А.Я. Архангельский

C++Builder 6

СПРАВОЧНОЕ ПОСОБИЕ

Книга 2
Классы и компоненты

Москва
ЗАО «Издательство БИНОМ»
2002

УДК 004.43
ББК 32.973.26-018.1
А87

Архангельский А.Я.

C++Builder 6. Справочное пособие. Книга *2.* Классы и компоненты. — М.: Бином-Пресс, 2002 г. — 528 с.: ил.

В книге даются справочные сведения по многим базовым классам и типам C++Builder. Описания снабжены таблицами, содержащими свыше 2000 кратких характеристик свойств, методов, событий. Дается краткая методика разработки прикладных программ с помощью C++Builder. Помимо кратких характеристик, книга содержит подробные описания около 450 свойств, методов, событий, присущих различным компонентам и классам.

Как справочник книга полезна пользователям любой квалификации: от начинающих до опытных разработчиков.

ISBN 5-9518-0009-9

Содержание

От автора

При написании этой книги я испытывал большие сложности, связанные с жестким ограничением ее размера и безбрежным морем справочного материала, который надо было бы в ней дать. Приходилось отбирать то, что по моим представлениям требуется большинству разработчиков, и у меня нет никакой уверенности в том, что отбор удовлетворит большинство читателей.

Я пока нашел только один выход из неразрешимого конфликта между полнотой материала и объемом книги. Справочные сведения из данной книги плюс немало дополнительного материала включено в справочные файлы [3], сведения о которых вы найдете в разделе «Дополнительные источники информации о C++Builder 6». Там, по крайней мере, нет ограничений на объем материала. Так что можно постоянно пополнять эти справки, что регулярно и делается. Да и стоимость справок заметно в лучшую сторону от стоимости книг. Конечно, справки не могут заменить книгу. Но в них есть и свои преимущества (см. в [3]). Так что я думаю, что справки могут служить хорошим и постоянно развивающимся дополнением к данной книге. По книге, конечно, удобнее изучать особенности того или иного класса и компонента. Но зато справки обеспечивают оперативную помощь в работе, простой способ воспроизведения содержащихся в них примеров и значительно больший объем справочных сведений.

Для читателей, знакомых с моей книгой «Программирование в C++Builder 6» [1], должен отметить, что по сравнению с той книгой справочного материала в данной книге во много раз больше. Но, конечно, здесь отсутствует детальная методика работы с компонентами и создания прикладных программ различного назначения. Так что думаю, что данная книга может служить неплохим дополнением к прежней.

Глава 1

Некоторые базовые классы, типы, переменные, константы

В этой главе представлены сведения о некоторых базовых классах и типах C++Builder 6. В библиотеках C++Builder объявлены тысячи классов и типов, так что рассмотреть все их в рамках данной книги невозможно. Поэтому был проведен достаточно жесткий отбор тех классов, которые используются чаще всего. В каждом классе даны в виде таблиц краткие описания и синтаксис свыше 1100 основных свойств, методов и событий (полные списки вы найдете в [3]). Основными считались те, которые можно использовать при работе с данными классами из приложения. Методы, предназначенные в основном для внутреннего применения в классе и, следовательно, требуемые только при создании новых классов-наследников, в приведенных таблицах, как правило, отсутствуют.

Объявления методов и свойств даны для экономии места в упрощенном виде. Например, пропущены спецификаторы __fastcall, присущие всем методам, спецификаторы **virtual** и др. В свойствах пропущено ключевое слово **property** и не указываются функции чтения и записи. Есть и другие упрощения.

В перечнях свойств, методов, событий, типов вы можете встретить идентификаторы, выделенные подчеркиванием. Например, **TCanvas.** Это означает, что в этой или других главах в соответствующем разделе вы сможете найти развернутое пояснение, комментарий или примеры, связанные с данным термином.

Более полную информацию о всех свойствах, методах, событиях классов и типов, описанных в данной главе, и ряда других вы можете найти в источнике [3].

AnsiString — тип строк

В C++Builder тип строк **AnsiString** реализован как класс, объявленный в файле **vcl/dstring.h** и аналогичный типу длинных строк в Delphi. Это строки с нулевым символом в конце. При объявлении переменные типа **AnsiString** инициализируются пустыми строками.

Для **AnsiString** определены операции отношения ==, !=, >, <, >=, <=. Сравнение производится с учетом регистра. Сравниваются коды символов, начиная с первого, и если очередные символы не одинаковы, строка, содержащая символ с меньшим кодом считается меньше. Если все символы совпали, но одна строка длиннее и в ней имеются еще символы, то она считается больше, чем более короткая.

Для **AnsiString** определены операции присваивания =, += и операция склеивания строк (конкатенации) +. Определена также операция индексации [ ]. Индексы начинаются с 1. Например, если S1 = "Привет", то **Sl[l]** вернет 'П', **Sl[2]** вернет 'р' и т.д.

Тип **AnsiString** используется для ряда свойств компонентов C++Builder. Например, для таких, как свойства **Text** окон редактирования, свойства **Caption** меток и разделов меню и т.д. Этот же тип используется для отображения отдельных строк в списках строк типа **TStrings.** Таким образом, постоянно имея дело с этими свойствами, вы постоянно работаете с **AnsiString.**

Рассмотрим некоторые примеры работы с **AnsiString.** Следующий оператор демонстрирует конкатенацию (склеивание) двух строк:

```
Label1->Caption = Edit1->Text + ' ' + Edit2->Text;
```

В данном случае в свойстве **Label1->Caption** отображается текст, введенный пользователем в окне редактирования **Edit1,** затем записывается символ пробела, а затем —

текст, введенный в окне редактирования **Edit2.** Как видите, склеивание строк типа **AnsiString** легко осуществляется перегруженной операцией сложения "+".

Рассмотрим теперь поиск в строке **S1** фрагмента, заданного строкой S2, и замену его текстом строки S3. Код, осуществляющий эти операции, может иметь вид:

```
AnsiString S1, S2, S3;
// операторы занесения текста в S1, S2, S3
...
int i = S1.Pos(S2);
if (i)
 Label1->Caption = S1.SubString(1,i-1) + S3 +
                   S1.SubString(i+S2.Length(),255);
else Label1->Caption = "Текст не найден";
```

В этом коде использован ряд функций-элементов класса **AnsiString: Pos, Substring, Length.** Обратите внимание на то, что доступ к ним осуществляется операцией точка (.), вместо более привычной в C++Builder операции доступа к методам компонентов стрелка (->). Дело в том, что к методам компонентов доступ осуществляется через указатель на объект, а в данном случае к методам AnsiString доступ осуществляется через сами объекты-строки.

Первый выполняемый оператор приведенного кода использует функцию **Pos.** Эта функция ищет в строке, к которой она применена (в нашем случае в S1), первое вхождение подстроки, заданной ее параметром (в нашем случае S2). Если поиск успешный, функция возвращает индекс первого символа найденного вхождения подстроки. Индексы начинаются с 1. Если подстрока не найдена, возвращается 0.

Следующий оператор с помощью структуры **if...else** проверяет, не равно ли нулю **(false)** возвращенное функцией **Pos** значение. Если не равно, то производится формирование строки с заменой найденной подстроки. Строка формируется склеиванием трех строк: начальной части строки S1, расположенной до найденного вхождения подстроки, строки S3, заменяющей найденное вхождение, и заключительной части строки S1, расположенной после найденного вхождения. Для получения фрагментов строки S1 использована функция **Substring.** Эта функция возвращает подстроку, начинающуюся с символа в позиции, заданной первым параметром функции, и содержащую число символов, не превышающее значение, заданное вторым параметром функции. Таким образом, выражение **Sl.SubString(l, i — 1)** возвращает подстроку строки S1, начинающуюся с первого символа и содержащую i - 1 символов, т.е. часть строки S1, расположенную до найденного вхождения подстроки S2. Аналогично, выражение **Sl.SubString(i + S2.Length(), 255)** возвращает подстроку строки **S1,** расположенную после найденного вхождения подстроки S2. При этом для определения начала этой подстроки использована функция **Length,** возвращающая число символов в строке (в нашем случае — в строке **S2,** содержащей заменяемый фрагмент). В приведенном выражении в качестве второго параметра функции **Substring** задано число 255, которое, как ожидается, превышает длину подстроки. В действительности будет возвращено менее 255 символов, столько, сколько имеется до завершающего S1 нулевого символа.

Если нам надо не отображать измененную строку в виде сообщения, а просто произвести замену фрагмента в исходной строке S1, это еще более упрощает код, который в этом случае сводится всего к двум операторам:

```
int i = S1.Pos(S2);
S1 = S1.SubString(1,i-1) + S3 + S1.SubString(i+S2.Length(),255);
```

Давайте еще более усложним задачу: пусть в строке S1 надо заменить все вхождения S2 на строку S3. Эту задачу можно было бы решить следующим кодом:

```
int iO = 0, i = S1.Pos(S2);
while(i)
{
 S1 = S1.SubString(1,i + iO - 1) + S3 +
```

```
      S1.SubString(i + i0 + S2.Length(),255);
 i0 += i - 1 + S3.Length();
 i = S1.SubString(i0 + 1, 255).Pos(S2);
}
```

Приведенный код мало отличается от рассмотренного ранее и не содержит каких-то новых функций. Основные отличия заключаются в следующем. Во-первых, вводится переменная **i0** — индекс, предшествующий первому символу еще не обработанной части строки **S1.** Значение **i0** изменяется после обработки очередной части строки. Во-вторых, очередное вхождение строки S2 в S1 определяется не по всей строке S1, а только по ее еще не обработанной части: **S1.SubString(i0 + 1, 255).**

Рассмотренную задачу контекстного поиска и замены в строке можно было бы решить иначе, воспользовавшись функциями **Delete и Insert** класса **AnsiString.** Функция **Delete** удаляет из строки, начиная с позиции, заданной первым параметром функции, число символов, заданное вторым параметром функции. Функция **Insert** вставляет в строку подстроку, заданную первым параметром функции, в позицию, заданную вторым параметром функции.

Применение этих функций позволяет выполнить контекстную замену с помощью, например, следующего кода:

```
int iO = 1, i = S1.Pos(S2);
while(i > iO)
{
 S1.Delete(i,S2.Length()); // удаление вхождения S2
 S1.Insert (S3,i);         // вставка S3
 iO = i + S3.Length();
 i = iO - 1 + S1.SubString(i0, 255).Pos(S2);
}
```

В заключение отметим метод, позволяющий переходить от типа AnsiString к типу **(char *).** Несмотря на то, что применение **AnsiString** практически всегда удобнее **(char** *), такие переходы приходится делать при передаче параметров в некоторые функции, требующие тип параметра **(char** *). Чаще всего это связано с вызовом функций API Windows или функций C++Builder, инкапсулирующих такие функции. Преобразование строки **AnsiString** в строку **(char** *) осуществляется функцией **c_str()** без параметров, возвращающей строку с нулевым символом в конце, содержащую текст той строки **AnsiString,** к которой она применена. Например, если вы имеете строки S1 и S2 типа **AnsiString,** которые хотите передать в функцию **Application->MessageBox** в качестве сообщения и заголовка окна, то вызов **Application->MessageBox** может иметь вид:

```
Application->MessageBox(S1.c_str(),S2.c_str(), MB_OK);
```

Возможно и обратное преобразование строки **(char *) в** строку **AnsiString.** Для этого используется функция

```
AnsiString(char *S)
```

которая возвращает строку типа **AnsiString,** содержащую текст, записанной в строке S, являющейся аргументом функции.

Основные методы класса **AnsiString** (в описаниях методов через S1 обозначена строка, метод которой используется):

Метод	Синтаксис / Описание
AnsiCompare	int AnsiCompare(const AnsiString& rhs) const Сравнивает данную строку **S1** с rhs с учетом регистра. Сравнение зависит от текущих установок Windows и может отличаться от сравнения, осуществляемого операциями сравнения. Возвращает значение > 0 при **S1** > rhs, значение < 0 при **S1** < **rhs** и значение 0 при Sl = rhs

Метод	Синтаксис / Описание	
AnsiCompareIC	int **AnsiCompareIC(const AnsiString&** rhs) const Осуществляет сравнение, аналогичное **AnsiCompare**, но без учета регистра	
AnsiLastChar	char* **AnsiLastChar()** const Возвращает указатель на последний значащий символ. Поддерживает многобайтные символы	
AnsiPos	int AnsiPos(const AnsiString& subStr) const Возвращает индекс первого символа первого вхождения subStr в S1. Индексы начинаются с 1. Если subStr не содержится в S1, возвращается 0. В отличие от Pos поддерживает многобайтные символы	
AnsiString	AnsiString(аргумент) Конструктор класса. В зависимости от типа аргумента создает:	
	Аргумент Отсутствует	Создает Пустую строку
	const char* **src**	Строку с нулевым символом в конце из массива символов
	const **AnsiString& src**	Копию AnsiString src
	const char* src, unsigned char **len**	Строку с нулевым символом в конце, являющуюся копией первых len символов из src
	const **wchar_t*** src	Строку с нулевым символом в конце из массива src символов типа **wchar_t**
	int src	Строку с нулевым символом в конце из массива src целых значений символов
	double src	Строку с нулевым символом в конце из массива **src** значений символов с плавающей запятой; преобразуются первые 15 значащих разрядов
c_str	char* c_str()const Возвращает указатель на строку с нулевым символом в конце, содержащую те же символы, что в AnsiString	
CurrToStr	static AnsiString CurrToStr(Currency value) Преобразует значение value типа Currency в строку	
CurrToStrF	static AnsiString **CurrToStrF(Currency** value, TStringFloatFormat format, int digits) Преобразует значение value типа Currency в строку, используя указанный формат преобразования чисел с плавающей запятой (см. разд. «TStringFloatFormat — тип»). Параметр определяет задаваемое число разрядов. Функция соответствует функции CurrToStrF с заданной точностью 19 разрядов	

Метод	Синтаксис / Описание
Delete	void Delete(int index, int count) Удаляет из строки, начиная с позиции index число символов, равное count
FloatToStrF	static AnsiString FloatToStrF(long double value, TStringFloatFormat format, int precision, int digits) Преобразует значение value с плавающей запятой в строку, используя указанный формат (см. разд. «TStringFloatFormat – тип»). Параметры precision и digits задают точность и число разрядов. Точность должна задаваться не более 7 для типа float, не более 15 для double и не более 18 для Extended. Число разрядов зависит от выбранного формата
Format	static AnsiString Format(const AnsiString& format, const TVarRec *args, int size) Формирует строку, используя строку формата format и массив аргументов args
FormatFloat	static AnsiString FormatFloat(const **AnsiString&** format, const long **double&** value) Преобразует значение value с плавающей запятой в строку, используя указанный формат format
Insert	void Insert(const AnsiString& str, int index) Вставляет в строку подстроку **str,** начиная с индекса index
IntToHex	static AnsiString **IntToHex(int** value, int digits) Преобразует значение value в строку, содержащую минимум digits шестнадцатеричных цифр
IsDelimiter	**bool IsDelimiter(const AnsiString&** delimiters, int index) const Возвращает true, если символ с индексом index является одним из разделителей, указанных в строке delimiters. Работает и для многобайтных символов
IsEmpty	bool **IsEmpty()** const Возвращает true, если строка пустая

LastDelimiter	int **LastDelimiter(const** AnsiString& delimiters) const
	Возвращает последний из символов строки, входящих в строку разделителей delimiters. Например, если
	`AnsiString s = "c:\\filename.ext";`
	то
	`s.LastDelimiter("\\.:");`
	вернет 12 (индекс символа точки)

Length	int Length() const Возвращает число символов в строке
LowerCase	AnsiString **LowerCase()** const Возвращает строку, в которой все символы приведены к нижнему регистру. Не влияет на исходную строку

Метод	Синтаксис / Описание
Pos	int Pos(const AnsiString& subStr) const Возвращает индекс первого символа первого вхождения subStr в S1. Индексы начинаются с 1. Если subStr не содержится в S1, возвращается 0. В отличие от AnsiPos не поддерживает многобайтные символы
SetLength	void SetLength(int newLength) Усекает строку до newLength символов. Если исходная строка короче, то она не увеличивается
StringOfChar	static AnsiString StringOfChar(char ch, int count) Возвращает строку, в которой символ ch повторен count раз. Например, `AnsiString s = AnsiString::StringOfChar('A', 10);` задаст строке s значение "AAAAAAAAAA"
Substring	AnsiString SubString(int index, int count) const Возвращает подстроку, начинающуюся с символа в позиции index и содержащую count символов
ToDouble	double ToDouble() const Преобразует строку в число с плавающей запятой. Если строка не соответствует формату числа с плавающей запятой, генерируется исключение EConvertError
ToInt	int ToInt() const Преобразует строку в целое число. Если строка не соответствует формату целого числа, генерируется исключение EConvertError
ToIntDef	int ToIntDef(int defaultValue) const Преобразует строку в целое число. Если строка не соответствует формату целого числа, возвращается значение по умолчанию defaultValue
Trim	AnsiString Trim() const Возвращает строку, соответствующую исходной, но без пробельных символов до и после значащих символов
TrimLeft	AnsiString TrimLeft() const Возвращает строку, соответствующую исходной, но без начальных пробельных символов
TrimRight	AnsiString TrimRight() const Возвращает строку, соответствующую исходной, но без заключительных пробельных символов
Unique	void Unique() Делает строку уникальной, т.е. устанавливает число ссылок на нее (refcnt) в 1. Таким образом, на нее ссылается только один объект
Uppercase	AnsiString UpperCase() const Возвращает строку, в которой все символы приведены к верхнему регистру. Не влияет на исходную строку

Метод	Синтаксис / Описание
WideChar	wchar_t* WideChar(wchar_t* dest, int destSize) const Преобразует строку в массив символов dest типа wchar_t и возвращает указатель на этот массив
WideCharBuf-Size	int WideCharBufSize() const Возвращает размер буфера, требуемого для функции WideChar

Application — переменная

Глобальная переменная типа **TApplication.**

Описание

В каждом приложении автоматически создается объект **Application** типа **TApplication** — приложение. Этот компонент отсутствует в палитре библиотеки, вероятно, только потому, что он всегда один в приложении. **Application** имеет ряд свойств, методов, событий, характеризующих приложение в целом.

Булево свойство **Active** (только для чтения) характеризует активность приложения. Оно равно **true,** если форма приложения находится в фокусе. Если же пользователь переключился на работу с другим приложением, свойство **Active** равно **false.**

Свойство **ExeName** является строкой, содержащей имя выполняемого файла с полным путем к нему. Это свойство удобно использовать, чтобы определить каталог, из которого запущено приложение и который может содержать другие файлы (настройки, документы, базы данных и т.п.), связанные с приложением. Выражение **ExtractFilePath(Application->ExeName)** дает этот каталог. Обычно свойство **ExeName** тождественно функции **ParamStr(0),** возвращающей нулевой параметр командной строки — имя файла с путем.

Свойство **Title** определяет строку, которая появляется около пиктограммы свернутого приложения. Если это свойство не изменяется во время выполнения, то оно равно опции Title, задаваемой во время проектирования на странице Application окна опций проекта (команда Project|Options). Свойство может изменяться программно, например, изменяя надпись в зависимости от режима работы приложения.

Свойство **MainForm** типа **TForm** определяет главную форму приложения. Булево свойство **ShowMainForm** определяет, должна ли главная форма быть видимой в момент запуска приложения на выполнение. По умолчанию оно равно **true,** что обеспечивает видимость главной формы в момент начала работы приложения. Если же установить в головном файле проекта **Application->ShowMainForm** равным **false** до вызова метода **Application->Run()** и если при этом свойство **Visible** главной формы тоже равно **false,** то главная форма в первый момент будет невидимой.

Свойство **HelpFile** указывает файл справки, который используется в приложении в данный момент как файл по умолчанию. Если это свойство не изменяется во время выполнения, то оно равно опции Help File, задаваемой во время проектирования на странице Application окна опций проекта (команда Project | Options). Свойство можно изменять программно, назначая в зависимости от режима работы приложения тот или иной файл справки.

Ряд свойств объекта **Application** определяет ярлычки подсказок компонентов приложения. Свойство **Hint** содержит текст подсказки **Hint** того визуального компонента или раздела меню, над которым в данный момент перемещается курсор мыши. Смена этого свойства происходит в момент события **OnHint,** которое будет рассмотрено позднее. Во время этого события текст подсказки переносится из свойства **Hint** компонента, на который переместился курсор мыши, в свойство **Hint** объекта **Application.** Свойство **Application->Hint** можно использовать для отображения этой подсказки или для установки и отображения в полосе состояния текста, характеризующего текущий режим приложения.

Свойство **HintColor** типа **TColor** определяет цвет фона окна ярлычка. По умолчанию это цвет **clInfoBk,** но его значение можно изменять программно. Свойство **HintPause** определяет задержку появления ярлычка в миллисекундах после переноса курсора мыши на очередной компонент (по умолчанию 500 миллисекунд или половина секунды). Свойство **HintHidePause** аналогичным образом определяет интервал времени, после которого ярлычок становится невидимым (по умолчанию 2500 миллисекунд или две с половиной секунды). Свойство **HintShortPause** определяет аналогичным образом задержку перед появлением нового ярлычка, если в данный момент отображается другой ярлычок (по умолчанию 50 миллисекунд). Это свойство позволяет предотвратить неприятное мерцание, если пользователь быстро перемещает курсор мыши над разными компонентами.

Теперь рассмотрим некоторые методы объекта **Application.** Методы **Initialize** — инициализация проекта, и Run — запуск выполнения приложения, включаются в каждый проект автоматически — вы можете это увидеть в головном файле проекта, если выполните команду Project| View Source. Там же вы можете увидеть применение метода создания форм **CreateForm** для всех автоматически создаваемых форм проекта. Если же *в* вашем проекте есть форма, которая исключена из списка автоматически создаваемых (команда Project | Options и соответствующая установка на странице Forms), то когда эта форма вам потребуется, вы должны будете вызвать этот метод:

```
void CreateForm(System::TMetaClass* InstanceClass,
                void *Reference);
```

где **InstanceClass** — класс создаваемой формы, который указывается операцией **__classid, a Reference** — ссылка на создаваемый объект (его имя). Например:

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

Метод **Terminate** завершает выполнение приложения. Если вам надо завершить приложение из главной формы, то вместо метода **Application->Terminate()** вы можете использовать метод **Close** главной формы. Но если вам надо закрыть приложение из какой-то вторичной формы, например, из диалога, то надо применять метод **Application->Terminate().**

Метод **Minimize** сворачивает приложение, помещая его пиктограмму в полосу задач Windows.

Ряд методов связан с работой со справочными файлами. Выше уже говорилось о свойстве **HelpFile,** указывающем текущий файл справки. Метод **HelpContext:**

```
bool HelpContext(Classes::THelpContext Context);
```

вызывает переход в файл справки на тему с идентификатором **Context.** Это идентификатор, который при проектировании справки поставлен в соответствие некоторой теме. Метод **HelpJump:**

```
bool HelpJump(const System::AnsiString JumpID);
```

выполняет аналогичные действия, но его параметр **JumpID** — не идентификатор темы, а имя соответствующей темы в файле справки, задаваемое в нем сноской *.

Метод **HelpCommand:**

```
bool HelpCommand(int Command, int Data);
```

позволяет выполнить указанную параметром **Command** команду API WinHelp с параметром **Data.** Метод генерирует событие OnHelp активной формы или приложения, а затем выполняет указанную команду WinHelp. Полный список команд WinHelp вы можете найти в теме WinHelp справочного файла **win32.hlp,** расположенного в каталоге ...\Program Files\Common Files\Borland Shared\MSHelp. Приведем только некоторые из них. Команда **HELP_CONTENTS с** параметром 0 отображает окно Содержание справки. Команда **HELP_INDEX** с параметром 0 отображает окно Указатель справки. Команда **HELP_CONTEXT** с параметром, равным идентификатору

темы, отображает тему с заданным идентификатором (это тождественно рассмотренному ранее методу HelpContext). Команда HELP_CONTEXTPOPUP с параметром, равным идентификатору темы, делает то же самое, но отображает тему во всплывающем окне.

Хотелось бы еще обратить внимание читателя на очень полезный метод MessageBox (см. в гл. 5), позволяющий вызывать диалоговое окно с указанным текстом, указанным заголовком и русскими надписями на кнопках (в русифицированных версиях Windows). Это наиболее удачный полностью русифицируемый стандартный диалог.

В классе TApplication определено множество событий, которые очень полезны для организации приложения. Ранее для использования этих событий было необходимо вводить соответствующие обработчики и указывать на них объекту Application специальными операторами. Начиная с C++Builder 5 в библиотеке имеется компонент ApplicationEvents, существенно облегчивший эту задачу. Этот компонент перехватывает события объекта Application и, следовательно, обработчики этих событий теперь можно писать как обработчики событий невизуального компонента ApplicationEvents.

DateSeparator — переменная

Определяет символ разделителя, используемого в формате отображения дат.

Модуль *SysUtils.hpp*.

Определение

```
extern PACKAGE char DateSeparator;
```

Описание

Глобальная переменная **DateSeparator** определяет символ разделителя, используемого в формате отображения дат **ShortDateFormat** и в ряде других форматов. По умолчанию в русифицированных версиях Windows это символ точки. Но вы можете программно задать другой символ разделителя. Правда, если вы не задаете явным образом формат отображения, то само по себе изменение **DateSeparator,** ни к чему не приведет. Но если вы, кроме того, измените значение переменной **ShortDateFormat,** используя в ее описании спецификатор "/", или если примените для отображения даты функции **FormatDateTime, DateTimeToString** и иные, описывающие формат, то в отображении будет использоваться новое значение **DateSeparator.**

Примеры

Оператор

```
Editl->Text = Date O;
```

отобразит в окне **Edit1** текущую дату в виде: "27.06.2002". Если вы предварительно выполните оператор

```
DateSeparator = '-';
```

изменяющий значение **DateSeparator,** то приведенный выше оператор вывода в окно **Editl** отобразит дату в прежнем формате. Но если вы будете отображать дату оператором

```
Editl->Text = FormatDateTime("dd/mm/yyyy",Date());
```

или операторами

```
ShortDateFormat = "dd/mm/yyyy";
Editl->Text = Date();
```

то отображение примет вид: "27-06-2002", т.е. сработает заданный вами разделитель "–".

