ОСНОВНІ ТИПИ ПРОГРАМ НА ОСНОВІ КЛАСІВ MFC

1. Архітектура “Документ/вигляд”.
2. Програми з одним документом (SDI).
3. Програми з багатьма документами (MDI).
4. Програми на основі діалогу.

Архітектура “Документ/вигляд”

До основних типів програм, що будуються під Windows на основі бібліотеки MFC, можна віднести:

- одно-документні програми (SDI);
- багато-документні програми (MDI);
- програми на основі діалогу.

Перші два типи програм будуються на основі архітектури „Документ/Вигляд”. В основі цієї архітектури лежать три глобальні поняття *рамка*, *документ* та *вигляд*. Розглянемо детальніше ці поняття.

Документ. Під „документом” фірма Microsoft розуміє ті дані, що обробляються програмою. Під даними можна розуміти, що завгодно; звичайний текст, зображення, структуровані дані та інше. Ці дані зберігаються в основному в постійній пам’яті комп’ютера і відображаються у рамці програми.

Вигляд. Це спеціальне вікно в якому відображається вміст документу та за допомогою якого відбувається взаємодія з користувачем.

Таким чином спосіб зберігання даних в постійній пам’яті не впливає на спосіб їх представлення користувачу.

Рамка. В архітектурі „Документ/Вигляд”, рамка виконує роль посередника між документом та його виглядом. В її завдання входить: створити відповідні, для кожного з типів документів, вікна виглядів; координувати взаємодію між документами різних типів та їх виглядами. Поняття *рамка* найчастіше асоціюється з основним вікном програми.

Рамка в загальному складається з основного вікна де розміщено загальні елементи керування (меню, піктограми інструментів, рядок статусу і т.п.) та клієнтської області в якій міститься дочірнє вікно вигляду (SDI- програма) або дочірні вікна (MDI- програма). На рис. __ показано взаємозв’язок між об’єктами рамки, документу та вигляду.

В архітектурі „Документ/Вигляд” за взаємодію користувача з рамкою відповідає операційна система. А керування відображенням документа повністю лягає на плечі розробника.

Для координації створення та взаємодії між трьома основними об’єктами архітектури в бібліотеці класів MFC використовується спеціальні класи шаблонів документів, які створюються та керуються класом „програма”. Таким чином архітектура „Документ/Вигляд” охоплює наступні основні класи:

- CWinApp – створення єдиного об’єкту програми.
- CFrameWnd – клас створення основного вікна програми, базовий для класів CMDIFrameWnd та CMDIChildWnd.

- CDocTemplate – базовий абстрактний клас для створення шаблонів документів: для однодокументних програм від нього створюється клас CSingleDocTemplate; для багатодокументних програм - клас CMultiDocTemplate.
- CDocument – клас для створення документу.
- CView – базовий клас, що разом зі своїми нащадками – CctrlView, CEditView, CListView та іншими, відповідає за відображення даних документу та взаємодіє з користувачем.

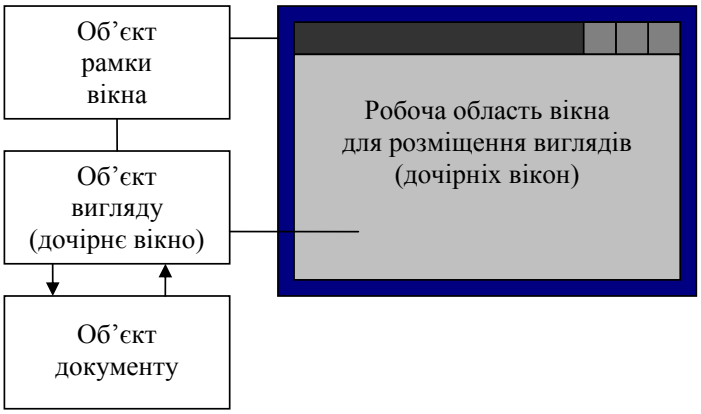

Рис. Взаємозв’язок між об’єктами рамки, документу та вигляду

Шаблон документу. Програма під Windows може одночасно працювати з багатьма різними типами документів, але їх (ці типи) необхідно зареєструвати при ініціалізації примірника програми. Для цього використовується шаблон (причому для кожного типу свій).

Клас шаблону документу призначений для організації взаємодій між класами:

- документу;
- вигляду;
- рамки.

Конструктор класу, фіксує та зв’язує між собою типи класів документу, вигляду та рамки.

```
CDocTemplate:: CDocTemplate(UINT nIDResouce, CRuntimeClass *pDocClass,
        CRuntimeClass *pFrameClass, CRuntimeClass *pViewClass);
```

де nIDResouce – ідентифікатор ресурсів (меню, акселератори, рядок описання типу шаблону), що асоціюються з даним типом документу; pDocClass, pFrameClass, pViewClass – вказівними типу CRuntimeClass на документ, рамку та вигляд, відповідно.

