Глава 9

Написание собственных диалогов

Рано или поздно вам понадобится создать собственный диалог, будь то простой диалог, состоящий из текста и нескольких кнопок, или сложный диалог с вкладками, множеством панелей, собственными элементами управления, контекстной помощью и т.п. В этой главе мы рассмотрим основные принципы создания диалогов, а также передачу данных между переменными C++ и элементами управления. Также будет рассказано об использовании ресурсов, которые позволяют загружать диалоги и другие элементы интерфейса из XML-файлов.

9.1 Шаги для создания собственных диалогов

Самое интересное программирование начинается, когда вы начнете создавать свои собственные диалоги. Вот список шагов, которые необходимо для этого сделать:

1. Создать наследника от `wxDialog`.

2. Решить, где будут храниться данные диалога и как приложение получит к ним доступ.

3. Написать код создания и размещения элементов управления.

4. Добавить код, отвечающий за пересылку данных между данными диалога и элементами управления.

5. Написать обработчики событий от элементов управления.

6. Добавить обработку обновления пользовательского интерфейса, чтобы элементы управления всегда были в правильном состоянии.

7. Добавить помощь, в частности всплывающие подсказки, контекстную помощь (данная возможность не реализована в Mac OS X) и справочную систему.

8. Вызвать диалог из подходящего места вашего приложения.

Проиллюстрируем указанные шаги на конкретном примере.

Рис. 9.1: Диалог с пользовательской информацией в Windows

9.2 Пример: PersonalRecordDialog

Как вы знаете из прошлой главы диалоги бывают двух видов: модальные и не модальные. Мы будем делать модальный диалог, так как это более распространенный и менее сложный в реализации тип. Приложение вызывает диалог с помощью `ShowModal`, а далее получает из диалога выбор пользователя. До того как `ShowModal` возвратит результат все взаимодействие с пользователем заключено в маленький мир внутри вашего диалога (и любых других модальных диалогов, которые ваш диалог может вызвать).

Многие шаги по созданию собственного диалога можно очень легко выполнить с использованием редакторов диалогов, таких как wxDesigner или DialogBlocks. С их помощью, в зависимости от сложности диалога, можно в автоматическом режиме выполнить очень большой объем работы по написанию кода. В этой главе исключительно для демонстрации базовых принципов мы все будем все делать вручную. Однако в реальной работе рекомендуется использовать такого рода инструменты для экономии кучи часов повторяющейся работы.

Мы проиллюстрируем шаги, необходимые для создания диалога с помощью диалога в котором пользователь вводит свое имя, возраст, пол и хочет ли он проголосовать. Диалог называется `PersonalRecordDialog` и показан на рис. 9.1.

Кнопка «Reset»[1] устанавливает значение всех элементов управления в их начальные значения. Кнопка «OK» закрывает диалог и возвращает `wxID_OK` из `ShowModal`. Кнопка «Cancel» возвращает `wxID_CANCEL` и не обновляет содержимое переменных диалога. Кнопка «Help»[2] выводит несколько строк с описанием диалога, (хотя в реальном приложении эта кнопка должна вызывать полноценную страницу помощи в справочной системе).

Хороший пользовательский интерфейс не должен позволять пользователю вводить данные, которые не имеют значения в текущей ситуации. Например, пользователь не должен иметь возможность использовать элемент управления «Vote»[3], если его возраст меньше чем допустимый для голосования возраст (18 лет для США или Великобритании). Поэтому необходимо убедиться, что при возрасте менее 18 лет чекбокс Vote заблокирован для ввода.

[1] Сброс (англ.)
[2] Помощь (англ.)
[3] Голосовать (англ.)

9.2.1 Создаем новый класс-наследник

Далее приведено объявление нашего PersonalRecordDialog. Информация времени выполнения о типе вводится с помощью макроса DECLARE_CLASS, а добавление таблицы событий — с помощью DECLARE_EVENT_TABLE.

```
/*!
 * Объявление класса PersonalRecordDialog
 */

class PersonalRecordDialog: public wxDialog
{
    DECLARE_CLASS( PersonalRecordDialog )
    DECLARE_EVENT_TABLE()

public:
    // Конструкторы
    PersonalRecordDialog( );
    PersonalRecordDialog( wxWindow* parent,
      wxWindowID id = wxID_ANY,
      const wxString& caption = wxT("Personal Record"),
      const wxPoint& pos = wxDefaultPosition,
      const wxSize& size = wxDefaultSize,
      long style = wxCAPTION|wxRESIZE_BORDER|wxSYSTEM_MENU );

    // Инициализация наших переменных
    void Init();

    // Создание
    bool Create( wxWindow* parent,
      wxWindowID id = wxID_ANY,
      const wxString& caption = wxT("Personal Record"),
      const wxPoint& pos = wxDefaultPosition,
      const wxSize& size = wxDefaultSize,
      long style = wxCAPTION|wxRESIZE_BORDER|wxSYSTEM_MENU );

    // Создание элементов управления и сайзеров
    void CreateControls();
};
```

Заметим, что следуя принятому в wxWidgets соглашению позволять одно- и двушаговое конструирование мы предоставляем конструктор по умолчанию и функцию Create, а также расширенный конструктор.

9.2.2 Проектируем хранение данных

У нас есть четыре части данных, которые нужно хранить: имя (строка), возраст (целое), пол (логическое) и предпочтения по голосованию (логическое). Чтобы

упростить использование элемента управления `wxChoice` мы будем использовать целочисленный тип для хранения логического типа для пола, но интерфейс класса будет представлять его как логическое значение: `true` для женского пола и `false` для мужского. Давайте добавим эти данные и методы для работы с ними в класс `PersonalRecordDialog`:

```
// Данные
wxString    m_name;
int         m_age;
int         m_sex;
bool        m_vote;

// Доступ к имени
void SetName(const wxString& name) { m_name = name; }
wxString GetName() const { return m_name; }

// Доступ к возрасту
void SetAge(int age) { m_age = age; }
int GetAge() const { return m_age; }

// Доступ к полу (мужской = false, женский = true)
void SetSex(bool sex) { sex ? m_sex = 1 : m_sex = 0; }
bool GetSex() const { return m_sex == 1; }

// Будет ли пользователь голосовать?
void SetVote(bool vote) { m_vote = vote; }
bool GetVote() const { return m_vote; }
```

9.2.3 Создание элементов управления и их размещение

Теперь добавим функцию `CreateControls`, которая будет вызываться из `Create`. `CreateControls` добавляет несколько элементов управления типа `wxStaticText`, `wxButton`, `wxSpinCtrl`, `wxTextCtrl`, `wxChoice` и `wxCheckBox`. Обратитесь к рисунку 9-1, чтобы посмотреть результирующий диалог.