Default8087CW -- переменная

Содержит значение по умолчанию управляющего слова FPU.

Заголовочный файл *System.hpp*.

Определение

```
extern PACKAGE Word Default8087CW;
```

Описание

Переменная **Default8087CW** содержит значение по умолчанию управляющего слова FPU. Это слово управляет точностью, округлением и генерацией исключений при выполнении операций с плавающей запятой.

Слово может устанавливаться функцией **Set8087CW**. Например:

```
Set8087CW(Default8087CW);
```

Infinity, NegInfinity, NaN — константы

При выполнении арифметических операций и математических функций могут возникать ошибки, связанные с переполнением или с принципиальной невозможностью вычислить результат. Для идентификации подобных ошибок введены три константы: **Infinity** (положительная бесконечность), **NegInfinity** (отрицательная бесконечность), NaN (нецифровое значение).

Они определены в модуле *Math.hpp* следующим образом:

```
static const Extended NaN = 0.0 / 0.0;
static const Extended Infinity = 1.0 / 0.0;
static const Extended NegInfinity = -1.0 / 0.0;
```

Если преобразовывать эти значения в строку обычной функцией **FloatToStr**, то будут выданы соответственно тексты "INF", "-INF", "NAN". Последующее использование полученных значений в арифметических операциях приведет к выдаче в качестве результата аналогичных значений.

Но все это будет так, если функцией **Set8087CW** или другими аналогичными функциями устранена генерация исключений при арифметических ошибках.

Константы **Infinity, NegInfinity** и NaN не могут использоваться в операциях сравнения. Для проверки результата в том же модуле *Math.hpp* введены функции **IsInfinite** и **IsNan**. Первая из них возвращает **true**, если значение аргумента равно **Infinity** или **NegInfinity**, а вторая — если значение аргумента равно NaN. Если функцией **IsInfinite** обнаружено, что значение бесконечное, то знак бесконечности можно определить функцией **Sign**.

LongDateFormat — переменная

Определяет длинный формат отображения дат.

См. разд. «ShortDateFormat, LongDateFormat — переменные».

LongTimeFormat, ShortTimeFormat — переменные

Определяют формат отображения времени.

Модуль *SysUtils.hpp*.

Определение

```
extern PACKAGE AnsiString LongTimeFormat;
extern PACKAGE AnsiString ShortTimeFormat;
```

Описание

Глобальная переменная **LongTimeFormat** определяет формат отображения времени в виде строки такими функциями, как **DateTimeToString**, **TimeToStr** и многими другими. По умолчанию в русифицированных версиях Windows она

обеспечивает отображение времени в формате час (две цифры), минуты (две цифры), секунды (две цифры). В качестве разделителей используется символ, определенный в глобальной переменной **TimeSeparator** (обычно двоеточие). Один из приведенных ниже спецификаторов обеспечивает отображение времени в кратком формате, определенном глобальной переменной **ShortTimeFormat**. По умолчанию это час и минуты (по два символа), разделенные символом **TimeSeparator.**

Задавая значения **LongTimeFormat и ShortTimeFormat** можно изменить отображение времени по умолчанию. В текстах строк могут использоваться следующие спецификаторы:

Спецификатор	Действие спецификатора
h	Отображает час числом без предшествующего нуля: 0-23
hh	Отображает час всегда двузначным числом: 00-23
m или n	Отображает минуты числом без предшествующего нуля: 0-59
mm или nn	Отображает минуты всегда двузначным: 00-59
s	Отображает секунды числом без предшествующего нуля: 0-59
ss	Отображает секунды всегда двузначным числом: 00-59
z	Отображает миллисекунды числом без предшествующего нуля: 0-999
zzz	Отображает миллисекунды всегда трехзначным числом: 000-999
t	Отображает время в формате, соответствующем глобальной переменной ShortTimeFormat ("час:минута")
am/pm	Используется при 12-часовом отображении времени для записи символов "am" или "pm". Регистр символов соответствует регистру, использованному в записи спецификатора
a/p	Используется при 12-часовом отображении времени для записи символов "a" или "p". Регистр символов соответствует регистру, использованному в записи спецификатора
ampm	Используется при 12-часовом отображении времени для записи символов, задаваемых глобальными переменными TimeAMString и TimePMString (в русифицированных Windows обычно пустые)
:	Отображает разделитель времени, заданный глобальной переменной TimeSeparator (обычно":")
'xx'/"xx"	Символы, заключенные в одинарные или двойные кавычки, как и символы, отличные от других спецификаторов, переносятся в результирующую строку без форматирования

Все спецификаторы могут записываться в любом регистре.

Примеры

Оператор

```
Edit1->Text = Time();
```

отобразит в окне **Edit1** время в формате по умолчанию, например: "16:05:02". Если же перед этим оператором вставить изменение переменной **LongTimeFormat:**

```
LongTimeFormat = "время: h:m:s";
```

то то же время отобразится в виде: "время: 16:5:2". А в случае оператора

```
LongTimeFormat = "время: t";
```

отображение будет иметь вид: "время: 16:05"., т.е. будет использована переменная **ShortTimeFormat.** Ее тоже можно изменить. Например, если перед отображением даты вы выполните оператор

```
ShortTimeFormat = "h часов m минут";
```

то приведенный выше оператор, задающий значение **LongTimeFormat,** приведет к отображению текста: "время: 16 часов 5 минут".

Можете записать операторы

```
LongTimeFormat = Edit2->Text;
Editl->Text = Time();
```

Тогда, вводя в окно Edit2 различные тексты, передаваемые в **LongTimeFormat,** вы сможете наблюдать в окне **Edit1** их влияние на отображение времени и подобрать форму, устраивающую вас.

NoErrMsg — переменная

Управляет сообщениями об ошибках.

Модуль *System.*

Определение

```
extern PACKAGE bool NoErrMsg = 0;
```

Описание

Переменная **NoErrMsg** управляет появлением системных сообщений об ошибках времени выполнения. При значении **false,** принятом по умолчанию, в случае ошибки пользователю предъявляется диалоговое окно с соответствующим сообщением. Если установить **NoErrMsg в true,** эти окна появляться не будут.

Модуль *SysUtils* трансформирует большинство ошибок времени выполнения в исключения. Поэтому, если в приложении подключен модуль *SysUtils,* системные сообщения об ошибках времени выполнения могут не появляться, даже если **NoErrMsg = false** (точнее, эти сообщения заменяются стандартной обработкой исключений).

Screen — переменная

Глобальная переменная типа **TScreen.**

Описание

В приложении C++Builder автоматически создается глобальный объект **Screen** (экран) типа **TScreen,** свойства которого определяются из информации Windows о мониторе, на котором запускается приложение. Вы можете в любом приложении использовать, например, такие свойства объекта **Screen,** как **Height** — высота экрана и **Width** — его ширина. Это, в частности, может потребоваться, если вы задаете значение свойства своих форм **Position** таким, что размеры форм остаются постоянными. Так как вы используете в процессе проектирования один тип монитора, а приложение в дальнейшем может работать на мониторе другого типа, то не исключено, например, что ваша форма не поместится на экране или наоборот — будет слишком маленького размера для данного монитора. Чтобы избежать этих неприятностей, можно автоматически масштабировать свою форму, вводя, например, в обработчик ее события **OnCreate** код:

```
Width = Screen->Width / 2;
Height = Screen->Height / 2;
```

Этот код задает размеры формы равными половине соответствующих размеров экрана.

Разрешающую способность экрана можно определить, воспользовавшись свойством **PixelsPerInch,** указывающим количество пикселов экрана на дюйм в вертикальном направлении. Это справедливо именно для вертикального направления, поскольку во многих мониторах коэффициенты масштабирования по горизонтали и вертикали различаются.

Screen имеет свойство **Forms[int Index],** содержащее список форм вашего приложения, отображаемых в данный момент на экране, и свойство **FormCount,** отражающее количество таких форм. Вы можете использовать это свойство, например, для того, чтобы гарантировать, что на данном типе монитора размеры ни одной формы не превысят размеров экрана. Соответствующий код может выглядеть так:

```
for (int i = 0; i < Screen->FormCount; i++)
  {
   if (Screen->Forms[i]->Height > Screen->Height)
     Screen->Forms[i]->Height = Screen->Height - 100;
   if (Screen->Forms[i]->Width > Screen->Width)
     Screen->Forms[i]->Width = Screen->Width - 100;
  }
```

Размеры форм, превышающие размер экрана, урезаются этим кодом с запасом в 100 пикселов.

В приведенных примерах надо, конечно, предусмотреть, чтобы при изменении размеров формы адекватно изменялось и расположение компонентов на ее поверхности.

Еще одно полезное свойство объекта **Screen — Fonts** (шрифты). Это свойство типа **TStrings** содержит список шрифтов, доступных на данном компьютере (свойство только для чтения). Его можно использовать в приложении, чтобы проверять, имеется ли на компьютере тот или иной шрифт, используемый в приложении. Если нет — то можно или дать пользователю соответствующее предупреждение, или сменить шрифт в приложении на один из доступных, или дать пользователю возможность самому выбрать соответствующий шрифт. Например, вы можете поместить в вашем приложении компонент списка **TComboBox** и при событии формы **OnCreate** загрузить его доступными шрифтами с помощью операторов:

```
ComboBox1->Items = Screen->Fonts;
ComboBox1->ItemIndex = 0;
```

Тогда в нужный момент пользователь может выбрать подходящий шрифт из списка, а для того, чтобы этот шрифт использовался, например, для текста в компоненте **RichEditl,** в обработчик события **OnClick** или **OnChange** списка вставьте операторы:

```
RichEdit1->SelAttributes->Name =
              ComboBox1->Items->Strings[ComboBox1->ItemIndex];
RichEditl->SetFocus();
```

Если хотите использовать выбранный шрифт для всех компонентов формы, в которых свойство **ParentFont** установлено в **true,** то приведенный выше оператор должен иметь вид:

```
Font->Name = ComboBox1->Items->Strings[ComboBox1->ItemIndex];
```

Свойство **Cursor** объекта **Screen** определяет вид курсора. Если это свойство равно **crDefault,** то вид курсора при перемещении над компонентами определяется установленными в них свойствами **Cursor.** Но если свойство **Cursor** объекта **Screen** отлично от **crDefault,** то соответствующие свойства компонентов отменяются и курсор имеет глобальный вид, заданный в **Screen.** Этим можно воспользоваться для такой частой задачи, как изменение курсора на форму «песочные часы» во время выполнения каких-то длинных операций. Подобное изменение формы курсора можно оформить следующим образом:

```
Screen->Cursor = crHourGlass;
try
{
// выполнение требуемых длинных операций
}
catch (...)
{
  Screen->Cursor = crDefault; // восстановление курсора
  throw;
}
Screen->Cursor = crDefault;
```

При успешном или аварийном окончании длинных операций курсор в любом случае возвращается в значение по умолчанию.

Если в приложении в какие-то отрезки времени используется отличный от **crDefault** глобальный вид курсора, то приведенный код можно изменить, чтобы по окончании длинных операций восстановить прежнее глобальное значение:

```
TCursor Save_Cursor = Screen->Cursor;
Screen->Cursor = crHourGlass;
try
{
 // выполнение требуемых длинных операций
}
catch (...)
{
  Screen->Cursor = Save_Cursor;
  throw;
}
Screen->Cursor = Save_Cursor;
```

Имеется также свойство **Cursors[I],** которое представляет собой список доступных приложению курсоров. Вы можете создать и использовать также свой собственный курсор. Создается он и включается в ресурс приложения встроенным в C++Builder Редактором Изображений (Image Editor). А регистрируется созданный вами курсор с помощью функции **LoadCursor.** Сделать это можно следующим образом.

Пусть, например, вы создали свой курсор и включили его в ресурс приложения под именем **NEWCURSOR.** Тогда в своем приложении вы можете ввести глобальную константу, обозначающую ваш курсор. Например:

```
const crMyCursor = 1;
```

Значение этой константы может лежать в пределах от -32768 до 32767. Но важно, чтобы она не совпадала с предопределенными значениями стандартных курсоров, лежащими в диапазоне от 0 до -21.

В обработчике события **OnCreate** формы вы можете ввести оператор, регистрирующий ваш курсор в свойстве **Cursors:**

```
Screen->Cursors[crMyCursor]=LoadCursor(HInstance,"NEWCURSOR");
```

Обратите внимание на то, что имя курсора пишется обязательно заглавными буквами, так как именно так хранятся имена курсоров в ресурсах приложения.

В нужный момент вы можете установить этот курсор в качестве глобального оператором

```
Screen->Cursor = crMyCursor;
```

а затем восстановить значение глобального курсора оператором

```
Screen->Cursor = crDefault;
```

Вы можете использовать ваш зарегистрированный курсор и как локальный, например, для панели **Panel1** оператором

```
Panell->Cursor = (TCursor)crMyCursor;
```

Подробнее методика создания и использования собственных курсоров изложена в [1].

С помощью **Screen** можно получить доступ к активной в текущий момент форме вашего приложения через свойство **ActiveForm.** Если в данный момент пользователь переключился с вашего приложения на какое-то другое и, следовательно, ни одна форма вашего приложения не активна, то **ActiveForm** указывает на форму, которая станет активной, когда пользователь вернется к вашему приложению. В момент переключения фокуса с одной вашей формы на другую, генерируется событие **OnActiveFormChange.**

Аналогично с помощью свойства **ActiveControl** можно получить доступ к активному в данный момент оконному компоненту на активной форме. При смене фокуса генерируется событие **OnActiveControlChange.**

В C++Builder предусмотрена возможность разработки мультиэкранных приложений, работающих одновременно с множеством мониторов. При этом приложение может решать, какие формы и диалоги надо отображать на том или ином мониторе. Свойства различных мониторов, используемых в таком приложении, можно найти с помощью свойства **Screen->Monitors[I],** где **I** — индекс монитора. Индекс 0 относится к первичному монитору. Свойство **Screen->Monitors[I]** является списком объектов типа **TMonitor,** содержащих информацию о конкретных мониторах.

Среди свойств объектов типа **TMonitor** имеются **Height** — высота и **Width** — ширина экрана монитора. Кроме того, имеются свойства **Left** и **Top.** Эти свойства означают следующее. Все доступное экранное пространство можно представить себе разбитым на экраны отдельных мониторов, размещающихся слева направо и сверху вниз. Соответственно свойства **Left** и **Top** определяют координаты левого верхнего угла экрана монитора в этом логическом экранном пространстве. Объекты типа **TMonitor** имеют также свойство **MonitorNum** — номер монитора, представляющий собой его индекс в свойстве **Screen->Monitors[I].**

Для управления тем, на каком мониторе должна появляться та или иная форма, служит свойство формы **DefaultMonitor.** Это свойство может принимать значения:

dmDesktop	не предпринимается попыток разместить форму на конкретном мониторе
dmPrimary	форма размещается на первом мониторе в списке **Screen->** Monitors
dmMainForm	форма появляется на том мониторе, на котором размещена главная форма
dmActiveForm	форма появляется на том мониторе, на котором размещена текущая активная форма

Set — шаблон класса

Является шаблоном, реализующим встроенный класс Delphi, используемый в библиотеке компонентов VCL.

Модуль *vcl/sysset.h.*

Определение

```
template<class T, unsigned char minEl, unsigned char maxEl>

class __declspec(delphireturn) Set;
```

Описание

В шаблоне должно быть задано три параметра:

type	тип элементов множества (обычно **int**, char или **enum**)
minval	минимальное значение элемента множества (не менее 0)
maxval	максимальное значение элемента множества (не более 255)

Тип каждого объекта класса Set определяется всеми тремя параметрами. Если какие-то из этих параметров различаются, считается, что это объекты разных типов и их нельзя, например, сравнивать друг с другом. Например, если объявлены объекты

```
Set <char, 'A', 'C'> s1;
Set <char, 'X', 'Z'> s2;
```

то оператор

```
if(s1 == S2) ...
```

будет воспринят как ошибка, поскольку типы **s1** и s2 различны.

Для множества определены следующие операции (в описании операций словами «данное множество» обозначается левый операнд):

Операция	Определение	Описание
–	Set__fastcall operator –(**const** Set& **rhs**) const;	данное множество равно разности двух множеств: данного и **rhs** (операция xor с их элементами)
-=	Set&__fastcall operator -=(const Set& rhs);	создание нового множества, определенного разностью двух множеств: данного и rhs (операция **xor** с их элементами)
*	Set& __fastcall operator *=(**const** Set& **rhs**);	создание нового множества, определенного пересечением двух множеств: данного и **rhs** (операция and с их элементами)
*=	Set__fastcall operator *(**const** Set& rhs) const;	данное множество равно пересечению двух множеств: данного и **rhs** (операция and с их элементами)
+	Set__fastcall operator +(const Set& **rhs**) const;	создание нового множества, определенного объединением двух множеств: данного и **rhs** (операция or с их элементами)
+=	Set&__fastcall operator +=(const Set& **rhs**);	данное множество равно объединению двух множеств: данного и rhs (операция or с их элементами)
<<	Set& __**fastcall** operator <<(**const**T el);	добавление элемента **el** в данное множество
<<	friend ostream& **operator**<<(**ostream&** os, const Set& arg);	поместить множество arg в поток ostream (выводится 0 или 1 для каждого элемента в зависимости от его наличия в множестве)
>>	Set&__fastcall operator >>(**const**T el);	удаление элемента el из данного множества

Операция	Определение	Описание
>>	friend istream& operator >>(istream& is, Set& arg);	извлечь множество **arg** из потока istream (вводится 0 или 1 для каждого элемента в зависимости от его наличия в множестве)
_	Set&__fastcall operator =(const Set& rhs);	присваивание данному множеству содержимого множества **rhs**
~	bool__fastcall operator ==(const Set& rhs) const;	эквивалентность двух множеств: данного и **rhs** (совпадение всех элементов)
!=	bool__fastcall operator !=(const Set& rhs) const ;	неэквивалентность двух множеств: данного и rhs

Все операции можно применять только к множествам одного типа, то есть к таким, при объявлении которых все аргументы объявления **(type, minval и maxval)** совпадают. В операциях, создающих новое множество (операции +, - и *), переменная, в которую заносится результат, также должна быть того же типа, что и операнды. Операция эквивалентности возвращает **true** в случае, когда оба операнда содержат только совпадающие элементы. Соответственно только в этом случае операция неэквивалентности возвращает **false.**

Для множеств Set определены также два метода:

Метод	Определение	Описание
Clear	**Set&__fastcall Clear();**	**очистка множества**
Contains	**bool__fastcall Contains(const T el) const;**	**проверка наличия в множестве элемента el**

Рассмотрим примеры работы с множествами. Пусть вы задаете пользователю в программе некоторый вопрос, подразумевающий ответ типа "Yes/No". Тогда возможные символы, вводимые пользователем в качестве ответа, являются множеством, содержащим символы "y", "Y", "n" и "N". Сформировать такое множество можно операторами:

```
Set <char, 0, 255> TrueKey;
...
TrueKey << 'y' << 'Y' << 'n' << 'N';
```

Тогда проверить, принадлежит ли введенный пользователем символ Key множеству допустимых ответов, можно с помощью метода **Contains:**

```
if (!TrueKey.Contains(Key))
   ShowMessage("Вы ввели ошибочный ответ");
else ...
```

Рассмотрим еще один пример. Пусть вы хотите, чтобы в окне редактирования **Edit1** пользователь мог вводить только число, т.е. только цифры от 0 до 9. Это можно сделать, включив в обработчик события **OnKeyPress** этого окна операторы:

```
Set <char, '0', '9'> Dig;
Dig << '0' << '1' << '2' << '3' << '4' << '5'
    << '6' << '7' << '8' << '9';
if (!Dig.Contains(Key))
   {Key = 0; Beep();}
```

При попытке пользователя ввести символ, отличный от цифры, раздастся звук (его обеспечит функция **Beep)** и символ не появится в окне.

ShortDateFormat, LongDateFormat — переменные

Определяют формат отображения дат.

Модуль *SysUtils.hpp*.

Определения

```
extern PACKAGE AnsiString ShortDateFormat;
extern PACKAGE AnsiString LongDateFormat;
```

Описание

Глобальная переменная **ShortDateFormat** определяет формат отображения дат в виде строки такими функциями, как **DateTimeToString, FormatDateTime, DateToStr, DateTimeToStr** и многими другими. По умолчанию в русифицированных версиях Windows она обеспечивает отображение даты в формате день (две цифры), месяц (две цифры), год (четыре цифры). В качестве разделителей используется символ, определенный в глобальной переменной **DateSeparator** (обычно точка). Один из приведенных ниже спецификаторов обеспечивает отображение даты в длинном формате, определенном глобальной переменной **LongDateFormat.** По умолчанию в русифицированных версиях Windows это день, название месяца, год (четырехзначное число с последующими символами "г."), разделенные пробелами.

Задавая значения **ShortDateFormat и LongDateFormat** можно изменить отображение дат по умолчанию. В текстах строк могут использоваться следующие спецификаторы:

Спецификатор	Действие спецификатора
d	Отображает день числом без предшествующего нуля: 1-31.
dd	Отображает день, причем всегда двузначным числом: 01-31.
ddd	Отображает день недели аббревиатурой, задаваемой глобальной переменной ShortDayNames. Для русифицированных Windows это обычно аббревиатуры: "Пн", "Вт", "Ср", "Чт", "Пт", " Сб", "Вс".
dddd	Отображает день недели полными наименованиями, задаваемыми глобальной переменной **LongDayNames:** "понедельник" "воскресенье".
dddddd	Отображает дату в формате, соответствующем глобальной переменной **LongDateFormat:** день, название месяца, год (четырехзначное число с последующими символами "г.")
m	Отображает месяц числом без предшествующего нуля: 1-12
mm	Отображает месяц двузначным числом: 01-12
mmm	Отображает месяц его аббревиатурой, задаваемой глобальной переменной **ShortMonthNames:** "янв" "дек"
mmmm	Отображает месяц его полным именем, задаваемым глобальной переменной LongMonthNames: "Январь" "Декабрь"
yy	Отображает год двузначным числом: 00-99
yyyy	Отображает год четырехзначным числом: 0000-9999
/	Отображает разделитель дат, заданный глобальной переменной DateSeparator (обычно "/")
'xx'/"xx"	Символы, заключенные в одинарные или двойные кавычки, как и символы, отличные от других спецификаторов, переносятся в результирующую строку без форматирования

Все спецификаторы могут записываться в любом регистре.

Примеры

Оператор

```
Editl->Text = Date O;
```

отобразит в окне **Edit1** дату в формате по умолчанию, например: "27.06.2002". Если же перед этим оператором вставить изменение переменной **ShortDateFormat:**

```
ShortDateFormat = "дата: dddd d/m/yy";
```

то та же дата отобразится в виде: "дата: четверг 27.6.02". А в случае оператора

```
ShortDateFormat = "дата: dddddd";
```

отображение будет иметь вид: "дата: 27 Июнь 2002 г.", т.е. будет использована переменная **LongDateFormat.** Ее тоже можно изменить. Например, если перед отображением даты вы выполните оператор

```
LongDateFormat = "dd rammm yyyy год";
```

то приведенный выше оператор, задающий значение **ShortDateFormat,** приведет к отображению текста: "дата: 27 Июнь 2002 год".

Можете записать операторы

```
ShortDateFormat = Edit2->Text;
Editl->Text = Date();
```

Тогда, вводя в окно **Edit2** различные тексты, передаваемые в **ShortDateFormat,** вы сможете наблюдать в окне **Editl** их влияние на отображение даты и подобрать форму, устраивающую вас.

ShortTimeFormat — переменная

Определяет краткий формат отображения времени.

См. разд. «LongTimeFormat, ShortTimeFormat — переменные».

TAlign, TAlignSet — типы

Определяют выравнивание компонента в контейнере.

Определения

```
enum TAlign {alNone, alTop, alBottom, alLeft, alRight,
             alClient, alCustom};
typedef Set<TAlign, alNone, alclient>  TAlignSet;
```

Описание

Типы **TAlignSet** и **TAlign** используются в ряде свойств и методов для определения способа выравнивания компонента при изменении размеров контейнера или при встраивании в новый контейнер. Возможные значения:

Значение	Описание
alNone	Компонент не выравнивается или остается там, где он размещен во время проектирования. Размеры его не изменяются. Это значение Align по умолчанию.
alTop	Компонент занимает всю верхнюю часть контейнера и во время выполнения приложения его ширина изменяется при изменении ширины контейнера. Высота компонента остается неизменной.
alBottom	Компонент занимает всю нижнюю часть контейнера и во время выполнения приложения его ширина изменяется при изменении ширины контейнера. Высота компонента остается неизменной.