У клас шаблону документу CDocTemplate реалізовано декілька чисто-віртуальних функцій (GetFirstDocPosition, GetNextDoc, OpenFileName та ін.), тому цей клас є абстрактним і, як наслідок, його не можна використовувати безпосередньо. Найчастіше використовуються його класи-нащадки CSingleDocTemplate та CMultiDocTemplate, допомогою яких створюються одно- та багато-документні програми, відповідно. При потребі створення програми, що суттєво відрізняється від архітектури „Документ/Вигляд” необхідно створити свій шаблон документу на основі CDocTemplate, в якому перевизначити всі чисто-віртуальні функції.

Реєстрація шаблону документу та зв’язування документу, рамки та вигляду відбувається в методі InitInstance класу CWinApp.

Клас CSingleDocTemplate. Визначає шаблон документу для одно-документного інтерфейсу. В SDI-програмах основна рамка найчастіше є основним вікном програми, тобто одночасно може бути відкритий тільки один документ (хоча це не обов’язково). У своїй захищеній частині клас містить член, в якому зберігається вказівник на документ, що приєднаний до програми. В касі перевизначено всі чисто-віртуальні функції базового класу, що робить його придатним для подальшого використання.

Клас CMultiDocTemplate. Визначає шаблон документу для багато-документного інтерфейсу. В MDI-програмах основна рамка використовується як робоче місце, в якому користувач може відкрити довільну кількість документів одного чи різних типів.

У класі визначено два загальнодоступних члени, що відповідають за спільне використання меню та таблиці акселераторів:
HMENU CMultiDocTemplate::m_hMenuShared;
HACCELCMultiDocTemplate::m_hAccelTables;
Для збереження списку одночасно відкрити документів використовується захищений член:
CPtrList CMultiDocTemplate::m_docList;

Доступ до елементів цього списку виконується через методи GetFirstDocPosition та GetNextDoc.

Конструктор класу має такі ж самі параметри як і конструктор базового класу.

При реєстрації та зв’язування документу, рамки та вигляду в SDI-програмі застосовується метод AddDocTemplate класу CWinApp. Нижче наведено приклад уривку коду функції InitInstance в якому реєструється два різних шаблона для MDI-програми.

```
   …
CmulteDocTemplate* pDocTemplate;
// створення об’єкту шаблону для роботи з текстом
pDocTemplate = new CMultiDocTemplate( IDR_NOTETYPE,
        RUNTIME_CLASS(CNoteDoc),
        RUNTIME_CLASS(CNoteFrame),
        RUNTIME_CLASS(CNoteView));
// додавання шаблону до списку
AddDocTemplate(pDocTemplate);

// створення об’єкту шаблону для роботи з текстом
pDocTemplate = new CMultiDocTemplate( IDR_DRAWTYPE,
        RUNTIME_CLASS(CDrawDoc),
        RUNTIME_CLASS(CDrawFrame),
        RUNTIME_CLASS(CDrawView));
// додавання шаблону до списку
```

```
AddDocTemplate(pDocTemplate);
…
```

Документ та його вигляд.

Основним достоїнством архітектури „Документ/Вигляд” є розділення даних та їх відображення. Довільні зміни даних здійснюються через спеціальний клас документа, який забезпечує:

- необхідний простір для збереження даних в пам’яті;
- читає та записує дані на диск;
- забезпечує інтерфейс для доступу та поновлення даних.

Відображення даних на екрані, вивід даних на друк чи інший пристрій забезпечує спеціальний клас CView чи похідний від нього. Клас вигляду є вікном, що з одної сторони взаємодіє з користувачем а з другої – організовує доступ до інтерфейсної частини документу.

Типова схема створення документу представлено на рис.

