

МЕТОДИЧНА РОЗРОБКА
для забезпечення лекційних занять

З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

«Інструментальні засоби візуального програмування»

для студентів денної форми навчання за спеціальністю

121 «Інженерія програмного забезпечення»

(код і назва спеціальності)

Розроблено викладачем

_____/ Гапоненко Н.В./

Розглянуто та схвалено

на засіданні ЦК ПП

Протокол № _____ від _____. 20 ____ р.

Голова ЦК ПП

_____/Ланська С.С./

„____” _____ 20 ____ р.

2023

Лекція № 1

з навчальної дисципліни "Інструментальні засоби візуального програмування"

Тема **Архітектура візуального додатку Windows. Основні інструменти інтегрованого середовища розробки. Огляд стандартних компонентів. Обробка подій**

Embarcadero C++ Builder — середовище візуального програмування, призначене для швидкого проектування та розробки додатків. Одне з головних завдань — розробка програм для роботи з базами даних. При роботі в середовищі візуального програмування використання елементів візуальної розробки автоматично веде до застосування концепції об'єктно-орієнтованого програмування (ООП).

Об'єкти в Embarcadero C++ Builder — це елементи, з яких будується додаток: форма, рамка, кнопка, мітка та ін. Об'єктом є візуальний компонент (наприклад, кнопка) в тому вигляді, як він представлений під час розміщення його на формі (проектування) і під час виконання програми. Об'єкти зображуються на екрані до виконання самої програми. Тому таке програмування — візуальне.

1 Архітектура візуального додатку Windows

Під архітектурою розуміють набір основних правил, що визначають організацію системи:

- сукупність структурних елементів системи і зв'язків між ними;
- поведінка елементів системи в процесі їх взаємодії;
- ієрархія підсистем, які об'єднують структурні елементи;
- архітектурний стиль (використовувані методи і засоби опису архітектури, а також архітектурні зразки).

Доступ до апаратних і програмних ресурсів комп'ютера в середовищі Windows здійснюється за допомогою набору системних функцій, реалізується за допомогою інтерфейсу виклику функцій у Windows. Набір цих функцій також іменується програмним інтерфейсом додатку (API, Application Programming Interface). Багато

функцій, яким властиві API, реалізовані бібліотекою класів VCL. Як правило, програми отримують доступ до цих функцій, використовуючи спеціальні методи.

Функції API знаходяться в бібліотеках динамічного завантаження (DLL, Dynamic Link Libraries), завантажуються в пам'ять в процесі виконання програми. Завдяки наявності спільно використовуваних бібліотек DLL, скорочується обсяг займаного простору на диску, а також спрощується подальше оновлення Windows-додатків.

Властивості програми Windows орієнтуються на платформу Win32.

Наведемо характеристики Windows-додатків. Внаслідок того, що програмна архітектура Windows-додатків заснована на принципі обміну повідомленнями, всі вони мають загальні програмні модулі, які зазвичай явно включаються до початкового коду. Завдяки використанню бібліотек компонентів цей процес здійснюється в автоматичному режимі.

Будь-який Windows-додаток включає спеціальну функцію, яка не використовується безпосередньо програмою, а викликається самою операційною системою. Подібна функція отримала найменування функції вікна. Вона викликається на виконання в Windows в тому випадку, коли система передає повідомлення програмі. Завдяки цьому здійснюється взаємодія між програмою і системою. Функція вікна передає повідомлення, використовуючи власні параметри.

Додатки (прикладні програми) Embarcadero C++ Builder є інтерактивними системами, в яких для організації взаємодії між користувачем і програмою використовуються методи (підпрограми), керовані подіями.

Обмін інформацією між об'єктами здійснюється за допомогою повідомлень. Повідомлення є результатом появи подій.

Програмування, кероване подіями, не виключає використання процедурного програмування.

Подія — це відгук на зовнішній вплив. Суть програмування, керованого подіями, складається у відстеженні таких подій, які вимагають реакції програми. У процесі функціонування операційної системи (ОС) Windows виникає багато різних подій, але тільки деякі з них вимагають відгуку конкретного додатка. Середовище Embarcadero C++ Builder пов'язує методи (реакцію) програми з подіями, що відбуваються в ОС. Мова програмування, що використовується в Embarcadero C++ Builder, на самому

нижньому рівні тісно пов'язана з внутрішніми функціями Windows. Цей зв'язок прихований в компонентах, об'єктах і методах. З їх допомогою система візуального програмування спрощує створення Windows-додатків.

Програма для Windows — це набір об'єктів, що посилають і приймають повідомлення. Кожен з об'єктів, що відповідають елементам інтерфейсу Windows, може містити обробники різних повідомлень.

Джерелами повідомлень можуть бути:

1 Події, які генеруються користувачем: введення символів з клавіатури, переміщення миші, натискання або відпускання кнопки миші.

2 Функції Windows, викликані додатком, які в свою чергу призведуть до передачі повідомлень від Windows до додатка.

3 Середовище Windows: (воно може посилати повідомлення додаткам Windows); в Windows, наприклад, є такі повідомлення при роботі з вікном: при активації, закриття, переміщення, оновлення частини вікна, при зміні розміру вікна.

4 Програми Windows можуть надсилати один одному повідомлення динамічного обміну даними (dynamic data exchange — dde) для того щоб обмінюватися даними.

Основним вікном, яке розробляється, є форма. У процесі розробки програми при розміщенні об'єкта на формі (наприклад, кнопки) у візуальному середовищі основні параметри об'єкта (розмір, положення на екрані, колір тощо) відразу відображаються у вигляді реального компонента на формі, а відповідний йому код мовою C++ автоматично записується в вихідний файл форми, який відображає об'єкт в процесі виконання програми. Потім цей вихідний код компілюється у виконуваний машинний код.

Не весь код програми, написаної в Embarcadero C++ Builder, знаходиться у додатку. Невелика його частина фактично є частиною Windows. Наприклад, коди для стандартних вікон діалогу і кнопок повністю отримані від Windows. Embarcadero C++ Builder використовує їх, виконуючи відповідні виклики з Windows DLL (Dynamic Linked Library).

Схематично взаємозв'язок програм з Embarcadero C++ Builder і Windows представлений на рисунку 1.1.

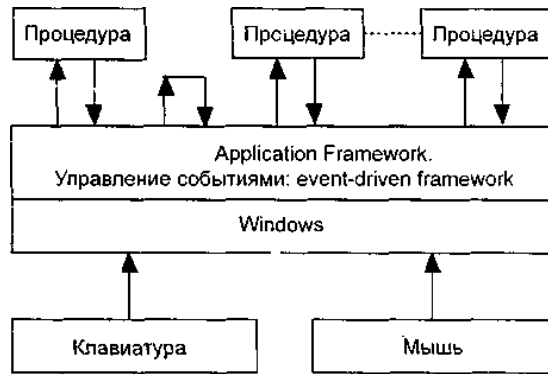


Рис.1.1 – Взаємозв'язок програм з Embarcadero C++ Builder і Windows

Верхня частина — це «Область програми. Повідомлення». Середня частина — це «Область середовища розробки». Нижня частина — це «Область Windows. Події».

Програми Windows використовують методи (підпрограми) обробки подій для управління взаємодією між програмою і користувачем і для реакції на дії ОС. Підпрограма, яка реагує на подію, називається обробником події (Events). Середовище працює з подіями шляхом виклику певних процедур-обробників (Handler) подій. Якщо процедура не пов'язана з даними подією, то воно ігнорується і виконується стандартна реакція системи або не виробляється ніякої дії.

Для виконання реакції на подію середовище шукає метод, наприклад `btnOKClick`, ім'я якого складається з імені об'єкта, що викликав подію (наприклад, `btnOK`-ім'я кнопки), та імені події (наприклад, `Click`)

2 Основні інструменти інтегрованого середовища розробки (IDE) Embarcadero C++ Builder

На рисунку 1.2 показаний Embarcadero C++ Builder відразу після запуску. Це інтегроване середовище розробки (IDE), що включає в себе чотири основних елементи. Нагорі знаходиться головне вікно. Воно містить звичайну лінійку меню , інструментальну панель (ліворуч) і палітру компонентів (багатосторінкова панель праворуч).

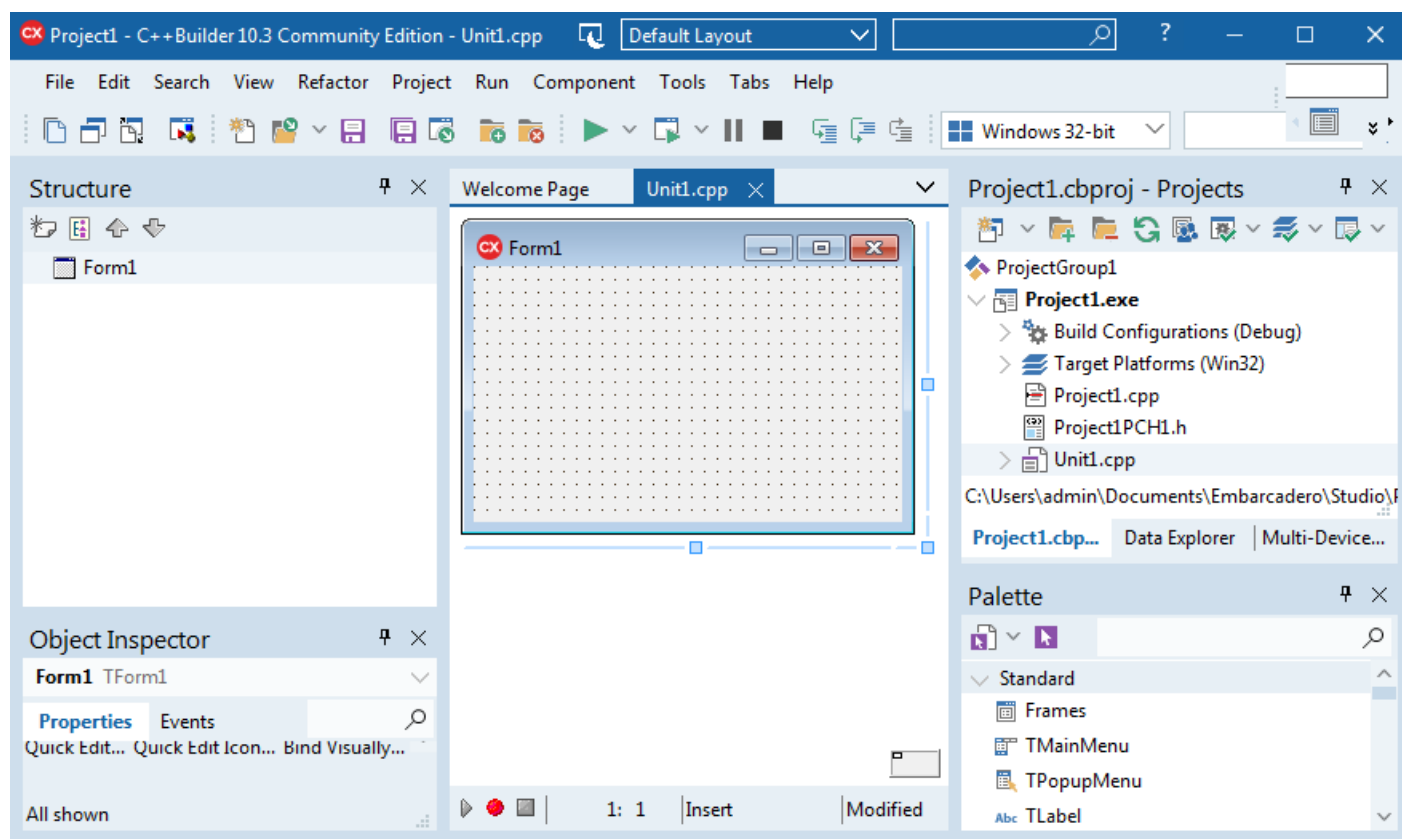


Рис. 1.2 – Вигляд інтегрованого середовища розробки

Правіше інспектора об'єктів розташовується конструктор форм. При створенні проекту конструктор відображає порожню форму. Форма — це центральний елемент візуального програмування. Вона може представляти головне вікно програми, дочірнє вікно, діалогову панель. На ній розміщуються різні елементи керування (типові і найпоширеніші — командна кнопка), звані візуальними компонентами. Існують також і невізуальні компоненти, наприклад, таймери і компоненти зв'язку з базами даних. У інспекторі об'єктів зіставляються подіям компонентів написані програмістом процедури обробки. Це, по суті, і є візуальне програмування, що базується на компонентній моделі.

Нарешті, під конструктором форм знаходиться вікно редактора коду. Його функції не обмежуються редагуванням вихідного тексту програми.

При першому запуску середовища до лівої сторони редактора коду пристиковане вікно оглядача класів, що наведено на рисунку 1.3.

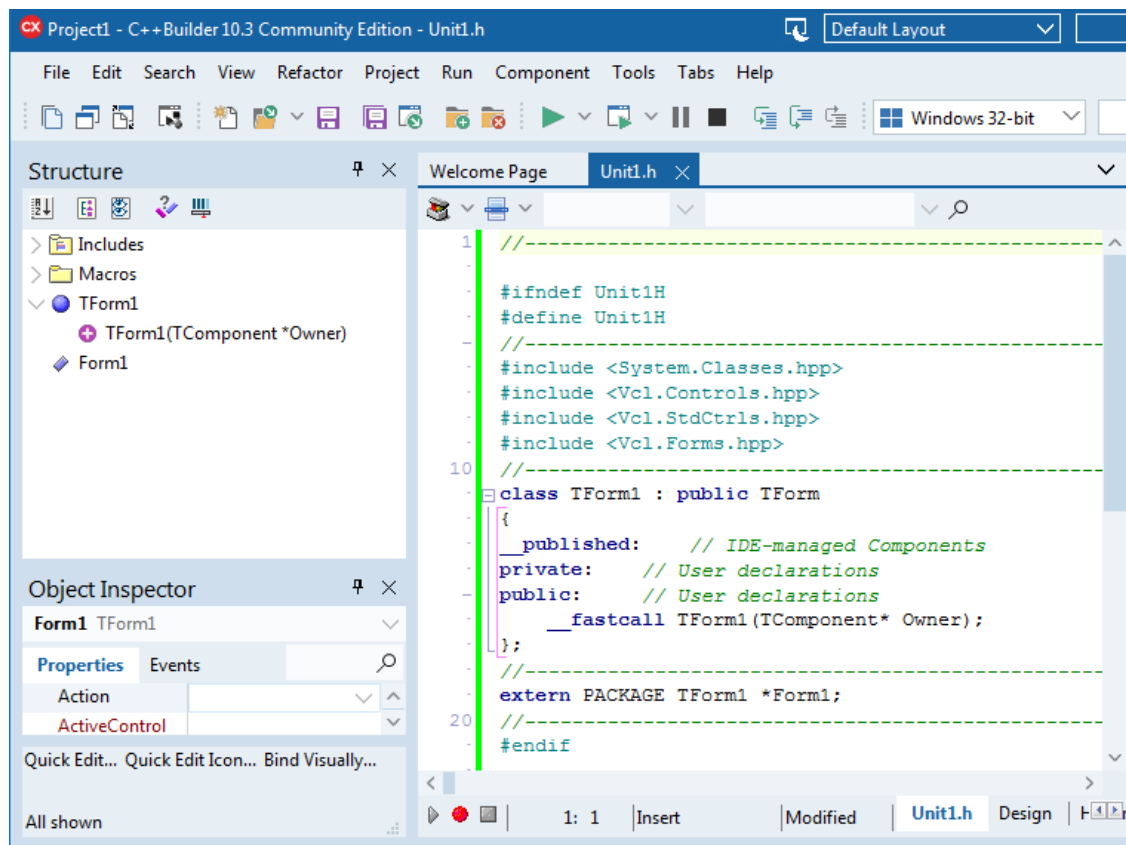


Рис. 1.3 – Вікно редактора коду з оглядачем класів

Інструментальне вікно Embarcadero C++ Builder може бути зістиковано з іншим вікном. У разі стикування по центру вікно стає багатосторінковим, з закладками, що дозволяють перемикатися між сторінками.

В Embarcadero C++ Builder, як і в багатьох інших сучасних програмах, багато операцій реалізуються через контекстні меню, які викликаються натисканням правої кнопки в тому чи іншому вікні.

В редакторі можна відкривати відразу декілька файлів вихідного коду. При цьому він також стає багатосторінковим вікном із закладками. Написи на закладках відображають імена файлів.

Основними елементами середовища програмування Embarcadero C++ Builder є:

- головне вікно , що включає головне меню, панелі інструментів, палітру компонентів;
- текстовий редактор (Code Editor) , що відображається одним або декількома вікнами;
- конструктор форм (Form) , що відображається одним або декількома вікнами - по одному на кожну форму;

– інспектор об'єктів (Object Inspector).

Головне меню Embarcadero C++ Builder складається з наступних підменю:

- File (робота з файлами) ;
- Edit (редагування) ;
- Search (пошук інформації) ;
- View (перегляд інформації) ;
- Project (параметри проекту) ;
- Run (виконання програми) ;
- Component (бібліотека компонентів);
- Database (база даних) ;
- Tools (інструментальні засоби) ;
- Help (довідкова система).

Панелі інструментів, що формуються при установці пакету Embarcadero C++ Builder, містять ряд кнопок з графічним зображенням дій, які вони виконують. Як правило, вони дублюють деякі найбільш часто використовувані команди головного меню, тому панелі інструментів можуть самі розглядатися як своєрідне однорівневе меню.

3 Проект у середовищі візуального програмування Embarcadero C++ Builder

Проект у середовищі візуального програмування Embarcadero C++ Builder створюється за натисканням пункту меню File => Windows VCL Application. В середовищі відразу відкривається візуальна форма.

Для кожного нового проекту в середовищі автоматично відображається вікно проектувальника форм (Form Designer), яке являє собою головне вікно майбутнього програми за замовчуванням має ім'я Form1.

Вікно редактора коду (Code Editor) має заголовок Unit1.cpp і знаходиться (після запуску середовища) позаду вікна проектувальника форм. Редагування програм здійснюється в редакторі. Embarcadero C++ Builder володіє найпотужнішим, вбудованим в редактор графічним відладчиком, що дозволяє знаходити і усувати помилки в коді.

Інспектор об'єктів служить для перегляду опублікованих властивостей компонентів форми (або самої форми), завдання їм значень, завдання оброблювачів подій. У ньому містяться характеристики або активна форма, або її активного компонента. Звичайно інспектор об'єктів виводиться на екран автоматично при запуску середовища, або за допомогою команди головного меню View|Object Inspector (F11).

Основну частину інспектора об'єктів займає двохсторінковий блокнот. На його першій сторінці (Properties) (Властивість) розміщені опубліковані властивості об'єкта із вказівкою імені властивості і його значень. Перед деякими іменами може стояти знак +. Це означає, що властивість є комбінованою і складається з декількох компонентів.

Значення властивостей відображаються або рядками уведення, або комбінованими рядками уведення. Комбінований рядок уведення відрізняється наявністю в правій частині кнопки із зображенням трикутника, спрямованого вниз, нажавши яку можна відкрити список, що випадає, припустимих значень властивості.

У деяких рядках уведення перебуває кнопка із зображенням трьох крапок. Це означає, що при натисканні кнопки виводиться вікно, за допомогою якого задається ряд параметрів комбінованої властивості.

Друга сторінка (Events) (Подія) блокнота містить імена подій, які може обробляти даний об'єкт, із вказівкою імен оброблювачів подій або відсутністю таких. Якщо імені оброблювача подій нічого не відповідає, то дана подія об'єктом не обробляється. При активізації оброблювача події відбувається перехід до сторінки текстового редактора, у якій перебуває текст цього оброблювача.

У верхній частині інспектора об'єктів перебуває комбінований рядок уведення, список якої містить імена активної форми й всіх її компонентів.

Інспектор об'єктів — це один із трьох найцінніших інструментів розробки інтерфейсу додатка. Два інших — це проектувальник форм (Form Designer) і редактор коду (Code Editor).

Висновки

Програми Windows використовують методи (підпрограми) обробки подій для управління взаємодією між програмою і користувачем і для реакції на дії ОС. Програмний код, який пише програміст в Embarcadero C++ Builder, забезпечуватиме реакцію на події. Підпрограма, яка реагує на подію, називається обробником події (Events). Середовище працює з подіями шляхом виклику певних процедур-обробників подій. Якщо процедура не пов'язана з подією, то остання ігнорується і виконується стандартна реакція системи або не виробляється ніякої дії.