Мы используем основанный на сайзерах макет для этого диалога, из-за чего код выглядит достаточно сложно для такого небольшого числа элементов управления (мы уже коротко описывали сайзеры в Главе 7 «Размещение элементов с помощью сайзеров», где рассказывали, что они позволяются создавать диалоги, которые нормально выглядят на множестве платформ и которые легко адаптировать для перевода и изменения размера). Вы можете использовать и другие методы, такие как загрузка диалога из ресурсов wxWidgets (XRC-файлов).

Основной принцип основанного на сайзерах размещения — поместить управляющие элементы во вложенные боксы (сайзеры), которые распределят место между элементами управления или растянуться, чтобы стать достаточными для их размещения. Сайзеры не являются окнами, они формируют отдельную иерархию и поэтому элементы управления остаются детьми своих родителей, независимо от сложности иерархии сайзеров. Вы можете освежить свою память и посмотреть на схематический вид размещения сайзеров, который показан на рисунке 7-2 в Главе 7.

В CreateControls мы используем вертикальный сайзер (boxSizer), вложенный в другой вертикальный сайзер (topSizer), чтобы получить достаточный отступ между элементам управления и границами диалога. Один горизонтальный сайзер используется для размещения wxSpinCtrl, wxChoice и wxCheckBox, а другой (okCancelSizer) — для кнопок Reset, OK, Cancel и Help.

```
/*!
 * Control creation for PersonalRecordDialog
 */
void PersonalRecordDialog::CreateControls()
{
    // Сайзер верхнего уровня

    wxBoxSizer* topSizer = new wxBoxSizer(wxVERTICAL);
    this->SetSizer(topSizer);

    // Второй сайзер, чтобы получить больше пространства вокруг элементов

    wxBoxSizer* boxSizer = new wxBoxSizer(wxVERTICAL);
    topSizer->Add(boxSizer, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

    // Некоторый текст

    wxStaticText* descr = new wxStaticText( this, wxID_STATIC,
        wxT("Please enter your name, age and sex, and specify whether\
        you wish to\nvote in a general election."),
        wxDefaultPosition, wxDefaultSize, 0 );
    boxSizer->Add(descr, 0, wxALIGN_LEFT|wxALL, 5);

    // Отступ

    boxSizer->Add(5, 5, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

    // Метка для текстового элемента

    wxStaticText* nameLabel = new wxStaticText ( this, wxID_STATIC,
        wxT("&Name:"), wxDefaultPosition, wxDefaultSize, 0 );
    boxSizer->Add(nameLabel, 0, wxALIGN_LEFT|wxALL, 5);

    // Тестовый элемент для получения имени пользователя

    wxTextCtrl* nameCtrl = new wxTextCtrl ( this, ID_NAME, wxT("Emma"),
        wxDefaultPosition, wxDefaultSize, 0 );
    boxSizer->Add(nameCtrl, 0, wxGROW|wxALL, 5);

    // Горизонтальный сайзер, содержащий возраст, пол и флаг голосования
```

```
wxBoxSizer* ageSexVoteBox = new wxBoxSizer(wxHORIZONTAL);
boxSizer->Add(ageSexVoteBox, 0, wxGROW|wxALL, 5);

// Метка для возраста

wxStaticText* ageLabel = new wxStaticText ( this, wxID_STATIC,
    wxT("&Age:"), wxDefaultPosition, wxDefaultSize, 0 );
ageSexVoteBox->Add(ageLabel, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

// Спиновый элемент для получения возраста

wxSpinCtrl* ageSpin = new wxSpinCtrl ( this, ID_AGE,
    wxEmptyString, wxDefaultPosition, wxSize(60, -1),
    wxSP_ARROW_KEYS, 0, 120, 25 );
ageSexVoteBox->Add(ageSpin, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

// Метка для пола

wxStaticText* sexLabel = new wxStaticText ( this, wxID_STATIC,
    wxT("&Sex:"), wxDefaultPosition, wxDefaultSize, 0 );
ageSexVoteBox->Add(sexLabel, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

// Создаем выпадающий список для выбора пола
wxString sexStrings[] = {
    wxT("Male"),
    wxT("Female")
};

wxChoice* sexChoice = new wxChoice ( this, ID_SEX,
    wxDefaultPosition, wxSize(80, -1), WXSIZEOF(sexStrings),
        sexStrings, 0 );
sexChoice->SetStringSelection(wxT("Female"));
ageSexVoteBox->Add(sexChoice, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

// Добавляем растяжимый отступ, который перемещает элемент для
// голосования направо

ageSexVoteBox->Add(5, 5, 1, wxALIGN_CENTER_VERTICAL|wxALL, 5);

wxCheckBox* voteCheckBox = new wxCheckBox( this, ID_VOTE,
   wxT("&Vote"), wxDefaultPosition, wxDefaultSize, 0 );
voteCheckBox ->SetValue(true);
ageSexVoteBox->Add(voteCheckBox, 0,
    wxALIGN_CENTER_VERTICAL|wxALL, 5);

// Создаем разделяющую черту перед кнопками OK и Cancel
```

```
    wxStaticLine* line = new wxStaticLine ( this, wxID_STATIC,
        wxDefaultPosition, wxDefaultSize, wxLI_HORIZONTAL );
    boxSizer->Add(line, 0, wxGROW|wxALL, 5);

    // Горизонтальный сайзер содержит кнопки Reset, OK, Cancel и Help

    wxBoxSizer* okCancelBox = new wxBoxSizer(wxHORIZONTAL);
    boxSizer->Add(okCancelBox, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

    // Кнопка Reset

    wxButton* reset = new wxButton( this, ID_RESET, wxT("&Reset"),
        wxDefaultPosition, wxDefaultSize, 0 );
    okCancelBox->Add(reset, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

    // Кнопка OK

    wxButton* ok = new wxButton ( this, wxID_OK, wxT("&OK"),
        wxDefaultPosition, wxDefaultSize, 0 );
    okCancelBox->Add(ok, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

    // Кнопка Cancel

    wxButton* cancel = new wxButton ( this, wxID_CANCEL,
        wxT("&Cancel"), wxDefaultPosition, wxDefaultSize, 0 );
    okCancelBox->Add(cancel, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

    // Кнопка Help

    wxButton* help = new wxButton( this, wxID_HELP, wxT("&Help"),
        wxDefaultPosition, wxDefaultSize, 0 );
    okCancelBox->Add(help, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);
}
```

9.2.4 Пересылка данных и их проверка

Теперь у нас в диалоге есть несколько элементов управления, но они пока не соеденены с переменными из диалога. Как можно реализовать такую связь?

Когда диалог показывается в первый раз wxWidgets вызывает `InitDialog`, который в свою очередь генерирует событие `wxEVT_INIT_DIALOG`. Обработчик по умолчанию для данного события вызывает `TransferDataToWindow` для диалога. Чтобы переслать информацию из элементов управления обратно в переменные вы должны вызвать `TransferDataFromWindow`, когда пользователь подтвердит свой выбор. Тоже самое делает wxWidgets в обработчике по умолчанию для управляющего события `wxID_OK`, который вызывает `TransferDataFromWindow` перед вызовом метода `EndModal` (который непосредственно закрывает диалог).

