Глава 11

Буфер обмена и поддержка перетаскивания (Drag and Drop)

Большинство приложений предоставляют возможность перемещения данных в или из буфера обмена, используя операции копирования и вставки данных. Это основной путь реализации взаимодействия вашей программы с другими приложениями. Более продвинутые программы позволяют пользователю перетаскивать объекты между окнами, как внутри одной программы, так и за ее пределы. К примеру, перетаскивание файла из окна менеджера файлов в рабочее окно программы, как правило, приводит к его открытию внутри программы, добавлению в список файлов или другому, предусмотренному автором, поведению. Это может быть наиболее эффективным способом открытия файла приложением, гораздо более быстрым, чем выбор соответствующей команды в меню. Скорее всего ваши пользователи по достоинству оценят подобную возможность.

Учитывая то, что работа с буфером обмена и реализация перетаскивания связаны с передачей данных, для них используется ряд общих классов. Поэтому в данной главе мы будем рассматривать эти вопросы параллельно.

Вы узнаете как использовать стандартные классы объектов данных, предоставляемые wxWidgets, и как реализовать свои собственные.

11.1 Классы объектов данных

`wxDataObject` является центральным классом при работе с буфером обмена и перетаскиванием. Данные, которые вы копируете или вставляете из буфера обмена или перетаскиваете мышкой, представляют собой экземпляры классов-наследников от `wxDataObject`. `wxDataObject` — это «умная» обертка над данными, которая знает, какие форматы объект поддерживает (`GetFormatCount` и `GetAllFormats`) и как правильно «отдать» данные в любом поддерживаемом формате (`GetDataHere`). Он также может принимать данные от других приложений в любом поддерживаемом формате, если вы реализуете метод `SetData`. Далее мы проиллюстрируем использование данного объекта на примерах.

Стандартные форматы данных, например `wxDF_TEXT`, идентифицируются целым числом, нестандартные (собственные) — строкой. `wxDataFormat` работает с любыми форматами, благодаря наличию специальных конструкторов для стандартных и собственных форматов. Список стандартных форматов данных представлен в таблице 11.1.

Таблица 11.1: Стандартные форматы данных

Идентификатор	Описание
wxDF_INVALID	Недействительный формат, используемый как аргумент по умолчанию для функций, получающих аргумент типа `wxDataFormat`.
wxDF_TEXT	Текстовый формат. Стандартный объект данных `wxTextDataObject`.
wxDF_BITMAP	Растровое изображение. Стандартный объект данных `wxBitmapDataObject`.
wxDF_METAFILE	Метафайл (только для Windows). Стандартный объект данных `wxMetafileDataObject`.
wxDF_FILENAME	Список имен файлов. Стандартный объект данных `wxFileDataObject`.

Вы также можете создать собственный формат данных, передав произвольную строку в конструктор `wxDataFormat`. Формат будет зарегистрирован в системе при первом использовании.

Буфер обмена и операции перетаскивания имеют дело как с источником (источник данных), так и с целью (приемник данных). Оба они могут находиться в одном приложении или даже в одном окне, например, когда вы перетаскиваете текст в пределах одного поля редактирования. Давайте посмотрим, за что отвечает каждый из этих компонентов.

11.1.1 Задачи источника данных (Data Source)

Источник данных отвечает за создание экземпляра класса `wxDataObject`, содержащего передаваемые данные. После чего он должен (с помощью `SetData`) поместить созданный объект в буфер обмена или передать объект в `wxDropSource`, когда пользователь начинает операцию перетаскивания, а после чего вызвать `DoDragDrop`.

Основное отличие при работе с буфером обмена — то, что объект, передаваемый в буфер, должен быть создан именно в куче (а не на стеке!) при помощи использования оператора `new`. Выделенная память будет освобождена автоматически, когда надобность в ней отпадет. Вы же действительно не можете знать, в какой момент данные будут вставлены из буфера обмена в другое приложение. С другой стороны, объект, необходимый для операции перетаскивания, должен существовать только во время выполнения `DoDragDrop`, а после этого может быть безопасно удален. Таким образом, он может быть создан либо в куче, либо на стеке (просто как локальная переменная).

Еще одно небольшое отличие связано с тем, что в случае операций с буфером обмена, приложение обычно заранее знает, копирует ли пользователь данные или выре-

зает. Если данные вырезаются в буфер, то они копируются и удаляются из редактируемого объекта (обычно это зависит от того, какой пункт меню выбрал пользователь). Для операций перетаскивания приложение получает эту информацию только после окончания перетаскивания, т.е. после возврата управления из `DoDragDrop`.

11.1.2 Задачи приемника данных (Data Target)

Для того, чтобы получить данные из буфера обмена (операция вставки), вы должны создать класс-наследник от `wxDataObject`, который поддерживает нужные вам форматы данных, и передать его в `wxClipboard::GetData`. Если функция вернула `false`, значит ни один из поддерживаемых форматов недоступен. Если вы получили `true`, то данные были успешно помещены в `wxDataObject`.

В случае операции перетаскивания, виртуальная функция `wxDropTarget::OnData` вызывается, когда пользователь «бросает» данные в окно приложения. Вы можете запросить нужные данные, используя метод `wxDropTarget::GetData`.

11.2 Работа с буфером обмена

Для работы с буфером обмена необходимо вызывать методы глобального указателя `wxTheClipboard`.

Перед тем, как скопировать или вставить данные, вы должны получить временное владение над буфером, вызвав `wxClipboard::Open`. Если эта функция вернула `true`, то вы стали владелецем буфера обмена. `wxClipboard::SetData` служит для помещения данных в буфер, а `wxClipboard::GetData` — для их получения. В конце работы с буфером необходимо вызвать `wxClipboard::Close`, чтобы вернуть буфер операционной системе. Помните, вы должны держать буфер обмена открытым, только пока он вам действительно нужен, и закрывать его сразу же после использования.

Существует небольшой класс-помощник `wxClipboardLocker`, который захватывает буфер обмена в своем конструкторе (если получится) и закрывает в деструкторе. Таким образом, вы можете просто писать:

```
wxClipboardLocker locker;
if (!locker)
{
   ... Сообщите об ошибке и выйдите  ...
}
... Используйте буфер обмена ...
```

Следующий код показывает, как поместить текст в буфер обмена и получить его обратно:

```
// Пишем текст в буфер обмена
if (wxTheClipboard->Open())
{
    // Буфер обмена получает переданный wxTextDataObject во владение.
    // Таким образом, мы только выделяем память, но не освобождаем ее!
    wxTheClipboard->SetData(new wxTextDataObject(wxT("Нужный Текст")));
```

```
    wxTheClipboard->Close();
}

// Читаем текст
if (wxTheClipboard->Open())
{
    if (wxTheClipboard->IsSupported(wxDF_TEXT))
    {
        wxTextDataObject data;
        wxTheClipboard->GetData(data);
        wxMessageBox(data.GetText());
    }
    wxTheClipboard->Close();
}
```