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
 - 1 Що таке процедурне програмування?
 - 2 Що являють собою принципи об'єктно-орієнтованого програмування?
- 2 Для узагальнення та перевірки засвоєного матеріалу на лекції
 - 1 Що являє собою архітектура візуальних додатків?
 - 2 Як використовуються принципи подійно-орієнтованого програмування?
 - 3 Що являє собою інтегроване середовище розробки Embarcadero C++ Builder?
 - 4 Охарактеризувати головне вікно середовища розробки Embarcadero C++ Builder.

Лекція

з навчальної дисципліни "Інструментальні засоби візуального програмування"

Тема Форма, її властивості та події. Компоненти для роботи зі списками

Додаток для Windows складається з одного або декількох вікон, звичайно званих в Embarcadero C++ Builder формами. При запуску Embarcadero C++ Builder автоматично створює основну форму з ім'ям Form1. Форма є компонентом типу TForm.

В процесі розробки додатку на формі розміщуються візуальні і не візуальні компоненти, створюючи інтерфейс додатку. Форма є прямокутним вікном з рамкою. Більшість вікон має область заголовка, в якій розташовані піктограма заголовка, заголовок і ряд кнопок (у правому верхньому кутку), що дозволяють скрутити (на панель задач Windows), розвернути (відновити) і закрити вікно. На формі можна розмістити головне меню (під областю заголовка) і рядок стану (панель статусу), звично в нижній частині вікна. При необхідності на формі можуть автоматично з'являтися смуги прокрутки. Решта частини вікна називається клієнтською областю. У ній можна розміщувати елементи управління процесом виконання додатку, виводити текст і графіку і ін.

1 Форма

Робота над новим проектом починається із створення стартової форми — головного вікна програми.

Стартова форма створюється шляхом зміни значень властивостей форми Form1 (налаштування форми) і додавання до форми необхідних компонентів (полів введення, полів виведення текстової інформації, командних кнопок). Основні властивості форми, які визначають її вигляд і поведінка під час роботи програми, наведено в таблиці 3.1. Таблиця 3.1 – Властивості форми (об'єкту Form)

Властивість	Опис
-------------	------

Name	Ім'я форми. У програмі ім'я форми використовується для управління формою і доступу до компонентів форми
Caption	Текст заголовка
Width	Ширина форми Розмір форми, а також розмір інших компонентів задають в пікселях, тобто точках екрану.
Height	Висота форми
Top	Відстань від верхньої межі форми до верхньої межі екрану
Left	Відстань від лівої межі форми до лівої межі екрану
BorderStyle	Межа може бути звичайною (bsSizeable), тонкої (bsSingle) або відсутньою (bsNone). Змінити розмір вікна з тонкою межею не можна. Якщо межа відсутня, то на екран під час роботи програми буде виведено вікно без заголовка, положення і розмір якого змінити не можна
BorderIcons	Значення властивості визначає, які кнопки управління вікном будуть доступні користувачеві під час роботи програми. Задається шляхом привласнення значень уточнюючим властивостям biSystemMenu, biMinimize, biMaximize і biHelp. Властивість biSystemMenu визначає доступність кнопки Згорнути і кнопки системного меню, biMinimize - кнопки Згорнути, biMaximize-кнопки Розвернути, biHelp - кнопки виведення довідкової інформації
Icon	Значок в заголовку діалогового вікна, що позначає кнопку виведення системного меню
Color	Колір фону. Колір можна задати, вказавши назву кольору або прив'язку до поточної колірної схеми операційної системи. У другому випадку колір визначається поточною колірною схемою, вибраним компонентом прив'язки і міняється при зміні колірної схеми операційної системи
Font	Шрифт. Шрифт, використовуваний "за замовчуванням" компонентами, що знаходяться на поверхні форми. Зміна властивості Font форми приводить до автоматичної зміни властивості Font компоненту, розташованого на поверхні форми. Тобто компоненти успадковують властивість Font від форми (є можливість заборонити спадкоємство)

Для зміни значень властивостей об'єктів, у тому числі і форми, використовується вкладка Properties (Властивості) діалогового вікна Object Inspector. У лівій колонці цієї вкладки перераховані властивості вибраного об'єкту, в правій вказані значення властивостей.

Форма — це звичайне вікно. Тому розмір форми можна змінити точно так же, як розмір якого вікна Windows, тобто шляхом пересування межі. Після закінчення переміщення кордону значення властивостей Height і width автоматично зміняться. Вони будуть відповідати встановленим розміром форми. Положення діалогового вікна на екрані після запуску програми відповідає положенню форми під час розробки, яке визначається значенням властивостей Top (відступ від верхньої межі екрану) і Left (відступ від лівої межі екрану).

Складний додаток звичайно містить декілька форм. За умовчанням всі форми створюються автоматично, і перша із створених форм вважається головною.

Якщо форма при розробці проекту переведена у вікні Forms із списку Autocreate forms в список доступних форм (Available forms), то оператор її створення виключається з файлу проекту і для використання форми треба в ході виконання додатку створити екземпляр цієї форми. Створити форму, що знаходиться в списку доступних форм можна час виконання додатку, наприклад `Application.CreateForm(TForm1, Form1);`

При створенні і використуванні форми відбуваються наступні події, для реакції на які можна сформулювати певні методи:

- OnCreate — при створенні форми;
- OnShow — при показі форми;
- OnActivate — при активізації форми;
- OnResize — при зміні розмірів форми;
- OnPaint — при перемальовуванні форми.

Якщо форма створена, зробити її видимою можна за допомогою методів Show або ShowModal або невидимою за допомогою методу Hide. Закрити форму можна методом Close. При виконанні методу Close виникає подія onCloseQuery В його обробник передається параметр CanClose, що визначає, чи можна продовжити закриття форми. За умовчанням значення параметра CanClose рівне True.

Якщо одна форма (наприклад, Form1) виконує які-небудь дії з іншою формою (наприклад, Form2), треба організувати їх взаємодію за допомогою uses.

Форма може бути модальною і немодальною. Немодальна форма дозволяє перемкнутися на іншу форму додатку без свого закриття. Модальну форму треба обов'язково закрити перед зверненням до будь-якої іншої форми додатку.

Форми, які відображають різні повідомлення і вимагають від користувача введення якої-небудь інформації, називають діалоговими (діалогами).

У Windows (і в Embarcadero C++ Builder) є два основні типи додатків: однодокументні (SDI - Single Document InterFace) і багатодокументні (MDI - Multiple Document Interface). Однодокументні додатки складаються з однієї або декількох форм (вікон), відносно незалежних одне від одного. У SDI-додатку жодне вікно на екрані не містить в собі інші вікна (наприклад, в Embarcadero C++ Builder). У MDI-додатку головне вікно (батьківське) містить дочірні вікна, розміщені в його межах.

Для закриття форми використовується метод Close(). При закритті основної форми припиняється робота всього додатку. Метод Close() не знищує створений екземпляр форми, і її можна знову викликати на екран за допомогою методів Show або Showmodal. Знищення форми можна виконати за допомогою методів Release, Free або Destroy. При закритті і знищенні форми відбуваються наступні події OnCloseQuery, OnClose, OnDeactivate, OnHide, OnDestroy.

Стиль форми визначається її властивістю FormStyle. Стиль форми можна змінювати в процесі виконання додатку. Він може приймати одне із значень:

- FsNormal — стандартний стиль, використовуваний для більшості вікон;
- FsMDIChild — дочірня форма в MDI;
- FsMDIForm — батьківська форма в MDI;
- FsStayOnTop — форма відображається поверх інших форм.

Властивість ActiveControl використовується для вибору елемента керування, який буде активним при зверненні до форми. Взяті разом, властивості AutoScroll, HorzScrollBar і VertScrollBar управляють лінійками прокрутки для форми.

За допомогою властивостей ClientWidth і ClientHeight ви можете вказати ширину і висоту робочої області замість ширини і висоти всієї форми.

Властивість `Position` визначає розмір і положення форми при першому відображенні. Можливі три варіанти: `poDesigned`, `poDefault` і `poScreenCenter`.

Властивість `WindowState` може бути зчитана для визначення поточного стану форми (розгорнута, згорнута або має нормальний розмір). Можливі значення: `wsMinimized`, `wsMaximized` і `wsNormal`.

2 Властивості, доступні тільки під час виконання програми

Доступ до деяких властивостей можливий тільки через код під час виконання програми.

При читанні властивості `ActiveMDIChild` повертається вказівник на активне в даний момент підлегле вікно MDI. Властивість доступна тільки для читання.

Полотно (`canvas`) представляє поверхню форми, доступну для рисування. Властивість `Canvas` забезпечує вам доступ до полотна форми. Використовуючи цю властивість, ви можете відображати у формі растрові зображення, лінії, криві або текст під час роботи програми.

Властивість `ClientRect` містить координати робочої області форми.

Властивість `ModalResult` використовується для закриття модального вікна. Якщо у вас є діалогове вікно з кнопками `OK` і `Cancel`, ви можете давати `ModalResult` значення `mrOK` при натисканні кнопки `OK` або `mrCancel` при натисканні кнопки `Cancel`. Викликає вікно форма може вважати `ModalResult` для визначення того, яка кнопка була натиснута. До інших можливих значень відносяться `mrYes`, `mrNo` і `mrAbort`.

Метод `BringToFront ()` використовується для знаходження даної форми поверх всіх інших форм програми.

Метод `Print ()` виводить на друк вміст форми. Друкується тільки робоча область форми, без підпису, рядки заголовка або рамок.

Метод `SetFocus ()` активізує форму і переміщує її поверх інших вікон.

Кілька методів форм виконують специфічні MDI-операції. Метод `ArrangeIcons ()` впорядковує значки всіх згорнутих підлеглих вікон у головному вікні. Метод `Cascade()` має каскадом всі несвернуті підлеглі вікна. Метод `Tile()` має відкриті підлеглі вікна мозаїкою. Метод `Next()` активізує (розпорядженні зверху) наступне вікно у списку

підлеглих вікон, а метод Previous() - попереднє. Методи MDI застосовні тільки до головних вікон MDI.

Форми можуть реагувати на самі різні події.

Подія OnActivate відповідає активізації форми. Форма може бути активізована в результаті її первісного створення, при переході від однієї форми в іншу чи при перемиканні між додатками.

При закритті програми виникає подія OnClose. Подія OnCreate відбувається при первісному створенні форми. Подія OnDestroy протилежно OnCreate. Подія OnDragDrop відбувається при приміщенні об'єкта в форму. Його можна використовувати, якщо ваша форма підтримує техніку drag - and - drop. Події OnMouseDown , OnMouseMove і OnMouseUp використовуються для реакції на клацання мишею на формі. Подія OnPaint відбувається тоді, коли форма з якої-небудь причини вимагає перемальовування. Подія OnResize виникає при кожній зміні розміру форми. Подія OnShow відбувається перед тим, як форма стане видимою.

Форма може містити, наприклад, наступні компоненти для роботи з текстом.

Label — мітка, що виводить незмінний текст. Основною властивістю компонента є Caption, яка містить текст. Для виводу тексту на мітку в два і більше рядків треба встановити її властивість WordWrap = True. Розмір міток визначається також властивістю Autosize.

Edit — рядок введення. TEdit — стандартний керуючий елемент Windows для введення. Він може бути використаний для відображення короткого фрагмента тексту і дозволяє користувачу вводити текст під час виконання програми. Основною властивістю компонента є Text, яка містить текст. Найчастіше в Borland Delphi використовуються однорядкові редактори Edit і MaskEdit.

Події OnEnter і OnExit виникають при отриманні і втраті фокусу компоненту.

Ряд властивостей мають практично, всі візуальні компоненти. Наприклад:

- Enabled, Visible — доступність та видимість компоненту відповідно ;
- Font — шрифт для представлення тексту;
- Height, Width — відповідно висота та ширина компоненту;
- Hint, ShowHint — текст оперативної підказки та чи показувати її;
- Name — ім'я компоненту;

– PopupMenu — спливаюче меню.

Крім того, однорядковий редактор Edit має окремі властивості:

– AutoSize — при True висота вікна автоматично підганяється під текст;

– BorderStyle — стиль бордюра;

– Color — колір поля вікна;

– ReadOnly — тільки для читання.

Властивість PasswordChar застосовується тільки для компонентів Edit при використуванні компоненту для введення пароля. Воно задає символ для відображення в полі введення при введенні пароля. При цьому у властивість Text вводяться символи пароля, а у вікні введення відображаються символи, визначені у властивості PasswordChar. За умовчанням PasswordChar = #0 і в рядку редагування відображаються реально введені символи.

Є ряд компонентів для відображення, введення і редагування тексту користувачем в процесі виконання додатку, вони наведені на рисунку 3.1.

Label (метка)	Standard	Отображение текста с помощью свойства Caption
StaticText	Additional	Аналогичен метке. Имеет рамку – бордюр
Panel	Standard	Контейнер группирования компонентов. Может использоваться для вывода текста с помощью свойства Caption
Edit	Standard	Отображение, ввод и редактирование текста в свойстве Text
MaskEdit	Additional	Для отображения, ввода и редактирования данных в соответствии с заданным шаблоном
Memo	Standard	Отображение, ввод и редактирование многострочных текстов. Основное свойство – Lines
RichEdit	Win32	Многострочное окно редактирования текста в формате RTF. Имеет выбор шрифта, поиск и пр.
ListBox	Standard	Стандартное окно списка для выбора строки. Основное свойство Items
CheckList Box	Additional	Подобен компоненту ListBox. Имеет индикатор CheckBox в каждой строке
Combo Box	Standard	Редактируемый список. Можно выбирать и редактировать строки списка
StringGrid	Additional	Отображение и редактирование текста в таблице из строк и столбцов. Основное свойство – Cells

Рис. 3.1 – Компоненти для відображення, введення і редагування тексту

Мемо — редактор тексту. ТМемо може переносити слова, зберігати в Clipboard фрагменти тексту і відновлювати їх, і інші основні функції редактора. Властивість Lines (типа TStrings) містить текст вікна у вигляді списку рядків. Початковий текст можна ввести в процесі розробки додатку. Для цього треба вибрати властивість Lines і клацнути у області його значення. Для додавання призначений метод Add. Весь текст міститься у властивості Text. В процесі виконання додатку можна вводити текст з

файлу за допомогою методу LoadFromFile, зберігати текст з вікна в текстовому файлі за допомогою методу SaveToFile.

Основною властивістю, призначеною для виводу тексту в компоненти типа Label, StaticText і Panel є властивість компонентів Caption. Вирівнювання тексту усередині компоненту визначається властивістю Alignment.

Компонент MaskEdit в порівнянні з компонентом Edit додатково надає можливість введення даних за шаблоном. За допомогою маски можна обмежити кількість символів і задати тип символів, що вводяться. Маска задається у властивості EditMask. Викликається редактор шаблонів клацанням в полі значення властивості EditMask. У масці можна застосовувати символи, приведені на рисунку 3.2.

	« » означает, что в свойстве Text подавляются ведущие пробелы; если нет « », то – конечные
>	Все символы после него должны вводиться в верхнем регистре
<	Все символы после него должны вводиться в нижнем регистре
<>	Анализ регистра не производится
\	Символ, следующий за ним, является литеральным
L	В данной позиции должна быть буква
I	В данной позиции может быть буква или ничего
A	В данной позиции должна быть буква или цифра
a	В данной позиции может быть буква или цифра или ничего
C	В данной позиции должен быть любой символ
c	В данной позиции может быть любой символ или ничего
0	В данной позиции должна быть цифра
9	В данной позиции может быть цифра или ничего
#	В данной позиции может быть цифра, знак + или - или ничего
:	Символ используется для разделения часов, минут и секунд
/	Символ используется для разделения месяцев, дней и годов в датах
	Пробел означает автоматическую вставку пробела в текст

Рис. 3.2 – Символы шаблону маски

Багаторядкові редактори призначені для створення і редагування текстів в процесі виконання додатку. Деякі властивості багаторядкових текстових редакторів показані на рисунку 3.3.

Lines	В Инспекторе объектов – для создания текста редактора
ScrollBars	Для определения наличия полос прокрутки текста
MaxLength	Максимально допустимое количество символов, вводимых пользователем. По умолчанию 0, и длина текста не ограничена
WantReturns	Разрешение ввода новой строки от нажатия клавиши Enter
WantTabs	Разрешение вводить в текст символы табуляции
WordWrap	Допустимость переноса для длинных строк

Рис. 3.3 – Властивості багаторядкових текстових редакторів

Основна властивість багаторядкових редакторів — Lines (типа TStrings) містить текст вікна у вигляді списку рядків. В процесі виконання додатку можна вводити текст з файлу за допомогою методу LoadFromFile; зберігати текст з вікна в текстовому файлі за допомогою методу SaveToFile, заносити або прочитувати текст з рядка із заданим номером за допомогою властивостей Lines і Strings[Index: Integer]. Початковий номер рядка = 0; додати рядок у вікно можна за допомогою методу Add або Append.

Компонент RichEdit працює у форматі RTF. Для зміни формату тексту, що вводиться, треба помістити на форму компонент FontDialog — діалогу вибору шрифту. Потім в процесі виконання додатку з тексту програми підключити для компоненту RichEdit властивість SelAttributes — вибору атрибутів шрифту методом Assign.

Компоненти для роботи із списками - ListBox, CheckListBox і ComboBox

Компоненти ListBox і ComboBox відображають списки рядків. Вони відрізняються тим, що ListBox тільки відображає дані і дозволяє користувачу вибрати з них необхідні рядки, а ComboBox дозволяє також редагувати дані. CheckListBox — це список з індикаторами.

ListBox — простий список, є прямокутним вікном, в якому розташовуються його рядки. Якщо кількість рядків більше, ніж може поміститися у видимій області списку, то справа у вікні автоматично з'являється смуга прокрутки. Орієнтація смуги прокрутки, а також число колонок (стовпців), які одночасно видно у області списку, залежать від значення властивості Columns (типу integer).

За умовчанням Columns = 0. При цьому всі елементи списку розташовані в один стовпець. Якщо властивість Columns >= 1, то у області списку завжди присутня горизонтальна смуга прокрутки, а список розбивається на таку кількість колонок, щоб можна було проглянути всі його рядки шляхом прокрутки списку по горизонталі.

Список є сукупністю рядків. Основна властивість компонентів — Items (типа Strings). Неопублікована властивість Count (цілого типу) визначає кількість рядків списку. Номер першого елемента списку = 0. За допомогою властивості Sorted = True можна сортувати список на етапі розробки або виконання додатку.

В процесі роботи із списком можна виконувати ряд дій, зокрема:

- очистити список методом `Clear`; наприклад: `ListBox1.Clear()`;
- завантажити в список вміст текстового файлу за допомогою методу `LoadFromFile`;
- зберегти список в текстовому файлі за допомогою методу `SaveToFile`;
- видалити заданий рядок методом `Delete`; наприклад: `ListBox1.Items.Delete(4)`;
- додати в кінець списку рядок методом `Add`;
- додати в кінець списку рядок методом `Append`;
- додати в список одразу декілька рядків з іншого списку методом `AddStrings`; наприклад: `ListBox1.Items.Add("Рядок в кінець списку")`;
- вставити в задане місце списку рядок методом `Insert`;
- порівняти вміст двох списків за допомогою функції `Equals`;
- копіювати вміст одного списку в іншій методом `Assign`; наприклад, в `ListBox2` з `ListBox1`: `ListBox2.Items.Assign(ListBox1.Items)`;
- звернутися до будь-якого рядка списку з метою його обробки (аналізу, коректування, видалення).

Приклад роботи з простими списками рядків наведений в лістингу 4.3.

Лістинг 4.3 – Приклад роботи з простими списками рядків

```
// Вибір рядка з ListBox1 і виведення на мітку по клацанню в ListBox1
Label1.Caption := "Вибране прізвище:" +
ListBox1.Items[ListBox1.ItemIndex]; // звернення до вибраного рядка
// Вибір рядка з CheckListBox1 і аналіз її поміченості по клацанню в
// CheckListBox1
Label1.Caption := "Вибране прізвище " +
// Звернення до вибраного рядка:
CheckListBox1.Items[CheckListBox1.ItemIndex];
// Звернення до індикатора рядка:
if CheckListBox1.Checked[CheckListBox1.ItemIndex] = True
Label2.Caption := "Помічений";
else Label2.Caption := "Не помічений";
//Приклад оператора привласнення елементам списку нових значень в
//процесі виконання додатку:
var i : integer;
...
for i:=0 to 10
ListBox1.Items.Add("Nomer="+IntToStr(i));
Або у такий спосіб:
for i:=0 to ListBox1.Items.Count
ListBox1.Items[i] := "Nomer="+IntToStr(i);
```

Властивість `Items` типа `TStrings` є масивом рядків. Доступ до рядка можна дістати по номеру, наприклад, вибраного рядка. Для вибору рядка треба під час виконання

додатку встановити фокус на компоненті, наприклад, клацнути на ньому. Потім для вибору рядка можна використовувати клавіші (із стрілками) для переміщення курсора, Home і End (для переходу в початок або кінець списку відповідно).

