Тема 3. Використання бібліотеки візуальних компонентів VCL

1. Огляд бібліотеки VCL

Бібліотека візуальних компонентів VCL (Visual Component Library) – об’єктно-орієнтована бібліотека для розробки програмного забезпечення, розроблена компанією Borland для підтримки візуального програмування. VCL [6-10, 25-29] входить у комплект поставки C++Builder, Delphi і Borland Developer Studio і є частиною середовища розробки, хоча розробка додатків у цих середовищах можлива й без використання VCL. VCL представляє величезну кількість (більше 360) готових до використання компонентів для роботи в самих різних областях програмування, таких, наприклад, як інтерфейс користувача (екранні форми, елементи керування та інші), робота з базами даних, взаємодія з операційною системою, програмування мережних додатків та інше.

Сукупність функцій, за допомогою яких здійснюється доступ до системних ресурсів, називається прикладним програмним інтерфейсом, або API (Application Programming Interface). Для взаємодії з Windows додаток використовує функції API, за допомогою яких реалізуються всі необхідні системні дії, такі як виділення пам'яті, вивід на екран, створення вікон і т.п. Бібліотека VCL є просто оболонкою для WinAPI. Класи VCL створені тільки для того, щоб полегшувати процес програмування. Наприклад, один з найважливіших класів TForm з бібліотеки VCL можна було б створити, не використовуючи класи VCL.

Компонент C++Builder – це особливий вид об'єктів – візуальний об'єкт (візуальний для проектування, а не для відображення користувача). Створювати й редагувати такий об'єкт можна як програмним шляхом, так і на етапі проектування.

При виконанні програми компоненти діляться на візуальні, які бачить користувач, і невізуальні, для яких немає можливості їхнього відображення, але доступ до властивостей яких дозволений.

Усі компоненти мають загального предка – клас TComponent. Усі компоненти C++Builder можуть бути доступні через палітру компонентів. Частина компонентів є елементами керування. В основному – це елементи керування Windows.

Елементи керування можна підрозділити на віконні й невіконні. Віконні елементи можуть одержувати фокус вводу і мають дескриптор вікна. Предком всіх віконних елементів керування є абстрактний клас TWinControl. Предком невіконних елементів керування є абстрактний клас TGraphicControl.

При додаванні у форму будь-якого компонента з палітри компонентів C++Builder автоматично формує програмний код для створення об'єкта (змінної) даного типу. Змінна додається як член класу даної форми.

Класи бібліотеки VCL використовують механізм простого успадкування: один клас може мати тільки одного предка. Коренем ієрархії класів є клас**TObject**. Нижче наведено ієрархію ключових базових класів бібліотеки VCL.

Рис. 1.

Будь-який ***клас*** VCL-бібліотеки успадковується від класу TObject. Клас **TPersistent** пішов безпосередньо від класу TObject. Він забезпечує своїх нащадків можливістю взаємодіяти з іншими об'єктами і процесами на рівні даних. Його методи дозволяють передавати дані в потоки, а також забезпечують взаємодію об’єкта з інспектором об'єктів. Клас TPersistent має метод Assign, який дозволяє передавати поля і властивості одного об’єкту іншому.

Клас **TComponent** є предком всіх ***компонентів*** VCL-бібліотеки. Всі нащадки даного класу можуть бути розташовані в палітрі компонентів.

Клас TComponent дозволяє визначати батьківський елемент керування й власника компонента. Батьківським елементом керування називається той, у який безпосередньо поміщений даний компонент. Власником всіх компонентів, розташованих у формі, є сама форма. Власником всіх форм є додаток.

Якщо компонент розташований не безпосередньо у формі, а, наприклад, у компоненті типу TPanel, то власник і батьківський елемент керування в нього будуть різні.

TControl – це базовий клас всіх елементів керування (включаючи й вікно форми). Ці компоненти можуть бути видимі під час виконання. Для них визначені такі властивості, як позиція, курсор, підказка, що спливає, методи для малювання або переміщення елемента керування, події для маніпуляцій за допомогою миші.

Клас **TWinControl** є базовим класом всіх віконних елементів керування.

Клас **TApplication** інкапсулює об'єкт "windows-додаток". За допомогою цього класу визначається інтерфейс між розробником і середовищем Windows. У кожному додатку C++Builder завжди автоматично створюється один об’єкт Application як екземпляр класу додатка. Для більшості додатків цей об’єкт є екземпляром класу TApplication. Компонент TApplication не відображається в палітрі компонентів і не має властивостей, що публікуються. Для того, щоб мати можливість перехоплювати події для додатка, можна додати в будь-яку форму проекту компонент TApplicationEvents.

