2: Стили wxProgressDialog

Тип	Описание
`wxPD_APP_MODAL`	Сделать диалог прогресса модальным. Если не указать данный стиль, то диалог будет «локально» модальным, т.е. будет заблокирован ввод в родительское окно.
`wxPD_AUTO_HIDE`	Диалог исчезнет с экрана как только достигнется максимальное значение для индикатора прогресса.

Таблица 8.2: Стили wxProgressDialog (продолжение)

Тип	Описание
wxPD_CAN_ABORT	Диалог будет иметь кнопку «Отмена», которую пользователь может нажать. Если это произойдет, то следующий вызов метода Update возвратит false.
wxPD_ELAPSED_TIME	Диалог будет показывать время, прошедшее с его создания.
wxPD_ESTIMATED_TIME	Диалог будет показывать предполагаемое время выполнения операции.
wxPD_REMAINING_TIME	Диалог будет показывать время, оставшееся до окончания операции.

После вызова диалога поток выполнения идет дальше, но родительское окно становится недоступным для ввода. Если указать флаг wxPD_APP_MODAL, то недоступными станут также все остальные окна приложения. Приложение должно вызывать метод Update с некоторым значением (от нуля до максимального, определяемого в конструкторе) и, возможно, новым сообщением, которое нужно показать в диалоге. Если определены соответствующие флаги, то прошедшее, предполагаемое и оставшееся времена будут посчитаны диалогом автоматически.

Если задан флаг wxPD_AUTO_HIDE, то при передаче максимального значения методу Update диалог прогресса будет скрыт (но не уничтожен). Приложение должно уничтожить диалог самостоятельно. Также вы можете вызвать Resume, чтобы показать диалог снова.

Пример использования wxProgressDialog

Пример использования диалога прогресса:

```
#include "wx/progdlg.h"

void MyFrame::ShowProgress()
{
    static const int max = 10;

    wxProgressDialog dialog(wxT("Progress dialog example"),
                            wxT("An informative message"),
                            max,    // range
                            this,   // parent
                            wxPD_CAN_ABORT |
                            wxPD_APP_MODAL |
                            wxPD_ELAPSED_TIME |
                            wxPD_ESTIMATED_TIME |
                            wxPD_REMAINING_TIME);

    bool cont = true;
    for ( int i = 0; i <= max; i++ )
    {
```

```
        wxSleep(1);
        if ( i == max )
            cont = dialog.Update(i, wxT("That's all, folks!"));
        else if ( i == max / 2 )
            cont = dialog.Update(i, wxT("Only a half left (very long message)!"));
        else
            cont = dialog.Update(i);

        if ( !cont )
        {
            if ( wxMessageBox(wxT("Do you really want to cancel?"),
                              wxT("Progress dialog question"),
                              wxYES_NO | wxICON_QUESTION) == wxYES )
                break;

            dialog.Resume();
        }
    }

    if ( !cont )
        wxLogStatus(wxT("Progress dialog aborted!"));
    else
        wxLogStatus(wxT("Countdown from %d finished"), max);
}
```

8.1.4 wxBusyInfo

На самом деле `wxBusyInfo` не является диалогом (он унаследован от `wxObject`), но функционирует похожим образом. Пока объект жив он показывает окно, содержащее указанное в конструкторе сообщение. Данный механизм обычно используется, если пользователю необходимо подождать, пока программа завершит некоторую работу. Вид окна в системе Windows показан на рисунке 8.5.

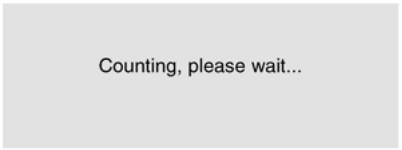

Рис. 8.5: wxBusyInfo в системе Windows

Создать объект `wxBusyInfo` можно на стеке или динамически, передав ему сообщение и родительское окно.

Пример использования wxBusyInfo

В следующем примере используется `wxBusyInfo`, но сначала создается `wxWindowDisabler` для того, чтобы заблокировать все открытые окна приложения.

```
#include "wx/busyinfo.h"

wxWindowDisabler disableAll;

wxBusyInfo info(wxT("Counting, please wait..."), parent);
for (int i = 0; i < 1000; i++)
{
    DoCalculation();
}
```

8.1.5 wxShowTip

Многие приложения при старте показывают окно с небольшими советами, которые помогают пользователю научиться более эффективно использовать приложение. Советы помогают изучать приложение маленькими, легко воспринимаемыми кусками, и особенно полезны пользователям, которые не любят читать документацию.

Рисунок 8.6 показывает вид диалога с советами в системе Windows.

Рис. 8.6: Диалог с советами в системе Windows

В отличие от большинства стандартных диалогов для вызова диалога советов необходимо позвать функцию: `wxShowTip`. Передайте ей родительское окно, указатель на объект `wxTipProvider` и необязательное булево значение, которое определяет показывать ли флажок «Show Tips at Startup»[1]. Диалог возвращает значение данного флажка.

Чтобы написать собственный источник советов вам необходимо унаследоваться от `wxTipProvider` и переопределить метод `GetTip`, который должен возвращать строку `wxString`, содержащую совет. Кроме того, wxWidgets уже содержит одну такую реализацию `wxCreateFileTipProvider`, которая принимает на вход имя файла с советами (по одному на строку) и номер строки в этом файле.

Приложение должно удалить объект класса `wxTipProvider` самостоятельно, когда он станет не нужным.

[1] Показывать диалог с советами при запуске (англ.)

Пример использования wxShowTip

Далее показан пример вызова диалога с советами, использующего стандартный источник для советов:

```
#include "wx/tipdlg.h"

void MyFrame::ShowTip()
{
    static size_t s_index = (size_t)-1;

    if ( s_index == (size_t)-1 )
    {
        // инициализируем генератор случайных чисел...
        srand(time(NULL));

        // ... и получаем номер нового совета
        s_index = rand() % 5;
    }
    // передаем файл с советами и выбранный номер
    wxTipProvider *tipProvider =
        wxCreateFileTipProvider(wxT("tips.txt"), s_index);

    m_showAtStartup = wxShowTip(this, tipProvider, true);
    delete tipProvider;
}
```

8.2 Диалоги для файлов и каталогов

Существует два диалога, которые можно использовать для получения файла и каталога от пользователя: `wxFileDialog` и `wxDirDialog`.

8.2.1 wxFileDialog

С помощью `wxFileDialog` можно позволить пользователю выбрать один или несколько файлов. Существуют также варианты для открытия и сохранения файла.

На рисунке 8.7 показан вид диалога в системе Windows.

Рисунки 8.8 и 8.9 показывают вид файлового диалога в системе Linux с использованием GTK+ версии 1 и 2 соответственно.

Внешний вид того же диалога в системе Mac OS X показан на рисунке 8.10.

При создании `wxFileDialog` ему необходимо передать родительское окно, сообщение для пользователя, каталог по умолчанию, имя файла по умолчанию, маску, стиль, расположение и размер (последние два параметра могут игнорироваться реализацией). Далее необходимо вызвать `ShowModal` и сравнить возвращаемое значение на равенство с `wxID_OK`, что означает, что пользователь подтвердил свой выбор.