За умовчанням в списку можна вибрати один рядок. При цьому властивість `MultiSelect = False`. Для вибору більше одного рядка властивості `MultiSelect` треба привласнити значення `True`.

`ComboBox` — комбінований список. Він об'єднує поле редагування (верхній рядок) і список. При цьому можна вибрати для редагування рядок із списку або вводити значення рядка безпосередньо в полі редагування. На відміну від простого списку (`ListBox`) комбінований список не має горизонтальної смуги прокрутки і властивості `Columns` — для установки кількості стовпців.

Властивість `Style` типа `TComboBoxStyle` визначає зовнішній вигляд і поведінку комбінованого списку. Воно може приймати одне з наступних значень:

- `csDropDown` — список, що розкривається, з полем редагування;
- `csSimple` — поле редагування з постійно розкритим списком; щоб був видний весь список, треба збільшити його висоту (властивість `Height`);
- `csDropDownList` — розкривається список, що допускає тільки вибір елементів із списку;
- `csOwnerDrawFixed` — список з елементами заданої фіксованої висоти (властивість `ItemHeight`);
- `csOwnerDrawVariable` — список з елементами, які можуть мати різну висоту.

Властивість `DropDownCount` визначає кількість рядків в списку, що розкривається. Якщо їх недосить, для всіх рядків списку автоматично з'являється вертикальна смуга прокрутки. Якщо список менше, ніж задано в `DropDownCount`, кількість рядків випадного списку зменшується.

Властивість `Text` містить значення вибраного рядка. Воно може бути заповнене в процесі розробки довільним текстом.

Методи `LoadFromFile` та `SaveToFile` дозволяють відповідно загрузити у список рядки з текстового файлу та зберегти рядки з списку у текстовому файлі.

З кожним рядком списку може бути пов'язаний певний об'єкт. Доступ до цих об'єктів здійснюється властивістю `Objects [ int Index ]` — індексованих масивом:

Занесення до списку рядків з об'єктами здійснюється методами `AddObject` і `InsertObject`.

Обидва методи додають до списку рядок з текстом `S` і пов'язують з нею покажчик на об'єкт `AObject`. Метод `AddObject` додає рядок в кінець списку, а `InsertObject` — в позицію `Index`.

З рядками списку можуть зв'язуватися будь-які об'єкти. Заносити у властивість `Objects` можна будь-які компоненти VCL, оскільки всі вони успадковують клас `TObject`.

Висновки

Як і будь-який візуальний компонент, форма має певні характеристики: властивості, методи і події. Окрім загальних для всіх візуальних компонентів форма має і специфічні властивості, методи і події.

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
 - 1 Як приєднати код до обробника події?
 - 2 Яка різниця між методом і обробником події?
 - 3 Що являє собою палітра компонентів Embarcadero C++ Builder?
- 2 Для узагальнення та перевірки засвоєного матеріалу на лекції
 - 1 Охарактеризувати основні властивості форм.
 - 2 Які є методи форми?
 - 3 Які події форми?
 - 4 Для чого призначені компоненти `Edit`, `Memo`?
 - 5 Які властивості мають компоненти `Label`, `Edit`, `Memo`?

Компонент таблиця комірок. Компоненти кнопки. Панелі інструментів

Актуальність і основна ціль теми:

- ознайомити з основними властивостями та методами компоненту таблиця комірок;
- ознайомити з основними властивостями та методами кнопок.

Студент повинен знати:

- основні властивості та методи компоненту – таблиця комірок;
- основні властивості та методи компоненту – кнопка.

Студент повинен уміти:

- використати основні властивості та методи таблиці комірок для розробки програм;
- використати основні властивості та методи кнопок для розробки програм.

Основні питання:

1. Основні властивості компоненту таблиця комірок (StringGrid).
2. Властивість Options.
3. Основні події компоненту.
4. Подія OnSelectCell.
5. Типи кнопок.
6. Загальні властивості кнопок.
7. Кнопки BitBtn.
8. Радіокнопки.

Питання для самоперевірки і контролю:

1. Для чого призначений компонент-таблиця?
2. Які властивості компоненту-таблиці вам відомі?
3. Які методи компоненту-таблиці вам відомі?

4. Як використати подію OnSelectCell?
5. Для чого призначені різні типи кнопок?
6. Як реагують на події кнопки?
7. Які загальні властивості кнопок?
8. Які властивості кнопки BitBtn?
9. Яке призначення радіокнопки?

Рекомендована література:

1. Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.
2. Архангельский, А. Я. Delphi 7. Справочное пособие [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2004. - 1024 с.

Таблиця комірок - компонент StringGrid

Компонент **StringGrid** є таблицею для відображення даних у вигляді рядків і стовпців. Цей компонент також називають таблицею, сіткою рядків або просто сіткою. Таблиця може мати смуги прокрутки.

Дані таблиці можуть бути тільки для читання або редагованими. Таблиця може мати задану кількість фіксованих (верхніх) рядків і стовпців (зліва). У них можна задати заголовки стовпців (у фіксованих рядках) і пояснення до вмісту кожного рядка (у фіксованих стовпцях).

Список основних опублікованих властивостей StringGrid даний в табл.1. Всі властивості доступні під час виконання додатку. Виводити тексти в таблицю можна програмно в кожен осередок або по стовпцях або по рядках за допомогою методів класу TStrings.

Таблиця 1 - Основні опубліковані властивості компоненту StringGrid

ColCount	Колличество строк таблицы
RowCount	Колличество столбцов таблицы
FixedCols	Колличество фиксированных (крайних левых) столбцов таблицы
FixedRows	Колличество фиксированных (верхних) строк таблицы
DefaultColWidth	Ширина столбца в пикселях
DefaultRowHeight	Высота строки в пикселях
Color	Цвет фона заповняемой части таблицы
FixedColor	Цвет фона фиксированной части таблицы
GridLineWidth	Толщина линий сетки таблицы в пикселях
ScrollBars	Возможность отображения полос прокрутки
Options	Параметры таблицы

Властивість ScrollBars визначає можливість відображення смуг прокрутки за допомогою одного з наступних значень:

- ssNone — смуги прокрутки не допускаються;
- ssHorizontal - допускається горизонтальна смуга;
- ssVertical - допускається вертикальна смуга;
- ssBoth - допускаються обидві смуги прокрутки (за умовчанням).

Якщо смуги прокрутки встановлені, вони з'являються і зникають в процесі виконання додатку автоматично, в міру необхідності.

Властивість Options

Властивість Options є безліччю параметрів, кожний з яких типа Boolean. Воно може приймати набір з наступних значень:

- goFixedVertLine і goFixedHorzLine - відображати в сітці для фіксованих елементів вертикальні і горизонтальні лінії відповідно;
- goVertLine і goHorzLine - відображати в сітці для області даних вертикальні і горизонтальні лінії відповідно;
- goRangeSelect - дозволити користувачу вибір розміру осередків; ігнорується при goEditing = True;

- `goDrawFocusSelected` - вибраний осередок виділяється прямокутною рамкою і кольором;
- `goRowSizing` і `goColSizing` - можна змінювати висоту рядків і ширину стовпців з даними;
- `goRowMoving` і `goColMoving` - можна переміщати рядки і стовпці з даними за допомогою миші;
- `goEditing` - можна редагувати дані;
- `goTabs` - допускається переміщення між осередками за допомогою клавіші Tab;
- `goRowSelect` - можна вибрати весь рядок;
- `goAlwaysShowJiditor` - сітка не блокує режим редагування;
- `goThumbTracking` - дані обновляються в процесі прокручування таблиці (а не тільки після відпуску бігунка прокрутки).

В процесі виконання додатку можна використовувати ряд неопублікованих властивостей компоненту. Зокрема:

- `LeftCol` і `TopCol` - індекси першого видимого на екрані прокручуваного (з даними) стовпця і першого видимого прокручуваного рядка;
- `ColWidths[int Index]` і `RowHeights[int Index]` - дозволяють задати в пікселях ширину стовпця і висоту рядка з номером Index.

Кількість рядків і стовпців можна змінити з програми за допомогою властивості `ColCount`. Наприклад, за допомогою операторів:

```
// - додати стовпець
StringGrid1.ColCount := StringGrid1.ColCount + 1;
// - додати рядок
StringGrid1.RowCount := StringGrid1.RowCount + 1;
```

Можна очистити доданий рядок. Наприклад, оператором:

```
StringGrid1.Rows[StringGrid1.RowCount - 1].Clear;
```

Видалення останнього рядка можна виконати оператором:

```
StringGrid1.RowCount := StringGrid1.RowCount - 1;
```

Основні властивості компонентів, що визначають дані, що відображаються у табл.2.

Таблиця 2 - Основні властивості компонентів, що визначають дані

Cells [col] [row]	Рядок із комірки з індексами col, row
Cols [Index]	Список рядків із стовпця з номером Index
Rows [Index]	Список стовпців із рядка з номером Index

При зверненні до вмісту осередку в списку її номерів першим указується номер стовпця (Col), другим - номер рядка (Row).

Рядки і стовпці нумеруються, починаючи з нуля до RowCount - 1 і до ColCount - 1 відповідно.

Всі ці властивості доступні під час виконання додатку. Виводити тексти можна програмно по окремих осередках, використовуючи властивість Cells, або відразу по стовпцях (властивістю Cols) або по рядках (властивістю Rows).

За допомогою властивостей Cols і Rows можна виконувати операції над рядками і стовпцями. Наприклад, копіювати їх значення в простий список за допомогою операторів:

```
// -копіювання стовпця  
ListBox1.Items.Assign(StringGrid1.Cols[3]);  
// - копіювання рядка  
ListBox2.Items.Assign(StringGrid1.Rows[4]);
```

В основному компонент StringGrid використовується для вибору або коректування користувачем значень, відображених в елементах таблиці.

Подія OnSelectCell

Серед безлічі подій компоненту StringGrid треба відзначити подію **OnSelectCell**. Вона виникає у момент вибору користувачем осередку (клацанням в осередку). Приклад заголовка методу для реакції додатку на вибір елементу таблиці:

```
void __fastcall TForm1::StringGrid1SelectCell(TObject *Sender,
int ACol, int ARow, bool &CanSelect)
{
    if ((ARow == 1) && (ACol == 1)) CanSelect = false;
}
де ACol, ARow - параметри, які містять номери стовпця і рядка
відповідно;
```

CanSelect - параметр, який визначає допустимість вибору даного осередку; з його допомогою можна дозволити (для CanSelect = True) або заборонити (для CanSelect = False) вибір будь-якого елементу таблиці.

Наприклад оператором можна заборонити вибір осередків 1-й рядка і 1-го стовпця:

```
if ((ARow = 1) or (ACol = 1)) CanSelect := false;
або
if ((ARow = 1) and (ACol = 1)) CanSelect := false;
StringGrid1.Cells[2][1] = "66";
```

При виборі осередку можна використовувати операторів для обробки даних вибраного рядка. Наприклад, можна вивести на мітки номера рядка і стовпця вибраного осередку і її значення операторами:

```
Label1.Caption := "Вибраний осередок " + IntToStr(ARow) + ":" +
IntToStr(ACol);
Label2.Caption := StringGrid1.Cells[ACol][ARow]; //вивід вмісту.
```

Компонент StringGrid може реагувати на різні події. Наприклад, на наступні:

OnClick - при клацанні на компоненті;

OnDblClick - при подвійному клацанні на компоненті;

OnSelectCell - при виборі осередку компоненту.

Класифікація, загальні властивості та події кнопок.

VCL містить ряд різних кнопок - елементів управління. Кнопки звичайно використовуються у вікнах діалогу. У табл.3 приведені імена типів кнопок і сторінки компонентів палітри, в яких вони розташовані.

Таблиця 3 - Типи кнопок системи Borland Delphi

<i>Ім'я компонента</i>	<i>Страница палитры</i>	<i>Назначение компонента</i>
<i>TButton</i>	<i>Standard</i>	<i>Обычная кнопка</i>
<i>TBitBtn</i>	<i>Additional</i>	<i>Графическая кнопка</i>
<i>TSpeedButton</i>	<i>Additional</i>	<i>Оперативная кнопка</i>
<i>TSpinButton</i>	<i>Samples</i>	<i>Спаренные кнопки</i>
<i>TRadioButton</i>	<i>Standard</i>	<i>Радиокнопка (круглая, зависимая)</i>
<i>TRadioGroup</i>	<i>Standard</i>	<i>Группа радиокнопок</i>
<i>TCheckBox</i>	<i>Standard</i>	<i>Кнопка с независимой фиксацией (квадратное окошко)</i>

Кнопку будь-якого названого типу можна помістити на форму з палітри компонентів. Багато кнопок мають однакові властивості. Текст на поверхні кнопки визначається властивістю *Caption* типу *AnsiString*

Якщо у складі тексту є амперсанд ('&'), то наступний за ним символ використовується в акселераторі: натиснення комбінації клавіш *Alt+<символ>* викликає 'натиснення' кнопки. Наприклад, якщо на кнопці розміщений текст *S&top*, то натиснення клавіш *Alt+t* викличе 'натиснення' кнопки.

Кнопки розрізняються тим, що саме означає 'натиснення кнопки'. Ряд кнопок (*TRadioButton*, *TRadioGroup*, *TCheckBox*) призначений для установки або перемикання фіксованих параметрів.

Кнопки *TButton*, *TBitBtn*, *TSpinButton*. як правило, своїм натисненням ініціюють негайні дії. Кнопка *TSpeedButton* може служити для вирішення обох задач.

Основною подією, пов'язаною з натисненням кнопки, є: *property OnClick: TNotifyEvent*; наприклад, від клацання на кнопці лівою клавішею миші. Подія *OnClick* може генеруватися для кнопки при натисненні кнопки і при натисненні клавіші *Esc*. Це звичайно використовується для кнопок, пов'язаних з відміною якої-небудь дії, наприклад, від кнопки *Cancel* в діалоговому вікні Властивість *Cancel* типа *Boolean* визначає можливість кнопки реагувати на натиснення клавіші *Esc*. Якщо встановлене *Cancel = True*, натиснення *Esc* діє так само, як якби була натиснута сама кнопка.

Використовування кнопок приведені практично у всіх подальших прикладах.

Button - компонент звичної кнопки Windows

Компонент Button має 33 опубліковані властивості. Список основних властивостей кнопки даний в табл. 4.

Таблиця 4 - Основні властивості звичної кнопки

Cancel	Возможность кнопки реагировать на нажатие клавиши Esc
Caption	Текст на поверхности кнопки
Constraints	Ограничения допустимых изменений размеров компонента
Enabled	Доступность компонента (Boolean)
Font	Шрифт для представления текста
Height	Высота компонента
Hint	Текст оперативной подсказки
Left	Координата X – левого верхнего угла компонента на контейнере
Name	Имя компонента
PopupMenu	Имя всплывающего меню для компонента
ShowHint	Показывать ли оперативную подсказку (Boolean)
Top	Ордината Y – левого верхнего угла компонента на контейнере
Visible	Видимость компонента (Boolean)
Width	Ширина компонента

Можлива реакція кнопки на події:

- 1) від миші: OnClick, OnMouseDown, OnMouseMove, OnMouseUp;
- 2) від клавіатури: Enter, Esc, KeyDown, KeyPress, KeyUp; подвійне клацання на кнопці не передбачене;
- 3) від перетягування компоненту: OnDragDrop, OnDragOver, EndDrag.

BitBtn - компонент графічної кнопки

Ця кнопка - нащадок TButton. Її основні властивості ідентичні приведеним в табл. 1. Вона схожа на звичайну кнопку, але дозволяє розмістити на кнопці разом з текстом піктограму - маленьку картинку. Піктограма звичайно пояснює функцію (призначення) кнопки. Borland Delphi надає стандартний набір видів кнопок з піктограмами у властивості Kind типу TBitBtnKind, значення властивості Kind може бути обране серед значень:

{bkCustom, bkOk, bkHelp, bkYes, bkNo, bkClose, bkAbort, bkRetry, bkAll}.

Деякі види кнопок при натисненні відразу виконують певні дії. Наприклад, кнопка виду `bkClose` закриває форму, якої вона належить, викликаючи її метод `Close`.

За допомогою властивості `Glyph` можна замінити картинку на кнопці або задати декілька (не більш 4-х) картинок для показу зміни картинки при натисненні кнопки. Властивість `Glyph` має тип `TBitmap`.

Властивість `Glyph` комплексна, і при клацанні мишею на його значенні у вікні Інспектора об'єктів з'являється діалогове вікно `Picture Editor` для створення піктограм. Її можна завантажити з тих, що є; для цього треба в редакторі малюнка натиснути кнопку `Load`. З'явиться діалогове вікно для завантаження файлу: малюнком (з розширенням `.bmp`): можна вибрати для цього диск, каталог та файл.

Файл зображення для кнопки може містити до чотирьох зображень піктограм розміру 16x16. В більшості випадків використовують 2 піктограми. Кількість піктограм визначається властивістю `NumGlyphs`, яка з'являється після завантаження піктограм.

Піктограму для кнопки можна створити і зберегти за допомогою редактора який викликається командою меню `ToolsMImage Editor`.

Кнопка `TBitBtn` може реагувати на ті ж події, що і кнопка `TButton`.

SpeedButton - компонент оперативної кнопки

На цій кнопці можна розмістити текст за допомогою властивості `Caption` і картинку за допомогою властивості `Glyph` (аналогічного цій властивості для кнопок типу `BitBtn`). Звичайно ці кнопки використовуються як швидкі - кнопок панелі інструментів.

Кнопка типу `TSpeedButton` може реагувати на ті ж події, що і кнопка типу `TButton`.

SpinButton - компонент спарених кнопок

Цей компонент є парою кнопок з двома вертикальними протилежно направленими стрілками. Вони призначені для збільшення або зменшення якої-небудь величини натисненням однієї з кнопок. Компонент не має заголовка. Малюнки на кнопках є трикутниками, вказуючими напрям вгору і вниз.

Кнопка не має події OnClick. При натисненні нижньої або верхньої кнопок відповідно виникають події:

// - при натисненні на верхню кнопку:

property OnDownClick: TNotifyEvent;

// - при натисненні на нижню кнопку:

property OnUpClick: TNotifyEvent;

Для реакції на ці події можна розробити методи, що змінюють значення заданої змінної.

CheckBox - компонент кнопки з незалежною фіксацією

Кнопка з незалежною фіксацією або прапорець - стандартний елемент управління Windows. Вона є маленьким білим квадратиком, усередині якого може бути галочка або порожньо.

Стан кнопки відображає властивість State, яка має тип TCheckBoxState та може приймати значення:

(cbUnchecked, cbChecked, cbGrayed).

Чи відмічена кнопка, показує властивість Checked типу boolean.

Ця властивість доступна для запису, тобто її значення можна встановлювати з програми. Значення цієї властивості означає:

true - кнопка відмічена (є галочка),

false - кнопка не відмічена (немає галочки).

Кнопка має ті ж властивості, що і TButton, і може реагувати на ті ж події, що і кнопка TButton.

Radio Button - компонент радіо кнопки

Компоненти-радіокнопки, згруповані разом, зручні для демонстрації варіантів для вибору, доступних користувачу. У групі об'єктів-радіокнопок тільки одна кнопка з

групи може бути вибрана (як вибирається одна програма радіоприймача, вибір кольору об'єкту). Особливістю радіокнопок є механізм їх перемикання: при виборі однієї з них вся решта кнопок однієї групи стає невибраними.

Чи вибрана дана радіокнопка, визначає властивість `Checked` типу `boolean`.

Зміна цієї властивості з `false` на `true` виробляється при виборі радіокнопки клацанням на невибраній радіокнопці; при цьому генерується подія `OnClick`. Радіокнопка може реагувати на всі ті ж події, що і `TButton`.

У радіокнопки є властивість `Caption`, що містить пов'язаний з нею текст підказку про її призначення - варіант вибору.

Компоненти групування та панелі. Використання статусного рядка

Актуальність і основна ціль теми:

- ознайомити з основними властивостями та методами компонентів групування та панелей;
- ознайомити з основними властивостями та методами компонентів-діалогів.

Студент повинен знати:

- основні властивості та методи компонентів-панелей;
- основні властивості та методами компонентів-діалогів.

Студент повинен уміти:

- використати основні властивості та методами компонентів-панелей для розробки програм;
- використати основні властивості та методи компонентів-діалогів для розробки програм.