А теперь то же самое, но для рисунков:

```
// Запишем изображение в буфер обмена
wxImage image(wxT("splash.png"), wxBITMAP_TYPE_PNG);
wxBitmap bitmap(image.ConvertToBitmap());
if (wxTheClipboard->Open())
{
    // Буфер обмена получает переданный объект данных во владение.
    // Таким образом, мы только выделяем память, но не освобождаем ее!
    wxTheClipboard->SetData(new wxBitmapDataObject(bitmap));
    wxTheClipboard->Close();
}

// Читаем изображение
if (wxTheClipboard->Open())
{
    if (wxTheClipboard->IsSupported(wxDF_BITMAP))
    {
        wxBitmapDataObject data;
        wxTheClipboard->GetData( data );
        bitmap = data.GetBitmap();
    }
    wxTheClipboard->Close();
}
```

Если вы решили реализовать поддержку буфера обмена, то необходимо обновлять элементы пользовательского интерфейса, связанные с буфером обмена (например, пункты меню, кнопки в панели инструментов или просто отдельные кнопки) так, чтобы определенные команды были доступными или недоступными в зависимости от текущей ситуации. Для этого стоит использовать механизм обновления пользовательского интерфейса, предоставляемый wxWidgets, который генерирует событие `wxUpdateUIEvent` во время простоя приложения (idle time). За более подробной информацией об этом обратитесь к Главе 9 «Написание собственных диалогов». Этот

механизм позволяет вашему приложению правильно обновлять интерфейс, даже если данные были скопированы в буфер обмена другим приложением без вашего ведома.

Некоторые элементы управления, подобные `wxTextCtrl`, самостоятельно обрабатывают события обновления пользовательского интерфейса (`wxUpdateUIEvent`). Если вы будете использовать стандартные идентификаторы `wxID_CUT`, `wxID_COPY` и `wxID_PASTE` для пунктов меню/кнопок в панели инструментов и сделаете так, чтобы сначала обрабатывались командные события от активного элемента управления, то интерфейс будет обновляться так, как этого ожидает пользователь. Глава 20 «Совершенствование вашего приложения» научит вас перенаправлять командные события в активный элемент управления с помощью замещения метода `wxFrame::ProcessEvent`.

11.3 Реализация перетаскивания (Drag and Drop)

Вы можете реализовать источник или приемник данных, а также их обоих в своем приложении.

11.3.1 Реализация источника данных

При реализации источника перетаскивания используйте экземпляр класса `wxDropSource` для предоставления данных, которые пользователь может перетаскивает в приемник. Далее мы рассмотрим что происходит после того, как ваше приложение обнаружило начало процесса перетаскивания. В нашем случае мы опускаем шаги, необходимые для отслеживания поведения курсора мыши и определения начала перетаскивания, оставляя это полностью на совести вашего приложения. Некоторые элементы управления помогают вам узнать о том, что пользователь инициировал перетаскивание, генерируя соответствующее событие. В таком случае вам нет необходимости реализовывать отслеживание самостоятельно (что теоретически может создать помехи правильному родному поведению мыши в данном элементе управления). Данная глава расскажет о том, что надо делать уже после того, как wxWidgets уведомит вас о начале перетаскивания.

С точки зрения источника данных, процесс перетаскивания состоит из перечисленных далее этапов.

Подготовка

Прежде всего необходимо создать объект данных и инициализировать его данными, которые вы собираетесь отдать. Например,

```
wxTextDataObject myData(wxT("Этот текст будет перемещен."));
```

Начало перетаскивания

Для того, чтобы инициировать процесс перетаскивания (обычно в ответ на захват объекта мышью) необходимо создать объект класса `wxDropSource` и вызвать метод `wxDropSource::DoDragDrop`, например, так:

```
wxDropSource dragSource(this);
dragSource.SetData(myData);
wxDragResult result = dragSource.DoDragDrop(wxDrag_AllowMove);
```

Список значений, которые понимает `DoDragDrop` можно найти в таблице 11.2.

Таблица 11.2: Возможные значения для DoDragDrop

Режим	Описание
wxDrag_CopyOnly	Разрешено только копирование данных.
wxDrag_AllowMove	Разрешено перемещение.
wxDrag_DefaultMove	Операция по умолчанию — перемещение данных.

При создании объекта данных `wxDropSource` вы можете задать окно-источник данных и три курсора, которые будут использоваться для отображения в соответствующих ситуациях — при копировании, перемещении и невозможности бросить данные. На самом деле для GTK+ используются иконки, в то время как для всех остальных платформ — курсоры. Макрос `wxDROP_ICON` может быть использован для того, чтобы скрыть это различие. Вскоре вы увидете как это можно сделать.

Перетаскивание

Вызов `DoDragDrop` блокирует программу до тех пор, пока пользователь не отпустит кнопку мыши (если вы не замещаете функцию `GiveFeedback` для выполнения специфических действий). Когда указатель мыши перемещается в пределы окна программы, понимающей используемый вами формат, вызывается соответствующий метод объекта `wxDropTarget` (смотри «Реализация приемника данных»).

Обработка результата

`DoDragDrop` возвращает значение типа `wxDragResult`, как результат завершения операции. Список возможных кодов приведен в таблице 11.3.

Таблица 11.3: Значения wxDragResult, возвращаемые функцией DoDragDrop

Результат	Описание
wxDragError	Произошла ошибка во время операции перетаскивания.
wxDragNone	Приемник данных не принял передаваемые данные.
wxDragCopy	Данные были успешно скопированы.
wxDragMove	Данные были успешно перемещены (только для Windows).
wxDragLink	Операция закончилась созданием связи.
wxDragCancel	Пользователь отменил операцию перетаскивания.

Реакция на конкретное значение `DoDragDrop` зависит от вашего приложения. Обычно приложение удаляет перемещенные данные и обновляет экран. Код `wxDragNone` означает, что пользователь отменил операцию. Например:

```
switch (result)
```

```
{
    case wxDragCopy: /* Данные были скопированы или связаны. Ничего не делаем */
    case wxDragLink:
        break;

    case wxDragMove: /* Данные были перемещены - удаляем их */
        DeleteMyDraggedData();
        break;

    default: break;  /* Или перетаскивание отменено, или данные не приняты,
                        или возникла ошибка - в любом случае ничего не делаем. */
}
```