Значение	Описание
alLeft	Компонент занимает всю левую часть контейнера и во время выполнения приложения его высота изменяется при изменении высоты контейнера. Ширина компонента остается неизменной.
alRight	Компонент занимает всю правую часть контейнера и во время выполнения приложения его высота изменяется при изменении высоты контейнера. Ширина компонента остается неизменной.
alClient	Компонент занимает всю клиентскую область контейнера, и во время выполнения приложения его размеры изменяются при изменении размеров контейнера. Если в контейнере часть клиентской области уже занята, компонент занимает всю ее оставшуюся часть.
alCustom	Введено в C++Builder 6. Позиция компонента определяется вызовами функций **CustomAlignPosition** и **CustomAlignInsertBefore.**

TBDEDataSet — базовый класс наборов данных BDE

Базовый класс компонентов наборов данных, использующих Borland Database Engine (BDE).

Иерархия *TObject — TPersistent — TComponent — TDataSet*

Модуль *dbtables.*

Описание

Класс **TBDEDataSet** инкапсулирует основные свойства, методы и события компонентов наборов данных, использующих Borland Database Engine (BDE). Основные дополнения в **TBDEDataSet** по сравнению с родительским классом TDataSet связаны с организацией кэширования данных.

В приложениях объекты **TBDEDataSet** непосредственно не используются. Используются только потомки этого класса и производного от него класса **TDBDataSet: TTable, TQuery, TStoredProc.** При создании новых классов наборов данных удобнее, обычно, в качестве базового использовать не **TBDEDataSet,** а его дочерний класс **TDBDataSet.**

Свойства

Свойство	Объявление / Описание
CacheBlobs	**bool** CacheBlobs Определяет, кэшируются ли поля BLOB в памяти
CachedUpdates	bool CachedUpdates Определяет, кэшируются ли изменения набора данных
ExpIndex	bool ExpIndex Указывает, использует ли набор данных индекс выражений dBASE
RecNo	int RecNo Номер записи
RecordSize	Word RecordSize Размер записи в наборе данных. Только для чтения
UpdateObject	TDataSetUpdateObject* UpdateObject Определяет объект, используемый для обновления кэшируемых результатов «только для чтения»

Свойство	Объявление / Описание
UpdateRecordTypes	**TUpdateRecordTypes UpdateRecordTypes** Определяет, какие записи должны быть видимы в наборе данных при кэшировании
UpdatesPending	**bool** UpdatesPending Определяет, имеются ли в кэше обновленные записи, не отправленные в базу данных

Кроме того, класс TBDEDataSet наследует или переопределяет такие наследуемые от TDataSet свойства, как Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, CanModify, DataSource, Eof, FieldCount, FieldDefs, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, Modified, Name, RecordCount, SparseArrays, State.

Методы

Метод	Объявление / Описание
ApplvUpdates	void ApplyUpdates(void) Записывает кэшированные изменения в базу данных
CancelUpdates	void CancelUpdates(void) Отменяет все кэшированные изменения и восстанавливает исходное состояние набора данных
CommitUpdates	void CommitUpdates(void) Очищает буфер кэшированных изменений
ConstraintsDisabled	bool ConstraintsDisabled(void) Показывает, блокированы, или нет ограничения сервера
DisableConstraints	void DisableConstraints(void) Блокирует ограничения сервера
EnableConstraints	void EnableConstraints(void) Снимает блокировку ограничений сервера, введенную ранее методом **DisableConstraints**
FetchAll	void FetchAll(void) Считывает с сервера и сохраняет локально все записи, начиная с текущей
FlushBuffers	void FlushBuffers(void) Пересылает в базу данных все изменения, сохраненные в буфере
RevertRecord	void **RevertRecord(void)** Отменяет исправления текущей записи

Кроме того, наследуются и переопределяются такие методы TDataSet. как Append, AppendRecord, BookmarkValid, Cancel, ClearFields, Close, CompareBookmarks, Delete, DisableControls, Edit, EnableControls, FieldByName, FindField, FindFirst, FindLast, FindNext, FindPrior, First, FreeBookmark, GetBookmark, GetDetailDataSets, GetDetailLinkFields, GetFieldNames, GotoBookmark, Insert, InsertRecord, IsEmpty, Last, Locate, Lookup, MoveBy, Next, Open, Post, Prior, Refresh, SetFields, Translate, UpdateStatus и некоторые другие.

События
В классе наследуются все события **TDataSet** и вводится два новых:

Событие	Описание
OnUpdateError	Наступает при генерации исключения в процессе пересылки в базу данных измененной записи
OnUpdateRecord	Наступает при пересылке кэшированной записи в базу данных

TBitmap — класс

Инкапсулирует битовую матрицу Windows (HBITMAP), включая палитру (HPALETTE).

Иерархия *TObiect — TPersistent — TGraphic*

Модуль *graphics.*

Описание
Класс **TBitmap** инкапсулирует битовую матрицу Windows, включая палитру. Обеспечивает быстрое и простое для пользователя выполнение операций создания, копирования, преобразования и сохранения битовой матрицы.

Свойства
Ниже приведен список основных свойств, определенных или переопределенных в **TBitmap.**

Свойство	Объявление / Описание
Canvas	TCanvas* Canvas Определяет пространство для изображения битовой матрицы. Свойство только для чтения
Empty	**bool** Empty Указывает, содержит ли объект битовую матрицу. Свойство только для чтения
Handle	HBITMAP Handle Обеспечивает доступ к обработке битовых матриц в GDI Windows. Используется при вызовах функций API Windows
HandleType	**enum** TBitmapHandleType {**bmDIB,** bmDDB}; TBitmapHandleType HandleType Указывает, является ли битовая матрица DDB (Device Dependent Bitmap — аппаратно зависимой), или DIB (Device Independent Bitmap — аппаратно независимой). Может изменяться пользователем
Height	**int** Height Указывает высоту изображения в пикселах. Может изменяться пользователем, что вызывает создание копии матрицы с указанным размером
IgnorePalette	bool IgnorePalette Определяет, использует ли матрица палитру. При установке в **true** ухудшается качество, но ускоряется рисование
Modified	bool Modified Определяет, было ли модифицировано изображение после его загрузки

Свойство	Объявление / Описание
MaskHandle	HBITMAP MaskHandle Обеспечивает доступ к обработке битовых матриц в GDI Windows. Используется при вызовах функций API Windows. Свойство только для чтения
Monochrome	**bool** Monochrome Определяет, является ли битовая матрица монохромной (значение true)
Palette	HPALETTE Palette Управляет цветами битовой матрицы. Если изображение не нуждается в палитре или не имеет палитры, то Palette = 0
PixelFormat	enum TPixelFormat **{pfDevice,** pflbit, pf4bit, **pf8bit, pf15bit, pf16bit, pf24bit,** pf32bit, **pfCustom};** TPixelFormat PixelFormat Определяет битовый формат отображения изображения. Используется для задания формата видеодрайверам, не способным прочитать собственный формат битовой матрицы
ScanLine	void * **ScanLine[int** Row] Обеспечивает доступ к отдельным строкам пикселов для их низкоуровневой обработки для матриц DIBs (Device Independent Bitmap — аппаратно независимых). Свойство только для чтения
Transparent	bool Transparent Определяет, должно ли изображение быть «прозрачным»
Transparent-Color	TColor TransparentColor Определяет, какой из цветов будет прозрачным при рисовании битовой матрицы. Читаемое значение зависит от значения TransparentMode
Transparent-Mode	enum TTransparentMode {tmAuto, tmFixed}; **TTransparentMode** TransparentMode Указывает, определяется цвет прозрачности левым нижним пикселом (tmAuto — по умолчанию), или свойством TransparentColor, сохраненным вместе с битовой матрицей
Width	**int** Width Указывает ширину изображения в пикселах. Может изменяться пользователем, что вызывает создание копии матрицы с указанным размером

Методы

Ниже приведены основные методы, объявленные или переопределенные в классе **TBitmap.**

Метод	Объявление / Описание
Assign	void Assign(Classes::TPersistent* Source) Копирует изображение из другого графического объекта, в частности, из буфера обмена Clipboard

Метод	Объявление / Описание
Dormant	void Dormant(void) Создает изображение битовой матрицы в памяти, чтобы освободить дескриптор матрицы и сэкономить ресурсы
FreeImage	void FreeImage(void) Освобождает память, занятую кэшированием изображения, экономит ресурсы, но может вести к потере глубины цвета
LoadFromClipboardFormat	void LoadFromClipboardFormat(Word AFormat, int AData, HPALETTE APalette) Читает изображение из буфера обмена Clipboard в заданном формате
LoadFromFile	void LoadFromFile(const AnsiString FileName) Читает изображение из файла FileName
LoadFromResourceID	void LoadFromResourceID(unsigned Instance, int ResID) Загружает битовую карту из ресурса по указанному идентификатору ResID
LoadFromResourceName	void LoadFromResourceName(unsigned Instance, const AnsiString ResName) Загружает битовую карту из ресурса по указанному имени ResName
LoadFromStream	void LoadFromStream(Classes::TStream* Stream) Читает графическое изображение из указанного потока Stream
Mask	void Mask(TColor TransparentColor) Преобразует изображение в монохромную маску, преобразуя цвет TransparentColor в белый, а остальные — в черный
ReleaseHandle	HBITMAP ReleaseHandle(void) Возвращает дескриптор типа HBitmap и очищает объект TBitmap от этого дескриптора
ReleaseMaskHandle	HBITMAP ReleaseMaskHandle(void) Возвращает дескриптор маски типа HBitmap и очищает объект TBitmap от этого дескриптора
ReleasePalette	HBITMAP ReleaseMaskHandle(void) Возвращает дескриптор палитры типа HPalette и разрывает связь палитры с объектом TBitmap
SaveToClipboardFormat	void SaveToClipboardFormat(Word &AFormat, int &AData, HPALETTE &APalette) Сохраняет изображение в буфере обмена Clipboard в заданном формате
SaveToFile	void SaveToFile(const AnsiString Filename) Сохраняет изображение *в* файле
SaveToStream	void SaveToStream(Classes::TStream* Stream) Записывает изображение в поток

События

Событие	Описание
OnChange	Событие при изменении графического объекта
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

TBrush — класс

Определяет свойства кисти: цвет и стиль заполнения фона окна.

Иерархия *TObiect — TPersistent — TGraphicsObject*

Модуль *Graphics.*

Описание

Тип TBrush инкапсулирует структуру **HBRUSH** Windows и используется для заполнения форм заданным цветом и стилем. **TBrush** используется во многих объектах, в частности, в свойстве кисть — **Brush.**

Свойства

Свойство	Объявление / Описание
Bitmap	TBitmap* Bitmap Указатель на внешнюю матрицу побитового отображения, используемую как шаблон заполнения
Color	TColor Color Цвет кисти. Если Style = bsClear, значение **Color** игнорируется
Handle	HBRUSH Handle Дескриптор кисти окна, определяющий доступ к дескриптору объекта GDI Windows
Style	enum **TBrushStyle** {bsSolid, bsClear, bsHorizontal, bsVertical, **bsFDiagonal,** bsBDiagonal, bsCross, bsDiagCross} TBrushStyle Style Определяет стиль заполнения окна

Методы

В классе **TBrush** не введено каких-то принципиально новых методов. Переопределены такие общие методы, как Assign, конструктор и деструктор. Остальные методы наследуются от классов-предков.

События

Класс **TBrush** наследует от класса **TGraphicsObject** событие **OnChange,** наступающее после изменения графического объекта. Обрабатывая его графический объект должен учесть новые установки **TBrush.**

Пример

Следующие операторы изменяют цвет и стиль заполнения объекта **Image1.Canvas** — канвы компонента **Image1:**

```
Image1->Canvas->Brush->Style = bsCross;
Image1->Canvas->Brush->Color=clRed;
Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));
```

Последний из приведенных операторов заполняет методом **FillRect** всю поверхность канвы.

TCanvas — класс

Обеспечивает пространство (холст, канву) для создания, хранения и модификации графических объектов.

Иерархия *TObiect — TPersistent*

Модуль *graphics.*

Описание

Класс **TCanvas** является основой графической подсистемы C++Builder. Канва обеспечивает:

- Загрузку и хранение графических изображений
- Создание новых и изменение хранимых изображений с помощью пера, кисти, шрифта
- Рисование и закраску различных фигур, линий, текстов
- Комбинирование различных изображений

Класс **TCanvas** имеет два дочерних класса — **TControlCanvas и TMetafileCanvas,** которые помогают в прорисовке управляющих элементов и в создании для объекта метафайла.

Свойства

Ниже приведен список основных свойств, определенных или переопределенных в **TCanvas.**

Свойство	Объявление / Описание
Brush	TBrush* Brush Определяет цвет и стиль заполнения замкнутых фигур и фона
Canvas Orientation	enum TCanvasOrientation { coLeftToRight, coRightToLeft }; TCanvasOrientation CanvasOrientation Определяет обычную (слева направо) или восточную (справа налево) ориентацию канвы и ее координат. Свойство только для чтения
ClipRect	TRect ClipRect Определяет доступную область рисования на канве и область, подлежащую перерисовке при событии OnPaint. Свойство только для чтения
CopvMode	int CopyMode Определяет режим копирования графического изображения на канву
Font	TFont* Font Определяет атрибуты шрифта, которым выводится текст
LockCount	int LockCount Определяет, сколько раз блокирована канва в многопоточных приложениях. Свойство только для чтения
Pen	TPen* Pen Определяет свойства пера, рисующего линии и фигуры

Свойство	Объявление / Описание
PenPos	**TPoint PenPos** **Определяет текущую позицию пера**
Pixels	**TColor Pixels[int X][int Y]** **Определяет цвета пикселов**
TextFlags	**int TextFlags** **Определяет способ вывода текста на канву**

Методы

Ниже приведены основные методы, объявленные в классе **TCanvas.**

Метод	Объявление / Описание
Arc	void Arc(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4) Рисует дугу окружности или эллипса. (X1,Y1) и (X2,Y2) определяют описанный прямоугольник. (X3,Y3) и (X4,Y4) — точки, через которые проходят радиусы, отмечающие начало и конец дуги
BrushCopy	void BrushCopy(const Types::TRect &Dest, TBitmap* Bitmap, const Types::TRect &Source, TColor Color) Копирует часть Source изображения битовой матрицы Bitmap на данную канву в область Dest, заменяя указанный цвет Color в изображении на значение, установленное для кисти канвы
Chord	void Chord(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4) Рисует замкнутую фигуру, ограниченную дугой окружности или эллипса и хордой. (X1,Y1) и (X2,Y2) определяют описанный прямоугольник. (X3,Y3) и (X4,Y4) — точки, через которые проходит хорда
CopyRect	void CopyRect(const TRect &Dest, TCanvas* Canvas, const TRect &Source) Копирует часть Source изображения с другой канвы Canvas на данную в область Dest
Draw	void Draw(int X, int Y, TGraphic* Graphic) Рисует графическое изображение Graphic в указанную позицию канвы (X, Y — левый верхний угол)
DrawFocus Rect	void DrawFocusRect(const TRect &Rect) Рисует изображение прямоугольника в виде, используемом для отображения рамки фокуса, операцией xor

Ellipse	void Ellipse(int X1, int Y1, int X2, int Y2);
	void Ellipse(TRect Rect);
	Рисует окружность или эллипс. (X1, Y1) и (X2, Y2) или Rect определяют описанный прямоугольник

FillRect	void FillRect(const TRect &Rect) Заполняет указанный прямоугольник канвы, используя текущее значение кисти Brush

Метод	Объявление / Описание
FloodFill	enum TFillStyle {fsSurface, fsBorder}; void FloodFill(int X, int Y, TColor Color, TFillStyle FillStyle) Закрашивает текущей кистью замкнутую область канвы, определенную цветом Color и начальной точкой закрашивания (X, Y). При FillStyle = fsSurface заполняется область, окрашенная цветом Color, а при FillStyle = fsBorder, заполняется область, окрашенная любыми цветами, не равными Color
FrameRect	void FrameRect(const Types::TRect &Rect) Рисует на канве текущей кистью прямоугольную рамку
LineTo	void LineTo(int X, int Y) . Рисует на канве прямую линию, начинающуюся с текущей позиции пера и кончающуюся указанной точкой (исключая ее)
Lock	void Lock(void) Блокирует канву, увеличивая LockCount на 1 и не разрешая другим нитям многопоточного приложения рисовать на ней
MoveTo	void MoveTo(int X, int Y) Изменяет текущую позицию пера на заданную, ничего не рисуя
Pie	void Pie(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4) Рисует сектор окружности или эллипса. (X1,Y1) и (X2,Y2) определяют описанный прямоугольник. (X3,Y3) и (X4,Y4) — точки, через которые проходят радиусы, ограничивающие сектор
PolyBezier	void PolyBezier(const TPoint* Points, const int Points_Size) Рисует на канве текущим пером кусочную кривую третьего порядка, сглаживающую заданное множество точек Points. Число точек должно быть на единицу больше числа, кратного 3 (т.е. i*3+1)
PolyBezier To	void PolyBezierTo(const TPoint * Points, const int Points Size) Рисует на канве текущим пером кусочную кривую третьего порядка, сглаживающую заданное множество точек Points. Число точек должно быть кратно 3 (т.е. i*3)
Polygon	void Polygon(const TPoint * Points, const int Points Size) Рисует замкнутую фигуру с кусочно-линейной границей
Polyline	void Polyline(const Types::TPoint* Points, const int Points Size) Рисует кусочно-линейную кривую

Rectangle	void Rectangle(int X1, int Y1, int X2, int Y2);
	void Rectangle(TRect Rect) d;
	Рисует прямоугольник, заданный углами (X1, Y1) и (X2, Y2) или Rect

RoundRect	void RoundRect(int X1, int Y1, int X2, int Y2, int X3, int Y3) Рисует прямоугольник со скругленными углами: (X1, Y1) и (X2, Y2) — прямоугольник, X3 и Y3 — ширина и высота эллипса скругления

Метод	Объявление / Описание
StretchDraw	void StretchDraw(const TRect &Rect, TGraphic* Graphic) Рисует графическое изображение Graphic в указанную прямоугольную область канвы Rect, подгоняя размер изображения под заданную область

TextExtent	typedef struct TSize
	{
	long cx;
	long cy;
	}
	TSize TextExtent(const AnsiString Text)
	Возвращает длину и высоту в пикселах текста, который предполагается написать на канве текущим шрифтом

TextHeight	int TextHeight(const AnsiString Text) Возвращает высоту в пикселах текста, который предполагается написать на канве текущим шрифтом
TextOut	void TextOut(int X, int Y, const AnsiString Text) Пишет указанную строку текста Text на канве, начиная с указанной позиции
TextRect	void TextRect(const Types::TRect &Rect, int X, int Y, const AnsiString Text) Пишет указанную строку текста Text на канве, начиная с указанной позиции и усекая текст, выходящий за пределы указанной прямоугольной области Rect
TextWidth	int TextWidth(const AnsiString Text) Возвращает длину в пикселах текста Text, который предполагается написать на канве текущим шрифтом
TryLock	bool TryLock(void) Блокирует канву, если она не была блокирована, не разрешая другим нитям многопоточного приложения рисовать на ней
Unlock	void Unlock(void) Уменьшает на единицу значение свойства LockCount, способствуя тем самым разблокированию канвы, когда LockCount станет равным 0

Кроме того, поддерживается множество методов, определенных в классах-предшественниках, в частности, **Assign, ClassName,** ClassNameIs, **Free** и многие другие.

События

Событие	Описание
OnChange	Событие после изменения изображения
OnChanging	Событие перед изменением изображения

TCollection — базовый класс собраний (коллекций) объектов

Контейнер собраний объектов TCollectionItem.

Иерархия *TObject — TPersistent*

Модуль *classes.*

Описание

Класс **TCollection** является контейнером для группы (собрания, коллекции) объектов класса **TCollectionItem**. Его основное свойство **Items** — массив объектов. Свойство **Count** определяет число объектов в контейнере. Методы **Add, Delete, Insert и Clear** позволяют добавлять и удалять объекты, содержащиеся в контейнере.

Классы-наследники **TCollection,** содержащие объекты классов-наследников **TCollectionItem,** используются во многих классах и компонентах C++Builder:

Наследник TCollection	Наследник TCollectionItem	Класс, компонент
TAggregates	TAggregate	TClientDataSet
TCheckConstraints	**TCheckConstraint**	компоненты, использующие TField
TCookieCollection	TCookie	объекты HTTP
TCoolBands	TCoolBand	TCoolBar
TDataBindings	**TDataBindItem**	связанные с данными ActiveX
TDBGridColumns	TColumn	TDBGrid
TDependencies	TDependency	TService
TDisplayDims	TDisplayDim	**TDecisionGrid**
TFieldDefs	**TFieldDef**	TDataSet
THeaderSections	THeaderSection	THeaderControl
TIndexDefs	TIndexDef	TTable
THTMLTableColumns	**THTMLTableColumn**	TQueryTableProducer
TListColumns	TListColumn	**TListView**
TParams	TParam	TQuery, **TStoredProc,** TClientDataSet
TParameters	**TParameter**	компоненты ADO
TServerCollection	**TServerItem**	**TSimpleObjectBroker**
TStatusPanels	TStatusPanel	TStatusBar
TWorkAreas	TWorkArea	TListView
TWebActionItems	**TWebActionItem**	**TWebDispatcher**

Свойства

Свойство	Объявление / Описание
Count	**int Count** **Число объектов в Items. Только для чтения**
ItemClass	**class PACKAGE TMetaClass;** **typedef TMetaClass* TClass;** **System::TMetaClass* ItemClass** **Класс содержащихся в Items объектов. Только для чтения**

Свойство	Объявление / Описание
Items	TCollectionItem* Items[int Index] Индексированный массив объектов, хранящихся в TCollection

Основные методы

Метод	Объявление / Описание
Add	TCollectionItem* Add(void) Создает новый объект TCollectionItem и добавляет его в массив Items
Assign	void fastcall Assign(TPersistent* Source) Копирует содержимое собрания объектов в другой контейнер
Begin Update	void BeginUpdate(void) Предотвращает перерисовку до выполнения метода EndUpdate
Clear	void Clear(void); Очищает массив Items, уничтожая все содержавшиеся в нем объекты типа TCollectionItem
Delete	void Delete(int Index) Удаляет объект с индексом Index (отсчитываются от 0) из Items
EndUpdate	void EndUpdate(void) Снимает запрет перерисовки, введенный методом BeginUpdate
FindItem ID	TCollectionItem* FindItemID(int ID) Возвращает элемент с заданным свойством ID, или NULL, если элемент не находится
Insert	TCollectionItem* Insert(int Index) Создает новый объект TCollectionItem и вставляет его в массив Items

TCollectionItem — класс объектов в собраниях TCollection

Класс объектов, содержащихся в собраниях **TCollection.**

Иерархия *TObject — TPersistent*

Модуль *classes.*

Описание

Класс **TCollectionItem** описывает объекты, которые могут содержаться в собраниях объектов **TCollection.** Каждый объект **TCollectionItem** имеет свойство Collection, указывающее на содержащий его контейнер. Объекты **TCollectionItem** создаются и уничтожаются методами своего контейнера **TCollection: Add. Clear. Delete.**

Класс **TCollectionItem** имеет множество классов-наследников (см. подробнее в разделе «TCollection»).

Свойства

Свойство	Объявление / Описание
Collection	TCollection* Collection Указывает собрание TCollection, к которому относится данный объект

Свойство	Объявление / Описание
DisplayName	AnsiString DispIayName Строка, появляющаяся в Редакторе Собраний (наборов) во время проектирования
ID	int ID Идентификатор объекта. Только для чтения
Index	int Index Определяет позицию объекта в свойстве Items его контейнера TCollection

Методы

Никаких новых методов в классе **TCollectionItem** не объявлено. Переопределены некоторые методы классов-предшественников.

TCoIor — тип

Определяет цвет объекта.

Модуль *Graphics*

Определение

```
type TColor = -(COLOR_ENDCOLORS + 1)..$02FFFFFF;
```

Описание

Тип **TColor** используется для описания цвета объекта. Применяется в свойствах **Color** многих компонентов, в подсвойствах объекта типа **TFont,** при прорисовке изображений, в таблицах и т.д.

В модуле *Graphics* определено множество констант типа **TColor.** Одни из них непосредственно определяют цвета (например **clBlue** — синий), другие определяют цвета элементов окон, которые могут меняться в зависимости от выбранной пользователем палитры цветов Windows (например, **clBtnFace** — цвет поверхности кнопок). Значение цвета может задаваться равным одной из перечисленных ниже предопределенных в C++Builder констант.

Константа	Значение цвета
clBlack	Черный
clMaroon	Темно-бордовый
clGreen	Зеленый
clOlive	Оливково-зеленый
clNavy	Темно-синий
clPurple	Пурпурный
clTeal	Морской воды
clGray	Серый
clSilver	Серебряный
clRed	Красный
clLime	Лимонно-зеленый
clBlue	Синий

Константа	Значение цвета
clYellow	Желтый
clFuchsia	Сиреневый
clAqua	Голубой
clWhite	Белый
clBackground	Текущий цвет фона стола Windows
clScrollBar	Текущий цвет полос прокрутки
clActiveCaption	Текущий цвет фона полосы заголовка в активном окне
clInactiveCaption	Текущий цвет фона полосы заголовка в неактивном окне
clMenu	Текущий цвет фона меню
clWindow	Текущий цвет фона окон
clWindowFrame	Текущий цвет рамок окон
clMenuText	Текущий цвет текста меню
clWindowText	Текущий цвет текста окон
clCaptionText	Текущий цвет текста заголовка в активном окне
clActiveBorder	Текущий цвет бордюра активного окна
clInactiveBorder	Текущий цвет бордюра неактивного окна
clAppWorkSpace	Текущий цвет рабочей области приложений
clHighlight	Текущий цвет фона выделенного текста
clHightlightText	Текущий цвет выделенного текста
clBtnFace	Текущий цвет поверхности кнопок
clBtnShadow	Текущий цвет теней, отбрасываемых кнопками
clGrayText	Текущий цвет текста недоступных элементов
clBtnText	Текущий цвет текста кнопок
clInactiveCaptionText	Текущий цвет текста заголовка в неактивном окне
clBtnHighlight	Текущий цвет выделенной кнопки
cl3DDkShadow	Цвет темных теней трехмерных элементов; только для Windows 95 или NT 4.0
cl3DLight	Светлый цвет на краях освещенных трехмерных элементов; только для Windows 95 или NT 4.0
clInfoText	Цвет текста советов; только для Windows 95 или NT 4.0
clInfoBk	Цвет фона советов; только для Windows 95 или NT 4.0
clNone	Белый для Windows 9x,черный для Windows NT/XP
clGradientActive Caption	В Windows 98/2000/XP правый цвет перелива заголовка активного окна
clGradientInactive Caption	В Windows 98/2000/XP правый цвет перелива заголовка неактивного окна
clDefault	Цвет компонента по умолчанию

Первая часть этих констант соответствует определенным цветам. А вторая часть определяется той схемой цветов, которую установил пользователь в Windows. Пользователь может менять эту схему с помощью «Панели Управления» Windows. Таким образом, эти цвета могут изменяться от системы к системе. Например, **clBtnFace** может соответствовать серому цвету в одной схеме и желтовато-коричневому в другой.

