Глава 6

# Обработка данных с устройств ввода

©Перевод сделан Грубниковым А.Г.

Все GUI-приложения должны каким-либо образом реагировать на данные, поступающие от устройства ввода. Эта глава покажет как можно взаимодействовать с мышью, клавиатурой и джойстиком.

## 6.1 Получение данных от мыши

Упрощенно говоря, приложение получает от мыши два вида событий: основные события мыши, посылаемые с помощью класса `wxMouseEvent`, и «сырые» события, передаваемые вашей функции-обработчику неинтерпретированными. При этом действия, связанные с элементами управления (такими как, например, кнопка), часто являются результатом интерпретации событий от мыши (или других устройств) как отдельных команд.

Например, когда вы добавляете макрос `EVT_BUTTON` в таблицу событий, то вы перехватываете событие `wxCommandEvent`, которое было сгенерировано классом `wxButton`. Это, в свою очередь, происходит следующим образом: кнопка перехватывает событие от мыши `EVT_LEFT_DOWN` и порождает управляющее событие. Конечно, на большинстве платформ, класс `wxButton` реализован как «родной» и не использует низкоуровневую обработку событий wxWidgets, но так будет происходить для несуществующих на платформе классов.

Так как примеры обработки управляющих событий были уже показаны, то мы сосредоточимся на основных событиях, генерируемых мышью.

Вы можете перехватить события отпускания (key up), нажатия (key down) и двойного клика левой, средней или правой кнопками мыши. Можно также перехватить движения мыши, как с нажатой кнопкой, так и без. Вы можете перехватить события, говорящие о входе курсора в область окна и о выходе из этой области. Наконец, возможно перехватить события от колесика прокрутки, если на мыши таковое присутствует.

После получения событий от мыши, вы можете проверить состояние ее кнопок, а также клавиш-модификаторов (Shift, Alt, Control и Meta). Также доступна информация о текущем положении указателя мыши относительно верхнего левого угла клиентской области окна.

Таблица 6.1 содержит макросы таблицы событий, которые вы можете использовать. Класс `wxMouseEvent` не передается по цепочке родителям окна, поэтому для обработки таких событий вы должны унаследовать свой класс от класса окна или от `wxEvtHandler`, а потом модифицировать обработчик событий для необходимых подокон с помощью функций `SetEventHandler` или `PushEventHandler`. Или же, вы можете выполнить динамическую привязку с помощью функции `Connect`.

Таблица 6.1: Макросы событий мыши

| | |
|---|---|
| EVT_LEFT_DOWN(func) | Обрабатывает событие wxEVT_LEFT_DOWN, генерируемое при смене состояния левой кнопки на «нажата». |
| EVT_LEFT_UP(func) | Обрабатывает событие wxEVT_LEFT_UP, генерируемое при смене состояния левой кнопки на «отпущена». |
| EVT_LEFT_DCLICK(func) | Обрабатывает событие wxEVT_LEFT_DCLICK, генерируемое при двойном клике левой кнопкой. |
| EVT_MIDDLE_DOWN(func) | Обрабатывает событие wxEVT_MIDDLE_DOWN, генерируемое при смене состояния средней кнопки на «нажата». |
| EVT_MIDDLE_UP(func) | Обрабатывает событие wxEVT_MIDDLE_UP, генерируемое при смене состояния средней кнопки на «отпущена». |
| EVT_MIDDLE_DCLICK(func) | Обрабатывает событие wxEVT_MIDDLE_DCLICK, генерируемое при двойном клике средней кнопкой. |
| EVT_RIGHT_DOWN(func) | Обрабатывает событие wxEVT_RIGHT_DOWN, генерируемое при смене состояния правой кнопки на «нажата». |
| EVT_RIGHT_UP(func) | Обрабатывает событие wxEVT_RIGHT_UP, генерируемое при смене состояния правой кнопки на «отпущена». |
| EVT_RIGHT_DCLICK(func) | Обрабатывает событие wxEVT_RIGHT_DCLICK, генерируемое при двойном клике правой кнопкой. |
| EVT_MOTION(func) | Обрабатывает событие wxEVT_MOTION, генерируемое при движении указателя мыши. |
| EVT_ENTER_WINDOW(func) | Обрабатывает событие wxEVT_ENTER_WINDOW, генерируемое при входе указателя в область окна. |
| EVT_LEAVE_WINDOW(func) | Обрабатывает событие wxEVT_LEAVE_WINDOW, генерируемое при выходе указателя из области окна. |
| EVT_MOUSEWHEEL(func) | Обрабатывает событие wxEVT_MOUSEWHEEL, генерируемое при движении колесика прокрутки у мыши. |

Таблица 6.1: Макросы событий мыши (продолжение)

| | |
|---|---|
| EVT_MOUSE_EVENTS(func) | Обрабатывает все события, производимые мышью. |

### 6.1.1 Обработка событий о движении мыши и нажатии кнопок

Ниже представлены главные функции событий мыши, которые вы можете использовать для обработки событий о передвижениях мыши и нажатиях на ее кнопки.

