

```

SetBkMode(hdc, TRANSPARENT); //Прозрачный фон
return CreateSolidBrush( RGB(180,180,255) ); //Покрасим список
//светло-синим
}
return FALSE; //Остальные элементы не обрабатываем
}

```

Воспользовавшись дескриптором контекста устройства `hdc`, поступающим в функции `DlgOnColorxxx`, мы смогли назначить для каждого списка в отдельности цвет его строк и установить режим прозрачного фона (иначе строки изображались бы на белом фоне). Чтобы покрасить списки по-разному, в программе выполняется анализ параметра `hwndCtl` – дескриптора конкретного элемента. Разумеется, если оба списка требуется покрасить одинаково, в анализе этого дескриптора нет необходимости, и функция перекрашивания значительно упростится:

```


HBRUSH DlgOnColorListBox( HWND, HDC hdc, HWND, int ) {
    SetTextColor( hdc, RGB(0,128,0) ); //Покрасим текст зеленым
    SetBkMode( hdc, TRANSPARENT ); //Прозрачный фон
    return CreateSolidBrush( RGB(180,255,180) ); //Покрасим список светло-зеленым
}

```

```

/*Нарисуем оси координат*/
MoveToEx(hdc,0,70,NULL);
LineTo(hdc,180,70); //Ось X
MoveToEx(hdc,100,0,NULL);
LineTo(hdc,100,150); //Ось Y
/*Нарисуем первую эллипсиду*/
for(float t=0;t<6.28;t+=0.01){
    int x=(2*cos(t)-3*cos(2*t))*15;
    int y=-((2*sin(t)-3*sin(2*t)))*15;
    Rectangle(hdc,x+100,70-y,x+102,72-y); //Рисуем точки квадратами
}
/*Нарисуем вторую эллипсиду*/
SelectPen(hdc,hPen);
for(float t=0;t<6.28;t+=0.01){
    int x=(2*cos(t)-1.5*cos(2*t))*15;
    int y=-((2*sin(t)-1.5*sin(2*t)))*15;
    Rectangle(hdc,x+100,70-y,x+102,72-y); //Рисуем точки квадратами
}
EndPaint(hwnd,&ps);
}

```

В файле ресурсов описывается простое диалоговое окно без каких-либо элементов управления (кроме заголовка и кнопки  системного меню, которые в данном случае включаются в состав диалога по умолчанию). Диалог создается, как обычно, в главной функции WinMain(), состоящей, фактически, из единственного содержательного предложения.

В оконной функции диалогового окна предусмотрена обработка трех сообщений: WM_INITDIALOG, WM_COMMAND и WM_PAINT. В функцииDlgOnInitDialog() создается красное перо для вывода второго графика (первый будет нарисован пером по умолчанию, т. е. черным). В функцииDlgOnCommand() осуществляется закрытие диалогового окна (и, соответственно, всего приложения) при щелчке по кнопке с крестиком.

Графики эллипсоид рисуются в функцииDlgOnPaint(), причем процедура вывода изображений ничем не отличается от используемой для обычных окон. В результате выполнения функцииBeginPaint() программа получает доступ к контексту устройства hdc. С помощью функцийMoveToEx() иLineTo() в окно выводятся оси координат, смещенные относительно начала окна на 100 пикселей по горизонтали и 70 пикселей по вертикали. Линии осей координат рисуются пером по умолчанию – черным, толщиной 1 пиксел. Далее в цикле вычисляются последовательные значения пар координат x и y и для каждой пары координат в окно выводится квадратик размером 2 пиксела. Поскольку при заданных значениях параметров a и b (2 и 1.5) значения x и y оказываются близкими к 1, обе координаты умножаются на 15, увеличивая тем самым размер выводимого изображения. При выводе в окно (диалога или обычное) графиков следует иметь в виду, что начало отсчета в окнах Windows располагается в верхнем левом углу, поэтому графики надо перевертывать, т. е. умножать их ординаты на -1 , и смещать вниз (если в них есть положительные значения). В нашем примере точки (точнее, квадратики) графиков выводятся со смещением 100 пикселей по горизонтали и 70 пикселей по вертикали, что позиционирует график правильно относительно осей координат. Значения ординат y умножаются на -1 , хотя в данном случае это излишне, так как график эллипсоид симметричен относительно оси абсцисс. После вывода первой эллипсоиды в контекст устройства выбирается красное перо и поверх первого рисуется второй график. Завершающей операцией, как и всегда, является вызов функцииEndPaint(), возвращающей Windows контекст устройства.

относительная простота программы, недостатком – меньшие возможности. Рассмотрим программу, в которой в диалоговое окно, являющееся главным окном приложения, выводится график некоторой математической функции, например, эпициклоиды, уравнение которой в параметрической форме имеет вид:

$$x = acost - b\cos 2t$$

$$y = asint - b\sin 2t$$

Для демонстрации графических возможностей диалога выведем в него две эпициклоиды при различных значениях параметров a и b , нарисовав их перьями разного цвета (рис. 6.37).