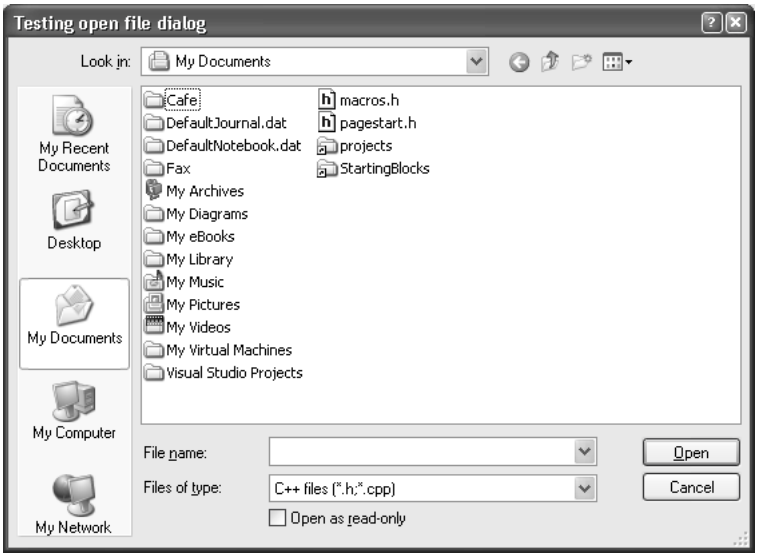

Рис. 8.7: wxFileDialog в системе Windows

Каталог и имя файла рассматриваются как отдельные элементы полного пути до файла. Если каталог пустой, то используется текущий каталог. Если имя файла пустое, то пользователю не предлагается имя файла по умолчанию.

Маска определяет какие файлы показываются в диалоге. Маска может содержать определения для нескольких типов файлов с описанием каждого из них, например,

```
"BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif"
```

Если пользователь напишет имя файла, содержащее групповые символы («*», «?») в поле для имени и нажмет «OK», то покажутся только файлы удовлетворяющие данному шаблону.

Стили wxFileDialog

В таблице 8.3 приведены стили для файлового диалога.

Таблица 8.3: Стили wxFileDialog

Тип	Описание
`wxSAVE`	Указывает, что диалог используется для сохранения файла.
`wxOPEN`	Указывает, что диалог используется для открытия файла (используется по умолчанию).
`wxOVERWRITE_PROMPT`	В диалоге сохранения пользователю будет выдан дополнительный запрос в случае, если выбраный файл уже существует.
`wxFILE_MUST_EXIT`	Пользователь должен выбрать существующий файл.
`wxMULTIPLE`	Пользователь может выбрать несколько файлов.

Методы wxFileDialog

`GetDirectory` возвращает каталог по умолчанию или каталожную часть выбранного файла в диалоге, где можно выбрать только один файл. Используйте `SetDirectory`, чтобы определить каталог по умолчанию.

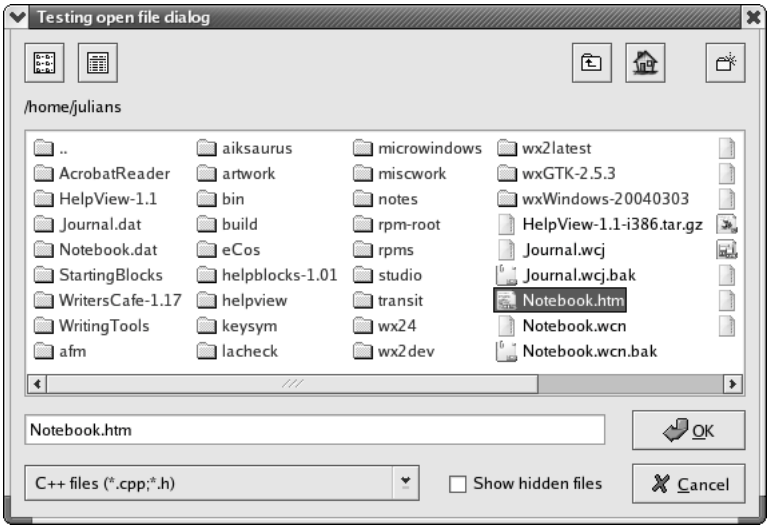

Рис. 8.8: Родной wxFileDialog в оконной системе GTK+

`GetMessage` возвращает заголовок диалога. Используйте `SetMessage`, чтобы установить новый заголовок.

`GetPath` возвращает полный путь (каталог и имя файла) до выбранных пользователем файлов или путь по умолчанию. Используйте `SetPath` для установки пути по умолчанию. Для диалога с множественным выбором `GetPaths` позволяет получить `wxArrayString` выбранных файлов, включая путь до них.

`GetWildcard` возвращает определение маски, а `SetWildcard` устанавливает маску.

Пример использования wxFileDialog

Вот пример использования `wxFileDialog` для открытия одного BMP- или GIF-файла:

```
#include "wx/filedlg.h"

wxString caption = wxT("Choose a file");
wxString wildcard =
   wxT("BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif");
wxString defaultDir = wxT("c:\\temp"));
wxString defaultFilename = wxEmptyString;

wxFileDialog dialog(parent, caption, defaultDir, defaultFilename,
    wildcard, wxOPEN);
if (dialog.ShowModal() == wxID_OK)
{
    wxString path = dialog.GetPath();
    int filterIndex = dialog.GetFilterIndex();
}
```

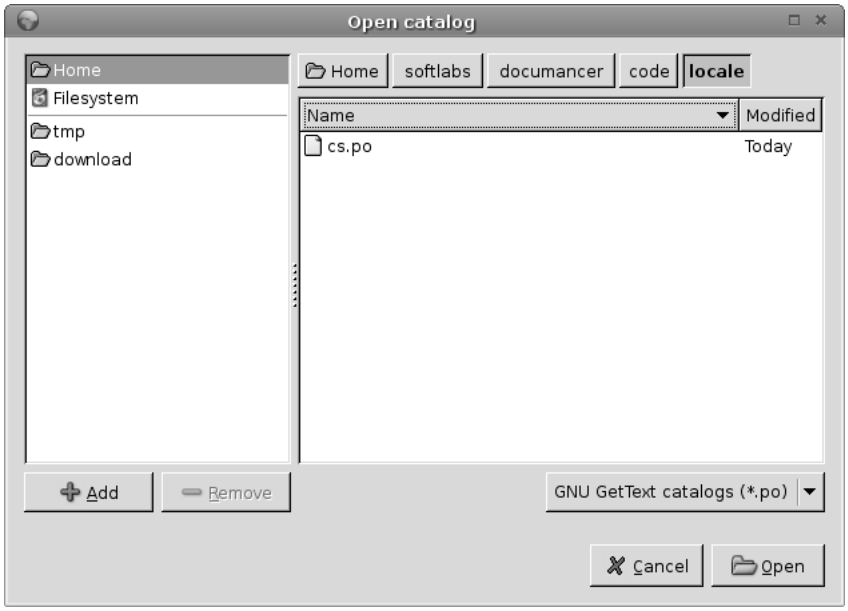

Рис. 8.9: Родной wxFileDialog в оконной системе GTK+ 2.4 и выше

8.2.2 wxDirDialog

Диалог `wxDirDialog` позволяет пользователю выбрать локальный или сетевой каталог (папку). Дополнительно, с помощью передачи стиля `wxDD_NEW_DIR_BUTTON` в конструктор можно позволить пользователю создать новые каталоги.

Рисунок 8.11 показывает вид `wxDirDialog` в системе Windows, где используется стандартный диалог системы. В GTK+ в системе Linux используется универсальная версия `wxDirDialog`, которая показана на рисунке 8.12.

В Mac OS X диалог `wxDirDialog` (рисунок 8.13) очень похож на файловый диалог.

При создании диалога ему необходимо передать родительское окно, сообщение для пользователя, каталог по умолчанию, стиль, расположение и размер (последние два параметра могут игнорироваться реализацией). Далее необходмио вызвать `ShowModal` и сравнить возвращаемое значение на равенство с `wxID_OK`, что означает, что пользователь подтвердил свой выбор.

Методы wxDirDialog

`SetPath` и `GetPath` позволяют устанавливать и получать выбранный каталог.