Для проверки, нажаты ли клавиши-модификаторы во время перехвата события, используйте функции AltDown, MetaDown, ControlDown или ShiftDown. Функция CmdDown используется для проверки, нажата ли клавиша Meta (на Mac OS X) или клавиша Control (на других платформах). Дополнительную информацию можно найти далее в этой главе в разделе «Разновидности клавиш-модификаторов».

Для определения, какая кнопка мыши нажата в данный момент, используйте функции `LeftIsDown`, `MiddleIsDown` и `RightIsDown`. Также можно проверить, нажата ли какая-либо кнопка, послав событие `wxMOUSE_BTN_LEFT`, `wxMOUSE_BTN_MIDDLE`, `wxMOUSE_BTN_RIGHT` или `wxMOUSE_BTN_ANY` объекту `Button`. Обратите внимание, что это проверка на нажатие какой-либо кнопки на момент совершения события, а не при смене его состояния.

В Mac OS X клавиша Command означает клавишу Meta, а клавиша Alt — Option. Так как часто на Mac-комьютерах мышь имеет только одну кнопку мыши, то пользователь должен удерживать Control во время клика, чтобы получить событие клика правой кнопкой. Это означает, что в Mac OS невозможно сделать правый клик одновременно с Control, если у вас нет обычной мыши с двумя или тремя кнопками.

Вы можете выяснить тип произошедшего события с помощью функций `Dragging` (движение курсора с удерживаемой кнопкой), `Moving` (без нее), `Entering`, `Leaving`, `ButtonDown`, `ButtonUp`, `ButtonDClick`, `LeftClick`, `LeftDClick`, `LeftUp`, `RightClick`, `RightDClick`, `RightUp`, `ButtonUp` и `IsButton`.

Чтобы получить позицию курсора в пикселях монитора (относительно верхнего левого угла клиентской области окна) используйте функции `GetPosition` или `GetX` и `GetY`. Также возможно получить позицию в логических координатах, передав контекст устройства функции `GetLogicalPosition`.

Вот пример обработки событий мыши в простом демонстрационном приложении:

```
BEGIN_EVENT_TABLE(DoodleCanvas, wxWindow)
  EVT_MOUSE_EVENTS(DoodleCanvas::OnMouseEvent)
END_EVENT_TABLE()

void DoodleCanvas::OnMouseEvent(wxMouseEvent& event)
{
  static DoodleSegment *s_currentSegment = NULL;

  wxPoint pt(event.GetPosition());

  if (s_currentSegment && event.LeftUp())
  {
```

```
    // Заканчиваем сегмент, если пользователь отпустил левую кнопку
    if (s_currentSegment->GetLines().GetCount() == 0)
    {
      // Если сегмент пустой, то удаляем его
      delete s_currentSegment;
      s_currentSegment = (DoodleSegment *) NULL;
    }
    else
    {
      // Сохраняем сегмент
      DrawingDocument *doc = GetDocument();

      doc->GetCommandProcessor()->Submit(
        new DrawingCommand(wxT("Add Segment"), DOODLE_ADD,
          doc, s_currentSegment));
      doc->Modify(true);
      s_currentSegment = NULL;
    }
  }
  else if (m_lastX > -1 && m_lastY > -1 && event.Dragging())
  {
    // При движении мыши с нажатой кнопкой, добавляем линию в текущий сегмент
    if (!s_currentSegment)
      s_currentSegment = new DoodleSegment;

    DoodleLine *newLine = new DoodleLine(m_lastX, m_lastY, pt.x, pt.y);
    s_currentSegment->GetLines().Append(newLine);

    wxClientDC dc(this);
    DoPrepareDC(dc);

    dc.SetPen(*wxBLACK_PEN);
    dc.DrawLine( m_lastX, m_lastY, pt.x, pt.y);
  }
  m_lastX = pt.x;
  m_lastY = pt.y;
}
```

В этом приложении сегменты линий хранятся в документе. Пока пользователь, удерживая левую кнопку, двигает мышь, функция добавляет линии к текущему сегменту и одновременно рисует их. Как только пользователь отпускает левую кнопку мыши, текущий сегмент передается в документ, используя управляющий процессор (часть архитектуры Документ/Вид), который отвечает за реализацию операций «отменить» и «повторить». В обработчике `OnPaint` этого приложения (не приводится) рисуются все сегменты линий документа. За полной реализацией действий «отменить» и «повторить», обратитесь к Главе 19 «Работа с архитектурой Документ/Вид».

