МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕ-СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ

Наманганский инженерно-педагогический институт

Кафедра: «Информатика и ИТ»

РЕФЕРАТ

На тему: «Windows»

Выполнил(а) студент группы 41 ООТ-14
Умарова А.

Принял(а) Абдуллаева О.

НАМАНГАН 2015 Г.

РЕФЕРАТ

НА ТЕМУ: Windows

ВВЕДЕНИЕ

Для эффективной работы системы и ее большого покупательского спроса недостаточно только того, чтобы аппаратура и программы обеспечивали правильные результаты - не менее важным фактором является удобство работы пользователя. Поэтому в настоящее время ни один программный продукт, лишенный более менее удобного интерфейса взаимодействия пользователя с компьютером и программ между собой, не может рассчитывать на успех.

В мире разработано огромное количество различных систем поддержки создания пользовательского интерфейса. Наиболее прогрессивному, по нашему мнению, являются обьектноориетированная система Turbo Vision фирмы Borland и операционная среда Windows фирмы Microsoft. Рассмотрением возможностей системы Turbo Vision занималась в своей работе Фомичева Т.Л., а в представленной работе производится изучение и обзор средств, входящих в состав операционной среды Windows.

Тот кто написал хотя бы одну коммерческую программу, то знает, сколько времени и сил уходит на организацию меню и выработку единого, непротиворечивого, интуитивно ясного и удобного пользовательского интерфейса. Интерфейс, разработанный корпорацией Microsoft является одним из лучших и стал своеобразным эталоном для подражания.

В Microsoft Windows программисту доступна вся мощь этого интерфейса - он избавляется от необходимости организовывать меню, работу с клавиатурой и мышью (достаточно воспользоваться стандартными средствами). Интерфейс с пользователем Windows является полным и цельным. В нем решены не только проблемы организации меню, но и все общение с пользователем организовано стандартными средствами.

ИСТОРИЯ СОЗДАНИЯ MICROSOFT WINDOWS

Корпорация Microsoft объявила о начале разработки графической операционной оболочки Windows 10 ноября 1983 года, хотя еще в конце 1982 года программисты Microsoft начали создавать универсальный набор графических процедур, названный Графическим Интерфейсом с Компьютером (CGI).

Первоначально предполагалось, что CGI как набор процедур будет поставляться с компиляторами Microsoft, позволяя разработчикам программ выводить графику на самые различные типы принтеров. Программисты могли бы использовать в своих программах функции CGI, которые затем переводились бы в команды нужного типа принтера.

Вскоре после начала работ над CGI корпорация Microsoft развернула работы по созданию графической операционной среды для компьютеров с MS-DOS. Создание такой оболочки было инспирировано неожиданным интересом пользователей IBM совместимых компьютеров к объявлению корпорацией VISICorp о начале работ над многооконной операционнной оболочкой VisiOn. Таким образом, старая добрая конкуренция сделала свое дело - в феврале 1983 года стало ясно, что Microsoft создаст свою собственную оболочку Windows.

Хотя многие особенности и свойства Windows кардинально изменились в последующем, некоторые положения были ясны с самого начала. Windows должна быть многозадачной, т.е. должна позволять запускать несколько программ одновременно. Windows должна работать со всеми типами дисплеев и принтеров. И поскольку пользователю очень трудно отказаться от привычных программных средств, Windows должна позволять запускать приложения MS-DOS. Последняя цель настолько оказалась трудной в реализации, что задержала весь проект на многие месяцы.

Задача оказалось не из простых. Типичной машиной был компьютер со сравнительно медленным процессором 8088 и оперативной памятью 64Kb. Если учесть, что MS-DOS накладывает принципиальное ограничение по оперативной памити в 640Kb и, в отличие от компьютеров Macintosh, не обеспечивает программноаппаратную поддержку графики, разработка многозадачной графической операционной оболочки грозила вырасти в неразрешимую задачу. Однако Microsoft развернула работы над Windows полным ходом. К осени 1983 года разработкой Windows было занято уже 15 высококвалифицированных программистов, привлеченных к проекту из различных филиалов Microsoft. Многие фирмы командировали своих специалистов для одновременного участия в проекте.

Когда работы над CGI перешли в фазу тестирования, это не был уже самостоятельный продукт. CGI был переменован в GDI (the Graphics Device Interface - "Интерфейс с графическими Устройствами") и стал частью Windows, включив в себя процедуры работы не только с принтером, но и с дисплеем. Для этого в CGI были добавлены функции, обеспечивающие поддержку типографско-издательских особенностей в работе с текстами и процедуры манипулирования графическими объектами.

При презентации первого варианта Windows было обещано, что коммерческая продажа начнется в мае 1984. Весной 1984 года дата начала продаж была отодвинута на ноябрь. В ноябре эта дата была перенесена на июнь 1985 года. Однако коробки с Windows появились в магазинах только 18 ноября 1985 года.

Несколько факторов привели к столь значительным задержкам. Однако главную роль сыграло нежелание Microsoft выходить на рынок с сырым продуктом. В феврале 1984 года глава корпорации Microsoft Билл Гейтс (Bill Gates) провел семинар, на котором представил Software Development Kit (SDK), пакет для написания приложений под Windows, продемонстрировав на нем возможности, предлагаемые программисту под Windows.

SDK произвел хорошее впечатление и к июню 1984 года было продано почти 100 копий. Начав работать под Windows, программисты присылали в Microsoft свои замечания, заставляя разработчиков еще и еще отшлифовывать Windows, облегчая написание приложений. Был заменен язык, на котором создавался Windows (сначала это был Microsoft Pascal, затем Lattice C и, наконец, Microsoft C Compiler). Ориентация на Microsoft C

Compiler также была одной из причин задержки работ над проектом, потому, что сам компилятор не был готовым продуктом и постоянно совершенствовался.

Однако, несмотря на то, что в ноябре 1985 года Windows все же вышел на прилавки магазинов, в течение двух лет (с 1985 по 1987 год) Windows не оказал большого влияния на рынок программного обеспечения и не стал альтернативой MS-DOS, как надеялся Microsoft. Большое число пользователей (51%) покупали Windows не ради созданных для него приложений (которых было очень мало по сравнению с обычными программами под MS-DOS) или графического интерфейса, а для того, чтобы иметь возможность быстро переключаться с одного DOS приложения на другое.

С 1987 года ситуация начала меняться. Начали появляться мощные и известные приложения, переписанные для работы под Windows. Сейчас можно назвать такие продукты как Page Maker под Windows, очень мощная электронная таблица Excel, Windows Graph, Word for Windows и многие другие.

С выходом третей версии Windows стало ясно, что Windows не просто завоюет прочное место на рынке программного обеспечения, но станет основой стратегической политики Microsoft в создании программного обеспечения. Windows 3.0 произвел настоящий фурор, предлагая совершенно потрясающую графику и новые, неожиданные возможности.

