# Тема 5. Створення власних компонентів

**1. Побудова каркасу компонента**

**2. Оголошення властивостей**

**3. Визначення подій**

**4. Додавання методів**

## 1. Побудова каркасу компонента

Новий компонент краще всього спочатку створювати у власному проекті. Для цього треба створити новий проект і викликати майстра по створенню компонентів**: Component| New**.

У вікні, що з'явиться, треба назвати новий компонент і вибрати пращура, тобто клас, від якого буде породжений новий компонент. Нехай, в якості пращура обраний клас **TCustomMemo**. Назвемо новий компонент **TMyMemo** і виберемо місце розташування майбутнього компонента – сторінку **Samples** з палітри компонентів.

Після таких дій майстер по створенню компонентів згенерує два наведених нижче файли.

***Файл MyMemo.h***

```
// каркас компонента TMyMemo-нащадка TCustomMemo

class PACKAGE TMyMemo : public TCustomMemo

{

private:

protected:

public:

__fastcall TMyMemo(TComponent* Owner);

__published:

};
```

***Файл MyMemo.cpp***

```
#include "MyMemo.h"

. . .

// функція перевірки наявності у компонента
```

```
// чистих віртуальних функцій
static inline void ValidCtrCheck(TMyMemo *)
{
new TMyMemo(NULL);
}
//порожній конструктора компонента TMyMemo
__fastcall TMyMemo::TMyMemo(TComponent* Owner)
: TCustomMemo(Owner)
{}
namespace Mymemo  // простір імен
{
void __fastcall PACKAGE Register()// функція реєстрації
{
// масив компонентів, що реєструються
TComponentClass classes[1] = {__classid(TMyMemo)};
// реєстрування компонентів
RegisterComponents(
"Samples", // ім'я вкладки палітри компонентів
classes, // масив компонентних класів
0             // індекс останнього елемента масиву
);
}
}
```

Тепер згенеровані файли (каркас компонента) треба наповнити кодами, тобто – додати необхідні властивості, методи і події.

## 2. Оголошення властивостей

Усі компоненти VCL дотримуються наступних угод про властивості:

- значення властивостей зберігають члени даних об'єкта;

- ідентифікатори членів даних, що зберігають значення властивостей, утворяться додаванням префікса F до імені цієї властивості. Так, наприклад, значення властивості Top компонента TControl зберігає член даних під іменем FTop.
- ідентифікатори членів даних, що зберігають значення властивостей, повинні бути оголошені як private. При цьому компонент, що оголосив ці властивості, має до них доступ, а користувач даного компонента і його похідних – немає.
- похідні компоненти повинні використовувати успадковану властивість, не намагаючись здійснити прямий доступ до пам'яті внутрішніх даних;
- тільки методи, що реалізують властивість, мають право доступу до своїх значень. Якщо якомусь іншому методу або компоненту знадобиться змінити ці значення, вони повинні здійснювати це за допомогою даної властивості, а не звертаючись прямо до його внутрішніх даних.

C++Builder використовує ключове слово __property для оголошення властивостей [1, 28]. Синтаксис опису властивості має вигляд:

***__property <тип властивості> <ім'я властивості> = {<список атрибутів>};***

де список атрибутів містить наступні атрибути властивості:

- write = < член даних або метод запису >;
- read = < член даних або метод читання >;
- default = < булева константа, що керує збереженням значення >;
- stored = < булева константа або функція, що зберігає значення >.

Існує два способи, за допомогою яких властивості забезпечують доступ до внутрішніх членів даних компонентів: прямий або непрямий за допомогою методів читання/запису.

Прямий доступ є найпростішим способом звертання до значень властивостей. Атрибути read і write вказують, що читання або присвоювання значень внутрішнім членам даних властивості відбувається безпосередньо, без виклику відповідних методів. Прямий доступ найчастіше використовується для читання значень властивостей.

Методи читання й запису заміщають імена членів даних в атрибутах read і write оголошення властивості. Вони повинні бути оголошені як приватні Оголошення методів приватними захищає користувача похідного компонента від випадкового виклику неадекватних методів, що модифікують властивості не так, як очікувалося.

Всі компоненти успадковують властивості своїх попередників, причому абстрактні базові класи зазвичай оголошують свої властивості у секціях з обмеженим доступом. Щоб такі властивості стали доступними користувачам похідних компонентів (як на стадії проектування, так і під час виконання програми), необхідно перевизначити їх з ключовим словом __published. У такому випадку тип властивості не вказується, оскільки він визначений у батьківському класі.

Імена методів для читання властивостей прийнято починати префіксом Get, а імена методів для запису – префіксом Set. Назву самої властивості бажано використовувати як суфікс.

Значення властивості за замовчуванням встановлює конструктор даного компонента. C++Builder використовує оголошене значення властивості за замовчуванням default, щоб визначити, чи варто зберігати властивість у файлі форми з розширенням .dim (якщо атрибут stored явно не забороняє це).

Властивість може бути будь-якого типу, який здатна повернути функція. Різні типи властивостей по-різному представлені у вікні інспектора об'єктів і визначають різні варіанти їх редагування. Деякі властивості мають власні редактори.