Более полное приложение захватывало бы мышь по нажатию левой кнопки и

освобождала ее после отпускания кнопки, чтобы при выходе курсора за пределы окна, оно бы по-прежнему получало события.

### 6.1.2 Обработка событий от колесика мыши

При обработке событий от колесика мыши, вы можете получить положительную или отрицательную величину вращения, используя функцию `GetWheelRotation`. Деление данного значения на величину, возвращаемую функцией `GetWheelDelta`, даст количество строк на которое произошла прокрутка. Большинство устройств порождают одно событие на изменение положения колесика, но некоторые могут делать это чаще, поэтому может понадобиться суммировать значения поворота и совершать необходимые действия, когда сумма достигнет величины одной строки. Не исключено, что вы захотите проматывать и на не целую часть строки. В любом случае следует использовать значение, возвращаемое функцией `GetLinesPerAction`, так как оно устанавливается пользователем в «Панели управления», и умножать него полученный результат, чтобы проматывать на необходимое количество строк.

В некоторых ситуациях мышь может быть настроена проматывать страницу за раз. В этом случае вам нужно вызвать `IsPageScroll` и если она вернет значение `true`, то проматывать страницу.

Ниже показана реализация обработки колесика прокрутки в классе `wxScrolledWindow`. В переменной `m_wheelRotation` суммируется вращение, и действие происходит только если количество строк ненулевое.

```
void wxScrollHelper::HandleOnMouseWheel(wxMouseEvent& event)
{
  m_wheelRotation += event.GetWheelRotation();
  int lines = m_wheelRotation / event.GetWheelDelta();
  m_wheelRotation -= lines * event.GetWheelDelta();

  if (lines != 0)
  {
    wxScrollWinEvent newEvent;

    newEvent.SetPosition(0);
    newEvent.SetOrientation(wxVERTICAL);
    newEvent.m_eventObject = m_win;

    if (event.IsPageScroll())
    {
      if (lines > 0)
          newEvent.m_eventType = wxEVT_SCROLLWIN_PAGEUP;
      else
          newEvent.m_eventType = wxEVT_SCROLLWIN_PAGEDOWN;

      m_win->GetEventHandler()->ProcessEvent(newEvent);
    }
    else
```

```
    {
      lines *= event.GetLinesPerAction();
      if (lines > 0)
          newEvent.m_eventType = wxEVT_SCROLLWIN_LINEUP;
      else
          newEvent.m_eventType = wxEVT_SCROLLWIN_LINEDOWN;

      int times = abs(lines);
      for (; times > 0; times)
          m_win->GetEventHandler()->ProcessEvent(newEvent);
    }
  }
}
```

## 6.2 Обработка событий от клавиатуры

События от клавиатуры представлены классом `wxKeyEvent`. В библиотеке wxWidgets существуют три типа сообщений: нажатие клавиши (key down), отпускание клавиши и набор символа. Нажатие и отпускание - не преобразуемые события, в то время как набор символа — преобразуемое. Если нажать на клавишу то, как правило, будет производиться много событий нажатия и лишь одно — отпускания, так что неверно полагать что каждому событию нажатия соответствует одно событие отпускания.

Чтобы получать события от клавиатуры ваше окно должно находиться в текстовом фокусе, который вы можете принудительно получить, вызвав метод `wxWindow::SetFocus`, например, по нажатию мыши.

Таблица 6.2 содержит три макроса для обработки событий от клавиатуры.

Таблица 6.2: Макросы событий клавиатуры

| | |
|---|---|
| EVT_KEY_DOWN(func) | Обрабатывает событие wxEVT_KEY_DOWN (не преобразуемое нажатие клавиши). |
| EVT_KEY_UP(func) | Обрабатывает событие wxEVT_KEY_UP (не преобразуемое отпускание клавиши). |
| EVT_CHAR(func) | Обрабатывает событие wxEVT_CHAR (преобразуемый набор символа). |

Ниже представлены главные функции класса `wxKeyEvent`, которые вы можете использовать в вашей функции-обработчике событий клавиатуры.