В июле 1990 года "PC Magazine" поместил статью о новой версии Windows 3.0, в которой отмечалось, что Microsoft Windows превратился в блестящую многозадачную операционную систему, которой еще пытается стать OS/2. Как и все программные продукты Microsoft, первые версии Windows не оправдывали надежд несмотря на то, что выходили со значительными задержками, но в своем последнем варианте оставляют конкурентов далеко позади.

ОБЗОР ОСНОВНЫХ ПРИНЦИПОВ ОРГАНИЗАЦИИ ИНТЕРФЕЙСА В WINDOWS

Для понимания принципов организации интерфейса Windows необходимо иметь представление об основных его элементах, на которых построено выполнение программ и взаимодействие программы и пользователя. Весь интерфейс Windows основывается на трех китах:

- аппаратно-независимая графика;
- стандартный оконно-ориентированный интерфейс;
- взаимодействие приложений с системой Windows и между собой посредством передачи сообщений;

Данные элементы системы Windows самым тесным образом связаны между собой, вместе образуют целостную систему и отдельное рассмотрение каждого из них в отрыве от других не имеет смысла. Перечисленные выше элементы системы Windows ниже будут рассмотрены более подробно, а пока вкратце остановимся на том, что они из себя представляют.

Аппаратно-независимый графический интерфейс (GDI)

Любая программа для Windows может выполнять вывод на любое устройство с помощью одного и того же набора вызываемых подпрограмм. Причем для Windows приложения все устройства выглядят одинаково и программисту не нужно заводится об управлении конкретным устройством на низком уровне.

Каждое устройство имеет свой драйвер, отвечающий за фактическое выполнение графического вывода. Для устройств, которым при этом необходима помощь, GDI обеспечивает программную эмуляцию, использующую для реализации функций высокого уровня средства низкого уровня этого устройства.

При выводе информации на экран дисплея GDI обеспечивает оконно-ориентированную графику. Это означает, что каждое окно рассматривается как отдельная область прорисовки. Когда программа выполняет в окне прорисовку, то координаты по умолчанию устанавливаются так, что точка начала координат (0,0) находится в верхнем левом углу клиентной области окна.

Кроме того, рисунки автоматически отсекаются по границам окна. Подобные механизм защиты работает двусторонне, то есть ни вы не можете нарисовать что-либо вне своего окна, ни другая программа нарисовать что-то в вашем окне.

Стандартный оконно-ориентированный интерфейс

Система Windows имеет встроенную поддержку ряда обьектов пользовательского интерфейса: окон, пиктограмм, меню, блоков диалога и т.п.

Окно

Окно представляет собой самую важную часть

пользовательского интерфейса. Оно играет ключевую роль.

Для программиста окно служит для организации прочих обьектов пользовательского интерфейса и направляет прохождение сообщений в системе, окно обеспечивает область экрана для связи с пользователем.

Окно - это самостоятельно существующий обьект, параметры которого описаны в специальных структурах данных, а поведение функцией окна.

Каждое окно принадлежит классу окон. Класс окон - это шаблон, по которому реализуются реальные окна. С каждым классом окон и, следовательно, с каждым окном связан специальный тип подпрограммы, называемый процедурой окна. Задача процедуры окна состоит в обработке поступающих окну сообщений.

Каждое приложение располагается в своем собственном окне и имеет по крайней мере хотя бы одно окно - главное окно приложения. Из приложения Windows непосредственно нельзя осуществить вывод на экран, так как экран является разделяемым ресурсом, а средством его разделения являются окна. Таким образом, прежде чем отобразить что-либо на экране, нужно создать окно, и только в окне можно осуществить вывод.

Пиктограммы

Пиктограмма представляет собой небольшой рисунок,

который служит для пользователя напоминанием о чем-либо и обозначают команду, программу или некоторые данные.

Меню

Меню представляет собой список команд и функций программы. Имеется пять типов меню:

- системные,
- горизонтальные,
- выпадающие,
- вложенные,
- всплывающие.

Системные меню обеспечивают стандартный набор операций, которые могут быть выполнены с окном (перемещение, изменение размеров, закрытие, переключение на другую задачу и т.д.). Это меню обязательно находится в главном меню каждого приложения.

Горизонтальное меню фиксировано привязывается к верху окна.

Выпадающие меню появляются при выборе соответствующих пунктов горизонтального меню.

Вложенные меню появляются при выборе соответствующих пунктов выпадающих меню. Прикладная программа может вкладывать одно меню в другое до любого уровня вложенности.

Всплывающие меню могут располагаться в произвольной позиции в окне и фактически в любой позиции на экране дисплея.

Полосы прокрутки

Используются в тех случаях, когда обьект данных больше размеров окна. Полосы прокрутки бывают вертикальными и горизонтальными. Они позволяют пользователю управлять отображением больших обьемов данных и иметь к ним доступ.

Существует ограничение на обьем данных в скроллинге. Обьем данных не может превышать 64 Кбайт.

Курсоры

Курсор представляет собой битовый образ, перемещаемый по экрану в ответ на перемещения мыши или другого координатного устройства. Программа может изменить формукурсора, чтобы отобразить некоторое изменение в системе.

Каретка

Каретка - это небольшая битовая матрица, которая является отметкой фокуса ввода с клавиатуры. Окно, управляющее вводом с клавиатуры может создать каретку, чтобы сообщить пользователю об этот факте.

Пользовательский интерфейс Windows поддерживает только одну каретку на экране.

Блоки диалога

Блок диалога - это стандартный способ приема программой ввода от пользователя. Типичный пример блока диалога - это блок диалога для открытия файла.

Блок диалога представляет собой окно, содержащее внутри себя отдельные окна, которые либо выводят некоторую информацию, либо позволяют принять ввод от пользователя. Каждое их этих окон называется элементом управления диалогом.

Система имеет шесть предопределенных классов окон, на базе которых создаются элементы управления блоками диалога:

- кнопки;
- комбинированные блоки;
- элементы управления редактированием;
- блоки списков;
- полосы прокрутки;
- статические элементы.

Более полное описание обьектов интерфейса представлено ниже.

Механизм сообщений

В системе Windows любое приложение строится как совокупность обработчиков различных событий, которые происходят совершенно независимо друг от друга. Каждое событие генерирует, сообщение, которое передаются всем приложениям, для которых оно может представлять интерес.

Приложение представляет собой 16-битовое значение без знака, которому для удобства присваивается символьная константа. Все сообщения имеют единый формат и являются единственным средством связи приложения с операционной оболочкой и с другими приложениями. Некоторые сообщения могут в свою очередь порождать другие сообщения.

При поступлении сообщения о произошедшем событии, это сообщение помещается в системную очередь Windows. Системная очередь в Windows одна. После этого сообщения из системной очереди распределяются между приложениями. Для каждого приложения Windows организует и поддерживает отдельную очередь, куда пересылаются все сообщения для этого приложения.Обработку очереди приложения осуществляет само приложение.