Следующий пример показывает, как реализовать источник данных, работающий с текстом. `DnDWindow` содержит член класса `strText`. Текст, который содержит данная переменная, используется в качестве данных для перетаскивания. Операция перетаскивания начинается в момент нажатия левой клавиши мыши. Результат сообщается пользователю в окне сообщений. В реальном приложении операция перетаскивания не должна начинаться сразу после нажатия на левую клавишу мыши — необходимо убедится, что курсор мыши сместился на минимальное расстояние для того, чтобы можно было отличить старт перетаскивания от обычного щелчка.

```
void DnDWindow::OnLeftDown(wxMouseEvent& event )
{
    if ( !m_strText.IsEmpty() )
    {
        // Начало операции перетаскивания
        wxTextDataObject textData(m_strText);
        wxDropSource source(textData, this,
                            wxDROP_ICON(dnd_copy),
                            wxDROP_ICON(dnd_move),
                            wxDROP_ICON(dnd_none));

        int flags = 0;
        if ( m_moveByDefault )
            flags |= wxDrag_DefaultMove;
        else if ( m_moveAllow )
            flags |= wxDrag_AllowMove;

        wxDragResult result = source.DoDragDrop(flags);

        const wxChar *pc;
        switch ( result )
        {
            case wxDragError:   pc = wxT("Ошибка!");    break;
            case wxDragNone:    pc = wxT("Ничего не произошло");   break;
            case wxDragCopy:    pc = wxT("Данные скопированы");    break;
```

```
            case wxDragMove:    pc = wxT("Данные перемещены");     break;
            case wxDragCancel:  pc = wxT("Операция Отменена"); break;
            default:            pc = wxT("Где я?");      break;
        }

        wxMessageBox(wxString(wxT("Результат операции: ")) + pc);
    }
}
```

11.3.2 Реализация приемника данных

Для реализации приемника данных, или, другими словами, для возможности в своем приложении принимать перетаскиваемые данные, вам необходимо связать объект `wxDropTarget` с окном, используя метод `wxWindow::SetDropTarget`. Вы должны унаследовать свой класс от `wxDropTarget` и реализовать все чисто виртуальные методы базового класса. В частности необходимо заменить `OnDragOver`, чтобы возвращать значения `wxDragResult`, указывающие как должен изменятся курсор при прохождении вашего окна, а также заменить `OnData`, чтобы правильно отреагировать на бросание данных. Как вариант, возможно унаследовать свой класс от `wxTextDropTarget` или `wxFileDropTarget` и заменить только `OnDropText` или `OnDropFiles` соответственно.

С точки зрения приемника данных, процесс перетаскивания состоит из перечисленных далее этапов.

Инициализация

`wxWindow::SetDropTarget` вызывается при создании окна, чтобы связать данное окно с объектом-приемником данных. Также при создании (или во время последующего выполнения программы) объект данных связывается с приемником данных при помощи `wxDropTarget::SetDataObject`. Этот объект отвечает за согласование форматов между источником и приемником данных.

Перетаскивание

Если в процессе перетаскивания курсор мыши перемещается над окном приемника, то вызываются соответствующие функции приемника `wxDropTarget::OnEnter`, `wxDropTarget::OnDragOver` и `wxDropTarget::OnLeave`. Каждая из них должна вернуть правильный `wxDragResult`. Таким образом код реализации перетаскивания может изменить курсор, чтобы визуально оповестить пользователя о возможности взаимодействия.

Бросание данных

Когда пользователь отпускает кнопку мыши над окном, wxWidgets опрашивает объект-наследник от `wxDropTarget`, сможет ли он принять данные, вызывая метод `wxDataObject::GetAllFormats`. Если данные могут быть приняты, то они помещаются в `wxDataObject`, связанный с приемником данных, после чего вызывается `wxDropTarget::OnData`. Метод `wxDropTarget::OnData` должен вернуть

wxDragResult, который, в свою очередь, будет возвращен источнику данных из wxDropSource::DoDragDrop.

11.3.3 Использование стандартных приемников данных

В wxWidgets реализовано несколько стандартных классов, унаследованных от wxDropTarget. Таким образом во многих часто встречающихся случаях вам не придется почти ничего программировать самостоятельно. Вы просто наследуете свой класс от подходящего стандартного класса, а после замещаете виртуальную функцию, чтобы получить и обработать событие о получении данных.

wxTextdropTarget самостоятельно получает текст, вам просто необходимо заменить функцию OnDropText, чтобы сделать что-либо с брошенным текстом. Для примера рассмотрим реализацию приемника данных, который добавляет брошенный текст в список.

```
// Приемник данных, который добавляет брошенный текст в список
class DnDText : public wxTextDropTarget
{
public:
    DnDText(wxListBox *owner) { m_owner = owner; }

    virtual bool OnDropText(wxCoord x, wxCoord y,
                            const wxString& text)
    {
        m_owner->Append(text);
        return true;
    }

private:
    wxListBox *m_owner;
};

// Связываем приемник данных с окном
wxListBox* listBox = new wxListBox(parent, wxID_ANY);
listBox->SetDropTarget(new DnDText(listBox));
```

Следующий пример показывает использование wxFileDropTarget, который может принимать файлы, перетаскиваемые из файлового менеджера (например, Explorer для Windows). Данная реализация сообщает имена брошенных файлов и их количество.

```
// Приемник данных, который добавляет имена файлов в список
class DnDFile : public wxFileDropTarget
{
public:
    DnDFile(wxListBox *owner) { m_owner = owner; }

    virtual bool OnDropFiles(wxCoord x, wxCoord y,
```

```
                                const wxArrayString& filenames)
    {
        size_t nFiles = filenames.GetCount();
        wxString str;
        str.Printf( wxT("Принято %d файлов"), (int) nFiles);
        m_owner->Append(str);
        for ( size_t n = 0; n < nFiles; n++ ) {
            m_owner->Append(filenames[n]);
        }

        return true;
    }

private:
    wxListBox *m_owner;
};

// Связываем приемник данных с окном
wxListBox* listBox = new wxListBox(parent, wxID_ANY);
listBox->SetDropTarget(new DnDFile(listBox));
```

11.3.4 Создание собственного приемника данных

Теперь мы создадим свой собственный приемник данных, который будет принимать веб-адреса (URL). В этот раз нам необходимо заместить `OnData` и `OnDragOver`. Также мы введем новую виртуальную функцию `OnDropURL`, которую наши наследники смогут замещать для добавления своего кода.

```
// Собственный приемник данных, который принимает веб-адреса (URL)
class URLDropTarget : public wxDropTarget
{
public:
    URLDropTarget() { SetDataObject(new wxURLDataObject); }

    virtual void OnDropURL(wxCoord x, wxCoord y, const wxString& text)
    {
        // Конечно, реальная программа будет делать здесь
        // что-то более полезное
        wxMessageBox(text, wxT("URLDropTarget: получен URL"),
                     wxICON_INFORMATION | wxOK);
    }

    // Адреса не могут быть перемещены, только скопированы
    virtual wxDragResult OnDragOver(wxCoord x, wxCoord y,
                                    wxDragResult def)
        {
            return wxDragLink;
```

```
        }

    // Просто перенаправляем вызов к OnDropURL()
    virtual wxDragResult OnData(wxCoord x, wxCoord y,
                                wxDragResult def)
    {
        if ( !GetData() )
            return wxDragNone;

        OnDropURL(x, y, ((wxURLDataObject *)m_dataObject)->GetURL());

        return def;
    }
};

// Связываем приемник данных с окном
wxListBox* listBox = new wxListBox(parent, wxID_ANY);
listBox->SetDropTarget(new URLDropTarget);
```

11.3.5 Подробнее про wxDataObject

Как вы уже поняли, `wxDataObject` представляет собой данные, которые могут быть скопированы в/из буфера обмена, а также переданы при помощи перетаскивания. Вы должны знать, что `wxDataObject` — это «умное» хранилище данных, которое отличается от обычных контейнеров типа буфера в памяти или файла. Говоря «умное», я подразумеваю, что объект данных знает, какие форматы данных он поддерживает, и умеет отдавать данные в любом из поддерживаемых форматов.