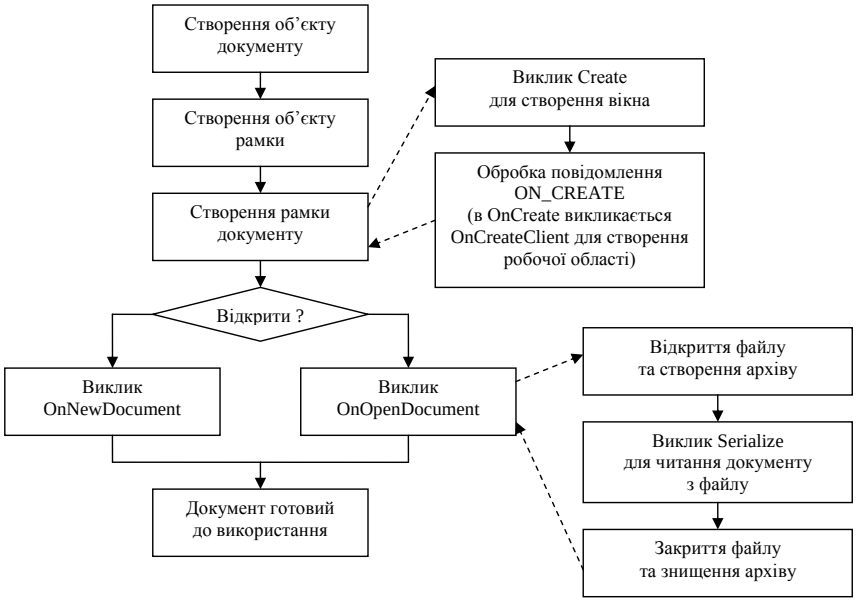

Рис. Схема дій при створенні документу

Документ. Базовим класом для створення документів розробника є CDocument. У нього закладено типові функціональні можливості, що можуть знадобитися при роботі з документами довільних типів, такі як відкриття/закриття файлу, читання/запис даних у файл, зв’язок з виглядом (виглядами). Бібліотека MFC працює з документом використовуючи наперед визначений інтерфейс.

Об’єкти класу CDocument отримують команди від елементів керування інтерфейсу користувача в першу чергу. Якщо вони деяких з них не обробляють, то передають далі на обробку до шаблону.

До основних методів цього класу, що найчастіше використовуються, можна віднести:

```
const CString& CDocument::GetPathName()
```

- повертає шлях до файлу документу.

virtual void CDocument::SetPathName(LPCTSTR lpszPathName, BOOL bAddToMRU = TRUE)

- встановлює новий шлях до файлу документу та при необхідності добавляє його до списку часто використовуваних документів програми.

BOOL CDocument::IsModifed()

- дозволяє визначити чи документ був змінений після останнього збереження.

void CDocument::SetModifedFlag(BOOL bModifed = TRUE)

- дозволяє позначити документ як змінений.

void CDocument::AddView(CView *pView)

- приєднує новий вигляд до документу.

void CDocument::RemoveView(CView *pView)

- від’єднує заданий вигляд від документу.

virtual POSITION CDocument::GetFirstViewPosition()

- повертає структуру для початку пошуку виглядів, що асоціюються з документом.

virtual CView* CDocument::GetNextView(POSITION &rPosition)

- повертає вигляд з наступної позиції в списку, що асоціюються з документом.

Наступні віртуальні функції викликаються з MFC в процесі роботи з документом. Їх назви розкривають їхнє призначення.

virtual void CDocument::OnNewDocument();
virtual void CDocument::OnOpenDocument();
virtual void CDocument::OnSaveDocument();
virtual void CDocument::OnCloseDocument();

Перевизначивши дані методи можна забезпечити гнучке та детальне керування документом.

Вигляд. Для відображення документів у MFC створена спеціальна група класів, що забезпечують створення та управління дочірніми вікнами в яких зображується вміст документу. Базовим класом для них, як показано на рис. __ є CView.

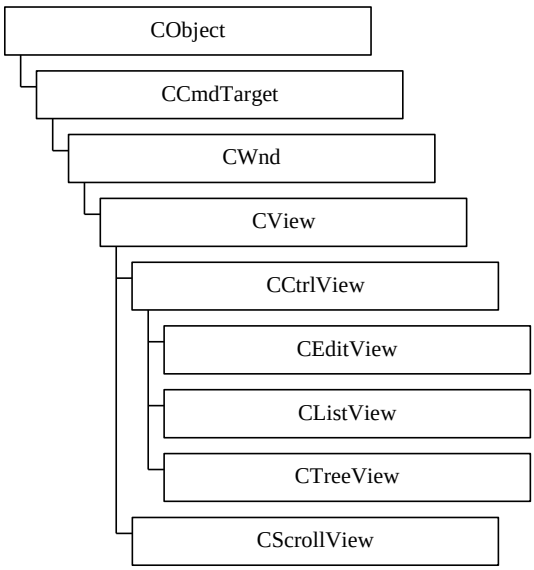

Рис. Ієрархія класів виглядів документу

На відміну від документу, що може мати декілька виглядів одночасно, вигляд асоціюється тільки з одним документом. Для одного документу можна створити декілька рамок з виглядами або розділити одну рамку між декількома виглядами: однотипними чи різнотипними. Усі перераховані можливості надаються бібліотекою MFC.

Для прикладу розглянемо основні методи, що реалізовано в базовому класі виглядів CView. В інших класах, реалізовано додаткові методи, що відображають специфіку представлення в них інформації.

У класах вигляду, окрім конструктора, найчастіше використовуються методи:

CDocument* CView::GetDocument()

- повертає вказівник на документ, що асоціюються з виглядом.

virtual void CView::OnInialUpdate()

- викликається один раз коли вигляд приєднується до документу. В ньому проводиться первинна ініціалізація класу вигляду.

virtual void CView::OnUpdate(CView *pSender, LPARAM lHint, CObject *pHint)

- викликається документом, щоб повідомити вигляду, що в ньому відбулися зміни.

virtual void CView::OnActivateView(BOOL bActivate,
CView *pActivateViev, CView *pDeactivateViev)

- викликається з MFC, коли вигляд активується (отримує фокус вводу) чи деактивується (втрачає фокус вводу).

virtual void CView::OnScroll(UINT nScrollCode, UINT nPos, BOOL bDoScroll)

- викликається з MFC при прокрутці даних у вигляді.