Как правило, лучше использовать в приложении эти системно-зависимые цвета. Тогда ваше приложение не будет выбиваться из общей гаммы цветов, на которую настроился пользователь.

Вместо использования этих констант можно задавать **TColor** как 4-байтовое шестнадцатеричное число, три младших разряда которого представляют собой интенсивности синего, зеленого и красного цвета соответственно. Например, значение $00FF0000 соответствует чистому синему цвету, $0000FF00 — чистому зеленому, $000000FF — чистому красному. $00000000 — черный цвет, $00FFFFFF — белый.

Если старший байт равен нулю ($00), то берется ближайший к заданному цвет из системной палитры. Если старший байт равен единице ($01), то берется ближайший к заданному цвет из текущей палитры. Если старший байт равен двум ($02), то берется ближайший к заданному цвет из логической палитры контекста данного устройства.

TComponent — базовый класс компонентов

Базовый класс всех компонентов Delphi.

Иерархия *TObiect — TPersistent*

Модуль *classes.*

Описание

Класс **TComponent** является прародителем всех компонентов C++Builder. Он инкапсулирует наиболее общие свойства и методы компонентов, включая:

- Возможность включать компонент в палитру компонентов и работать с ним при визуальном проектировании.
- Способность быть владельцем других компонентов или управляться другими компонентами.
- Возможности обмена с потоками и файлами.
- Возможность служить оболочкой элементов ActiveX и других объектов.

Объекты типа **TComponent** не создаются. Класс **TComponent** используется как базовый, когда объявляется класс невизуального компонента, который может присутствовать в палитре компонентов и применяться в процессе проектирования. Для создания визуальных компонентов в качестве базового используется класс **TControlI** или его потомки. Для создания оконных компонентов в качестве базового используется класс **TWinControl** или его потомки.

Свойства

Свойство	Объявление / Описание
ComObject	**typedef** System::DelphiInterface< **IUnknown** > _di_IUnknown; _di_IUnknown ComObject Защищенное свойство, возвращающее ссылку на интерфейс в компонентах, поддерживающих стандарт COM. Только для чтения

Свойство	Объявление / Описание
ComponentCount	int ComponentCount Число компонентов, которыми владеет данный компонент. Равно количеству элементов в массиве Components. На 1 меньше индекса последнего компонента ComponentIndex, поскольку индексы отсчитываются от 0. Может использоваться вместе с ComponentIndex в циклах, когда надо изменить какие-то свойства всех компонентов. Только для чтения
ComponentIndex	int ComponentIndex Индекс компонента в массиве Components владельца данного компонента. Отсчитывается от 0, т.е. индекс первого компонента — 0. Может использоваться вместе с ComponentCount в циклах, когда надо изменить какие-то свойства всех компонентов
Components	**TComponent* Components[int** Index] Массив компонентов, которыми владеет данный компонент. Позволяет сослаться на любой компонент с помощью его ComponentIndex. Только для чтения
ComponentState	enum Classes__34 {csLoading, csReading, **csWriting,** csDestroying, csDesigning, csAncestor, **csUpdating,** csFixups, **csFreeNotification, csInline, csDesignInstance};** typedef Set<Classes__34, csLoading, **csDesignInstance**> TComponentState; **TComponentState** ComponentState Состояние компонента в процессе визуального проектирования. Свойство только для чтения. Во время выполнения не используется. Только для чтения
ComponentStyle	**TComponentStyle** ComponentStyle Определяет множество флагов стиля компонента, в частности, стиль **csInheritable** указывает, что компонент может принадлежать форме. Только для чтения
DesignInfo	int DesignInfo Используется только в среде C++Builder при проектировании форм. В приложениях не используется
Name	**AnsiString** Name Имя компонента, по которому производится ссылка на него из других компонентов
Owner	TComponent* Owner Определяет владельца данного компонента
Tag	int Tag Это свойство не используется в C++Builder. Разработчик может использовать его по своему усмотрению
VCLComObject	void * VCLComObject Используется только в среде C++Builder для компонентов, поддерживающих стандарт COM

Методы

В классе TComponent определено множество методов. Ниже приводятся только те из них, которые наиболее часто используются пользователями при работе с компонентами. Остальные используются для внутренних потребностей C++Builder.

Метод	Объявление / Описание
ExecuteAction	**bool ExecuteAction(TBasicAction* Action)** Вызывает указанное действие Action, связанное с данным компонентом
FindComponent	**TComponent* FindComponent(const AnsiString AName)** Ищет в списке **Components** компонент с заданным именем **AName**
FreeNotification	**void FreeNotification(TComponent* AComponent)** Гарантирует, что указанный в вызове компонент **AComponent** будет разрушен. Используется только по отношению к компонентам, расположенным на других формах. Для компонентов на текущей форме вызывается автоматически
InsertComponent	**void InsertComponent(TComponent* AComponent)** Добавление указанного компонента **AComponent** в конец списка компонента-владельца. При визуальном проектировании вызывается автоматически. Специально может потребоваться вызов этого метода только при добавлении компонента в список другого владельца
RemoveComponent	**void RemoveComponent(TComponent* AComponent)** Удаление указанного компонента **AComponent** из списка компонента-владельца. При визуальном проектировании вызывается автоматически. Специально может потребоваться вызов этого метода только при удалении компонента из список другого владельца

Класс **TComponent** наследует также методы Assign, ClassName, ClassNameIs, Free и многие другие.

TControl — базовый класс визуальных компонентов

Абстрактный базовый класс всех визуальных компонентов Delphi.

Иерархия *TObiect — TPersistent— TComponent*

Модуль *controls.*

Описание

Класс **TControl** является базовым абстрактным классом для всех визуальных компонентов Delphi, т.е. для компонентов, которые пользователь может видеть и которыми манипулирует во время выполнения приложения. Все они имеют общие свойства, методы и события, определяющие место их размещения, расцветку, реакцию на нажатие клавиш или кнопок мыши и т.д.

Защищенные свойства и методы класса **TControl** используются в их потомках. Если требуется создать новый класс визуального компонента, его надо создавать как производный от **TControl** или от его потомков.

Свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данным управляющим элементом. Это действие определяется компонентом типа **TActionList**
Align	TAlign Align Определяет способ выравнивания компонента в контейнере (родительском компоненте): без выравнивания, по верху, по низу, по левой стороне, по правой стороне, по всей клиентской области и заказное
Anchors	property Anchors: **TAnchors;** Определяет привязку данного компонента к родительскому при изменении размеров последнего: к верней грани, к левой грани, к правой грани, к нижней грани
AutoSize	**enum TAnchorKind** { akLeft, akTop, akRight, akBottom }; typedef **Set<TAnchorKind,** akLeft, akBottom> TAnchors; TAnchors Anchors Определяет, будет ли высота элемента автоматически адаптироваться к размеру символов текста
BiDiMode	enum TBiDiMode { bdLeftToRight, bdRightToLeft, bdRightToLeftNoAlign, bdRightToLeftReadingOnly }; **Classes**"TBiDiMode BiDiMode Определяет двунаправленный характер компонента. По умолчанию равно bdLeftToRight, что характерно для европейских языков. Другие значения используются для стран Востока
BoundsRect	**Types::TRect** BoundsRect Определяет координаты углов компонента в координатах содержащего его контейнера
Caption	AnsiString Caption Строка текста, идентифицирующая компонент для пользователя. Обычно это надпись на метке, кнопке и др. компонентах
ClientHeight	int ClientHeight Высота клиентской области в пикселах
ClientOrigin	Types::TPoint ClientOrigin Координаты положения на экране левого верхнего угла клиентской области компонента. Свойство только для чтения
ClientRect	Types::TRect ClientRect Определяет координаты углов клиентской области компонента
ClientWidth	int ClientWidth Горизонтальный размер клиентской области в пикселах
Color	Graphics::TColor Color Цвет фона компонента

Свойство	Объявление / Описание
Constraints	TSizeConstraints* Constraints Позволяет задавать ограничения на допустимые изменения размеров компонента при изменениях размеров окна приложения: MaxHeight, MaxWidth, MinHeight, MinWidth — максимальную и минимальную высоту и ширину
ControlState	TControlState ControlState Характеризует текущее состояние компонента во время выполнения приложения. Используется при создании новых классов
ControlStyle	TControlStvle ControlStyle Определяет характеристики стиля компонента. Используется при создании новых классов
Cursor	TCursor Cursor Определяет вид курсора мыши при попадании его в область компонента
DesktopFont	bool DesktopFont Определяет, использует ли компонент для отображения текста изображение шрифта Windows
Dock Orientation	enum TDockOrientation {doNoOrient, doHorizontal, do Vertical}; TDockOrientation DockOrientation Определяет позицию данного встраиваемого компонента относительно других встроенных компонентов
DragCursor	TCursor DragCursor Определяет вид курсора мыши, при попадании его в область компонента в процессе перетаскивания
DragKind	enum TDragKind { dkDrag, dkDock }; TDragKind DragKind Определяет, будет ли объект перетаскиваться по технологии Drag&Drop (dkDrag), или Drag&Doc (dkDock)
DragMode	enum TDragMode { dmManual, dmAutomatic }; TDragMode DragMode Определяет автоматическое (dmAutomatic) или программное (dmManual) начало процесса перетаскивания
Enabled	bool Enabled Определяет, реагирует ли компонент на события, связанные с мышью, клавиатурой и таймером
Floating	bool Floating Определяет, находится ли компонент в состоянии «плавающего» окна. Свойство только для чтения
FloatingDock SiteClass	class PACKAGE TMetaClass; typedef TMetaClass* TClass; TMetaClass* FIoatingDockSiteClass Определяет класс временного компонента, управляющего плавающим окном

Свойство	Объявление / Описание
Font	Graphics::TFont* Font Определяет атрибуты шрифта
Height	int Height Высота компонента в пикселах
HelpContext	typedef int THelpContext; Classes::THelpContext HelpContext Определяет номер темы контекстной справки, связанной с компонентом. Свойство работает, если HelpType = htContext
HelpKeyword	AnsiString HelpKeyword Определяет идентификатор темы контекстно-зависимой справки. Свойство работает, если HelpType = htKeyword
HelpType	enum THelpType {htKeyword, htContext}; Classes::THelpType HelpType Определяет, какое свойство — HelpContext или HelpKeyword указывает тему контекстно-зависимой справки
HostDockSite	TWinControl* HostDockSite Определяет контейнер, в который встроен данный компонент
Hint	property Hint: string; Определяет текст подсказки
IsControl	bool IsControl Определяет, сохраняет ли форма свои специфические свойства в поток. Свойство защищенное. Используется при создании новых компонентов
Left	int Left Координата левого края компонента в пикселах
LRDockWidth	int LRDockWidth Ширина компонента, когда он в последний раз размещался в контейнере горизонтально. Свойство только для чтения
MouseCapture	bool MouseCapture Определяет, может ли компонент захватываться мышью. Свойство защищенное. Используется при создании новых компонентов
Name	AnsiStringName Имя компонента
Parent	TWinControl* Parent Определяет родительский компонент, в площади которого располагается данный компонент
ParentBiDiMode	bool ParentBiDiMode Определяет, будет ли для компонента заимствовано свойство BiDiMode родительского компонента
ParentColor	bool ParentColor Определяет, будет ли для компонента заимствован цвет родительского компонента

Свойство	Объявление / Описание
ParentFont	**bool** ParentFont Включает и выключает использование шрифта родительского компонента
ParentShowHint	bool **ParentShowHint** Включает и выключает использование свойства ShowHint родительского компонента
PopupMenu	**Menus::TPopupMenu*** PopupMenu Определяет связанный с компонентом объект всплывающего меню
ScalingFlags	enum Controls__9 { sfLeft, **sfTop**, sfWidth, **sfHeight**, sfFont, **sfDesignSize** }; **typedef Set<Controls__6**, sfLeft, **sfFont>** TScalingFlags; TScalingFlags ScalingFlags Показывает, какие атрибуты компонента еще не отмасштабированы. Используется при разработке новых компонентов
ShowHint	bool ShowHint Разрешает или запрещает показывать ярлычок подсказки, текст которого задается свойством Hint
TBDockHeight	int TBDockHeight Высота компонента, когда он в последний раз размещался в контейнере вертикально. Свойство только для чтения
Text	AnsiString Text Текст, связанный с данным компонентом
Top	int Top Координата верхнего края компонента в пикселах
UndockHeight	int UndockHeight Высота компонента, которая была в последний раз, когда он отображался плавающим окном. Свойство только для чтения
UndockWidth	int UndockWidth Ширина компонента, которая была в последний раз, когда он отображался плавающим окном. Свойство только для чтения
Visible	bool Visible Делает компонент видимым или невидимым
Width	int Width Горизонтальный размер компонента в пикселах
WindowProc	type **TWndMethod** = **procedure(var** Message: TMessage) of object; Classes::TWndMethod WindowProc Указывает на оконную процедуру обработки сообщений, поступающих компоненту. Используется при создании новых компонентов
WindowText	char* WindowText Содержит текст, связанный с данным компонентом

Помимо перечисленных свойств класс **TControl** наследует также ряд свойств **TComponent: ComObject,** ComponentCount, ComponentIndex, **Components,** ComponentState, **ComponentStyle,** DesignInfo, **Owner,** *Tuag*, **VCLComObject.**

Методы

Ниже приводится таблица только тех методов, которые могут применяться пользователями компонентов или разработчиками не очень сложных компонентов. Помимо перечисленных в классе определено еще много методов, интересных только для разработчиков сложных новых компонентов.

Метод	Объявление / Описание
BeginDrag	void **BeginDrag(bool** Immediate, int Threshold = **-1)** Начинает процесс перетаскивания компонента
BringToFront	void **BringToFront(void)** Переносит компонент в Z-последовательности выше других компонентов на той же форме
Changed	void Changed(void) Используется, чтобы послать сообщение CM_CHANGED родительскому компоненту, если в свойствах данного компонента сделаны какие-то изменения, на которые должен прореагировать родительский компонент
ChangeScale	void ChangeScale(int M, int D) Изменяет масштаб и расположение компонента в отношении M /D
Click	void Click(void) Вызывает обработчик события OnClick при щелчке мыши. Используется при проектировании новых классов
ClientToParent	TPoint **ClientToParent(const** TPoint Point, **TWinControl** *AParent = **(TWinControl*)** NULL) Пересчитывает координаты точки компонента в координаты указанного родительского компонента
ClientToScreen	Types::TPoint **ClientToScreen(const Types::TPoint** &Point) Преобразует координаты Point клиентской области в координаты экрана
Create	virtual TControl(Classes::TComponent* **AOwner)** Создает экземпляр TControl. Но так как TControl — абстрактный класс, создавать его экземпляры нельзя. Метод переопределен в потомках. Но и их экземпляры надо создавать операцией new
DblClick	void DblClick(void) Вызывает обработчик события OnDblClick при двойном щелчке мыши. Используется при проектировании новых классов
DefaultHandler	virtual void DefaultHandler(void *Message) Обработчик сообщений Windows по умолчанию. Используется, наряду с методом, на который ссылается **WindowProc,** при проектировании новых классов компонентов

Метод	Объявление / Описание
DoEndDock	void DoEndDock(System::TObject* Target, int X, int Y) Вызывает обработчик события **OnEndDock**. Используется при проектировании новых классов
DoEndDrag	void **DoEndDrag(System::TObject*** Target, int X, int Y) Вызывает обработчик события **OnEndDrag**. Используется при проектировании новых классов
DoMouseWheel	**bool** DoMouse**Wheel(Classes::TShiftState** Shift, int **WheelDelta,** const **Types::TPoint** &MousePos) Автоматически вызывается при вращении колесика мыши. Если находится обработчик события **OnMouseWheel**, вызывает его и возвращает true. Иначе вызывает **DoMouseWheel**-Down или DoMouseWheel**Up**. Используется при проектировании новых классов
DoMouseWheel Down	bool **DoMouseWheelDown(Classes::TShiftState** Shift, const Types::TPoint &MousePos) Генерирует событие **OnMouseWheelDown**. Возвращает true, если находится обработчик этого события. Используется при проектировании новых классов
DoMouseWheel Up	bool **DoMouseWheelUp(Classes::TShiftState** Shift, const **Types::TPoint** &MousePos Генерирует событие OnMouseWheelUp. Возвращает **true**, если находится обработчик этого события. Используется при проектировании новых классов
DoStartDock	void **DoStartDock(TDragObject* &DragObject)** Вызывает обработчик события OnStartDock. Используется при проектировании новых классов
DoStartDrag	void **DoStartDrag(TDragObject* &DragObject)** Вызывает обработчик события OnStartDrag. Используется при проектировании новых классов
DragCanceled	void DragCanceled(void) Прерывает перетаскивание. Используется при проектировании новых классов
DragDrop	void **DragDrop(System::TObject*** Source, int X, int Y) Вызывает обработчик события OnDragDrop. X и Y — координаты мыши. Используется при проектировании новых классов
EndDrag	void EndDrag(bool Drop) Завершает успешно (при **Drop** = true) или неуспешно перетаскивание. Используется при проектировании новых классов
GetTextBuf	int **GetTextBuf(char** * Buffer, int BufSize) Записывает в заданный буфер Buffer фиксированного размера BufSize значение свойства Text. Для получения строки текста типа AnsiString надо вместо этого метода использовать само свойство Text

Метод	Объявление / Описание
GetTextLen	int GetTextLen(void) Возвращает длину строки свойства Text, необходимую для задания размера буфера в методе GetTextBuf
Hide	void **Hide(void)** Делает компонент невидимым
Invalidate	void Invalidate(void) Вызывает полную перерисовку испорченного изображения компонента
ManualDock	**bool ManualDock(TWinControl*** NewDockSite, TControl* DropControl = (TControl*) NULL, **TAlign ControlSide** = alNone) Изымает встроенный компонент из текущего контейнера и встраивает его в NewDockSite. DropControl — компонент внутри нового контейнера (если таковой есть), в который производится встраивание (например, страница в многостраничном контейнере). ControlSide определяет выравнивание компонента в контейнере
ManualFloat	bool ManualFloat(const **Types::TRect** &ScreenPos) Переводит встроенный компонент в состояние плавающего окна в область ScreenPos
MouseDown	void **MouseDown(TMouseButton** Button, Classes::TShiftState Shift, int **X,** int Y) Вызывает обработчик события **OnMouseDown.** Используется при проектировании новых классов
MouseMove	void **MouseMove(Classes::TShiftState** Shift, int X, int Y) Вызывает обработчик события **OnMouseMove.** Используется при проектировании новых классов
MouseUp	void **MouseUp(TMouseButton** Button, Classes::TShiftState Shift, int X, int Y) Вызывает обработчик события OnMouseUp. Используется при проектировании новых классов
ParentToClient	**Types::TPoint** ParentToClient(const Types::TPoint &Point, **TWinControl*** AParent = **(TWinControl*)** NULL) Переводит координаты точки Point из системы координат компонента AParent в систему координат данного компонента. Компонент AParent должен быть одним из родительских компонентов, который может быть найден по цепочке ссылок свойств Parent. Если AParent = NULL, подразумевается непосредственный родительский компонент
Perform	int **Perform(unsigned** Msg, int WParam, int **LParam)** Посылает компоненту сообщение Msg с параметрами WParam и LParam
Refresh	void Refresh(void) Немедленно перерисовывает компонент на экране, вызывая метод Repaint

Метод	Объявление / Описание
Repaint	void **Repaint(void)** Немедленно перерисовывает компонент на экране
ReplaceDocked Control	**bool ReplaceDockedControl(TControl*** Control, **TWinControl*** NewDockSite, **TControl* DropControl,** TAlign ControlSide) Встраивает данный компонент на место компонента Control, встроенного в контейнер NewDockSite. DropControl — компонент внутри контейнера NewDockSite (например, страница многостраничного компонента). ControlSide определяет выравнивание компонента
ScreenToClient	Types::TPoint ScreenToClient(const **Types::TPoint** &Point) Преобразует координаты Point экрана в координаты клиентской области компонента
SendCancel Mode	void **SendCancelMode(TControl*** Sender) Прерывает модальное состояние элемента
SendToBack	void SendToBack(void) Переносит компонент в Z-последовательности ниже других компонентов на той же форме
SetBounds	void SetBounds(int ALeft, int ATop, int **AWidth,** int **AHeight)** Задает сразу 4 свойства: Left, Top, Width и Height
SetTextBuf	void **SetTextBuf(char*** Buffer) Записывает в заданный буфер значение свойства Text. Используется, если нужна обратная совместимость с 16-битными кодами
Show	void Show(void) Делает видимым невидимый компонент
UpdateBounds Rect	void **UpdateBoundsRect(const** Types:**:TRect** &R) Изменяет, как и SetBounds, полное описание BoundsRect, но не перерисовывает изображение компонента на экране

Помимо перечисленных методов и многих других вспомогательных, класс **TControl** наследует методы своих классов-предков, в частности, **Assign, FindComponent, InsertComponent, RemoveComponent, ClassName, ClassNameIs, Free** и многие другие.

События

В классе **TControl,** в отличие от предшествующих ему в иерархии, описаны не только свойства и методы, но и следующие события:

Событие	Описание
OnCanResize	Событие при попытке изменить размеры компонента
OnClick	Событие при щелчке на компоненте и некоторых других действиях пользователя
OnConstrainedResize	Событие при изменении размеров компонента, наступающее после OnCanResize и позволяющее определить новые размеры

Событие	Описание
OnContextPopup	Событие при щелчке правой кнопкой мыши на компоненте с целью вызвать контекстное меню
OnDblClick	Событие при двойном щелчке на компоненте
OnDragDrop	Событие при отпускании перетаскиваемого компонента
OnDragOver	Событие при перетаскивании объекта над компонентом
OnEndDoc	Событие при окончании или прерывании перетаскивания и встраивания
OnEndDrag	Событие при окончании или прерывании перетаскивания
OnMouseDown	Событие при нажатии кнопки мыши над объектом
OnMouseMove	Событие при перемещении указателя мыши над объектом
OnMouseUp	Событие при отпускании нажатой кнопки мыши над объектом
OnMouseWheel	Событие при вращении колесика мыши
OnMouseWheelDown	Событие при вращении колесика мыши вниз
OnMouseWheelUp	Событие при вращении колесика мыши вверх
OnResize	Заключительное событие при изменении размеров компонента, наступающее после **OnCanResize** и **OnConstrainedResize**
OnStartDoc	Событие при начале перетаскивания и встраивания объекта
OnStartDrag	Событие при начале перетаскивания объекта

TControlState — тип

Тип определяет множество значений, характеризующих состояние компонента во время выполнения приложения.

Определение

```
enum Controls__01 {csLButtonDown, csClicked, csPalette,
                   csReadingState, csAlignmentNeeded,
                   csFocusing, csCreating, csPaintCopy,
                   csCustomPaint, csDestroyingHandle,
                   csDocking};
typedef Set<Controls__01, csLButtonDown, csDocking> TControlState;
```

Описание

Тип TControlState используется в свойстве **ControlState** для определения различных условий, действующих на данный экземпляр компонента, например, щелчок мыши или необходимость выравнивания компонента. Свойство может содержать следующие флаги:

csLButtonDown	Левая кнопка мыши нажата, но еще не освобождена
csClicked	То же самое, что csLButtonDown, но только в том случае, если свойство компонента **ControlStyle** содержит флаг csClickEvents, означающее, что событие, связанное с нажатием кнопки, интерпретируется как щелчок

csPalette	Компонентом или одним из его родителей получено сообщение WM_PALETTCHANGED
csReadingState	Компонент читает свое состояние из потока
csAlignmentNeeded	Компонент должен осуществить выравнивание
csFocusing	Приложение получило сообщение о переключении фокуса на данный компонент. Это не гарантирует, что компонент получит фокус, но позволяет предотвратить рекурсивные вызовы
csCreating	Создается данный компонент, или его владелец, или управляемый им компонент. Этот флаг очищается, когда создание компонента завершено
csPaintCopy	Компонент должен быть перекрашен. Это состояние возможно, если свойство ControlStyle содержит флаг csReplicatable
csCustomPaint	Компонент находится в состоянии заказной прорисовки
csDestroyingHandle	Оконный компонент должен быть уничтожен
csDocking	Компонент находится в процессе встраивания

TControlStyle – тип

Тип, характеризующий множество значений стиля компонента.