Если сообщение поступило например от устройств ввода, таких как мышь или клавиатура, то для определения адресата сообщения используется понятие "фокус ввода". Так как пользователь в каждый момент времени может работать только с одним приложением. Таким образом, говорят, что приложение, в которое попадают сообщения от клавиатуры в момент ввода, имеет фокус ввода, поэтому все сообщения от устройств ввода информации поступают из системной очереди в очередь приложения, имеющего фокус ввода в данный момент.

Для обработки поступающих сообщений в программе организуется цикл сообщений, который создается при при создании окна приложения. Цикл сообщений извлекает сообщения из очереди и передает их функции управления соответствующим окном приложения, причем не напрямую а через Window. О функции окна приложений пойдет речь позже.

Все стандартные сообщения, определенные в системе Windows можно разделить на несколько групп:

- аппаратные (входные данные от мыши и клавиатуры);
- об организации окна (уведомление, требование действия, запрос);
- об организации интерфейса пользователя (меню,

указатель мыши, линейка прокрутки, блоки диалога, MDI);

- о завершении (закрытие прикладной программы или системы);
- частные (элементы управления блоком диалога: редактор кнопка, блок списка, комбо-блок);
- уведомление о системном ресурсе (изменение цвета, шрифты, буферизация печати, режимы работы устройств);
- о совместном использовании данных (буфер вырезанного изображения и динамический обмен данными DDE);
- внутренние системные (недокументированные сообщения).

Ниже будут более подробно рассмотрены перечисленные классы сообщений в контексте их применения. Особое внимание будет уделено сообщениям по взаимодействию между приложениями

и совместному использованию данных.

ИНТЕРФЕЙС ГРАФИЧЕСКИХ УСТРОЙСТВ (GDI)

В данном разделе рассматриваются вопросы связанные с созданием графического вывода, обсуждаются различные типы графического вывода, поддерживаемых интерфейсом графических устройств (GDI).

GDI представляет собой библиотеку графического вывода Windows. GDI обеспечивает графический вывод на экран дисплея и на устройства для получения твердых копий, например, принтеры и плоттеры. GDI отвечает за создание отображения каждой линии, буквы или графического знака, выводимого программой для Windows. Сама Windows использует GDI при подборке элементов, составляющих пользовательский интерфейс, - окон, пиктограмм, меню, блоков диалога и т.д.

Устройства GDI

Интерфейс графических устройств позволяет выполнять графический вывод на различные устройства. Для того, чтобы GDI работал с конкретным устройством, необходим специальный элемент программного обеспечения - драйвер устройства, который преобразует запросы графического вывода в конкретные действия для рисования на конкретном устройстве.

Помимо этого, драйвер устройства представляет GDI набор флагов, которые сообщают, какими графическими возможностями обладает данное устройство. Существует пять наборов таких флагов: для кривых линий, для прямых, многоугольников, битовых образов и текстов, которые сообщают GDI, когда можно направить устройству непосредственно запрос, а когда такой запрос надо предварительно преобразовать в последовательность запросов низкого уровня. Это зависит то возможностей конкретного устройства.

Помимо физических устройств GDI поддерживает логические устройства, или псевдоустройства. Псевдоустройства служат для хранения изображений.

В отличие то физических устройств, которые выводят изображения на определенной аппаратной базе, псевдоустройства позволяют "перехватить" образ изображения в оперативной памяти или на диске. GDI поддерживает два типа псевдоустройств: битовые образы и метафайлы.

Битовые образы имеют прямоугольную форму и хранят изображения в памяти в таком виде, в каком графические образы хранятся дисплейным адаптером, и обеспечивают быстрое получение копии картинки. Битовые образы используются и для хранения образов, которые нужно быстро выводить на экран, например, пиктограммы, курсоры и т.д.

Метафайлы создаются средствами записи-воспроизведения GDI. С точки зрения расходуемой памяти метафайлы предпочтительнее, чем битовые образы, однако работа сними происходит медленнее. Обычно в метафайлах хранятся крупные изображения.

Логические обьекты графики GDI

Одним из средств достижения аппаратной независимости GDI является использование логических обьектов графики. Такой обьект описывает, каким образом должен выполняться вывод, это высокоуровневый аппаратно-независимый запрос. GDI поддерживает следующие логические обьекты графики:

- перья (для рисования линий);
- кисти (для закрашивания областей);
- шрифты (для вывода текстов);
- логические цвета (описывающие цвета вывода).

После создания логического обьекта он может быть использован в отношении любого устройства, при этом драйвер каждого устройства по своему интерпретирует логический обьект способом, соответствующий возможностям устройства.

Контекст устройства

Контекст устройства представляет собой некоторое множество атрибутов графического вывода, в которое входит одно перо для рисования линий, одна кисть для закрашивания областей и один шрифт для вывода текстов, которые можно изменить в любой момент. Вместе взятые, атрибуты графического вывода дают полный контроль над тем, как выглядит и где выполняется графический вывод программы.

Каждый контекст устройства включает в себя 20 атрибутов графического вывода, которые приведены ниже (в скобках приведено значение по умолчанию):

- Цвет фона (белый);
- Режим фона (OPAQUE);
- Логический номер кисти (белая кисть);
- Начало координат кисти (0,0);
- Логический номер области прорисовки (вся поверхность);
- Логический номер цветовой палитры (палитра по умолчанию);

- Текущая позиция пера (0,0);
- Режим графического вывода (R2_COPYPEN);
- Логический номер шрифта (системный шрифт);
- Межсимвольный интервал (0);
- Режим отбражения (MM_TEXT);
- Логический номер пера (черное перо);
- Режим закрашивания многоугольников (альтернативный);
- Режим растяжения (черный по белому);
- Выравнивание границ текста (по левому и верхнему краям);
- Цвет текста (черный для текста и кистей с монохромным шаблоном закрашивания);
- Выравнивание строк текста (0,0);
- Протяженность окна данного экрана (1,1);
- Начало координат окна данного экрана (0,0);
- Протяженность окна экрана (1,1);
- Начало координат окна экрана (0,0).

Контекст устройства связывает программу с конкретной поверхностью рисования. Такое соединение является логическим, а не физическим. Чтобы избежать конфликтов, связанных с совместным использованием устройств, программа получает у контекста устройства "пропуск" к устройству. Работа системы пропусков зависит от типа устройства. На устройствах получения твердых копий это делается путем буферизации ввода, а на видеоустройствах - путем выделения так называемой области прорисовки, вне границ которой программа рисовать не может.

Основные атрибуты контекста устройства

Для рисовании линий самым важным атрибутом контекста устройства является перо, определяющее, как будет выглядеть линию: ее цвет, ширина и стиль (или шаблон, например, сплошная линия, пунктир и т.п.) и представляющее из себя запрос к устройству на рисование линии определенного вида.