Рис. 6.37. График функции в диалоговом окне

*/*Пример 6-8. График функции в модальном диалоге*/*

*/*Файл 6-8.РС*/*

```
#include "6-8.h" Dlg DIALOG 20, 20, 90, 70
CAPTION "Эпициклоиды" {
}
```

*/*Файл 6-8.СРР*/*

```
#include <windows.h>
#include <windowsx.h>
#include <math.h>
#include "6-8.h"
HPEN hPen;
```

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int) {
    DialogBox(hInst, "Dlg", NULL, DlgProc);
    return 0;
}
```

```
BOOL CALLBACK DlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    switch(msg) {
        HANDLE_MSG(hwnd, WM_INITDIALOG, DlgOnInitDialog);
        HANDLE_MSG(hwnd, WM_COMMAND, DlgOnCommand);
        HANDLE_MSG(hwnd, WM_PAINT, DlgOnPaint);
        default:
            return FALSE;
    }
}
```

```
BOOL DlgOnInitDialog(HWND hwnd, HWND, LPARAM) {
    hPen = CreatePen(PS_SOLID, 1, RGB(255, 100, 100));
    return TRUE;
}
```

```
void DlgOnCommand(HWND hwnd, int id, HWND, UINT) {
    if(id == IDCANCEL)
        EndDialog(hwnd, 0);
}
```

```
void DlgOnPaint(HWND hwnd) {
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hwnd, &ps);
```

```
InvalidateRect(GetParent(hwnd), NULL, TRUE); // Иницилируем WM_PAINT
```

Однако можно было ввести, как и для дескриптора экземпляра приложения, глобальную переменную типа `HWND`. Сохранив в ней значение дескриптора главного окна после его создания вызовом функции `CreateWindow()`, далее можно было пользоваться этим сохраненным значением.

Наконец, есть и третий способ. Дескриптор родительского окна хранится в той же дополнительной области окна со смещением `GWL_HWNDPARENT`. Таким образом, получить его в любой точке программы можно с помощью той же функции `GetWindowLong()`:

```
InvalidateRect((HWND)GetWindowLong(hwnd, GWL_HWNDPARENT), NULL, TRUE);
```

Здесь результат, полученный от функции `GetWindowLong()`, приводится к типу `HWND`.

Режимы вывода графика

Вывод графика осуществляется в функции `OnPaint()` обработки сообщений `WM_PAINT` при условии, что из файла прочитаны данные (переменная `bDataOK = TRUE`). Вывод осуществляется в том режиме, который установлен с помощью немодального диалога. Если `Mode=DOTS`, выводится точечный график так же, как и в предыдущем примере. Если же `Mode=CURVE`, то прежде всего с помощью функции `MoveToEx()` выполняется перемещение текущей графической позиции в первую точку графика. По умолчанию текущая позиция устанавливается в точке с координатами (0,0), т. е. в верхнем левом углу экрана. Если ее не сместить в первую точку графика, то график всегда будет начинаться линией, идущей из верхнего левого угла. После этого в цикле из оставшихся 499 шагов функцией `LineTo()` проводятся прямые линии от текущей позиции к следующей точке графика. Поскольку функция `LineTo()` не только рисует линию, но и смещает при этом текущую позицию, не возникает необходимости в циклическом использовании функции `MoveToEx()`.

Графика диалогового окна

Диалоговые окна широко используются для управления ходом выполнения программы или режимами ее работы. В этом случае им обычно придают стандартный вид, не забывая особо о графическом оформлении диалогового окна. Однако в действительности возможности диалоговых окон гораздо шире. В диалоговые окна можно выводить графики и рисунки; пользуясь им, например, как средством наблюдения результатов измерений или расчетов; угрюмую серую поверхность диалогового окна можно покрасить в любой привлекательный цвет; поясняющие надписи диалогового окна также не обязательно должны быть черными. Возможности графического оформления относятся не только к самому диалоговому окну, но также и к отдельным элементам управления: спискам, кнопкам и проч. Рассмотрим некоторые из этих возможностей.

Вывод в диалоговое окно графиков

В диалоговые окна можно выводить любую графическую информацию, как и в обычное окно. При этом начальные действия, такие, как создание кистей или перьев, следует выполнять в ответ на сообщение `WM_INITDIALOG`, а собственно вывод в диалоговое окно – в ответ на сообщение `WM_PAINT`. Как и в случае обычных окон, Windows посылает это сообщение в диалоговое окно всякий раз, когда оно появляется из-под закрывающих его окон и, следовательно, требует перерисовки. Преимуществом использования модального диалога для отображения графической информации является

После этого вызовом функции `InvalidateRect()` инициируется посылка в приложение сообщения `WM_PAINT` и перерисовывание графика.

Очевидно, что перерисовывать график надо также при смене режима его вывода. Поэтому функция `InvalidateRect()` вызывается и при поступлении сообщений от кнопок смены режима (с идентификаторами `ID_DOTS` и `ID_CURVE`).

Определение значений дескрипторов

Многие функции Windows требуют указания в качестве одного из параметров (обычно первого) того или иного дескриптора. Так, для функции создания немодального диалога `CreateDialog()` первым параметром должен выступать дескриптор данного экземпляра приложения. Создание диалога обычно осуществляется в ответ на выбор той или иной команды меню, т. е. в функции обработки сообщения `WM_COMMAND`. Однако дескриптор приложения отсутствует среди аргументов оконной функции и, следовательно, автоматически в функции обработки сообщений не передается. Получить его можно двумя способами. Один из них заключается в объявлении в программе глобальной переменной и помещении в нее значения требуемого дескриптора в той точке программы, где он оказывается известен. В настоящей программе для этого использовалась глобальная переменная `hI`. Значение дескриптора экземпляра приложения помещалось в нее в самом начале функции `WinMain()`, перед началом действий по регистрации класса окна.

Другой метод связан с использованием функций Windows, предназначенных специально для получения тех или иных дескрипторов. Естественно, все действующие дескрипторы хранятся в Windows, и существуют функции для их извлечения. Конкретно дескриптор экземпляра приложения хранится в специальной области памяти, связанной с любым окном (так называемая дополнительная память окна). Получить информацию из этой области памяти в 32-разрядном приложении можно с помощью функции `GetWindowLong()`, в качестве первого параметра которой указывается дескриптор окна, а в качестве второго – символическое обозначение требуемого смещения в дополнительной памяти. Для дескриптора экземпляра приложения это смещение равно `GWL_HINSTANCE`.

Таким образом, для создания диалога можно было поступить так, как сделано в программе, т. е. сохранить в переменной `hI` дескриптор в какой-либо точке функции `WinMain()`, а при создании диалога воспользоваться этой переменной:

```
hSettingsBox=CreateDialog(hI, "SettingsDlg", hwnd, DlgProc);
```

но можно было обойтись без глобальной переменной, а значение дескриптора приложения получить из дополнительной области окна в тот момент, когда он оказывается нужен:

```
hSettingsBox=CreateDialog((HINSTANCE)GetWindowLong(hwnd, GWL_HINSTANCE),  
                           "SettingsDlg", hwnd, DlgProc);
```

Приведение результата выполнения функции `GetWindowLong()` к типу `HINSTANCE` здесь необходимо, так как эта функция для любого смещения возвращает “безликий” результат типа `LONG`, а для функции `CreateDialog()` требуется параметр типа `HINSTANCE`.

Схожая ситуация имеет место и для дескрипторов окон. Правда, в функциях обработки сообщений *главного* окна проблем с этим дескриптором не возникает, так как он передается во все эти функции в качестве одного из параметров, а вот в порожденных окнах, в частности в диалогах, он неизвестен. В нашем случае дескриптор главного окна нужен для функции `InvalidateRect()`, которая вызывается в оконной функции диалога, но должна организовать перерисовку главного окна с графиком.

Для получения дескриптора главного окна, которое для диалога является родительским, мы воспользовались функцией Windows `GetParent()`:

переменной Mode и, вообще говоря, не связан с внешним видом кнопок. Изменение состояния кнопок программным образом, путем отправки сообщений *в диалог*, не изменяет текущий режим. Для того чтобы изменить режим вывода графика, надо щелкнуть по кнопке мышью. В этом случае программа получит сообщение *из диалога* и, обрабатывая это сообщение, может, в частности, изменить режим.

Схожая ситуация имеет место с альтернативностью кнопок. Вполне возможно, послать одинаковые сообщения обеим кнопкам, обе их нажать или отпустить. Программное нажатие одной кнопки не отжимает автоматически другую. Альтернативность кнопок со стилем BS_AUTORADIOBUTTON проявляется только при воздействии на них мышью. Эти воздействия перехватываются Windows, которая, помимо отправки сообщения из диалога в программу, нажимает кнопку, по которой щелкнули, и отжимает другую кнопку, придавая им свойство альтернативности.

Следующее действие по инициализации диалога заключается в заполнении пустого пока списка с идентификатором ID_SCALE значениями допустимых масштабов. Для этого в элемент управления ID_SCALE посылаются сообщения LB_ADDSTRING с указанием в качестве второго параметра адреса пересылаемой в список текстовой строки S:

```
SendDlgItemMessage(hwnd, ID_SCALE, LB_ADDSTRING, 0, (LPARAM)S); //Пересылка строки
```

Строка S предварительно формируется с помощью функции `wsprintf()`, которая преобразует числовые значения масштабов (массив `nScales[8]`) в символы с добавлением знака процента. Оба эти действия осуществляются в цикле из восьми шагов по всем элементам массива `nScales[]`.

Наконец, последнее сообщение LB_SETCURSEL посылается в список для выделения текущего масштаба изображения. Как уже отмечалось, для удобства в программе хранится не только текущее значение масштаба (в переменной `CurrentScales`), но и индекс `nScaleIndex` текущего значения в массиве масштабов. Массив масштабов, ради удобства вывода этих чисел в список, заполнен значениями масштабов в процентах; в переменной же `CurrentScales` хранится коэффициент преобразования исходных данных в виде доли единицы (естественно, в переменной с плавающей точкой). При запуске приложения обе эти переменные получают начальные значения, характеризующие масштаб 100 %.

Второе место программы, где в диалоговое окно посылается сообщение, – это функция `DlgOnCommand()`, конкретно фрагмент этой функции, обрабатывающий сообщение от списка с идентификатором ID_SCALE. Весь этот фрагмент представляет собой условный блок, выполняемый в том случае, если код уведомления, поступающий в программу через последний параметр `codeNotify` функции `DlgOnCommand()`, равен константе `LBN_SELCHANGE`. Такой код уведомления посылается при изменении пользователем номера выбранной строки списка. Другие коды уведомления посылаются, например, при получении списка фокуса ввода (при управлении списком от клавиатуры) или при потере фокуса.

Сообщение, посылаемое в программу при смене строки списка, не содержит сведений о том, какая строка выбрана. Для того чтобы получить номер новой выбранной строки, программа должна с помощью функции `SendDlgItemMessage()` послать в список сообщение-запрос LB_GETCURSEL. Результат, возвращаемый в этом случае функцией `SendDlgItemMessage()`, одновременно является номером выделенной строки списка и индексом массива масштабов `nScales[8]`.

Программа извлекает из массива масштабов соответствующее этому индексу значение масштаба, преобразует его в дробное значение ($1\% = 0.01$) и выполняет заново масштабирование массива данных `nBuf[]`, записывая новые значения в массив `nBufScaled[]`.

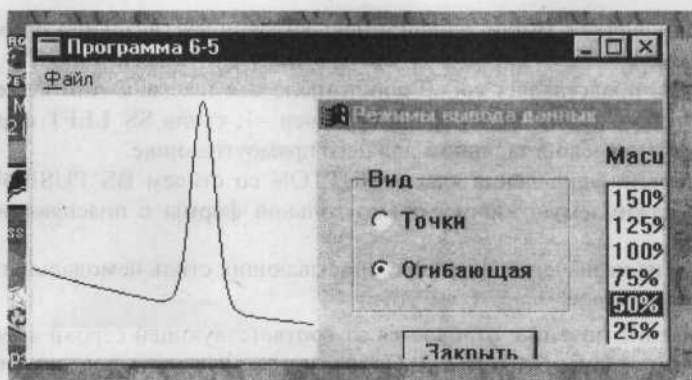


Рис. 6.36. Поведение дочернего (CHILD) окна

Взаимодействие с немодальным диалогом

Создание немодального диалога осуществляется в функции `OnCommand()` при выборе пользователем команды меню `Файл>Режимы`, за которой закреплен идентификатор `MI_SETTINGS`. Диалог создается функцией `CreateDialog()`, имеющей те же параметры, которые имеет и функция создания модального диалога `DialogBox()`, однако, в отличие от последней, возвращающей дескриптор созданного диалогового окна. Этот дескриптор (`hSettingsBox`) используется в программе в качестве флага существования диалога для защиты от его повторного создания. Перед вызовом функции `CreateDialog()` проверяется флаг `hSettingsBox`; если он не равен нулю, т.е. если диалог уже создан, функция `CreateDialog()` не выполняется.

Полезно иметь в виду, что дескрипторы окон не обнуляются системой при уничтожении окон; поэтому в функции `DlgOnCommand()` в случае прихода идентификаторов `IDOK` или `IDCANCEL` не только закрывается окно диалога функцией `Destroy Window()`, но и обнуляется его дескриптор `hSettingsBox`, что дает возможность после закрытия диалога открыть его повторно.

Использование дескриптора в качестве флага не имеет в данном случае каких-то серьезных преимуществ перед организацией специального флага кроме того, что мы экономим на одном предложении его установки.

Рассмотрим программное взаимодействие с диалогом с помощью функции `SendDlgItemMessage()`, посылающей в диалог сообщения. Большая часть вызовов этой функции приходится на функцию `DlgOnInitDialog()`, выполняющую инициализацию диалога после его создания системой. Вызов