Клас CView містить одну чисто-віртуальну функцію OnDraw, яка власне і відповідає за промальовування документу на екрані або іншому пристрої.

virtual void CView:: OnDraw (CDC *pDC) = 0

Її необхідно обов’язково перевизначити при створення класу нащадку.

Інші базові класи вигляду забезпечують специфічний інтерфейс для представлення даних у вигляді: тексту - CEditView; списку - CListView; дерева - CTreeView. У MFC реалізовано також і вигляди з розширеними можливостями.

Документ та його вигляд.

4. Програми на основі діалогу.

ЛЕКЦІЯ. ЕЛЕМЕНТИ ІНТЕРФЕЙСУ КОРИСТУВАЧА НА ОСНОВІ КЛАСІВ MFC

1. Меню.
2. Панелі елементів керування.
3. Елементи керування.
4. Блоки діалогу. Модальні та не модальні діалоги.

Меню

Більшість віконних програм під Windows містять однин або декілька видів меню. За допомогою меню в основному і здійснюється керування програмою. До основних типів меню які реалізовані в бібліотеці MFC відносяться:

- основне меню;
- системне меню;
- контекстне (плаваюче) меню.

Основне меню. розміщується у верхній частині основного вікна у вигляді смуги, що складається з окремих текстових полів (menu bar). Видима частина основного меню є верхнім рівнем ієрархії системи меню програми. Кожен елемент верхнього рівня може розкриватися та відображати елементи нижнього рівня.

Елементи меню нижнього рівня мати різні типи та містити ряд модифікаторів:

- *акселератор* – повідомляє користувачу набір клавіш для швидкого виклику команди, що зв’язана з цим підпунктом меню;
- *стрілка* – вказує, що з даним підпунктом меню зв’язане підменю, яке буде розкрито при його виборі;
- *маркер* – підказує, що даний пункт (режим) вже вибраний;
- *трикрапка* – повідомляє користувачу, що при виборі цього пункту з’явиться блок діалогу;
- напівтоновий надпис – означає, що в даний момент цей пункт меню недоступний;
- підкреслена літера – дозволяє швидко перейти до елемента меню, шляхом натискання клавіші з літерою яка підкреслена. Такі літери називаються мнемонічними.

Основне меню є основним засобом керування програмою. У ньому, зазвичай, містяться всі команди якими програма керується. Інші засоби керування можуть дублювати функції доступні з основного меню.

Системне меню. Присутнє у вікнах, що мають заголовок та розміщене в правому верхньому куті. У нього входять загальні команди керування програмою, такі як: розгорнути та згорнути вікно; закрити вікно; завершити програму.

Контекстне (плаваюче) меню. Викликається в основному при натисканні правої кнопки миші і контекстно зв’язане з елементом або ділянкою екрану на якому знаходиться курсор миші. Відповідно, воно може міститися в довільному місці екрану. Його вміст залежить від ділянки екрану на якій знаходився курсор миші. Служить для швидкого виклику найбільш вживаних команд для елементу інтерфейсу, що асоціюється з вибраною ділянкою екрану.

В MFC для побудови та програмування меню створений клас CMenu, що увібрав у себе спеціальні функції для роботи з меню реалізовані в Win32 API.

Клас CMenu. Інкапсулює дескриптор HMENU та надає розробнику методи для створення та маніпуляції з елементом керування типу меню. Меню може бути створено:

- динамічно;
- за шаблоном з ресурсів;

- за шаблоном створеним в пам’яті.

Розглянемо основні члени класу CMenu.

CMenu::m_hMenu

- містить дескриптор типу HMENU вікна меню, що приєднано до класу.

CMenu::CMenu()

- конструктор, створює об’єкт з порожнім меню, яке далі може бути наповнене відповідними методами класу.

BOOL CMenu::CreateMenu()

- створює порожнє меню як вікно Windows та приєднує його до об’єкту класу.

BOOL CMenu::CreatePopupMenu()

- створює порожнє підменю як вікно Windows та приєднує його до об’єкту класу.

BOOL CMenu::LoadMenu(LPCTSTR lpszResourceName)

- створює меню як вікно Windows, приєднує його до об’єкту класу та заповнює його елементами з ресурсів програми. Єдиний параметр (символьний рядок або число) служить ідентифікатором цих ресурсів.

BOOL CMenu::LoadMenuIndirect(const void * lpMenuTemplate)

- завантажує меню із шаблону створеного в пам’яті та приєднує його до об’єкту класу. Як параметр приймає адресу цього шаблону.

