Глава 1

Окна

- ❑ Привлечение внимания к окну
- ❑ Окно приложения
- ❑ Растягиваемые формы
- ❑ Окна нестандартной формы
- ❑ Области окна
- ❑ Настраиваемый интерфейс
- ❑ Окна других приложений

При работе в операционной системе Windows окна нам встречаются повсюду. Отсюда, собственно, и название этой операционной системы, в которой идея организации оконного пользовательского интерфейса прижилась очень хорошо.

Для пользователя понятие «окно» чаще всего означает прямоугольную область в рамке с каким-то содержимым, например полем для редактирования текстового документа и, возможно, полосами прокрутки. Для программиста все несколько интереснее: окнами могут быть как окна приложений (которые видит пользователь), так и кнопки, надписи, сами полосы прокрутки и все, что угодно, — все, что пользователь видит или не видит на экране. Только одни окна будут перекрывающимися (например, главное окно приложения), а другие — дочерними (кнопки, полосы прокрутки и т. д.). Дочерние окна располагаются поверх родительских окон, визуально подчеркивая свою принадлежность родительскому окну.

Теперь стоит оговориться, почему в книге иногда употребляется понятие «окно», а иногда — «форма». Под «окном» понимается любое окно, действительно созданное в системе. Форма — это тоже окно, но свойства формы, а также расположение на ней компонентов задаются визуально в среде разработки. Процесс создания самой формы, равно как и всех ее компонентов, скрыт от программиста. Другими словами, форма — это окно, созданное по некоторому шаблону, причем как внутреннее представление шаблона, так и способ создания окна по нему не должны волновать программиста.

На форму в среде разработки Borland C++ Builder могут быть помещены компоненты. Они могут быть как отображаемыми (надпись, кнопка, набор вкладок), так и неотображаемыми (таймер, серверный сокет и т. д.). Отображаемые компоненты (в книге также называются элементами управления) могут быть как оконными, то есть являться полноценными дочерними окнами Windows, так и не обладающими собственным окном, как, например, компонент `TLabel`, который отображается на форме, но собственного окна не имеет — он просто в нужный момент отображается в нужном месте формы.

В Windows окно является не только частью графического интерфейса — оно также служит для взаимодействия приложения с пользователем или системой. При возникновении какого-либо события в системе, которое затрагивает работающее приложение (например, пользователь провел указателем мыши над окном приложения, изменились системное время или дата, разрешение монитора), окно приложения получает сообщение. Получение сообщения окном представляется как вызов определенной пользовательской функции, зарегистрированной в системе как функция-обработчик сообщения окна (оконная функция).

Каждое окно принадлежит к определенному оконному классу. К одному оконному классу может принадлежать множество окон. Класс определяет свойства и способ реализации одинаковых окон (например, класс окон кнопок). Так, оконная функция одна для всех окон одного класса. В приведении более подробного описания оконных классов особой необходимости нет, тем более что здесь они не пригодятся.

В завершение несколько слов о том, как обрабатываются полученные окном сообщения. Классическая реализация оконной функции предполагает написание подобия большой конструкции switch, в каждой ветви типа case которой должна быть записана реакция на определенное сообщение. Если же для реализации оконного приложения используется библиотека классов наподобие той, что реализована в Borland, оконная функция скрывается в недрах библиотеки. Реакцией на приход определенного сообщения в таком случае является вызов одного из методов объекта, соответствующего окну — получателю сообщения. Такой метод часто называется обработчиком события, хотя его можно назвать и обработчиком сообщения. За исключением написания сложных алгоритмов, прикладной программист, реализующий приложение с пользовательским интерфейсом, по большей части занимается именно написанием нужных обработчиков событий.

Привлечение внимания к окну

Существует несколько способов привлечь внимание к окну приложения. Одними из самых простых и в то же время достаточно эффективных из неназойливых способов являются инверсия заголовка окна, требующего внимания, изменение окраски кнопки приложения на Панели задач или и то, и другое вместе. Эти способы часто используются в приложениях.