Поэтому вы можете переопределить `TransferDataToWindow` и

TransferDataFromWindow для пересылки ваших данных. Для нашего диалога код может выглядеть следующим образом:

```
/*!
 * Посылаем данные в окно
 */

bool PersonalRecordDialog::TransferDataToWindow()
{
    wxTextCtrl* nameCtrl = (wxTextCtrl*) FindWindow(ID_NAME);
    wxSpinCtrl* ageCtrl = (wxSpinCtrl*) FindWindow(ID_SAGE);
    wxChoice* sexCtrl = (wxChoice*) FindWindow(ID_SEX);
    wxCheckBox* voteCtrl = (wxCheckBox*) FindWindow(ID_VOTE);

    nameCtrl->SetValue(m_name);
    ageCtrl->SetValue(m_age);
    sexCtrl->SetSelection(m_sex);
    voteCtrl->SetValue(m_vote);

    return true;
}

/*!
 * Получаем данные из окна
 */

bool PersonalRecordDialog::TransferDataFromWindow()
{
    wxTextCtrl* nameCtrl = (wxTextCtrl*) FindWindow(ID_NAME);
    wxSpinCtrl* ageCtrl = (wxSpinCtrl*) FindWindow(ID_SAGE);
    wxChoice* sexCtrl = (wxChoice*) FindWindow(ID_SEX);
    wxCheckBox* voteCtrl = (wxCheckBox*) FindWindow(ID_VOTE);

    m_name = nameCtrl->GetValue();
    m_age = ageCtrl->GetValue();
    m_sex = sexCtrl->GetSelection();
    m_vote = voteCtrl->GetValue();

    return true;
}
```

Однако существует более простой путь пересылки данных. wxWidgets поддерживает вылидаторы, которые служат для создания связи между переменными и соответствующими элементами управления. Хотя эта технология применима не всегда, но использование валидаторов может сэкономить вам кучу времени и избавит от написания функций TransferDataToWindow и TransferDataFromWindow. Для нашего примера вы можете написать слудующий код вместо предыдущих двух функций:

```
FindWindow(ID_NAME)->SetValidator(
      wxTextValidator(wxFILTER_ALPHA, &m_name));
FindWindow(ID_AGE)->SetValidator(
      wxGenericValidator(& m_age));
FindWindow(ID_SEX)->SetValidator(
      wxGenericValidator(& m_sex);
FindWindow(ID_VOTE)->SetValidator(
      wxGenericValidator(& m_vote);
```

Эти несколько строк в конце `CreateControls` заменяют две переопределенные функции. В качестве бонуса пользователю будет запрещено использовать цифры в строке с именем.

Валидаторы могут выполнять две работы. Кроме передачи данных они также могут проверять данные и показывать сообщение об ошибке, если данные не удовлетворяют заданным критериям. В нашем примере кроме небольшой проверки имени не делается других проверок. `wxGenericValidator` относительно простой класс предназначенный только для передачи данных. Однако он может работать с множеством стандартных базовых элементов управления. Другие валидаторы (например `wxTextValidator`) имеют более сложное поведение и могут даже перехватывать нажатия на клавиши, чтобы запретить ввод недопустимых символов. Для нашего примера мы используем стандартный стиль `wxFILTER_ALPHA`, но можно также явно определить какие символы должны или не должны считаться допустимыми с помощью методов `SetIncludes` и `SetExcludes`.

Изучим подробнее каким образом wxWidgets обрабатывает валидаторы. Как было упомянуто ранее стандартный обработчик `OnOK` вызывает `TransferDataToWindow`, но до этого он вызывает `Validate`, запрещая вызов `TransferDataToWindow` и `EndModal`, если вызов закончится неудачно. Вот стандартная реализация `OnOK`:

```
void wxDialog::OnOK(wxCommandEvent& event)
{
    if ( Validate() && TransferDataFromWindow() )
    {
        if ( IsModal() )
            EndModal(wxID_OK); // Если диалог модальный
        else
        {
            SetReturnCode(wxID_OK);
            this->Show(false); // Если диалог не модальный
        }
    }
}
```

Обычная реализация `Validate` перебирает всех детей диалога (и их детей, если определен дополнительный стиль окна `wxWS_EX_VALIDATE_RECURSIVELY`) и вызывает `Validate` для каждого `wxValidator` элементов управления. Если любой из этих вызовов возвратит ошибку, то проверка диалога закончится неудачно и диалог не будет закрыт. Ожидается, что сообщение об ошибке будет вызвано снаружи функции `Validate`, если процедура проверки закончится неудачей.

Похожим образом будут автоматически вызваны `TransferDataToWindow` и `TransferDataFromWindow` для валидаторов элементов управления диалога. Валидатор обязан переслать данные, но процедура проверки является необязательной.

Валидаторы являются обработчиками событий и механизм обработки событий сначала перенаправляет приходящие события в валидатор, если он существует, перед тем как послать его элементу управления. Это позволяет валидаторам, например, перехватывать пользовательский ввод и блокировать символы, которые недопустимы для элемента управления. Такого рода блокировки обычно сопровождаются коротким гудком, который извещает пользователя, что клавиша, которую он нажал, не принята.

Так как двух представленных выше классов-валидаторов может быть не достаточно для ваших нужд (особенно если вы пишете свои собственные элементы управления), то у вас есть возможность унаследовать свой класс от `wxValidator`. Этот класс должен иметь конструктор копирования и функцию `Clone`, которая возвращает копию объекта-валидатора, а также реализацию для передачи данных и проверки. Валидатор обычно также хранит указатель на переменную языка C++. В файлах include/wx/valtext.h и src/common/valtext.cpp дистрибутива wxWidgets можно посмотреть как реализовать валидатор. Также обратитесь к разделу «Написание собственных элементов управления» в Главе 12 «Продвинутые классы окон».

9.2.5 Обработка событий

Для нашего примера нам не потребуется писать обработчики для кнопок OK и Cancel, так как мы можем использовать стандартные обработчики. Для этого достаточно использовать стандартные идентификаторы `wxID_OK` и `wxID_CANCEL` для этих кнопок.