Наступна група функцій призначена для динамічної зміни складу меню. Відповідно вони застосовуються для наповнення порожнього меню.

BOOL CMenu::AppendMenu(UINT nFlags, UINT nIDNewItem = 0, LPCTSTR lpszNewIt = NULL)

- добавляє новий елемент в кінець меню або підменю з ідентифікатором nIDNewItem та назвою lpszNewItem. nFlags – задає тип та вигляд елементу меню (MF_ENABLED, MF_DISABLED, MF_CHECKED та інше).

BOOL CMenu::InsertMenu(UINT nPosition, UINT nFlags,
UINT nIDNewItem = 0, LPCTSTR lpszNewItem = NULL)

- добавляє новий елемент в меню або підменю в позицію задану параметром nPosition.

BOOL CMenu::ModyfyMenu(UINT nPosition, UINT nFlags,
UINT nIDNewItem = 0, LPCTSTR lpszNewItem = NULL)

- замінює елемент в меню або підменю в позиції заданій параметром nPosition. Пункт меню не знищується, а повністю міняються всі його параметри.

BOOL CMenu::DeleteMenu(UINT nPosition, UINT nFlags)

- знищує елемент в меню або підменю в позиції заданій параметром nPosition.

Наступні функції використовуються для зміни стану та властивостей елементів меню. Назви функцій говорять самі за себе, тобто дають контексну підкузку про їх призначення. Наприклад.

UINT CMenu::EnableMenuItem(UINT nIDEnableItem, UINT nEnable);
UINT CMenu::CheckMenuItem(UINT nIDCheckItem, UINT nCheck);

UINT CMenu::CheckMenuRadioItem(…);
UINT CMenu::SetMenuItemBitmaps(…);

В класі також реалізовані функції, що дозволяють отримати різноманітну інформацію як про саме меню так і про окремі його елементи.

На закінчення розглянемо функцію за допомогою якої створюється плаваюче меню.

BOOL CMenu::TrackPopupMenu(UINT nFlags, int x, int y,
CWnd *pWnd, LPCRECT lpRect = 0);

- виводить на екран контекстне меню та створює свій цикл обробки повідомлень. Після вибору елементу меню або відмови від вибору меню знищується. pWnd – визначає вікно яке отримає повідомлення WM_COMMAND після вибору користувачем елемента меню.

Панелі елементів керування.

Панелі елементів керування є додатковими засобами керування програмою. Вони найчастіше використовуються для швидкого виклику деяких команд, що найчастіше вживаються, таких як „відкрити файл”, „зберегти зміни у файлі”, „роздрукувати” та інші.

Базовим класом для панелей елементів керування є клас CControlBar. Ієрархія класів приведена на рис. __

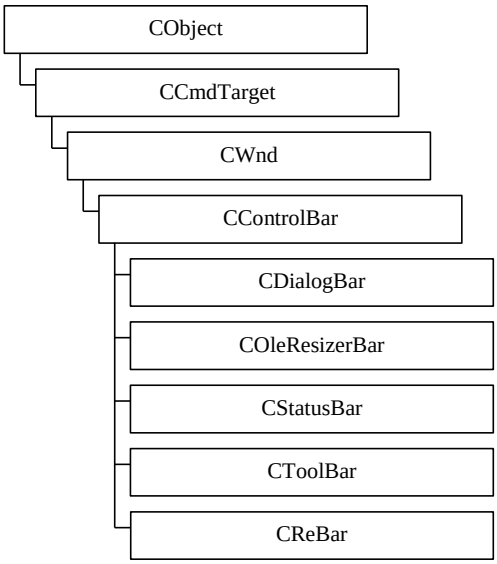

Рис. __. Ієрархія класів панелей елементів керування

Відповідно до приведеної ієрархії класів, панель керування є вікном, і може містити дочірні вікна, які можуть отримувати та відправляти повідомлення (комбінований список COMBO BOX, текстове поле EDIT) або тільки керуватися програмою (кнопки з малюнком BITMAP BUTTONS, ділянка рядка стану STATUS BAR PANES). Базові функціональні можливості для всіх типів панелей елементів керування подібні – всі їх об’єкти є дочірніми вікнами деякої батьківської рамки.

Панель керування може бути *фіксованою* (dockings) біля довільного боку її батьківської рамки або плаваючою (floating), що визначається встановленням

відповідного стилю. Крім того, об’єкти CControlBar можуть самостійно позиціонуються при зміні розмірів батьківської рамки.