При рисовании линий также используется атрибут - режим графического вывода, в котором можно задать логическую операцию, чтобы применить ее при выводе между новым и старым пикселями.

Для закрашивания областей используется атрибут кисть, определяющий как будет выглядеть закрашиваемая область и характеризующийся тремя характеристиками: стилем, цветом и шаблоном. Размер кисти составляет 8X8 пикселей.

При изображении текста ключевым атрибутом является шрифт. Шрифт - совокупность шаблонов для вывода текста. GDI распознает два вида шрифтов: логические и физические.

Логический шрифт описывает текст стандартным не
зависящим от внешних устройств способом. Логический шрифт
задается структурой LOGFONT. Логический шрифт - это запрос на
на текст с определенными характеристиками.
typedef struct tagLOGFONT

int lfHeight; // высотасимвола
int lfWidth; // средняяширина
int lfEscapement; // уголнаклонатекста
int lfOrientation; // уголнаклонасимвола
int lfWeight; // среднеечислопикселей/1000
BYTE lfItalic; // не 0, если курсив
BYTE lfUnderline; // не 0, если подчеркнуто
BYTE lfStrikeOut; // не 0, если вычеркнуто
BYTE lfCharSet; // набор символов ANSI, OEM
BYTE lfOutPrecision; // точнось отображения
BYTE lfClipPrecision; // точность вырезки
BYTE lfQuality; // качество печати
BYTE lfPitchAndFamily; // флаг для стиля шрифта
BYTE lfFaceName[LF_FACESIZE]; // названиешрифта
LOGFONT;

Физический шрифт - это набор шаблонов, зависящий от устройства. Он выбирается по описанию, содержащемуся в логическом шрифте и может быть аппаратно реализован.

Для задания атрибута контекста устройства используется функция SelectObject, описанная следующим образом:

HANDLE FAR PASCAL SelectObject(HDC, HANDLE);

Здесь первый параметр - логический номер контекста устройства, а второй - логический номер значения атрибута.

Для того, чтобы получить логический номер требуемого значения атрибута контекста, используется функция GetStockObject. Ее прототип:

HANDLE FAR PASCAL GetStockObject(int);

Параметром является значение атрибута контекста
устройства, обычно для удобства задаваемое в виде набора
символов.

Программист может использовать либо уже заданные значения атрибутов, либо создавать свои новые.

Перерисовка области и изображение пикселей

Из-за того, что Windows не накладывает ограничений на размеры и расположение окон приложений, могут возникнуть ситуации, что окно одного приложения перекроет окно другого приложения, поэтому при переключении между приложениями необходимо перерисовыть поврежденные области окна.

Для этой цели используется подпрограмма BeginPaint. При получении сообщения WM_PAINT, говорящее о необходимости перерисовки окна из изменения его размеров или восстановления поврежденной области, подпрограмма BeginPaint получает контекст устройства и определяет область, которую надо перерисовать.

Подпрограмма BeginPaint принимает два параметра:
логический номер окна и указатель на структуру данных

PAINTSTRUCT, Она возвращает логический номер контекста устройства, необходимый для рисования пикселя. Прототип подпрограммы имеет вид:

HDC FAR PASCAL BeginPaint(HWND, LPPAINTSTRUCT);

Структура PAINTSTRUCT определенатак:

typedef struct tagPAINTSTRUCT

HDC hdc;

BOOL fErase;

RECT rcPaint;

BOOL fRestore;

BOOL flncUpdate;

BYTE rgbReserved[16];

PAINTSTRUCT;

hdc - логический номер контекста устройства;

fErase - флаг, задающий необходимость стирания окна; rcPaint - описывает прямоугольник, ограничивающий

поврежденную область;

fRestore, flncUpdate и rgbReserved предназначены для внутреннего использования Windows.

Для отображения пикселя используется подпрограмма

SetPixel. Еепрототип:

DWORD FAR PASCAL SetPixel(HDC, int, int, DWORD);

HDC - логический контекст устройства; следующие два параметра - координаты пикселя; последний параметр - цвет.

После завершения перерисовки, программа вызывает функцию EndPaint, чтобы вернуть контекст устройства дисплея Менеджеру Окон и сообщить, что окно восстановлено. Когда тот получает контекст устройства, он восстанавливает все его атрибуты по умолчанию, и контекст устройства готов к передаче информации любой программе, которой понадобится нарисовать окно. Подпрограмма EndPaint имеетпрототип:

void FAR PASCAL EndPaint(HWND, LPPAINTSTRUCT);

Рисование линий

Каждая линия имеет начальную и конечную точку, и GDI рисует линию так, начальная точка включается в линию, а конечная исключается из линии.

GDI имеет 4 подпрограммы для рисования линий: MoveTo, LineTo, PolyLine и Arc.

Подпрограмма MoveTo помещает пару координат XY в атрибут контекста устройства, который называется текущей позицией. Подпрограммаимеетпрототип:

DWORD FAR PASCAL MoveTo(HDC, int x1, int y1);

Подпрограмма LineTo берет начальную точку из атрибута текущей позиции и рисует линию до конечной точки, передаваемой как параметр. После этого она устанавливает новое значение атрибута текущей позиции. Подпрограмма имеет прототип:

BOOL FAR PASCAL LineTo(HDC, int x1, int y1);

Подпрограмма Polyline позволяет нарисовать ломанную линию, но для ее работы необходимо предварительно создать массив координат точек. Подпрограммаимеетпрототип:

BOOL FAR PASCAL Polyline(HDC, LPPOINT points, int num);

points - указатель на массив координат, котором координаты оси абсцисс и оси ординат чередуются; num - число пар координат в массиве.

Подпрограмма Arc позволяет нарисовать дуги эллипса. Передаваемые параметры определяют прямоугольник, ограничивающий фигуру, если бы она была полной, начальную точку и конечную. Подпрограммаимеетпрототип:

BOOL FAR PASCAL Arc(HDC,int,int,int,int,int,int,int,int);

Рисование закрашенных фигур

Для рисования закрашенных фигур существует семь различных функций.

Функция Polygon соединяет расположенные последовательно точки с помощью текущего установленного в контексте устройства пера. Функцияопределенатак:

BOOL FAR PASCAL Polygon(HDC, LPPOINT, int);

HDC - логический номер контекста устройства;

LPPOINT - указатель на массив координат, соединяемых граничных точек;

Третий параметр - число соединяемых точек.

Функция PolyPolygon позволяет нарисовать за один вызов несколько многоугольников и определена так:

BOOL FAR PASCAL PolyPolygon(HDC, LPPOINT, LPINT, int);

HDC - логический номер контекста устройства;

LPPOINT - указатель на массив координат, соединяемых граничных точек всех создаваемых многоугольников;

LPINT - указатель на массив типа int. Элементы массива задают число точек в каждом многоугольнике*

Последний параметр задает количество точек в массиве LPINT, то есть количество многоугольников.