`SetMessage` устанавливает сообщение, отображаемое в диалоге. `GetMessage` позволяет получить это сообщение.

Пример использования wxDirDialog

Использовать `wxDirDialog` легко, что и показывает следующий пример:

```
#include "wx/dirdlg.h"

wxString defaultPath = wxT("/");
```

Рис. 8.10: wxFileDialog в системе Mac OS X

```
wxDirDialog dialog(parent,
    wxT("Testing directory picker"),
    defaultPath, wxDD_NEW_DIR_BUTTON);

if (dialog.ShowModal() == wxID_OK)
{
    wxString path = dialog.GetPath();
    wxMessageBox(path);
}
```

8.3 Диалоги выбора

В этом разделе мы изучим диалоги, которые позволяют пользователя сделать некоторый выбор: `wxColourDialog`, `wxFontDialog`, `wxSingleChoiceDialog` и `wxMultiChoiceDialog`.

8.3.1 wxColourDialog

Диалог позволяет пользователю выбрать цвет из предложенной палитры или из всего многообразия цветов.

В Windows используется обычный системный диалог выбора цвета. Данный диалог содержит три основных региона: в левом верхнем углу демонстрируется палитра из 48 часто используемых цветов. Под ним располагается палитра из 16 настраиваемых цветов, которая может изменяться приложением. Кроме того, пользователь может добавить свой собственный цвет в последнюю палитру

Рис. 8.11: wxDirDialog в системе Windows

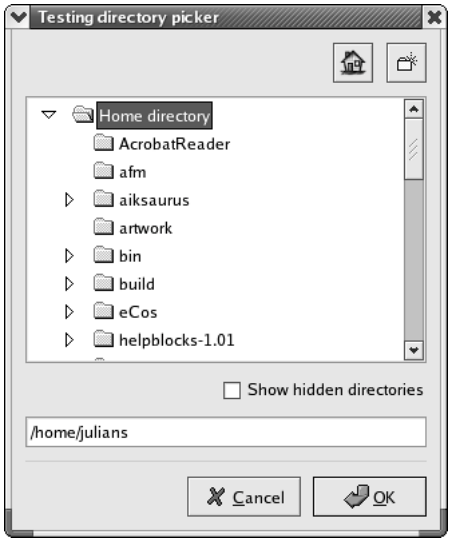

Рис. 8.12: wxDirDialog в системе GTK+

с помощью распахиваемой части диалога, в которой можно выбрать точные характеристики цвета на специальной панели. На рисунке 8.14 показан диалог выбора цвета в распахнутом режиме.

Универсальный диалог выбора цвета (показан на рисунке 8.15 для GTK+ 1 и X11) отображает палитру из 48 стандартных и 16 настраиваемых цветов со специальной зоной справа, содержащей три слайдера, которые определяют красную, зеленую и синюю компонены цвета. Созданный цвет может быть добавлен в палитру настраиваемых цветов. При этом новый цвет заменяет выбранный цвет или самый первый, если никакой из цветов палитры не выбран. Слайдеры в универсальном диалоге показываются всегда. Универсальный диалог выбора цвета доступен в Windows и других платформах под именем `wxGenericColourDialog`.

Рисунок 8.15 показывает родной диалог выбора цвета в GTK+.

Рисунок 8.17 показывает данный диалог в Mac OS X, который предоставляет

Рис. 8.13: wxDirDialog в системе Mac OS X

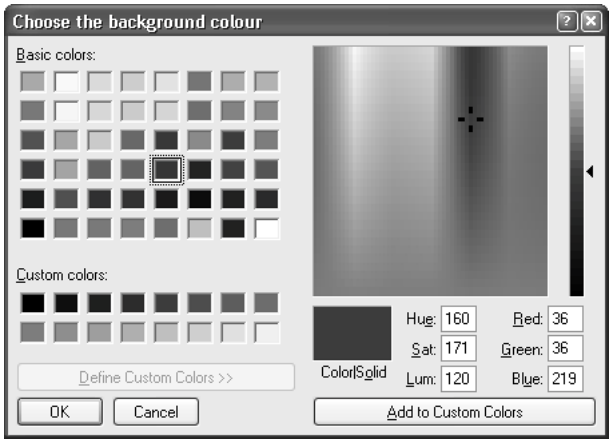

Рис. 8.14: wxColourDialog в системе Windows

другой способ получения цвета от пользователя.

Для использования диалога создайте объект класса `wxColourDialog` (в динамической памяти или на стеке) и передайте ему родительское окно и указатель на `wxColourData`. Информация из `wxColourData` будет использована, чтобы заполнить диалог значениями по умолчанию. Далее необходимо вызвать `ShowModal`, чтобы запустить модальный диалог, а когда управление вернется в ваш код вы сможете получить выбор пользователя, вызвав `GetColourData`.

Методы wxColourData

`SetChooseFull` говорит о том, что диалог должен быть полностью распахнутым. В данный момент флаг работает только под Windows. `GetChooseFull` позволяет получить значение данного флага.

`SetColour` устанавливает цвет по умолчанию, показываемый в диалоге, а `GetColour` получает цвет, выбранный пользователем.

`SetCustomColour` получает индекс (число от 0 до 15) и объект класса `wxColour` и

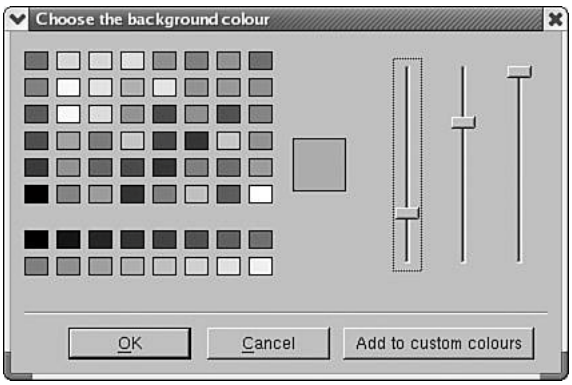

Рис. 8.15: Универсальный wxColourDialog под X11

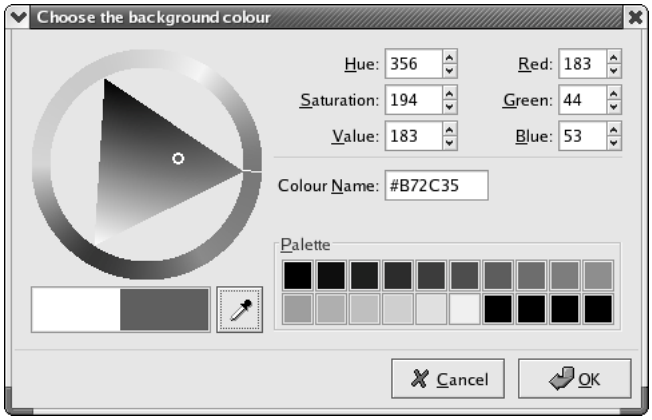

Рис. 8.16: wxColourDialog под GTK+

устанавливает один из 16 настраиваемых цветов в палитре. Метод `GetCustomColour` позволяет получить настраиваемый цвет, что имеет смысл, так как у пользователя есть возможность изменять эту палитру в диалоге.