Основні питання:

9. Загальні характеристики панелей.
10. Основні властивості панелей.
11. Основні події компоненту.
12. Використання у програмах.
13. Призначення статусного рядка.
14. Основні властивості компоненту StatusBar.

Питання для самоперевірки і контролю:

1. Які компоненти-панелі ви знаєте?
2. Дати загальну характеристику панелей.
3. Які основні властивості панелей?

4. Основні події компоненту.
5. Як використовуються компоненти-панелі у програмах?
6. Для чого призначений компонент StatusBar?
7. Які властивості компоненту StatusBar?

Рекомендована література:

1. Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.
2. Архангельский, А. Я. Delphi 7. Справочное пособие [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2004. - 1024 с.

Групуючі компоненти і панелі

Загальні характеристики

Панелі є контейнерами, призначеними для об'єднання інших компонентів. Панель є власником компонентів, розташованих на ній. Вона може виконувати як декоративну функцію, об'єднуючи компоненти, пов'язані між собою, так і функцію управління. Наприклад, видимість або доступність панелі визначає видимість або доступність компонентів, розташованих на ній.

У табл. 1 наведено список деяких панелей і групуючих компонентів.

Таблиця 1. Групуючі компоненти і панелі, їх призначення

Пикто- грамма	Компонент	Страница	Назначение
	Panel	Standard	Контейнер для размещения органов управления и других контейнеров
	Splitter	Additional	Разделитель панелей. Для создания панелей с изменяемыми пользователем размерами
	RadioGroup	Standard	Контейнер для группы радиокнопок
	GroupBox	Standard	Контейнер для группы связанных органов управления (например, CheckBox, RadioButton)
	Bevel	Additional	Рамка выступающая или утопленная
	ScrollBar	Additional	Для создания зон отображения с прокруткой
	ControlBar	Additional	Перестраиваемая панель инструментов – потомок компонента ToolBar
	ToolBar	Win 32	Панель инструментов
	CoolBar	Win 32	Перестраиваемая пользователем панель инструментов
	PageScroller	Win 32	Для прокрутки больших окон, например, инструментальных панелей
	TabControl	Win 32	Страница с закладками
	PageControl	Win 32	Многостраничный блокнот
	Frame	Standard	Панель с возможностью наследования. Контейнер любых компонентов

Панелі загального призначення

Компоненти Panel і Splitter - для зміни розмірів панелі за допомогою компонента Panel можна:

- Компонувати різні елементи інтерфейсу, функціонально пов'язані між собою;
- Організувати інструментальну панель;
- Створити рядок статусу (стану).

Зовнішній вигляд панелі визначається сукупністю властивостей BevelInner (внутрішня, рамка), BevelOuter (зовнішня рамка), BevelWidth (товщина рамки), BorderStyle (стиль бордюру), BorderWidth (ширина бордюру).

На панелі можна групувати кнопки RadioButton і CheckBox.

Кнопки `RadioButton` будуть працювати як одна злагоджена група; так само як і при розміщенні на панелі `RadioGroup`, можна буде «вибрати» тільки одну з радіокнопок, розташованих на панелі.

За допомогою компонента `Splitter` можна створити програму, де користувач зможе змінювати розмір панелі під час виконання програми. Для цього треба:

- 1) розмістити на формі панель `Panel1`; встановити її властивість `Align = alBottom`; панель займе нижню частину форми;
- 2) розмістити на формі панель `Panel2`; встановити її властивість `Align = alLeft`; панель займе ліву частину форми;
- 3) розмістити на формі роздільник `Splitter`;
встановити його властивість `Align = alLeft` (воно ж - за умовчанням); роздільник пригорнеться до правого боку панелі `Panel2`;
- 4) розмістити на формі панель `Panel3`; задати у неї властивість `Align = alClient`.

Після запуску програми можна переміщати роздільник, змінюючи розміри панелей `Panel1` і `Panel2`. Вид форми в процесі виконання подано на рис.1.



Рисунок 1. Вид форми з панелями зі змінюваними розмірами

Властивість `MinSize` компонента `Splitter` встановлює мінімальний розмір (в пікселях) панелей, між якими він встановлений.

Компоненти `RadioGroup`, `GroupBox`

Панель `RadioGroup` призначена для розміщення на ній радіокнопок. Радіокнопки компонента утворюють групу взаємопов'язаних індикаторів, з яких

може бути вибраний тільки один. Вони використовуються для вибору користувачем однієї з кількох взаємовиключних альтернатив.

Панель `RadioGroup` має заголовок - напис у лівій частині верхньої рамки (властивість `Caption`). Розміщення радіокнопок проводиться за допомогою властивості `Items` (типу `TStrings`). Для розміщення радіокнопок і написів поруч з ними треба викликати редактор `String List editor` клацанням на кнопці з трьома крапками в області значення властивості `Items`. У кожному рядку редактора треба розмістити тільки напис для кожної радіокнопки. Радіокнопки є власністю компонента `RadioGroup`. І при його переміщенні переміщуються разом з ним.

Якщо кнопок багато, їх можна розмістити на панелі `RadioGroup` в декілька стовпців. Це визначається властивістю `Columns`. За умовчанням значення `Columns` = 1, і кнопки розміщуються в одному стовпці. Номер вибраної (поміченої точкою) радіокнопки визначається властивістю `ItemIndex`. Нумери радіокнопок починаються з 0. Якщо не вибрана жодна радіокнопка, значення `ItemIndex` = -1. Значення `ItemIndex` можна встановлювати як при розробці програми, так і під час його виконання - клацанням миші на кнопці або програмно.

Допускається розміщення інших компонентів (у тому числі компонентів типу `RadioButton`) всередині рамки компонента `RadioGroup`. Але вони не будуть власністю цього компонента. Вони будуть незалежні від нього.

Панель `GroupBox` призначена для розміщення на ній інших компонентів. Вона має заголовок - властивість `Caption`. Компоненти, розміщені на компоненті `GroupBox`, є його власністю: переміщення компонента відбувається разом з іншими компонентами, розташованими на `GroupBox`.

На `GroupBox` можна групувати кнопки `RadioButton` і `CheckBox`. Всі кнопки `RadioButton` будуть працювати як одна злагоджена група; так само як і при розміщенні на панелі `RadioGroup`, можна буде вибрати тільки одну з радіокнопок, розташованих на панелі. З кнопок `CheckBox` можна помітити необхідний набір кнопок. Позначене кнопки визначає властивість `Checked` (типу `Boolean`).

Смуга стану `StatusBar`

Компонент StatusBar являє собою ряд панелей, які відображають смугу стану в стилі Windows. Зазвичай ця смуга розміщується внизу форми. Приклад смуги стану ви можете побачити на рис. 1.

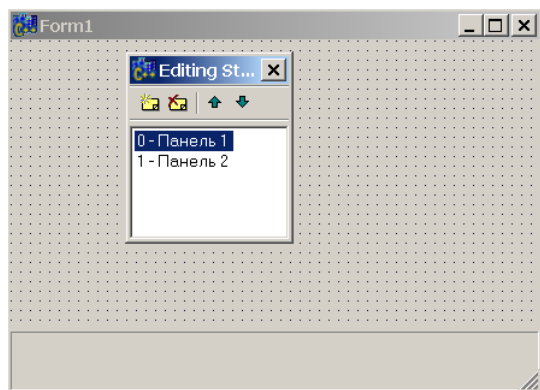


Рис. 1.

Компонент StatusBar розміщений на сторінці Win32 палітри компонентів.

Властивість SimplePanel визначає, чи включає смуга стану одну або безліч панелей. Якщо SimplePanel = true, то вся смуга стану представляє собою єдину панель, текст якої задається властивістю SimpleText. Якщо ж SimplePanel = false, то смуга стану є набором панелей, яка задається властивістю Panels. У цьому випадку властивість SizeGrip визначає, чи може користувач змінювати розміри панелей в процесі виконання додатку.

Кожна панель смуги стану є об'єктом типу TStatusPanels. Властивості панелей ви можете задавати спеціальним редактором наборів.

Роботу з цим інструментом демонструє вікно програми (рис.2).

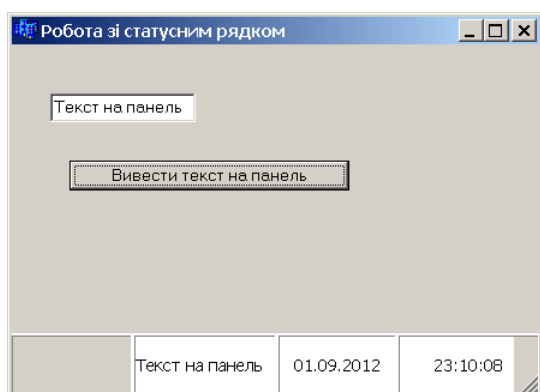


Рис. 2.

Нижче приведено відповідний програмний код.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Form1->StatusBar1->Panels->Items[0] -> Text = "Текст";
    Form1->StatusBar1->Panels->Items[1] -> Text = Edit1.Text;
}
//-----
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    Form1->StatusBar1->Panels->Items[2] -> Text = DateToStr(Now());
    Form1->StatusBar1->Panels->Items[3] -> Text = TimeToStr(Now());
}
```

Викликати редактор можна трьома способами: з Інспектора Об'єктів кнопкою з трьома крапками близько властивості Panels, подвійним клацанням на компоненті StatusBar або з контекстного меню, вибравши команду Panels Editor. У вікні редактора ви можете переміщатися по панелям, додавати нові або знищувати існуючі. При переміщенні по панелям у вікні Інспектора Об'єктів ви будете бачити їх властивості.

Основна властивість кожної панелі - Text, в який заноситься відображаємий в панелі текст . Його можна занести в процесі проектування , а потім можна змінювати програмно під час виконання . Інша істотна властивість панелі - Width (ширина).

Програмний доступ до текстів окремих панелей можна здійснювати через властивість Panels і його індексовані подсвойство Items . Наприклад , оператор :

StatusBar1 -> Panels -> Items [0] -> Text := " текст 1" ; надрукує текст «текст 1» в першій панелі .

Кількість панелей смуги стану можна визначити з властивості Count властивості Panels . Наприклад , наступний оператор очищає тексти всіх панелей:

```
for (i = 0; i < StatusBar1.Panels.Count; i++)
    StatusBar1 -> Panels -> Items [ i ] -> Text = " " ;
```

Підказки та контекстно-залежні довідки

Додаток повинен гранично полегшувати роботу користувача , постачаючи його системою підказок, допомагають зорієнтуватися в додатку .

Тексти ярличків і підказок панелі стану встановлюються для будь-яких візуальних компонентів у властивості Hint у вигляді рядка тексту, що складається з двох частин, розділених символом вертикальної риси '| *'. Перша частина, зазвичай дуже коротка, призначена для відображення в ярличку; друга більш розгорнута підказка призначена для відображення в панелі стану чи іншому заданому місці екрану. Наприклад, в кнопці, відповідній команді відкриття файлу, у властивості Hint може бути заданий текст:

Відкрити | Відкриття текстового файлу

Як окремий випадок, у властивості Hint може бути задана тільки перша частина підказки без символу '| *'.

Для того щоб перша частина підказки з'являлася у спливаючому ярличку, коли користувач затримає курсор миші над даними компонентом, треба зробити наступне:

1. Вказати тексти властивості Hint для всіх компонентів, для яких ви хочете забезпечити ярличок підказки.

2. Встановити властивості ShowHint (показати підказку) цих компонентів в true або встановити в true властивість ParentShowHint (відобразити властивість Show-Hint батька) і встановити в true властивість ShowHint контейнера, утримуючого дані компоненти.

Звичайно, ви можете встановлювати властивості в true або false програмно, включаючи і відключаючи підказки в різних режимах роботи додатку.

При ShowHint, встановленому в true, вікно підказки буде спливати, навіть якщо компонент в даний момент недоступний (властивість Enabled = false).

Якщо ви не задали значення властивості компонента Hint, але встановили в true властивість ShowHint або встановили в true властивість ParentShowHint, а в батьківському компоненті ShowHint = true, то у вікні підказки відображатиметься текст Hint з батьківського компонента.

Правда, все описане вище справедливо при значенні властивості ShowHint додатка Application рівному true (це значення задано за замовчуванням). Якщо

встановити `Application-> ShowHint` в `false`, то вікна підказки не будуть з'являтися незалежно від значень `ShowHint` в будь-яких компонентах.

Властивості `Hint` компонентів можна також використовувати для відображення текстів ув'язнених у них повідомлень в якій: то мітці або панелі за допомогою функцій `GetShortHint` і `GetLongffint`, перша з яких повертає першу частину повідомлення, а друга - другу (якщо другий частини немає , то повертається перша частина). Наприклад, ці функції можна використовувати в обробниках подій `OnMouseMove`, відповідних проходженню курсору миші над даними компонент.

Тема лекції: „Компоненти діалоги. Компоненти меню”

Актуальність і основна мета:

- ознайомити з основними властивостями та методами компонентів-діалогів;
- ознайомити з принципами роботи з компонентами-меню.

Студент повинен знати:

- основні властивості та методи компонентів-діалогів;
- компоненти для роботи з меню;
- принципи роботи з компонентами-меню.

Студент повинен уміти:

- використовувати основні властивості та методи компонентів-діалогів для розробки програм;
- користуватися компонентами – меню у програмах.

Основні питання:

- 15.Огляд основних компонентів-діалогів.
- 16.Основні властивості компонентів-діалогів.
- 17.Основні події компонентів-діалогів.
- 18.Компоненти для роботи з меню.
- 19.Компонент TMainMenu.
- 20.Компонент TPopupMenu.

Питання для самоперевірки і контролю:

- 10.Для чого призначені компоненти-діалоги?
- 11.Як використовуються властивості компонентів-діалогів?
- 12.Які методи компонентів-діалогів вам відомі?
- 13.Привести основні властивості компонентів-меню.
- 14.Як використати компонент TMainMenu?

15.Охарактеризувати основні події для компонентів-меню.

16.Як використати компонент TMainMenu і як TPopupMenu?

Рекомендована література:

Архангельский, А. Я. Программирование в С++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.

Компоненти стандартних діалогів

На сторінці Dialogs палітри компонентів розміщені піктограми компонентів. Вони реалізують стандартні діалоги загального призначення, які використовуються в додатках Windows при відкритті і збереженні файлів, виборі шрифтів, виборі та налаштуванні кольорів і принтерів. Використання цих компонентів полягає в наступному:

1) Налаштування параметрів діалогу; всі діалоги мають для цього властивість Options.

2) Виклик методу Execute, який показує діалогове вікно на екрані і ініціює взаємодію з користувачем; Execute - це функція, вона повертає:

– True, якщо користувач підтвердив введення значення (натиснув кнопку ОК у вікні діалогу або клавішу Enter);

- False, якщо він відмовився від вибору, тобто натиснув кнопку Cancel (або Скасувати) або клавішу Esc.

3) У разі позитивної відповіді - виконання заданих функцій діалогу.

Щоб використовувати стандартний діалог, його компонент треба помістити на форму, встановити необхідні значення його властивостей. Зв'язати виклик діалогу з якою-небудь подією. Найчастіше діалог викликається за допомогою команди меню або натисканням кнопки.

Для виклику будь-якого стандартного діалогу використовується метод Execute-функція, яка повертає логічне значення. При закритті діалогу кнопкою ОК (Відкрити

або Зберегти) Execute повертає значення True, а при скасуванні діалогу кнопкою Скасувати - значення False.

Після закриття стандартного діалогу він повертає через свої властивості значення, обрані або встановлені в процесі діалогу. Наприклад, при відкритті файлу повертається значенням є ім'я файлу (значення властивості OpenFileDialog -> FileName), а при виборі кольору - новий колір (значення властивості ColorDialog -> Color).

Компоненти вибору імені для відкриття і збереження файлу

Діалоги вибору імені файлу використовуються в процесах відкриття і збереження файлу. Компонент OpenFileDialog реалізує діалог відкриття файлу. При запуску цього діалогу з'являється вікно, в якому можна вибрати ім'я файлу. У разі успішного відкриття файлу (після натискання кнопки Відкрити) як результату повертаються - функцією Execute значення True і у властивості OpenFileDialog -> FileName - значення вибраного імені файлу.

Компонент SaveDialog реалізує діалог збереження файлу з ім'ям, заданим користувачем.

Основні властивості компонентів OpenFileDialog і SaveDialog: FileName, Title, InitialDir, DefaultExt, Filter, FilterIndex, Options. 4 перших властивості типу string Filter і Options - складні.

Властивість FileName - складне, воно відкриває діалогове вікно відкриття файлу що дозволяє на етапі розробки програми вибрати ім'я необхідного файлу Властивість FileName - містить ім'я і повний шлях файлу, обраного в вікні діалогу.

Title - задає заголовок вікна діалогу, якщо треба його змінити.

InitialDir - визначає каталог, вміст якого відображається при виклику вікна діалогу. Якщо каталог не вказано, відображається вміст поточного каталогу.

DefaultExt - задає розширення файлу, якщо немає фільтра.

Filter - задає маски імен файлів, які треба відобразити у вікні. За замовчуванням Filter є порожній рядок, що відповідає відображенню файлів усіх типів.

FilterIndex - номер фільтра в списку імен фільтрів, файли для яких відображаються при відкритті діалогу; за замовчуванням FilterIndex = 1.

Options використовується для налаштування близько 20 параметрів, які керують зовнішнім виглядом і функціональними можливостями діалогу.

Фільтр містить значення, розділені символом '('. Кожне значення складається з опису фільтру і маски. Опис - це звичайний текст, що пояснює маску. Маска є шаблоном для відбору файлів і містить ім'я файлу (або *) та розширення. Фільтр звичайно формується при розробці програми. Для цього з Інспектора об'єктів клацанням в області значення властивості Filter викликається редактор фільтра (Filter Editor).

Компоненти OpenFileDialog і SaveFileDialog викликають стандартні діалоги відкриття і збереження графічних файлів. Вони відрізняються від OpenFileDialog і SaveDialog видом вікон і встановленими за замовчуванням значеннями властивості Filter. Їх властивість Filter встановлено за умовчанням для відображення графічних файлів таких форматів: JPG (*. jpg), бітовий масив (*. bmp), піктограма (*. ico), метафайл розширеного формату (*. emf), метафайл (*. wmf) .

Компоненти меню

Для роботи з меню у C++Builder призначені компоненти TMainMenu і TPopupMenu (сторінка Standard палітри компонентів).

Компонент TMainMenu – це головне меню, яке відповідає стандартам Windows і розміщується звичайно під рядком заголовку вікна форми.

Компонент TPopupMenu – це спливаюче (або контекстне, локальне) меню, яке може бути створене майже для всіх компонентів. Воно з'являється при натисканні на праву клавішу миші у момент, коли курсор знаходиться над компонентом.

Конструктор меню

Конструктор меню призначений для наповнення вмістом головного або локального меню форми й викликається тільки тоді, коли відбувається робота із цими компонентами, поміщеними в ту або іншу форму.

Відкриття конструктора меню

Для того щоб викликати конструктор меню, треба, виділивши зображення, що відповідає головному або локальному меню у формі, перейти до сторінки Properties

(властивості) інспектора об'єктів і активізувати в ньому властивість Items (компоненти). У результаті цієї операції відкривається вікно конструктора меню, що містить структуру наявного меню або один порожній прямокутник (якщо створюється нове меню), що відповідає першому створюваному підменю Всі зміни головного меню відразу ж відбиваються на його виді, що з'являється у формі (не слід плутати цей вид із самим елементом, поміщеним у форму з палітри компонентів, вид якого ніяк не міняється).

Додавання нового елемента

При створенні нового меню перший елемент міститься в нього автоматично, варто тільки наповнити його відповідною інформацією.

Коли створюється нове головне меню, в інспекторі об'єктів з'являється інформація, що відповідає новому створюваному елементу меню, у якій важливим у цей час є властивість Caption (назва), куди варто помістити назву елемента, що буде відображатися на екрані. Перед якою-небудь буквою можна поставити символ &, що задає клавішу швидкого переходу (швидкий перехід на цей елемент при натисканні комбінації клавіші Alt і клавіші, що відповідає букві, що йде за символом &). Тут же можна задати й клавіші швидкого керування, що дозволяють виконати певну для елемента команду. Для цього можна скористатися властивістю ShortCut (швидке виконання) інспектора об'єктів, що представляє собою комбінований рядок введення, зі списку, що випадає, якої потрібно вибрати відповідну комбінацію клавіш швидкого керування (або ввести власну комбінацію). Варто мати на увазі, що для назв підменю головного меню клавіші швидкого керування не відображаються, однак виконують свої функції, хоча навряд чи доцільно ними в цьому випадку користуватися - простіше вжити тут клавіші швидкого переходу.

Для локального меню його елементи безпосередньо в конструкторі форми не відображаються. Перший елемент цього меню звичайно є його командою за замовчуванням або назвою підменю наступного рівня. Для елементів локального меню також варто задавати значення властивості Caption, як і для головного меню.

Після завершення роботи з елементом меню (наприклад, у випадку переходу до конструктора меню) елемент одержує уведене ім'я й для головного меню з'являються два інших елементи: один - того ж з (наступне підменю), а іншої - наступного рівня

(перший елемент підміню), а для локального меню - один елемент того ж рівня, розташований нижче першого. З елементами того ж рівня робота відбувається аналогічно описаній. Робота ж з елементами наступного рівня може трохи відрізнятися від описаній вище.

Справа в тому, що кожний елемент цього рівня (а для локального меню й для елементів першого рівня) може виконувати одну із трьох функцій:

- Елемент може бути командою, що викликає тієї або іншої дії програми. Для підключення до такого елемента оброблювача події, що реагує на його активізацію, варто спочатку ввести ім'я команди у властивість `Caption` (заголовок), потім на сторінці `Events` (події) інспектора об'єктів активізувати вміст події `OnClick` (натискання), що приведе до створення заготівлі оброблювача цієї події й переходу до цієї заготівлі. Іншим способом формування заготовки оброблювача події після завдання імені команди є активізація мишею відповідного елемента меню.

- Елемент може бути розділовою лінією. У цьому випадку у властивість `Caption` варто помістити символ - (мінус).

- Елемент може являти собою підменю наступного рівня. Для того щоб елемент став саме таким елементом, варто спочатку задати ім'я підменю у властивості `Caption`, а потім, виділивши цей елемент у конструкторі меню, виконати команду `Create Submenu` (створити підменю) локального меню конструктора. Елемент одержить символ у вигляді трикутника й відкриється перший елемент нового підміню наступного рівня.

Для додавання елемента в будь-якому місці меню необхідно виділити елемент, перед яким (або ліворуч від якого) варто помістити новий елемент, і виконати команду `Insert` (помістити) локального меню конструктора. У результаті цього з'явиться порожній елемент у відповідному місці, з яким далі проводиться робота, як зазначено вище.

При створенні контекстного (локального або спливаючого) меню треба по можливості уникати багаторівневих структур, тому що це меню є оперативним і працювати з декількома рівнями буде незручно.

Варто розрізняти ім'я елемента меню, що буде відображатися при роботі програми і назву елемента в програмі, обумовлену його ідентифікатором. Назва

елемента в програмі збігається з уведенням для нього ім'ям, якщо воно є унікальним і написано латинськими буквами (може бути, з використанням цифр). Якщо є кілька однакових імен, написаних латинськими буквами, то назви одержують додатковий порядковий номер, наприклад Element1, Element2, Element3 (правда, по можливості такої ситуації варто уникати, щоб виключити неоднозначне розуміння окремих елементів). Якщо елемент має ім'я, написане буквами кирилиці, то для його назви в програмі використовується ідентифікатор N з відповідним порядковим номером, наприклад N1, N2, N3.

Для того, щоб задати "спливання" контекстного меню над конкретним компонентом, слід компоненту у властивості PopupMenu призначити назву цього меню.

Для більшої наочності пункти меню можуть мати піктограми. Для їх використання слід розмістити на формі компонент ImageList і додати в нього подвійним кліком і кнопкою Add декілька піктограм. Після цього для компонента-меню (головного або контекстного) призначити властивість Images = ImageList1. Після цього для кожного конкретного пункту меню можна призначити індекс піктограми, що міститься в ImageList1.

Тема лекції: „Компоненти роботи з датою та часом”

Актуальність і основна мета:

- ознайомити з основними властивостями та методами компонентів для роботи з датою та часом;
- ознайомити з принципами роботи з компонентів для роботи з датою та часом.

Студент повинен знати:

- основні властивості та методи компонентів для роботи з датою та часом;
- компонентів для роботи з датою та часом;
- принципи роботи з компонентами.

Студент повинен уміти:

- використовувати основні властивості та методи компонентів для роботи з датою та часом для розробки програм;
- користуватися компонентів для роботи з датою та часом у програмах.

Основні питання:

- 21.Огляд основних компонентів для роботи з датою та часом.
- 22.Основні властивості компонентів для роботи з датою та часом.
- 23.Основні події компонентів для роботи з датою та часом.
- 24.Компонент TMonthCalendar.
- 25.Компонент TDateTimePicker.

Питання для самоперевірки і контролю:

- 17.Для чого призначені компоненти для роботи з датою та часом?
- 18.Як використовуються властивості компонентів для роботи з датою та часом?
- 19.Які методи компонентів для роботи з датою та часом відомі?
- 20.Як використати компонент TMonthCalendar?
- 21.Як використати компонент TDateTimePicker?

Рекомендована література:

Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.

Типи, призначені для роботи з часом.

Для роботи з часом у Borland C++ використовується тип `TDateTime`, (модуль `System`). Змінна цього типу являє собою число з плаваючою точкою, ціла частина якого містить кількість днів, відраховане від деякого початку календаря, а дробна частина дорівнює частині 24-годинного дня, тобто характеризує час і не відноситься до дати. Додавання до значення змінної типу `TDateTime` цілого числа `D` означає збільшення дати на `D` днів. Різниця двох змінних типу `TDateTime` дає різницю двох дат з точністю до частин дня.

Робота з календарем відбувається на основі типу `TDate` (йди його аналога `TDateTime`). Поточна вибрана дата записується у властивість `Date` цього типу. Саме значення зберігається в запакованому форматі в полі типу `Double`.

Компонент Календар - `TMonthCalendar` (Win32). Компонент `TMonthlCalendar` допомагає швидко і легко вибрати потрібну дату і використовується для введення дати з клавіатури. Якщо вибраний діапазон дат, то завершальна дата запишеться у властивість `EndDate`. Для програмної установки дати можна використовувати `MonthlCalendar1.CalendarDate:=Date();` // встановити на календарі поточну дату.

Основні властивості:

- `MaxDate, MinDate` Максимальна дата, мінімальна дата;
- `MultiSelect` Має значення `true`, якщо дозволяється вибирати діапазон дат
- `ShowToday` Має значення `true`, якщо поточна дата додатково відображається у нижній частині календаря.

Компонент Поле введення дати /часу (`TDateTimePicker`) (Win32) є списком, що розкривається, і використовується для введення дати і часу з клавіатури (по

формату відповідно до локальних налагоджень Windows). При розкритті списку відкривається календар (компонент TMonthCalendar).

На етапі проектування настраюються наступні властивості компоненту.

DateFormat Представлення дати в короткому (dfShort) або довгому (dfLong) форматі

DateMode Вибір способу роботи компоненту - як список (dmComboBox) або мати рахівник для змінення дати (dmUpDown).

Kind залежно від значення, компонент використовується для введення дати (dtkDate) або для введення часу (dtkTime).

ShowCheckbox Має значення true, якщо поряд з полем відображається прапорець (його стан можна перевірити, звернувшись до властивості Checked).

Основні властивості:

Date – містить дату;

Time – містить час;

MaxDate - містить максимальну дату;

MinDate – містить мінімальну дату.

TTimer - компонент таймера (System)

Об'єкт-таймер під час роботи програми може ініціювати яку-небудь дію через заданий проміжок часу. Він видає єдину подію OnTimer. Ця подія виникає через кожні 'X' мілісекунд. Величина 'X' задається властивістю Interval. Вимкнути таймер можна, встановивши у властивості Interval негативне або нульове значення. Змінити значення можна і в процесі виконання програми. Максимальне значення інтервалу таймера = 32 767. Компонент інкапсулює системний таймер Windows.

Основні властивості таймера:

property Enabled типу bool; - включає і вимикає таймер; property Interval типу word; - задає інтервал в мілісекундах;

property OnTimer типу TNotifyEvent; - ініціює обробник події.

Література

1. Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.

ВСТУП

Використання псевдоніма для доступу до бази даних забезпечує незалежність програми від розміщення даних в системі, дозволяє розмішати програму роботи з даними і базу даних на різних дисках комп'ютера, в тому числі і на мережевому. Разом з тим для локальних баз даних типовим рішенням є розміщення бази даних в окремому підкаталозі того каталога, в якому знаходиться програма роботи з базою даних. Таким чином, програма роботи з базою даних " знає " , де знаходяться дані . Очевидно, що такий підхід, коли створюється псевдонім до підключення, полегшує адміністрування бази даних .

Розробка застосувань для баз даних. Компоненти доступу до бази даних
Компонент TDataSource.

Компонент DataSource діє як посередник між компонентами TDataSet (TTable , TQuery , TStoredProc) і компонентами Data Controls — елементами управління , що забезпечують представлення даних на формі. Компоненти TDataSet управляють зв'язками з бібліотекою Borland Database Engine (BDE) , а компонент DataSource управляє зв'язками з даними в компонентах Data Controls.

У типових додатках БД компонент DataSource , як правило , пов'язаний з одним компонентом TDataSet (TTable або TQuery) і з одним або більше компонентами Data Controls (такими , як DBGrid , DBEdit та ін.) Зв'язок цього компоненту з компонентами TDataSet і DataControls здійснюється з використанням наступних властивостей і подій.

Властивість DataSet компонента DataSource ідентифікує ім'я компонента TDataSet. Можна привласнити значення властивості DataSet на етапі виконання або за допомогою інспектора об'єктів на етапі проектування .

Властивість Enabled компонента DataSource активізує або зупиняє взаємозв'язок між компонентами TDataSource і Data Controls. Якщо значення властивості Enabled одно true, то компоненти Data Controls, пов'язані з TDataSource, сприймають зміни набору даних.

Використання властивості Enabled дозволяє тимчасово роз'єднувати візуальні компоненти Data Controls і TDataSource, наприклад, для того, щоб у разі пошуку в таблиці з великою кількістю записів не відображати на екрані перегортання всієї таблиці.

Властивість AutoEdit компонента DataSource контролює, як ініціюється редагування в компонентах Data Controls. Якщо значення властивості AutoEdit одно true, то режим редагування починається безпосередньо при отриманні фокусу компонентом Data Controls, зв'язаним з даним компонентом TDataSet. В іншому випадку режим редагування починається, коли викликається метод Edit компонента TDataSet, наприклад, після натиснення користувачем кнопки Edit на компоненті DBNavigator.

Подія OnDataChange компонента DataSource настає, коли відбувається зміна значення поля, запису, таблиці, запиту.

Подія OnUpdateData компонента DataSource настає, коли користувач намагається змінити поточний запис в TDataSet. Оброблювач цієї події слід створювати , коли потрібно дотримати умови посиловної цілісності або обмеження, що накладаються на значення полів змінної бази даних.

Компонент TTable.

Найбільш простим способом звернення до таблиць баз даних є використання компонента TTable , що надає доступ до однієї таблиці.

Для цієї мети найбільше часто використовуються наведені нижче властивості.

Active — вказує , відкрита (true) чи ні (false) дана таблиця .

DatabaseName — ім'я каталогу, що містить шукану таблицю , або псевдонім (alias) видаленої БД (псевдоніми встановлюються за допомогою відповідних утиліт,

наприклад, MeSQLConnector) . Ця властивість може бути змінена тільки в разі , якщо таблиця закрита.

Exclusive — якщо ця властивість приймає значення true , то ніякий інший користувач не може відкрити таблицю , якщо вона відкрита даним додатком . Якщо ця властивість одно false (значення за замовчуванням) , то інші користувачі можуть відкривати цю таблицю .

IndexName — ідентифікує вторинний індекс для таблиці . Цю властивість не можна змінити , поки таблиця відкрита .

MasterFields — визначає ім'я поля для створення зв'язку з іншою таблицею .

MasterSource — ім'я компонента TDataSource , за допомогою якого TTable буде отримувати дані з пов'язаної таблиці .

ReadOnly — якщо ця властивість true , таблиця відкрита в режимі " тільки для читання " . Не можна змінити властивість ReadOnly , поки таблиця відкрита .

Eof , Bof — ці властивості приймають значення true , коли покажчик поточного запису розташований на останній або відповідно першого запису таблиці .

Fields — масив об'єктів TField . Використовуючи цю властивість , можна звертатися до полів по номеру, що зручно, коли заздалегідь невідома структура таблиці.

Найбільш часто при роботі з компонентом TTable використовуються наведені нижче методи.

Open і Close встановлюють значення властивості Active рівними True і False відповідно.

Refresh дозволяє заново вважати набір даних з БД .

MoveBy переміщає покажчик на вказане число рядків (воно може бути і негативним) в межах таблиці

Приклади використання методів наведені в лістингу 9.1.

Лвстинг 9.1 – Приклади використання методів.

// First , Last , Next , Prior переміщують покажчик поточного запису на першу, останню , наступну і попередню запису відповідно.

```
Table1->First();  
while (Table1->Eof!=true)  
{
```

```

//...
Table1->Next();
};
// Insert , Edit , Delete , Append - переводять таблицю в режими
вставки запису, редагування , видалення, додавання запису відповідно.
Post - здійснює фізичне збереження змінених даних . наприклад :
Table2->Insert();
Table2->Fields[0].AsInteger = 100;
Table2->Fields[1].AsString =Edit1->Text;
Table2->Post();
// Cancel - відмінляє внесені зміни , не збережені фізично .
// FieldByName - надає можливість звернення до даних в полях по імені
поля :
S = Table1 -> FieldByName ('area') . AsString ;
// . SetKey перемикає таблицю в режим пошуку .
// . GotoKey починає пошук рядка, значення Fields [ n ] якої одно
//вибраному, де n - номер колонки таблиці , починаючи з 0 :
Table1->SetKey();
Table1->Fields[0].AsString=Edit1.Text;
Table1->GotoKey();
//. SetRangeStart, SetRangeEnd , ApplyRange дозволяють вибрати потрібні
//рядки на основі діапазону значень якого-небудь поля .
Table1->SetRangeStart();
Table1->Fields[0].AsString = Edit1.Text;
Table1->SetRangeEnd();
Table1->Fields[0].AsString = Edit2.Text;
Table1->ApplyRange();

```

FreeBookmark , GetBookmark , GotoBookmark дозволяють створити позначену рядок у таблиці і потім повернутися до неї пізніше . Методи Bookmark використовують клас TBookmark . Метод GetBookmark встановлює закладку на поточній Рядок таблиці . GotoBookmark здійснює переміщення в таблиці до рядка , раніше зазначеної закладкою . Метод FreeBookmark використовується для знищення об'єкту типу TBookmark.

Події компонента TTable дозволяють будувати і контролювати поведінку додатків з БД . Наприклад , подія BeforePost настає перед вставкою або зміною

запису, подія AfterPost - після збереження вставленої або зміненої запису , подія AfterDelete — після видалення запису і т.д. Аналогічно ADOTable.

Властивості DatabaseName присвоїти ім'я каталогу , де знаходиться БД, або псевдонім БД .

Властивості TableName привласнити ім'я таблиці або вибрати таблицю з випадного списку .

Внести до форми компонент DataSource і встановити значення властивості DataSet рівним імені компонента TTable .

Внести компоненти Data Controls (наприклад, DBGrid) і пов'язати їх з компонентом DataSource для того , щоб відобразити на екрані дані з таблиці БД .

Набори даних.

Таблиці БД розташовуються на диску і є фізичними об'єктами. Для операцій з даними, що містяться в таблицях, використовуються набори даних. В термінах системи Borland Delphi набір даних є сукупністю записів, узятих з однієї або декількох таблиць БД. Записи, що входять в набір даних, відбираються за певними правилами, при цьому в окремих випадках набір даних може включати всі записи з пов'язаної з ним таблиці або не містити жодного запису. Набір даних є логічною таблицею, з якою можна працювати при виконанні додатку. Взаємодія таблиці і набору даних нагадує взаємодія фізичного файлу і файлової змінної.

Багато СУБД замість терміну набір даних використовують терміни вибірка або таблиця.

Базові можливості доступу до БД забезпечує клас TDataSet, що представляє набори даних у вигляді сукупності рядків і стовпців (записів і полів). Цей клас надає основні засоби навігації і редагування даних.

Компоненти Table і Query є похідними від класу TDBDataSet. Вони демонструють схожі з базовими класами характеристики і поведінку, але кожний з них має свої особливості. Велика частина властивостей, методів і подій вивчається на прикладі операцій з наборами даних.

Бажано задавати ім'я БД через псевдонім. Це помітно полегшує перенесення додатку і файлів БД в інші каталоги і на інші комп'ютери, оскільки для забезпечення

працездатності додатку після зміни розташування БД достатньо змінити назву каталогу, на який посилається псевдонім БД.

Для компоненту Table використання властивості DatabaseName є єдиною можливістю задати місцезнаходження таблиць БД. Для компонентів та query додатково можна вказати в запиті SQL шлях доступу до кожної таблиці.

Число записів, складаючих набір даних, визначає властивість RecordCount типа longint. Перебір всіх записів набору даних здійснюється в циклі. Перед початком перебору викликом методу First виконується перехід до першого запису набору даних.

Компоненти відображення даних

Компонент TDBGrid забезпечує табличний спосіб відображення на екрані рядків даних з компонентів TTable або TQuery. Програма може використовувати TDBGrid для відображення, вставки, знищення, редагування даних БД.

Зазвичай DBGrid використовується в поєднанні з DBNavigator, хоча можна використовувати й інші інтерфейсні елементи, включивши до їх обробники подій методи First, Last, Next, Ptor, Insert, Delete, Edit, Append, Post, Cancel компонента TTable.

Зовнішній вигляд таблиці (наприклад, написи в заголовках стовпців) може бути змінений за допомогою редактора властивостей Columns Editor. Для виклику Columns Editor потрібно або вибрати відповідну опцію в контекстному меню компонента DBGrid або клацнути мишею в колонці значень навпаки властивості Columns в інспектора об'єктів, що наведено на рисунку 7.1.

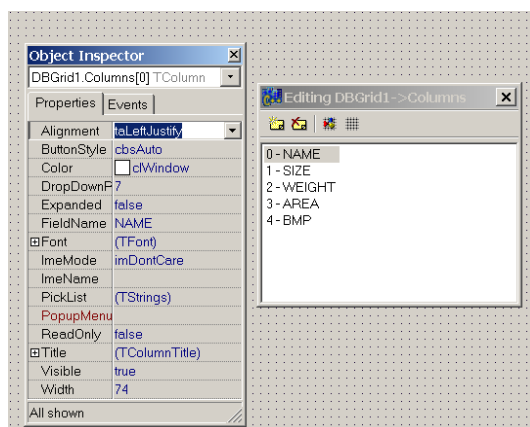


Рис. 7.1 – Установка властивостей стовпців DBGrid за допомогою Columns Editor

Другим способом отримання контролю над характеристиками DBGrid або іншими компонентами є створення описаним вище способом статичного набору компонентів TField. Маючи компонент типу TField, створений для кожного з полів у наборі даних, можна встановити ширину, формат, маску, розташування, мітку для відображення в DBGrid та інші характеристики.

Компонент DBNavigator має ряд кнопок, що служать для керування даними бази даних. Перелічимо їхні назви і призначення, починаючи з лівої кнопки:

- nbFirst → переміщення до першого запису;
- nbPrior → переміщення до попереднього запису;
- nbNext → переміщення до наступного запису;
- nbLast → переміщення до останнього запису;
- nbInsert → уставити новий запис перед поточною;
- nbDelete → видалити поточний запис;
- nbEdit → редагувати поточний запис;
- nbPost → послати відредаговану інформацію в базу даних.

Для виконання операцій з наборами даних використовуються два способи доступу до даних: навігаційний; реляційний.

Навігаційний спосіб доступу полягає в обробці кожного окремого запису набору даних. Цей спосіб звичайно використовується або у видалених БД невеликою розміру. При навігаційному способі доступу кожен набір даних має невидимий покажчик поточного запису. Покажчик визначає запис, з якою можуть виконуватися такі операції, як редагування або видалення. Поля поточному запису доступні для перегляду. Наприклад, компоненти DBEdit і DBText відображають зміст відповідних полів саме поточного запису. Компонент DBGrid указує поточний запис за допомогою спеціального маркера.

Реляційний спосіб доступу заснований на обробці групи записів. Якщо вимагається обробити один запис, все одно обробляється група, що складається з одного запису. При реляційному способі доступу використовуються SQL-запити, тому його називають також SQL-орієнтованим. Реляційний спосіб доступу орієнтований на роботу з видаленими БД і є для них переважним.

Проте його можна використовувати і для локальних БД.

Реляційний спосіб доступу до даних в додатку можна реалізувати з допомогою компоненту Query.

Висновки

Навігаційний спосіб доступу полягає в обробці кожного окремого запису набору даних. Цей спосіб звичайно використовується або у видалених БД невеликою розміру. При навігаційному способі доступу кожен набір даних має невидимий покажчик поточного запису. Покажчик визначає запис, з якою можуть виконуватися такі операції, як редагування або видалення.

Реляційний спосіб доступу заснований на обробці групи записів. Якщо вимагається обробити один запис, все одно обробляється група, що складається з одного запису. При реляційному способі доступу використовуються SQL-запити, тому його називають також SQL-орієнтованим. Реляційний спосіб доступу орієнтований на роботу з видаленими БД і є для них переважним.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Для чого призначені компоненти TDataSource, TADOTable, TADOQuery?
- 2 Які властивості мають компоненти TDataSource, TADOTable, TADOQuery?
- 3 Які властивості мають компоненти TDBGrid, TDBNavigator?
- 4 Як використовуються компоненти TDBGrid, TDBNavigator?

Література

1. Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.

Є візуальні засоби для побудови запитів, наприклад, Visual Query Builder (VQB), що базуються на SQL. За допомогою цього засобу можна будувати комплексні запити, володіючи деякими знаннями SQL або не маючи таких знань зовсім. Запити будуються крок за кроком шляхом послідовного додавання виразів, таблиць, полів і відносин, поки не вийде потрібний результат.

Вибір інформації із бази даних. Фільтрація даних

Компонент ADOTable дозволяє не тільки відображати, редагувати та впорядковувати дані, але і фільтрувати записи за певними критеріями. Користувачеві могло б знадобитися мати можливість переглядати не всю базу даних, а окремо записи з того чи іншого відділу, або, наприклад, переглянути записи співробітників, що мають вік в певному діапазоні.

Фільтрація може задаватися властивостями Filter, Filtered і FilterOptions компонента ADOTable. Властивість Filtered включає або вимикає використання фільтру. А сам фільтр записується у властивість Filter у вигляді рядка, що містить певні обмеження на значення полів. Наприклад, можна задати у властивості Filter Dep:= 'Цех 1 ', встановити властивість Filtered в true, і побачити, що вже в процесі проектування в таблиці відобразяться тільки ті записи, в яких поле Dep має значення 'Цех 1'.

В умовах порівняння рядків можна використовувати символ зірочки "*", який, як у звичайних шаблонах, означає: "будь-яку кількість будь-яких символів".

Опція у властивості FilterOptions — foCaseInsensitive робить порівняння рядків нечутливим до регістру, у якому записано умову фільтру. Якщо включити цю опцію, то слова 'Цех 1' і 'цех 1' будуть вважатися ідентичними.

При записі умов можна використовувати операції відносини =, >, >=, <, <=, <>, А також логічні операції and, or і not. Звичайно, властивості, що визначають фільтрацію, можна задавати не тільки в процесі проектування, але і програмно, під час виконання.

2 Доступ до полів

Поля відображаються об'єктами класу TField і похідних від нього класів TStringField, TSmallintField, TBooleanField і т.п. Ці об'єкти можуть створюватися трьома способами:

- автоматично генеруватися для кожного компонента набору даних (ADOTable та ін);
- створюватися в процесі проектування за допомогою редактора полів
- створюватися програмно в процесі виконання програми.

Автоматична генерація об'єктів класу TField відбувається в момент відкриття бази даних, якщо ці об'єкти не створюються іншими способами: в процесі проектування за допомогою редактора полів або програмно під час виконання. Об'єкти генеруються для всіх полів таблиці (це може бути надмірно для конкретного додатка).

Доступ до об'єктів полів можливий трьома способами:

- по порядковому індексу об'єкта;
- на ім'я поля;
- на ім'я об'єкта.

Доступ за порядковим індексом здійснюється через властивість Fields [int i], де i - індекс об'єкта. Індксація починається з 0. Наприклад, ADOTable1-> Fields [0] - це перший об'єкт поля таблиці ADOTable1.

Доступ на ім'я поля здійснюється за допомогою методу FieldByName ("ім'я "). Наприклад, ADOTable1-> FieldByName ("Fam ") - це об'єкт, пов'язаний з полем Fam.

Значення поля зберігається у властивості Value. Тип цієї властивості - Variant, тобто тип визначається типом поля. Наприклад,

ADOTable1-> FieldByName ("Fam")->Value - це рядок, а

ADOTable1->FieldByName ("Yearjb")->Value - це ціле число.

Є також ряд властивостей, які переводять один тип значень в іншій. Наприклад, властивість AsString переводить тип будь-якого поля в рядок при читанні значення поля, і здійснює зворотний переклад рядка в тип поля при запису значення поля.

Крім AsString є ще аналогічні властивості AsInteger, AsFloat, AsBoolean, AsDateTime.

У деяких випадках вам може знадобитися дізнатися тип даного поля. Ви можете це дізнатися з властивості об'єкта поля DataType, що може приймати значення: ftUnknown (невідоме), ftAutoInc (автоматично наростаюче), ftString (рядок) і т.ін.

Оскільки бази даних призначені не тільки для зберігання, але і для вибору та обробки інформації, одним з найважливіших аспектів їх використання є створення запитів до них.

Запит — це об'єкт, що представляє собою набір даних. Зазвичай для створення запиту використовується компонент TADOQuery — нащадок абстрактного класу TDataSet.

Для виконання операцій з наборами даних використовуються два способи доступу до даних: навігаційний; реляційний.

Навігаційний спосіб доступу полягає в обробці кожного окремого запису набору даних. Цей спосіб звичайно використовується або у видалених БД невеликою розміру. При навігаційному способі доступу кожен набір даних має невидимий покажчик поточного запису. Покажчик визначає запис, з якою можуть виконуватися такі операції, як редагування або видалення. Поля поточному запису доступні для перегляду. Наприклад, компоненти DBEdit і DBText відображають зміст відповідних полів саме поточного запису. Компонент DBGrid указує поточний запис за допомогою спеціального маркера.

У додатку, що розробляється, навігаційний спосіб доступу до даних можна реалізувати, використовуючи будь-який з компонентів ADOTable або ADOQuery.

Реляційний спосіб доступу заснований на обробці групи записів. Якщо вимагається обробити один запис, все одно обробляється група, що складається з одного запису. При реляційному способі доступу використовуються SQL-запити, тому його називають також SQL-орієнтованим. Реляційний спосіб доступу орієнтований на роботу з видаленими БД і є для них переважним.

Проте його можна використовувати і для локальних БД.

Реляційний спосіб доступу до даних в додатку можна реалізувати з допомогою компоненту ADOQuery.

Компонент TADOQuery, як і компонент TADOTable, має всі властивості компонента TDataSet.

Як і у випадку з компонентом TADOTable, компонент TDataSource управляє взаємодією між компонентами Data Controls і компонентом TADOQuery. Зазвичай додаток має один компонент DataSource для кожного компонента TADOQuery.

Найбільш часто використовуються наступні властивості компонента TADOQuery:

Active - вказує, відкритий (true) або закритий (false) даний запит

Eof, Bof - ці властивості приймають значення true, коли покажчик поточного запису розташований на останній або відповідно першому рядку набору даних, що є результатом виконання запиту.

Fields - це властивість доступно лише під час виконання (run-time only) і використовується для читання чи модифікації поля, обумовленого по порядковому номеру.

Params - містить параметри для параметризованого запиту, як SomeNo в наступному прикладі:

Select * from Orders where CustNo =: SomeNo

SQL - рядковий масив, що містить текст оператора запиту SQL.

Компонент TФВЩQuery дозволяє використовувати оператори SQL для того, щоб визначати або створювати набори даних, які можна відобразити на екрані, вставляти, видаляти і редагувати рядки.

Найбільш часто використовуються наступні методи компонента TQuery:

ExecSQL - виконує SQL - запит, що міститься у властивості SQL, якщо запит не повертає дані. Слід вживати цей метод при вставці, редагуванні або видалення даних. При виконанні ж оператора SELECT (вибір даних) слід використовувати метод Open. Наступний приклад, наведений в лістингу 10.2, показує застосування методу ExecSQL.

Лістинг 10.2 – Застосування методу ExecSQL