```
SendDlgItemMessage(hwnd, Mode, BM_SETCHECK, 1, 0L); // Нажать кнопку текущего режима
```

приводит к показу в нажатом состоянии кнопки того режима ("Точки" или "Огибающая"), который сохранен в переменной `Mode`. Эта переменная перечислимого типа была объявлена таким образом, что ее допустимые значения совпадают с идентификаторами двух альтернативных кнопок переключения режима; это дало нам возможность указывать ее в качестве второго параметра функции `SendDlgItemMessage()`. Значение четвертого параметра (1) указывает на то, что кнопка должна быть показана в нажатом состоянии; указание в качестве этого параметра нуля отжало бы кнопку.

Важно отметить, что описанные манипуляции с кнопкой относятся только к ее внешнему виду; они не определяют ни текущий режим, ни состояние второй кнопки. Текущий режим, анализируемый и используемый в функции `OnPaint()`, определяется значением

чувствительны к щелчкам мыши, список имеет линейку вертикальной прокрутки и окружен простой рамкой.

Класс `STATIC` представляет собой просто поле для текста. С ним не осуществляется взаимодействия поэтому его идентификатор равен `-1`; стиль `SS_LEFT` определяет позиционирование текста влево в заданном для него прямоугольнике.

Наконец, элемент управления класса `BUTTON` со стилем `BS_PUSHBUTTON` представляет собой нажимаемую кнопку прямоугольной формы с поясняющим текстом на ней.

Остановимся на назначении констант, описывающих стиль немодального диалога:

`STYLE WS_VISIBLE | WS_SYSMENU | WS_POPUP`

Это предложение почти не отличается от соответствующей строки сценария модального диалога из программ 6-1 или 6-2. Однако для модального диалога минимально необходимый набор характеристик диалога действует по умолчанию и предложение `STYLE`, как уже упоминалось, можно было опустить. В диалоге же немодальном это предложение должно присутствовать обязательно.

Константа `WS_VISIBLE` делает диалог видимым, т. е. как бы автоматически вызывает для него функцию `Windows ShowWindow()`. Без этой константы окно диалога не появится на экране.

Константа `WS_SYSMENU` определяет наличие в строке заголовка диалога кнопки его закрытия **X**. В нашем случае эта кнопка не обязательна, так как в самом диалоге мы предусмотрели кнопку “Закрыть”, однако удобнее создавать диалоговые окна стандартного, привычного вида.

Константа `WS_POPUP` объявляет диалоговое окно “всплывающим”. Вообще порожденные окна `Windows`, т. е. окна (как диалоговые, так и обычные), созданные внутри главного окна приложения, бывают двух типов: всплывающие, описываемые константой `WS_POPUP`, и дочерние (константа `WS_CHILD`). Между ними имеется ряд различий в поведении. Всплывающие окна при их перемещении по экрану могут выходить за рамки родительского окна (рис. 6.35). Дочерние окна всегда остаются внутри породившего их родительского окна (рис. 6.36). Имеются и другие различия, в частности заголовок всплывающего окна, если его сделать активным, выделяется ярким цветом, а у дочернего окна заголовок всегда тусклый (см. те же рисунки).

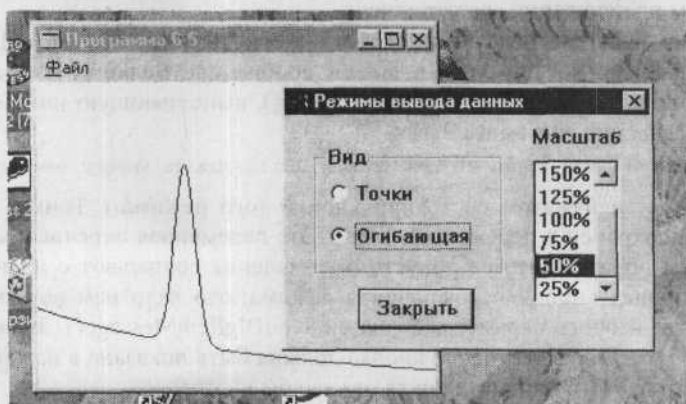


Рис. 6.35. Поведение всплывающего (`POPUP`) окна

Описание элементов управления в файле ресурсов

Модальные и немодальные диалоги практически ничем не различаются по своим возможностям и правилам описания, за исключением того, что модальный диалог полностью перехватывает управление и не позволяет до своего завершения работать с другими элементами окна, а немодальный не имеет этого ограничения. Придание диалогу свойства модальности или немодальности осуществляется вызовом соответствующей функции создания диалога – `DialogBox()` для модального диалога и `CreateDialog()` для немодального. Кроме того, по-разному осуществляется закрытие диалога: модальный диалог закрывается функцией `EndDialog()`, а немодальный – более универсальной функцией `DestroyWindow()`. Содержимое же диалогового окна, описываемое в файле ресурсов, и для того и для другого диалога может быть каким угодно. Таким образом, приводимые ниже правила описания элементов управления немодального диалогового окна для приложения 6-7 (рис. 6.34) можно с таким же успехом использовать и при разработке модального диалога.

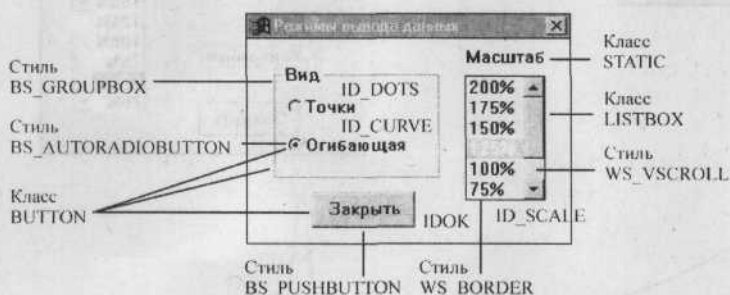


Рис. 6.34. Элементы управления в окне диалога с указанием их классов, стилей и идентификаторов

Все предложения файла ресурсов используют стандартный формат описания элементов управления. Они начинаются с ключевого слова `CONTROL`, за которым следуют: поясняющий текст, идентификатор (значение которого определено в файле 6-7.H), предопределенный класс элемента управления, стиль (точнее, комбинация необходимых стилей) и координаты.

Класс `BUTTON` с указанием стиля `BS_AUTORADIOBUTTON` описывает альтернативные кнопки. В группе таких кнопок в любой момент времени может быть нажата только одна, причем, если пользователь нажимает какую-то кнопку, она утапливается и подсвечивается, а нажатая ранее кнопка автоматически отжимается. В нашем диалоге всего две такие кнопки и они по умолчанию образуют группу. Если кнопок больше и они по смыслу должны образовывать несколько групп (например, группы "Вид", "Цвет", "Форма" и др.), в стиль первой кнопки из каждой группы надо добавить бит `WS_GROUP`.

Класс `BUTTON` с указанием стиля `BS_GROUPBOX` позволяет включить в диалоговое окно рамку с заголовком, в которую затем будут помещены альтернативные кнопки или другие элементы управления. К рамке не выполняется программных обращений, и в качестве идентификатора принимается значение `-1`, однако "заголовок" у нее есть – это надпись "Вид" в верхнем разрыве рамки.

Класс `LISTBOX` позволяет вывести на экран список некоторых данных (у нас – значений масштаба графика в процентах). Комбинация стилей `LBS_NOTIFY` | `WS_VSCROLL` | `WS_BORDER` определяет следующие свойства списка: элементы списка