Для получения кода клавиши вызывайте функцию `GetKeyCode` (в Unicode-сборках, можно использовать `GetUnicodeKeyCode`). Все действительные коды клавиш указаны в таблице 6.3.

Таблица 6.3: Идентификаторы клавиш

| | |
|---|---|
| WXK_BACK | WXK_RIGHT |
| WXK_TAB | WXK_DOWN |

Таблица 6.3: Идентификаторы клавиш (продолжение)

| | |
|---|---|
| WXK_RETURN | WXK_SELECT |
| WXK_ESCAPE | WXK_PRINT |
| WXK_SPACE | WXK_EXECUTE |
| WXK_DELETE | WXK_SNAPSHOT |
| | WXK_INSERT |
| WXK_START | WXK_HELP |
| WXK_LBUTTON | |
| WXK_RBUTTON | WXK_NUMPAD0 |
| WXK_CANCEL | WXK_NUMPAD1 |
| WXK_MBUTTON | WXK_NUMPAD2 |
| WXK_CLEAR | WXK_NUMPAD3 |
| WXK_SHIFT | WXK_NUMPAD4 |
| WXK_CONTROL | WXK_NUMPAD5 |
| WXK_MENU | WXK_NUMPAD6 |
| WXK_PAUSE | WXK_NUMPAD7 |
| WXK_CAPITAL | WXK_NUMPAD8 |
| WXK_PRIOR | WXK_NUMPAD9 |
| WXK_NEXT | |
| WXK_END | WXK_MULTIPLY |
| WXK_HOME | WXK_ADD |
| WXK_LEFT | WXK_SEPARATOR |
| WXK_UP | WXK_SUBTRACT |
| WXK_NUMPAD_DECIMAL | WXK_PAGEDOWN |
| WXK_NUMPAD_DIVIDE | |
| | WXK_NUMPAD_SPACE |
| WXK_F1 | WXK_NUMPAD_TAB |
| WXK_F2 | WXK_NUMPAD_ENTER |
| WXK_F3 | WXK_NUMPAD_F1 |
| WXK_F4 | WXK_NUMPAD_F2 |
| WXK_F5 | WXK_NUMPAD_F3 |
| WXK_F6 | WXK_NUMPAD_F4 |
| WXK_F7 | WXK_NUMPAD_HOME |
| WXK_F8 | WXK_NUMPAD_LEFT |
| WXK_F9 | WXK_NUMPAD_UP |
| WXK_F10 | WXK_NUMPAD_RIGHT |
| WXK_F11 | WXK_NUMPAD_DOWN |
| WXK_F12 | WXK_NUMPAD_PRIOR |
| WXK_F13 | WXK_NUMPAD_PAGEUP |
| WXK_F14 | WXK_NUMPAD_NEXT |
| WXK_F15 | WXK_NUMPAD_PAGEDOWN |
| WXK_F16 | WXK_NUMPAD_END |
| WXK_F17 | WXK_NUMPAD_BEGIN |
| WXK_F18 | WXK_NUMPAD_INSERT |
| WXK_F19 | WXK_NUMPAD_DELETE |
| WXK_F20 | WXK_NUMPAD_EQUAL |

Таблица 6.3: Идентификаторы клавиш (продолжение)

| | |
|---|---|
| WXK_F21 | WXK_NUMPAD_MULTIPLY |
| WXK_F22 | WXK_NUMPAD_ADD |
| WXK_F23 | WXK_NUMPAD_SEPARATOR |
| WXK_F24 | WXK_NUMPAD_SUBTRACT |
| WXK_NUMLOCK | WXK_NUMPAD_DEMICAL |
| WXK_SCROLL | WXK_NUMPAD_DEVIDE |
| WXK_PAGEUP | |

Для проверки, была ли нажата клавиша-модификатор во время образования события, используйте функции `AltDown`, `MetaDown`, `ControlDown` или `ShiftDown`. Функция `HasModifiers` возвращает значение `true`, если Control или Alt были нажаты во время генерации сообщения нажатия/отпускания (но не Shift или Meta).

Вместо использования функций `ControlDown` или `MetaDown` лучше использовать более высокоуровевую функцию `CmdDown`, которая вызывает `MetaDown` на Mac OS X и `ControlDown` на прочих платформах. См. также «Разновидности клавиш-модификаторов» в следующем разделе для более подробных объяснений.

