16. Основы программирования пользовательского интерфейса на базе Win32 API

Стремление сделать прикладные программы как можно более удобными для рядового пользователя приводит к тому, что большая часть кода приложения реализует именно интерфейс с пользователем. В течение 90-х годов сформировался типичный оконный стиль взаимодействия с приложениями, основанный на использовании стандартного набора управляющих элементов (УЭ), меню и диалоговых окон. Системы Windows традиционно поддерживают 6 базовых УЭ и около 15 дополнительных. К основным УЭ относятся: кнопки разных типов, одно- и многострочные текстовые редакторы, простые и комбинированные списки, простой статический текст, полосы прокрутки.

Каждый УЭ с точки зрения системы является дочерним окном, прикрепленным к какому-либо родителю. В качестве родительского окна может выступать главное окно приложения или подчиненное окно любого типа, но сами УЭ родителями быть не могут. Каждый базовый УЭ является представителем своего стандартного оконного класса, в котором задаются общие свойства данного типа УЭ, в том числе – предопределенная стандартная оконная функция, реализующая стандартное поведение УЭ. Таким образом, ОС отвечает за стандартное поведение своих УЭ и их прорисовку на экране.

Базовые УЭ можно создавать с помощью системного вызова CreateWindow, который в случае успеха возвращает дескриптор созданного УЭ. Пример создания управляющих элементов:

```
MyButton1 := CreateWindow (‘BUTTON’, ‘Просто Кнопка!’,
                bs_PushButton or ws_Child or ws_Visible, 50, 50, 100, 50,
                ParentWin, id_But1, …);
MyEdit1 := CreateWindow(‘EDIT’, 0, es_MultiLine or ws_Child or
                ws_Visible, 200, 50, 100, 200, ParentWin, 0, …);
```

Вызовы CreateWindow, которые создают УЭ, можно располагать в приложении в двух местах: в главной функции вслед за созданием главного окна и в оконной функции в обработчике какого-либо события.

Важным аспектом взаимодействия приложения со своими УЭ является отдача этим элементам определенных команд, таких как “добавить новый элемент в список”, “удалить выбранный элемент из списка”, “вернуть номер выбранного в списке элемента”, “вернуть введенный в редакторе текст”, “установить состояние включено/выключено для кнопки с независимой фиксацией” и т.д. Для отдачи команд приложение посылает управляющим элементам соответствующее сообщение. Фактически, основная работа приложения с УЭ строится на использовании механизма посылки сообщений. Для этого можно использовать одну из двух системных функций – либо SendMessage, либо PostMessage. Различие в работе этих функций состоит в следующем:

- SendMessage посылает сообщение сразу в оконную функцию приложения и ждет обработки этого сообщения оконной функцией.
- PostMessage посылает сообщение в очередь сообщений данного приложения и не ждет пока это сообщение будет обработано.

Чаще используется вызов SendMessage. Оба этих вызова имеют 4 формальных параметра:

- дескриптор УЭ, которому посылается сообщение (кому?);
- константа, определяющая действие, которое должен выполнить УЭ; для каждого типа УЭ определен свой набор констант;
- дополнительная информация о выполняемом действии (два параметра).

Рассмотрим примеры кодирования наиболее типичных действий:

- добавление элемента-строки в список:

```
SendMessage (MyList, lb_addString, 0, longint(строка));
```

при этом новая строка добавляется либо в конец не отсортированного списка, либо в соответствующую позицию отсортированного списка;

- запрос номера текущего выбранного элемента в списке:

Nomer := SendMessage (MyList, lb_GetCurSel, 0, 0);

- удаление выбранного элемента из списка (элемента с номером Nomer):

SendMessage (MyList, lb_DeleteString, Nomer, 0);

- получение из списка текста выбранной строки с номером Nomer:

SendMessage (MyList, lb_GetText, Nomer, строка);

- установка состояния включено/выключено у независимого переключателя:

SendMessage (MyCheck1, bm_SetCheck, 1/0, 0);

- запрос состояния переключателя:

Sost := SendMessage (MyCheck1, bm_GetCheck, 0, 0);

Более подробное описание констант-действий можно получить во встроенной помощи.

Теперь рассмотрим некоторые возможности по управлению системой меню с помощью API-функций. Всего Win32 API предоставляет около 40 подобных функций. Естественно, начинать надо с функций, позволяющих создать в приложении систему меню программным способом.