Однако, для нетривиальных диалогов вам возможно потребуется перехватывать и обрабатывать события от элементов управления. В нашем примере у нас есть кнопка Reset, которую можно нажать в любой момент, чтобы сбросить значения в диалоге в их начальные значения. Мы добавим обработчик `OnResetClick` и соответствующую запись в таблицу обработчиков событий. Реализация `OnResetClick` будет очень простой: вначале мы сбрасываем значение переменных, вызвав функцию `Init` (которую введена, чтобы сделать единую точку инициализации переменных), а далее вызываем `TransferDataToWindow`, чтобы отобразить эти данные.

```
BEGIN_EVENT_TABLE( PersonalRecordDialog, wxDialog )
    ...
    EVT_BUTTON( ID_RESET, PersonalRecordDialog::OnResetClick)
    ...
END_EVENT_TABLE()

void PersonalRecordDialog::OnResetClick( wxCommandEvent& event )
{
    Init();
    TransferDataToWindow();
}
```

9.2.6 Обработка обновлений интерфейса

Одна из задач с которой обычно сталкивается программист — это быть уверенным, что пользователь не может выбрать элемент управления или пункт меню, который в данный момент не применим. Ленивые программисты обычно делают выскакивающую надпись «Эта опция в данный момент недоступна». Но если опция недоступна, то она должна выглядеть недоступной, и при нажатии на соответствующий элемент ничего не должно происходить. Для этого программе необходимо обновлять элементы интерфейса, чтобы они отражали правильное состояние в каждый момент времени.

В нашем примере мы должны отключить чек-бокс «Vote» когда возраст пользователя меньше 18 лет, так как в этом случае пользователь не может принимать решение об этом. Вашей первой мыслью, наверное, будет добавить обработчик событий от спина Age, который будет включать и выключать элемент Vote в соответствии со своим значением в спине. Хотя данное решение может хорошо работать для простых пользовательских интерфейсов, но представьте что будет, если состояние элемента зависит от множества факторов. Существуют более плохие ситуации, когда вы не можете перехватывать изменения некоторого параметра. Например, вам необходимо включать пункт меню «Вставить» в зависимости от доступности данных в буфере обмена. Перехват этого события очень сложен, так как данные могут поступать и из другого окна.

Чтобы решить такую проблему в wxWidgets существует специальный класс событий `wxUpdateUIEvent`, который посылается все окнам, когда программа простаивает, что происходит когда петля событий закончила обработку всех остальных событий. Вы можете добавить обработчик `EVT_UPDATE_UI` в ваш диалог, по одному для каждого элемента управления, состояние которых вы хотите контролировать. Соответствующий обработчик вычисляет текущее состояние и вызывает функции объекта-события (а не для элемента управления) для включения, выключения, установки или снятия галочки. Эта технология позволяет перенести логику обновления состояния элемента в одно место, вызывая обновления даже в то время, когда в приложение не приходят реальные события. И это хорошо, так как не может возникнуть ситуация, когда вы забудете обновить состояние элементов управления.

Далее приведен обработчик события об обновлении интерфейса для элемента «Vote». Заметим, что мы не можем использовать переменную `m_age`, так как перенос данных из элементов управления в переменные происходит только после того, как пользователь нажмет кнопку «OK».

```
BEGIN_EVENT_TABLE( PersonalRecordDialog, wxDialog )
    ...
    EVT_UPDATE_UI( ID_VOTE, PersonalRecordDialog::OnVoteUpdate )
    ...
END_EVENT_TABLE()

void PersonalRecordDialog::OnVoteUpdate( wxUpdateUIEvent& event )
{
    wxSpinCtrl* ageCtrl = (wxSpinCtrl*) FindWindow(ID_AGE);
    if (ageCtrl->GetValue() < 18)
    {
```

```
        event.Enable(false);
        event.Check(false);
    }
    else
        event.Enable(true);
}
```

Не волнуйтесь слишком сильно об эффективности такого решения, так как оно занимает не очень много циклов работы процессора. Однако, если у вас достаточно сложное приложение и вы столкнулись с проблемами производительности по вине обновления интерфейса, то посмотрите документацию по классу `wxUpdateUIEvent` о функциях `SetMode` и `SetUpdateInterval`, которые можно использовать, чтобы уменьшить время, которое wxWidgets тратит на обработку таких событий.

9.3 Добавление помощи

Существует по крайней мере три пути как можно реализовать помощь для вашего диалога:

- Всплывающие подсказки
- Контекстная помощь
- Справочная система

Кроме того у нас уже есть некоторый текст на самой панели диалога. Возможно вы захотите также использовать другие технологии, явно не поддерживаемые в wxWidgets. Например, для более сложных диалогов вы можете создать `wxHtmlWindow` вместо `wxStaticText` и загрузить в него HTML-файл, содержащий необходимое описание. В качестве альтернативы вы можете поместить маленькую кнопку помощи справа от каждого элемента управления, при нажатии на которую выводился бы некоторый текст с описанием.

Три основных типа помощи, поддерживаемые в wxWidgets, описаны далее.

9.3.1 Всплывающие подсказки

Всплывающие подсказки — это маленькие окна, содержащие короткое описание данного элемента управления, которые всплывают, когда указатель находится над элементом управления некоторое время. С помощью `SetToolTip` можно установить подсказку для элемента управления. Так как эти окна могут раздражать продвинутых пользователей, то в приложении должна быть возможность отключить их (для этого не нужно вызывать `SetToolTip` при создании или показе диалога).

9.3.2 Контекстная помощь

Контекстная помощь выглядит как небольшое выскакивающее описание, похожее на всплывающие подсказки. Пользователю необходимо сначала нажать на специальную кнопку, а после на элемент управления о котором он хочет узнать или нажать

F1, чтобы узнать информацию об элементе на котором в данный момент находится фокус (последнее работает только в системе Windows). В Windows вы можете также определить специальный стиль окна `wxDIALOG_EX_CONTEXTHELP`, чтобы создать маленькую иконку со знаком вопроса в заголовке окна. На других платформах вы можете создать `wxContextHelpButton` в диалоге (обычно ее располагают рядом с кнопками «OK» и «Cancel»). В вашем приложении вы должны написать:

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

...
    wxHelpProvider::Set(new wxSimpleHelpProvider);
...
```

Этот фрагмент кода указывает библиотеке, что необходимо предоставлять строки для поддержки контекстной помощи. Вы можете вызвать `SetHelpText`, чтобы установить текста помощи для элемента. Напишем функцию, которая настраивает контекстную помощь и всплывающие подсказки в нашем диалоге:

```
// Устанавливаем текст помощи для элементов диалога
void PersonalRecordDialog::SetDialogHelp()
{
    wxString nameHelp = wxT("Enter your full name.");
    wxString ageHelp = wxT("Specify your age.");
    wxString sexHelp = wxT("Specify your gender, male or female.");
    wxString voteHelp = wxT("Check this if you wish to vote.");

    FindWindow(ID_NAME)->SetHelpText(nameHelp);
    FindWindow(ID_NAME)->SetToolTip(nameHelp);

    FindWindow(ID_AGE)->SetHelpText(ageHelp);
    FindWindow(ID_AGE)->SetToolTip(ageHelp);

    FindWindow(ID_SEX)->SetHelpText(sexHelp);
    FindWindow(ID_SEX)->SetToolTip(sexHelp);

    FindWindow(ID_VOTE)->SetHelpText(voteHelp);
    FindWindow(ID_VOTE)->SetToolTip(voteHelp);
}
```

Если вы хотите реализовать контекстную помощь самостоятельно, позволив диалогу или `wxContextHelpButton` перехватывать соответствующие запросы, то можно просто написать в обработчике событий:

```
wxContextHelp contextHelp(window);
```

Данная команда добавляет дополнительный цикл, который обнаруживает нажатие левой клавиши мыши на элементе управления, после чего посылает сообщение

wxEVT_HELP элементу, чтобы инициировать показ контекстной помощи для данного элемента.

Однако вы не обязаны ограничивать себя только возможностями, заложенными в wxWidgets для хранения и показа текста помощи. Вы можете создать свой собственный класс-наследник от wxHelpProvider и реализовать методы GetHelp, SetHelp, AddHelp, RemoveHelp и ShowHelp.

9.4 Справочная система

Большая часть приложений поставляется с файлом помощи, в котором находятся детальные инструкции по использованию программы. В wxWidgets для этого есть элементы управления нескольких типов, которые являются наследниками от wxHelpControllerBase. Обратитесь к Главе 20 «Усовершенствование ваших приложений», за более детальной информацией о том, как реализовать такого вида помощь.

В нашем примере мы просто используем wxMessageBox, чтобы вывести некоторое сообщение, когда пользователь нажмет на кнопку «Help»[4].

```
BEGIN_EVENT_TABLE( PersonalRecordDialog, wxDialog )
    ...
    EVT_BUTTON( wxID_HELP, PersonalRecordDialog::OnHelpClick )
    ...
END_EVENT_TABLE()

void PersonalRecordDialog::OnHelpClick( wxCommandEvent& event )
{
    // В нормальном приложении мы должны вывести соответствующий раздел помощи
    /*
    wxGetApp().GetHelpController().DisplaySection(wxT("Personal record dialog"));
    */

    // В нашем примере мы просто выводим окно с сообщением
    wxString helpText =
      wxT("Please enter your full name, age and gender.\n")
      wxT("Also indicate your willingness to vote in general elections.\n\n")
      wxT("No non-alphabetical characters are allowed in the name field.\n")
      wxT("Try to be honest about your age.");

    wxMessageBox(helpText,
        wxT("Personal Record Dialog Help"),
        wxOK|wxICON_INFORMATION, this);
}
```

[4] Помощь (англ.)

9.5 Полная реализация класса

Полная реализация класса приводится в приложении J «Исходники», а также в каталоге examples/chap09 на нашем CD-ROM.

9.6 Вызов диалога

Теперь, когда диалог поностью закончен, мы можем его вызвать:

```
PersonalRecordDialog dialog(NULL, ID_PERSONAL_RECORD,
    wxT("Personal Record"));
dialog.SetName(wxEmptyString);
dialog.SetAge(30);
dialog.SetSex(0);
dialog.SetVote(true);
if (dialog.ShowModal() == wxID_OK)
{
    wxString name = dialog.GetName();
    int age = dialog.GetAge();
    bool sex = dialog.GetSex();
    bool vote = dialog.GetVote();
}
```

9.7 Адаптация диалогов для портативных устройств

wxWidgets можно использовать для мобильных и других встраиваемых устройств, используя для этого порты GTK+, X11 и Windows CE (или иные в будущем). Самым важной особенностью таких устройств является ограниченный размер экрана, который для некоторых смартфонов может быть не больше 176x220 пикселей.

Многие диалоги нуждаются в альтернативном (адаптированном для маленьких экранов) размещении элементов, а некоторые элементы управления могут быть объеденены или удалены, особенно в связи с тем, что мобильные версии более ограниченны по сравнению с их десктопными версиями. С помощью метода `wxSystemSettings::GetScreenType` можно получить полный размер экрана. Например:

```
#include "wx/settings.h"
bool isPda = (wxSystemSettings::GetScreenType() <= wxSYS_SCREEN_PDA);
```

`GetScreenType` возвращает одно из значений, указанных в таблице 9.1. Так как целочисленное значение типа растет вместе с величиной экрана, то вы можете использовать оператор целочисленного сравнения, чтобы ограничить работоспособность программы на некоторых платформах, как это сделано, например, в нашем прошлом примере.

Если вам необходимо получить точный размер экрана, то существует три способа получить его:

Таблица 9.1: Типы экранов

Тип	Описание
wxSYS_SCREEN_NONE	Неизвестный тип экрана
wxSYS_SCREEN_TINY	Миниатюрный экран, менее 320x240
wxSYS_SCREEN_PDA	Экран наладонника, 320x240 или больше, но менее 640x480
wxSYS_SCREEN_SMALL	Маленький экран, 640x480 или больше, но менее 800x600
wxSYS_SCREEN_DESKTOP	Обычный экран, 800x600 или больше

- Использовать функцию `wxSystemSettings::GetMetric`, передав ей `wxSYS_SCREEN_X` или `wxSYS_SCREEN_Y`.
- Вызвать `wxGetDisplaySize`, которая возвратит объект типа `wxSize`.
- Создать объект `wxDisplay` и вызвать метод `GetGeometry`, который возвратит `wxRect`, содержащий координаты прямоугольника для дисплея.

Теперь, когда вы умеете получать размер экрана, как можно использовать эту информацию? Вот несколько стратегий, которые можно использовать:

- Заменить весь макет диалога с помощью его загрузки из другого XRC-файла или выполнения другой процедуры создания. Если у элементов не изменится тип, то вам даже не понадобится менять обработчик событий.
- Уменьшить число элементов управления и расстоянием между ними.
- Изменить тип некоторых элементов управления, чтобы они занимали на экране меньше места (например, с `wxListBox` в `wxComboBox`). Это потребует некоторой модификации кода обработчиков событий.
- Изменить ориентацию одного или нескольких сайзеров. Некоторые портативные устройства имеют гораздо больше пространства в одном направлении, чем в другом.

Кроме того вам необходимо использовать расширения API для некоторых платформ. Смартфоны Microsoft имею две специальные кнопки, которым можно присвоить некоторые метки, такие как «OK» и «Cancel». На таких платформах вместо создания двух объектов типа `wxButton` вы должны вызвать `wxDialog::SetLeftMenu` и `wxDialog::SetRightMenu` с соответствующим идентификатором, меткой и, возможно, меню. Так как эти функции имеются только в портах для смартфонов, то необходимо добавить операторы включения в ваш код. Например:

```
#ifdef __SMARTPHONE__
    SetLeftMenu(wxID_OK, wxT("OK"));
    SetRightMenu(wxID_OK, wxT("Cancel"));
#else
    wxBoxSizer* buttonSizer = new wxBoxSizer(wxHORIZONTAL);
    GetTopSizer()->Add(buttonSizer, 0, wxALL|wxGROW, 0);
    buttonSizer->Add(new wxButton(this, wxID_OK), 0, wxALL, 5);
    buttonSizer->Add(new wxButton(this, wxID_CANCEL), 0, wxALL, 5);
#endif
```

9.8 Дополнительные заметки о дизайне диалогов

Несколько советов, которые помогут вашим диалогам выглядеть профессионально выглядящими.

9.8.1 Навигация с помощью клавиатуры

Указывайте мнемоники для меток со статическим текстом и для других элементов управления с метками. Для этого необходимо поставить перед необходимым символом амперсанд (&). На некоторых платформах (особенно Windows и GTK+) мнемоники помогают пользователю быстро передвигаться между элементами управления.

Всегда предоставляйте пользователю возможность закрыть диалог без записи, предпочтительно при помощи нажатия клавиши клавиатуры Escape. Если диалог имеет кнопку с идентификатором `wxID_CANCEL`, то ее обработчик будет автоматически вызван, если пользователь нажмет клавишу Esc. Поэтому, если у вас на диалоге есть кнопка Close[5], то разумно дать ей идентификатор `wxID_CANCEL`.

Указывайте кнопку по умолчанию (обычно «OK»), используя метод `wxButton::SetDefault`. Команда для этой кнопки вызовется, если пользователь нажмет клавишу Enter.

9.8.2 Отделение интерфейса от данных

В целях упрощения примера все используемые `PersonalRecordDialog` данные мы храним внутри диалога. Однако более правильным будет создание отдельного класса данных, который будет отделен от класса диалога. С помощью конструктора копирования и оператора присваивания вы можете передать копию данных в диалог и получить модифицированную копию данных из диалога только если пользователь подтвердил изменения. Это же правило применимо и для некоторых стандартных диалогов. В качестве упражнения вы можете переписать `PersonalRecordDialog`, чтобы он использовал класс `PersonalRecordData`. При такой реализации конструктор диалога должен получать ссылку на `PersonalRecordData`. В диалоге также желательно создать функцию `GetData`, которую может вызывать приложение, чтобы получить данные из диалога.

Обычно лучше всегда таким образом отделять пользовательский интерфейс от неинтерфейсной функциональности. Результатом будет являться код, который более компактный и легкий для понимания и отладки. Не бойтесь вводить новые классы, чтобы сделать свой код более элегантным. Использование конструктора копирования и оператора присваивания облегчит копирование и присваивание объектов, избавив от повторения больших кусков низкоуровневого кода.

Если в вашем диалоге нет кнопки «Apply»[6], которая сохраняет ваши изменения, то нажатие на «Cancel» должно оставлять данные данные приложения в том же состоянии, как они были до открытия диалога. Использование отдельного класса данных очень легко решает эту задачу, так как диалог редактирует не «настоящие» данные, а только их копию.

[5] Закрыть (англ.)
[6] Применить (англ.)

9.8.3 Макет

Если диалог выглядит излишне компактным или сжатым, то можно добавить в него немного пространства. Попробуйте увеличить пространство около границы диалога, используя дополнительные сайзеры (как это сделано в `PersonalRecordDialog`) и добавив пространство между группами элементов. Рекомендуется использовать `wxStaticBoxSizer` и `wxStaticLine`, чтобы логически сгруппировать или разделить элементы управления. Сайзеры `wxGridSizer` и `wxFlexGridSizer` позволяют выравнять элементы и их метки так, чтобы у них не появлялся случайный отступ. Для выравнивания групп элементов очень часто используют расширяющиеся спейсеры. Например, кнопки «OK», «Cancel» и «Help» обычно помещают в отдельную группу, которую выравнивают по правой границе диалога. Этого можно достичь поместив спейсер и кнопки в горизонтальный `wxBoxSizer` и установив у спейсера расширение по горизонтали (задав положительный коэффициент расширения).

Если это возможно и допустимо — делайте ваши диалоги с изменяемым размером. Традиционно диалоги в Windows очень редко позволяют изменять свой размер пользователю, однако не существует причины почему это надо запрещать, так как маленькие элементы управления на большом экране могут быть очень неудобными. wxWidgets упрощает создание таких диалогов с помощью сайзеров, поэтому необходимо использовать их везде, что позволит диалогу учитывать текущий шрифт и размер элементов, а также упростит поддержку различных языке. Внимательно выбирайте какие из элементов управления могут расти; например, многострочный элемент редактирования является хорошим кандидатом и при увеличении размера даст пользователю дополнительное место для редактирования. Расширяющиеся спейсеры можно использовать, чтобы реализовать выравнивание в изменяющемся диалоге. Заметим, что не нужно самостоятельно изменять размер элементов управления увеличивая или уменьшая текст в нем, просто давайте больше или меньше пространства элементу управления. Обратитесь к Главе 7 за дополнительной информацией.

Если ваш диалог получается слишком большим — разбейте его на панели и используйте `wxNotebook`, `wxListbook` или `wxChoicebook`, чтобы можно было работать только с одной из этих панелей. Использование множества независимых диалогов раздражает пользователя и перегружает меню, поэтому использование нескольких панелей более предпочтительно. Вы должны избегать прокручивающихся панелей, пока не появится реальная причина в их использовании. Возможность прокручивания элементов управления поддерживается не на всех платформах, поэтому может оказаться, что диалог выглядит не так как планировалось. Если у вас есть множество свойств для редактирования, то возможно стоит использовать редактор свойств на основе `wxGrid` или сторонних классов (например, класс `wxPropertyGrid`, который указан в Приложении E «Сторонние инструменты для wxWidgets»).

9.8.4 Эстетика

Используйте очень умеренное число меток, записанные заглавными буквами. Не увлекайтесь использованием нестандартных шрифтов и цветов в ваших диалогах, иначе диалог будет выглядеть чужеродным вне текущей темы оформления и сильно отличаться от других диалогов приложения. Для лучшей переносимости между платформами не изменяйте шрифты и цвета диалогов — пусть оформлением занимается

wxWidgets. Если вам необходимо настроить отображение некоторой части диалога, то это лучше сделать через использование `wxStaticBitmap`.

9.8.5 Альтернатива диалогам

Наконец, можно создать независимый диалог, который будет немодальным, такой как вкладка в главном окне приложения. Основная часть принципов построения и реализации диалогов применима к немодальным диалогам и панелям, но появляются дополнительные особенности с размерами (окно имеет меньше контроля над своими размерами) и синхронизацией (окно больше эксклюзивно не владеет данными, которые отображает).

9.9 Использование ресурсных файлов wxWidgets

Вы можете загружать определения диалогов, фреймов, меню, панелей инструментов и так далее из специальных XML-файлов с расширением xrc, вместо того, чтобы создавать все эти элементы непосредственно с помощью C++. Это позволяет гораздо элегантнее разделить код и пользовательский интерфейс, а также дает возможность поменять дизайн диалогов приложения во время выполнения программы. Файлы XRC могут быть экспортированы во множество инструментов по редактированию пользовательского интерфейса, включая такие как wxDesigner, DialogBlocks, XRCed и wxGlade.

9.9.1 Загрузка ресурсов

Чтобы использовать XRC-файлы в вашем приложении вам необходимо включить файл wx/xrc/xmlres.h в код вашего приложения.

Если вы преобразуете свои XRC-файлы в бинарные XRS-файлы (как это сделать будет показано далее), то вам также необходимо включить поддержку файловой системы zip, поместив соответствующий вызов `AddHandler` в вашу функцию `OnInit`:

```
#include "wx/filesys.h"
#include "wx/fs_zip.h"

wxFileSystem::AddHandler(new wxZipFSHandler);
```

Далее инициализируем поддержку XRC, добавив следующую строку в `OnInit`:

```
wxXmlResource::Get()->InitAllHandlers();
```

XRC-файлы загружаются с помощью следующего кода:

```
wxXmlResource::Get()->Load(wxT("resources.xrc"));
```

Данная команда позволит wxWidgets брать ресурсы из этого файла. Для того, чтобы создать реальные элементы управления вам необходимо вызвать другие функции. Например, следующий фрагмент когда создает диалог, чье имя в ресурсах dialog1:

```
MyDialog dlg;
wxXmlResource::Get()->LoadDialog(&dlg, parent, wxT("dialog1"));
dlg.ShowModal();
```

Следующий код показывает как загрузить панель меню, пункты меню, панель инструментов, изображения, иконки и панели из ресурсов:

```
MyFrame::MyFrame(const wxString& title): wxFrame(NULL, -1, title)
{
    SetMenuBar(wxXmlResource::Get()->LoadMenuBar(wxT("mainmenu")));
    SetToolBar(wxXmlResource::Get()->LoadToolBar(this,
                                                 wxT("toolbar")));

    wxMenu* menu = wxXmlResource::Get()->LoadMenu(wxT("popupmenu"));

    wxIcon icon = wxXmlResource::Get()->LoadIcon(wxT("appicon"));
    SetIcon(icon);

    wxBitmap bitmap = wxXmlResource::Get()->LoadBitmap(wxT("bmp1"));

    // Заканчиваем создание panelA и создаем ее экземпляр
    MyPanel* panelA = new MyPanel;
    panelA = wxXmlResource::Get()->LoadPanel(panelA, this,
                                                   wxT("panelA"));

    // Второй метод: сразу из XRC создаем и загружаем panelB
    wxPanel* panelB = wxXmlResource::Get()->LoadPanel(this,
                                                   wxT("panelB"));
}
```

wxWidgets содержит один глобальный объект `wxXmlResource`, который вы можете использовать, но также можете самостоятельно создать объект `wxXmlResource`, загрузить ресурсы, а после уничтожить его. Вы также можете вызвать `wxXmlResource::Set`, чтобы установить новый глобальный объект, уничтожив старый.

Чтобы определить таблицу сообщений для окна, загруженного из ресурсного файла, вы не можете использовать целочисленные идентификаторы, так как ресурсы имеют строковое представление. Вместо этого вы должны использовать макрос `XRCID`, который принимает в качестве параметра имя ресурса и возвращает целочисленный идентификатор, ассоциированный с этим именем. `XRCID` является просто синонимом функции `wxXmlResource::GetXRCID`. Вот пример использования `XRCID`:

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(XRCID("menu_quit"),  MyFrame::OnQuit)
    EVT_MENU(XRCID("menu_about"), MyFrame::OnAbout)
END_EVENT_TABLE()
```

9.9.2 Использование бинарных и встраиваемых ресурсных файлов

Очень часто удобнее скомбинировать несколько ресурсных файлов в один бинарный файл (с расширением xrs). Чтобы скомпилировать XRC-файлы в один zip-файл из которого можно загружать ресурсы используйте утилиту wxrc, расположенную в каталоге utils/wxrc вашего дистрибутива wxWidgets:

```
wxrc resource1.xrc resource2.xrc -o resource.xrs
```

Метод `wxXmlResource::Load` позволяет загрузить бинарный файл, абсолютно также как и в случае с обычными ресурсами.

Совет: вместо создания отдельного zip-файла с вашими ресурсными файлами, вы можете включить их в один zip-файл, который включает другие файлы, необходимые вашему приложению, такие как HTML-файлы, изображения и так далее. `wxXmlResource::Load` принимает спецификатор виртуальной файловой системы, как будет описано в Главе 14 «Файлы и потоки», поэтому вы можете написать:

```
wxXmlResource::Get()->Load(wxT("resources.bin#zip:dialogs.xrc"));
```

Также можно скомпилировать ваши XRC-файлы в C++ код, который можно встроить в ваше приложение, чтобы исключить возможность переноса ресурсных файлов в неправильное место. Для этого надо сделать следующее:

```
wxrc resource1.xrc resource2.xrc c -o resource.cpp
```

Далее скомпилируйте этот файл как обычный C++ файл и прилинкуйте к вашему приложению. Файл включает функцию `InitXmlResource`, которую вам необходимо вызвать. Например,

```
extern void InitXmlResource(); // определена в сгенерированом файле

wxXmlResource::Get()->InitAllHandlers();
InitXmlResource();
```

Таблица 9.2 содержит аргументы и опции командной строки для программы wxrc.

Таблица 9.2: Команды wxrc

Короткое имя	Длинное имя	Описание
`-h`	`help`	Показывает помощь.
`-v`	`verbose`	Включает подробную информацию о работе.
`-c`	`cpp-code`	Генерирует исходники C++, вместо XRS-файла.
`-p`	`python-code`	Генерирует исходники Python, вместо XRS-файла.

Таблица 9.2: Команды wxrc (продолжение)

Короткое имя	Длинное имя	Описание
`-e`	`extra-cpp-code`	Если используется совместно с командой `-c`, генерирует заголовочный файл C++, содержащий определение для окон, содержащихся в XRC-файле.
`-u`	`uncompressed`	Не сжимать XML файлы (только для C++).
`-g`	`gettext`	Выводит все строки с подчеркиваниями, чтобы poEdit или gettext смогли их просканировать. Вывод производится в `stdout` или файл, если задан ключ `-o`.
`-n`	`function <имя>`	Определяет функцию инициализации для C++ (используется совместно с `-c`)
`-o <имя_файла>`	`output <имя_файла>`	Определяет файл вывода, такой как resource.xrs или resource.cpp.
`-l <имя_файла>`	`list-of-handlers <имя_файла>`	Выводит список обработчиков ресурсов, необходимых для заданных файлов.

9.9.3 Перевод ресурсов

Если объект `wxXmlResource` создан с флагом `wxXRC_USE_LOCALE` (это поведение по умолчанию), то все отображаемые строки являются объектами перевода, как это описано в Главе 16 «Написание многоязыковых приложений». Однако poEdit не может просканировать XRC-файлы, чтобы выбрать строки для перевода, как это можно сделать с программой на C++ , поэтому вам необходимо создать файл, содержащий такие строки, используя опцию -g. Например:

```
wxrc -g resources.xrc -o resource_strings.cpp
```

Теперь вы можете запустить poEdit, чтобы найти все строки в этом и других файлах.