Функция `GetPosition` возвращает позицию указателя мыши в координатах клиентской области во время получения соответствующих событий.

Совет: если событие нажатия клавиши захвачено и его обработчик не вызвал метод `event.Skip()`, то данное событие не будет передаваться следующим обработчикам. Если вы не вызываете `event.Skip()` для событий, которые не обрабатываете, то на некоторых платформах могут перестать работать горящие клавиши.

### 6.2.1 Пример обработчика события ввода символа

Ниже представлен обработчик клавиш из примера `wxThumbnailCtrl`, который вы можете найти в папке examples/chap12/thumbnail на прилагаемом CD-ROM'е.

```
BEGIN_EVENT_TABLE( wxThumbnailCtrl, wxScrolledWindow )
  EVT_CHAR(wxThumbnailCtrl::OnChar)
END_EVENT_TABLE()

void wxThumbnailCtrl::OnChar(wxKeyEvent& event)
{
  int flags = 0;
  if (event.ControlDown())
    flags |= wxTHUMBNAIL_CTRL_DOWN;
  if (event.ShiftDown())
    flags |= wxTHUMBNAIL_SHIFT_DOWN;
  if (event.AltDown())
    flags |= wxTHUMBNAIL_ALT_DOWN;

  if (event.GetKeyCode() == WXK_LEFT ||
    event.GetKeyCode() == WXK_RIGHT ||
    event.GetKeyCode() == WXK_UP ||
```

```
    event.GetKeyCode() == WXK_DOWN ||
    event.GetKeyCode() == WXK_HOME ||
    event.GetKeyCode() == WXK_PAGEUP ||
    event.GetKeyCode() == WXK_PAGEDOWN ||
    event.GetKeyCode() == WXK_PRIOR ||
    event.GetKeyCode() == WXK_NEXT ||
    event.GetKeyCode() == WXK_END)
  {
    Navigate(event.GetKeyCode(), flags);
  }
  else if (event.GetKeyCode() == WXK_RETURN)
  {
    wxThumbnailEvent cmdEvent(
      wxEVT_COMMAND_THUMBNAIL_RETURN,
      GetId());
    cmdEvent.SetEventObject(this);
    cmdEvent.SetFlags(flags);
    GetEventHandler()->ProcessEvent(cmdEvent);
  }
  else
    event.Skip();
}
```

Для ясности обработчик клавиш навигации выделен в отдельную функцию `Navigate`. Нажатие клавиш возврата и ввода генерирует высокоуровневое управляющее событие, которое приложение использует для захвата управления. Для всех остальных нажатых клавиш вызывается функция `Skip`, чтобы позволить другим частям приложения обрабатывать не задействованные здесь события клавиатуры.

### 6.2.2 Преобразование кода клавиши

События нажатия/отпускания предоставляют не преобразуемые коды клавиш, тогда как событие ввода символа предоставляет преобразуемый код клавиши. Не преобразуемый код для алфавитно-цифровых клавиш всегда соответствует значению верхнего регистра. Для остальных клавиш, это значения вида `WXK_XXX` из таблицы кодов символов. Преобразуемая клавиша, в основном является символом, появление которого ожидает пользователь как результат комбинации клавиш при вводе в текстовое поле.

Вот несколько поясняющих примеров. Когда клавиша «A» нажата, код события нажатия равен коду «A» из таблицы ASCII (65), но код события ввода символа - ASCII «a» (97). Притом, если вы нажали «A» с Shift, код клавиши в событии нажатия по-прежнему будет «A», в то время как код клавиши события ввода символа теперь тоже станет «A».

В это простом случае ясно, что в обработчике события нажатия ASCII-код может быть получен путем проверки как не преобразуемого кода клавиши, так и значения, возвращенного функцией `ShiftDown`. Но в общем случае, если вам нужен ASCII-код клавиш, вы должны использовать событие ввода символа (с `EVT_CHAR`), так как для

не алфавитно-цифровых клавиш преобразование зависит от раскладки клавиатуры и может быть выполнена правильно самой системой.

Другой вид преобразования происходит когда зажат Control, например, Ctrl+A. Событие нажатия передает тот же код «A», но событие ввода символа будет иметь код «1». Это ASCII-значение этой комбинации клавиш.

Вы можете исследовать работу других клавиш у себя в системе, запустив пример samples/keyboard и понажимав разные клавиши.