Для инверсии заголовка окна или кнопки на Панели задач никаких функций перерисовки создавать не нужно — достаточно воспользоваться простой API-функцией FlashWindowEx. Данная функция принимает единственный аргумент — указатель на структуру FLASHWINFO, объявленную следующим образом (листинг 1.1).

Листинг 1.1. Объявление структуры FLASHWINFO

```
typedef struct {
    UINT  cbSize;
    HWND  hwnd;
    DWORD dwFlags;
    UINT  uCount;
    DWORD dwTimeout;
} FLASHWINFO;
```

В табл. 1.1 приведены описания полей структуры FLASHWINFO.

Значение параметра dwFlags формируется из приведенных ниже констант с использованием операции побитового ИЛИ:

- FLASHW_CAPTION — инвертирует состояние заголовка окна;
- FLASHW_TRAY — заставляет мигать кнопку на Панели задач;
- FLASHW_ALL — реализует совместное использование операторов FLASHW_CAPTION и FLASHW_TRAY;

- FLASHW_TIMER — заставляет периодически изменяться состояние заголовка окна и/или кнопки на Панели задач вплоть до того момента, пока функция FlashWindowEx не будет вызвана с флагом FLASHW_STOP;
- FLASHW_TIMERNOFG — заставляет периодически изменяться состояние заголовка окна и/или кнопки на Панели задач до тех пор, пока окно не станет активным;
- FLASHW_STOP — восстанавливает исходное состояние окна и кнопки на Панели задач.

Таблица 1.1. Поля структуры FLASHWINFO

Поле	Тип	Назначение
cbSize	UINT	Размер структуры FLASHWINFO (для отслеживания версий)
hwnd	HWND	Дескриптор окна
dwFlags	DWORD	Набор флагов, задающий режим использования функции FlashWindowEx. Значения этого флага и их описания приведены после таблицы
uCount	UINT	Количество инверсий заголовка окна и/или кнопки на Панели задач
dwTimeout	DWORD	Время между изменениями состояния заголовка окна и/или кнопки на Панели задач. Если значение равно нулю, используется системное значение таймаута

Несмотря на необходимость заполнять поля структуры, пользоваться описываемой функцией крайне просто. Так, в листинге 1.2 приведена функция, инвертирующая заголовок окна (hWnd) и его кнопки на Панели задач с интервалом, определяемым параметром time_out в миллисекундах, определенное количество раз (параметр count).

Листинг 1.2. Инвертирование заголовка окна и кнопок на Панели задач

```
void FlashWindow( HWND hWnd, UINT count, DWORD time_out )
{
  FLASHWINFO flash_info = {0};
  flash_info.cbSize = sizeof(flash_info);
  flash_info.hwnd = hWnd;
  flash_info.uCount = count;
  flash_info.dwTimeout = time_out;
  flash_info.dwFlags = FLASHW_ALL;
  ::FlashWindowEx( &flash_info );
}
```

Чтобы использовать приведенную в листинге 1.2 функцию, достаточно определиться с интервалом и количеством «миганий» и получать дескриптор окна, например, следующим образом:

```
FlashWindow( pForm->Handle, 3, 100 );
```

Однако в этом случае не изменяется окраска кнопки на Панели задач. Дело в том, что главным окном приложения при использовании Borland C++ Builder (как и Delphi) является не какая-нибудь их форма, а еще одно окно, которое видимо, но имеет нулевой размер. Дескриптором этого окна является как раз значение свойства `Handle` объекта `Application`. Тогда одновременную инверсию заголовка окна и кнопки на Панели задач можно организовать так:

```
FlashWindow( pForm->Handle, 3, 100 );
FlashWindow( Application->Handle, 3, 500 );
```

Окно приложения

Как было сказано в предыдущем разделе, оконное приложение, реализованное в Borland C++ Builder, имеет окно, которое не отображается, но все же играет важную роль. Это окно можно очень просто увидеть, для чего достаточно задать ему ненулевой размер. Например, так, как показано в листинге 1.3.