Так як клас CControlBar є абстрактним, об’єктів на основі нього створювати не можна. Панелі керування створюються на основі дочірніх класів. До методів які реалізовані в класі CControlBar та часто використовуються відносяться:

virtual CSize CControlBar::CalcFixedLayout(BOOL bStretch, BOOL Horz);

- визначає горизонтальний розмір панелі елементів керування. Парметр Horz задає горизонтальну чи вертикальну орієнтацію панелі.

void CControlBar::SetBarStyle(DWORD dwStyle);

- задає стиль панелі елементів керування. Наприклад, фіксація біля визначеної сторони батьківської рамки, підкреслення краю панелі, приоритетне розміщення, дозвіл на зміну розмірів та інше.

DWORD CControlBar::GetBarStyle();

- повертає поточний стиль панелі елементів керування.

BOOL CControlBar::IsFloating();

- дозволяє перевірити, чи панель елементів керування є плаваючою в даний момент.

З панелями керування тісно зв’язані деякі функції батьківської рамки (класу CFrameWnd), серед них:

void CFrameWnd::SaveBarState(LPCTSTR lpszProfileName);

void CFrameWnd::LoadBarState(LPCTSTR lpszProfileName);

- дозволяють зберегти та завантажувати інформацію про положення, видимість, орієнтацію, фіксацію кожної панелі керування в INI-файлі або в реєстрі системи.

void CFrameWnd::ShowControlBar(CControlBar * pBar, BOOL bShow, BOOL bDelay);

- гасить або візуалізує панель елементів керування.

void CFrameWnd::DockControlBar(CControlBar * pBar, …);

- прив’язує панель до певної сторони рамки.

void CFrameWnd::FloatControlBar(CControlBar * pBar, …);

- переводить панель в плаваючий стан.

Як приклад дочірнього класу CControlBar розглянемо *панель інструментів* CToolBar. У цьому класі реалізовано відображення елементів керування у вигляді кнопок з піктограмами. Інші класи панелей елементів керування організовані подібно, тільки реалізують специфічні для них властивості. Тому розгляд цього класу дасть ключ до розуміння інших класів панелей

Клас CToolBar. Цей клас визначає поведінку панелі інструментів, що задаються кнопками з піктограмами (розміром 23x22 піксела) та розділювачами. Кнопки, подібно до пунктів меню, активізовують команди, що посилаються об’єктам програми та обробляються ними. Кнопки можуть працювати, також, як прапорці та перемикачі.

Для створення об’єкту класу CToolBar використовується конструктор.

void CToolBar::CToolBar();

- конструктор, створює об’єкт класу.

void CToolBar::Create(CWnd * pParentWnd, DWORD Style, UINT nID = AFX_IDW_TOOLBAR);

- створює вікно та приєднує його до об’єкту. pParentWnd – вказівник на батьківське вікно панелі інструментів; Style - стиль панелі; nID – оригінальний ідентифікатор панелі.

void CToolBar::LoadToolBar(LPCTSTR lpszResourceName);

- створює вікно та приєднує його до об’єкту та завантажує дані для нього з ресурсів зданих через ім’я lpszResourceName.

void CToolBar::LoadBitmap(LPCTSTR lpszResourceName);

- завантажує з ресурсів програми бітові масиви з піктограмами (через ім’я lpszResourceName).

Тут приведені тільки функції створення та ініціалізації панелі інструментів, вони є типовими для багатьох елементів графічного інтерфейсу віконних програм. З інші функціями класу, що дозволяють гнучко керувати поведінкою панелей, можна ознайомитися в технічній документації.

Елементи керування

У бібліотеці MFC створено ряд елементів керування. Вони призначенні для створення графічного інтерфейсу з користувачем, а саме безпосереднього введення інформації, що керує програмою. Для кожного з них створено відповідний клас. До класів, що реалізовують прості елементи керування відносяться;

CStatic – статичний текст або зображення (мітки);

CButton – командні кнопки, що спрацьовують при натисканні, перемикачі, прапорці;

CListBox – список;

CEdit – текстове поле;

CComboBox – комбінований блок списку та текстового поля;

CScrollBar – смуга прокрутки.

Окрім того в бібліотеці реалізовано ряд елементів керування з розширеними можливостями. Вони, в деякій мірі дублюють прості елементи, але мають деякі додаткові функції. До них відносяться:

CBitmapButton (CButton) – кнопки із заданими картинками для всіх станів;

CComboBoxEx (CComboBox) – комбінований список з можливістю роботи із зображеннями;

CCheckListBox (CListBox) – список елементів, що реалізовані як прапорці;

CDragListBox (CListBox) – список з можливістю переносу його елементів за допомогою миші.

При виборі між простим та розширеним елементом керування необхідно вирішувати наскільки повно будуть використовуватися розширені можливості. При неповному їх використанні краще обмежитися простими елементами. Вони простіші в застосуванні та працюють швидше.

Способи створення елементів керування.

Типові елементи керування можна створити двома способами:

- в редакторі ресурсів;

- безпосередньо в коді програми.