```

case ID_SCALE://Сообщение от списка; проанализируем извещение
if (codeNotify==LBN_SELCHANGE){//Если изменилось выделение в списке
    nScaleIndex=(int) SendDlgItemMessage (hwnd, ID_SCALE, LB_GETCURSEL, 0, 0L);
    nCurrentScales=
        (float)nScales[nScaleIndex]/100;//Преобразование строки в число
    for (int i=0; i<500; i++){//Масштабирование спектра в соответствии
        nBufScaled[i]=nBuf[i]*nCurrentScales;//с новым масштабом
    }
    InvalidateRect (GetParent (hwnd), NULL, TRUE); //Иницилируем WM_PAINT
} //Конец if
} //Конец switch(id)
}

```

Результат работы этой программы приведен на рис. 6.33.

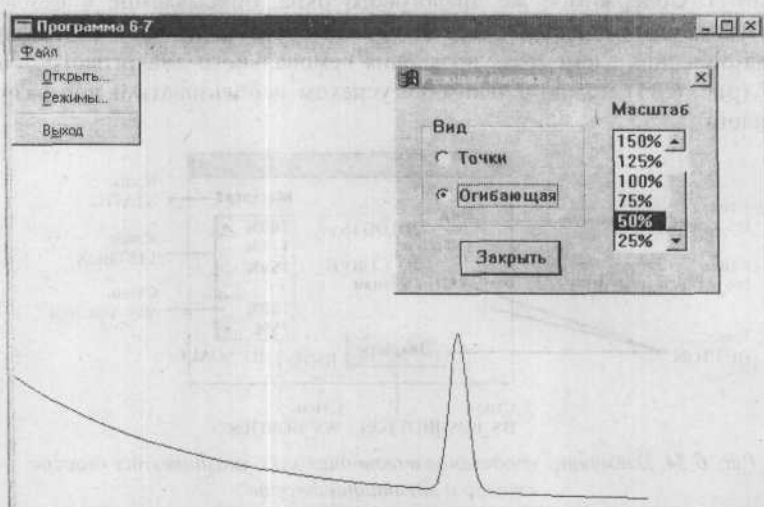


Рис. 6.33. Немодальный диалог в главном окне

Рассмотрим отличия приведенного примера от программы 6-6.

Файлы заголовков и ресурсов

Включение в приложение немодального диалогового окна с программными блоками обслуживания его элементов управления заметно усложнило программу. В заголовочный файл 6-7.H включены определения дополнительных констант – идентификаторов элементов управления диалогом. Там же определен перечислимый тип `GraphModes` (см. описание перечислимых типов в гл. 3). Переменные этого типа смогут принимать всего два значения. Одно из них называется `DOTS` и равно значению символьной константы `ID_DOTS`; другое называется `CURVE` и совпадает со значением символьной константы `ID_CURVE`. Переменная типа `GraphModes` (в программе она названа `Mode`) служит для хранения текущего режима изображения графика – в виде точек (`DOTS`) или огибающей (`CURVE`). Наконец, в файл 6-7.H добавлены прототипы прикладных функций обслуживания диалогового окна.

В файле ресурсов 6-7.RC расширен список команд меню “Файл” (появился пункт “Режимы” с идентификатором `MI_SETTINGS` для активизации немодального диалога) и задан сценарий немодального диалога `SettingsDlg`. Рассмотрим описание элементов управления диалоговым окном.

```

HDC hdc=BeginPaint(hwnd,&ps); //Получение контекста устройства
HPEN hOldPen=SelectPen(hdc,hPen);
GetClientRect(hwnd,&r); //Получение координат рабочей области
if(bDataOK){
    switch(Mode){
        case DOTS:{
            for(int i=0;i<500;i++){
                SetPixel(hdc,i,r.bottom-5-nBufScaled[i],RGB(0,0,255)); //Вывод точек
            }
            break;
        } //Конец case DOTS
        case CURVE:{
            MoveToEx(hdc,0,r.bottom-5-nBufScaled[0],NULL);
            for(int i=1;i<500;i++){
                LineTo(hdc,i,r.bottom-5-nBufScaled[i]); //Вывод отрезков
            }
        } //Конец case CURVE
    } //Конец switch (Mode)
} //Конец if(bDataOK)
SelectPen(hdc,hOldPen);
EndPaint(hwnd,&ps);
}

/*функция обработки сообщения WM_DESTROY*/
void OnDestroy(HWND){
    DeleteObject(hPen);
    PostQuitMessage(0);
}

/*Оконная процедура немодального диалога*/
BOOL CALLBACK DlgProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam){
    switch(msg){
        HANDLE_MSG(hwnd,WM_INITDIALOG,DlgOnInitDialog);
        HANDLE_MSG(hwnd,WM_COMMAND,DlgOnCommand);
        default:
            return FALSE;
    }
}