Листинг 1.3. Отображение окна приложения

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ::SetWindowPos(Application->Handle, 0, 0, 0, 300, 200,
                 SWP_NOZORDER | SWP_NOMOVE);
}
```

Обратите внимание, что текст окна (рис. 1.1) и является текстом кнопки на Панели задач. Этот текст может быть легко изменен с помощью свойства `Title` объекта `Application`.

Рис. 1.1. Окно приложения

Изменяя видимость (не размер, а именно стиль видимо/невидимо) окна приложения, можно с легкостью скрывать или отображать кнопку на Панели задач. Так, скрыть или показать окно приложения, а с ним и кнопку на Панели задач можно следующим образом (листинг 1.4).

Листинг 1.4. Скрытие и отображение кнопки на Панели задач

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  ::ShowWindow(Application->Handle, SW_HIDE); //Скрытие
                                              //кнопки
}
void __fastcall TForm1::Button4Click(TObject *Sender)
{
  ::ShowWindow(Application->Handle, SW_SHOW); //Отображение
                                              //кнопки
}
```

Растягиваемые формы

При создании сложных форм бывает очень полезно дать пользователю возможность изменять их размер и пропорции. Но как быть с компонентами на растягиваемой или, наоборот, уменьшаемой форме? В данном случае разработчики, использующие среду программирования Borland C++ Builder, могут радоваться: у них в распоряжении есть встроенная возможность привязки компонентов к сторонам формы (так называемый механизм якорей).

Взгляните на группу свойств для задания якорей компонента, показанную на рис. 1.2.

Четыре булевых значения этой группы предназначены для задания привязки краев компонента к краям формы:

- akLeft — левый край компонента сохраняет постоянное расстояние до левого края формы (по умолчанию);
- akTop — верхний край компонента сохраняет постоянное расстояние до верхнего края формы (по умолчанию);
- akRight — правый край компонента сохраняет постоянное расстояние до правого края формы;
- akBottom — нижний край компонента сохраняет постоянное расстояние до нижнего края формы.

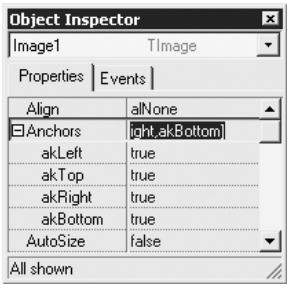

Рис. 1.2. Настройка якорей компонента

Таким образом, несколькими щелчками кнопки мыши можно легко создать форму, компоненты которой изменяют свои размеры автоматически при ее растягивании. Пример такой формы приведен на рис. 1.3.

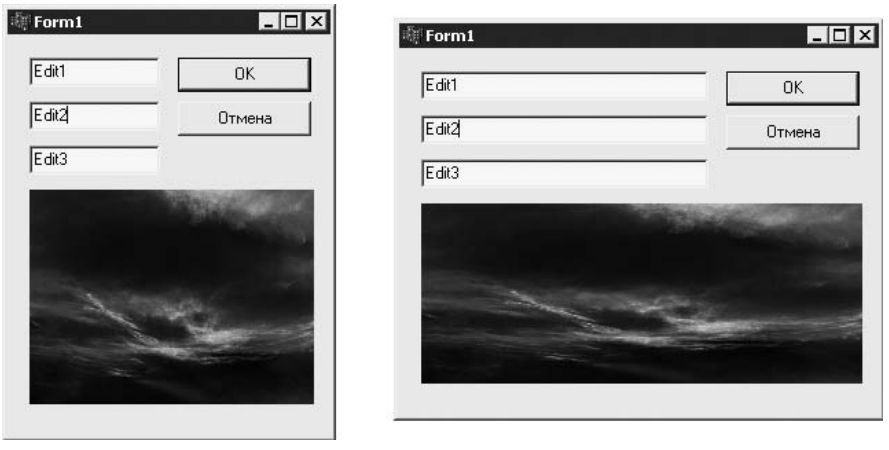

Рис. 1.3. Форма до и после изменения размера

Для компонентов показанной на рисунке формы назначены якоря, описанные в табл. 1.2.