Поддерживаемый формат — это формат, в котором данные могут быть затребованы из объекта данных или в котором их можно поместить в него. В общем случае объект может поддерживать различные форматы для ввода и для вывода. То есть объект данных может уметь отдавать данные в конкретном формате, но не может принять их, или наоборот.

Если вам необходимо реализовать наследник от `wxDataObject`, то у вас есть несколько вариантов решения этой проблемы:

1. Использовать один из встроенных классов. Вы можете использовать `wxTextDataObject`, `wxBitmapDataObject` или `wxFileDataObject` в простейших случаях, когда вам нужно поддерживать только один заданный формат и ваши данные могут быть представлены в виде текста, картинки или списка файлов.

2. Использовать `wxDataObjectSimple`. Наследование от `wxDataObjectSimple` является самым простым вариантом для поддержки собственных типов данных. Но вы сможете поддерживать только один формат, а поэтому вы, скорее всего, не сможете взаимодействовать с другими программами, но передача данных будет работать в пределах вашего приложения или между разными его копиями.

3. Наследовать от `wxCustomDataObject` (наследника от `wxDataObjectSimple`) для работы с форматами, определенными пользователем.

4. Использовать `wxDataObjectComposite`. Это простое, но достаточно мощное решение, которое позволяет вам поддерживать любое количество форматов (стандартных или собственных, реализованных на базе `wxDataObjectSimple`).

5. Использовать `wxDataObject` напрямую. Это наиболее гибкое и эффективное решение, но и наиболее трудоемкое в реализации.

Использование `wxDataObjectComposite` является самым простым способом обеспечения поддержки перетаскивания данных и работы с буфером обмена при использовании сразу нескольких форматов данных. Но, учитывая то, что каждый `wxDataObjectSimple` содержит все данные своего формата целиком, то такое решение являться не самым эффективный. Представьте себе, что вы хотите вставить в буфер обмена 200 страниц текста в вашем личном формате, а также в стандартных форматах RTF, HTML, Unicode и в виде обычного текста. Даже для современных компьютеров это будет очень медленно. Из соображений эффективности вам необходимо напрямую унаследовать свой класс от `wxDataObject`, научить его перечислять поддерживаемые форматы и выдавать данные в требуемом формате по запросу.

Механизм передачи данных, используемый для реализации операций, связанных с буфером обмена и перетаскиванием данных, не копирует данные до тех пор, пока приложение-приемник явно их не потребует. Пользователю обычно кажется, что данные помещаются в буфер обмена сразу после выбора команды копирования, однако в действительности они могут просто быть просто объявлены доступными без их физического копирования.

11.3.6 Наследование от wxDataObject

Давайте посмотрим какие действия необходимо сделать, чтобы создать класс-наследник от `wxDataObject`. Наследование от других упомянутых ранее классов делается аналогично, но проще в реализации, поэтому мы не будем рассматривать здесь все возможные случаи.

Каждый класс, унаследованный напрямую от `wxDataObject`, должен реализовывать все его виртуальные функции. Объекты данных, которые могут только отдавать или только принимать данные (другими словами, работают только в одном направлении), должны возвращать 0 из `GetFormatCount` для неподдерживаемого направления.

`GetAllFormats` принимает массив значений типа `wxDataFormat` и требуемое направление передачи данных получение (`Get`) или установка (`Set`). Скопируйте все поддерживаемые форматы для указанного направлении в массив форматов. Количество элементов в массиве определяется методом `GetFormatCount`.

`GetDataHere` принимает значение типа `wxDataFormat`, а также указатель на буфер (`void*`) и возвращает `true` в случае успеха и `false` при ошибке. Функция должна записать данные в указанном формате в переданный ей буфер. Это могут быть как текстовые или двоичные данные в произвольном формате, но они должны распознаваться функцией `SetData`.

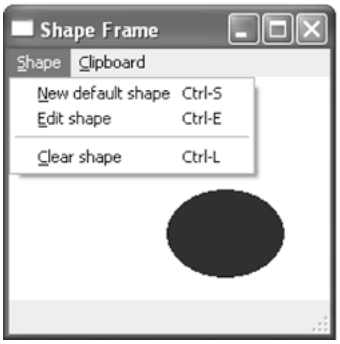

Рис. 11.1: Пример реализации перетаскивания в wxWidgets

`GetDataSize` принимает значение типа `wxDataFormat` и возвращает размер данных для указанного формата.

`GetFormatCount` возвращает количество форматов, доступных для передачи или приема данных.

`GetPreferredFormat` принимает направление и возвращает предпочтительный `wxDataFormat` для указанного направления.

`SetData` принимает значение типа `wxDataFormat`, размер буфера и указатель (типа `void*`) на буфер. Вы интерпретируете данные в буфере так, как это требуется для данного объекта, например, копируя их во внутреннюю структуру. Данная функция должна вернуть `true`, если все прошло удачно, и `false` в противном случае.

11.3.7 Пример реализации перетаскивания на wxWidgets

Для демонстрации написания собственных наследников от `wxDataObject`, работающих с пользовательским форматом данных, мы будем использовать стандартный пример реализации перетаскивания из дистрибутива wxWidgets, который вы можете найти в папке samples/dnd. Данный пример показывает простые фигуры: треугольник, прямоугольник или эллипс, позволяет их редактировать, перемещать в новую позицию, копировать в буфер обмена и вставлять обратно. Выберите команду «New frame»[1] в меню «File»[2], чтобы показать новое окно для фигур. Такое окно показано на рис. 11.1.

Фигуры создаются с использованием классов, унаследованных от `DnDShape`. `DnDShapeDataObject` — это пользовательский объект данных. Перед началом изучения реализации `DndShapeDataObject` давайте посмотрим, как приложение использует этот класс.

При вызове операции копирования копия текущей фигуры записывается в `DndShapeDataObject` (на случай если фигура будет удалена из приложения в то время пока данные все еще будут находится в буфере обмена) и помещается в буфер обмена. Вот код, который это делает:

```
void DnDShapeFrame::OnCopyShape(wxCommandEvent& event)
{
```

[1] Новый фрейм (англ.)
[2] Файл (англ.)

```
    if ( m_shape )
    {
        wxClipboardLocker clipLocker;
        if ( !clipLocker )
        {
            wxLogError(wxT("Не могу открыть буфер обмена"));
            return;
        }

        wxTheClipboard->AddData(new DnDShapeDataObject(m_shape));
    }
}
```

Вставка также реализуется достаточно просто. Необходимо вызвать `wxClipboard::GetData` для того, чтобы получить данные из буфера, а затем извлечь информацию о фигуре из полученного объекта данных. Создадим функцию обновления пользовательского интерфейса, которая будет делать доступной команду «Paste»[3] в меню, только если в буфере обмена в данный момент находится наш собственный формат данных для фигур. `shapeFormatId` — это глобальная переменная, содержащая имя используемого нами формата данных `wxShape`.

```
void DnDShapeFrame::OnPasteShape(wxCommandEvent& event)
{
    wxClipboardLocker clipLocker;
    if ( !clipLocker )
    {
        wxLogError(wxT("Не могу открыть буфер обмена"));
        return;
    }

    DnDShapeDataObject shapeDataObject(NULL);
    if ( wxTheClipboard->GetData(shapeDataObject) )
    {
        SetShape(shapeDataObject.GetShape());
    }
    else
    {
        wxLogStatus(wxT("Нет фигуры в буфере обмена"));
    }
}

void DnDShapeFrame::OnUpdateUIPaste(wxUpdateUIEvent& event)
{
    event.Enable( wxTheClipboard->
                      IsSupported(wxDataFormat(shapeFormatId)) );
}
```

[3] Вставка (англ.)