Определение

```
enum Controls__11 {csAcceptsControls, csCaptureMouse,
                   csDesignInteractive, csClickEvents,
                   csFramed, csSetCaption, csOpaque,
                   csDoubleClicks, csFixedWidth, csFixedHeight,
                   csNoDesignVisible, csReplicatable,
                   csNoStdEvents, csDisplayDragImage,
                   csReflector, csActionClient, csMenuEvents};
typedef Set<Controls__11, csAcceptsControls, csMenuEvents>
                                                TControlStyle;
```

Описание

Тип TControlStyle используется в свойстве **ControlStyle** для определения различных атрибутов компонента, например, может ли он быть захвачен мышью или имеет ли он фиксированные размеры. Множество **TControlStyle** может содержать следующие флаги:

csAcceptsControls	Компонент становится родителем любого компонента, перенесенного на него в процессе проектирования
csCaptureMouse	Захват компонента мышью при щелчке на нем
csDesignInteractive	Компонент устанавливает соответствие во время проектирования щелчка правой кнопки мыши щелчку левой кнопки для манипуляций с компонентом
csClickEvents	Компонент получает сообщение о щелчке мыши и реагирует на него
csFramed	Компонент имеет объемную рамку

csSetCaption	Компонент должен изменять надпись на нем в соответствии со свойством Name, если только надпись не задана явным образом
csOpaque	Компонент полностью заполняет свою клиентскую область
csDoubleClicks	Компонент получает сообщение о двойном щелчке мыши и реагирует на него. Если флаг не установлен, то двойной щелчок интерпретируется как просто щелчок
csFixedWidth	Ширина компонента не меняется и не масштабируется
csFixedHeight	Высота компонента не меняется и не масштабируется
csNoDesignVisible	Компонент невидим во время проектирования
csReplicatable	Компонент может копироваться методом **PaintTo** для прорисовки произвольной канве
csNoStdEvents	Игнорируются стандартные события, такие, как нажатие кнопок мыши, клавиш, щелчки. Этот флаг надо устанавливать, если ваш код не должен реагировать на эти события; в результате ваше приложение будет выполняться быстрее
csDisplayDragImage	Компонент может отображать изображение из списка изображений, когда мышь перемещается на него. Этот флаг устанавливается, если компонент реализует список изображений для отображения при перемещении на него мыши
csReflector	Компонент реагирует на сообщения Windows, поступающие из диалогов, сообщения о фокусировке, сообщения об изменении размеров. Этот флаг устанавливается, если компонент может использоваться как элемент ActiveX и должен реагировать на эти события
csActionClient	Компонент связан с объектом действия. Флаг устанавливается при задании свойства Action и очищается при очистке Action
csMenuEvents	Компонент реагирует на команды системного меню

Конструктор класса **TControl** инициализирует свойство **ControlStyle** значениями **[csCaptureMouse, csClickEvents, csSetCaption, csDoubleClicks].**

TCursor — тип

Определяет изображение курсора мыши.

Определение

```
enum TCursor {crMin=0x7fff-1, crMax=0x7fff};
```

Описание

Тип TCursor используется для выбора вида изображения курсора мыши в таких свойствах, как **Cursor, DragCursor** и некоторых других. Значения **TCursor** являются индексом списка возможных курсоров, управляемого глобальной переменной Screen. Ниже приводится перечень встроенных в **TScreen** типов курсоров:

Константа	Значение	Вид	Константа	Значение	Вид
crDefault	0	Курсор по умолчанию. Обычно это crArrow.	crDrag	-12	
crNone	-1		crNoDrop	-13	
crArrow	-2		crHSplit	-14	
crCross	-3		crVSplit	-15	
crIBeam	-4		crMultiDrag	-16	
crSizeNESW	-6		crSQLWait	-17	
crSizeNS	-7		crNo	-18	
crSizeNWSE	-8		crAppStart	-19	
crSizeWE	-9		crHelp	-20	
crUpArrow	-10		crHandPoint	-21	
crHourGlass	-11		crSize — устарело crSizeAll	-22	

TCustomClientDataSet — базовый класс клиентских наборов данных

Клиентский набор данных.

Иерархия *TObject — TPersistent — TComponent — TDataSet*

Модуль *DBClient*.

Описание

Класс **TCustomClientDataSet** является базовым для таких компонентов клиентских наборов данных, как **TClientDataSet, TBDEClientDataSet, TSQLClientDataSet. В TCustomClientDataSet** реализованы свойства и методы, позволяющие создавать наборы данных, хранящие данные в памяти *в* виде пакетов транспортного формата. Для осуществления своих функций объекты **TCustomClientDataSet** используют возможности технологии Midas, реализованные *в* библиотеке *Midas.dll.*

Объекты класса **TCustomClientDataSet** создавать нельзя. Класс используется только как базовый для своих наследников **TClientDataSet, TBDEClientDataSet, TSQLClientDataSet** и других, а также для создания новых классов клиентских наборов данных.

Клиентские наборы данных, реализуемые компонентами **TBDEClientDataSet, TClientDataSet** и другими наследниками класса **TCustomClientDataSet,** обладают всеми свойствами и методами, наследуемыми ими от класса **TDataSet,** и многими возможностями, присущими наборам данных, связанным с таблицами. Иначе говоря, упорядочивание данных с помощью индексов, задание ограничений, созда-

ние и модификация таблиц может осуществляться в клиентских наборах данных так же, как и в других наследниках **TDataSet.** Но во всех этих операциях добавляется немало новых возможностей.

Особенность клиентских наборов данных заключается в том, что данные хранятся в памяти и могут сохраняться в файле на диске и читаться из этого файла. Таким образом, клиентские наборы данных могут выступать как автономные основанные на файлах MyBase наборы данных в однопоточных приложениях. Другая немаловажная функция клиентских наборов — создание так называемой «портфельной» базы данных, в которую первоначально записываются данные сервера или какой-то иной базы данных, а затем вся работа проходит с файлом на компьютере клиента. Пользователь может изменять эти данные, причем при закрытии клиентского набора все изменения запоминаются в файле. А при открытии набора данные из файла записываются в клиентский набор. Таким образом, «портфельная» база данных — альтернатива кэширования. Преимуществом является то, что изменения могут производиться не обязательно в одном сеансе работы, а в нескольких. И когда пользователь решит, что изменения заслуживают пересылки в базу данных, он может занести в нее все изменения, сделанные на протяжении этих сеансов.

Клиентские наборы данных обладают еще одной полезной особенностью. С их помощью наиболее просто осуществлять взаимную трансляцию таблиц баз данных, созданных в разных СУБД и работающих на разных платформах.

Основные области применения клиентских наборов данных: автономные наборы данных, портфельные наборы данных, интерфейс к другим наборам данных. Кроме того, клиентские наборы данных предоставляют дополнительные возможности при индексации и фильтрации данных и обеспечивают расчет совокупных характеристик по всем или указанному подмножеству записей набора. Рассмотрение всех этих областей применения и особенностей клиентских наборов выходит за рамки данной книги. Эти вопросы подробно рассмотрены в источнике [3].

Свойства

Основные свойства класса **TCustomClientDataSet:**

Свойство	Объявление / Описание
Aggregates	TAggregates* Aggregates Коллекция всех совокупных характеристик, поддерживаемых данным набором данных
AggregatesActive	**bool** AggregatesActive Определяет, управляет ли набор совокупными характеристиками. Для расчета совокупных характеристик надо установить true
CommandText	**AnsiString** CommandText Запрос SQL, или таблица, или хранимая процедура, определяющие соединение набора данных
ConnectionBroker	**TConnectionBroker*** ConnectionBroker Указывает брокера, обрабатывающего соединение на сервере приложений
Data	**System::OleVariant** Data Данные клиентского набора в транспортном формате
Delta	System::OleVariant Delta Пакет изменений в данных, еще не зафиксированных в базе данных

Свойство	Объявление / Описание
Filter	property Filter: string; Строка фильтра набора данных (намного шире, чем в родительском классе TDataSet)
GroupingLevel	int GroupingLevel Указывает максимальную глубину группирования, поддерживаемую текущим индексом
IndexDefs	Db::TIndexDefs* IndexDefs Коллекция определений индексов набора
IndexFieldNames	AnsiString IndexFieldNames Список имен индексов, разделенных точками с запятой
IndexName	AnsiString IndexName Имя текущего индекса
LogChanges	bool LogChanges Определяет, хранятся ли изменения в свойстве Delta, отдельно от Data. Задавать значение false может иметь смысл только в автономных наборах данных для повышения производительности
MasterFields	AnsiString MasterFields Во вспомогательной таблице определяет ключевые поля головной таблицы, используемые для связи со вспомогательной таблицей
MasterSource	Db::TDataSource* MasterSource Во вспомогательной таблице определяет источник данных головной таблицы
ProviderName	AnsiString ProviderName Имя провайдера, снабжающего набор данными
RemoteServer	TCustomRemoteServer* RemoteServer Указывает компонент, используемый набором для связи с сервером приложений
SavePoint	int SavePoint Указывает точку текущего состояния результатов редактирования. Позволяет вернуться к предыдущему состоянию

Кроме приведенных свойств, класс наследует множество свойств своего родительского класса **TDataSet: Active. AggFields, AutoCalcFields. BlockReadSize. Bof. Bookmark. CanModify. DataSource. Eof. FieldCount. FieldDefs. Fields. FieldValues. Filtered. FilterOptions. Found, Modified, Name, RecordCount, SparseArrays, State.** В классе имеется еще немало свойств, наследуемых от TComponent.

Методы

Ниже приведен список основных методов, определенных в **TCustomClientDataSet.**

Метод	Объявление / Описание
AddIndex	void AddIndex(const AnsiString Name, const AnsiString Fields, Db::TIndexOptions Options, const AnsiString DescFields = "", const AnsiString CaseInsFields = "", const int GroupingLevel = 0) Формирует новый индекс с заданными условиями сортировки
AppendData	void AppendData(const System::OleVariant &Data, bool HitEOF) Добавляет новый пакет записей к уже существующим в клиентском наборе
ApplyUpdates	int ApplyUpdates(int MaxErrors) Отправляет измененные данные провайдеру для записи в базу данных
CancelUpdates	void CancelUpdates(void) Отменяет все изменения, отображенные в свойстве Delta
CreateData Set	void CreateDataSet(void) Создает новый автономный набор данных
DeleteIndex	void DeleteIndex(const AnsiString Name) Удаляет индекс с заданным именем
EmptyDataSet	void EmptyDataSet(void) Удаляет все записи из набора данных
GetBookmark	function GetBookmark: TBookmark; Создает и возвращает новую закладку набора данных
GetIndexNames	void GetIndexNames(Classes::TStrings* List) Возвращает список имен индексов
GetNextPacket	int GetNextPacket(void) Обеспечивает получение следующего пакета от провайдера с числом записей, не превышающим PacketRecords
LoadFromFile	void LoadFromFile(const AnsiString FileName = "") Загружает клиентский набор данных из указанного файла
RefreshRecord	void RefreshRecord(void) Обновляет текущую запись, перенося ее из базы данных
RevertRecord	void RevertRecord(void) Отменяет исправления текущей записи
SaveToFile	void SaveToFile(const AnsiString FileName = "", TDataPacketFormat Format = dfBinary) Сохраняет набор данных в файле
UndoLastChange	bool UndoLastChange(bool FollowChange) Удаляет результаты последней операции редактирования

Кроме приведенных методов, класс наследует множество методов своего родительского класса TDataSet: Append. AppendRecord, BookmarkValid. Cancel. ClearFields, Close. CompareBookmarks, ControlsDisabled, CreateBlobStream, Delete. DisableControls. Edit. EnablcControls. FieldByName. FindField, FindFirst, FindLast. FindNext. FindPrior. First. FreeBookmark. GetBookmark. GetDetailDataSets. GetDetailLinkFields, GetFieldNames, GotoBookmark, Insert. InsertRecord. IsEmpty, Last. Locate. Lookup. MoveBy. Next. Open. Post. Prior. Refresh. SetFields. Translate. UpdateStatus и ряд других.

События

Ниже приведен список основных событий, определенных в TCustomClientDataSet.

Событие	Описание
AfterApplyUpdates	Наступает после выполнения метода AppIyUpdates
AfterExecute	Наступает после выполнения метода Execute
AfterGetParams	Наступает после выполнения метода **FetchParams**
AfterGetRecords	Наступает после пересылки набору данных, затребованных от провайдера. Может использоваться для внесения в пакет каких-то нестандартных данных
AfterRowRequest	Наступает после пересылки набору данных новой информации о текущей записи
BeforeApplyUpdates	Наступает перед выполнением метода AppIyUpdates
BeforeExecute	Наступает перед выполнением метода Execute
BeforeGetParams	Наступает перед выполнением метода FetchParams
BeforeGetRecords	Наступает перед пересылкой набору данных, затребованных от провайдера
BeforeRowRequest	Наступает перед пересылкой набору данных новой информации о текущей записи
OnReconcileError	Наступает для каждой записи, вызвавшей ошибку при занесении в базу данных. Используется для принятия пользователем решения по разрешению возникшего конфликта

TCustomEdit — базовый класс окон редактирования

Абстрактный базовый класс окон редактирования.

. Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl*

Модуль *stdctrls*.

Описание

Класс TCustomEdit является базовым абстрактным классом для окон редактирования Edit и Memo. В нем инкапсулированы основные методы и свойства, используемые при редактировании текстов. Они обеспечивают:

- Такие функции редактирования текста, как выделение фрагмента, преобразования выделенного текста, чувствительность к регистру.
- Возможность откликаться на изменения в тексте.
- Управление доступом к тексту, например, доступ «только для чтения» или символы пароля, делающие невидимыми вводимые символы.

Создавать экземпляры объектов типа **TCustomEdit** невозможно. Этот класс используется только для создания производных классов, наследующих особенности обработки текстов.

Свойства

Основные свойства класса **TCustomEdit:**

Свойство	**Объявление** / Описание
AutoSelect	**bool** AutoSelect Определяет, выделяется ли весь текст при получении элементом фокуса
AutoSize	bool AutoSize; Определяет, будет ли высота элемента автоматически адаптироваться к размеру символов текста
CanUndo	bool CanUndo Указывает, были ли произведены в тексте операции редактирования, которые могут быть удалены командой Undo. Может использоваться для регулирования доступа к соответствующему разделу меню. Свойство только для чтения
CharCase	enum TEditCharCase { ecNormal, **ecUpperCase, ecLowerCase** }; **TEditCharCase** CharCase Определяет регистр отображаемого текста: ecNormal — все символы отображаются с учетом регистра, в котором был введен каждый из них, ecUpperCase — все символы текста приводятся к верхнему регистру, ecLowerCase — все символы текста приводятся к нижнему регистру
HideSelection	bool HideSelection При значении true делает невидимым выделение части текста, когда компонент теряет фокус. В противном случае выделение остается и при потере фокуса
MaxLength	**int** MaxLength Устанавливает максимальную длину текста. Значение 0 указывает на неограниченную длину.
Modified	bool Modified Указывает, был ли изменен текст
PasswordChar	char PasswordChar Указывает символ, появляющийся в окне редактирования вместо вводимого пользователем символа. При значении 0 отображается вводимый символ, в противном случае — заданный символ ввода пароля
Readonly	bool Readonly Устанавливает запрет редактирования текста пользователем
SelLength	**int** SelLength Указывает длину выделенного фрагмента текста
SelStart	**int** SelStart Указывает первый символ выделенного фрагмента текста. Если выделения нет, то указывает символ, перед которым расположен курсор

Свойство	Объявление / Описание
SelText	AnsiString SelText Содержит выделенный фрагмент текста. Можно задавать значение SelText, чтобы заменить им выделенный текст. Если выделения нет, то задание SelText приведет к вставке заданного текста в позицию курсора (SelStart)

Кроме того, класс наследует много свойств от родительских классов, в частности, **Action, Align, Anchors, Brush, ClientRect, Constraints, Cursor, Enabled, Hint, TabOrder, TabStop, Visible** и много других.

Методы

Ниже приводится таблица основных методов класса **TCustomEdit.**

Метод	Объявление / Описание
Clear	void Clear(void) Удаление всего текста
ClearSelection	void ClearSelection(void) Удаление выделенного текста
ClearUndo	void ClearUndo(void) Очистка буфера, используемого для команды Undo. После этого сделанные изменения в тексте не могут быть отменены
CopyToClipboard	void CopyToClipboard(void) Копирование в буфер Clipboard выделенного текста
CutToClipboard	void CutToClipboard(void) Вырезание в буфер Clipboard выделенного текста
PasteFromClipboard	void PasteFromClipboard(void) Вставка текста из буфера Clipboard вместо выделенного текста или, если нет выделения, то вставка текста в позицию курсора
SelectAll	void SelectAll(void) Выделение всего текста
Undo	void Undo(void) Отмена результатов последней операции редактирования

События

В классе **TCustomEdit** определено только одно событие:

Событие	Описание
OnChange	Событие при попытке изменения пользователем текста. Произошло ли изменение в действительности, можно определить по значению свойства Modified.

TDataSet — базовый класс всех наборов данных

Базовый класс всех компонентов наборов данных, представляющих данные в виде строк и столбцов.

Иерархия *TObject — TPersistent — TComponent*

Модуль *db.*

Описание

Класс **TDataSet** инкапсулирует основные свойства, методы и события, используемые при работе с базами данных. Многие из этих свойств и событий абстрактные и реализуются в наследниках класса **TDataSet,** таких, как **TClientDataSet, TBDEData, TDBDataSet, TQuery, TStoredProc и TTable.**

Свойства

Ниже приведен список основных свойств, определенных в **TDataSet.**

Свойство	Объявление / Описание
Active	**bool** Active Определяет открытие и закрытие набора данных
AggFields	TFields* AggFields Содержит массив полей совокупных (агрегированных) характеристик
AutoCalcFields	bool AutoCalcFields Управляет генерацией событий OnCalcField
BIockReadSize	int BIockReadSize Число записей, помещаемых при чтении в буфер
Bof	bool Bof Указывает, находится ли курсор на первой записи
Bookmark	AnsiString Bookmark Определяет текущую закладку набора данных
CanModifv	bool CanModify Определяет, возможно ли редактирование набора данных
Constraints	TCheckConstraints* Constraints Ограничения на допустимые значения параметров на уровне записи
DataSource	TDataSource* DataSource Указывает источник данных другого набора данных, снабжающего информацией данный набор
Eof	bool **Eof** Указывает, находится ли курсор на последней записи
FieldCount	int FieldCount Число объектов полей, связанных с набором данных
FieldDefs	TFieldDefs* FieldDefs Указывает на список объектов, определяющих поля набора данных

Свойство	Объявление / Описание
Fields	TFields* Fields Список всех объектов полей (кроме полей совокупных характеристик) набора данных. Свойство только для чтения
FieldValues	System::Variant FieldValues[AnsiString FieldName] Обеспечивает доступ к значениям всех полей активной записи
Filter	AnsiString Filter Строка фильтра набора данных
Filtered	bool Filtered Разрешает или запрещает фильтрацию
FilterOptions	enum TFilterOption { foCaseInsensitive, foNoPartialCompare }; typedefSet<TFilterOption, foCaseInsensitive, foNoPartialCompare> TFilterOptions; TFilterOptions FilterOptions Опции фильтрации
Found	bool Found Булево свойство, показывающее успешность перемещения по набору данных методами FindFirst, FindLast, FindNext, FindPrior. Только для чтения
Modified	bool Modified Указывает, была ли модифицирована активная запись. Только для чтения
Name	AnsiString Name: Имя набора данных. По умолчанию совпадает с именем соответствующего компонента (Tablel, Query2 и т.п.). При изменении Name соответственно изменяются и использующие его имена полей (Table1Fam, Query2Nam и т.п.)
RecordCount	int RecordCount Количество записей в наборе данных. В TDataSet всегда = -1. Так что читать это свойство можно только на уровне классов-наследников. Только для чтения
SparseArrays	bool SparseArrays Указывает, создаются ли уникальные объекты полей для каждого элемента массива полей. Только для чтения
State	TDataSetState State Состояние массива данных. Только для чтения

Кроме приведенных свойств в классе имеется еще немало свойств, наследуемых от **TComponent.**

Методы

Ниже приведен список основных методов, определенных в **TDataSet.**

Метод	**Объявление / Описание**
Append	**void Append(void)** **Добавляет новую пустую запись в конец набора данных**

Метод	Объявление / Описание
AppendRecord	void AppendRecord(const System::TVarRec * Values, const int Values_Size) Добавляет в набор данных новую запись, заполняет ее и пересылает в базу данных
BookmarkValid	**bool BookmarkValid(void *** Bookmark) Определяет, присвоено ли значение указанной закладке Bookmark набора данных. Возвращает false, если не присвоено (в частности, если набор данных не поддерживает закладки)
Cancel	void Cancel(void) Отменяет результаты редактирования
ClearFields	void ClearFields(void) Очищает содержимое всех полей активной записи набора данных
Close	void Close(void) Закрывает набор данных
Compare Bookmarks	int **CompareBookmarks(void** * **Bookmark1,** void * Bookmark2) Сравнивает две закладки Bookmark набора данных
Controls **Disabled**	bool ControlsDisabled(void) Проверяет наличие блокировки методом DisableControls компонентов, отображающих данные
CreateBlob Stream	Classes::TStream* **CreateBlobStream(TField*** Field, TBlobStreamMode Mode) Создает поток для чтения и записи данных поля BLOB
Delete	void **Delete(void)** Удаляет активную запись и позиционирует курсор на следующую запись
Disable Controls	void DisableControls(void) Блокирует вывод данных в компоненты, связанные с данными
Edit	void Edit(void) Перевод набора данных в режим редактирования
EnableControls	void EnableControls(void) Снимает блокировку отображения данных, введенную ранее методом DisableControls
FieldByName	TField* FieldByName(const AnsiString FieldName) Находит поле по его имени. При неверном имени генерирует исключение
FindField	TField* FindField(const AnsiString FieldName) Находит поле по его имени. При неверном имени возвращает NULL
FindFirst	bool FindFirst(void) Перемещает курсор к первой записи и возвращает true в случае успеха

Метод	Объявление / Описание
FindLast	bool FindLast(void) Перемещает курсор к последней записи и возвращает true в случае успеха
FindNext	bool FindNext(void) Перемещает курсор к следующей записи и возвращает true в случае успеха
FindPrior	bool FindPrior(void) Перемещает курсор к предыдущей записи и возвращает true в случае успеха
First	void First(void) Перемещает курсор к первой записи
FreeBookmark	void FreeBookmark(void * Bookmark) Уничтожает закладку набора данных
GetBookmark	void * GetBookmark(void) Создает и возвращает новую закладку набора данных
GetDetailDataSets	procedure GetDetailDataSets(List: TList); Заполняет список данными о вспомогательном наборе данных, находящемся с данным набором в соотношении master/detail
GetDetailLinkFields	void GetDetailDataSets(Classes::TList* List) Во вспомогательном наборе выдает списки ключевых полей этого и головного наборов данных
GetFieldNames	void GetFieldNames(Classes::TStrings* List) Выдает список имен всех полей набора данных
GotoBookmark	void GotoBookmark(void * Bookmark) Осуществляет переход на указанную закладку набора данных
Insert	HIDESBASE void Insert(void) Вставляет новую пустую запись в набор данных
InsertRecord	void InsertRecord(const System::TVarRec * Values, const int Values_Size) Вставляет новую заполненную запись в набор данных
IsEmpty	bool IsEmpty(void) Возвращает true, если набор данных пуст, т.е. не содержит ни одной записи
Last	void Last(void) Перемещает курсор к последней записи
Locate	bool Locate(const AnsiString KeyFields, const System::Variant &KeyValues, TLocateOptions Options) Осуществляет поиск записи в наборе данных

Метод	Объявление / Описание
Lookup	System::Variant Lookup(const AnsiString KeyFields, const Variant &KeyValues, const AnsiString ResultFields); Осуществляет поиск записи в наборе данных и возвращает значения указанных полей этой записи
MoveBy	int MoveBy(int Distance) Перемещает курсор на заданное число записей
Next	void Next(void) Перемещает курсор к следующей записи
Open	void Open(void) Открывает набор данных
Post	void Post(void) Пересылает отредактированную запись в базу данных
Prior	void Prior(void) Перемещает курсор к предыдущей записи
Refresh	void Refresh(void) Обеспечивает обновление отображаемых данных
SetFields	void SetFields(const System::TVarRec * Values, const int Values_Size) Устанавливает значения всех полей записи
Translate	void Translate(char * Src, char * Dest, bool ToOem) Взаимно преобразует строку символов ANSI (используются в VCL) и строку символов OEM (используются в BDE)
UpdateStatus	TUpdateStatus UpdateStatus(void) Определяет состояние обновления текущей записи

События

Ниже приведен список основных событий, определенных в **TDataSet.**

Событие	Описание
AfterCancel	Наступает после выполнения метода Cancel
AfterClose	Наступает после закрытия набора данных
AfterDelete	Наступает после удаления записи методом Delete
AfterEdit	Наступает сразу после перехода набора данных в режим редактирования
AfterInsert	Наступает сразу после вставки записи в набор данных
AfterOpen	Наступает сразу после открытия набора данных
AfterPost	Наступает сразу после пересылки измененной записи в базу данных или буфер кэширования
AfterRefresh	Наступает сразу после обновления отображаемых данных методом Refresh
AfterScroll	Наступает сразу после перемещения на новую запись
BeforeCancel	Наступает в начале выполнения метода Cancel

Событие	Описание
BeforeClose	Наступает перед закрытием набора данных
BeforeDelete	Наступает перед удалением записи методом Delete
BeforeEdit	Наступает перед началом перехода набора данных в режим редактирования
BeforeInsert	Наступает перед переходом набора данных в режим вставки записи
BeforeOpen	Наступает перед открытием набора данных
BeforePost	Наступает перед пересылкой методом Post изменений в текущей записи в базу данных или буфер кэширования
BeforeRefresh	Наступает перед обновлением отображаемых данных методом Refresh
BeforeScroll	Наступает перед перемещением на новую запись
OnCalcFields	Наступает, когда требуется пересчет вычисляемых полей
OnDeleteError	Наступает, если в процессе удаления записи генерируется исключение
OnEditError	Наступает при генерации исключения в процессе редактирования или вставки записи
OnFilterRecord	Наступает каждый раз при перемещении на другую запись, если разрешена фильтрация
OnNewRecord	Наступает при вставке в набор данных новой записи
OnPostError	Наступает при генерации исключения в процессе пересылки в базу данных измененной записи

TDateTime — класс

Тип, используемый функциями и процедурами, работающими с датами и временем.