Пример использования wxColourDialog

Ниже показан пример использования `wxColourDialog`. В нем устанавливаются различные параметры объекта `wxColourData`, а также создается палитра, состоящая из оттенков серого. Если пользователь не отменил диалог, то приложение получает выбранный цвет и использует его в качестве фона для окна.

```
#include "wx/colordlg.h"

wxColourData data;
data.SetChooseFull(true);
for (int i = 0; i < 16; i++)
{
    wxColour color(i*16, i*16, i*16);
    data.SetCustomColour(i, color);
```

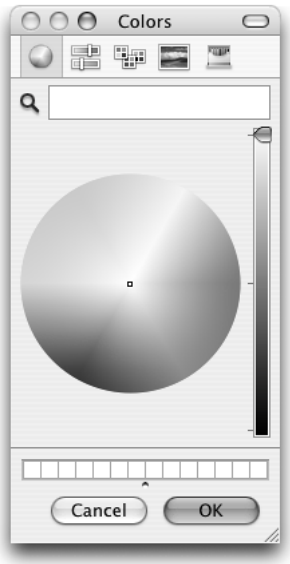

Рис. 8.17: wxColourDialog в системе Mac OS X

```
}

wxColourDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxColourData retData = dialog.GetColourData();
    wxColour col = retData.GetColour();
    myWindow->SetBackgroundColour(col);
    myWindow->Refresh();
}
```

8.3.2 wxFontDialog

`wxFontDialog` позволяет пользователю определить шрифт, а на некоторых платформах, также и его цвет шрифта.

В системе Windows используется стандартный диалог выбора шрифта. Этот диалог позволяет задать имя шрифта, размер, стиль, вес, подчеркивание, зачеркивание, а также цвет фона за шрифтом. Внизу диалога показывается образец получаемого шрифта. Заметим, что при конвертации шрифта из Windows в wxWidgets зачеркивание игнорируется, а также происходит замена шрифта (такого как Arial или Courier) на соответствующее семейство (такое как Swiss или Modern). В GTK+ также используется системный диалог выбора шрифта, который не позволяет выбрать цвет.

Рисунок 8.18 показывает диалог выбора шрифта в Windows.

Рисунок 8.19 показывает родной диалог выбора шрифта в GTK+.

На платформах, отличных от Windows и GTK+, диалог выглядит проще: на рисунке 8.18 изображен вид универсального диалога в Mac OS X. Есть возможность выбрать семейство шрифта, размер, стиль, вес, подчеркивание и цвет. В диалоге

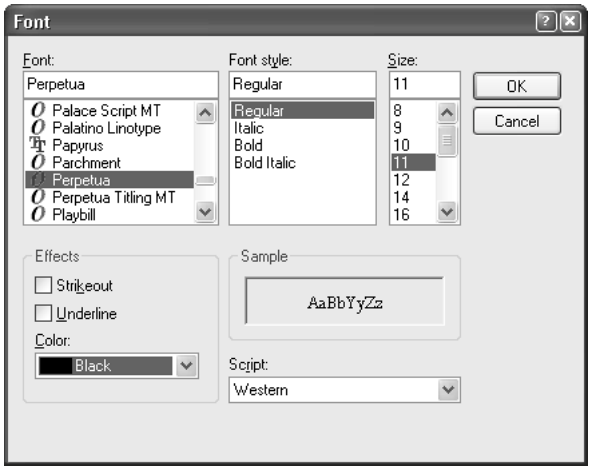

Рис. 8.18: wxFontDialog в системе Windows

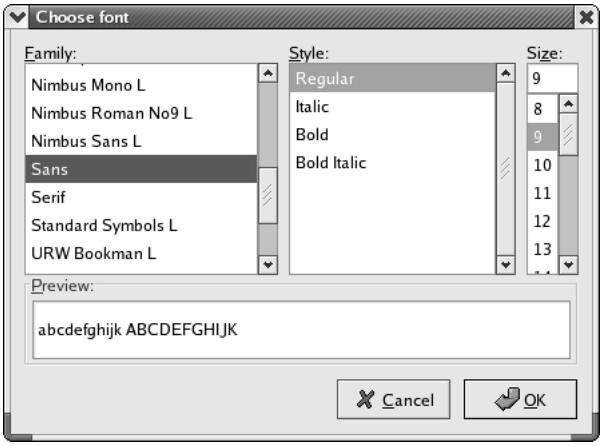

Рис. 8.19: wxFontDialog под GTK+

также есть предпросмотр. Универсальный диалог выбора шрифта доступен на всех платформах под именем `wxGenericFontDialog`.

Для использования `wxFontDialog` создайте этот объект динамически или на стеке и передайте ему родительское окно и объект класса `wxFontData`. Далее вызовете `ShowModal` и сравните, что возвращаемое значение равно `wxID_OK`. Далее можете вызвать `GetChosenFont` или `GetChosenColour` в зависимости от того, что вам требуется (шрифт или цвет).

Методы wxFontData

`EnableEffects` активирует возможность выбора цвета и подчеркивания в системе Windows или в универсальном диалоге (не имеет эффекта в GTK+). `GetEnableEffects` возвращает текущее булево значение данного свойства. Заметим, что даже при выключенных эффектах текущий цвет шрифта будет сохранен.

`SetAllowSymbols` позволяет выбрать шрифт для символа (только в системе Windows), а `GetAllowSymbols` позволяет получать текущее состояние этого свойства.

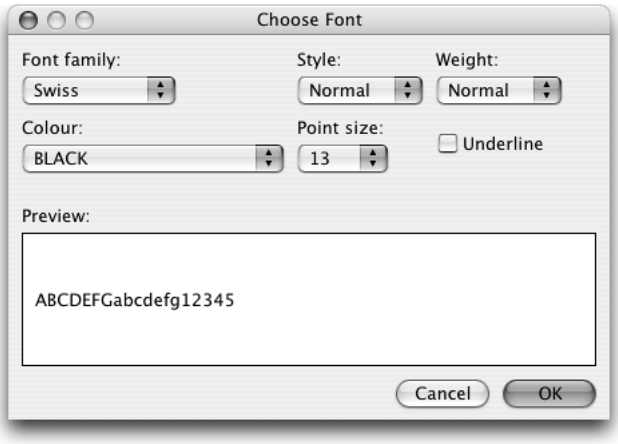

Рис. 8.20: Универсальный wxFontDialog в системе Mac OS X

`SetColour` устанавливает цвет шрифта по умолчанию, а `GetColour` позволяет получить цвет, выбранный пользователем.

`SetInitialFont` устанавливает шрифт по умолчанию, который будет выбран когда диалог будет откроется. `GetChosenFont` получает `wxFont`, выбранный пользователем.

`SetShowHelp` позволяет показать специальную кнопку помощи (только под Windows). `GetShowHelp` позволяет получить текущее значение этого свойства.

Вызов `SetRange` с минимальным и максимальным размерами (измеряется в точках) определяет размер шрифта, который пользователь может выбрать. Значение по умолчанию (0, 0) означает, что любые ограничения отсутствует. Данный метод имеет эффект только в системе Windows.

Пример кода для выбора шрифта

В следующем фрагменте кода приложение использует возвращаемый шрифт и цвет для отображения текста в окне:

```
#include "wx/fontdlg.h"

wxFontData data;
data.SetInitialFont(m_font);
data.SetColour(m_textColor);

wxFontDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxFontData retData = dialog.GetFontData();
    m_font = retData.GetChosenFont();
    m_textColor = retData.GetColour();

    // Обновить окно, чтобы отразить новый цвет и шрифт
```

```
    myWindow->Refresh();
}
```

8.3.3 wxSingleChoiceDialog

`wxSingleChoiceDialog` показывает пользователю список строк и предлагает ему выбрать из них одну. Примерный вид диалога показан на рисунке 8.21.

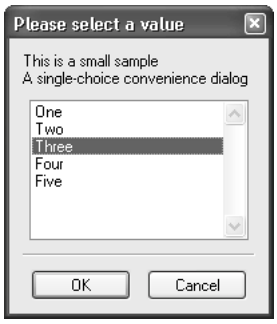

Рис. 8.21: wxSingleChoiceDialog в системе Windows

При создании в конструктор диалога передается родительское окно, сообщение, показываемое в диалоге, заголовок и `wxArrayString`, содержащий строки из которых диалог делает список. Вместо последнего параметра можно передать размер массива и C-шный массив строк (`wxChar**`).