### 6.2.3 Разновидности клавиш-модификаторов

В ОС Windows клавиши-модификаторы это Control и Alt, а также особая win-клавиша, работающая как Meta. В ОС UNIX, клавиша, работающая как Meta настраивается (запустите утилиту xmodmap, чтобы посмотреть, как настроена ваша система). Иногда NumLock настроена как Meta, и из за этого функция `HasModifiers` не возвращают значение `true` при удержанной клавише Meta, так как при этом нормально обрабатываются нажатия клавиш с включенным NumLock.

В Mac OS X клавиша Command (с символом apple) выполняет функции Meta, а Option — функции Alt.

Эти различия показаны в таблице 6.4, где первая колонка — название модификаторов в wxWidgets, остальные три — клавиши, используемые для этих модификаторов на основных платформах. Клавиши Mac проиллюстрированы для наглядности.

Таблица 6.4: Клавиши-модификаторы для Windows, UNIX и Mac OS X

| Модификатор | Клавиша в Windows | Клавиша в UNIX | Клавиша в Mac OS X |
|---|---|---|---|
| Shift | Shift | Shift | Shift |
| Control | Control | Control | Control |
| Alt | Alt | Alt | Option |
| Meta | Windows | (Настраивается) | Command |

Так как в Mac OS X клавиша Command используется для тех же целей, что и Control на других платформах, то можно использовать функцию `CmdDown` объекта `wxKeyEvent` вместо `ControlDown` или `MetaDown` для перехвата этих команд на разных платформах.

Следует отметить, что также можно передать код клавиши функции `wxGetKeyState`, чтобы проверить была ли нажата клавиша. Последняя возможность особенно полезна в обработчике событий, пришедших не от клавиатуры.

### 6.2.4 Акселераторы

Акселераторы — это «горячие» клавиши для различных команд меню, которые позволяют пользователю вызывать команды очень быстро. Такие горячие клавиши имеют преимущество в обработке перед другими обработчиками клавиатуры, такими как `EVT_CHAR`. Стандартный набор таких акселераторов включает в себя Ctrl+O (чтобы открыть файл) и Ctrl+V (чтобы вставить данные в приложение). Самый легкий способ реализовать акселераторы — определить их непосредственно в пунктах

меню. Например:

```
menu->Append(wxID_COPY, wxT("Copy\tCtrl+C"));
```

wxWidgets интерпретирует текст после символа табуляции как акселератор и добавляют его в таблицу акселераторов меню. В этом примере при нажатии пользователем Ctrl+C посылается команда `wxID_COPY`, как будто был выбран соответствующий пункт меню.

Вы можете использовать Control, Alt или Shift в различных сочетаниях, дополняя символом или функциональной клавишей, следующими за «+» или «-». Вот примеры правильных сочетаний клавиш-акселераторов: Ctrl+B, G, Shift-Alt-K, F9, Ctrl+F3, Esc и Del. Вы можете использовать следущие зарезервированные имена клавиш: Del, Back, Ins, Insert, Enter, Return, PgUp, PgDn, Left, Right, Up, Down, Home, End, Space, Tab, Esc и Escape. Регистр в именах клавиш не важен (любые сочетания верхнего и нижнего регистра будут работать).

Напоминаем, что в Mac OS X описание горячих клавиш, использующих Ctrl, будет в действительности использовать клавишу Command.

Другой способ включить акселераторы — заполнить таблицу `wxAcceleratorTable` объектами `wxAcceleratorEntry` и связать ее с окном, используя метод `wxWindow::SetAcceleratorTable`. Каждый объект `wxAcceleratorEntry` инициализируется битовым списком модификаторов (один или несколько `wxACCEL_ALT`, `wxACCEL_CTRL`, `wxACCEL_SHIFT` и `wxACCEL_NORMAL`), кодом клавиши (см. таблицу 6.3) и идентификатором. Например,

```
wxAcceleratorEntry entries[4];
entries[0].Set(wxACCEL_CTRL,  (int) 'N',     wxID_NEW);
entries[1].Set(wxACCEL_CTRL,  (int) 'X',     wxID_EXIT);
entries[2].Set(wxACCEL_SHIFT, (int) 'A',     wxID_ABOUT);
entries[3].Set(wxACCEL_NORMAL, WXK_DELETE,   wxID_CUT);

wxAcceleratorTable accel(4, entries);
frame->SetAcceleratorTable(accel);
```

Можно использовать несколько таблиц акселераторов в оконной иерархии, а также сочетать наборы акселераторов с определенной таблицей `wxAcceleratorTable`. Это полезно, если у вас есть альтернативные акселераторы для отдельной команды, которые вы не можете целиком вписать в текст пункта меню.