Для рисования закрашенного эллипса используется функция Ellipse. Онаопределяетсятак:

BOOL FAR PASCAL Ellipse(HDC, int, int, int, int);

HDC - логический номер контекста устройства;

Остальные параметры определяют координаты ограничивающего прямоугольника.

Функция Chord используется для рисования частичной дуги, концы которой соединены сегментом линии. Для этого также используется ограничивающий прямоугольник. Определение функции следующее:

BOOL FAR PASCAL Chord(HDC,
int, int,
int, int,
int, int,
int, int);

HDC - логический номер контекста устройства;

Следующие четыре параметры определяют координаты

ограничивающего прямоугольника.

Далее указываются координаты начальной и конечной точки сегмента линии.

Функция Pie аналогична функции Chord, только рисует не хорду, а сектор эллипса. Она определена следующим образом:

BOOL FAR PASCAL Pie(HDC,

int, int,

int, int,

int, int,

int, int);

HDC - логический номер контекста устройства;

Следующие четыре параметры определяют координаты

ограничивающего прямоугольника.

Далее указываются координаты начальной и конечной точки сектора.

Для рисования прямоугольника используется функция

Rectangle.Ее прототип:

BOOL FAR PASCAL Rectangle(HDC, int, int, int, int);

HDC - логический номер контекста устройства;

Остальные параметры определяют координаты

прямоугольника.

Если требуется нарисовать прямоугольник с закругленными краями, то имеется функция RoundRect. Онаопределената:

BOOL FAR PASCAL RoundRect(HDC,

int, int,

int, int,

int, int);

HDC - логический номер контекста устройства;

Следующие четыре параметры определяют координаты

прямоугольника.

Последние два параметра задают ширину и высоту ограничивающего прямоугольника для эллипса, используемого при скруглении угла.

Вывод текста

Для вывода текста имеется пять основных функций.

Функция TextOut предназначена для вывода одной строки. Она определена следующим образом:

BOOL FAR PASCAL TextOut(HDC, int, int, LPSTR, int);

HDC - логический номер контекста устройства;

Следующие два параметры определяют координаты точки привязки выводимой стоки текста.

LPSTR - указатель на выводимую символьную строку.

Последний параметр - число символов в строке текста.

Более мощный вариант функции TextOut представляет собой функция ExtTextOut. Она позволяет управлять интервалом между строками и вырезкой

BOOL FAR PASCAL ExtTextOut(HDC,
int, int,
WORD wOptions,
LPRECT lpRect,
LPSTR lpString,
WORD nCount,
LPINT lpDx);

HDC - логический номер контекста устройства;

Следующие два параметры определяют координаты точки привязки выводимой стоки текста.

wOptions - флаг, принимающий значение 0, ETO_CLIPPED, ETO_OPAQUE и ETO_CLIPPED|ETO_OPAQUE, позволяющий устанавливать прямоугольные области вырезки и при выводе текста затирать фон.

lpRect - указатель на структуру прямоугольника;

lpString - указатель на выводимую символьную строку.

nCount - число символов в строке текста. lpDx -
указатель на массив значений интервалов между
символами.

Функция TabbedTextOut при выводе текста распространяет знаки табуляции до позиций табуляции. Это обеспечивает удобный способ выравнивания столбцов данных. Прототип функции:

LONG FAR PASCAL TabbedTextOut(HDC,
int, int,
LPSTR, int, LPINT, int);

HDC - логический номер контекста устройства;

Следующие два параметры определяют координаты точки привязки выводимой стоки текста.

LPSTR - указатель на выводимую символьную строку.

Следующий параметр - число символов в строке текста.

LPINT - указатель на массив позиций табуляции;

Последний параметр - число элементов массиве позиций табуляции.

Функция DrawText обеспечивает некоторую возможность форматирования и переход в автоматическом режиме на новую строчку при большом количестве строк текста. Еепрототип:

int FAR PASCAL DrawText(HDC, LPSTR, int, LPRECT, WORD);

HDC - логический номер контекста устройства;

LPSTR - указатель на выводимую символьную строку.

Следующий параметр - число символов в строке текста. LPRECT - указатель на структуру прямоугольника,

определяющего позицию вывода и границы для форматирования. Последний параметр определяет режим форматирования.

Для отображения заблокированных пунктов меню и
заблокированных элементов управления блоками диалога Менеджер Окон использует функцию GrayString.

ЭЛЕМЕНТЫ ОКОННОГО ИНТЕРФЕЙСА

Окна Windows

Главное окно приложения

Каждое приложение располагается в своем собственном окне и имеет по крайней мере хотя бы одно окно - главное окно приложения. Из приложения Windows непосредственно нельзя осуществить вывод на экран, так как экран является разделяемым ресурсом, а средством его разделения являются окна. Таким образом, прежде чем отобразить что-либо на экране, нужно создать окно, и только в окне можно осуществить вывод.

Функция главного окна приложения выполняет в программе ту же роль, что функция main() программы на Си для MS-DOS. При создании окна функция библиотеки SDK выполняет специальные действия начальной подготовки в DOS, не явно осуществляемых функцией main(). Поэтому приложение не содержит функции main(), роль которой выполняет функция WinMain(), получающая управление в начальный момент загрузки приложения. Функция WinMain() выполняет следующие основные действия:

- регистрация класса окна приложения и другие
инициализации;
- создание основного окна приложения и, возможно, других, подчиненных окон;
- запуск цикла обработки сообщений, помещаемых в очередь приложения;
- завершение работы приложения при извлечении из очереди сообщения WM_QUIT.

Прототип функции WinMain выглядит так:

```
int PASCAL WinMain
(
HANDLE instance, // дескрипторпредыдущейкопии
HANDLE prevInstance, // предыдущаякопия
LPSTR cmdLine, // указательнакомандуюстроку
int cmdShow // флаг "окно открыто/закрыто"
);
```

instance - однозначно определяет каждую копию
приложения, если приложение запущено несколько раз.

prevInstance - определяет копию данного приложения, которая была последней активной копией. Если этот параметр равен 0, то других копий приложения, исполняемых в данный момент не существует.

cmdLine - дальний указатель на командную строку, оканчивающуюся нулем. Он позволяет приложениям получать данные через командную строку.

cmdShow - определяет, как приложение первоначально отображать на экране: пиктограммы (cmdShow = SW_SHOWMINNOACTIVE) или в виде открытого окна (cmdShow = SW_SHOWNORMAL). Константы

SW_SHOWMINNOACTIVE и SW_SHOWNORMAL определены во включаемом файле windows.h.

Регистрация класса окна

Любое окно принадлежит к одному из существующих классов. Класс окна должен быть создан до того, как окно будет отображено на экране. Класс окна определяет общие свойства всех окон данного класса, например: форму курсора при перемещении его в области окна или имя меню, определенного для окон этого класса.