Метод `SetSelection` позволяет установить выбор по умолчанию. После того как диалог будет закрыт вы сможете получить выбор пользователя с помощью метода `GetSelection` (возвращает индекс) или `GetStringSelection` (возвращает строку).

Также можно передать массив `char*` в качестве клиентских данных в конструктор. В таком случае после закрытия диалога `GetSelectionClientData` возвратит `char*` соответствующий выбору пользователя.

Пример использования wxSingleChoiceDialog

Ниже находится просто пример использования `wxSingleChoiceDialog`:

```
#include "wx/choicdlg.h"

const wxArrayString choices;
choices.Add(wxT("One"));
choices.Add(wxT("Two"));
choices.Add(wxT("Three"));
choices.Add(wxT("Four"));
choices.Add(wxT("Five"));

wxSingleChoiceDialog dialog(this,
    wxT("This is a small sample\nA single-choice convenience dialog"),
    wxT("Please select a value"),
    choices);
```

```
dialog.SetSelection(2);

if (dialog.ShowModal() == wxID_OK)
    wxMessageBox(dialog.GetStringSelection(), wxT("Got string"));
```

8.3.4 wxMultiChoiceDialog

`wxMultiChoiceDialog` очень похож на `wxSingleChoiceDialog`. Он также отображает пользователю список строк, но позволяет выбрать сразу несколько из них. Типичный вид диалога показан на рисунке 8.22.

Рис. 8.22: wxMultiChoiceDialog в системе Windows

При создании в конструктор диалога передается родительское окно, сообщение, показываемое в диалоге, заголовок и `wxArrayString`, содержащий строки из которых диалог делает список. Как и в случае с диалогом `wxSingleChoiceDialog`, вместо последнего параметра можно передать размер массива и C-шный массив строк (`wxChar**`). Но в отличие от `wxSingleChoiceDialog` в конструктор нельзя передать клиентские данные.

Для установки начального выбора необходимо вызвать `SetSelections`, передав ему `wxArrayInt`, где каждый элемент определяет индекс в переданном массиве строк. Получение выбора пользователя из диалога делается с помощью `GetSelections`, который возвращает `wxArrayInt` с индексами выбранных элементов.

Пример использования wxMultiChoiceDialog

Вот пример как можно использовать `wxMultiChoiceDialog`:

```
#include "wx/choicdlg.h"

const wxArrayString choices;
choices.Add(wxT("One"));
choices.Add(wxT("Two"));
```

```
choices.Add(wxT("Three"));
choices.Add(wxT("Four"));
choices.Add(wxT("Five"));

wxMultiChoiceDialog dialog(this,
                           wxT("A multi-choice convenience dialog"),
                           wxT("Please select several values"),
                           choices);

if (dialog.ShowModal() == wxID_OK)
{
    wxArrayInt selections = dialog.GetSelections();
    wxString msg;
    msg.Printf(wxT("You selected %u items:\n"),
        selections.GetCount());

    for ( size_t n = 0; n < selections.GetCount(); n++ )
    {
        msg += wxString::Format(wxT("\t%d: %d (%s)\n"),
                                n, selections[n],
                                choices[selections[n]].c_str());
    }

    wxMessageBox(msg, wxT("Got selections"));
}
```

8.4 Диалоги ввода

Диалоги данной группы просят пользователя ввести некоторую несложную информацию. К группе относятся `wxNumberEntryDialog`, `wxTextEntryDialog`, `wxPasswordEntryDialog` и `wxFindReplaceDialog`.

8.4.1 wxNumberEntryDialog

`wxNumberEntryDialog` запрашивает у пользователя целое число из некоторого интервала. Диалог показывает поле со спином, поэтому число можно ввести напрямую или с помощью нажатий на стрелочки вверх/вниз. Диалог реализован средствами wxWidgets, а поэтому имеет одинаковую функциональность на всех платформах.

При создании `wxNumberEntryDialog` ему передается родительское окно, текст сообщения, текст запроса (он предшествует контролу со спином), заголовок, начальное, минимальное и максимальное значения, а также расположение. После чего необходимо вызвать `ShowDialog` и, если он возвратит `wxID_OK`, то получить данные с помощь метода `GetValue`.

Рисунок 8.23 показывает вид диалога в системе Windows.

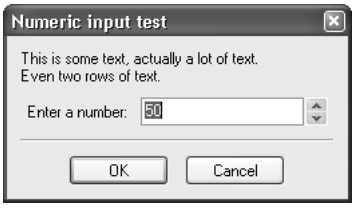

Рис. 8.23: wxNumberEntryDialog в системе Windows

Пример использования wxNumberEntryDialog

Рисунок 8.23 был создан с помощью следующего кода:

```
#include "wx/numdlg.h"

wxNumberEntryDialog dialog(parent,
  wxT("This is some text, actually a lot of text\nEven two rows of text"),
  wxT("Enter a number:"), wxT("Numeric input test"), 50, 0, 100);
if (dialog.ShowModal() == wxID_OK)
{
  long value = dialog.GetValue();
  ...
}
```

8.4.2 wxTextEntryDialog и wxPasswordEntryDialog

`wxTextEntryDialog` и `wxPasswordEntryDialog` — это диалоги с одним полем для редактирования и сообщением. Их методы полностью идентичны, за исключением того, что текст, набираемый в `wxPasswordEntryDialog`, скрывается, а поэтому не может быть прочитан. Рисунок 8.24 показывает вид диалога `wxTextEntryDialog` в системе Windows.

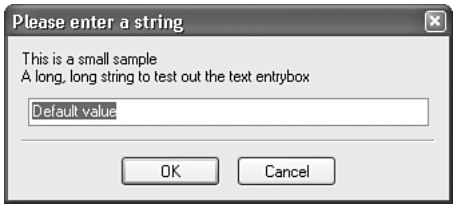

Рис. 8.24: wxTextEntryDialog в системе Windows

В конструктор передается родительское окно, сообщение, заголовок, начальное значение и стиль. Стиль является комбинацией флагов наподобие `wxOK`, `wxCANCEL` и `wxCENTRE` (или `wxCENTER`), а также специфичных для `wxTextCtrl` стилей, таких как `wxTE_CENTRE` (или `wxTE_CENTER`).

Значение по умолчанию можно установить и отдельно с помощью метода `SetValue`. Метод `GetValue` позволяет получить введенное пользователем значение.

Пример использования wxTextEntryDialog

Диалог на рисунке 8.24 был создан с помощью следующего кода:

```
#include "wx/textdlg.h"

wxTextEntryDialog dialog(this,
    wxT("This is a small sample\n")
    wxT("A long, long string to test out the text entrybox"),
    wxT("Please enter a string"),
    wxT("Default value"),
    wxOK | wxCANCEL);

if (dialog.ShowModal() == wxID_OK)
    wxMessageBox(dialog.GetValue(), wxT("Got string"));
```

8.4.3 wxFindReplaceDialog

`wxFindReplaceDialog` является немодальным диалогом, позволяющим пользователю искать некоторый текст и заменять его на что-то другое (если требуется). Сам поиск должен быть реализован в классе-наследнике, либо в родительском окне — соответствующие события генерируются диалогом. В отличие от большинства стандартных диалогов у этого обязательно должно быть определено родительское окно. Диалог также нельзя использовать модально, он всегда (по дизайну и реализации) является немодальным.

Вид диалога поиска и замены в системе Windows показан на рисунке 8.25.