Для реализации перетаскивания нам потребуется класс приемника данных, чтобы уведомить приложение, когда данные будут брошены. Объект класса `DnDShapeDropTarget` содержит `DnDShapeDataObject`, который служит буфером для принимаемых данных и заполняется в момент вызова `OnData`. Далее приведены объявление и реализация класса `DnDShapeDropTarget`:

```
class DnDShapeDropTarget : public wxDropTarget
{
public:
    DnDShapeDropTarget(DnDShapeFrame *frame)
        : wxDropTarget(new DnDShapeDataObject)
    {
        m_frame = frame;
    }

    // Заменяем чисто виртуальные функции базового класса
    virtual wxDragResult OnEnter(wxCoord x, wxCoord y, wxDragResult def)
    {
        m_frame->SetStatusText(_T("Курсор мыши вошел в пределы окна"));
        return OnDragOver(x, y, def);
    }

    virtual void OnLeave()
    {
        m_frame->SetStatusText(_T("Курсор мыши вышел за пределы окна"));
    }

    virtual wxDragResult OnData(wxCoord x, wxCoord y, wxDragResult def)
    {
        if ( !GetData() )
        {
            wxLogError(wxT("Ошибка получения данных"));

            return wxDragNone;
        }

        // Уведомим окно о том, что данные были брошены
        m_frame->OnDrop(x, y,
                ((DnDShapeDataObject *)GetDataObject())->GetShape());

        return def;
    }

private:
    DnDShapeFrame *m_frame;
};
```

Приемник данных связывается с окном при инициализации:

```
DnDShapeFrame::DnDShapeFrame(wxFrame *parent)
             : wxFrame(parent, wxID_ANY, _T("Shape Frame"))
{
    ...
    SetDropTarget(new DnDShapeDropTarget(this));
    ...
}
```

Перетаскивание начинается в тот момент, когда пользователь нажимает левую клавишу мыши. Функция-обработчик сначала создает `wxDropSource`, передавая ему `DnDShapeDataObject`, а затем, вызывая `DoDragDrop`, инициализирует операцию перетаскивания. `DndShapeFrame::OnDrag` выглядит так:

```
void DnDShapeFrame::OnDrag(wxMouseEvent& event)
{
    if ( !m_shape )
    {
        event.Skip();
        return;
    }

    // Начинаем операцию перетаскивания
    DnDShapeDataObject shapeData(m_shape);
    wxDropSource source(shapeData, this);

    const wxChar *pc = NULL;
    switch ( source.DoDragDrop(true) )
    {
        default:
        case wxDragError:
            wxLogError(wxT("Во время перетаскивания возникла ошибка"));
            break;

        case wxDragNone:
            SetStatusText(_T("Ничего не произошло"));
            break;

        case wxDragCopy:
            pc = _T("скопирована");
            break;

        case wxDragMove:
            pc = _T("перемещена");
            if ( ms_lastDropTarget != this )
            {
                // Не удаляем фигуру, если мы
                // бросили ее саму на себя
```

```
                SetShape(NULL);
            }
            break;

        case wxDragCancel:
            SetStatusText(_T("Операция перетаскивания отменена"));
            break;
    }

    if ( pc )
    {
        SetStatusText(wxString(_T("Фигура успешно ")) + pc);
    }
    //else: в строку статуса уже добавлено соответствующее сообщение
}
```

Когда фиксируется бросание данных (пользователь отпускает кнопку мыши) wxWidgets вызывает `DnDShapeDropTarget::OnData`, которая, в свою очередь, вызывает `DndShapeFrame::OnDrop` с новым объектом `DndShape` для того, чтобы установить фигуру в новую позицию. После чего операция перетаскивания завершается.

```
void DnDShapeFrame::OnDrop(wxCoord x, wxCoord y, DnDShape *shape)
{
    ms_lastDropTarget = this;

    wxPoint pt(x, y);

    wxString s;
    s.Printf(wxT("Фигура брошена в (%d, %d)"), pt.x, pt.y);
    SetStatusText(s);

    shape->Move(pt);
    SetShape(shape);
}
```

Осталось реализовать собственный класс-наследник от `wxDataObject`. Для ясности мы будем разбирать его реализацию постепенно. Сначала мы покажем объявление идентификатора собственного формата, объявление класса `DndShapeDataObject`, его конструктор и деструктор, а также данные-члены.

Идентификатор нашего формата (`shapeFormatId`) — это глобальная переменная, используемая во всем коде примера. Конструктор, используя `GetDataHere`, принимает копию фигуры (если она передана). Получение копии также можно реализовать с помощью функции `DndShape::Clone`, если вы ее создадите. Деструктор класса `DnDShapeDataObject` просто удаляет фигуру.

`DndShapeDataObject` может создать растровое изображение или метафайл (на платформах, которые его поддерживают) для рисования нашей фигуры. Таким образом, необходимо иметь `wxBitmapDataObject` и `wxMetaFileDataObject` среди членов класса, чтобы создать и при необходимости закешировать данные в этих форматах,