Створення елементів керування в редакторі ресурсів. Елементи керування створюються візуально безпосередньо в редакторі ресурсів при проектуванні діалогового вікна. В цьому випадку можна задати розміри, зовнішній вигляд, початковий стан та деяку поведінку цих елементів. Але деякі типи елементів керування (наприклад CListBox, CComboBox) не можна створити повністю повноцінними за допомогою редактора ресурсів. Вони будуть вимагати написання додаткового коду, щоб активізувати усі їх можливості.

Створення елементів керування в коді програми. Як і інші візуальні об’єкти класів MFC, елементи керування в коді програми створюються в два етапи:

- створення об’єкту (екземпляру) певного класу (змінної типу);
- створення об’єкту вікна для цього екземпляру (виклик функції Create()).

Зазвичай створення вікон елементів керування відбувається у функції, що ініціалізує батьківське вікно. Для блоків діалогу (на яких найчастіше розміщуються елементи керування), такою функцією є OnInitDialog(), для інших вікон OnCreate().

Розглянемо типовий приклад створення елементу керування текстову поле в блоці діалогу.

Спочатку, агрегація включення об’єкту m_Edit в клас CPrimeDialog.

```
class CPrimeDialog : public CDialog
{
   …
protected:
   CEdit m_Edit;
   …
public:
   virtual BOOL OnInitDialog();
   …
};
```

У методі OnInitDialog() для елементу керування текстове поле (m_Edit) задається прямокутник, що задає його розміри та положення в межах вікна діалогу, потім викликається його метод Create(), для створення об’єкту вікна та зв’язування його з базовим об’єктом. На завершення викликається його метод SetFocus(), що захоплює для нього фокус вводу в межах діалогу.

```
BOOL CPrimeDialog::OnInitDialog()
{
   CDialog::OnInitDialog();
      …
   CRect winRect(20, 20, 100, 50);
   m_Edit.Create(WS_CHILD | WS_VISIBLE | WS_TABSTOP | WS_BORDER |
                 ES_CENTER | ES_LOWERCASE | ES_MULTILINE,
                 winRect,
                 this,
```

```
                    IDC_EDIT);
  m_Edit.SetFocus();
  …
  return FALSE;
}
```

Метод Create() має ряд параметрів загальних для всіх елементам керування. Розглянемо ці параметри на прикладі відповідного методу для елементу керування класу CEdit – текстове поле.

```
BOOL CEdit::Create(DWORD dwStyle, const RECT & rect, CWnd * pParentWnd,
UINT nID);
```

Параметри:

dwStyle – задає стилі елемента керування, серед них можуть бути як загальні стилі вікон (WS_CHILD | WS_VISIBLE | WS_TABSTOP | WS_BORDER), так і особливі стилі притаманні тільки певному класу елементів керування (ES_CENTER | ES_LOWERCASE | ES_MULTILINE). Стилі сумуються за допомогою побітового оператора „або”.

rect – задає габарити та положення, в межах вікна діалогу, прямокутника вікна елементу керування, параметри прямокутника сформовані у вигляді структури RECT. Для розглянутого прикладу лівий верхній кут прямокутника матиме координати (20, 20), а правий нижній (100, 50).

pParentWnd – вказівник на батьківське вікно до якого належить елемент керування. Через нього елемент керування буде обмінюватися повідомлення з іншими елементами інтерфейсу користувача.

nID – ідентифікатор елементу керування, оригінальний в межах батьківського вікна. Він використовується деякими методами класу CWnd для доступу до елементу керування.

Кожен з класів елементів керування може мати свої, додаткові параметри для методу Create(), але перелічені має кожен клас.

Елементи керування ведуть себе як звичайні вікна, вони можуть посилати, приймати повідомлення та обробляти повідомлення. Більшість загальних функціональних можливостей вони наслідують від батьківських класів (наприклад, від батька всіх віконних об’єктів - CWnd). Безпосередньо в них реалізується тільки особлива, притаманна тільки їм функціональність.

Блоки діалогу

Блоки діалогу (dialog boxes) – спеціальні вікна, що забезпечують користувача гнучкими засобами взаємодії з програмним забезпеченням. Вони, як правило включають дочірні вікна, що є елементами керування (розглядалися раніше) через які користувач налаштовує програму та водить дані для обробки. Вони є однією з основних та традиційних частин інтерфейсу користувача в Windows.

Модальні та не модальні діалоги.

Відповідно до поведінки, розрізняють два типи діалогів – *модальний* (modal) та *немодальний* (modeless). Різниця між ними полягає в способі керування потоками повідомлень.

Модальні блок діалогу блокують всі інші вікна програми доки дане вікно діалогу не буде закрито. Тобто, вони блокують потоки повідомлень від миші та клавіатури, що йдуть до батьківських вікон та вікон того ж рівня (sibling), що робить їх недоступними в межах одної програми. Можна створити системне модальне вікно, що заблокує всі програми в системі доки не буде закрито.