Рис. 8.25: wxFindReplaceDialog в системе Windows

На других платформах, таких как GTK+ или Mac OS X, wxWidgets использует универсальную версию диалога, показанную на рисунке 8.26.

Обработка событий от диалога

`wxFindReplaceDialog` генерирует события, если пользователь щелкает на элементах управления диалога. Обработчики этих событий принимают в качестве параметра объект класса `wxFindDialogEvent`. Макросы для таблицы событий, получающие идентификатор диалога и функцию-обработчик, указаны в таблице 8.4.

Рис. 8.26: wxFindReplaceDialog под GTK+

Таблица 8.4: События wxFindReplaceDialog

Имя	Описание
`EVT_FIND(id, func)`	Обрабатывает нажатие на кнопку «Find».
`EVT_FIND_NEXT(id, func)`	Обрабатывает нажатие на кнопку «Next».
`EVT_FIND_REPLACE(id, func)`	Обрабатывает нажатие на кнопку «Replace».
`EVT_FIND_REPLACE_ALL(id, func)`	Обрабатывает нажатие на кнопку «Replace All».
`EVT_FIND_CLOSE(id, func)`	Обрабатывает событие, которое генерируется если пользователь закрыл диалог, нажав «Отмена» или другим подобным способом.

Методы wxFindDialogEvent

`wxFindDialogEvent` имеет следующие методы:

`GetFlags` возвращает флаги для текущего выбора в диалоге. Является комбинацией бинарных флагов `wxFR_DOWN`, `wxFR_WHOLEWORD` и `wxFR_MATCHCASE`.

`GetFindString` возвращает строку для поиска, которую ввел пользователь.

`GetReplaceString` возващает строку замены, которую ввел пользователь.

`GetDialog` возвращает указатель на `wxFindReplaceDialog`, который сгенерировал данное событие.

Передача данных диалогу

При создании `wxFindReplaceDialog` ему передается родительское окно, указатель на объект класса `wxFindReplaceData`, заголовок диалога и стиль, который является комбинацией бинарных флагов, указанных в таблице 8.5.

Таблица 8.5: Стили wxFindReplaceData

Имя	Описание
`wxFR_REPLACEDIALOG`	Определяет диалог для поиска и замены. Без указания флага будет создан диалог, предназначенный для поиска.
`wxFR_NOUPDOWN`	Определяет, что направление поиска нельзя изменять.
`wxFR_NOMATCHCASE`	Определяет, что пользователь не сможет изменить чувствительность к регистру.

Он также обновляется каждый раз при генерации события `wxFindDialogEvent`, поэтому вместо использования методов `wxFindDialogEvent`, можно получать нужные значения напрямую от объекта. Метод `GetData` позволяет получить указатель на объект, который был передан в конструктор диалога.

Методы wxFindReplaceData

Перечисленные далее методы предназначены для установки и получения данных из `wxFindReplaceData`. Заметим, что методы установки данных можно вызывать только перед тем как диалог будет показан. После этого их вызов будет игнорироваться.

`GetFindString` и `SetFindString` позволяют получить доступ к строке поиска, определенной приложением или введенной пользователем.

`GetFlags` и `SetFlags` позволяют получить доступ к флагам, определяющим состояние диалога (таблица 8.5).

`GetReplaceString` и `SetReplaceString` позволяют получить доступ к строке замены, определенной приложением или введенной пользователем.

Пример реализации поиска и замены

Далее показан фрагмент примера работы с `wxFindReplaceDialog`, использующий гипотетические функции `DoFind` и `DoReplace`, которые предназначены для реального поиска и замены в приложении. Эти функции должны поддерживать зависящие от приложения переменные, хранящие последнюю найденную позицию, поэтому при каждом вызове данных функций будет найден следующий фрагмент текста. Функции также меняют позицию в документе и делают подсветку найденного значения.

```
#include "wx/fdrepdlg.h"

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
     EVT_MENU(ID_REPLACE, MyFrame::ShowReplaceDialog)
     EVT_FIND(wxID_ANY, MyFrame::OnFind)
     EVT_FIND_NEXT(wxID_ANY, MyFrame::OnFind)
     EVT_FIND_REPLACE(wxID_ANY, MyFrame::OnReplace)
     EVT_FIND_REPLACE_ALL(wxID_ANY, MyFrame::OnReplaceAll)
     EVT_FIND_CLOSE(wxID_ANY, MyFrame::OnFindClose)
END_EVENT_TABLE()

void MyFrame::ShowReplaceDialog( wxCommandEvent& event )
{
    if ( m_dlgReplace )
    {
        delete m_dlgReplace;
        m_dlgReplace = NULL;
    }
    else
    {
```

```
        m_dlgReplace = new wxFindReplaceDialog
                           (
                            this,
                            &m_findData,
                            wxT("Find and replace dialog"),
                            wxFR_REPLACEDIALOG
                           );

        m_dlgReplace->Show(true);
    }
}

void MyFrame::OnFind(wxFindDialogEvent& event)
{
    if (!DoFind(event.GetFindString(), event.GetFlags()))
    {
        wxMessageBox(wxT("No more matches."));
    }
}

void MyFrame::OnReplace(wxFindDialogEvent& event)
{
    if (!DoReplace(event.GetFindString(), event.GetReplaceString(),
        event.GetFlags(), REPLACE_THIS))
    {
        wxMessageBox(wxT("No more matches."));
    }
}

void MyFrame::OnReplaceAll(wxFindDialogEvent& event)
{
    if (DoReplace(event.GetFindString(), event.GetReplaceString(),
        event.GetFlags(), REPLACE_ALL))
    {
        wxMessageBox(wxT("Replacements made."));
    }
    else
    {
        wxMessageBox(wxT("No replacements made."));
    }

}

void MyFrame::OnFindClose(wxFindDialogEvent& event)
{
    m_dlgReplace->Destroy();
    m_dlgReplace = NULL;
```

```
}
```

8.5 Диалоги печати

Диалоги `wxPageSetupDialog` и `wxPrintDialog` используются в приложениях, у которых есть необходимость в печати документов. Если вы используете подсистему печати wxWidgets (включающую `wxPrintout`, `wxPrinter` и другие классы), то у вас не будет необходимости явно вызывать эти диалоги в своем коде. За дополнительной информацией о подсистеме печати обратитесь к главе 5 «Рисование и печать».

8.5.1 wxPageSetupDialog

`wxPageSetupDialog` содержит элементы управления для установки размера страницы (A4, письмо и т.п.), ориентации (книжная или альбомная), а также размеров отступа слева, сверху, снизу и справа. Пользователь с помощью специальной кнопки также может установить специфичные для принтера опции.

Рисунок 8.27 показывает вид диалога `wxPageSetupDialog` в системе Windows.

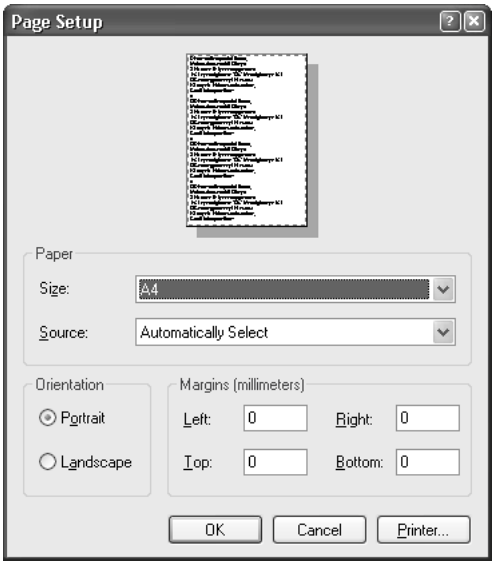

Рис. 8.27: wxPageSetupDialog в системе Windows

Рисунок 8.28 показывает `wxPageSetupDialog`, использующий универсальную реализацию под GTK+. Если установлены библиотеки печати для GNOME, то wxWidgets будет использовать родной диалог настройки страниц GNOME, показанный на рисунке 8.29.