Правилами мови C++ встановлюються наступні узагальнені групи типів компонентних властивостей [29]:

- ***Simple***. Прості числові, символьні і рядкові властивості показуються інспектором об'єктів у вигляді чисел, символів або символьних рядків відповідно. Можна безпосередньо вводити і редагувати значення простих властивостей.
- ***Enumerated.*** Властивості перелічуваного типу (зокрема булеві) показуються інспектором об'єктів у вигляді значень, визначених в початковому тексті програми. Можна вибирати можливі значення з випадного списку.
- ***Set.*** Властивості типу множини показуються інспектором об'єктів у вигляді елементів множини. При розширенні множини з кожним його елементом слід поводитися як з булевим значенням: true, якщо елемент належить множині, або false – інакше.
- ***Object.*** Властивості, які самі є об'єктами, зазвичай обслуговуються своїми власними редакторами. Інспектор об'єктів дозволяє індивідуально редагувати ті об'єктні властивості, які оголошені як _published. Об'єктні властивості повинні бути похідними від TPersistent
- ***Array.*** Властивості типу масив повинні обслуговуватися своїми власними редакторами властивостей. Інспектор об'єктів не має вбудованих засобів для редагування таких властивостей.

Для прикладу визначимо одну батьківську і одну власну властивість компонента TMyMemo:

***class PACKAGE TMyMemo: public TCustomMemo***

***{***

***. . .***

***private: // закриті члени-дані класу***

***bool FModified;***

***public: // відкриті члени класу***

***// власна властивість часу виконання***

***__property bool Modified = {read=FModified, default=false};***

***__published: //загальновідомі члени класу***

***// батьківська властивість ScrollBars***

***// стала видимою інспектору об\'єктів***

***__property  ScrollBars;***

***};***

## 3. Визначення подій

Формально C++Builder визначає подію як покажчик, що типізується, на метод в специфічному екземплярі класу:

***<тип> (__closure *<ім'я методу>)(<список параметрів>);***

Для розробника компонента closure є деякою програмною заглушкою: коли користувач визначає реакцію на деяку подію, місце заглушки займає його обробник, який викликається програмою користувача при виникненні цієї події.

Всі компоненти VCL успадковують більшість загальних повідомлень Windows – стандартні події. Такі події вбудовані в захищені секції компонентів, тому користувачі не можуть під'єднувати до них свої обробники.

При створенні нового компонента, щоб стандартні події були видимі інспекторові об'єктів на стадії проектування або під час виконання програми, треба перевизначити властивості події в секції public або __published.

***class PACKAGE TMyMemo: public TCustomMemo***

***{***

***. . .***

***__published:***

***// OnClick стало видимим інспектору***

***__property  OnClick;***

***};***

Тут тип властивості опущений, оскільки властивість є батьківською і її тип відомий.

Всі стандартні події мають відповідні захищені динамічні методи, успадковані від TControl, імена яких утворені від назви події без частинки "On". Наприклад, події OnClick викликають метод Click.

Необхідність у визначенні абсолютно нових подій виникає досить рідко. Зазвичай досить перевизначити обробку вже існуючих подій.

Якщо ж виникла необхідність створити власну подію, то треба оголосити тип і властивість для події. Якщо тип властивості описаний як __closure, то інспектор об'єктів визначає, що дана властивість є подією, і представляє її на вкладці events.

Нехай потрібно створити власну подію OnClear [1] з одним стандартним параметром типу TObject* і другим параметром булевого типу. Для цього треба оголосити новий тип події. Назвемо його, наприклад, TClear.

Оголошення робиться за допомогою ключового слова __closure. Тип TClear* можна визначити так:

```
typedef void __fastcall ( __closure *TClear)

(System::TObject *Sender, bool CanClear);
```

Отже, визначено тип вказівника на функцію, що приймає два параметри: традиційний для всіх оброблювачів параметр Sender і параметр булевого типу CanClear (цей параметр користувач зможе змінювати).

Тепер нову подію можна визначити так:

```
class PACKAGE TMyMemo: public TCustomMemo

{

private :

TClear FOnClear;

. . .

public:

// власний метод

virtual void __fastcall Clear();

. . .

__published:

__property TClear OnClear = {read=FOnClear , write=FOnClear};

};
```

Оскільки подія OnClear викликає метод Clear, то необхідно змінити метод Clear так, щоб у ньому враховувався новий параметр CanClear:

```
void __fastcall TMyMemo::Clear()

{
```

```
bool CanClear = true;

if (OnClear) OnClear (this, CanClear);

if (CanClear)

{

TCustomMemo::Clear();// виклик батьківського методу

FModified = false;

}

}
```

У цій функції вводиться булева змінна CanClear. При наявності обробника користувача він викликається й CanClose передається в цей обробник другим параметром. Потім, залежно від значення CanClose, викликається або не викликається батьківський метод очищення Clear.

## 4. Додавання методів

Компонентні методи нічим не відрізняються від інших функцій-членів класу. Всі методи (включаючи конструктори), до яких мають право звертатися користувачі компонента, слід оголошувати як public. Всі методи, до яких мають право звертатися похідні об'єкти компоненту, слід оголошувати як protected. Це забороняє користувачам передчасно викликати методи, дані для яких ще не створені.

Якщо треба не додати новий метод у компонент, а перевизначити метод з батьківського класу, то необхідно звернутися до довідки C++Builder і точно відтворити у новому класі оголошення перевизначеного методу (скопіювати його прототип).

Наприклад, для компонента TMyMemo треба перевизначити метод ClearSelection, що очищає текст, виділений у вікні редагування. Оголошення методу ClearSelection у класі-предку TCustomMemo має вигляд:

```
void __fastcall ClearSelection(void);
```

Отже, оголошення й реалізація перевизначеного методу ClearSelection може мати вигляд:

```
class PACKAGE TMyMemo: public TCustomMemo

{

. . .

public:
```

```
void __fastcall ClearSelection(void);
};
void __fastcall TMyMemo::ClearSelection()
{
. . .      // власний код
// виклик батьківського метода TCustomMemo::ClearSelection();
}
```