Таблица 1.2. Якоря формы, показанной на рис. 1.3

Компонент	Якоря
Edit1, Edit2, Edit2	akLeft, akRight, akTop
Button1 (кнопка OK), Button2 (кнопка Отмена)	akRight, akTop
Image1 (единственный рисунок на форме)	akLeft, akRight, akTop, akBottom

При применении описанного здесь механизма стоит учесть еще и возможность задания ограничений на минимальный и максимальный размер как формы, так и любого из ее компонентов. Группа свойств для задания ограничений размера показана на рис. 1.4.

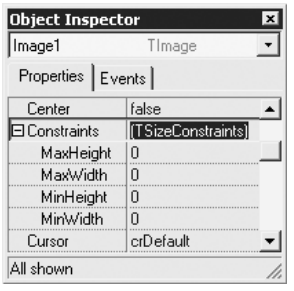

Рис. 1.4. Ограничения размеров компонента

При нулевых значениях ограничения на размер формы или компонента не накладываются. Если задать, например, максимальное значение ширины, то форму или компонент нельзя будет растянуть до размера, превышающего заданный. При этом если это ограничение задано для одного из компонентов формы и применяется упомянутый выше механизм якорей, то размер формы станет невозможно увеличивать, как только будет достигнута максимальная ширина одного из компонентов.

Окна нестандартной формы

Ниже приведены несколько примеров, демонстрирующих способы изменения внешнего вида приложений. Речь идет о создании окон нестандартных непрямоугольных форм. Но сначала ознакомьтесь немного с теорией, чтобы было понятно, как все работает.

Создание и использование регионов

Рассмотренные далее эффекты основаны на использовании регионов (областей) отсечения — в общем случае сложных геометрических фигур, ограничивающих область прорисовывания окна. По умолчанию окна (в том числе и окна элементов управления) имеют область отсечения, заданную прямоугольным регионом с высотой и шириной, равной высоте и ширине самого окна.

Однако использование регионов прямоугольных форм для указания областей отсечения совсем не обязательно, в чем могут убедиться пользователи Windows XP. Пример использования отсечения по заданному непрямоугольному региону при рисовании произвольного окна наглядно продемонстрирован на рис. 1.5. На этом рисунке буквой «а» обозначен внешний вид, который имела бы форма, будь область

отсечения прямоугольной. Буквой «б» обозначен применяемый регион, формирующий область отсечения. Буквой «в» обозначен вид формы, полученный в результате рисования с отсечением по границам заданного региона.

Теперь вместо рассмотрения подробностей обработки регионов отсечения, которые имеют место в функциях рисования, лучше сразу перейти к рассмотрению операций, позволяющих создавать, удалять и модифицировать регионы.

Создание и удаление регионов

Создавать несложные регионы различной формы можно с помощью следующих API-функций:

```
HRGN CreateRectRgn( int x1, int y1, int x2, int y2);
HRGN CreateEllipticRgn( int x1, int y1, int x2, int y2);
HRGN CreateRoundRectRgn( int x1, int y1, int x2, int y2, int w,
int h);
```

Все перечисленные здесь и ниже функции создания регионов возвращают дескриптор GDI-объекта «регион». Этот дескриптор впоследствии передается в различные функции, работающие с регионами.

а *б* *в*

Рис. 1.5. Рисование окна по заданному региону

Итак, первая из приведенных функций (`CreateRectRgn`) предназначена для создания регионов прямоугольной формы. Параметры этой функции необходимо толковать следующим образом:

- `x1` и `y1` — горизонтальная и вертикальная координаты левой верхней точки прямоугольника;
- `x2` и `y2` — горизонтальная и вертикальная координаты правой нижней точки прямоугольника.

Следующая функция (`CreateEllipticRgn`) предназначена для создания региона эллиптической формы. Параметры этой функции — координаты прямоугольника (аналогично `CreateRectRgn`), в который вписывается эллипс.

Третья функция (`CreateRoundRectRgn`) создает регион — прямоугольник со скругленными углами. При этом первые четыре параметра функции аналогичны

соответствующим параметрам функции CreateRectRgn. Параметры h и w — ширина и высота сглаживающих углы эллипсов (рис. 1.6).