## 6.3 Обработка событий от джойстика

Класс `wxJoystick` дает вашему приложению возможность получать информацию от одного или двух джойстиков в ОС Windows или Linux. Как правило, вы создаете объект `wxJoystick`, передавая `wxJOYSTICK1` или `wxJOYSTICK2` и храните объект в ОЗУ, пока он нужен. Когда вам нужно получить с него данные, пошлите функцию `SetCapture`, передав указатель на окно, которое будет получать события джойстика, а затем вызовите `ReleaseCapture`, если вам больше не нужны события. Вы можете

поставить захват на все время работы приложения вызвав `SetCapture` при инициализации и `ReleaseCapture` при выходе.

Для более детального описания событий и методов рассмотрим пример samples/joystick из дистрибутива wxWidgets. Управляя джойстиком пользователь может рисовать последовательности линий на холсте, нажимая любую из кнопок джойстика. При нажатии также воспроизводится звук.

Ниже представлен фрагмент кода инициализации. Сначала приложение, создавая временный объект джойстика, проверяет, установлен ли джойстик и завершается, если нет. Звуковой файл buttonpress.wav загружается в объект `wxSound`, хранящийся в объекте приложения. Минимальная и максимальная позиции джойстика также сохраняются, чтобы линии помещались в окно.

```
#include "wx/wx.h"
#include "wx/sound.h"
#include "wx/joystick.h"

bool MyApp::OnInit()
{
  wxJoystick stick(wxJOYSTICK1);
  if (!stick.IsOk())
  {
    wxMessageBox(wxT("No joystick detected!"));
    return false;
  }

  m_fire.Create(wxT("buttonpress.wav"));
  m_minX = stick.GetXMin();
  m_minY = stick.GetYMin();
  m_maxX = stick.GetXMax();
  m_maxY = stick.GetYMax();

  // Создаем главный фрейм окна
  ...
  return true;
}
```

`MyCanvas` — это окно, которое хранит объект джойстика, а также получает от него события. Вот реализация `MyCanvas`:

```
BEGIN_EVENT_TABLE(MyCanvas, wxScrolledWindow)
  EVT_JOYSTICK_EVENTS(MyCanvas::OnJoystickEvent)
END_EVENT_TABLE()

MyCanvas::MyCanvas(wxWindow *parent, const wxPoint& pos,
  const wxSize& size):
  wxScrolledWindow(parent, wxID_ANY, pos, size, wxSUNKEN_BORDER)
{
  m_stick = new wxJoystick(wxJOYSTICK1);
```

```
  m_stick->SetCapture(this, 10);
}

MyCanvas::~MyCanvas()
{
  m_stick->ReleaseCapture();
  delete m_stick;
}

void MyCanvas::OnJoystickEvent(wxJoystickEvent& event)
{
  static long xpos = -1;
  static long ypos = -1;

  wxClientDC dc(this);

  wxPoint pt(event.GetPosition());

  // Если возможны отрицательные координаты, то делаем, чтобы минимум был 0
  int xmin = wxGetApp().m_minX;
  int xmax = wxGetApp().m_maxX;
  int ymin = wxGetApp().m_minY;
  int ymax = wxGetApp().m_maxY;

  if (xmin < 0) {
    xmax += abs(xmin);
    pt.x += abs(xmin);
  }

  if (ymin < 0) {
    ymax += abs(ymin);
    pt.y += abs(ymin);
  }

  // Масштабируем по размеру холста
  int cw, ch;
  GetSize(&cw, &ch);

  pt.x = (long) (((double)pt.x/(double)xmax) * cw);
  pt.y = (long) (((double)pt.y/(double)ymax) * ch);

  if (xpos > -1 && ypos > -1 && event.IsMove() && event.ButtonIsDown())
  {
    dc.SetPen(*wxBLACK_PEN);
    dc.DrawLine(xpos, ypos, pt.x, pt.y);
  }
```

```
  xpos = pt.x;
  ypos = pt.y;

  wxString buf;
  if (event.ButtonDown())
    buf.Printf(wxT("Joystick (%d, %d) Fire!"), pt.x, pt.y);
  else
    buf.Printf(wxT("Joystick (%d, %d)"), pt.x, pt.y);

  frame->SetStatusText(buf);

  if (event.ButtonDown() && wxGetApp().m_fire.IsOk())
  {
    wxGetApp().m_fire.Play();
  }
}
```

### 6.3.1 События wxJoystick

Класс `wxJoystick` создает события типа `wxJoystickEvent`. Соответствующие им макросы представлены в таблице 6.5. Каждый макрос принимает один аргумент — функцию обработки события.