```
Query1->Close();  
Query1->SQL->Clear();  
Query1->SQL->Add("Delete emp where empno=1010");  
Query1->ExecSQL();
```


Open - відкриває компонент TADOQuery. Він еквівалентний присвоєнню властивості Active значення true. Використовується, якщо результатом запиту є набір даних (такі запити зазвичай починаються з оператора SELECT). Приклад використання методу Open: Query1->Open () ;

Close - закриває компонент TQuery. Виклик Close еквівалентний присвоєнню властивості Active значення false. Приклад використання методу Close: Query1->Close();

Prepare - забезпечує передачу сервера баз даних запиту, що міститься у властивості SQL, для оптимізації та компіляції. Повний запит з параметрами не передається, поки не викликані методи Open або ExecSQL. Навіть якщо метод Prepare не викликається явно, він буде викликаний неявно, якщо використовуються методи Open або ExecSQL (в цьому можна переконатися, запустивши утиліту SQL Monitor).

Компоненти TADOQuery володіють великою різноманітністю методів, успадкованих від TDataSet. Найбільш часто використовуються наступні методи:

First, Last, Next, Prior переміщують покажчик поточного запису на першу, останню, наступну і попередню запису відповідно, наприклад:

MoveBy переміщає покажчик на певну кількість рядків.

Insert, Edit, Delete, Append, Post, Cancel - дозволяють модифіковані результат запиту. Метод Insert дозволяє вносити в результат запиту рядки.

Метод Post підтверджують операції Insert, Update або Delete, здійснюючи реальні фізичні зміни в базі даних. Метод Cancel відміняє незавершені операції Insert, Delete, Edit або Append.

FreeBookmark, GetBookmark, GotoBookmark - - дозволяють створювати закладки (марковані рядки) в запиті і потім повернутися до такої рядку пізніше.

Управління полями полягає у вказівці полів таблиці (таблиць) , які включаються в результуючий набір даних. Як зазначалося вище, при відборі всіх записів таблиці умови відбору записів не вказуються : якщо замість списку полів вказати " * " , то в наборі даних виявляються всі поля записів. При цьому можна не замислюватися про назви полів. Порядок проходження полів у наборі даних відповідає порядку фізичних полів таблиці, визначеному при її створенні.

Приклад відбору всіх полів у таблиці: SELECT * FROM Personnel

В результаті виконання цього запиту з таблиці Personnel в набір даних потрапляють всі поля і всі записи.

Висновки

Оскільки бази даних призначені не тільки для зберігання, але і для вибору та обробки інформації, одним з найважливіших аспектів їх використання є створення запитів до них.

Запит — це об'єкт, що представляє собою набір даних. Зазвичай для створення запиту використовується компонент TADOQuery — нащадок абстрактного класу TDataSet.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 За допомогою яких властивостей може задаватися фільтрація?
- 2 Як можуть створюватися об'єкти TField?
- 3 Що являє собою мова запитів SQL?
- 4 Для чого призначений компонент TADOQuery, які його властивості?

Тема **Запити до баз даних. Об'єкти класу TField. Робота з редактором полів.**

Розробка мережеских застосунків

Мета заняття навчитися використовувати запити до БД, працювати з об'єктами класу TField та редактором полів, розробляти мережескі застосунки

Література

Архангельский, А. Я. Программирование в C++ Builder [Текст] / А. Я. Архангельский. - М. : Бином-Пресс, 2010. - 1230 с.

Об'єкт TField має кілька властивостей, що дозволяють встановити або повернути значення поля, наприклад, AsString, AsBoolean, AsFloat, AsInteger. Найбільш часто використовуються властивості Text (дані для тексту, виведеного в пов'язаний з даним полем інтерфейсний елемент) і FieldName (ім'я поля бази даних).

Мова SQL (Structured Query Language — мова структурованих запитів) дозволяє формувати досить складні запити до баз даних, які повертають запис або безліч записів.

При роботі в мережі, коли до однієї і тієї ж бази даних одночасно може звертатися кілька користувачів, можуть виникати проблеми, пов'язані з одночасною модифікацією одних і тих же записів. Ці проблеми вирішуються за допомогою механізму транзакцій (transaction).

Запити до баз даних. Об'єкти класу TField. Робота з редактором полів

Об'єкти класу TField є властивістю об'єкта TDataSet (деякі властивості об'єктів самі є об'єктами з їх власними наборами властивостей, і TField — один з них).

Властивість Fields об'єкта типу TDataSet дозволяє звертатися до окремих полів набору даних. Властивість Fields є масивом або набором об'єктів TField, динамічно створюються під час виконання додатку. Елементи масиву відповідають колонкам таблиці.

Fields Editor дозволяє створити так званий статичний список полів таблиці, що додаються до опису класу форми. Коли вперше використовуються такі компоненти TDataSet, як компонент TTable або TQuery, список полів для них динамічно генерується в процесі виконання додатку на основі наявних стовпців таблиць або результатів SQL-запиту. Fields Editor дозволяє визначити і потім модифікувати статичний список компонентів Field на етапі проектування додатку. При внесенні колонок з використанням Fields Editor для кожного з полів, доданих до TDataSet, виникають об'єкти TField, після чого можна побачити ці поля в інспекторі об'єктів і використовувати в додатках їх властивості, події та методи.

Використовувати Fields Editor потрібно наступним чином.

- 1 Розмістити компонент TADOTable або TADOQuery на формі.
- 2 Встановити властивість Connection для TADOTable або TADOQuery.
- 3 Встановити властивість TableName компонента TADOTable або властивість SQL компонента TADOQuery.
- 4 Вибрати компонент TDataSet на формі і натиснути праву клавішу миші, після чого з'явиться контекстне меню.
- 5 З контекстного меню вибрати Fields Editor. З'явиться порожнє вікно із заголовком, що збігається з ім'ям компонента TADOTable.
- 6 Знову натиснути праву клавішу миші над порожнім вікном і з контекстного меню вибрати опцію Add Fields. Імена всіх колонок таблиці або запиту з'являться в діалоговій панелі Add Fields, як наведено на рисунках 11.1, 11.2.

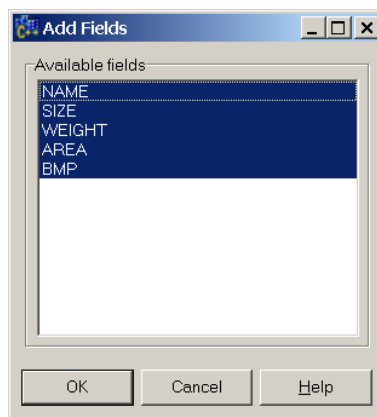


Рис. 11.1 – Формування списку полів

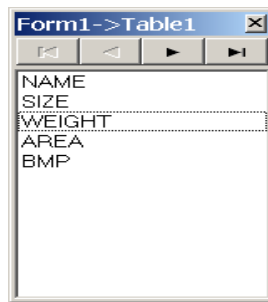


Рис. 11.2 – Сформований список полів, доступних на етапі проектування

7 Якщо потрібно створити обчислюване поле на основі наявних полів, натиснути праву клавішу миші і з контекстного меню вибрати New Field для створення нового поля на основі існуючого або для створення обчислюваного поля (в подальшому слід створити код обробника події OnCalcFields компонента TADOTable, де і проводяться необхідні обчислення).

8 Якщо необхідно видалити статичне поле зі списку полів в наборі даних, потрібно натиснути праву клавішу миші і з контекстного меню вибрати Delete.

Якщо тепер застосувати операцію drag-and-drop до виділених в Fields Editor полях, перенісши їх на форму, то можна отримати готову форму з необхідним набором інтерфейсних елементів. Якщо до такої форми додати компонент TDBNavigator (цей компонент реалізує основні методи TADOTable і TADOQuery, пов'язані з редагуванням даних) і пов'язати його з наявним компонентом TDataSource, а потім скомпілювати проект, отримаємо додаток для перегляду і редагування даних в таблиці, що наведено на рисунку 11.3.

Рис. 11.3 – Результат перенесення на форму полів з Fields Editor

При роботі Fields Editor створюються об'єкти, відповідні видимим в інспектора об'єктів полях. Ці об'єкти є нащадками об'єктного типу TField. Таблиця 11.1 описує існуючі класи таких об'єктів :

Таблиця 11.1 – Нащадки TField

Нащадок	Опис
TStringField	Текстові дані фіксованої довжини до 8192 символів .
TAutoIncField	Цілі числа від -2,147,483,648 до 2,147,483,647 . Призначений для нумерації рядків у наборі даних . Нащадок TIntegerField .
TIntegerField	Цілі числа від -2,147,483,648 до 2,147,483,647 .
TSmallIntField	Цілі числа від -32768 до 32767 .
TWordField	Цілі числа від 0 до 65535 .
TFloatField	Дійсні числа з абсолютною величиною від 1.2×10^{-324} до 1.7×10^{308} з точністю до 15-16 цифри .
TCurrencyField	Дійсні числа з абсолютною величиною від 1.2×10^{-324} до 1.7×10^{308} з точністю до 15-16 цифри .
TBooleanField	Значення true або false .
TDateTimeField	Значення дати й часу .
TDateField	Значення дати .
TTimeField	Значення часу .
TBlobField	Довільне поле даних без обмежень розміру .
TBytesField	Довільне поле даних без обмежень розміру .
TVarBytesField	Довільне поле даних до 65535 символів з фактичною довжиною , представленої в перших двох байтах .
TMemoField	Текст довільної довжини .
TGraphicField	Графічне поле довільної довжини , наприклад , бітовий масив .

Поля відображаються об'єктами класу TField і похідних від нього класів TStringField, TSmallintField, TBooleanField і т.п.

Автоматична генерація об'єктів класу TField відбувається в момент відкриття бази даних, якщо ці об'єкти не створюються іншими способами. Використання редактора полів дуже зручно, оскільки дозволяє задати в процесі проектування безліч корисних властивостей для кожного поля.

У деяких випадках вам може знадобитися дізнатися тип даного поля. Ви можете це дізнатися з властивості об'єкта поля `DataType`, що може приймати значення: `ftUnknown` (невідоме), `ftAutoInc` (автоматично наростаюче), `ftString` (рядок) і т.д.

Розробка мережевих застосувань

Створення додатків для роботи з базами даних в мережі.

Мова SQL (`Structured Query Language` - мова структурованих запитів) дозволяє формувати досить складні запити до баз даних, які повертають запис або безліч записів.

При роботі в мережі, коли до однієї і тієї ж бази даних одночасно може звертатися кілька користувачів, може виникати безліч проблем, пов'язаних з одночасною модифікацією одних і тих же записів. Ці проблеми вирішуються за допомогою механізму транзакцій (`transaction`).

Транзакція — це групова операція, пов'язана з передачею повідомлення. З точки зору програми транзакція - це група логічно пов'язаних операторів SQL, причому тільки успішне виконання всіх операторів повинно приводити до зміни даних сервером. У тих додатках, які ми створювали досі, транзакції теж використовувалися, але неявно. Кожен виклик методу `Post`, кожна зміна таблиці автоматично починається і закінчується транзакцією.

Навіть без спеціального управління транзакціями вони автоматично здійснюються при кожній модифікації даних. Це проводиться компонентом типу `TDatabase`, який включається автоматично в будь-який додаток, що працює з базами даних. Щоб свідомо керувати транзакціями, потрібно явним чином включити компонент `Database` в свій додаток. Він розташований в бібліотеці на сторінці BDE.

`Database` зв'язується з компонентами наборів даних `ADOTable`, `ADOQuery` та іншими через ім'я бази даних, до якої він підключається (властивість `DatabaseName` через псевдонім бази даних або повний шлях до неї). Якщо задається база даних, що має псевдонім BDE, то властивості `Alias-Name`, `DriverName` і `Params` можна не ставити. В іншому випадку треба задати або властивість `AliasName`, або властивості `DriverName` і `Params`.

Компонент TADOQuery, як і компонент TADOTable, має всі властивості компонента TDataSet.

Як і у випадку з компонентом TADOTable, компонент TDataSource управляє взаємодією між компонентами Data Controls і компонентом TADOQuery. Зазвичай додаток має один компонент DataSource для кожного компонента TADOQuery.

Найбільш часто використовуються наступні властивості компонента TADOQuery:

DatabaseName - ім'я каталогу або псевдонім (alias) видаленої БД, до якої здійснюється запит.

DataSource - вказує джерело даних для параметризованих запитів (тобто запитів з параметрами, значення яких заздалегідь невідомо).

Fields - це властивість доступно лише під час виконання (run-time only) і використовується для читання чи модифікації поля, обумовленого по порядковому номеру.

Params - містить параметри для параметризованого запиту.

Висновки

Поля відображаються об'єктами класу TField і похідних від нього класів TStringField, TSmallintField, TBooleanField і т.п. Ці об'єкти можуть створюватися трьома способами:

- автоматично генеруватися для кожного компонента набору даних (Table та in);
- створюватися в процесі проектування за допомогою редактора полів;
- створюватися програмно в процесі виконання програми.

При роботі в мережі, коли до однієї і тієї ж бази даних одночасно може звертатися кілька користувачів, може виникати безліч проблем, пов'язаних з одночасною модифікацією одних і тих же записів. Ці проблеми вирішуються за допомогою механізму транзакцій (transaction).

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1 За допомогою яких властивостей може задаватися фільтрація?

2 Що являє собою мова запитів SQL?

3 Для чого призначений компонент TADOQuery, які його властивості?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1 Для чого призначені об'єкти класу TField?

2 Які властивості мають об'єкти класу TField?

3 Як ведеться робота з редактором полів?

Побудова графіків на основі наборів даних

Для побудови графіків та діаграм в C++ Builder можна використати компонент Chart. Компонент має значну кількість властивостей, методів та подій. Розглянемо основні характеристики.

Компонент Chart є контейнером об'єктів Series типу TChartSeries — серій даних, де кожна серія відповідає одній кривій на графіку. Для деяких видів діаграм можна накласти декілька різних серій одна на одну.

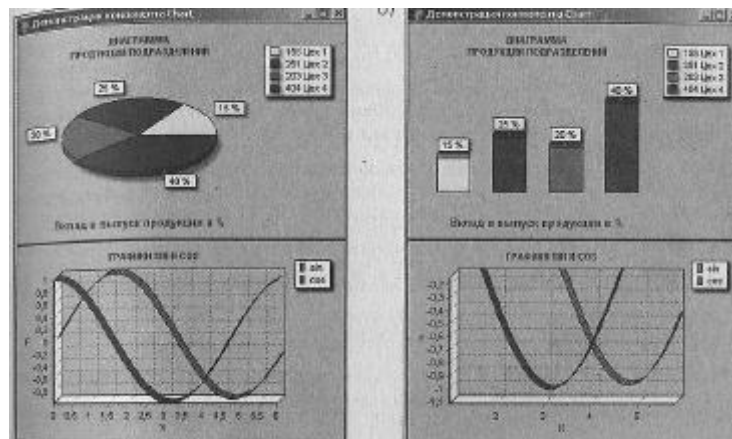


Рис. 1 – Приклади діаграм

Властивість Title визначає заголовок діаграми, Legend — легенду (список визначень).

Властивості графіку наведені в таблиці.

Frame	Визначає рамку навколо діаграми
Legend	Легенда діаграми — список визначень
MarginLeft, MarginRight, MarginTop, MarginBottom	Значення лівого, правого, верхнього і нижнього полів
BottomAxis, LeftAxis, RightAxis	Ці властивості визначають характеристики відповідно нижньої, лівої і правої осі. Задання цих властивостей має сенс для графіків і деяких типів діаграм
LeftWall, BottomWall, BackWall	Ці властивості визначають характеристики відповідно лівої, нижньої і задньої граней області тривимірного відображення графіки
SeriesList	Список серій даних, що відображаються в компоненті
View3D	Дозволяє або забороняє тривимірне відображення діаграми
View3DOptions	Характеристики тривимірного відображення

Chart3DPercent	Масштаб тривимірності (наприклад, товщина діаграми і ширина стрічок графіка)
----------------	--

В Інспекторі об'єктів поруч з батьма властивостями справа є кнопка з трьома крапками, яка дозволяє викликати одну зі сторінок багатосторінкового вікна редактора діаграм, де можна встановити значення властивостей. Редактор діаграм можна також викликати подвійним кліком на компонент Chart, розташований на формі.

Для додавання нової серії в редакторі діаграм на сторінці Series натисніть кнопку Add, виберіть тип діаграми, задайте заголовок. В результаті на формі під час проектування відтворюється діаграма з деякими умовними даними.

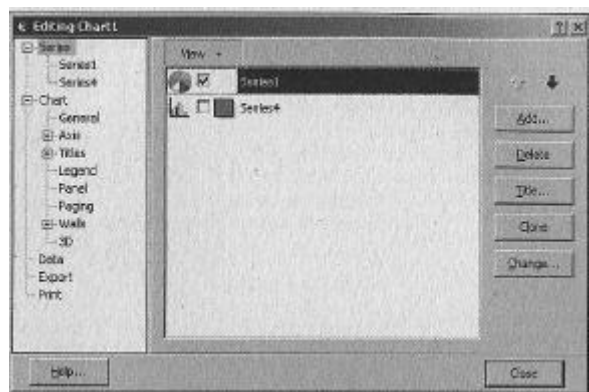


Рис.— Редактор діаграм, сторінка редагування серій

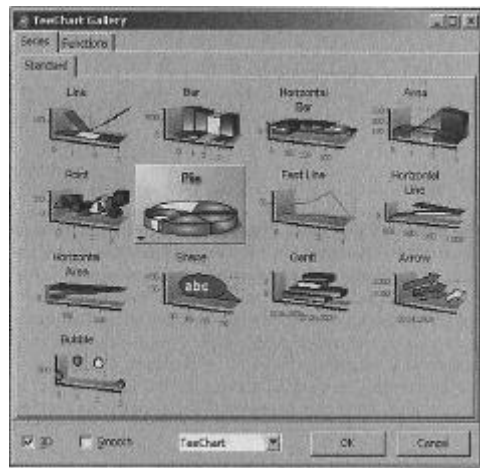


Рис. – Вибір типу діаграми в редакторі діаграм

Під час роботи з редактором діаграм після вибору типу діаграми в компонентах Chart на формі відтворюється її вигляд з занесеними в неї умовними даними. Тому відразу можна спостерігати результат застосування різних опцій в додатку, що є зручним.

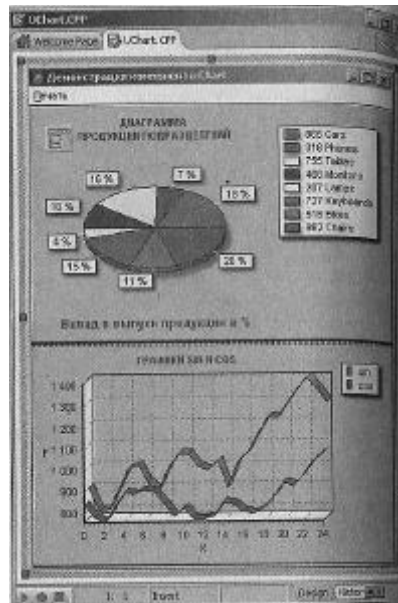


Рис. – Форма додатку з занесеними в неї умовними даними

Після вибору в дереві сторінок в лівій частині вікна ту чи іншу серію (або перейшовши на сторінку Series), можна побачити закладки, які дозволяють вказати додаткові характеристики відтворення серії. Наприклад, для кругової діаграми на вкладці Format корисно ввімкнути опцію Circled Pie, яка забезпечить при будь-якому розмірі компонента Chart відтворення діаграми у вигляді кола. Аналогічна опція в залежності від версії середовища може бути також на вкладці Circled. На вкладці Marks кнопки групи Style визначають, що буде написано на ярликах, які відносяться до певних сегментів діаграми: Value — значення, Percent — відсотки, Label — назви даних та ін. В прикладі включена кнопка Percent, а на вкладці General встановлений шаблон відсотків "##0 %", що забезпечує відображення тільки цілих значень.

При необхідності можна додати на цей компонент Chart ще одну тотожну серію, натиснувши у вікні кнопку Clone, а потім для цієї нової серії натиснути кнопку Change (змінити) і вибрати інший тип діаграми, наприклад, Bar. Очевидно, два різних типи діаграми в одному вікні — погано, але можна вимкнути індикатор цієї нової серії у вікні, а користувачу надати можливість вибрати той чи інший вид відображення діаграми.

Можна задати на графіку дві серії, якщо треба відтворювати одночасно дві криві, і вибрати тип діаграми line. Так як йдеться про графіки, можна використати вкладки

Axis та Walls для того, щоб задати координатні характеристики осей та тривимірних граней графіка.

Після завершення проектування зовнішнього вигляду графіку залишається написати код, який задає дані, які треба відтворити. Для тестового прикладу можна задати в круговій діаграмі деякі константні дані, а в графіках — функції синус та косинус.

Для того, щоб задати відтворюємі значення треба використати методи серій Series. Основні три методи наведені нижче.

Метод Clear очищує серію від внесених попередньо даних.

Метод Add дозволяє додати в діаграму нову точку:

```
long int Add (const double AValue, const String ALabel, TColor AColor);
```

Параметр AValue відповідає значенню, що додається, параметр ALabel — назва, яка буде відтворюватися на діаграмі та в легенді (не обов'язковий, може бути порожнім), AColor — колір.

Метод AddXY дозволяє додати нову точку в графік функції:

```
long int AddXY (const double AXValue, const double AYValue, const String ALabel, TColor AColor);
```

Параметри AXValue та AYValue відповідають аргументу та функції. Параметри ALabel та AColor ті ж, що і в методі Add.

Процедура, що забезпечує завантаження даних в прикладі, може мати вигляд, наведений нижче.

```
int A1=155;
int A2=251;
int A3=203;
int A4=404;
const Pi=3.14159;
Series1->Clear();
Series1->Add(A1,"Цех 1", clYellow);
Series1->Add(A1,"Цех 2", clBlue);
Series1->Add(A1,"Цех 3", clRed);
Series1->Add(A1,"Цех 4", clPurple);
Series2->Clear();
Series3->Clear();
for (int i=0; i<=100; i++) {
```

```

Series2->AddXY(0.02*Pi*i, sin(0.02*Pi*i), "", clRed);
Series3->AddXY(0.02*Pi*i, cos(0.02*Pi*i), "", clBlue);
}

```

Цей програмний код можна включити в обробку кліка кнопки, в команду меню або в подію OnCreate форми. Оператори Clear потрібні, якщо в процесі роботи додатку потрібно оновлювати дані. Без цих операторів повторне виконання методів Add та AddXY тільки додасть точки, не видаливши попередні.

Якщо передбачено, наприклад, для даних, відтворюваних в діаграмі, дві серії, Series1 та Series4 різних видів — Pie та Bar, можна ввести процедуру (наприклад, по натисканню на кнопку, команду меню або клік по компоненту Chart), що змінює за вимогою користувача тип діаграми.

Приклад побудови графіку (двох серій) на основі набору даних:

```

Chart1->Series[0]->Clear ();
Chart1->Series[1]->Clear ();
dMod->ADOTable1->First();
while (!(dMod->ADOTable1->Eof)) {
    Chart1->Series[0]->Add(dMod->ADOTable1->FieldByName("price")-
>AsFloat*1.2, dMod->ADOTable1->FieldByName("id")->AsString+"a", clBlue) ;
    Chart1->Series[1]->Add(dMod->ADOTable1->FieldByName("price")-
>AsFloat, dMod->ADOTable1->FieldByName("id")->AsInteger, clBlue) ;
    dMod->ADOTable1->Next();
}

```

Висновки

Для побудови графіків та діаграм в C++ Builder можна використати компонент Chart. Компонент має значну кількість властивостей, методів та подій.

Компонент Chart є контейнером об'єктів Series типу TChartSeries — серій даних, де кожна серія відповідає одній кривій на графіку.

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
- 1 Для чого призначені об'єкти класу TField?
- 2 Які властивості мають об'єкти класу TField?
- 3 Як ведеться робота з редактором полів?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1 Для чого призначений компонент TChart?

2 Як створити лінійний графік?

3 Як побудувати графік на основі даних логічного набору?

Документування даних в розроблюваних програмах можна виконувати різними способами, один із яких — це використання готових компонентів для побудови звітів. Такі компоненти передбачають вбудовані можливості як для перегляду звіту, так і друку і збереженню в файл.

Створення звітів в C++ Builder за допомогою QuickReport наведено нижче.

Пакет компонентів QuickReport є на офіційному сайті і має відповідати встановленій версії C++ Builder. Після інсталяції можна побачити, що в палітрі компонентів з'явилася вкладинка QReport з переліком компонентів для формування звіту.

QuickReport використовує генератор звітів, що складаються з безлічі смуг. Смуга (band) — це область звіту або розділ, що містить деякий текст, зображення, графіки, діаграми і т.п. Смуги є контейнерами для інших компонентів, які вносять до звіту інформацію. Якщо смуга та розміщені на ній компоненти пов'язані з базою даних, зміст цієї смуги друкується стільки разів, скільки відповідних записів є у джерелі даних.

Таким чином, достатньо розташувати компоненти, пов'язані з даними, на смузі, а значення, що друкуються, і їх кількість будуть автоматично керуватися базою даних. Як у компонентах, пов'язаних з даними, можна задавати головну та допоміжну таблиці, так і між смугами можна задавати аналогічні зв'язки.

Тобто усі можливості, що реалізуються у додатках, реалізуються з тим самим успіхом у звітах.

Розглянемо побудову звіту на основі даних про відділи та співробітників. Першу сторінку такого звіту показано на рис. 1.

КАДРОВЫЙ СОСТАВ ПРЕДПРИЯТИЯ
ПО СОСТОЯНИЮ НА 18.05.2009

ОТДЕЛЫ

Бухгалтерия
Цех 1
Цех 2

СПИСОЧНЫЙ СОСТАВ ОТДЕЛОВ

СОТРУДНИКИ ОТДЕЛА Бухгалтерия
Антонова
Иванова
Петрова

СОТРУДНИКИ ОТДЕЛА Цех 1
Борисов
Иванов
Петров

СОТРУДНИКИ ОТДЕЛА Цех 2
Андреев
Иванов
Сидоров
Хорошее

ЛИЧНЫЕ ДЕЛА

СОТРУДНИК ОТДЕЛА Бухгалтерия

Сотрудник: Антонова, Антонова, Антонова
Пол: Жен.
Дата рождения: 1985
Возраст: 24
Место рождения: Антоний, Антонов, Антонов
1985 г.р.
А.А. Антонова работает по контракту в качестве бухгалтера
И.И. Иванов

Рис. 1 – Перша сторінка звіту

Звіт, озаглавленный "КАДРОВЫЙ СКЛАД ПІДПРИЄМСТВА СТАНОМ НА ..." должен включать наступні розділи:

Заголовок розділу	Інформація, що роздруковується у розділі
ВІДДІЛИ	Список усіх відділів
СПИСКОВИЙ СКЛАД ВІДДІЛІВ	Складається з ряду підрозділів, кожен з яких має свій підзаголовок "СПІВРОБІТНИКИ ВІДДІЛУ ...", після якого слідує список прізвищ співробітників
ОСОБИСТІ СПРАВИ	Складається з низки підрозділів. кожен з яких має свій підзаголовок "СПІВРОБІТНИКИ ВІДДІЛУ ...", після якого слідує інформація про прізвище, ім'я, по батькові співробітника, його стать, рік народження, фотографія співробітника та його характеристика
ВИСНОВКИ	Містить довільний текст, який користувач вводить

У розділі "ОСОБИСТІ СПРАВИ" кожен підрозділ "СПІВРОБІТНИКИ ВІДДІЛУ ..." повинен починатися з нової сторінки. З нової сторінки повинен починатися розділ "ВИСНОВКИ". Усі сторінки повинні мати нижній колонтитул, у якому зліва друкується напис "Кадровий склад".

Під час побудови звіту розглядаються необхідні властивості різних компонентів. Основним компонентом, на якому будується весь звіт, є QuickRep. Він надає ряд можливостей з управління створюваним звітом, включаючи формування заголовка, смуг, шрифтів, установок принтера та ін. Цей компонент є візуальним і після його з'єднання з базою даних може використовуватися як контейнер смуг, що становлять звіт.

Компонент QuickRep має низку властивостей, що визначають характеристики друку звіту, серед яких ReportTitle — заголовок вікна попереднього перегляду. Властивість DataSet визначає набір даних, до якого підключається звіт. Цим набором може бути компонент типу TTable, TQuery тощо.

Компонент QuickRep має два основних методи: Preview — попередній перегляд, і Print — друк.

Компоненти QRLabel, QRMemo, QRRichText, QRShape, QRImage, розміщені на шпальтах звіту, є аналогами звичайних компонентів — Label, Memo, RichEdit, Shape, Image. Основною особливістю відповідних компонентів QuickReport є їхня здатність друкуватись і тих шпальтах звіту, в яких вони розмічені. Деякі компоненти мають властивість AlignToBand — вирівнювання у смузі. Якщо цю властивість встановити в true, компонент буде вирівняний по краю смуги, заданому властивістю Alignment: taLeftJustify — вліво, taCenter — по центру, taRightJustify — вправо.

Для побудови звіту треба перенести на форму компонент QuickRep та передбачити для зв'язку з наборами даних з інформацією про відділи та співробітників, наприклад, компоненти Table (або будь-який інший набір даних), DataSource. Якщо потрібно, можна встановити між двома таблицями звичайний зв'язок, щоб таблиця відділів була головною таблицею, а таблиця співробітників зв'язувалася з нею по полю зовнішнього ключа (властивості IndexName, MasterSource, MasterFields). Відкриємо набори даних.

Тепер розглянемо одну з основних властивостей компонента QuickRep - Bands. Воно має низку підвластивостей:

HasTitle	Є смуга заголовка звіту, яка друкується один раз на початку звіту
HasDetail	Є смуга деталізації, яка друкується стільки разів, скільки записів до неї передається
HasPageHeader	Є верхній колонтитул (заголовок) на кожній сторінці звіту
HasPageFooter	Є нижній колонтитул на кожній сторінці звіту
HasColumnHeader	Є заголовок друкованої таблиці

Для побудови звіту треба встановити true підсвойства HasTitle і HasPageFooter — смуги заголовка і нижнього колонтитулу. На компоненті QuickRep з'являться слабо видимі смуги з відповідними написами. На них і розміщуватимуться компоненти, які відображають ту чи іншу інформацію. Розмістимо на смузі заголовка мітку QRLabel і в її властивості Caption надрукуємо першу частину заголовка звіту "КАДРОВИЙ СКЛАД ПІДПРИЄМСТВА". Вирівняємо цю мітку по центру смуги і нижче помістимо компонент QRSysData. Цей компонент дозволяє відображати у звіті системні дані. Його основна властивість Data, яка може приймати значення, серед яких:

qrsDate	поточна дата
qrsDetailCount	кількість записів
qrsPageNumber	номер поточної сторінки
qrsReportTitle	заголовок звіту

Для того, щоб організувати друк заголовка розділу і далі циклічний друк назв відділів, потрібна смуга деталізації у вигляді компонента QRSubDetail. Перенесемо її на QuickRep і розглянемо деякі її властивості. Властивість DataSet визначає набір даних, якого повинна підключатися смуга. Якщо встановити в true HasHeader, у смузі Group Header, що з'явилася, можна помістити мітку QRLabel з заголовком розділу "Відділи", а в самій смузі деталізації помістити компонент QRDBText — мітку, пов'язану з даними. У ній, потрібно задати набір даних DataSet та його поле DataField, яке має відображатись.

Для друку списку співробітників треба організувати вкладений цикл, щоб для кожного відділу пробігати по записах таблиці співробітників, що належать до нього. Для завдання таких вкладених циклів у компоненті QRSubDetail передбачено властивість Master.

Помістимо до звіту ще один компонент QRSubDetail та зв'яжемо його з таблицею співробітників. Його властивість Master встановимо у QRSubDetail2 — ім'я другої смуги QRSubDetail. На цю нову смугу помістимо мітки QRDBText, налаштувавши їх на поля з прізвищем та ім'ям співробітника.

Для відображення фотографій потрібно використовувати компонент QRDBImage, а для друку характеристик - компонент QRDBRichText.

Для перегляду звіту з головного меню в обробник команди меню "Перегляд" вставимо оператор

```
FRep->QuickRep1->Preview();
```

Для друку звіту з головного меню в обробник команди меню "Друк" вставимо оператор