а также связанные с ними флаги, указывающие, содержат ли эти переменные корректные данные.

```
// Идентификатор собственного формата данных
static const wxChar *shapeFormatId = wxT("wxShape");

class DnDShapeDataObject : public wxDataObject
{
public:
    // Конструктор не копирует указатель, так как данные не
    // должны исчезнуть, пока объект находится в буфере обмена
    DnDShapeDataObject(DnDShape *shape = (DnDShape *)NULL)
    {
        if ( shape )
        {
            // Мы должны скопировать фигуру, так как оригинал
            // может быть удален пока наш объект
            // все еще находится в буфере обмена (к примеру) -
            // и мы используем функцию сериализации здесь для
            // копирования данных
            void *buf = malloc(shape->DnDShape::GetDataSize());
            shape->GetDataHere(buf);
            m_shape = DnDShape::New(buf);

            free(buf);
        }
        else
        {
            // Нечего копировать
            m_shape = NULL;
        }

        // Произвольная строка однозначно идентифицирующая наш формат
        m_formatShape.SetId(shapeFormatId);

        // Мы не рисуем фигуру в растровое изображение до тех пор, пока
        // это действительно не понадобится (пока нас не попросят)
        m_hasBitmap = false;
        m_hasMetaFile = false;
    }

    virtual ~DnDShapeDataObject() { delete m_shape; }

    // После вызова этой функции фигура принадлежит вызывающему
    // коду и поэтому код ответственен за ее удаление
    DnDShape *GetShape()
    {
```

```
        DnDShape *shape = m_shape;

        m_shape = (DnDShape *)NULL;
        m_hasBitmap = false;
        m_hasMetaFile = false;

        return shape;
    }

    // Остальные функции класса пока пропущены
    ...

    // Члены класса с данными
private:
    wxDataFormat         m_formatShape; // наш формат данных
    wxBitmapDataObject   m_dobjBitmap;  // данные в формате растрового изображения
    bool                 m_hasBitmap;   // true, если m_dobjBitmap корректен
    wxMetaFileDataObject m_dobjMetaFile;// хранит данные в формате метафайла
    bool                 m_hasMetaFile; // true, если dobjMetaFile корректен
    DnDShape             *m_shape;      // наша фигура
};
```

Теперь напишем функции, отвечающие на вопросы относительно поддерживаемых нашим объектом типов данных. `GetPreferredFormat` просто возвращает «родной» формат для объекта — значение переменной `m_formatShape`, которую мы инициализировали строкой «wxShape» в конструкторе. `GetFormatCount` возвращает количество форматов, поддерживаемых для каждого направления. В нашем случае мы поддерживаем только данные в формате метафайла или растрового изображения, которые доступны только при получении данных (т.е. работают только в одном направлении). `GetDataSize` возвращает размер данных для указанного формата. В нашем случае для этого придется сначала создать растровое изображение или метафайл, а потом возвратить размер созданного объекта.

```
virtual wxDataFormat GetPreferredFormat(Direction dir) const
{
    return m_formatShape;
}

virtual size_t GetFormatCount(Direction dir) const
{
    // Наш собственный формат поддерживается как для GetData(),
    // так и для SetData()
    size_t nFormats = 1;
    if ( dir == Get )
    {
        // Но данные в виде растрового изображения и метафайла
        // поддерживаются только для вывода
```

```
        nFormats += m_dobjBitmap.GetFormatCount(dir);
        nFormats += m_dobjMetaFile.GetFormatCount(dir);
    }

    return nFormats;
}

virtual void GetAllFormats(wxDataFormat *formats, Direction dir) const
{
    formats[0] = m_formatShape;
    if ( dir == Get )
    {
        // В случае вывода данных мы дополнительно поддерживаем
        // изображения и метафайл (если работаем под Windows)
        m_dobjBitmap.GetAllFormats(&formats[1], dir);

        // Даже и не думайте что m_dobjBitmap поддерживает только один формат
        m_dobjMetaFile.GetAllFormats(
            &formats[1 + m_dobjBitmap.GetFormatCount(dir)], dir);
    }
}

virtual size_t GetDataSize(const wxDataFormat& format) const
{
    if ( format == m_formatShape )
    {
        return m_shape->GetDataSize();
    }
    else if ( m_dobjMetaFile.IsSupported(format) )
    {
        if ( !m_hasMetaFile )
            CreateMetaFile();

        return m_dobjMetaFile.GetDataSize(format);
    }
    else
    {
        wxASSERT_MSG( m_dobjBitmap.IsSupported(format),
                      wxT("Неожиданный формат") );

        if ( !m_hasBitmap )
            CreateBitmap();

        return m_dobjBitmap.GetDataSize();
    }
}
```

GetDataHere копирует требуемые данные в переданный буфер (void*):

```
virtual bool GetDataHere(const wxDataFormat& format, void *pBuf) const
{
    if ( format == m_formatShape )
    {
        // Использует структуру ShapeDump, чтобы сериализировать себя в буфер
        m_shape->GetDataHere(pBuf);

        return true;
    }
    else if ( m_dobjMetaFile.IsSupported(format) )
    {
        if ( !m_hasMetaFile )
            CreateMetaFile();

        return m_dobjMetaFile.GetDataHere(format, pBuf);
    }
    else
    {
        wxASSERT_MSG( m_dobjBitmap.IsSupported(format),
                      wxT("Неожиданный формат") );

        if ( !m_hasBitmap )
            CreateBitmap();

        return m_dobjBitmap.GetDataHere(pBuf);
    }
}
```

SetData поддерживает только «родной» формат, а значит нам необходимо только вызвать DndShape::New, чтобы создать новую фигуру из переданных данных.

```
virtual bool SetData(const wxDataFormat& format,
                     size_t len, const void *buf)
{
    wxCHECK_MSG( format == m_formatShape, false,
                 wxT( "Неожиданный формат") );

    delete m_shape;
    m_shape = DnDShape::New(buf);

    // Фигура была изменена
    m_hasBitmap = false;
    m_hasMetaFile = false;

    return true;
}
```

Способ, которым DndShape сериализует себя в буфер и восстанавливает из него очень прост: он использует структуру ShapeDump, которая хранит данные о фигуре. Примерно вот так:

```
// Статическая функция, которая создает фигуру из буфера
DnDShape *DnDShape::New(const void *buf)
{
    const ShapeDump& dump = *(const ShapeDump *)buf;
    switch ( dump.k )
    {
        case Triangle:
            return new DnDTriangularShape(
                            wxPoint(dump.x, dump.y),
                            wxSize(dump.w, dump.h),
                            wxColour(dump.r, dump.g, dump.b));

        case Rectangle:
            return new DnDRectangularShape(
                            wxPoint(dump.x, dump.y),
                            wxSize(dump.w, dump.h),
                            wxColour(dump.r, dump.g, dump.b));

        case Ellipse:
            return new DnDEllipticShape(
                            wxPoint(dump.x, dump.y),
                            wxSize(dump.w, dump.h),
                            wxColour(dump.r, dump.g, dump.b));

        default:
            wxFAIL_MSG(wxT("Странная фигура!"));
            return NULL;
    }
}

// Получаем размер данных
size_t DndShape::GetDataSize() const
{
    return sizeof(ShapeDump);
}

// Сериализация в буфер void*
void DndShape::GetDataHere(void *buf) const
{
    ShapeDump& dump = *(ShapeDump *)buf;
    dump.x = m_pos.x;
    dump.y = m_pos.y;
    dump.w = m_size.x;
```

```
    dump.h = m_size.y;
    dump.r = m_col.Red();
    dump.g = m_col.Green();
    dump.b = m_col.Blue();
    dump.k = GetKind();
}
```