/*функция обработки сообщения WM_INITDIALOG для диалогового окна*/
BOOL DlgOnInitDialog(HWND hwnd,HWND,LPARAM){
    char S[10]; //Строка для обмена данными со списком LISTBOX
    SendDlgItemMessage(hwnd,Mode,BM_SETCHECK,1,0L); //Нажать кнопку режима
    for(int i=0;i<8;i++){ //Заполнение списка LISTBOX строками текста
        wsprintf(S,"%d%",nScales[i]); //Пересылка в строку числа и знака %
        SendDlgItemMessage(hwnd,ID_SCALE,LB_ADDSTRING,0,(LPARAM)S);
    }
    SendDlgItemMessage(hwnd,ID_SCALE,LB_SETCURSEL,nScaleIndex,0L);
    return TRUE;
}

/*функция обработки сообщений WM_COMMAND для диалогового окна*/
void DlgOnCommand(HWND hwnd,int id,HWND,UINT codeNotify){
    switch(id){ //Код элемента управления
        case IDOK: //Сообщение от кнопки "Закреть"
        case IDCANCEL: //Сообщения от системного меню диалога
            DestroyWindow(hwnd); //Уничтожение окна немодального диалога
            hSettingsBox=NULL; //Сбросим в нуль его дескриптор
            break;
        case ID_DOTS: //Сообщение от кнопки "Точки"
            Mode=DOTS; //Установим режим "Точки"
            InvalidateRect(GetParent(hwnd),NULL,TRUE); //Иницилируем WM_PAINT
            break;
        case ID_CURVE: //Сообщение от кнопки "Огибающая"
            Mode=CURVE; //Установим режим "Огибающая"
            InvalidateRect(GetParent(hwnd),NULL,TRUE); //Иницилируем WM_PAINT
            break;
    }
}

```

```

wc.lpszClassName=szClassName;
RegisterClass(&wc);
HWND hwnd=CreateWindow(szClassName,szTitle,WS_OVERLAPPEDWINDOW,
    0,0,600,400,HWND_DESKTOP,NULL,hInst,NULL);
ShowWindow(hwnd,SW_SHOWNORMAL);
hPen=CreatePen(PS_SOLID,1,RGB(0,0,255));
while(GetMessage(&msg,NULL,0,0)){
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return 0;
}

/*Оконная процедура главного окна*/
LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam){
    switch(msg){
        HANDLE_MSG(hwnd,WM_COMMAND,OnCommand);
        HANDLE_MSG(hwnd,WM_PAINT,OnPaint);
        HANDLE_MSG(hwnd,WM_DESTROY,OnDestroy);
        default:
            return(DefWindowProc(hwnd,msg,wParam,lParam));
    }
}

/*функция обработки сообщения WM_COMMAND от меню*/
void OnCommand(HWND hwnd,int id,HWND,UINT){
    OPENFILENAME ofn;//Структура для функции GetOpenFileName
    char szFilter[]="Файлы данных (*.DAT)\0*.dat\0Все файлы (*.*)\0*.*\0";
    DWORD nCnt;//Вспомогательная переменная для файловых операций
    switch(id){
        case MI_OPEN:
            ZeroMemory(&ofn, sizeof(OPENFILENAME));
            ofn.lStructSize=sizeof(OPENFILENAME);//Размер структуры
            ofn.hwndOwner=hwnd;//Дескриптор главного окна-владельца диалога
            ofn.lpstrFilter=szFilter;//Строка с шаблонами имен искоемых файлов
            ofn.lpstrFile=szFile;//Буфер для имени файла
            ofn.nMaxFile=sizeof(szFile);//Его размер
            ofn.Flags=OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
            if(GetOpenFileName(&ofn)){
                HANDLE hFile=CreateFile(szFile,GENERIC_READ,0,0,OPEN_EXISTING,0,NULL);
                if(hFile==INVALID_HANDLE_VALUE)break;//Если не удалось открыть файл
                ReadFile(hFile,nBuf,2*500,&nCnt,NULL);//Чтение из файла
                CloseHandle(hFile);
                bDataOK=TRUE;//Данные имеются, можно выводить
                for(int i=0;i<500;i++)//Выполним начальное масштабирование спектра
                    nBufScaled[i]=nBuf[i]*nCurrentScales;
                InvalidateRect(hwnd,NULL,TRUE);//Иницилируем WM_PAINT
                break;
            }
        else break;//Если файл не выбран
        case MI_EXIT:
            DestroyWindow(hwnd);
            break;
        case MI_SETTINGS:
            if(hSettingsBox==NULL)//Если диалог еще не открыт, создадим его
                hSettingsBox=CreateDialog(hI,"SettingsDlg",hwnd,DlgProc);
            break;
    }
}

/*функция обработки сообщения WM_PAINT
void OnPaint(HWND hwnd){
    RECT r;
    PAINTSTRUCT ps;

```



```

void OnCommand(HWND, int, HWND, UINT);
void OnPaint(HWND);
void OnDestroy(HWND);
/*Прототипы функций для диалогового окна*/
BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);
BOOL DlgOnInitDialog(HWND, HWND, LPARAM);
void DlgOnCommand(HWND, int, HWND, UINT);

/*файл 6-7.rc*/
#include "6-7.h"
Main MENU(
    POPUP "&Файл"{
        MENUITEM "&Открыть...", MI_OPEN
        MENUITEM "&Режимы...", MI_SETTINGS
        MENUITEM SEPARATOR
        MENUITEM "В&ыход", MI_EXIT
    }
)