С помощью трех приведенных функций, если нужно, можно создавать регионы даже очень сложной формы. Это достигается посредством использования многочисленных операций над простыми регионами, как в приведенном далее примере создания региона по битовому шаблону. Однако существует еще одна несложная функция, которая позволяет сразу создавать регионы-многоугольники по координатам вершин многоугольников:

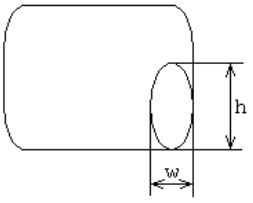

Рис. 1.6. Скругление углов прямоугольного региона функцией CreateRoundRectRgn

```
HRGN CreatePolygonRgn(const POINT *points,
int count, int fillmode);
```

Функция CreatePolygonRgn принимает следующие параметры:

- points — указатель на массив структур типа POINT; каждый элемент массива описывает одну вершину многоугольника; координаты не должны повторяться;
- count — количество элементов в массиве, на который указывает параметр points;
- fillmode — режим заливки региона (в данном случае определяет, попадает ли внутренняя область многоугольника в заданный регион).

Параметр FillMode может принимать значения WINDING (попадает любая внутренняя область) или ALTERNATE (попадает внутренняя область, если она находится между четной и следующей нечетной сторонами многоугольника).

Поскольку регион является GDI-объектом (подробнее о данных объектах читайте в гл. 2), то для его удаления, если он не используется системой, применяется функция удаления GDI-объектов DeleteObject:

```
BOOL DeleteObject(HGDIOBJ hObj);
```

Единственным параметром этой функции является дескриптор GDI-объекта, который после вызова DeleteObject становится недействительным.

Регион как область отсечения при рисовании окна

Обычно необходимо удалять регион, если он не используется системой. Однако после того как регион назначен окну в качестве области отсечения, удалять его не следует. Функция назначения региона окну имеет следующий вид:

```
int SetWindowRgn(HWND hWnd, HRGN hRgn, BOOL bRedraw);
```

Функция возвращает 0, если произвести операцию не удалось, и ненулевое значение в противном случае. Параметры функции SetWindowRgn следующие:

- hWnd — дескриптор окна, для которого устанавливается область отсечения (свойство Handle формы или элемента управления);
- hRgn — дескриптор региона, назначаемого в качестве области отсечения (в простейшем случае является значением, возвращенным одной из функций создания региона);
- bRedraw — флаг перерисовки окна после назначения новой области отсечения; для видимых окон обычно используется значение true, для невидимых — false.

Чтобы получить копию региона, формирующего область отсечения окна, можно использовать API-функцию GetWindowRgn:

```
int GetWindowRgn(HWND hWnd, HRGN hRgn);
```

Первый параметр функции — дескриптор рассматриваемого окна. Второй параметр — дескриптор предварительно созданного региона, который в случае успеха модифицируется функцией GetWindowRgn так, что становится копией региона, формирующего область отсечения окна. Описания целочисленных констант — возможных возвращаемых значений функции — приведены ниже:

- NULLREGION — пустой регион;
- SIMPLEREGION — регион в форме прямоугольника;
- COMPLEXREGION — регион сложнее, чем прямоугольник;
- ERROR — при выполнении функции возникла ошибка (либо окну неверно задана область отсечения).

Ниже приведен пример использования функции GetWindowRgn (предполагается, что приведенный ниже код является телом одного из методов класса формы) (листинг 1.5).

Листинг 1.5. Использование функции GetWindowRgn

```
HRGN rgn = ::CreateRectRgn(0,0,0,0); //Первоначальная форма
региона не важна
if ( ::GetWindowRgn(Handle, rgn) != ERROR )
{
  //Операции с копией региона, формирующего область отсечения
  //окна
  //...
}
::DeleteObject(rgn); //Использовалась копия региона, которую
                     //нужно удалить
                     //(здесь или в ином месте, но
                     //самостоятельно)
```