Характеристики окна задаются при регистрации класса окна (в структуре класса окна) и при создании окна. Наиболее общие характеристики окон задаются при регистрации класса окна. Окна, создаваемые при помощи функции CreateWindiw, должны иметь зарегистрированный ранее класс окон.

Есть несколько стандартных классов окон с заранее определенными свойствами. Однако, как правило, каждое приложение регистрирует свой собственный класс с тем, чтобы можно было управлять всеми свойствами окна приложения.

Для того, чтобы зарегистрировать класс окон, следует правильно заполнить структуру типа WNDCLASS и передать эту структуру в виде параметра функции RegisterClass. Структура класса окна имеет вид:

typedef struct tagWNDCLASS

WORD style; // тип окна

LONG (FAR PASCAL *lpfnWndProc)(); // функция окна int cbClsExtra; // размер доп. памяти

int cbWndExtra; // размер доп. памяти

HANDLE hInstance; // индекс копии приложения

HICON hIcon; // индекс пиктограммы

HCURSOR hCursor; // индекс курсора

HBRUSH hbrBackground; // цвет фона окна

LPSTR lpszMenuName; // имя меню

LPSTR lpszClassName; // имя класса окна

WNDCLASS;

lpszClassName - указатель на строку, содержащую имя класса. Поскольку определенный в приложении класс доступен всем приложениям, имя класса не должно повторятся в разных приложениях.

hInstance - манипулятор копии, создающей класс окна; должно содержать индекс копии приложения.

lpfnWndProc - указатель на функцию поддержки окна. Краткое описание этой функции смотри в следующем разделе.

style - содержит набор флагов, определяющих свойства окна. По умолчанию присваивается NULL.

hBrBackground - определяет цвет фона окна.

hCursor - определяет курсор, используемый в данном окне по умолчанию.

hIcon - определяет пиктограмму (icon), которая будет отображаться при переводе окна в неактивное состояние.

lpszMenuName - указатель на имя меню окна, определенное в файле ресурсов.

cbClsExtra - определяет число байт, которое необходимо дополнительно запросить у Windows под эту структуру. Этот обьем памяти будет зарезервирован в конце структуры для всех окон данного класса.

clWndExtra - определяет число байт, которое необходимо дополнительно запросить у Windows для размещения всех структур, создаваемых совместно с данным классом.

После определения полей структуры WNDCLASS необходимо зарегистрировать класс при помощи функции RegisterClass.

BOOL FAR PASCAL RegisterClass(LPWNDCLASS winClass);

Если регистрация класса прошла успешно, то возвращаемое значение TRUE, в противном случае - FALSE.

При регистрации класса окна Windows копирует структуру, описывающую класс окна, в системную область, чтобы другим копиям приложения уже не надо было ее регистрировать.

Функция окна приложения

Функция окна приложения занимается тем, что обрабатывает все сообщения для окон данного класса. Это функция всегда

вызывается неявно Windows при поступлении сообщений в окно, за

которым оно закреплено. Функция окна имеет вид:

long far PASCAL WndProc (HWND hwnd,

WORD msg,

WORD wParam,

LONG lParam);

hwnd - логический номер окна, идентифицирующий окно, связанное с приложением;

msg - идентификатор приложения;

wParam и lParam определяют дополнительную информацию и зависят от типа сообщения.

Для облегчения работы программиста существует

специальная функция обработки сообщений DefWindowProc с теми

же параметрами, которая производит стандартную обработку всех

сообщений. Кроме того она играет ключевую роль в формировании

информационных потоков сообщений Windows, и ее указание в

функции окна обязательно.

Создание окна

Создание окна производится при помощи функции

CreateWindow. Она создает окно, имеющее указанный тип и принадлежащее к указанному классу. Прототип функции имеет вид:

HWND FAR PASCAL CreateWindow

(

LPSTR, // имя класса окна

LPSTR, // заголовок окна

DWORD, // тип окна

int, // X-координата
int, // Y-координата
int, // Ширина окна
int, // Высота окна
HWND, // Дескриптор копии-родителя
HMENU, // Дескриптор меню
HANDLE, // Дескриптор копии
LPSTR // Дополнительная информация
);

Тип окна является комбинацией битовых флагов,

определяющих стиль окна. Возможные значения типа окна рассмотрены ниже.

Дескриптор копии-родителя определяет порождающее окно. Он определяет, где существует окно и может ли окно быть автоматически показано/скрыто/уничтожено (для всех типов окон). Когда показывается, скрывается или уничтожается, все порожденные им окна разделяют его судьбу

Дескриптор меню позволяет определить меню, которое будет изображаться в окне.

Дескриптор копии позволяет идентифицировать владельца окна, то есть указывает Windows, какой именно экземпляр программы создал окно. В результате Windows получает возможность правильно установить регистр сегмента данных для

инициализации окна.

Последний параметр функции позволяет передать указатель на данные в оконную процедуру. Указатель передается с самым первым сообщением WM_CREATE, что необходимо для обеспечения данных при инициализации окна.

В случае успешного создания окна функция CreateWindow возвращает индекс окна.

Отображение и обновление окна

Окно не отображается на экране сразу после создания; для отображения окна используется функция ShowWindow. Еепрототип:

BOOL FAR PASCAL ShowWindow(HWND wnd, int cmdShow);

wnd - дескриптор отображаемого окна;

cmdShow - определяет, как окно первоначально будет отображаться на экране:

SW_SHOWNORMAL - обычное окно;

SW_SHOWMINIMIZED - минимизированное в виде пиктограммы; SW_SHOWMAXIMIZED - максимизорованное на весь экран;

Для обновления окна используется функция UpdateWindow. Ее прототип выглядит так:

void FAR PASCAL UpdateWindow(HWND wnd);

Типы окна

Тип окна задается 32-битовым целым числом, которое представляет собой комбинацию битовых флагов, определяющих различные свойства окна.

WS_OVERLAPPED - перекрывающееся окно. Перекрывающиеся окна - это основной наиболее универсальный тип окон Windows. Главное окно приложения обычно имеет такой вид.

WS_POPUP - вспомогательные окна. Они используются чаще всего для отображения окон диалога. Вот некоторые свойства вспомогательных окон:

- если такое окно имеет родительское окно, то всегда отображаются поверх всех окон на экране, даже когда пользователь делает активным другое окно;
- вспомогательные окна не имеют заголовка и часто должны иметь фиксированный размер.

WS_CHILD - дочернее окно. Окна такого типа создаются, если у приложения есть главное (а значит и перекрывающее окно) и связаны некоторыми характеристиками с тем окном из которого были вызваны. Все органы управления также являются дочерними окнами. Вот некоторые их свойства:

- дочерние окна никогда не отображаются вне своего родительского окна ни в раскрытом виде, ни в виде пиктограммы;
- координаты дочерних окон отчитываются от верхнего левого угла рабочей области окна-родителя и при перемещении последнего, дочерние окна перемещаются вместе с ним;
- дочернее окно никогда не может стать активным окном.