SettingsDlg DIALOG 150,0,125,80
STYLE WS_VISIBLE | WS_SYSMENU | WS_POPUP
CAPTION "Режимы вывода данных"{
    CONTROL "Вид", -1, "BUTTON", BS_GROUPBOX, 10, 10, 65, 45
    CONTROL "Точки", ID_DOTS, "BUTTON", BS_AUTORADIOBUTTON, 15, 20, 45, 14
    CONTROL "Огибающая", ID_CURVE, "BUTTON", BS_AUTORADIOBUTTON, 15, 35, 55, 14
    CONTROL "", ID_SCALE, "LISTBOX", BS_NOTIFY|WS_VSCROLL|WS_BORDER, 85, 15, 30, 50
    CONTROL "Масштаб", -1, "STATIC", SS_LEFT, 85, 3, 40, 10
    CONTROL "Закрыть", IDOK, "BUTTON", BS_PUSHBUTTON, 25, 60, 40, 14
}

/*файл 6-7.cpr*/
/*Операторы препроцессора*/
#include <windows.h>
#include <windowsx.h>
#include "6-7.h"
/*Глобальные переменные*/
char szClassName[]="MainWindow";
char szTitle[]="Программа 6-5";
char szFile[128]; //Для спецификации выбранного файла
short nBuf[500]; //Буфер для чтения массива из файла
BOOL bDataOK=FALSE; //Если TRUE, данные прочитаны в буфер nBuf
HINSTANCE hI; //Дескриптор приложения
short nBufScaled[500]; //Буфер для масштабированного массива
int nScales[8]={200,175,150,125,100,75,50,25}; //Массив масштабов
int nScaleIndex=4; //Текущее (и начальное) значение индекса в массиве масштабов
float nCurrentScales=1; //Текущее (и начальное) значение масштаба
HWND hSettingsBox; //Дескриптор немодального диалога
GraphModes Mode=DOTS; //Режим наблюдения и его начальное значение
HPEN hPen; //Дескриптор нового пера
/*Главная функция WinMain*/
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int){
    MSG msg;
    WNDCLASS wc;
    hI=hInst;
    ZeroMemory (&wc, sizeof(wc));
    wc.style=CS_VREDRAW;
    wc.lpfnWndProc=WndProc;
    wc.hInstance=hInst;
    wc.hIcon=LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor=LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground=GetStockBrush(WHITE_BRUSH);
    wc.lpszMenuName="Main";

```

Третий параметр является флагом перерисовки фона. Если его значение равно TRUE, фон окна перерисовывается, что приводит к стиранию прежнего изображения. Если флаг равен FALSE, новое изображение рисуется поверх старого. В нашем случае, когда на экран выводятся отдельные точки в произвольных местах, перерисовывать весь фон, т. е. стирать точки старого графика, необходимо. В тех случаях, однако, когда на экран выводятся прямоугольные элементы изображения, которые целиком перекрывают старое изображение (например, знакоместа с буквами или цифрами), можно обойтись без перерисовывания фона, что сэкономит процессорное время и, главное, предотвратит мерцание изображения, возникающее при перерисовке окна.

Вывод на экран графика

Вывод в главное окно приложения точечного графика осуществляется в функции OnPaint() обработки сообщения WM_PAINT. Здесь после получения с помощью функции BeginPaint() дескриптора контекста устройства вызывается функция GetClientRect(), которая возвращает в структурную переменную *g* типа RECT координаты рабочей области главного окна, точнее говоря, ее размеры, так как координаты начала рабочей области всегда считаются равными нулю. Размер рабочей области по вертикали нам нужен для того, чтобы выводимый на экран график строился относительно нижней границы окна (значения *g.bottom*), независимо от текущего размера окна.

Вывод графика осуществляется в цикле из 500 шагов при условии наличия прочитанных из файла данных (переменная *bDataOK=TRUE*) функцией отображения точек SetPixel(). В качестве параметров этой функции указывается дескриптор контекста устройства, *x*- и *y*-координаты выводимой точки и ее цвет (с помощью макроса RGB()). Ордината точки с *x*-координатой *i* вычисляется как разность между нижней текущей границей окна *g.bottom* и высотой выводимой кривой в этой точке *pBuf[i]*. Из полученного значения вычитается 5, чтобы поднять график на 5 пикселей над нижней границей окна.

Немодальный диалог

Как уже отмечалось ранее, немодальные диалоги отличаются от модальных тем, что при их выводе на экран остальные элементы управления окна не блокируются, что позволяет одновременно работать и с немодальным диалогом, и с другими средствами управления приложением. Для иллюстрации процедуры создания немодального диалога и работы с ним добавим к программе предыдущего примера немодальное диалоговое окно, позволяющее с помощью содержащихся в нем элементов управления выполнять следующие действия:

- масштабирование выводимого графика;
- переключение вида графика – отдельные точки или точки, соединенные линиями (огibaющая).

```
/*Программа 6-7. Немодальный диалог с элементами управления*/
/*Файл 6-7.h*/
/*Константы*/
#define MI_OPEN 101
#define MI_EXIT 102
#define MI_SETTINGS 103
#define ID_DOTS 201
#define ID_CURVE 202
#define ID_SCALE 203
/*Определение нового типа GraphModes*/
enum GraphModes {DOTS=ID_DOTS, CURVE=ID_CURVE};
/*Прототипы функций для главного окна*/
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
```