Немодальні діалоги більше подібні на звичайні вікна, вони дають користувачу можливість паралельно працювати і з іншими вікнами програми. Вони не блокують потоків повідомлень, що йдуть в довільну частину програми. Більшість блоків діалогу є *модальними*, це пов’язано з тим, що спосіб їх поведінка (блокування всіх інших вікон) якнайкраще концентрує увагу користувача при введенні даних. Блоки діалогів, модальні та немодальні, як і деякі інші вікна, можуть створюватися вдома способами:

- за шаблоном з ресурсів;
- за шаблоном створеним в пам’яті.

Найбільш розповсюдженим є спосіб створення діалогів (модальних та немодальних) на основі шаблону в ресурсах. Цей спосіб передбачає створення заготовки діалогу (шаблону) за допомогою середовища візуального програмування та розміщення цього шаблону в спеціальному форматі у файлі ресурсів програми (resource.rc). При необхідності шаблон діалогу можна створити відредагувавши ресурси текстовим редактором (файл ресурсів має текстову структуру), але такий спосіб не практикується. В ресурсах шаблони ідентифікуються унікальним цілим числом без знаку або символьним рядком з унікальною назвою елементу керування. Цей ідентифікатор є одним з параметрів для створення діалогу за шаблоном з ресурсів. Розглянемо способи створення модальних та немодальних діалогів.

Створення та робота з модальними діалогами. Об’єкт модального діалогу за шаблоном з ресурсів створюється шляхом виклику необхідного конструктора, що проводить початкову ініціалізацію змінних класу та зв’язує об’єкт з шаблоном.

```
CDialog::CDialog(LPCTSTR lpszTemplateName, CWnd * pParentWnd = NULL);
CDialog::CDialog(UINT nIDTemplate, CWnd * pParentWnd = NULL);
```

де lpszTemplateName та nIDTemplate – відповідно символьний рядок та число, що ідентифікують діалог в ресурсах; pParentWnd – вказівник на батьківське вікно, якщо він нульовий, то батьківським є основне вікно програми.

При створенні діалогу на основі шаблону в пам’яті спочатку необхідно створити порожній об’єкт діалогу викликавши конструктор за замовчуванням. Потім заповнити спеціальну структуру типу DLGTEMPLATE в якій, подібно до шаблону в ресурсах, описується вид на наповнення діалогу елементами керування. На закінчення викликається функція

```
BOOL CDialog::InitModalIndirect(LPCDLGTEMPLATE lpDialogTemplate,
                                CWnd * pParentWnd = NULL);
```

для ініціалізації діалогу.

Діалог створений за першим або другим способом буде невидимим. Для його активізації необхідно викликати метод

virtual int CDialog::DoModal();
який активізує і відобразить діалог (модальний). Ця функція захопить та буде утримувати керування доти, доки діалог не буде закритий. Вона повертає ціле число, через яке можна визначити код завершення діалогу. Наприклад, була натиснута кнопка „Так” чи „Ні”.

Створення та робота з немодальними діалогами. При створенні немодальних діалогів в обох випадках спочатку створюється об’єкт з викликом конструктора за замовчування. Потім для діалогу, що створюється за шаблоном з ресурсів, викликається один з методів
CDialog::Create(LPCTSTR lpszTemplateName, CWnd * pParentWnd = NULL);
CDialog::Create(UINT nIDTemplate, CWnd * pParentWnd = NULL);

Процес створення немодального діалогу на основі шаблону з пам’яті майже нічим не відрізняється від аналогічного для модального діалогу, окрім методу який його завершує
BOOL CDialog::CreateIndirect(LPCDLGTEMPLATE lpDialogTemplate,
CWnd * pParentWnd = NULL);

На відміну від модального діалогу – немодальний за замовчуванням створюється невидимим. Для його візуалізації та заковування необхідно явно викликати метод ShowWindow().

Ще одна функціональна відмінність між модальним та немодальним діалогами. Метод DoModal(), створює та знищує об’єкт вікна при кожному своєму запуску. Для немодального діалогу об’єкт вікна, після виклику методу Create(), існує завжди. Для його знищення необхідно явно викликати метод DestroyWindow().

Для обміну даними з діалоговими вікнами існують різні механізми, які стандартно реалізовані в бібліотеці, і на жаль розглянути їх в короткому курсі не має можливості.

Класи типових діалогів Windows.

У бібліотеці повністю реалізовано та пропонується до використання (у загальному без потреби в модифікації коду) декілька класів типових діалогів. Серед них:

CFileDialog – діалог для навігації системою каталогів та вибору файлу.
CColorDialog – діалог вибору або створення палітри кольору.
CFontDialog – діалог вибору та на лаштування параметрів шрифту.
CFindReplaceDialog – діалог для організації пошуку та заміни тексту.
CPrintDialog – діалог вибору та налаштування друкарки.
CPageSetupDialog – діалог налаштування параметрів сторінки.