```
FRep->QuickRep1->Print();
```

На цьому розробка додатка на підготовку звіту закінчена. За його допомогою будь-якої миті можна роздрукувати повний звіт про поточний стан бази даних. При виборі розділу меню Перегляд користувачеві буде відображатися вікно попереднього перегляду. У цьому вікні за допомогою швидких кнопок, розташованих вгорі на інструментальній панелі, користувач зможе змінювати масштаб відображення сторінки (три ліві кнопки), переміщатися по сторінках (кнопки навігатора), здійснювати встановлення принтера і друк, зберігати звіт або відкривати файл якогось з минулих звітів. Кнопка Close закриває вікно попереднього перегляду, після чого можна надрукувати звіт.

SDI та MDI інтерфейси додатків

Керування формами у додатках з інтерфейсом множини документів (додатках MDI). Типовим додатком MDI є звичний для всіх Word.

У додатку MDI є батьківська (первинна) форма та ряд дочірніх форм (називаються також формами документів). Вікна документів можуть створюватись самим користувачем у процесі виконання програми за допомогою команд типу Вікно | Нове. Число дочірніх вікон заздалегідь невідоме — користувач може створити їх стільки, скільки потрібно. Вікна документів розміщуються у клієнтській області батьківської форми. Тому найчастіше доцільно в батьківській формі обмежуватися лише головним меню, інструментальними панелями і, якщо необхідно, панеллю змагання, залишаючи решту місця у вікні для вікон дочірніх форм. При цьому зазвичай вікно батьківської форми у вихідному стані розгортають на весь екран.

З батьківської форми можна керувати дочірніми формами. Дочірню форму не можна знищити, доки знищено батьківську форму. Для створення програми MDI необхідно спроектувати батьківську та дочірню форми. У батьківській формі властивість `FormStyle` встановлюється у `fsMDIForm`, а в дочірній - у `fsMDIChild`. Оскільки дочірні вікна створюватиме сам користувач у процесі виконання програми, дочірню форму необхідно виключити з числа створюваних автоматично. Розглянемо тепер, як можна зробити обробник команди, за яким користувач задає у батьківському вікні створення нового вікна документів - нового екземпляра дочірньої форми.

Розробимо програму з інтерфейсом множини документів — простий багатовіконний додаток.

Основним елементом будь-якої програми є форма — контейнер, в якому розміщуються інші візуальні і невізуальні компоненти. З точки зору користувача форма — це вікно, в якому він працює з додатком. Кожній новій формі, введеній в додаток, відповідає свій модуль (`unit`), що описує цю форму як клас і включає, якщо необхідно, якісь додаткові константи, змінні, функції і процедури.

В додатках MDI дочірні форми не можуть бути автоматично створюваними, так як число таких форм визначає користувач під час роботи програми, створюючи кожну нову форму командою типу «Нове вікно».

Для кожної автоматично створюваної форми додається у файл програми відповідний оператор її створення методом `CreateForm`. Це можна побачити, якщо виконати команду `Project | View Source` і переглянути файл проекту. Він може, наприклад, містити наступні виконувані оператори:

```
Application->CreateForm(__classid(TForm1), &Form1);  
Application->CreateForm(__classid(TForm2), &Form2);  
Application->Run();
```

Перший і другий з них створюють відповідні форми, а третій починає виконання програми.

Для форм, які були виключені зі списку автоматично створюваних, аналогічний метод `CreateForm` треба виконати в той момент, коли форма повинна бути створена.

У момент створення форми виникає подія `OnCreate`. Обробка цієї події широко використовується для налагодження якихось компонентів форми, створення списків і т.ін.

Для створення програми MDI необхідно спроектувати батьківську і дочірню форми. У батьківській формі властивість `FormStyle` встановлюється в `fsMDIForm`, а в дочірній — в `fsMDIChild`. Оскільки дочірні вікна буде створювати сам користувач в процесі виконання програми, дочірню форму необхідно виключити з числа створюваних автоматично за допомогою вікна опцій проекту.

Почнемо розробку інтерфейсу з побудови форми вікна документа.

1. Створіть новий додаток.
2. Назвіть форму, що відкрилася `FDoc`.
3. Помістіть на формі компонент `RichEdit1` типу `TRichEdit`. Його властивість `Align` зробіть рівним `alClient`, щоб вікно редагування зайняло всю площу вікна. Зітріть текст, що з'явився в `RichEdit1` (властивість `Lines`).
4. Змініть властивість форми `FormStyle` на `fsMDIChild`.
5. Збережіть проект, давши спроектованому модулю ім'я, наприклад, `UDoc`.
6. Спроектуємо батьківську форму. Відкрийте у вашому проекті нову форму (`File | New Form`). Назвіть її `FMDI`. Збережіть її модуль з ім'ям, наприклад, `UMDI` (`File | Save As`).
7. Змініть властивість `FormStyle` нової форми на `fsMDIForm`. Можете задати властивість `WindowState` рівним `wsMaximized`, щоб вікно цієї форми пред'являлося

користувачеві в перший момент розгорнутим на весь екран. В include модуля UMDI введіть посилання на модуль дочірньої форми UDoc.

8. Виконайте команду Project | Options і у вікні на сторінці Forms переведіть дочірню форму FDoc зі списку автоматично створюваних в список доступних. При цьому головною стане батьківська форма FMDI — єдина, що залишилася в списку автоматично створюваних форм.

9. Помістіть на формі список зображень ImageList і диспетчер дій ActionList. Пошліться у властивості Images компонента ActionList1 на ImageList1.

10. В редакторі ActionList1 (відкрити подвійним кліком) за натисканням правої клавіші миші введіть дію NewAction, яке буде відповідати створенню нового вікна документа, впишіть власне прізвище. Оброблювач її події OnExecute може мати вигляд:

```
void __fastcall TFMDI::Action1Execute(TObject *Sender) {
    TFDoc *NewF=new TFDoc(Application);
    NewF->Caption="Документ Іванова І.І. "+IntToStr(FMDI->MDIChildCount);
    NewF->Show();
}
```

11. Призначте цей же обробник у події форми OnShow, щоб у перший момент відкривалося одне вікно документа.

12. У диспетчері дій ActionList введіть стандартні дії розділу Window: WindowCascade, WindowTileHorizontal, WindowTileVertical. Напишіть обробники їх подій OnExecute. Для дії «Каскад»: Cascade(); Для дії «Впорядкувати по горизонталі»: TileMode=tbHorizontal; Tile(); Для дії «Впорядкувати по вертикалі»: TileMode=tbVertical; Tile();

Введіть на форму компонент MainMenu. Сформууйте в ньому розділ меню "Вікно" і пошліться в його підрозділах на введені вами дії.

На цьому етап проектування багатовіконного додатку завершений.

Додайте до RichEdit1 контекстне меню, за допомогою якого напишіть завантаження тексту з текстового файлу.

Висновки

Типовим додатком MDI є звичний для всіх Word. З батьківської форми можна керувати дочірніми формами. Для створення програми MDI необхідно спроектувати

батьківську та дочірню форми. Оскільки дочірні вікна створюватиме сам користувач у процесі виконання програми, дочірню форму необхідно виключити з числа створюваних автоматично.

Узагальнення знань і умінь студентів. Обов'язкова контрольна робота

1 Охарактеризувати інструменти середовища розробки Embarcadero C++ Builder – головне меню, інспектор об'єктів та інші.

1 Охарактеризувати файли, які входять до складу проекту у середовищі розробки Embarcadero C++ Builder, які відносяться до форми.

1 Для чого призначена палітра компонентів середовища розробки Embarcadero C++ Builder? Охарактеризувати компоненти введення та відображення тексту.

1 Охарактеризувати компоненти Button та RadioButton палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати основні властивості та методи форми. Чим відрізняються модальна та немодальна форми?

1 Охарактеризувати компоненти Memo та GroupBox палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ListBox та Panel палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Як використовуються файли модулів, які входять до складу проекту або форми у середовищі розробки Embarcadero C++ Builder?

1 Охарактеризувати властивості, методи та події компонентів Memo, Edit.

1 Охарактеризувати компоненти Button та RadioButton палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати основні події та методи форми. Чим відрізняються модальна та немодальна форми?

1 Охарактеризувати властивості, методи та події компонента StringGrid.

1 Для чого призначена палітра компонентів середовища розробки Embarcadero C++ Builder? Охарактеризувати компоненти Edit та Label.

1 Охарактеризувати компоненти CheckBox, UpDown та RadioButton палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ADOTable, ADOQuery. Для чого вони призначені, які основні властивості, методи та події мають?

1 Охарактеризувати компоненти Memo та GroupBox палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ListBox та Panel палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти, призначені для введення та відображення тексту у середовищі розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти, призначені для роботи зі списками рядків.

1 Охарактеризувати компоненти Button та RadioButton палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ADOTable, DataSource, пояснити їх основні властивості.

1 Для чого призначена палітра компонентів середовища розробки Embarcadero C++ Builder? Охарактеризувати компоненти Edit та Label.

1 Охарактеризувати основні властивості та методи форми. Чим відрізняються модальна та немодальна форми?

1 Охарактеризувати компоненти Memo та GroupBox палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ADOTable, ADOQuery. Для чого вони призначені, які основні властивості, методи та події мають?

1 Охарактеризувати основні властивості та методи форми. Чим відрізняються модальна та немодальна форми?

1 Охарактеризувати компоненти Button та RadioButton палітри компонентів середовища розробки Embarcadero C++ Builder.

1 Охарактеризувати компоненти ADOTable, DataSource, пояснити їх основні властивості.

Приклади завдань 2, 3:

2 Прокоментувати наведений фрагмент програмного коду та дати відповіді на такі запитання:

а) Для чого призначений наведений програмний код?

б) Які компоненти використовуються у наведеному фрагменті?

в) Які змінні оголошені у фрагменті та з якими компонентами пов'язані?

```
void __fastcall TForm1::Button1Click(TObject *Sender) {  
    double x, y, r1, r2;  
    x= StrToFloat(Edit1->Text);  
    y= StrToFloat(Edit2->Text);  
    r1= pow(x,2) - x / (2 + pow(x,2) + pow(y,2));  
    r2= sqrt(y)+ (x+2)/(1+pow(y,2));  
    Edit3->Text= FloatToStr(r1);  
    Edit4->Text= FloatToStr(r2);  
}
```

3 Створити програму, яка:

– вводить масив цілих чисел (кількість введених значень елементів масиву може бути від 1 до 12);

– за допомогою радіогрупи дозволяє обрати варіант розрахунку: суму елементів масиву, максимальне значення у масиві;

– результат роботи розміщує у текстовий файл.

Тема Створення власних компонентів. Вибір базового класу для власних компонентів

Література

Культин, Н. Б. Delphi 6. Программирование на Object Pascal [Текст] / Н. Б. Культин. - СПб. : БХВ-Петербург, 2001. - 528 с.

Будь-який компонент є похідним від загального прабатька TComponent, від його більше спеціалізованих спадкоємців (таких як TControl або TGraphicControl) або від існуючого компонентного класу. Компонентом може стати практично будь-який елемент вашої програми, поводженням якого ви хочете управляти на стадії проектування.

Створення власних компонентів. Вибір базового класу для власних компонентів

Процес розробки власного компонента (будемо називати його TMyComponent) проходить через виконання наступних етапів, які перелічені нижче.

- 1 Створення модуля для нового компонента.
- 2 Спадкування похідного класу від існуючого базового компонентного класу.
- 3 Додавання потрібних властивостей, подій і методів.
- 4 Реєстрація компонента в середовищі.
- 5 Налаштування.
- 6 Інсталяція компонента на Палітру.
- 7 Збереження файлів компонента.

Далі буде показано, як виконуються деякі з перерахованих дій. Майстер компонентів здатний виконати автоматично створення файлів модуля, спадкування компонентного класу, оголошення нового конструктора й реєстрацію компонента.

Спадкування компонента.

У таблиці 13.1 приведені рекомендації щодо спадкування компонента.

Таблиця 13.1 – Рекомендації по вибору базового класу в залежності від мети

Мета	Базовий клас
Модифікація існуючого компонента	Будь-який існуючий компонент, наприклад, TListBox, або абстрактний компонентний клас

	TCustomListBox
Створення оригінального компонента	TCustomControl
Створення графічного компонента	TGraphicControl
Створення невидимого компонента	TComponent

Модифікація існуючих компонентів.

Найпростіший спосіб побудувати новий компонент — це почати з існуючого й змінити його властивості. Метою може бути додавання, виключення або заміна значень за замовчуванням деяких властивостей компонента-зразка. Ви можете використати для цієї мети будь-який підходящий абстрактний клас VCL, у назву якого входить слово "Custom".

Наприклад, ви можете зробити новий компонент списку зі спеціальними властивостями, яких немає в стандартному класі TListBox. Оскільки не можна прямо модифікувати TListBox, ви повинні почати з її найближчого попередника в ієрархії класів. Для цієї мети найкраще підходить TCustomListBox, що реалізує всі мислимі властивості похідних компонентів списку, однак не виставляє всіх їх у секції `_published`.

Успадковуючи ваш компонент від одного з абстрактних типів (таких як TCustomListBox), ви всього лише повідомляєте в секції `_published` ті властивості, які хочете включити у ваш компонент, залишаючи інші в секції `protected`.

Створення оригінальних віконних компонентів.

З віконним інтерфейсним елементом, видимим під час роботи програми, користувач звичайно може взаємодіяти. Всі віконні компоненти є похідними від базового класу TWinControl. Стандартний елемент віконного керування характеризує так званий "віконний дескриптор" (window handle), що укладений у властивості `Handle`. Завдяки віконному дескриптору, Windows "довідається" даний компонент, зокрема, що вона може прийняти фокус уведення й передавати віконний дескриптор функціям Windows API для ідентифікації робочого вікна.

Хоча ви можете створити оригінальний інтерфейсний елемент (який не має існуючих аналогів і ніяк не пов'язаний з ними), використовуючи TWinControl як

відправну крапку, середовище надає компоненту TCustomControl саме для цієї мети. TCustomControl — це спеціалізований віконний елемент керування, що спрощує малювання складних візуальних зображень. Якщо вашому компоненту не потрібно приймати фокус уведення, ви можете успадковувати неї від графічного елемента керування, що дає економію системних ресурсів. Усі компоненти стандартного віконного керування: кнопки, списки, поля редагування (за винятком TLabel, що ніколи не приймає фокус уведення) є непрямими похідними TWinControl.

Створення графічних компонентів.

Оригінальні віконні й графічні компоненти дуже подібні. Однак на відміну від похідних TCustomControl, графічні компоненти позбавлені віконного дескриптора й не можуть прийняти фокус уведення. Слід робити нові графічні компоненти від базового абстрактного класу TGraphicControl (який, у свою чергу, є нащадком TControl). TGraphicControl надає вибір канви для малювання й обробляє повідомлення WM_PAINT. Усе, що вам належить зробити, це перевизначити метод малювання Paint у відповідності заданими вимогами.

Створення невидимих компонентів.

Всі компоненти мають загального прабатька абстрактний клас TComponent. Однак, тільки невидимі компоненти можна успадковувати безпосередньо від TComponent.

Будь-який похідний клас від TComponent успадковує всі убудовані в нього властивості й методи. Невидимі компоненти зустрічаються досить рідко й використовуються, головним чином, у якості інтерфейсних елементів з іншими компонентами (доступу до баз даних або як власники діалогових вікон).

Створення модуля компоненту на прикладі середовища Borland Delphi

Програмний модуль складається із файлу MyComp.pas, який компілюється в об'єктний файл із розширенням MyComp.obj. Borland Delphi використовує модулі в різних цілях — кожна форма й більшість компонент мають свій власний модуль. При розробці компоненту ви або створюєте новий модуль для компоненту, або добавляєте його до існуючого модуля.

Щоб створити модуль, виконаєте команду File | New і в діалозі New Items, що відкрився, виберіть значок Unit. Щоб додати компонент до існуючого модулю,

виконайте команду File | Open і в діалозі, що відкрився, відшукайте ваш файл MyComp.pas. Маючи відкритий модуль у вікні редактора коду, ви можете приступитися до розробки компонента. Для початку потрібно перелічити в uses необхідні файли фази предкомпіляції й оголосити похідний клас вашого компонента.

Бажано вибрати для спадкування найбільш близький в ієрархії VCL базовий компонентний клас. Ви не зможете видалити властивості або події з базового компонента, а тільки піднятися вище по ієрархії об'єктів на шляху до загального предка TComponent. Це означає, що вибравши занадто далекого попередника, ви ризикуєте заплутати потенційного користувача (та й самого себе теж) достатком надлишкових властивостей і подій, які по суті не потрібні вашому компоненту. Навпроти, вибравши занадто близького попередника, вам доведеться самотійно й повністю програмувати функціональне поводження свого компонента.

Перед тим як ваш компонент зможе працювати на стадії проектування додатка, Borland Delphi повинен зареєструвати його. При реєстрації стає відомим положення нового компонента в палітрі компонентів Borland Delphi.

Щоб зареєструвати компонент, виконайте наступне.

1 Додайте функцію Register до файлу MyComp.pas, уклавши її ім'я в простір імен (назва простору імен починається із заголовної букви, а всі інші букви прописні).

2 У тілі функції Register оголосіть масив типу TComponentClass, у який уведіть компонент, який реєструєте.

3 У тілі функції Register викличте функцію RegisterComponents з назвою вкладки палітри компонент. Це раведено на рисунку 13.1.

```
procedure Register;  
  
implementation  
  
procedure Register;  
begin  
    RegisterComponents('Samples', [TGraphicClock]);  
end;
```

Рис. 13.1 – Реєстрація компонента

Коли компонент зареєстрований, ви можете випробувати й інстальовати його на палітру, завершуючи тим самим процес розробки нового компонента.

Налагодження неінстальованого компонента.

Вам належить випробувати поведження створеного компонента при виконанні програми до її інсталяція на Палітру. По суті, вам доведеться змодельовати ті дії, які робить Borland Delphi, коли користувач переміщає компонент із Палітри на форму. Цей процес вимагає виконання наступних кроків.

1 Включіть файл модуля `Musomr` вашого компонента в заголовний файл деякої форми.

2 Додайте об'єктний член даних, що представляє випробовуваний компонент, у кінець секції `public` оголошень класу форми, поза областю оголошень, які робить Borland Delphi.

3 Приєднасте оброблювач до події `OnCreate` форми.

4 Викличте конструктор компонентного об'єкта (компонент відповідає за самознищення) з оброблювача цієї події, передавши йому параметр, що вказує на власника компонента.

5 Відразу ж за викликом конструктора встановіть властивість `Parent` батька компонента, що звичайно представляє собою форму, що групує рамку або панель інструментів. Якщо ваш компонент не є елементом керування, тобто ви не успадковували від `TControl`, пропустіть цей крок.

6 Ініціюйте значення інших властивостей компонента.

Компонентні модулі, написані на Об'єктному Паскалі, мають розширення `.pas`.

При інсталяції нового компонента або при виконанні команди `Component | Rebuild Library`, бібліотека візуальних компонентів перебудовується, і Borland Delphi створює тимчасовий файл вихідних текстів `VCL`. Щоб зберегти цей файл, за допомогою команди `Options | Environment | Library` відкрийте діалог опцій і встановіть прапор `Save Library Source Code`.

Щоб додати до `VCL` компонента, виконаєте наступні кроки.

1 За допомогою команди `Component | Install` відкрийте діалогове вікно інсталяції компонент.

2 Натисніть кнопку `Add`, що відкриває діалог додавання модуля. Уведіть ім'я модуля безпосередньо в поле `Module Name` або знайдіть його місце розташування, нажавши на кнопку з `Browse`. Ім'я доданого вами компонентного модуля з'явиться

внизу списку назв групи Installed Components. У списку Component classes ви побачите імена компонентних класів, що вже перебувають в обраній групі бібліотеки. У знову уведеного модуля ім'я компонентного класу відсутнє.

3 Натисніть кнопку ОК, закриваючи діалог інсталяції компоненту. Бібліотека буде перебудована і новий компонент установлений на ту вкладку Палітри, що ви визначили як параметр функції реєстрації.

Щоб видалити компоненту з VCL, виконаєте наступні кроки.

1 Виконайте команду Component | Install, що відкриває діалогове вікно установки компонент.

2 Знайдіть непотрібний вам більше компонентний клас у списку Component classes обраної групи Бібліотеки й натисніть кнопку Remove.

3 Натисніть кнопку ОК. Бібліотека буде перебудована і новий компонент вилучений з Палітри.

Щоб перебудувати палітру, виконаєте наступні кроки.

1 Відкрийте діалог установки опцій Палітри за допомогою команд Component | Configure Palette або Options | Environment | Palette.

2 Натисніть кнопку Add і виберіть ім'я для нової вкладки. Ім'я доданої вами вкладки з'явиться внизу списку Pages назв вкладок.

3 Перетягнете мишею обраний компонент у списку Components на потрібну вкладку списку Pages.

4 Натисніть кнопку ОК. Бібліотека й палітра будуть перебудовані.

Висновки

Найпростіший спосіб побудувати новий компонент — це почати з існуючого й змінити його властивості. Вашою метою може бути додавання, виключення або заміна значень за замовчуванням деяких властивостей компонента-зразка.

Перед тим як ваш компонент зможе працювати на стадії проектування додатка, Borland Delphi повинен зареєструвати його. Після цього можна інсталювати компонент на палітру.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1 Які етапи включає схема розробки компоненту?

2 Які інструменти використовуються для розробки компоненту?

3 Як відбувається реєстрація компоненту?