Возвратимся к нашему классу `DnDShapeDataObject`. Функции, которые при необходимости создают данные в требуемом формате (растровое изображения или метафайл) выглядят так:

```
void DnDShapeDataObject::CreateMetaFile() const
{
    wxPoint pos = m_shape->GetPosition();
    wxSize size = m_shape->GetSize();

    wxMetaFileDC dcMF(wxEmptyString, pos.x + size.x, pos.y + size.y);

    m_shape->Draw(dcMF);

    wxMetafile *mf = dcMF.Close();

    DnDShapeDataObject *self = (DnDShapeDataObject *)this;
    self->m_dobjMetaFile.SetMetafile(*mf);
    self->m_hasMetaFile = true;

    delete mf;
}

void DnDShapeDataObject::CreateBitmap() const
{
    wxPoint pos = m_shape->GetPosition();
    wxSize size = m_shape->GetSize();
    int x = pos.x + size.x,
        y = pos.y + size.y;
    wxBitmap bitmap(x, y);
    wxMemoryDC dc;
    dc.SelectObject(bitmap);
    dc.SetBrush(wxBrush(wxT("white"), wxSOLID));
    dc.Clear();
    m_shape->Draw(dc);
    dc.SelectObject(wxNullBitmap);

    DnDShapeDataObject *self = (DnDShapeDataObject *)this;
    self->m_dobjBitmap.SetBitmap(bitmap);
    self->m_hasBitmap = true;
}
```

На этом мы закончили реализацию нашего объекта данных, опустив подробности о том, как фигуры себя отрисовывают, а также код для создания пользовательского интерфейса. Если вас интересуют эти детали, то вы всегда можете посмотреть исходный код приложения в папке samples/dnd.

11.3.8 Существующие реализации перетаскивания в wxWidgets

Ряд элементов управления, присутствующих в библиотеке, могут сильно облегчить вам реализацию перетаскивания.

wxTreeCtrl

Вы можете использовать специальные макросы `EVT_TREE_BEGIN_DRAG` или `EVT_TREE_BEGIN_RDRAG` в таблице событий для перехвата начала и конца перетаскивания с использованием левой или правой клавиши мыши соответственно. Всю работу по определению начала перетаскивания возьмет на себя внутренний код элемента управления. Если вы хотите разрешить дереву использовать внутреннюю реализацию перетаскивания (которая в конце генерирует событие `EVT_TREE_END_DRAG`), то просто вызовите `wxTreeEvent::Allow` в вашей процедуре-обработчике `EVT_TREE_BEGIN_DRAG`. При такой реализации будет автоматически создан курсор для перетаскивания, и библиотека сама проконтролирует остальные аспекты процесса. Результат перетаскивания полностью зависит от приложения и определяется кодом функции-обработчика завершающего события `EVT_TREE_END_DRAG`.

Следующий пример показывает, как правильно использовать связанные с процессом перетаскивания события, генерируемые деревом (`wxTreeCtrl`). В следующем примере когда пользователь перетаскивает ветку дерева в новое место, то в дерево вставляется копия оригинальной ветви.

```
BEGIN_EVENT_TABLE(MyTreeCtrl, wxTreeCtrl)
    EVT_TREE_BEGIN_DRAG(TreeTest_Ctrl, MyTreeCtrl::OnBeginDrag)
    EVT_TREE_END_DRAG(TreeTest_Ctrl, MyTreeCtrl::OnEndDrag)
END_EVENT_TABLE()

void MyTreeCtrl::OnBeginDrag(wxTreeEvent& event)
{
    // Вы должны явно разрешить перетаскивание
    if ( event.GetItem() != GetRootItem() )
    {
        m_draggedItem = event.GetItem();

        wxLogMessage(wxT("OnBeginDrag: начато перетаскивание '%s'"),
                     GetItemText(m_draggedItem).c_str());

        event.Allow();
    }
    else
```

```
    {
        wxLogMessage(wxT("OnBeginDrag: Нельзя перетаскивать эту ветвь."));
    }
}

void MyTreeCtrl::OnEndDrag(wxTreeEvent& event)
{
    wxTreeItemId itemSrc = m_draggedItem,
                 itemDst = event.GetItem();
    m_draggedItem = (wxTreeItemId)0l;

    // Куда вставлять копию
    if ( itemDst.IsOk() && !ItemHasChildren(itemDst) )
    {
        // Вставляем копию на один уровень выше
        itemDst = GetItemParent(itemDst);
    }

    if ( !itemDst.IsOk() )
    {
        wxLogMessage(wxT("OnEndDrag: Нельзя перетащить сюда."));
        return;
    }

    wxString text = GetItemText(itemSrc);
    wxLogMessage(wxT("OnEndDrag: '%s' скопирована в '%s'."),
                 text.c_str(), GetItemText(itemDst).c_str());

    // Добавляем ветвь сюда
    int image = wxGetApp().ShowImages() ? TreeCtrlIcon_File : -1;
    AppendItem(itemDst, text, image);
}
```

Если вы предпочитаете самостоятельно реализовать перетаскивание (к примеру, с использованием `wxDropSource`), то просто не вызывайте `wxTreeEvent::Allow` в функции `OnBeginDrag`. Вместо этого добавьте код для реализации своего виденья процесса. В таком случае вы не получите уведомления о конце перетаскивания (`EVT_TREE_END_DRAG`), так как `wxDropSource::DoDragDrop` вернет вам управление и код результата сразу после завершения операции.

wxListCtrl

В данном элементе управления, в отличие от предыдущего, отсутствует готовый курсор для визуализации перетаскивания и сообщение об окончании процесса. Все, что он может — сообщить вам о его начале. Для поддержки такой функциональности необходимо использовать макросы `EVT_LIST_BEGIN_DRAG` или `EVT_LIST_BEGIN_RDRAG` и самостоятельно написать код реализующий перетаскивание. Используя макросы `EVT_LIST_COL_BEGIN_DRAG`, `EVT_LIST_COL_DRAGGING` и `EVT_LIST_COL_END_DRAG`, вы

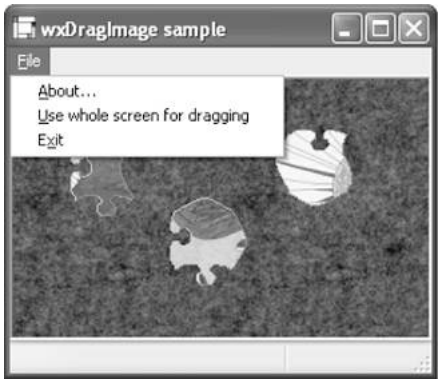

Рис. 11.2: Пример использования wxDragImage

можете узнать о попытках пользователя изменить размеры колонок (перетаскивание разделителя) и соответственно отреагировать.

wxDragImage

`wxDragImage` является удобным классом, который может понадобиться вам при реализации перетаскивания. Он рисует картинку-курсор поверх окна и обеспечивает способ перемещать ее не «повреждая» низлежащее окно. В стандартной реализации этот эффект достигается путем сохранения копии части окна, находящийся под нашим рисунком, и перерисовывания ее вместе с картинкой перетаскивания по мере необходимости.