Программное создание меню выполняется по принципу “снизу - вверх”, т.е. сначала создаются подменю самого нижнего уровня, которые после этого подключаются к соответствующим пунктам более высокого уровня. В самую последнюю очередь создается полоса основного меню приложения, к которому подключаются подменю более низкого уровня, а уж затем вся созданная система меню подключается к приложению.

В качестве примера рассмотрим создание двухуровневого меню.

Шаг 1. Создание двух пустых подменю нижнего уровня (функция CreatePopUpMenu):

SubMenu1 := CreatePopUpMenu; // SubMenu1 и SubMenu2 – дескрипторы

SubMenu2 := CreatePopUpMenu; // подменю, переменные типа HMenu

Шаг 2. Заполнение созданных подменю элементами. Каждый элемент добавляется с помощью своего API-вызова AppendMenu, принимающего 4 параметра:

- дескриптор меню, в которое добавляется новый элемент;
- стиль добавляемого элемента, определяемый как комбинация (с помощью операции логического сложения) следующих основных флагов:
  - mf_string – элемент является текстовой строкой;
  - mf_enabled – элемент является активным;
  - mf_disabled – элемент не активный;
  - mf_grayed – элемент показан серым цветом, что говорит о его неактивности в данный момент;
  - mf_PopUp – используется при добавлении элемента-подменю;
- уникальный идентификатор элемента (например, idm_About);
- текст элемента (например, ‘About’, ‘Open’).

Например:

AppendMenu (SubMenu1, mf_string OR mf_enabled, 100, ‘Открыть’);

AppendMenu (SubMenu2, mf_string OR mf_grayed, idm_About, ‘О программе’);

Шаг 3. Создание основного меню с помощью вызова CreateMenu:

MainMenu := CreateMenu; // MainMenu – дескриптор основного меню

Шаг 4. Присоединение к основному меню созданных подменю с помощью вызова AppendMenu, который имеет те же самые параметры, но со следующими отличиями:

- во втором параметре надо использовать флаг mf_PopUp;
- третий параметр – это дескриптор присоединяемого подменю (например, SubMenu1).

Например:

AppendMenu (MainMenu, mf_PopUp OR mf_enabled, SubMenu1, ‘Файл’);

Шаг 5. Присоединение созданного меню к главному окну приложения с помощью вызова SetMenu, принимающего два параметра:

- дескриптор окна, к которому присоединяется меню;
- дескриптор присоединяемого меню.

В этом случае в описании оконного класса в поле lpszMenuName ничего задавать не надо.

Шаг 6. Прорисовка полосы меню выполняется после отображения главного окна с помощью вызова DrawMenuBar, который имеет только один параметр - дескриптор окна.

После создания меню можно организовать динамическое управление его структурой, что очень распространено в современных приложениях, поскольку позволяет в каждый момент времени предоставлять пользователю только актуальный набор действий. Для этого можно использовать следующие функции:

1. Изменение состояния элемента - **EnableMenuItem** с параметрами:
   - дескриптор меню или подменю, которому принадлежит изменяемый элемент;
   - либо идентификатор элемента, либо его порядковый номер (отсчет – с нуля), в зависимости от третьего параметра;
   - комбинация флагов из следующего набора:
     - **mf_ByCommand** – элемент задается идентификатором (установлен по умолчанию);
     - **mf_ByPosition** – элемент задается порядковым номером;
     - **mf_enabled** – элемент активен;
     - **mf_disabled** – элемент не активен;
     - **mf_grayed** – элемент показан серым цветом.
2. Изменение текста элемента – **ModifyMenu** с параметрами:

- дескриптор меню или подменю, которому принадлежит изменяемый элемент;
- либо идентификатор элемента, либо его порядковый номер (отсчет – с нуля), в зависимости от третьего параметра;
- комбинация флагов **mf_ByCommand**, **mf_ByPosition**, **mf_String** (по умолчанию используются первый и третий флаги);
- повторение второго параметра;
- текст элемента.

3. Добавление нового элемента **в конец** заданного набора – **AppendMenu.**
4. Добавление нового элемента **перед заданным** элементом – **InsertMenu** с пятью параметрами, аналогичными функции **AppendMenu** (добавлен второй параметр – идентификатор элемента, перед которым должен быть добавлен новый элемент).
5. Удаление элемента – **DeleteMenu** с тремя параметрами: дескриптор подменю, идентификатор удаляемого элемента, флаги (описаны выше).