WS_MINIMIZE - создаваемое окно будет отображено в виде пиктограммы.

WS_VISIBLE - Окно становится видимым сразу после создания. Используется для диалоговых окон.

WS_DISABLED - создается неактивное окно.

WS_CLIPSIBLINGS - исключение областей, занимаемых
другими дочерними окнами из изменяемой области дочернего окна.
Используется только для дочерних окон.

WS_CLIPCHILDREN - исключение областей, занимаемых
другими дочерними окнами при изменении рабочей области
родительского окна. Используется только для родительских окон.

WS_MAXIMIZE - создаваемое окно будет отображено в максимально возможном виде.

WS_CAPTION - окно имеет рамку и заголовок, а следовательно пользователь может перемещать его при помощи мыши.

WS_BORDER - окно имеет широкую рамку без заголовка. Используется при создании диалоговых окон.

WS_DLGFRAME - окно имеет тонкую рамку без заголовка.

WS_VSCROLL - окно имеет вертикальную полосу просмотра.

WS_HSCROLL - окно имеет горизонтальную полосу просмотра.

WS_SYSMENU - окно имеет системное меню.

WS_THICKFRAME - создаваемое окно имеет рамку существенно заметной толщины.

WS_MINIMIZEBOX - окно имеет кнопку минимизации.

WS_MAXIMIZEBOX - окно имеет кнопку максимизации.

Построение меню

Для создания меню нужно проделать:

1. Задать структуру меню в файле ресурсов,
последовательно определив пункты меню в виде текстовых строк.
2. Каждому пункту меню поставить в соответствие
уникальный идентификатор.
3. Указать имя меню в структуре класса окна.

Определение меню

Определение меню в файле ресурсов должно иметь вид:

MenuName MENU [опции загрузки][опции памяти]
BEGIN
MENUITEM "Item1" IDM_Item1 [, опции]
MENUITEM "Item2" IDM_Item3 [, опции]
...
POPUP "Item3" [, опции]
BEGIN
MENUITEM "Item3-1" IDM_Item3-1 [, опции] MENUITEM "Item3-2" IDM_Item3-2 [, опции] ...
END
END

MenuName - имя ресурса меню для обращения из подпрограммы.

Опции загрузки определяет как следует поступить с ресурсом при загрузке приложения на выполнение - сразу загрузить или при неоходимости.

Опции памяти определяют, как Windows должна обращаться с сегментом памяти, куда загружается ресурс.

Пункты меню определяются между словами BEGIN и END. Они могут быть двух видов: MENUITEM и POPUP. Пункт типа MENUITEM является конечным пунктом меню. При выборе этого пункта функции окна сообщения передается сообщение WM_COMMAND с идентификатором пункта меню в качестве параметра. Пункт типа POPUP является заголовком подменю.

Опции пункта меню могут комбинироваться. В качестве опций пункта меню могут быть следующие значения:

GRAYED - пункт меню не активен. Текст пункта меню отображается в сером цвете.

INACTIVE - пункт меню не активен. Текст пункта меню отображается также как и в других пунктах.

MENUBREAK - этот и следующий за ним пункты меню отображаются в новом столбце (если указан для главного меню, то в новой строке).

MENUBARBREAK - этот и следующий за ним пункты меню отображаются в новом столбце (если указан для главного меню, то в новой строке); предыдущий и новый столбец разделяются вертикальной чертой.

CHECKED - пункт меню помечен галочкой, помещенной слева от него. Не действует для пунктов главного меню.

SEPARATOR - определяет разделитель, который выделяет в группы связанные списки меню.

HELP - пункт меню выравнивается по правой стороне меню.

Обьекты диалога

Обьекты диалога, в состав которых входят кнопки, комбинированные блоки, элементы управления редактированием, блоки списков, полосы прокрутки, статические элементы, являются с точки зрения Windows обычными дочерними окнами.

Взаимодействие между родительским окном (окном диалога) и обьектами диалога осуществляется посредством сообщений. Когда пользователь производит какое-либо действие с обьектом диалога, функции окна родителя передается сообщение WM_COMMAND, в качестве параметра wParam которого передается индекс обьекта диалога, а в качестве параметра lParam - специальная дополнительная информация.

Для того, чтобы сконструировать обьект диалога нужно:

1. Зарегистрировать класс окна диалога.
2. Создать дочернее окно функцией CreateWindow, указав зарегистрированный класс окна.
3. В функции окна обьекта диалога определить дескриптор дочернего окна при помощи функции GetParent.
4. По тому или иному действию пользователя уведомлять родительское окно соответствующими сообщениями при помощи функции SendMessage.

Для обьектов диалога как дочерних окон не требуется регистрировать класс окна - в Windows определены стандартные классы окон - обьектов диалога: "button", "edit", "scrollbar", "listbox" и пр.

При использовании стандартных классов Windows для создания обьекта диалога нужно вызвать только функцию CreateWindow.

Рассмотрим основные обьекты диалога.

Кнопки и переключатели

Кнопка "Button" обычно используется для осуществления каких - либо немедленных действий, без переключения или включения/выключения каких-либо опций.

Переключатель "И" ("CheckBox") используется как
переключатель опций вкл/выкл. Будучи соединенными в группу
переключатели реализуют логику "И".

Переключатель "ИЛИ" ("RadioButton") используется как переключатель опций вкл/выкл. Будучи соединенными в группу переключатели реализуют логику "ИЛИ".

Вокруг этих трех основных типов имеются некоторые вариации. Ниже приводится описание стандартных классов кнопок
и переключателей.

BS_PUSHBUTTON - определяет кнопку с жирной рамкой.

BS_DEFPUSHBUTTON - определяет кнопку с жирной рамкой. Обычно используется для определения действия по умолчанию.

BS_CHECKBOX - определяет квадратик, имеющий два

состояния: отмеченное (перечеркнут крестиком) и не отмеченное (квадратик пуст). В момент отметки рамка квадрата выделяется жирной линией.

BS_AUTOCHECKBOX - тоже, что предыдущий, только состояние кнопки при отметке отслеживается автоматически.

BS_RADIOBUTTON - определяет круглую кнопку, которая может быть нажата (внутри окружности жирная точка) и отпущена. Справа от кнопки может быть любой поясняющий текст.

BS_AUTORADIOBUTTON - тоже, что и предыдущее, только при отметке кнопки пользователем ранее сделанная отметка автоматически снимается.

BS_3STATE - тоже, что и BS_CHECKBOX, только добавлено состояние, что действие или свойство не действительно (обозначается штриховкой кнопки).

BS_AUTO3STATE - тоже, что и предыдущее, только смена состояний поддерживается автоматически.