Заголовочный файл *systdate.h*.

Описание

Тип **TDateTime** был введен в Object Pascal как число с плавающей запятой, целая часть которого содержит число дней, отсчитанное от некоторого начала календаря, а дробная часть равна части 24-часового дня, т.е. характеризует время и не относится к дате. Для 32-разрядных версий за начало календаря принята дата 00 часов 30 декабря 1899 года. Прибавление к значению типа **TDateTime** целого числа D равносильно увеличению даты на D дней. Разность двух значений типа **TDateTime** дает разность двух дат с точностью до долей дня.

В C++Builder тип **TDateTime** реализован в виде класса. Впрочем, его можно использовать точно так же, как в Object Pascal. Но в действительности, возможности класса **TDateTime** шире. В частности, можно использовать конструктор, инициирующий переменную заданным значением **TDateTime.** Например, оператор

```
TDateTime T(Now());
```

объявляет переменную T и передает в нее текущую дату и время с помощью функции **Now.**

В классе определен ряд операций: "+" и " - сложение и вычитание числа дней, включая дробную часть дня, "++" и " - прибавление и вычитание од-

ного дня, **double** — перевод в форму действительного числа, типичную для Delphi, операции отношения и ряд других.

Можно также использовать ряд полезных функций-элементов данного класса:

Функция-элемент	Объявление / Описание
CurrentDate	TDateTime CurrentDate() Возвращает текущую дату с нулевым временем
CurrentDateTime	TDateTime CurrentDateTime() Возвращает текущую дату и время
CurrentTime	TDateTime CurrentTime() Возвращает текущее время с нулевой датой
DateString	AnsiString DateString() const Возвращает дату объекта TDateTime в виде строки, отформатированной в соответствии с глобальной переменной ShortDateFormat
DateTimeString	AnsiString DateTimeString() const Возвращает дату и время объекта TDateTime в виде строки. Дата форматируется в соответствии с глобальной переменной ShortDateFormat. Время форматируется в соответствии с глобальной переменной LongTimeFormat
DayOfWeek	int DayOfWeek() const Возвращает день недели объекта TDateTime (1 — воскресенье, 7 — суббота)
DecodeDate	void DecodeDate(unsigned short* year, unsigned short* month, unsigned short* day) const Выделяет год year, месяц month и день day из объекта TDateTime
DecodeTime	void DecodeTime(unsigned short* hour, unsigned short* min, unsigned short* sec, unsigned short* msec) const Выделяет час hour, минуту min, секунду sec и миллисекунды msec из объекта TDateTime
FileDate	int FileDate() const Возвращает объект TDateTime, переведенный в формат дат и времени DOS
FileDateToDateTime	TDateTime FileDateToDateTime(int fileDate) Переводит в объект TDateTime дату и время fileDate, заданные в формате DOS
FormatString	AnsiString FormatString(const AnsiString& format) Возвращает строку объекта TDateTime, сформированную по строке форматирования format
TimeString	AnsiString TimeString() const Возвращает строку, содержащую время, записанное в объекте, отформатированное с помощью глобальной переменной LongTimeFormat

Примеры

```
Label1->Caption = T;                    // 26.05.2002 18:44:07
Label2->Caption = T.DateTimeString();
Label3->Caption = T.DateString();       // 26.05.2002
Label4->Caption = T.TimeString();       //18:44:07
```

Два первых оператора отобразят одно и то же: дату и время, в виде, показанном в комментарии к первому из них. Третий и четвертый операторы отобразят соответственно дату и время, вид которых указан в комментариях.

TDBDataSet — класс компонентов наборов данных

Базовый класс компонентов наборов данных.

Иерархия *TObject — TPersistent — TComponent — TDataSet — TBDEDataSet*

Модуль *dbtables.*

Описание

Класс **TDBDataSet** является базовым для таких компонентов наборов данных, как **TTable, TQuery, TStoredProc.** При разработке новых классов компонентов базовым для них также целесообразно выбирать **TDBDataSet.** Он наследует свойства и методы своих предшественников — **TDataSet** и **TBDEDataSet.** добавляя к ним свойства соединения с базами данных.

Свойства

Класс наследует все свойства **TBDEDataSet.** добавляя к ним следующие:

Свойство	Объявление / Описание
AutoRefresh	**bool** AutoRefresh Определяет, обновляются ли автоматически значения полей, вычисляемые сервером
Database	TDatabase Database Указывает компонент базы данных Database, связанный с набором данных. Свойство только для чтения
DatabaseName	AnsiString DatabaseName Указывает имя базы данных, с которой связан набор данных
DBSession	TSession* DBSession Связанный с набором данных компонент Session
SessionName	AnsiString SessionName Имя компонента Session, связанного с набором данных

Методы

Класс наследует все методы **TBDEDataSet.** добавляя к ним следующие:

Метод	**Объявление / Описание**
CheckOpen	**bool CheckOpen(Word Status)** **Проверяет, отсутствует ли ошибка при попытке доступа к данным**
CloseDatabase	**void CloseDatabase(TDatabase* Database)** **Закрывает базу данных**
OpenDatabase	**TDatabase* OpenDatabase(void)** **Открывает базу данных**

События

Все события класс наследует от своих предшественников: **TBDEDataSet и TDataSet.**

TField — базовый класс всех объектов полей

Базовый класс всех объектов полей наборов данных.

Иерархия *TObject — TPersistent — TComponent*

Модуль ***db.***

Описание

Класс **TField** описывает свойства, методы, события общие для всех объектов полей наборов данных:

- Изменение значений полей
- Взаимные преобразования значений полей различных типов
- Ограничения вводимых значений полей
- Отображение данных полей
- Расчет значений вычисляемых полей в обработчике события **OnCalcFields**
- Поля просмотра

Объекты типа **TField** не используются. Реально генерируются объекты многочисленных потомков класса **TField:**

TADTField	TDateField	TReferenceField
TAggregateField	TDateTimeField	**TSmallIntField**
TArrayField	TFloatField	**TStringField**
TAutoIncField	TGraphicField	TTimeField
TBCDField	TGuidField	**TVarBytesField**
TBlobField	TIDispatchField	**TVariantField**
TBooleanField	TIntegerField	**TWideStringField**
TBytesField	**TLargeIntField**	**TWordField**
TCurrencyField	**TMemoField**	

Кроме того, имеется ряд абстрактных классов — наследников **TField: TBinaryField, TDataSetField, TInterfaceField, TNumericField, TObjectField.**

Свойства

Свойство	Объявление / Описание
Alignment	Classes::TAlignment Alignment Определяет выравнивание значений полей при их отображении в компонентах, связанных с данными (только taLeftJustify, taRightJustify, **taCenter**)
AsBoolean	**bool** AsBoolean Переводит тип любого поля в булево значение при чтении значения поля и осуществляет обратный перевод булева значения в тип поля при записи
AsCurrency	**System::Currency AsCurrency** Переводит тип любого поля в Currency при чтении значения поля и осуществляет обратный перевод значения Currency в тип поля при записи
AsDateTime	System::TDateTime AsDateTime Переводит тип любого поля в TDateTime при чтении значения поля и осуществляет обратный перевод значения TDateTime в тип поля при записи

Свойство	Объявление / Описание
AsFloat	double AsFloat Переводит тип любого поля в действительное число при чтении значения поля и осуществляет обратный перевод действительного числа в тип поля при записи
AsInteger	int AsInteger Переводит тип любого поля в целое при чтении значения поля и осуществляет обратный перевод целого значения в тип поля при записи
AsString	AnsiString AsString Переводит тип любого поля в строку при чтении значения поля и осуществляет обратный перевод строки в тип поля при записи значения поля
AsVariant	System::Variant AsVariant Переводит тип любого поля в Variant при чтении значения поля и осуществляет обратный перевод значения Variant в тип поля при записи
AttributeSet	AnsiString AttributeSet Имя множества атрибутов словаря данных, применяемого к данному полю
AutoGenerate Value	enum TAutoRefreshFlag {arNone, arAutoInc, arDefault}; TAutoRefreshFlag AutoGenerateValue Показывает, может ли значение поля генерироваться автоматически в базе данных на сервере
Calculated	bool Calculated Определяет, является ли поле вычисляемым в обработчике события OnCalcFields
CanModify	bool CanModify Определяет, может ли данное поле модифицироваться
Constraint ErrorMessage	AnsiString ConstraintErrorMessage Определяет строку сообщения появляющуюся при вводе пользователем значения, нарушающего ограничение, установленное сервером или свойством CustomConstraint
CurValue	System::Variant CurValue Текущее значение поля с учетом изменений, внесенных другими пользователями
Custom Constraint	AnsiString CustomConstraint Строка SQL, накладывающая ограничение на вводимое пользователем значение поля
DataSet	TDataSet* DataSet Набор данных TDataSet, к которому относится объект поля. Значение DataSet устанавливается автоматически и не должно задаваться явно
DataSize	int DataSize Размер в байтах, необходимый для хранения значения поля. Только для чтения

Свойство	Объявление / Описание
DataType	**TFieldType** DataTvpe: Определяет тип **TFieldType** данных, хранящихся в поле
Default Expression	AnsiString **DefaultExpression** Выражение SQL, которое присваивает значение поля, если пользователь не задал никакого значения
DisplavLabel	AnsiString DisplayLabel Строка, отображаемая как заголовок столбца в таблице данных
DisplavName	AnsiString DisplavName Строка, отображаемая как заголовок столбца в таблице данных. Только для чтения
DisplayText	AnsiString DisplavText Строка, отображающая значение поля в компонентах, связанных с данными. Только для чтения
DisplayWidth	int DisplayWidth Число символов, которые отображаются в компоненте, связанном с данными
EditMask	typedef AnsiString TEditMask; TEditMask EditMask Маска, используемая для ввода данных
EditMaskPtr	typedef AnsiString TEditMask; TEditMask EditMaskPtr Маска, используемая для ввода данных. Только для чтения
FieldKind	enum TFieldKind **{fkData,** fkCalculated, **fkLookup, fkInternalCalc, fkAggregate };** TFieldKind FieldKind Указывает, является ли поле обычным полем базы данных, или вычисляемым полем, полем просмотра, вычисляемым и хранимым в базе данных, полем совокупной характеристики
FieldName	AnsiString FieldName Имя поля набора данных
FieldNo	int FieldNo Порядковый номер поля в таблице базы данных. Номера начинаются с 1. Могут не совпадать с индексом в массиве Fields набора данных. Свойство используется только при прямом обращении к API Borland Database Engine
FullName	AnsiString FullName Полное имя с перечислением имен родительских полей (если они имеются). Только для чтения
Has Constraints	**bool** HasConstraints Указывает, имеются ли ограничения на вводимые значения поля. Только для чтения
Imported Constraint	AnsiString ImportedConstraint Строка SQL, накладывающая ограничение со стороны сервера на значение поля

Свойство	Объявление / Описание
Index	int Index Индекс объекта поля в свойстве Fields набора данных
IsIndexField	bool IsIndexField Указывает, является ли поле полем индекса. Только для чтения
IsNull	bool IsNull Указывает, отсутствует ли у поля значение (при true), или какое-то значение присвоено. Только для чтения
KeyFields	AnsiString KeyFields Строка, содержащая список ключевых полей набора данных, используемый в поле просмотра
Lookup	bool Lookup Указывает, является ли данное поле полем просмотра
LookupCache	bool LookupCache Определяет должны ли значения поля просмотра кэшироваться, или просмотр должен осуществляться при каждом изменении текущей записи
LookupDataSet	TDataSet* LookupDataSet Набор данных, который просматривается при определении значения поля просмотра
LookupKevFields	AnsiString LookupKeyFields Строка, содержащая список ключевых полей набора данных LookupDataSet, используемый в поле просмотра
LookupList	TLookupList* LookupList Кэш значений набора данных LookupDataSet, индексированных множеством полей KeyFields. Только для чтения
LookupResultField	AnsiString LookupResultField Поле набора данных LookupDataSet, значение которого переносится в поле просмотра
NewValue	System::Variant NewValue Текущее (новое) значение поля
Offset	int Offset Число байтов, добавляемых в конец текущей записи в буфер кэширования TClientDataSet для размещения вычисляемых полей и полей BLOB
OldValue	System::VariantOldValue Прежнее (до редактирования) значение поля. Только для чтения
Origin	AnsiString Origin Имя поля в наборе данных, включая имена базы данных и таблицы
ParentField	TObjectField ParentField Объект родительского поля для данного дочернего поля (в Oracle8 или ADT)

Свойство	Объявление / Описание
ProviderFlags	enum TProviderFlag { pfInUpdate, pfIn Where, pfInKey, pf Hidden }; typedef Set<TProviderFlag, pflnUpdate, pfHidden> TProviderFlags; TProviderFlags ProviderFlags Флаги, определяющие работу провайдера при обновлении данных
ReadOnly	bool Readonly Указывает, является ли значение поля значением только для чтения
Required	bool Required Указывает, должно ли это поле обязательно иметь значение, или оно может оставаться пустым
Size	int Size Размер, определенный в описании поля базы данных, для типов полей, поддерживающих различные размеры
Text	AnsiString Text Строка, отображающая значение поля в компонентах, связанных с данными, в режиме редактирования
ValidChars	typedef Set<char, 0, 255> TFieldChars; TFieldChars ValidChars Определяет символы, которые могут использоваться при редактировании значения поля
Value	System::Variant Value Значение поля
Visible	bool Visible Определяет видимость поля в таблицах данных TDBGrid

Методы

Ниже приведен список основных методов, определенных в **TField** и используемых в приложениях явным образом.

Метод	Объявление / Описание
Assign	void Assign(Classes::TPersistent* Source) Копирует в свойство Value значение другого поля или другого объекта
AssignValue	void AssignValue(const System: :T VarRec &Value) Задает значение поля, используя свойства AsInteger, AsBoolean, AsString, AsFloat. В основном для внутреннего использования
Clear	void Clear(void) Устанавливает значение поля в NULL (очищает его)
FocusControl	void FocusControl(void) Переключает фокус на первый из компонентов, отображающих данное поле. Позволяет обратить внимание пользователя на значение поля

Метод	Объявление / Описание
GetData	**bool** GetData(void * Buffer, **bool** NativeFormat = true) Возвращает неформатированное («сырое») значение поля
GetParentComponent	Classes::TComponent* **GetParentComponent(void)** Возвращает компонент, управляющий загрузкой и сохранением поля: для дочернего поля — родительский объект поля, для остальных — набор данных
HasParent	**bool HasParent(void)** Указывает, является ли поле составной частью структуры другого поля (имеет ли оно родительское поле)
IsBlob	bool **IsBlob()** { IsBlob(__**classid(TField)**)} Определяет, является ли объект полем типа BLOB (binary large object)
IsValidChar	bool **IsValidChar(char InputChar)** Проверяет, является ли указанный символ допустимым для данного поля
RefreshLookupList	void **RefreshLookupList(void)** Обновляет кэш **LookupList**
SetData	void SetData(void * Buffer, bool NativeFormat = true) Присваивает полю неформатированные данные
SetFieldType	void **SetFieldType(TFieldType** Value) Устанавливает интерфейс для методов, задающих тип поля. Используется в классах, производных от TField и работающих с различными типами данных
Validate	void Validate(void * Buffer) Генерирует событие On Validate. Обычно вызывается автоматически при занесении в запись новых данных

События

Событие	Описание
OnChange	Наступает после занесения данных в буфер записи
OnGetText	Наступает при обращении к свойствам DisplayText или Text
OnSetText	Событие наступает при задании свойству Text нового значения
OnValidate	Событие наступает перед занесением данных в буфер записи

TFieldDef — класс описания поля

Класс описания поля таблицы.

Иерархия *TObject — TPersistent — TCollection — TNamedItem*

Модуль ***db.***

Описание

Класс **TFieldDef** содержит описание поля таблицы базы данных, включающее такие атрибуты, как имя поля, тип данных и размер. Используется в собрании полей — свойстве **FieldDefs** класса **TDataSet**.

Описание автоматически генерируется для каждого поля существующей таблицы набора данных. Через объект **TFieldDef** можно получить информацию о полях, даже не открывая таблицу. При создании новой таблицы методом **TTable.CreateTable TFieldDef** используется для задания атрибутов полей таблицы.

Каждое описание соответствует объекту **TField.** но не все объекты **TField** имеют соответствующие им объекты описаний **TFieldDef.** Например, вычисляемые поля не имеют описаний.

Свойства

Свойство	Объявление / Описание
Attributes	TFieldAttributes Attributes Атрибуты поля
ChildDefs	TFieldDefs* ChildDefs Объект, содержащий массив дочерних полей
Collection	TCollection* Collection Указывает собрание FieldDefs класса TDataSet. к которому относится данное описание поля
DataType	**TFieldType** DataTvpe Определяет тип данных, хранящихся в поле
DisplayName	**AnsiString** DisplayName Имя, появляющееся в Редакторе Полей во время проектирования
FieldClass	typedef TMetaClass*TFieldClass; **System::TMetaClass*** FieldClass Класс объекта поля, соответствующий описанию **TFieldDef**; только для чтения
FieldNo	**int** FieldNo Порядковый номер поля в таблице (начинается с 1)
ID	int ID Идентификатор поля в собрании FieldDefs. Только для чтения
InternalCalc Field	**bool** InternalCalcField Определяет, является ли поле вычисляемым в источнике данных; только для чтения
Name	AnsiString Name Имя поля
Precision	int Precision Число разрядов, определяющее точность поля BCD
Required	bool Required Указывает, обязательно ли должно быть задано значение данного поля
Size	int Size Размер поля

Методы

Ниже приведен список основных методов, определенных в **TFieldDef.**

Метод	Объявление / Описание
AddChild	**TFieldDef* AddChild(void)** **Добавляет новый элемент в массив ChildDefs**
Assign	**void Assign(Classes::TPersistent* Source)** **Копирует свойства одного описания поля в другое**
CreateField	**TField* CreateField(Classes::TComponent* Owner,** **TObjectField* ParentField = NULL,** **const AnsiString FieldName = "",** **bool CreateChildren = true)** **Создает объект TField по данному описанию**
HasChildDefs	**bool HasChildDefs(void)** **Определяет, имеет ли объект дочерние описания**

TFieldDefs — класс собрания описаний полей

Класс собрания описаний полей набора данных.

Иерархия *TObiect — TPersistent — TCollection — TOwnedCollection — TDefCollection*

Модуль *db.*

Описание

Класс **TFieldDefs** является косвенным наследником базового класса собраний (коллекций) объектов **TCollection** и характеризует собрание описаний полей **TFieldDef.** В объектах **TTable и TClientDataSet** используется также при создании новых таблиц.

Объекты **TFieldDefs** типов **TADTField и TArrayField** имеют свои объекты того же типа **TFieldDefs** для собраний дочерних полей. В этих случаях создается иерархия объектов **TFieldDef** в отличие от списка **TFieldDefList,** в котором объекты дочерних описаний **TFieldDef** перечисляются последовательно после родительского описания.

Свойства и методы **TFieldDefs** позволяют:

- Получить доступ к описанию любого поля
- Добавлять или удалять описания полей из списка при создании новых таблиц
- Определить количество описаний полей
- Копировать множество описаний полей из одной таблицы в другую

В собрание описаний полей **TFieldDefs** входят все поля таблицы, кроме ключевых.

Свойства

Свойство	Объявление / Описание
Count	**int** Count Число описаний в массиве Items. Только для чтения
Dataset	TDataSet* Dataset Указывает объект TDataSet. к которому относится данный TFieldDefs. Только для чтения

Свойство	Объявление / Описание
HiddenFields	bool HiddenFields Определяет, будут ли видны в «живом» наборе данных вспомогательные невидимые поля. Свойство в основном для внутреннего использования
ItemClass	class PACKAGE TMetaClass; typedef TMetaClass* TClass; System::TMetaClass* ItemClass Класс объектов описаний полей в Items. Только для чтения
Items	TFieldDef* Items[int Indexl Индексированный список описаний полей
ParentDef	TFieldDef* ParentDef Ссылка в описании дочернего поля на описание родительского поля (типов TADTField или T Array Field). Только для чтения
Updated	bool Updated Указывает, соответствуют ли описания в массиве Items реальным экземплярам набора данных

Методы

Ниже приведен список основных методов, определенных в **TFieldDefs** и используемых в приложениях явным образом.

Метод	Объявление / Описание
Add	HIDESBASE void Add(const AnsiString Name, TFieldType DataType, int Size = 0, bool Required = false) Создает новый объект описания поля и добавляет его в свойство Items объекта TFieldDefs. Вместо этого метода рекомендуется AddFieldDef
AddFieldDef	TFieldDef* AddFieldDef(void) Создает новый объект описания поля и добавляет его в свойство Items объекта TFieldDefs
Assign	void Assign(TPersistent* Source) Копирует одно описание поля в другое
Clear	void Clear(void) Очищает массив Items, уничтожая все содержавшиеся в нем объекты типа TCollectionItem
Delete	void Delete(int Index); Удаляет объект с индексом Index из Items
Find	HIDESBASE TFieldDef* Find(const AnsiString Name) Ищет в Items описание поля по его имени Name
GetItemNames	void GetItemNames(Classes::TStrings* List) Возвращает имена всех описаний полей

Метод	Объявление / Описание
IndexOf	**int IndexOf(const AnsiString AName)** **Определяет индекс описания поля в массиве Items по его имени AName**
Insert	**TCollectionItem* Insert(int Index)** **Создает новый объект TCollectionItem и вставляет его в массив Items**
Update	**HIDESBASE void Update(void)** **Обновляет описания в массиве Items, чтобы они отражали реальные свойства полей**

TFields — класс списков объектов полей

Класс совокупности объектов полей набора данных.

Иерархия *TObject*

Модуль *system*.

Описание

Класс **TFields** используется для описания совокупности объектов полей наборов данных TDataSet. включая иерархические структуры с дочерними полями.

Если значение **TDataSet.ObjectView** равно **true,** то поля хранятся в **TFields** иерархически: родительское поле ссылается на дочерние поля. Если значение **TDataSet.ObjectView** равно **false,** то поля хранятся в **TFields** «плоско», т.е. дочерние поля хранятся после родительского поля.

Свойства и методы **TFields** позволяют:

- Получить доступ к определенному полю
- Добавлять и удалять поля списка
- Определять число полей в списке

Свойства

Свойство	Объявление / Описание
Count	int Count Число объектов полей в свойстве Fields. Только для чтения
DataSet	TDataSet* DataSet Набор данных, с которым связан объект TFields. Только для чтения
Fields	TField* Fields[int Index] Список ссылок на поля

Методы

Ниже приведен список основных методов, определенных в **TFields** и используемых в приложениях явным образом.

Метод	Объявление / Описание
Add	**void Add(TField* Field)** **Добавляет поле в список**

Метод	Объявление / Описание
CheckFieldName	void CheckFieldName(const AnsiString FieldName) Проверяет наличие в списке поля с именем FieldName. Если такое поле уже есть, генерируется исключение EDatabaseError
CheckFieldNames	void CheckFieldNames(const AnsiString FieldNames) Проверяет наличие в списке полей, перечисленных в FieldNames с разделителями в виде точек с запятой. Если хоть одного поля нет, генерируется исключение EDatabaseError
FieldByName	TField* FieldByName(const AnsiString FieldName) Дает доступ к полю по его имени FieldName. Если поля с заданным именем не обнаружено, генерируется исключение
FieldByNumber	TField* FieldByNumber(int FieldNo) Дает доступ к полю по его индексу FieldNo в Fields, если порядковый номер поля известен
FindField	TField* FindField(const AnsiString FieldName) Обеспечивает доступ к полю по его имени FieldName
GetFieldNames	void GetFieldNames(Classes::TStrings* List) Возвращает список имен всех полей объекта TFields
IndexOf	int IndexOf(TField* Field) Возвращает номер индекса объекта Field в списке TFields
Remove	void Remove(TField* Field) Удаляет из списка поле Field

TFieldType — тип

Определяет тип данных, хранящихся в поле.

Модуль *DB*.