Кожний додаток C++Builder має глобальну змінну Screen типу TScreen. Компонент **TScreen**, так само як і компонент TApplication, недоступний з інспектора об'єктів. Цей компонент призначений для забезпечення доступу до пристрою виводу – екрану. Його властивості містять інформацію про використовуване розширення монітора, курсорах і шрифтах, доступних для додатка, списку форм додатка й активній формі.

TForm є базовим класом для створення вікна форми. За замовчуванням кожна нова створювана форма реалізується як нащадок класу TForm. Форма може бути головним вікном додатка, діалоговим вікном, дочірнім або MDI-вікном.

Клас форми є контейнером для всіх компонентів, розташовуваних на формі. Для доступу до властивостей форми або іменам компонентів можна використовувати вказівник this. Якщо перед іменем властивості відсутній який-небудь вказівник, то за замовчуванням вважається, що це властивість форми.

2. Динамічні компоненти форми

Компоненти на форму можна добавляти не тільки під час проектування. Їх можна створювати і знищувати динамічно під час роботи програми.

Для того, щоб створити на формі новий екземпляр, наприклад, компонента типу TButton, спочатку в заголовочному файлі в секції private форми треба описати вказівник на майбутній компонент:

```
private:

TButton *pButton;
```

Далі у файлі реалізації, можливо – у деякому обробнику подій, динамічно створювати сам компонент:

```
// власник компонента - форма

pButton = new pButton(this);

//батько компонента – форма (задається обов'язково)

pButton->Parent = this;

// параметри розміщення на формі

pButton->Left = 10;

pButton->Width = 100;

pButton->Top = 10;

pButton->Height = 20;
```

Цей код створює компонент типу TButton, розташований на формі в позиції 10,10 і протягнений на 100 пикселей вправо й 20 пикселей вниз. Подібний код може бути використаний для будь-якого типу компонентів, оскільки всі компоненти підтримують ці атрибути.

Якщо треба встановити інші властивості, то це робиться аналогічно через вказівник pButton. Якщо ж новостворений компонент повинен мати обробник деякої події, то заголовок майбутнього обробника цієї події треба додати в заголовочний файл форми у секцію private. Причому, прототип заголовку має повністю відповідати типу події.

Наприклад, для створеної кнопки вимоги для прототипу обробника події OnClick мають вигляд:

- один вхідний аргумент типу TObject* ;
- повертається тип void;
- обов'язково використання ключового слова __ fastcall.

Отже, прототип обробника події OnClick описується так:

private:

void __fastcall OnButtonClick(TObject * Sender);

У файлі реалізації при створенні компонента додається рядок

pButton->OnClick=OnButtonClick;

А сам обробник має вигляд

void __fastcall TForm1::OnButtonClick(TObject* Sender)

{. . .}

Зазначимо, що є можливість переключення обробників подій під час роботи програми. Це робиться присвоюванням властивості pButton->OnClick імені іншого обробника подій.

Оскільки компонент Button був створений динамічно за допомогою оператора new, то і знищувати його треба за допомогою оператор delete, наприклад, так:

if(pButton){delete pButton; pButton=NULL;}

3. Автоматичне і динамічне створення форм

За замовчуванням при запуску додатка C++Builder автоматично створює по одному екземпляру кожного класу форм у проекті і звільняє їх при завершенні програми. Автоматичне створення обробляється C++Builder кодом, що генерується у трьох місцях.

Перше – інтерфейс класу форми у файлі із роширенням .h:

```
class TForm1 : public TForm

{

__published:  // IDE-managed Components

private:      // User declarations

public:           // User declarations

__fastcall TForm1(TComponent* Owner);

};
```

У даному фрагменті коду описується клас TForm1. Друге місце – це файл реалізації класу форми з розширенням .cpp.

```
TForm1 *Form1;
```

Тут описана змінна Form1, що вказує на екземпляр класу TForm1 і доступна з будь-якого модуля. Звичайно вона використовується під час роботи програми для керування формою.

Третє місце знаходиться в функції WinMain вихідного тексту проекту, доступ до якого можна одержати за допомогою меню **View | Project Source**. Цей код виглядає як:

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

. . .

Application->CreateForm(__classid(TForm1), &Form1);

. . .

}
```

Процес видалення форм обробляється за допомогою концепції власників об'єктів: коли об'єкт знищується, автоматично знищуються всі об'єкти, якими він володіє. Створена описаним вище способом форма належить об'єкту Application і знищується при закритті додатка.

Хоча автоматичне створення форм корисно при розробці SDI-додатків, при створенні MDI-додатку воно, як правило, неприйнятно. Для створення нового екземпляра форми використовується конструктор класу форми. Наведений нижче код створює новий екземпляр TForm1 під час роботи програми і встановлює його властивість Caption рівною “Нова форма”.