Версия диалога в системе Mac OS X показана на рисунке 8.30.

Чтобы создать этот диалог передайте в его конструктор родительское окно и указатель на объект класса `wxPageSetupDialogData`, который будет содержать начальные данные и возвращать обратно выбор пользователя. Диалог можно создать на стеке или в динамической памяти. При создании диалог копирует передаваемые ему настройки — с помощью `GetPageSetupData` вы можете получить ссылку на внутренние данные диалога.

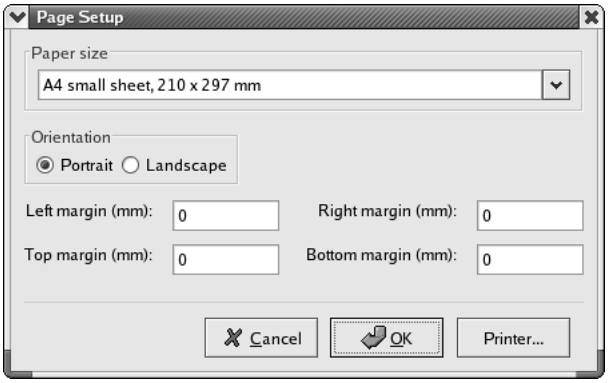

Рис. 8.28: wxPageSetupDialog в GTK+ без поддержки системы печати GNOME

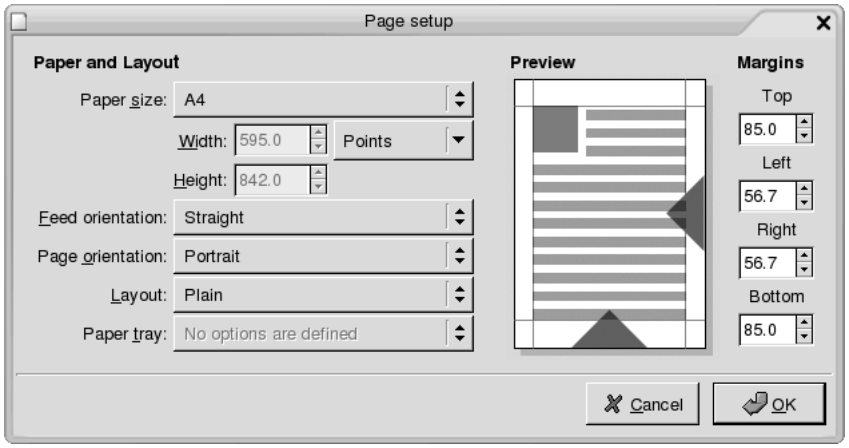

Рис. 8.29: wxPageSetupDialog в GTK+ с поддержкой системы печати GNOME

Методы wxPageSetupData

`wxPageSetupDialogData` поддерживает следующие методы:

`Ok` возвращает `true`, если данные, ассоциированные с объектом, являются корректными. Например, метод может возвратит `false` в системе Windows, если в системе не установлен принтер. На всех других платформах он возвращает `true`.

`SetMarginTopLeft` получает на вход объект класса `wxPoint` и устанавливает отступы слева и сверху (задается в миллиметрах). Вызов `GetMarginTopLeft` позволяет получить указанное значение.

`SetMarginBottomRight` получает на вход объект класса `wxPoint` и устанавливает отступы снизу и справа (задается в миллиметрах). Вызов `GetMarginBottomRight` позволяет получить указанное значение.

`SetPaperId` устанавливает идентификатор бумаги, который определяет размеры бумаги. Данный метод используется вместо `SetPaperSize`. Обратитесь к документации, чтобы получить список доступных идентификаторов. `GetPaperId` получает идентификатор бумаги.

`SetPaperSize` получает на вход объект класса `wxSize`, который содержит размер бумаги в миллиметрах. Используйте метод `GetPaperSize` для получения указанного

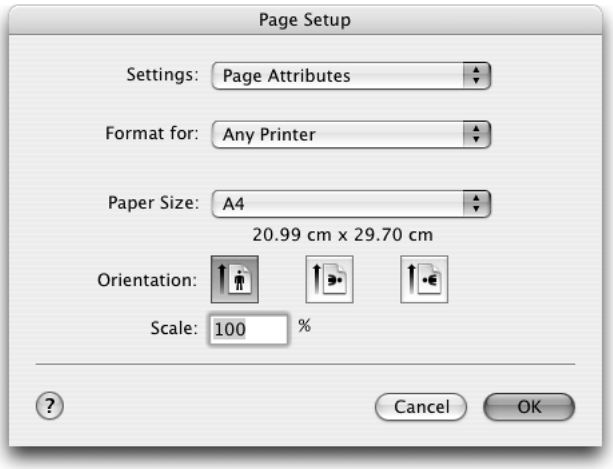

Рис. 8.30: wxPageSetupDialog в системе Mac OS X

значения.

`EnableMargins` включает и выключает показ настроек отступов (в версии только для Windows). Вызов `GetEnableMargins` позволяет получить значение данной настройки.

`EnableOrientation` включает и выключает показ настроек ориентации страницы (в версии только для Windows). Вызов `GetEnableOrientation` позволяет получить значение данной настройки.

`EnablePaper` включает и выключает показ настроек размера страницы (в версии только для Windows). Вызов `GetEnablePaper` позволяет получить значение данной настройки.

`EnablePrinter` включает и выключает показ специальной кнопки «Принтер», которая обеспечивает доступ к специальным настройкам принтера. Вызов `GetEnablePrinter` позволяет получить значение данной настройки.

Пример использования wxPageSetupDialog

Вот пример кода, использующего `wxPageSetupDialog`:

```
#include "wx/printdlg.h"

void MyFrame::OnPageSetup(wxCommandEvent& event)
{
    wxPageSetupDialog pageSetupDialog(this, & m_pageSetupData);
    if (pageSetupDialog.ShowModal() == wxID_OK)
        m_pageSetupData = pageSetupDialog.GetPageSetupData();
}
```

8.5.2 wxPrintDialog

Данный класс представляет стандартный диалог печати и ее настройки. После успешного закрытия диалога вы получите правильный контекст устройства

`wxPrinterDC`.

Рисунок 8.31 показывает вид `wxPrintDialog` в системе Windows.

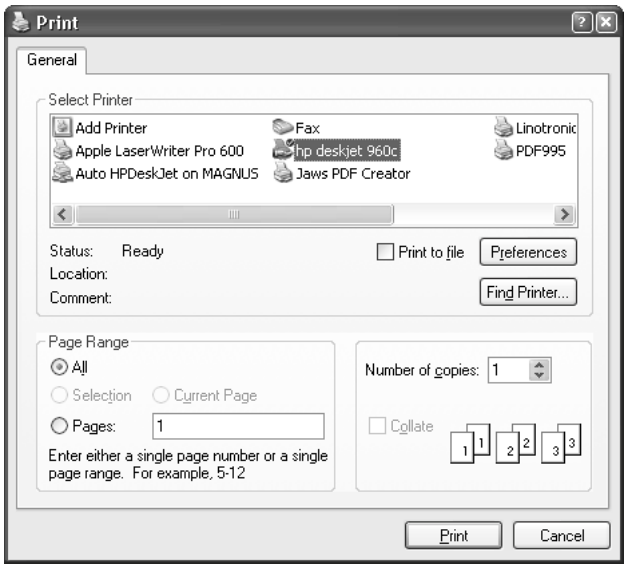

Рис. 8.31: wxPrintDialog в системе Windows

Рисунок 8.32 показывает `wxPrintDialog` в GTK+ без библиотек поддержки печати в GNOME, а рисунок 8.33 показывает тот же диалог, если эти библиотеки будут установлены.