Определение

```
enum TFieldType {ftUnknown, ftString, ftSmallint, ftInteger,
                 ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD,
                 ftDate, ftTime, ftDateTime, ftBytes,
                 ftVarBytes, ftAutoInc, ftBlob, ftMemo,
                 ftGraphic, ftFmtMemo, ftParadoxOle,
                 ftDBaseOle, ftTypedBinary, ftCursor,
                 ftFixedChar, ftWideString, ftLargeint, ftADT,
                 ftArray, ftReference, ftDataSet, ftOraBlob,
                 ftOraClob, ftVariant, ftInterface,
                 ftIDispatch, ftGuid, ftTimeStamp, ftFMTBcd};
```

Описание

Тип TFieldType используется во многих классах: **TField. TFieldDef. TParam, TParameter, T Aggregate,** а также во многих методах. Возможные значения **TFieldType:**

ftUnknown	неизвестный, неопределенный тип
ftString	символ или строка
ftSmallint	целое размером в 16 бит

ftInteger	целое размером в 32 бита
ftWord	целое без знака размером в 16 бит
ftBoolean	булево значение
ftFIoat	число с плавающей запятой
ftCurrency	денежная сумма
ftBCD	десятичное число в двоичной кодировке
ftDate	дата
ftTime	время
ftDateTime	дата и время
ftBytes	двоичное представление с фиксированным числом байтов
ftVarBytes	двоичное представление с переменным числом байтов
ftAutoInc	автоматически нарастающее целое размером в 32 бита
ftBlob	большой двоичный объект (Binary Large Object)
ftMemo	большой текст
ftGraphic	изображение в виде битовой матрицы
ftFmtMemo	большой форматированный текст
ftParadoxOle	объект OLE Paradox
ftDBaseOle	объект OLE dBASE
ftTypedBinary	типизированное двоичное значение
ftCursor	выходной курсор хранимой процедуры Oracle (применяется только в TPaгat)
ftFixedChar	фиксированные символы
ftWideString	строка типа Wide string
ftLargeInt	большое целое
ftADT	абстрактный тип даты (Abstract Data Type)
ftArray	массив
ftReference	ссылка REF
ftDataSet	набор данных DataSet
ftOraBlob	поля BLOB в таблицах Oracle 8
ftOraClob	поля CLOB в таблицах Oracle 8
ftVariant	данные типа Variant — неизвестные или неопределенные
ftInterface	ссылка на интерфейс **IUnknown**
ftIDispatch	ссылка на интерфейс IDispatch
ftGuid	глобальный уникальный идентификатор GUID
ftTimeStamp	поля даты и времени в dbExpress
ftFMTBcd	двоично-кодированное десятичное число, слишком большое для ftBCD

TFont — класс объекта шрифта

Определяет характеристики шрифта.

Иерархия *TObject — TPersistent — TGraphicsObject*

Модуль *graphics*.

Описание

Объект типа **TFont** определяет множество характеристик, описывающих шрифт, используемый при отображении текстов: высоту шрифта, его имя, атрибуты (полужирный, курсив) и т.д. Используется в свойстве **Font.**

Свойства

Свойство	Объявление / Описание
Charset	typedef Byte **TFontCharset;** TFontCharset Charset Определяет набор символов шрифта
Color	TColor Color Определяет цвет текста
FontAdapter	_di_IChangeNotifier FontAdapter Интерфейс для передачи информации о шрифте в элементы ActiveX
Handle	HFONT Handle Дескриптор шрифта, используемый как параметр функций API Windows, требующих обработки шрифтов. Применяется только в специальных случаях
Height	int Height Характеризует высоту шрифта в пикселах
Name	AnsiString Name Вид (имя) шрифта
Pitch	**enum** TFontPitch { fpDefault, fpVariable, fpFixed **};** TFontPitch Pitch Определяет способ установки ширины символов: по умолчанию (зависит от шрифта), с изменяемой или фиксированной шириной
PixelsPerInch	int PixelsPerlnch Число пикселов принтера или экрана на дюйм. Используется при копировании шрифта с канвы формы на принтер, чтобы обеспечить сортветствие размеров шрифта на экране и принтере. Влияет только на печать. Изменяться пользователем не должно
Size	int Size Размер шрифта в кеглях (пунктах)
Style	enum TFontStyle { fsBold, **fsItalic, fsUnderline,** fsStrikeOut }; typedef Set<TFontStyle, fsBold, fsStrikeOut> TFontStyles; TFontStyles Style Стиль шрифта: полужирный, курсив, подчеркнутый, перечеркнутый

Если система не может найти шрифта с заданной комбинацией свойств **Name, CharSet, Pitch и Size,** Windows использует другой шрифт.

Методы

TFont имеет методы, унаследованные от базовых классов, но переопределенные с учетом особенностей данного класса. Среди них можно отметить два:

Assign	**void Assign(Classes::TPersistent* Source)** Копирование свойств одного объекта типа **TFont** в другой объект. Свойство **PixelsPerInch** методом Assign не копируется. Поэтому метод можно использовать для копирования экранных шрифтов в шрифты принтера и наоборот
TFont	**TFont(void)** Конструктор выделяет память и инициализирует свойства объекта **TFont.** Свойство **Color** устанавливается равным **clWindowText.** Устанавливает свойство **PixelsPerInch,** исходя из разрешения экрана

События

TFont наследует событие **OnChange** от базового класса **TGraphicsObject.**

TGraphic — базовый класс графических объектов

Абстрактный базовый класс графических объектов типа битовых матриц, пиктограмм, метафайлов и типов, определенных пользователем.

Иерархия *TObject — TPersistent*

Модуль *Graphics.*

Описание

Класс **TGraphic** обеспечивает производные от него классы методами хранения, манипулирования и визуализации графических объектов, методами работы с объектами типа **TPicture** и с буфером обмена Clipboard. Свойства класса **TGraphic** дают информацию о состоянии и размерах изображения.

Когда тип графики, с которой ведется работа, известен: битовая матрица, пиктограмма или метафайл, можно использовать объекты производных от **TGraphic** классов **TBitmap. TIcon** или **TMetafile** соответственно. Если формат графики неизвестен, то можно использовать класс **TPicture,** способный работать с графическими объектами любых типов, производных от **TGraphic.**

Свойства

Ниже приведен список свойств, определенных в **TGraphic.**

Свойство	Объявление / Описание
Empty	**bool** Empty Указывает, содержит ли объект изображение. Свойство только для чтения
Height	int Height Указывает высоту изображения в пикселах. Для битовых матриц может изменяться пользователем, что вызывает создание копии матрицы с указанным размером
Modified	bool Modified Указывает, был ли изменен графический объект. Может принимать значение true (изменен) только для битовых матриц. Для пиктограмм и метафайлов всегда равно false, даже если изменения были

Свойство	Объявление / Описание
Palette	HPALETTE Palette Управляет цветами графического изображения. Если изображение не нуждается или не имеет палитры, то Palette = 0
PaletteModified	bool PaletteModified Указывает, была ли изменена палитра графического объекта. Используется в обработчиках событий OnChange
Transparent	bool Transparent Указывает, является ли изображение прозрачным. Используется только для битовых матриц
Width	int Width Указывает ширину изображения в пикселах. Для битовых матриц может изменяться пользователем, что вызывает создание копии матрицы с указанным размером

Методы

Ниже приведены основные методы, объявленные или переопределенные в классе **TGraphic.**

Метод	**Объявление / Описание**
LoadFrom ClipboardFormat	**void LoadFromClipboardFormat(Word AFormat, int AData, HPALETTE APalette)** **Читает изображение из буфера обмена Clipboard в заданном формате**
LoadFromFile	**void LoadFromFile(const AnsiString FileName)** **Читает изображение из файла FileName**
LoadFromStream	**void LoadFromStream(Classes::TStream* Stream)** **Читает графическое изображение из потока Stream**
SaveToClipboard Format	**void SaveToClipboardFormat(Word &AFormat, int &AData, HPALETTE &APalette)** **Сохраняет изображение в буфере обмена Clipboard в заданном формате**
SaveToFile	**void SaveToFile(const AnsiString Filename)** **Сохраняет изображение в файле FileName**
SaveToStream	**void SaveToStream(Classes::TStream* Stream)** **Записывает изображение в поток Stream**

Кроме того, наследуется много методов, в частности, Assign, **Free** и другие.

События

Событие	Описание
OnChange	Событие при изменении графического объекта
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

TIcon — класс

Инкапсулирует пиктограмму Windows.

Иерархия *TObject — TPersistent — TGraphics*

Модуль *Graphics.*

Описание

Класс **TIcon** инкапсулирует пиктограмму Windows. Свойства кла'сса **TIcon** имеют такие объекты, как **TForm и TPicture.**

Пиктограмма соответствует формату файлов **.ico** Windows. Рисовать пиктограммы на канве можно методом канвы **Draw,** но метод **StretchDraw** к ним не применим, поскольку пиктограммы не могут изменять своих размеров.

Свойства

Ниже приведен список свойств, определенных или переопределенных в **TIcon.**

Свойство	Объявление / Описание
Empty	**bool** Empty Указывает, содержит ли объект пиктограмму. Свойство только для чтения
Handle	**HICON** Handle Обеспечивает доступ к обработке пиктограмм в GDI Windows. Используется при вызовах функций API Windows
Height	int Height Указывает высоту изображения в пикселах. Размеры всех пиктограмм в приложении одинаковы и задаются установками Windows. Попытка изменить Height ведет к генерации исключения
Transparent	bool Transparent Указывает, является ли изображение прозрачным. Может использоваться только для чтения. Попытка изменить Transparent ведет к генерации исключения
Width	int Width Указывает ширину изображения в пикселах. Размеры всех пиктограмм в приложении одинаковы и задаются установками Windows. Попытка изменить Width ведет к генерации исключения

Наследуются также свойства **Modified, Palette, PaletteModified.**

Методы

Ниже приведены основные методы, объявленные или переопределенные в классе **TIcon.**

Метод	Объявление / Описание
Assign	**void Assign(Classes::TPersistent* Source)** **Копирует изображение из другого графического объекта, в частности, из буфера обмена Clipboard**
LoadFrom ClipboardFormat	**void LoadFromClipboardFormat(Word AFormat, unsigned AData, HPALETTE APalette)** **Читает изображение из буфера обмена Clipboard в заданном формате**

Метод	Объявление / Описание
LoadFromStream	void **LoadFromStream(Classes::TStream* Stream)** Читает графическое изображение из потока Stream
ReleaseHandle	**HICON ReleaseHandle(void)** Возвращает дескриптор типа **HIcon** и устанавливает дескриптор объекта **TIcon** в NULL
SaveToClipboardFormat	void **SaveToClipboardFormat(Word &Format,** unsigned &Data, **HPALETTE** &APalette) Сохраняет изображение в буфере обмена Clipboard в заданном формате
SaveToStream	void **SaveToStream(Classes::TStream*** Stream) Записывает изображение в поток

Кроме того, наследуются методы **LoadFromFile, SaveToFile** и ряд других.

События

Событие	Описание
OnChange	Событие при изменении графического объекта
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

TimeSeparator — переменная

Определяет символ разделителя, используемого в формате отображения времени.

Модуль *SysUtils.hpp.*

Определение

```
extern PACKAGE char TimeSeparator;
```

Описание

Глобальная переменная **TimeSeparator** определяет символ разделителя, используемого в формате отображения дат **LongTimeFormat** и в ряде других форматов. По умолчанию в русифицированных версиях Windows это символ двоеточия. Но вы можете программно задать другой символ разделителя.

Примеры

Оператор

```
Edit1->Text = Time O;
```

отобразит в окне **Edit1** текущее время в виде: "16:05:02". Если вы предварительно выполните оператор

```
TimeSeparator = '-';
```

изменяющий значение **TimeSeparator,** то приведенный выше оператор вывода в окно **Editl** отобразит дату в формате "16-05-02", т.е. сработает заданный вами разделитель "-".

TIniFile — класс

Класс объекта, отображающего файл .INI.

Иерархия *TObject — TCustomIniFile*

Модуль *inifiles.*

Описание

Файлы .INI — это текстовые файлы, предназначенные для хранения информации о настройках различных приложений. Информация логически группируется в разделы, каждый из которых начинается оператором заголовка, заключенным в квадратные скобки. Например, [Desktop]. В строках, следующих за заголовком, содержится информация, относящаяся к данному разделу, в форме:

```
<ключ>=<значение>
<keyname>=<value>
```

В Windows 95 и NT использование файлов .INI не поощряется и вместо типа **TIniFile,** инкапсулирующего свойства этих файлов, используются типы **TRegistry, TRegIniFile и TRegistryIniFile,** инкапсулирующие свойства системного реестра. Однако большинство программ, в частности, разработанных в Microsoft, и, в частности, Windows, по-прежнему используют файлы .INI для хранения информации о настройке.

Когда в приложении создается операцией **new** объект типа **TIniFile,** ему передается как **FileName** имя файла, с которым он будет связан. Например:

```
#include "inifiles.hpp";
...
TIniFile *Ini = new TIniFile("MY.INI");
```

Свойства и методы **TIniFile** позволяют читать из файла .INI, записывать в него информацию, удалять целые разделы.

Подробно работа с файлами .INI описана в [1] и [2].

Свойства

Ниже указано единственное свойство класса **TIniFile.**

Свойство	Объявление / Описание
FileName	**AnsiString FileName** **Файл .INI, с которым связан объект**

Методы

Ниже приведен список основных методов, определенных в **TIniFile.**

Метод	Объявление / Описание
DeleteKey	**void DeleteKey(const AnsiString Section, const AnsiString Ident)** **Удаляет значение ключа Ident в разделе Section**
EraseSection	**void EraseSection(const AnsiString Section)** **Удаляет раздел Section со всеми его ключами**
ReadBinary Stream	**int ReadBinaryStream(const AnsiString Section, const AnsiString Name, TStream Value)** **Читает в текущую позицию потока Value значение ключа Name раздела Section, являющееся последовательностью шестнадцатеричных символов (не более 1023 символов). Возвращает число занесенных в поток символов**

Метод	Объявление / Описание
ReadBool	**bool ReadBool(const AnsiString Section, const AnsiString Ident, bool Default)** Возвращает булево значение ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadDate	**System::TDateTime ReadDate(const AnsiString Section, const AnsiString Name, System::TDateTime Default);** Возвращает значение даты типа **TDateTime** ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadDate Time	**System::TDateTime ReadDateTime(const AnsiString Section, const AnsiString Name, System::TDateTime Default)** Возвращает значение даты и времени типа **TDateTime** ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadFloat	**double ReadFloat(const AnsiString Section, onst AnsiString Name, double Default)** Возвращает действительное значение ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadInteger	**int ReadInteger(const AnsiString Section, const AnsiString Ident, int Default)** Возвращает целое значение ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadSection	**void ReadSection(const AnsiString Section, Classes::TStrings* Strings)** Читает в Strings типа **TStrings** имена всех ключей раздела **Section**
ReadSections	**void ReadSections(Classes::TStrings* Strings)** Читает **в Strings** типа **TStrings** имена всех разделов файла
ReadSection Values	**void ReadSectionValues(const AnsiString Section, Classes::TStrings* Strings)** Читает в Strings типа TStrings значения всех ключей раздела Section
ReadString	**AnsiString ReadString(const AnsiString Section, const AnsiString Ident, const AnsiString Default)** Возвращает строку значения ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения
ReadTime	**System::TDateTime ReadTime(const AnsiString Section, const AnsiString Name, System::TDateTime Default)** Возвращает значение времени типа **TDateTime** ключа **Ident** раздела **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось раздела, ключа или значения

Метод	Объявление / Описание
SectionExists	bool SectionExists(const AnsiString Section) Возвращает true, если в файле существует раздел Section
UpdateFile	void UpdateFile(void) Очищает буфер и записывает файл на диск. Действует только на Windows 95 и 98, т.к. Windows NT не использует буфер
ValueExists	bool ValueExists(const AnsiString Section, const AnsiString Ident) Возвращает true, если в файле существует ключ Ident в разделе Section
WriteBinary Stream	void WriteBinaryStream (const AnsiString Section, const AnsiString Name, TStream Value) Записывает из потока Value двоичные данные в ключ Name раздела Section в виде последовательности шестнадцатеричных символов
WriteBool	void WriteBool(const AnsiString Section, const AnsiString Ident, bool Value) Записывает булево значение Value в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteDate	void WriteDate(const AnsiString Section, const AnsiString Name, System::TDateTime Value) Записывает значение даты Value типа TDateTime в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteDate Time	void WriteDateTime(const AnsiString Section, const AnsiString Name, System::TDateTime Value) Записывает значение даты и времени Value типа TDateTime в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteFloat	void WriteFloat(const AnsiString Section, const AnsiString Name, double Value) Записывает действительное значение Value в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteInteger	void WriteInteger(const AnsiString Section, const AnsiString Ident, int Value) Записывает целое значение Value в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteString	void WriteString(const AnsiString Section, const AnsiString Ident, const AnsiString Value) Записывает значение Value в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются
WriteTime	void WriteTime(const AnsiString Section, const AnsiString Name, System::TDateTime Value) Записывает значение времени Value типа TDateTime в строку ключа Ident раздела Section. Если нужной строки или раздела нет, они создаются

События

Никаких событий в классе **TIniFile** не определено.

Примеры

Следующие операторы создают объект **Ini** типа **TIniFile** и связывают его с файлом *My.ini* (предполагается, что файл расположен в каталоге Windows, так что его имя указывается без пути):

```
#include "inifiles.hpp";
...
TIniFile *Ini = new TIniFile("MY.INI");
```

Следующий оператор проверяет успешность создания объекта Ini и наличие в файле раздела *My Section,* содержащего ключ *MyKey*:

```
if ((Ini != NULL) && Ini->ValueExists("My Section","MyKey"))
 ...
```

Следующий оператор заносит в ключ *MyKey* раздела *My Section* значение '5':

```
Ini->WriteInteger("My Section","MyKey",5);
```

Следующий оператор удаляет ключ *MyKey* раздела *My Section:*

```
Ini->DeleteKey("My Section","MyKey");
```

Следующие операторы сохраняют содержимое объекта Ini в файле на диске и разрушают объект Ini:

```
Ini->UpdateFile();
delete Ini;
```

TList — класс

Содержит список указателей на любые объекты.

Иерархия *TObject*

Модуль *classes.*

Описание

Объект типа **TList** предназначен для хранения и управления списком указателей на объекты. Свойства и методы **TList** позволяют добавлять и удалять элементы списка, изменять их расположение в списке, сортировать элементы и проводить другие манипуляции с данными.

Свойства

Свойство	Объявление / Описание
Capacity	**int Capacity** **Число указателей, которые могут храниться в объекте. Всегда не меньше Count**
Count	**int Count** **Число указателей, хранящихся в объекте**
Items	**void * Items[int Index]** **Элементы массива указателей в объекте**
List	**typedef void *TPointerList[134217727];** **typedef TPointerList *PPointerList;** **PPointerList List** **Указатель на массив указателей в объекте**

Методы

Метод	Объявление / Описание
Add	int Add(void * Item) Добавление нового указателя Item в список
Assign	enum TListAssignOp { laCopy, laAnd, laOr, laXor, laSrcUnique, laDestUnique }; void Assign(TList* ListA, TListAssignOp AOperator = laCopy, TList* ListB = NULL) Комбинирует элементы данного списка с элементами других списков ListA и ListB. Параметр AOperator определяет опции копирования. Если ListB не NULL, то сначала все элементы данного списка заменяются элементами ListA, а затем к ним присоединяются элементы ListB в соответствии с опциями AOperator. Если ListB = NULL, то элементы ListA переносятся в данный список в соответствии с опциями AOperator
Clear	void Clear(void) Очистка списка
Delete	void Delete(int Index) Удаление из списка элемента по его индексу Index
Error	void Error(const AnsiString Msg, int Data) { Error(__classid(TList), Msg, Data)} Генерация исключения EListError с передачей ему параметров Msg (сообщение) и Data (число)
Exchange	void Exchangc(int Indexl, int Index2) Взаимная перестановка двух элементов списка с индексами Index1 и Index2
Expand	TList* Expand(void) Расширяет емкость списка Capacity и копирует список
First	void * First(void) Возвращает первый указатель списка
IndexOf	int IndexOf(void * Item) Возвращает индекс первого вхождения в список заданного указателя Item
Insert	void Insert(int Index, void * Item) Вставляет элемент Item в список в заданную позицию Index
Last	void * Last(void) Возвращает последний указатель списка
Move	void Move(int CurIndex, int NewIndex) Изменяет текущую позицию CurIndex элемента в списке на NewIndex
Pack	void Pack(void) Удаляет из списка указатели NULL

Метод	Объявление / Описание
Remove	int Remove(void * Item) Удаляет из списка элемент Item
Sort	typedef int (*TListSortCompare) (void * Item1, void * Item2); void Sort(TListSortCompare Compare) Сортирует список с использованием указанной функции сравнения Compare

Остальные методы наследуются от базового класса TObiect, в частности, **Free, ClassName, ClassNameIs** и ряд других.

Примеры

Ниже приведен пример, в котором создается и реорганизуется список структур, содержащих информацию о сотрудниках какой-то организации.

```
// Объявление структуры
struct TPers {
  AnsiString  Fam,Nam,Par;
  unsigned    Year;
  AnsiString  Dep;
};
TList *List = new TList; // Создание экземпляра списка
TPers *Pnew;             // Объявление указателя на структуру
.................
// Выделение памяти под новую структуру
 Pnew = new TPers;
// Заполнение элементов структуры
 Pnew->Fam = "Иванов";
 Pnew->Nam = "Иван";
 Pnew->Par = "Иванович";
 Pnew->Year = 1960;
 Pnew ->Dep = "Бухгалтерия";
// Включение указателя в список
 List->Add(Pnew);

// Выделение памяти под новую структуру
 Pnew = new TPers;
// Заполнение элементов структуры
 Pnew->Fam = "Петров";
 Pnew->Nam = "Петр";
 Pnew->Par = "Петрович";
 Pnew->Year = 1970;
 Pnew ->Dep = "Цех 1";
// Включение указателя в список
 List->Add(Pnew);
```

Просмотр всего сформированного списка можно осуществить следующей функцией:

```
void ShowList()
{
 for (int i = 0; i < List->Count; i++)
    ShowMessage(((TPers *)List->Items[i])->Fam +
    ' '+((TPers *)List->Items[i])->Nam +
    ' '+((TPers *)List->Items[i])->Par +
    '\n'+IntToStr(((TPers *)List->Items[i])->Year) +
    " г.р.\n Отдел '"+((TPers *)List->Items[i])->Dep +     "');
}
```

Она выдаст поочередно сведения обо всех сотрудниках, занесенных в список.

В приведенной функции доступ к элементу **List->Items[i]** можно заменить на **List->List[i].**

Оператор

```
List->Exchange(0,1);
```

поменяет местами две первые структуры (первым станет "Петров", вторым - "Иванов").

Можно обеспечить сортировку списка методом Sort. Для этого надо определить функцию сравнения двух элементов списка. В нашем случае, если важна сортировка по фамилиям, эта функция может иметь вид:

```
int MyCompare(void * Item1, void * Item2)
{
 return ((TPers *)Item1)->Fam.AnsiCompare(((TPers *)Item2)->Fam);
}
```

Тогда оператор, обеспечивающий сортировку списка должен иметь вид:

```
List->Sort(MyCompare);
```

Следующий оператор изменяет значения поля Dep первой структуры в списке:

```
((TPers *)List->Items[0])->Dep = "Уволен";
```

Следующий оператор удаляет первую структуру в списке:

```
List->Delete(0);
```

При этом число элементов списка уменьшается на 1. Можно удаление элемента оформить иначе:

```
List->Items[0] = NULL;
```

Но при этом число элементов не изменяется и надо в программах обработки списка принять меры, чтобы не появлялись ошибки при обработке нулевого указателя. Впрочем, если выполнить оператор

```
List->Pack();
```

то все нулевые указатели из списка будут удалены.

Следующие операторы формируют новую структуру и вставляют ее первой в список:

```
// Выделение памяти под новую структуру
 Pnew = new TPers;
// Заполнение элементов структуры
 Pnew->Fam = "Сидоров";
 Pnew->Nam = "Сидор";
 Pnew->Par = "Сидорович";
 Pnew->Year = 1980;
 Pnew ->Dep = "Управление";
// Включение указателя первым в список
 List->Insert(0,Pnew);
```

Следующие операторы очищают память, удаляя все объекты, на которые имеются ссылки в списке, и удаляя сам список:

```
for (int i = 0; i < List->Count; i++)
{
  Pnew = (TPers *)List->Items[i];
  delete Pnew;
}
delete List;
```

TMenuItem — класс

Описывает свойства элементов меню.