```
TForm1 * pForm1;
```

pForm1 = new TForm1 (Application);

pForm1->Caption = "Нова форма";

pForm1->Visible= true;

Конструктор одержує як параметр нащадка TComponent, що і буде власником форми. Звичайно в якості власника виступає Application, щоб усі форми були автоматично закриті по закінченні роботи додатка. Можна також передати конструктору параметр NULL, створивши форму без власника, але тоді закривати і знищувати її треба вручну.

Знищення форми відбувається за допомогою виклику методу Release() для неї. Це можна зробити, наприклад, так:

if(pForm1)pForm1->Release();

4. Створення багатодокументного інтерфейсу MDI

Термін MDI (Multiple Document Interface) дослівно означає багатодокументний інтерфейс. Він описує додатки, здатні завантажити і використовувати одночасно кілька документів чи об'єктів. Прикладом такого додатка може служити диспетчер файлів File Manager або текстовий редактор Microsoft Word.

Звичайно MDI-додатки складаються мінімум із двох форм – батьківської і дочірньої. Щоб форма була батьківською, властивість форми FormStyle установлюється рівною fsMDIForm. Для дочірньої форми стиль FormStyle встановлюється рівним fsMDIChild.

Батьківська форма служить контейнером, що містить дочірні форми, що укладені в клієнтську область і можуть переміщатися, змінювати розміри, мінімізуватися чи максимізуватися. У додатку можуть бути дочірні форми різних типів, наприклад одна – для обробки зображень, а інша – для роботи з текстом.

У MDI-додатку, як правило, потрібно створювати кілька екземплярів класів дочірньої форми. Оскільки кожна форма являє собою об'єкт, вона повинна бути створена перед використанням і звільнена, коли вона більше не потрібна. При створенні MDI-додатку екземпляри дочірньої форми створюються і знищуються динамічно.

Об'єкт типу TForm має кілька властивостей, методів і подій, специфічних для MDI-додатків.

4.1. MDI-властивості форми

ActiveMDIChild – ця властивість повертає дочірній об'єкт TForm, що має в поточний час фокус введення. Це корисно, коли батьківська форма містить панель інструментів чи меню, команди яких поширюються на відкриту дочірню форму.

MDIChildren і MDIChildCount. Властивість MDIChildren є масивом об'єктів TForm, що надає доступ до створених дочірніх форм. Властивість MDIChildCount повертає кількість елементів у масиві MDIChildren. Звичайно ця властивість використовується при виконанні якої-небудь дії над усіма відкритими дочірніми формами. Наприклад, код згортання всіх дочірніх форм деякою командою Minimize All може виглядати так:

void__fastcall TMainForm::WindowMinimizeItemClick(TObject *Sender)

{

for(int iCount=MDIChildCount-1; iCount>=0; iCount--)

MDIChildren[iCount]->WindowState = wsMinimized;

}

TileMode – властивість типу, що визначає, як батьківська форма розміщає дочірні при виклику методу Tile. Використовуються значення tbHorizontal (за замовчуванням) і tbVertical для розміщення форм по горизонталі і вертикалі.

WindowMenu*.* Професійні MDI-додатки дозволяють активізувати необхідне дочірнє вікно, вибравши його зі списку в меню. Властивість WindowMenu визначає об'єкт TMenuItem, що C++Builder буде використовувати для виводу списку доступних дочірніх форм. Для виводу списку TMenuItem повинне бути меню верхнього рівня. Це меню має властивість Caption, рівне swindow.

4.2. MDI-методи форми

Специфічні для MDI-форм методи перелічено нижче.

- ***Arrangelcons*** вибудовує піктограми мінімізованих дочірніх форм у нижній частині батьківської форми;
- ***Cascade*** розташовує дочірні форм каскадом, так що видно всі їхні заголовки;
- ***Next*** і ***Previous*** переходить від однієї дочірньої форми до іншої в прямому і зворотному напрямі відповідно;
- ***Tile*** вибудовує дочірні форми так, що вони не перекриваються.

4.3. MDI-події

У MDI-додатку подія OnActivate запускається тільки при переключенні між дочірніми формами. Якщо фокус уведення передається з не MDI-форми в MDI-форму, генерується подія OnActivate батьківської форми, хоча її властивість Active ніколи і не встановлюється рівною true. Ця подія насправді строго логічна: адже, якби OnActivate генерувався тільки для дочірніх форм, не було б ніякої можливості довідатися про перехід фокуса уведення з іншого додатка.