9.9.4 Формат XRC-файлов

Так как формат книги не позволяет подробно рассмотреть формат XRC-файлов, то дальше просто приводится пример маленького диалога с сайзерами:

```
<?xml version="1.0"?>
<resource version="2.3.0.1">
```

```
<object class="wxDialog" name="simpledlg">
    <title>A simple dialog</title>
    <object class="wxBoxSizer">
      <orient>wxVERTICAL</orient>
      <object class="sizeritem">
        <object class="wxTextCtrl">
          <size>200,200d</size>
          <style>wxTE_MULTILINE|wxSUNKEN_BORDER</style>
          <value>Hello, this is an ordinary multiline\n textctrl....</value>
        </object>
        <option>1</option>
        <flag>wxEXPAND|wxALL</flag>
        <border>10</border>
      </object>
      <object class="sizeritem">
        <object class="wxBoxSizer">
          <object class="sizeritem">
            <object class="wxButton" name="wxID_OK">
              <label>Ok</label>
              <default>1</default>
            </object>
          </object>
          <object class="sizeritem">
            <object class="wxButton" name="wxID_CANCEL">
              <label>Cancel</label>
            </object>
            <border>10</border>
            <flag>wxLEFT</flag>
          </object>
        </object>
        <flag>wxLEFT|wxRIGHT|wxBOTTOM|wxALIGN_RIGHT</flag>
        <border>10</border>
      </object>
    </object>
  </object>
</resource>
```

Детальную спецификация формата XRC можно найти в технических замечаниях в файле docs/tech/tn0014.txt вашего дистрибутива wxWidgets. Если вы используете какой-нибудь редактор для пользовательского интерфейса, то вам не нужно знать о формате XRC.

Вам может быть интересно как можно использовать текстовые XRC-файлы, чтобы поместить в них изображения и иконки. Эти ресурсы необходимо определить как URL, а механизм виртуальных файловых систем извлечет их из необходимого источника, такого как zip-файл. Например:

```
<object class="wxBitmapButton" name="wxID_OK">
```

```
  <bitmap>resources.bin#zip:okimage.png</bitmap>
</object>
```

Обратитесь к Главе 10 «Работа с изображениями» и Главе 14 «Файлы и потоки» за детальной информацией об использовании виртуальных файловых систем для загрузки ресурсов, таких как изображения.

9.9.5 Определение обработчика ресурсов

Система XRC использует обработчики ресурсов, чтобы опознать определения XML для каждого типа ресурсов. Если вы напишите ваш собственный элемент управления, то вы возможно захотите написать такой обработчик, чтобы иметь возможность загружать ваш элемент управления из XRC.

Для иллюстрации посмотрим как реализован обработчик ресурсов для стандартного класса wxButton:

```
#include "wx/xrc/xmlres.h"

class wxButtonXmlHandler : public wxXmlResourceHandler
{
DECLARE_DYNAMIC_CLASS(wxButtonXmlHandler)
public:
    wxButtonXmlHandler();
    virtual wxObject *DoCreateResource();
    virtual bool CanHandle(wxXmlNode *node);
};
```

Реализация обработчика достаточно простая. В конструкторе обработчика вы, используя макрос `XRC_ADD_STYLE`, добавляете стили, которые должна поддерживать кнопка, а в конце вызываете `AddWindowStyles`, чтобы добавить стандартные стили для окон. В методе `DoCreateResource` объект-кнопка создается в два шага, используя `XRC_MAKE_INSTANCE` и `Create`, извлекая параметры, такие как метка, положение и размер. И, наконец, `CanHandle` проверяет может ли обработчик обработать узел, который ему передали. Последняя возможность позволяет в одном обработчике реализовать обработку нескольких типов ресурсов.

```
IMPLEMENT_DYNAMIC_CLASS(wxButtonXmlHandler, wxXmlResourceHandler)

wxButtonXmlHandler::wxButtonXmlHandler()
: wxXmlResourceHandler()
{
    XRC_ADD_STYLE(wxBU_LEFT);
    XRC_ADD_STYLE(wxBU_RIGHT);
    XRC_ADD_STYLE(wxBU_TOP);
    XRC_ADD_STYLE(wxBU_BOTTOM);
    XRC_ADD_STYLE(wxBU_EXACTFIT);
    AddWindowStyles();
}
```

```
wxObject *wxButtonXmlHandler::DoCreateResource()
{
   XRC_MAKE_INSTANCE(button, wxButton)

   button->Create(m_parentAsWindow,
                  GetID(),
                  GetText(wxT("label")),
                  GetPosition(), GetSize(),
                  GetStyle(),
                  wxDefaultValidator,
                  GetName());

    if (GetBool(wxT("default"), 0))
        button->SetDefault();
    SetupWindow(button);

    return button;
}

bool wxButtonXmlHandler::CanHandle(wxXmlNode *node)
{
    return IsOfClass(node, wxT("wxButton"));
}
```

Чтобы использовать обработчик приложение должно включить заголовочный файл и зарегистрировать обработчик, как показано ниже:

```
#include "wx/xrc/xh_bttn.h"
wxXmlResource::AddHandler(new wxBitmapXmlHandler);
```

9.9.6 Сторонние элементы управления

XRC-файл может определить сторонние, или «неизвестные» элементы управления, указав `class="unknown"` в определении объекта. Вы можете использовать такую возможность для элементов управления, которые создаются с помощью кода C++, после того как их родительское окно будет загружено из XRC. После того, как XRC загрузит сторонний объект создается окно, которое помещается на место данного объекта. Далее приложение вызывает `AttachUnknownControl` для помещения реального окна в окно, созданное на прошлом шаге с правильными размерами и расположением. Например,

```
wxDialog dlg;

// Загружаем диалог
wxXmlResource::Get()->LoadDialog(&dlg, this, wxT("mydialog"));

// Создаем экземпляр нашего элемента управления
```

```
MyCtrl* myCtrl = new MyCtrl(&dlg, wxID_ANY);

// Добавляем его к диалогу
wxXmlResource::Get()->AttachUnknownControl(wxT("custctrl"), myCtrl);

// Показываем диалог
dlg.ShowModal();
```

Определение стороннего элемента управления может выглядеть следующим образом:

```
<object class="unknown" name="custctrl">
  <size>100,100</size>
</object>
```

Используя данную технологию вы можете поместить элемент в утилиту редактирования интерфейса, которая ничего не знает про ваш элемент управления, а также избежите необходимости писать свой обработчик ресурсов.

9.10 Итоги

В этой главе мы изучили основные принципы проектирования и реализации диалогов, включая краткое описание сайзеров, валидаторов и преимущества использования событий об обновлении интерфейса. Примеры создания диалогов смотрите в каталоге samples/dialogs вашего дистрибутива wxWidgets. Также в каталоге samples/validate есть пример использования валидаторов. В следующей главе будет рассказано о работе с изображениями.