Приложение должно уметь перехватывать сгенерированные системой сообщения от команд меню и выполнять их обработку. При выборе пользователем рабочего пункта меню генерируется **командное сообщение** wm_Command. В этом сообщении в поле wParam передается идентификатор данной команды, назначенный ей при создании меню (например, idm_About или 200). Для обработки данной команды в оконную функцию вводится обработчик сообщения wm_Command с анализом поля wParam:

```
case Mess of
    wm_Command : case wParam of
                     100 : …. ;
                     103 : …. ;
                     idm_About : …. ;
                     idm_Exit : ….. ;
                 end;
```

end;

Далее кратко рассмотрим использование в приложениях клавиатуры и мыши. В большинстве случаев события от клавиатуры направляются в окно активного в данный момент приложения. Основные клавиатурные события:

- wm_KeyDown и wm_KeyUp генерируются при нажатии и отпускании любой клавиши (в том числе и алфавитно-цифровой);
- wm_Char генерируется при нажатии только алфавитно-цифровой клавиши.

При возникновении события wm_Char наибольший интерес для программы представляет код нажатой клавиши. Этот код передается в поле wParam данного сообщения. Для обработки сообщения wm_Char надо ввести его в оконную функцию и проанализировать значение поля wParam. Аналогично, поле wParam сообщения wm_KeyDown содержит внутренний код нажатой клавиши. Для удобства работы внутренние коды всех функциональных клавиш заменяются специальными текстовыми константами вида vk_F1, vk_F10, vk_Escape, vk_Return (соответствует нажатию клавиши Enter). Такие текстовые константы принято называть **виртуальными кодами** (vk – virtual key).

Фрагмент оконной функции для обработки функциональных и алфавитно-цифровых клавиш:

```
case Message of
    wm_KeyDown :        // обработка функциональных клавиш
        case wParam of
            vk_F1 : … ;
            vk_Escape : … ;
        end;
    wm_Char :           // обработка алфавитно-цифровых клавиш
        Case wParam of
            ‘a’ .. ‘z’ : … ;
            ‘к’, ‘л’, ‘м’ : … ;
```

```
                end;
        end;
```

Кроме перечисленных, можно использовать события wm_SysKeyDown/ wm_SysKeyUp, которые генерируются при нажатии комбинации клавиш вместе с управляющими клавишами типа Alt.

В отличие от клавиатурных событий “мышиные” события направляются в то приложение, над окном которого в данный момент находится курсор мыши. Наиболее часто используются следующие “мышиные” события:

- wm_MouseMove генерируется при перемещении мыши;
- wm_LButtonDown и wm_LButtonUp генерируются при нажатии и отпускании левой кнопки мыши;
- wm_RButtonDown и wm_RButtonUp – аналогично для правой кнопки.

При обработке этих событий наибольший интерес представляют координаты текущего положения мыши. Во всех перечисленных сообщениях координаты текущего положения передаются в параметре lParam. Этот параметр имеет размер 4 байта, причем младшие 2 байта несут значение координаты x, а старшие 2 байта - значение координаты y. Для выделения из lParam нужных частей x и y можно использовать специальные стандартные функции LoWord(lParam) и HiWord(lParam).

Таймер – это системный объект, который способен через заданные интервалы времени генерировать специальные сообщения. Приложение может создать несколько таймеров с разными интервалами генерации сообщений. Для использования таймера достаточно двух API-функций:

- SetTimer создает новый таймер с заданными параметрами;
- KillTimer прекращает работу указанного таймера.

Функция SetTimer принимает 4 параметра:

- первый - дескриптор окна, создающего данный таймер;
- второй - порядковый номер созданного таймера (идентификатор);
- третий - интервал в миллисекундах генерации сообщений;
- четвертый обычно задается как nil.

Вызов KillTimer имеет 2 параметра: дескриптор окна и номер уничтожаемого таймера.

Каждый созданный таймер через заданный интервал времени генерирует сообщение wm_Timer. Это сообщение надо ввести в оконную функцию и реализовать необходимый обработчик. Если приложение одновременно использует несколько таймеров, то при обработке сообщения wm_Timer можно узнать номер таймера, который сгенерировал данное сообщение. Этот номер передается в поле wParam сообщения wm_Timer.