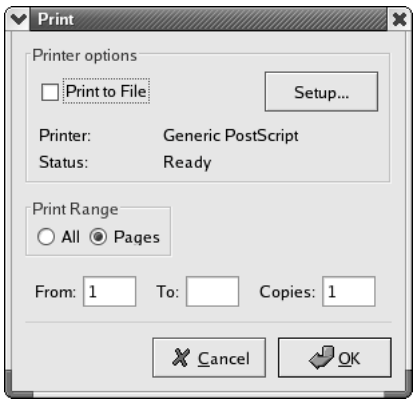

Рис. 8.32: wxPrintDialog в GTK+ без поддержки системы печати GNOME

Рисунок 8.34 показывает вид `wxPrintDialog` в системе Mac OS X. Если вы посмотрите на кнопки внизу диалога, то заметите, что Mac OS X дает дополнительную возможность сохранить ваш документ в формате PDF-файла, а также использовать стандартный предпросмотр Mac OS X в качестве альтернативы окну предпросмотра приложения.

Создайте `wxPrintDialog` в динамической памяти или на стеке, передайте его конструктору родительское окно и указатель на объект класса `wxPrintDialogData`, содержимое которого будет скопировано во внутренние данные диалога. Вызовете

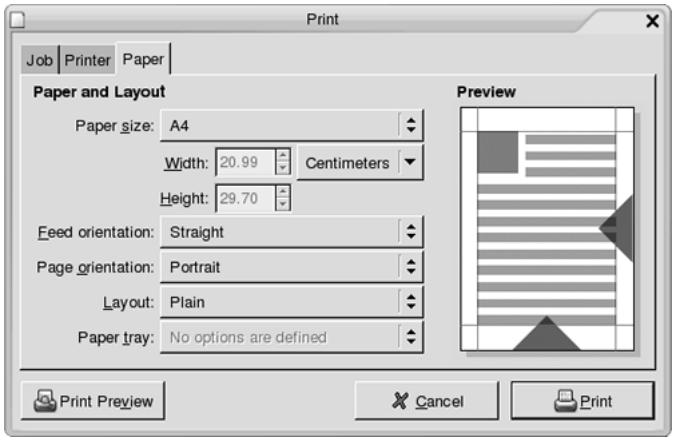

Рис. 8.33: wxPrintDialog в GTK+ с поддержкой системы печати GNOME

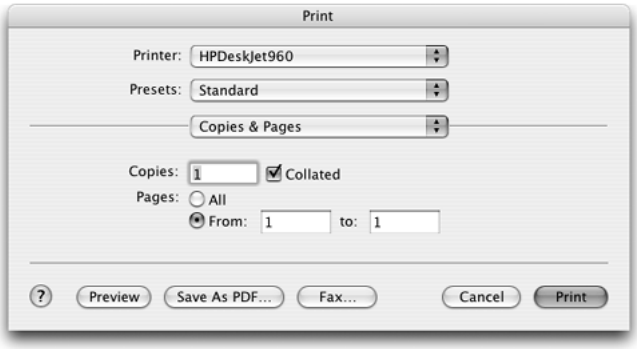

Рис. 8.34: wxPrintDialog в системе Mac OS X

метод `wxPrintDialogData::SetSetupDialog` с `true` до того, как передадите объект диалогу, если хотите показать диалог настройки печати вместо обычного диалога печати. Следуя рекомендациям компании Microsoft, диалог настройки печати был заменен диалогом `wxPageSetupDialog`, однако для совместимости некоторым приложениям может потребоваться диалог настройки.

После того, как диалог успешно завершится, вы сможете получить `wxPrintDialogData`, используя метод `GetPrintDialogData`.

Метод `GetPrintDC` диалога позволяет получить контекст принтера с выбранными пользователем настройками. Если этот метод возвратит ненулевой указатель, то приложение станет ответственным за удаление возвращенного контекста устройства.

`Ok` возвращает `true`, если данные, ассоциированные с диалогом, являются корректными. Например, метод может возвратить `false`, если в системе Windows отсутствует настроенный принтера. На других платформах метод возвращает `true`.

Методы wxPrintDialogData

Здесь перечислены наиболее часто используемые методы класса `wxPrintDialogData`.

`EnableHelp` показывает или скрывает кнопку «Помощь». Метод `GetEnableHelp` используется для получения этого значения.

`EnablePageNumbers` показывает или скрывает элемент для управления нумерацией страниц, а `GetEnablePageNumbers` позволяет получить указанное значение.

`EnablePrintToFile` показывает или скрывает флажок «Печать в файл». Используйте `GetEnablePrintToFile` для получения данного значения.

`EnableSelection` показывает или скрывает радио-кнопку «Диапазон печати», которая позволяет пользователю определить страницы для печати. Метод `GetEnableSelection` возвращает значение данной настройки.

`SetCollate` устанавливает значение флажка «Разобрать по копиям» в `true` или `false`. Метод `GetCollate` возвращает значение данной настройки.

`SetFromPage` и `SetToPage` позволяют определить диапазон печати для страниц. Методы `GetFromPage` и `GetToPage` позволяют получить этот диапазон.

`SetMinPage` и `SetMaxPage` устанавливают минимальный и максимальный номера страниц, которые можно напечатать. `GetMinPage` и `GetMaxPage` используются для получения этих значений.

`SetNoCopies` устанавливает начальное количество копий, которые необходимо напечатать. Метод `GetNoCopies` возвращает значение данной настройки.

`SetPrintToFile` устанавливает значение для флажка «Печать в файл» в `true` или `false`. Используйте `GetPrintToFile` для получения значения данной настройки.

`SetSelection` устанавливает значение для радио-кнопки «Диапазон печати». Используйте `GetSelection` для получения значения данной настройки.

`SetSetupDialog` указывает, что необходимо показать диалог настройки печати (`true`) или обычный диалог печати (`false`). Используйте `GetSetupDialog` для получения значения данной настройки.

`SetPrintData` устанавливает внутренний объект класса `wxPrintData`. `GetPrintData` позволяет получить ссылку на внутренний объект класса `wxPrintData`.

Пример использования wxPrintDialog

Следующий пример показывает использование `wxPrintDialog` для получения подходящего контекста устройства принтера:

```
#include "wx/printdlg.h"

void MyFrame::OnPrint(wxCommandEvent& event)
{
    wxPrintDialogData dialogData;
    dialogData.SetFromPage(0);
    dialogData.SetToPage(10);

    wxPrintDialog printDialog(this, & m_dialogData);
    if (printDialog.ShowModal() == wxID_OK)
    {
        // После вызова GetPrintDC() приложение начинает
        // владеть возвращаемым контекстом
```

```
        wxDC* dc = printDialog.GetPrintDC();

        // Рисуем на контексте устройства
        ...

        // Уничтожаем его
        delete dc;
    }
}
```

Однако, обычно вы можете избежать явного вызова диалогов печати. Вместо этого рекомендуется использовать высокоуровневую подсистему печати (обратитесь к Главе 5), тогда диалог печати будет показан как побочных эффект вызова метода `wxPrinter::Print`.

8.6 Заключение

В этой главе вы научились использовать стандартные диалоги для представления информации и получения выбора пользователя с помощью небольшого количества кода. За дополнительными примерами использования обратитесь к примеру samples/dialogs в дистрибутиве wxWidgets. В следующей главе мы расскажем как написать ваш собственный диалог.