Таблица 6.5: Макросы событий джойстика

| | |
|---|---|
| EVT_JOY_BUTTON_DOWN(func) | Обрабатывает событие wxEVT_JOY_BUTTON_DOWN, генерируемое при нажатии кнопки. |
| EVT_JOY_BUTTON_UP(func) | Обрабатывает событие wxEVT_JOY_BUTTON_UP, генерируемое при отпускании кнопки. |
| EVT_JOY_MOVE(func) | Обрабатывает событие wxEVT_JOY_MOVE, генерируемое при движении джойстика в плоскости XY. |
| EVT_JOY_ZMOVE(func) | Обрабатывает событие wxEVT_JOY_ZMOVE, генерируемое при движении джойстика по оси Z. |
| EVT_JOYSTICK_EVENTS(func) | Обрабатывает все события джойстика. |

### 6.3.2 Методы класса wxJoystickEvent

Ниже представлено описание функции класса `wxJoystickEvent`, которые вы можете вызывать для получения дополнительной информации о событии. Как обычно, вы можете вызвать функцию `GetEventType`, чтобы получить тип события, что полезно при использовании макроса `EVT_JOYSTICK_EVENTS`, который отлавливает все события от джойстика.

Вызов ButtonDown позволяет проверить, было ли событие вызвано нажатием

кнопки. Вы можете дополнительно передать идентификатор кнопки `wxJOY_BUTTONn` (где n = 1, 2, 3 или 4) чтобы определить, какая конкретно кнопка была нажата, или `wxJOY_BUTTON_ANY`, если вам это не важно. Метод `ButtonUp` работает также, но проверяет событие отпускание соответствующей кнопки. Метод `IsButton` равносилен применению конструкции `ButtonDown() || ButtonUp()`.

Чтобы проверить, была ли кнопка нажата во время генерации события (если, например, событие представляет не само нажатие кнопки), вызывите метод `ButtonIsDown` с теми же аргументами, что и `ButtonDown`. В противном случае используйте `GetButtonState` (с теми же аргументами), чтобы получить битовый список идентификаторов `wxJOY_BUTTONn`.

Вызовете метод `IsMove` чтобы проверить, было ли событие движением в плоскости XY и `IsZMove` — по оси Z.

Функция `GetPosition` возвращает объект `wxPoint` с текущей позицией в плоскости XY, а `GetZPosition` возвращает целое, представляющее собой Z-координату, если таковая поддреживается.

Наконец, вы можете определить, какой джойстик послужил причиной события (`wxJOYSTICK1` или `wxJOYSTICK2`) вызвав `GetJoystick`.

### 6.3.3 Методы класса wxJoystick

Мы не хотим приводить здесь все методы джойстика полностью, для этого вы можете обратиться к справочной информации по классу `wxJoystick`, но укажем наиболее интересные.

Как было показано в примере, функцию `SetCapture` необходимо вызывать для прямого получения данных с джойстика в определенное окно, и, соответственно, `ReleaseCapture` для освобождения и использования его другими приложениями. В случае, если джойстик занят другим приложением или неисправен, вызывайте `IsOK` перед попыткой захвата. Вы также можете определить характеристики джойстика при помощи функций `GetNumberButtons`, `GetNumberJoysticks`, `GetNumberAxes`, `HasRudder` и прочих.

Вы можете получить состояние джойстика извне обработчика события с такими фунциями как `GetPosition` и `GetButtonState`.

Вашему приложению почти всегда потребуется вызвать `GetXMin`, `GetXMax` и подобные функции для определения диапазона, поддерживаемого джойстиком.

## 6.4 Заключение

В этой главе вы узнали о получении данных с мыши, клавиатуры и джойстика. Теперь вы сможете добавить взаимодействие с устройствами ввода в ваше приложение. Для углубления степени понимания рекомендуем изучить примеры samples/keyboard, samples/joytest и samples/dragimag, а также класс `wxThumbnailCtrl` в examples/chap12 на прилагаемом CD-ROM.

Следующая глава расскажет, как можно добиться гибкой, переносимой, легко переводимой, и, прежде всего, привлекательной компоновки элементов окна с помощью нашего гибкого друга — сайзера (sizer).