Рисунок 11.2 показывает главное окно примера `wxDragImage` из дистрибутива wxWidgets, который вы можете найти в папке samples/dragimag. При визуализации перетаскивания для каждой из трех фигур используется оригинальная картинка (`wxDragImage`): собственно фигура, иконка приложения и картинка, динамически создаваемая из строки текста. Если вы активируете пункт меню «Use Whole Screen for Dragging»[4], то сможете перетащить фигуру за пределы окна. При компиляции данного примера под Windows вы можете выбрать между стандартной реализацией `wxDragImage` и «родным» классом, предоставляемым системой. Для управления выбором используется директива `wxUSE_GENERIC_DRAGIMAGE` в файле dragimag.cpp.

Как только стали уверены, что обнаружили начало перетаскивания, необходимо создать `wxDragImage`. Вызовете `BeginDrag`, чтобы инициировать и `EndDrag`, чтобы остановить операцию. При визуализации перемещения рисунка по экрану вначале покажите его функцией `Show`, а после этого используйте метод `Move`. Если вы хотите обновить содержимое экрана во время перетаскивания (например, подсветить предмет, как это сделано в нашем примере), то вызовете `Hide`, обновите окно, вызовете `Move` и снова `Show`.

Вы можете ограничится пределами одного окна или разрешить перетаскивать объект за его пределы. Во втором случае, вы можете ограничиться какой-либо областью для экономии ресурсов или использовать весь экран. Если вы хотите дать пользователю возможность перетаскивать объекты между двумя окнами с разными родителями, то вы должны разрешить перетаскивание по всему экрану. К сожале-

[4]Разрешить перетаскивать по всему экрану (англ.)

нию, полноэкранное перетаскивание не идеально, так как его реализация подразумевает создание снимка экрана вначале и не принимает во внимание изменения, происходящие в других приложениях во время всего процесса.

В следующем фрагменте кода, основанном на примере, показанном на рисунке 11.2, `MyCanvas` отображает несколько фигур класса `DragShape`, каждая из которых имеет связанное растровое изображение. Вначале операции перетаскивания создается новый `wxDragImage` с использованием связанного с фигурой изображения и вызывается `BeginDrag`. При обнаружении движения мыши вызывается `wxDragImage::Move`, чтобы отобразить рисунок в соответствующей позиции окна. И, наконец, когда пользователь отпускает левую кнопку мыши, мы перерисовываем передвигаемую фигуру в новой позиции и удаляем `wxDragImage`.

```
void MyCanvas::OnMouseEvent(wxMouseEvent& event)
{
    if (event.LeftDown())
    {
        DragShape* shape = FindShape(event.GetPosition());
        if (shape)
        {
            // Мы фиксируем нажатие кнопки, но ждем минимального
            // перемещения курсора, чтобы начать перетаскивание
            m_dragMode = TEST_DRAG_START;
            m_dragStartPos = event.GetPosition();
            m_draggedShape = shape;
        }
    }
    else if (event.LeftUp() && m_dragMode != TEST_DRAG_NONE)
    {
        // Заканчиваем перетаскивание
        m_dragMode = TEST_DRAG_NONE;

        if (!m_draggedShape || !m_dragImage)
            return;

        m_draggedShape->SetPosition(m_draggedShape->GetPosition()
                            + event.GetPosition() - m_dragStartPos);

        m_dragImage->Hide();
        m_dragImage->EndDrag();
        delete m_dragImage;
        m_dragImage = NULL;

        m_draggedShape->SetShow(true);
        m_draggedShape->Draw(dc);
        m_draggedShape = NULL;
    }
    else if (event.Dragging() && m_dragMode != TEST_DRAG_NONE)
```

```
{
    if (m_dragMode == TEST_DRAG_START)
    {
        // Мы начинаем перетаскивание, если курсор сдвинулся
        // на несколько пикселей
        int tolerance = 2;
        int dx = abs(event.GetPosition().x - m_dragStartPos.x);
        int dy = abs(event.GetPosition().y - m_dragStartPos.y);
        if (dx <= tolerance && dy <= tolerance)
            return;

        // Начинаем перетаскивание
        m_dragMode = TEST_DRAG_DRAGGING;

        if (m_dragImage)
            delete m_dragImage;

        // Удаляем перетаскиваемую фигуру из окна
        m_draggedShape->SetShow(false);

        wxClientDC dc(this);
        EraseShape(m_draggedShape, dc);
        DrawShapes(dc);

        m_dragImage = new wxDragImage(
                                    m_draggedShape->GetBitmap());

        // Смещение между верхним левым углом рисунка фигуры
        // и текущей позицией фигуры
        wxPoint beginDragHotSpot = m_dragStartPos
                                   m_draggedShape->GetPosition();

        // Всегда используем координаты относительно нашего
        // окна (клиентские координаты)
        if (!m_dragImage->BeginDrag(beginDragHotSpot, this))
        {
            delete m_dragImage;
            m_dragImage = NULL;
            m_dragMode = TEST_DRAG_NONE;

        } else
        {
            m_dragImage->Move(event.GetPosition());
            m_dragImage->Show();
        }
    }
    else if (m_dragMode == TEST_DRAG_DRAGGING)
```

```
        {
            // Перемещаем картинку
            m_dragImage->Move(event.GetPosition());
        }
    }
}
```

Если вы захотите самостоятельно рисовать картинку вместо того, чтобы просто установить ее как картинку перетаскивания, то используйте для этого `wxGenericDragImage`, предварительно заместив в нем функции `wxDragImage::DoDrawImage` и `wxDragImage::GetImageRect`. На всех платформах, кроме Windows, `wxDragImage` — это псевдоним для `wxGenericDragImage`. Реализация в Windows не поддерживает `DoDrawImage` и, к тому же, ограничена использованием полупрозрачных рисунков, поэтому вы, возможно, предпочтете использовать `wxGenericDragImage` для всех платформ.

Когда вы начинаете перетаскивание, то как правило, вы должны сначала стереть объект, который пользователь собирается перетащить, и только потом вызвать `wxDragImage::Show`. Таким образом, `wxDragImage` захватит изображение окна уже без перетаскиваемого объекта и, соответственно, он будет накладываться на сохраненный изображение так, как будто он уже перетаскивается. Такой подход вызывает легкое мерцание вначале. Чтобы избежать этого (применимо только к `wxGenericDragImage`), замените функцию `UpdateBackingFromWindow` и нарисуйте содержимое окна, уже без перетаскиваемого элемента, на контексте устройства в памяти, переданного в эту процедуру. Теперь вам нет необходимости стирать объект перед тем, как показать рисунок перетаскивания, и, когда он сдвинется на новое место, будет отрисовано правильное содержимое окна без побочных эффектов.

11.4 Итоги

В этой главе мы изучили работу с буфером обмена. Также мы коснулись вопросов, связанных с реализацией перетаскивания (как с точки зрения окна-источника, так и с точки зрения окна-приемника), подробно разобрав большинство аспектов, связанных с поддержкой этого процесса. Дополнительную информацию по данному вопросу вы можете получить, ознакомившись с примерами приложений, поставляемыми с wxWidgets. Вы можете найти их в папках samples/dnd, samples/dragimag и samples/treectrl.

В следующем разделе мы вернемся к вопросу использования элементов управления и опишем некоторые продвинутые классы, которые помогут вам поднять ваше приложение на новый уровень.