BS_GROUPBOX - определяет рамку, охватывающую другие обьекты диалога.

BS_OWNERDRAW - обьект диалога, определяемый пользователем, который полностью берет на себя работу с ним.

BS_LEFTTEXT - используется для выравнивания пояснительного текста по левой стороне кнопок.

Кнопки посылают функции окна родителя сообщение WM_COMMAND, в качестве параметра lParam указывается дескрипотор органа управления и код нотификации, служащий для определения того, какое действие произвел пользователь с кнопкой.

Статические обьекты диалога

Статические обьекты диалога используются для отображения текста и отрисовки оформительских примитивов. Обьекты этого класса могут быть созданы указанием функции CreateWindow имени класса "static". Определены следующие классы статических обьектов:

SS_LEFT - текст, выравненный по левому краю;

SS_CENTER - центрированный текст;

SS_RIGHT - текст, выравненный по правому краю;

SS_ICON - пиктограмма;

SS_BLACKRECT - прямоугольник цвета рамки фона;

SS_GRAYRECT - прямоугольник цвета фона окна;

SS_WHITERECT - прямоугольник цвета окна;

SS_BLACKFRAME - рамка цвета рамки окна;

SS_GRAYFRAME - рамка цвета фона окна;

SS_WHITEFRAME - рамка цвета окна;

SS_USERITEM - обьект, определяемый пользователем.

Редактор

Редактор "edit" позволяет создать дочернее окно и редактировать в нем текст. По умолчанию редактор может редактировать только одну строку. Типы классов редактора приведены ниже.

ES_LEFT - текст, выравненный по левому краю;

ES_CENTER - центрированный текст;

ES_RIGHT - текст, выравненный по правому краю;

ES_MULTILINE - определяет многостраничный редактор;

ES_UPPERCASE - символы переводятся в верхний регистр;

ES_LOWERCASE - символы переводятся в нижний регистр;

ES_PASSWORD - стиль для ввода паролей;

ES_AUTOHSCROLL - автоскроллинг вправо на 10 позиций,

если курсор находится в конце строки;

ES_AUTOVSCROLL - при нажатии ENTER в конце страницы текст автоматически сдвигается вверх на один экран.

В качестве параметров сообщения WM_COMMAND передаются идентификатор дочернего окна-редактора, индекс дочернего окна и код сообщения. Код сообщения может быть:

EN_SETFOCUS - редактору текста передали фокус ввода;

EN_KILLFOCUS - редактор текста потерял фокус ввода;

EN_CHANGE - содержимое редактора изменено;

EN_ERRSPACE - переполнение буфера редактора;

EN_HSCROLL - нажата клавиша горизонтального просмотра;

EN_VSCROLL - нажата клавиша горизонтального просмотра.

Окно список

Окно список "listbox" представляет собой прямоугольник, внутри которого находится листаемый список из текстовых строк. Пользователь может выделить строки списка при помощи курсора. Окно-список используется для просмотра и выбора элементов древовидного списка. Приведем типы окна списка.

LBS_NOTIFY - родительское окно получает информацию о любом действии пользователя в списке;

LBS_SORT - строки сортируются по алфавиту; LBS_MULTIPLESEL - множественный выбор с переключением

выбора для каждой строки;

LBS_OWNERDRAWFIXED - отображение содержимого списка возлагается на функцию родительского окна, все элементы списка могут иметь разную высоту;

LBS_OWNERDRAWVARIABLE - отображение содержимого списка возлагается на функцию родительского окна, все элементы списка могут иметь разную высоту;

LBS_HASSTRINGS - определяет пользовательское окно-список с произвольными строками;

LBS_USETABSTOPS - символы табуляции заменяются на пробелы;

LBS_MULTICOLUMN - определяет многостолбцовый список; LBS_EXTENDEDSEL - в окне-списке можно делать

множественный выбор с помощью мыши и клавиши Shift.

В качестве параметров сообщения WM_COMMAND передаются идентификатор дочернего окна-редактора, индекс дочернего окна и код сообщения. Код сообщения может быть:

LBN_ERRSPACE - списку не хватает памяти;

LBN_SELCHANGE - изменен выбор элемента;

LBN_DBLCLK - выбор двойным нажатием кнопки мыши.

Комбинированный список

Комбинированный список представляет сроку

редактирования, к которой привешено окно-список. Стандартные

типы обьекта:

CBS_SIMPLE - список отображается все время, и текущее выделение отслеживается среди элементов списка;

CBS_DROPDOWN - то же, что и предыдущее, но список не отображается, пока пользователь на нажмет на левую кнопку мыши;

CBS_DROPDOWNLIST - то же, но строка редактирования заменяется на статическую текстовую строку, ее нельзя редактировать;

CBS_OWNERDRAWFIXED - элементы списка отрисовываются пользователем, их высота одинакова;

CBS_OWNERDRAWVARIABLE - элементы списка отрисовываются пользователем, их высота одинакова;

CBS_AUTOHSCROLL - горизонтальная прокрутка в строке редактирования;

CBS_SORT - сортировка автоматическая элементов списка.

ОБМЕН ДАННЫМИ МЕЖДУ ПРИЛОЖЕНИЯМИ

Средства обмена данными между приложениями

Одним из средств, обеспечивающим программную

совместимость, является механизм обмена данными между

различнами приложениями. Специальный почтовый ящик (clipboard)

Windows позволяет пользователю переносить информацию из одного приложения в другое, не заботясь об ее форматах и представлении.

В отличие от профессиональных операциональных

операционных систем, где механизм обмена данными между программами доступен только программисту, в Windows это делается очень просто и наглядно для пользователя.

Механизм обмена данных между приложениями - жизненно важное свойство многозадачной среды. И в настоящее время

производители программного обеспечения пришли уже к выводу,

что для переноса данных из одного приложения в другое

почтового ящика уже недостаточно. Появился новый, более

универсальныймеханизм - OLE (Object Linking and Embedding)

- Встроенная объектная связь, который позволяет переносить из одного приложения в другое разнородные данные. Например, с помощью этого механизма данные, подготовленные в системе сетевого планирования Time Line for Windows (Symantec), можно переносить в текстовый процессор Just

Write (Symantec), а затем, скажем, в генератор приложений Object Vision (Borland). Правда, это уже нестандартное средство Microsoft Windows, но тем не менее реализация OLE стала возможной именно в Windows.

Кроме механизма почтового ящика, предназначенного, в основном, для пользователя, программисту в Windows доступны специальные средства обмена данными между приложениями.

Программным путем можно установить прямую связь между задачами, например, принимая данные из последовательного порта, автоматически помещать их, скажем, в ячейки электронной таблицы Excel, средствами которой можно тут же отображать сложные зависимости в виде графиков или осуществлять их обработку в реальном режиме времени (этот механизм носит название динамического обмена данными - Dynamic Data Exchange, DDE).