Иерархия *TObiect - TPersistent - TComponent*

Модуль *menus.*

Описание

Объекты **TMenuItem** характеризуют разделы меню. Создаются эти объекты Конструктором Меню, который вызывается двойным щелчком на компоненте меню. Каждой команде меню соответствует свой объект **TMenuItem.**

Свойства

Ниже приведен список основных свойств класса **TMenuItem.**

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Указывает действие, связанное с данным разделом меню
AutoCheck	**bool** AutoCheck При значении true при каждом щелчке на разделе меню его свойство Checked автоматически переключается перед наступлением события OnClick. При значении false переключение свойства Checked, если это требуется, надо осуществлять программно в обработчике события OnClick
AutoLine Reduction	**enum TMenuItemAutoFlag** { **maAutomatic,** maManual, maParent }; typedef **TMenuItemAutoFlag TMenuAutoFlag;** TMenuItemAutoFlag AutoLineReduction При значении maAutomatic из меню автоматически удаляются разделители в начале и в конце меню, а также оставляет один из нескольких следующих подряд разделителей. При значении maManual автоматическая коррекция меню не производится. Значение maParent может быть задано только для подменю и означает, что возможность автоматической коррекции определяется свойством AutoLineReduction родительского меню
Bitmap	Graphics::TBitmap* Bitmap Указывает изображение, появляющееся в заголовке раздела меню. Если установлено свойство **ImageIndex** и в родительском меню свойство Images не равно NULL, изображение определяется этими свойствами, а не Bitmap
Break	enum TMenuBreak { mbNone, mbBreak, mbBarBreak }; TMenuBreak Break Определяет, не надо ли начинать новый столбец разделов меню
Caption	**AnsiString** Caption Определяет надпись раздела меню
Checked	bool Checked Указывает, будет ли изображаться в разделе меню флажок, показывающий, что данный раздел выбран
Command	Word Command Определяет идентификатор Command ID Windows, связанный с данным разделом меню. Используется в приложениях, напрямую обрабатывающих сообщения Windows WM_COMMAND. Свойство только для чтения

Свойство	Объявление / Описание
Count	int Count Указывает число подразделов данного раздела меню, раскрываемых как выпадающий список. Учитываются только подразделы, относящиеся непосредственно к данному разделу, хотя сами они могут в свою очередь тоже иметь подразделы. Свойство только для чтения
Default	TMenuItem* Items[int Index] Определяет, является ли данный раздел разделом по умолчанию своего подменю, т.е. разделом, выполняемым при двойном щелчке пользователя на родительском разделе. Подменю может содержать только один раздел по умолчанию, выделяемый жирным шрифтом
Enabled	bool Enabled Определяет, доступен ли раздел. Недоступный раздел изображается серой надписью и не реагирует на щелчок пользователя
GroupIndex	Byte GroupIndex Определяет логическую группу, к которой относится раздел. Используется при объединении меню нескольких форм
Handle	HMENU Handle Дескриптор выпадающего меню Windows, связанного с данным разделом. Используется при вызовах функций API Windows. Имеет смысл только при Count > 0. Свойство только для чтения
HelpContext	typedef int THelpContext; Classes::THelpContext HelpContext Номер темы справки, связанной с данным разделом
Hint	AnsiString Hint Текст, отображаемый в ярлычке подсказки или в строке состояния
ImageIndex	typedef int TImageIndex; TImageIndex ImageIndex Индекс изображения раздела в списке изображений родительского меню Images
Items	TMenuItem* Items[int Index] Список разделов подменю, относящегося к данному разделу. Индексы начинаются с 0 и соответствуют последовательности разделов подменю. Свойство позволяет в цикле изменять какие-то свойства подразделов, например, Enabled. Свойство только для чтения
MenuIndex	int MenuIndex Индекс раздела, определяющий его место в свойстве Items родительского меню. Изменение MenuIndex перемещает раздел в Items. Индекс может не соответствовать видимой пользователем позиции раздела в меню, т.к. некоторые разделы могут быть невидимыми
Parent	TMenuItem* Parent Указывает на родительский раздел меню, в выпадающем списке которого содержится данный раздел. Если раздел находится на верхнем уровне меню, то свойство Parent указывает на свойство Items самого меню. Свойство только для чтения

Свойство	Объявление / Описание
RadioItem	bool RadioItem Определяет, должен ли работать данный раздел в режиме радиокнопки совместно с другими разделами, имеющими то же значение свойства GroupIndex. См. подробнее в описании GroupIndex
Shortcut	typedef Word TShortCut; TShortCut Shortcut Определяет комбинацию «горячих» клавиш, обеспечивающих быстрый доступ к разделу меню
Visible	bool Visible Определяет видимость раздела меню. Отличие от свойства доступности Enabled в том, что недоступный раздел видим, а при Visible = false раздел невидим, соседние разделы смыкаются, занимая пустующее место

Методы

Ниже приведены основные методы, объявленные или переопределенные в классе TMenuItem.

Метод	Объявление / Описание

Add	void Add(TMenuItem* Item);
	void Add(const TMenuItem* * AItems, const int AItems_Size);
	Добавляет новый элемент Item или массив элементов Altems в свойство Items данного раздела, содержащее список подразделов его выпадающего меню

Delete	void Delete(int Index) Удаляет подраздел, указанный своим индексом Index, из массива свойства Items данного раздела меню. Индексы начинаются с 0
Find	TMenuItem* Find(AnsiString ACaption) Возвращает первый элемент массива, имеющий заданное значение надписи ACaption (символ амперсанда, отмечающий символ ускоренного доступа, при поиске игнорируется), или NULL, если раздел не найден
GetParent Menu	TMenu* GetParentMenu(void) Возвращает меню типа TMenu, частью которого является данный раздел
IndexOf	int IndexOf(TMenuItem* Item) Возвращает индекс раздела меню, в подменю которого расположен данный раздел. Индекс соответствует позиции в массиве Items. Если данный раздел расположен не в подменю, возвращается -1
Insert	HIDESBASE void Insert(int Index, TMenuItem* Item) Вставляет указанный подраздел Item в выпадающее меню данного раздела и задает индекс Index свойства Items вставляемого раздела
IsLine	bool IsLine(void) Определяет, является ли раздел разделителем
Remove	HIDESBASE void Remove(TMenuItem* Item) Удаляет указанный подраздел Item из массива свойства Items данного раздела меню

События

Событие	Описание
OnAdvanced DrawItem	Наступает, когда необходимо перерисовать нестандартное изображение раздела меню (если свойство OwnerDraw = true). Отличается от OnDrawItem параметрами
OnClick	Происходит при щелчке левой кнопки мыши на разделе меню
OnDrawItem	Наступает, когда необходимо перерисовать нестандартное изображение раздела меню (если свойство OwnerDraw = true)
OnMeasureItem	Происходит перед прорисовкой нестандартного (если свойство OwnerDraw = true) изображения раздела, чтобы определить его размер

TMessage — тип

Тип параметра, характеризующего сообщения Windows и передаваемого в метод **WndProc.**

Модуль *Messages.*

Объявление

Messages

```
struct TMessage
{
 Cardinal Msg;
 union
 {
  struct
  {
   Word WParamLo;
   Word WParamHi;
   Word LParamLo;
   Word LParamHi;
   Word ResultLo;
   Word ResultHi;
  };

  struct
  {
   int WParam;
   int LParam;
   int Result;
  };

 };
} ;
```

Описание

Тип TMessage представляет в **WndProc** и других процедурах параметры сообщений Windows.

TMetafile— класс

Инкапсулирует метафайл Win32 Enhanced.

Иерархия *TObject — TPersistent — TGraphic*

Модуль *graphics.*

Описание

Класс **TMetafile** позволяет хранить изображения, соответствующие графическим метафайлам .emfWindows. Свойства **TMetafile** описывают размер и характеристики метафайла. Рисовать метафайлы на канве можно методами канвы **Draw** и **StretchDraw.** Свойство **Enhanced** определяет, как метафайл хранится на диске: **true** соответствует формату **.emf** (Win32 Enhanced Metafile), а **false** - - **.wmf** (Windows 3.1 Metafile).

Свойства

Ниже приведен список свойств, определенных или переопределенных в TMetafile.

Свойство	Объявление / Описание
CreatedBy	AnsiString **CreatedBy** Указывает имя автора или приложения, создавшего метафайл. Свойство только для чтения. Чтобы задать CreatedBy нового метафайла, надо вызвать метод **CreateWithComment** объекта **TMetafileCanvas**
Description	**AnsiString** Description Определяет не обязательный текст описания метафайла. Свойство только для чтения. Чтобы задать Description нового метафайла, надо вызвать метод **CreateWithComment** объекта **TMetafileCanvas**
Empty	**bool** Empty Указывает, содержит ли объект метафайл. Только для чтения
Enhanced	**bool** Enhanced Значение true соответствует формату хранения на диске **.emf,** а false — **.wmf.** В памяти метафайл всегда хранится в формате **.emf.** Формат **.wmf** ведет к частичной потере информации и оставлен только для обратной совместимости
Handle	**int** Handle Дескриптор, используемый для доступа к GDI Windows и вызова функций API Windows
Height	int Height Указывает высоту изображения в пикселах. Может изменяться пользователем
Inch	Word Inch Используется для метафайлов WMF и указывает число единиц на дюйм, необходимое для масштабирования. Метафайлы EMF хранят эту информацию внутри себя
MMHeight	int MMHeight Содержит высоту изображения в единицах 0.01 мм. MMHeight дает более точное значение высоты, чем свойство Height, значение которого измеряется в пикселах
MMWidth	int MMWidth Содержит ширину изображения в единицах 0.01 мм. MMWidth дает более точное значение ширины, чем свойство Width, значение которого измеряется в пикселах

Свойство	Объявление / Описание
Palette	HPALETTE Palette Управляет цветами графического изображения. Если изображение не нуждается или не имеет палитры, то Palette == 0
Transparent	bool Transparent Указывает, может ли изображение быть прозрачным. Только для чтения
Width	int Width Указывает ширину изображения в пикселах. Может изменяться пользователем

От TGraphic наследуются свойства **Modified и PaletteModified.**

Методы

Ниже приведены основные методы, объявленные или переопределенные в классе **TMetafile.**

Метод	Объявление / Описание
Assign	**void Assign(Classes::TPersistent* Source)** **Копирует свойства Enhanced и Palette изображения из другого графического объекта, в частности, из буфера обмена Clipboard**
Clear	**void Clear(void)** **Очищает изображение**
LoadFrom ClipboardFormat	**void LoadFromClipboardFormat(Word AFormat, int AData, HPALETTE APalette)** **Читает изображение из буфера обмена Clipboard в заданном формате**
LoadFromFile	**void LoadFromFile(const AnsiString FileName)** **Читает изображение из файла FileName**
LoadFromStream	**void LoadFromStream(Classes::TStream* Stream)** **Читает графическое изображение из потока**
ReleaseHandle	**int ReleaseHandle(void)** **Возвращает дескриптор типа HENHMETAFILE и устанавливает дескриптор объекта в NULL**
SaveToClipboard Format	**void SaveToClipboardFormat(Word &AFormat, int &AData, HPALETTE &APalette)** **Сохраняет изображение в буфере обмена Clipboard в заданном формате**
SaveToFile	**void SaveToFile(const AnsiString Filename)** **Сохраняет изображение в файле Filename**
SaveToStream	**void SaveToStream(Classes::TStream* Stream)** **Записывает изображение в поток Stream**

Ряд методов наследуется от классов **TPersistent и TObject.**

События

Событие	Описание
OnChange	Событие при изменении графического объекта
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

TMouseButton — тип

Тип определяет кнопку мыши.

Определение

```
enum TMouseButton ( mbLeft, mbRight, mbMiddle };
```

Описание

Тип TMouseButton используется в обработчиках различных событий, связанных с мышью, для идентификации нажатой кнопки: **mbLeft** — левая, **mbRight** — правая, **mbMiddle** — средняя.

TObject — базовый класс всех объектов

Базовый класс всех объектов VCL в C++Builder.

Модуль *System*.

Описание

Класс **TObject** инкапсулирует основные функции, свойственные всем объектам VCL в C++Builder. Интерфейс **TObject** обеспечивает:

- Возможность создания, управления и разрушения экземпляров объектов, включая выделение под них памяти, инициализацию и освобождение памяти после их уничтожения.
- Поддержка информации об объектах и типах (run-time type information – RTTI).
- Поддержка обработки сообщений.

Все классы библиотеки VCL в C++Builder являются прямыми или косвенными наследниками **TObject.** Прямое наследование используется только при объявлении простых классов, объекты которых не являются компонентами, не могут присваиваться друг другу и не участвуют в операциях обмена с потоками. Подавляющее большинство классов являются косвенными наследниками **TObject** и производятся от промежуточных классов.

Большинство методов **TObject** не используются непосредственно в компонентах, с которыми имеет дело пользователь. Исходные методы **TObject** обычно перегружены в классах-наследниках или заменены другими, построенными на их основе. Большинство методов являются методами класса.

Хотя формально **TObject** не является абстрактным классом, но объекты этого класса создавать нельзя.

Методы

Большинство методов класса **TObject** пользователями непосредственно не используется. Ниже приводятся все методы этого класса.

Метод	Объявление / Описание
AfterConstruction	void **AfterConstruction()** Вызывается автоматически после создания объекта
BeforeDestruction	void **BeforeDestruction()** Вызывается автоматически перед срабатыванием деструктора
ClassInfo	typedef TMetaClass* TClass; static void * ClassInfo(TClass **cls);** void * **ClassInfo(){**return **ClassInfo(ClassType());** } Возвращает указатель на таблицу информации времени выполнения (RTTI) о типе объекта. Таблица RTTI содержит информацию о типе компонента, типе его родителя, о его опубликованных свойствах. Classlnfo используется средой проектирования Delphi, но явным образом обычно не вызывается, так как доступ к RTTI проще получить другими методами
ClassName	typedef TMetaClass* TClass; static ShortString ClassName(TClass **cls);** **ShortString ClassName()** { return **ClassName(ClassType());}** Возвращает имя типа объекта
ClassNameIs	typedef TMetaClass* TClass; static **bool** ClassNameIs(TClass **cls,** const AnsiString string); **bool ClassNameIs(const** AnsiString string) { return ClassNameIs(ClassType(), string); } Возвращает true, если передаваемое в функцию имя Name совпадает с именем данного класса
ClassParent	typedef TMetaClass* TClass; static TClass **ClassParent(TClass cls)** TClass ClassParent() { return **ClassParent(ClassType());}** Возвращает тип непосредственного предка данного класса
ClassType	typedef TMetaClass* TClass; TClass **ClassType()** Возвращает указатель на таблицу информации времени выполнения (RTTI) о типе объекта
CleanupInstance	void **CleanupInstance()** Очищает длинные строки и объекты variant. Вызывается автоматически в методе **Free**
DefaultHandler	void **DefaultHandler(void*** Message) Процедура обработки сообщения по умолчанию. Вызывается, из Dispatch, если не найден обработчик данного сообщения
Dispatch	void Dispatch(void *Message) Вызывает обработчик сообщения, полученного объектом
FieldAddress	void * FieldAddress(const ShortString &Name) Возвращает указатель на опубликованное поле объекта

Метод	Объявление / Описание
Free	**Free()** Уничтожает объект и освобождает выделенную под него память. Явным образом из приложений не вызывается. Для уничтожения объекта надо использовать операцию **delete**
FreeInstance	**void FreeInstance()** Освобождает память, выделенную ранее вызванным методом **NewInstance.** Автоматически вызывается деструктором. Непосредственный вызов пользователем не требуется. Должен быть перегружен, если перегружен метод **NewInstance.** Использует **InstanceSize** для определения размера выделенной области памяти
GetInterface	**bool GetInterface(const TGUID &IID, void *Obj);** **template <typename T>** **bool GetInterface(DelphiInterface<T>& smartIntf)** **{** **return GetInterface(__uuidof(T),** **reinterpret_cast<void*>(static_cast<T**>(&smartIntf)));** **}** Возвращает интерфейс, указанный идентификатором
GetInterfaceEntry	**PInterfaceEntry GetInterfaceEntry(const TGUID IID)** Возвращает вход интерфейса, указанного идентификатором
GetInterfaceTable	**struct PACKAGE TInterfaceTable{** **int EntryCount;** **TInterfaceEntry Entries[];** **};** **typedef TInterfaceTable *PInterfaceTable;** **static PInterfaceTable * GetInterfaceTable(void)** Возвращает указатель на структуру, содержащую все интерфейсы класса
InheritsFrom	**typedef TMetaClass* TClass;** **static bool InheritsFrom(TClass cls, TClass aClass);** **bool InheritsFrom(TClass aClass)** **{ return InheritsFrom(ClassType(), aClass);}** Определяет, является ли указанный класс **AClass** предком данного объекта
InitInstance	**typedef TMetaClass* TClass;** **static TObject * InitInstance(TClass cls, void *instance);** **TObject * InitInstance(void *instance)** **TObject::InitInstance(this, instance); }** Инициализирует новый объект и указатель на его таблицу виртуальных методов. Вызывается автоматически методом **NewInstance.** Не может быть перегружен
InstanceSize	**typedef TMetaClass* TClass;** **static long InstanceSize(TClass cls);** **long InstanceSize(){ return InstanceSize(ClassType()); }** Возвращает число байтов, занимаемых объектом. Может использоваться для анализа затрат памяти различными типами объектов

Метод	Объявление / Описание
MethodAddress	typedef TMetaClass* TClass; static void * MethodAddress(TClass cls, const ShortString &Name); void * MethodAddress(const ShortString &Name) { return MethodAddress(ClassType(), Name); } Возвращает адрес указанного метода Name. Непосредственно не вызывается
MethodName	typedef TMetaClass* TClass; static ShortString MethodName(TClass cls, void *Address); ShortString MethodName(void *Address) { return MethodName(ClassType(), Address); } Возвращает имя метода, расположенного по указанному адресу
NewInstance	typedef TMetaClass* TClass; virtual TObject* NewInstance(TClass cls); Выделяет область памяти под объект и возвращает указатель на нее. Автоматически вызывается всеми конструкторами. Использует InstanceSize. Может перегружаться в классах-наследниках.
SafeCallException	HResult SafeCallException(TObject *ExceptObject, void *ExceptAddr) Обрабатывает исключения OLE и Com

TOleServer — базовый класс серверов OLE

Базовый класс компонентов, представляющих импортируемые серверы COM.

Иерархия *TObject — TPersistent — TComponent*

Модуль *OleServer*.

Описание

Класс **TOleServer** является базовым для множества импортируемых серверов COM, имеющихся в библиотеке компонентов C++Builder 6 и 5. В самом классе **TOleServer** объявлены абстрактные методы и свойства, позволяющие устанавливать связь с сервером. Поэтому нельзя создавать объекты класса **TOleServer.**

Многочисленные потомки класса **TOleServer** — серверы COM создаются импортом библиотек типов, осуществляемым в среде C++Builder командой Project | Import Type Library. Когда приложение вызывает какой-то метод класса — наследника **TOleServer** или устанавливает какое-то из его свойств, компонент автоматически устанавливает соединение с соответствующим сервером COM. Соединение можно установить и явным образом, выполнив метод **Connect.**

Свойства

Ниже приведен список основных свойств, объявленных в **TOleServer.**

Свойство	Объявление / Описание
AutoConnect	**bool** AutoConnect Определяет, должен ли сервер автоматически загружаться с началом выполнения приложения (при AutoConnect = true), или связь с ним должна устанавливаться только при вызове его методов, установке его свойств или вызове метода Connect. Значение AutoConnect = **true** влияет только при запуске приложения на выполнение. Последующая установка AutoConnect = **true** ни на что не влияет
ConnectKind	**enum TConnectKind** {ckRunningOrNew, **ckNewInstance, ckRunningInstance,** ckRemote, **ckAttachToInterface**}; TConnectKind ConnectKind Указывает тип процесса соединения с сервером: ckRunningOrNew — подсоединиться к выполняющемуся серверу или создать новый экземпляр сервера, ckNewInstance — всегда создавать новый экземпляр сервера, ckRunningInstance — подсоединиться только к выполняющемуся серверу, ckRemote — подсоединиться к удаленному серверу (эта опция должна сочетаться с заданием **RemoteMachineName**), ckAttachToInterface — не подсоединяться к серверу (вместо этого приложение обеспечивает интерфейс методом ConnectTo)
Remote MachineName	AnsiString RemoteMachmeName Указывает компьютер, на котором выполняется удаленный сервер. Используется только при ConnectKind равном ckRemote

Методы

Основные методы **TOleServer:**

Метод	Объявление / Описание
Connect	void Connect(void) Устанавливает связь с сервером COM. Если свойство AutoConnect = true, этот метод вызывается автоматически при начале выполнения приложения. При значении свойства ConnectKind равном ckAttachToInterface вместо Connect надо использовать ConnectTo, который вводится классами, наследующими TOleServer. Если ConnectKind равен ckRemote, то перед вызовом Connect надо задать значение свойства RemoteMachmeName
ConnectTo	Этот метод установления связи с сервером отсутствует в самом классе TOleServer, но вводится во многих компонентах классов — наследников, которые подключаются не к основному интерфейсу сервера, а к составляющим его объектам. К таким компонентам относятся **WordDocument, WordParagraphFormat**, Excel Workbook и др.
Disconnect	void Disconnect(void) Разрывает связь с сервером COM

Класс наследует также множество методов, характерных для всех компонентов, от **TComponent. TPersistent. TObiect.**

TPen — класс

Определяет свойства пера, используемые при рисовании линий и фигур на канве.

Иерархия *TObject — TPersistent — TGraphicsObject*

Модуль *Graphics.*

Описание

Класс **TPen** инкапсулирует атрибуты пера Windows при рисовании на канве — объекте типа **TCanvas.**

Свойства

Свойство	Объявление / Описание
Color	TColor Color Цвет пера. По умолчанию — clBlack
Handle	HPEN Handle Дескриптор кисти окна. Используется для доступа к функциям API Windows
Mode	TPenMode Mode Определяет режим рисования линий
Style	TPenStyle Style Определяет стиль рисования линий
Width	int Width Определяет толщину линии в пикселах. Влияет на Style

Методы

В классе **TPen** не введено каких-то принципиально новых методов. Переопределены такие общие методы, как **Assign** и конструктор. Остальные методы наследуются от классов-предков.

События

Класс **TPen** наследует от класса **TGraphicsObject** событие **OnChange,** наступающее после изменения графического объекта.

TPersistent — базовый класс объектов, участвующих в операциях с потоками

Базовый класс всех объектов VCL C++Builder, допускающих операцию присваивания или участвующих своими свойствами в операциях с потоками.

Иерархия *TObject*

Модуль *classes.*

Описание

Класс **TPersistent** инкапсулирует фундаментальные свойства объектов, которые должны иметь возможность присваиваться друг другу или читать и записывать свои свойства в поток. Методы класса **TPersistent** обеспечивают:

- Определение процедур загрузки и сохранения данных в потоке.
- Присваивание значений свойствам.
- Присваивание содержимого одного объекта другому.

Объекты **TPersistent** создаваться не могут. Класс используется только для создания производных классов.

Класс **TPersistent,** является предшественником большинства классов C++Builder и всех классов компонентов.

Методы

Класс имеет следующие методы:

Метод	Объявление / Описание
Assign	**void Assign(TPersistent* Source)** **Копирует данные одного объекта в другой**
AssignTo	**void AssignTo(TPersistent* Dest)** **Защищенная реализация метода, аналогичного Assign**
DefineProperties	**void DefineProperties(TFiler* Filer)** **Виртуальный защищенный метод обмена данными с потоком**
GetNamePath	**DYNAMIC AnsiString GetNamePath()** **Возвращает строку, используемую в Инспекторе Объектов**
GetOwner	**DYNAMIC TPersistent* GetOwner(void)** **Защищенный метод, возвращающий владельца объекта**

Из всех этих методов только Assign может непосредственно использоваться пользователем при работе с объектами. Остальные могут потребоваться только при создании новых классов. Класс наследует также множество методов от класса **TObject.**

TPicture — класс

Описывает объект, являющийся контейнером графических объектов типа битовых матриц, пиктограмм, метафайлов и определенных пользователем типов графических объектов.

Иерархия *TObiect — TPersistent*

Модуль *Graphics.*

Описание

Объект типа **TPicture** является контейнером любого графического объекта **TGraphic.** тип которого указывается свойством **Graphic.** Объект **TPicture** имеет полиморфные методы файлового чтения и записи **LoadFromFile** и **SaveToFile,** автоматически подстраивающиеся под тип объекта.

В зависимости от типа хранимого объекта — битовой матрицы, пиктограммы, метафайла, определены соответствующие свойства **Bitmap, Icon или Metafile,** указывающие на графический объект. При ошибочном обращении к этим свойствам (если объект в действительности имеет другой тип) прежнее содержимое объекта стирается и открывается новый пустой объект указанного типа. Вместо этих свойств можно получать доступ к графическому объекту непосредственно через свойство **Graphic.** Таким образом, например, обращения **Imagel->Picture-> Graphic и Imagel->Picture->Bitmap** эквивалентны, если графический объект - битовая матрица.

Обмен объекта **TPicture** с буфером обмена Clipboard может осуществляться методом Assign объекта **TClipboard.**

Свойства
В классе **TPicture** объявлены следующие свойства:

Свойство	Объявление / Описание
Bitmap	**TBitmap*** Bitmap Указывает на хранящийся объект как на битовую матрицу (формат файла .bmp)
Graphic	TGraphic* Graphic Указывает на хранящийся объект как на битовую матрицу, пиктограмму, метафайл или определенный пользователем тип
Height	**int** Height Указывает собственную, не измененную высоту изображения в пикселах. Свойство только для чтения
Icon	**TIcon*** Icon Указывает на хранящийся объект как на пиктограмму (формат файла .ico)
Metafile	TMetafile* Metafile Указывает на хранящийся объект как на метафайл (формат файла .emf)
PictureAdapter	_di_IChangeNotifier PictureAdapter Интерфейс OLE для изображения. Используется только для внутренних целей
Width	int Width Указывает собственную, не измененную ширину изображения в пикселах. Свойство только для чтения

Методы
Ниже приведены основные методы, объявленные в классе **TPicture.**

Метод	**Объявление / Описание**
Assign	**void Assign(Classes::TPersistent* Source)** **Копирует один графический объект в другой. Выполняемые действия зависят от вида копируемого объекта**
LoadFrom ClipboardFormat	**void LoadFromClipboardFormat(Word AFormat, int AData, HPALETTE APalette)** **Читает изображение из буфера обмена Clipboard в заданном формате**
LoadFromFile	**void LoadFromFile(const AnsiString Filename)** **Читает изображение из файла FileName**
Register ClipboardFormat	**void RegisterClipboardFormat(System::TMetaClass* vmt, Word AFormat, System::TMetaClass* AGraphicClass)** **Регистрирует новый формат Clipboard графического объекта для использования в методе LoadFromClipboardFormat**

Метод	Объявление / Описание
RegisterFile Format	void RegisterFileFormat(System::TMetaClass* vmt, const AnsiString AExtension, const AnsiString ADescription, System::TMetaClass* AGraphicClass) Регистрирует новый файловый формат графического объекта для использования в методе LoadFromFile
RegisterFile FormatRes	void RegisterFileFormatRes(System::TMetaClass* vmt, const AnsiString AExtension, int ADescriptionResID, System::TMetaClass* AGraphicClass) Регистрирует новый файловый формат графического объекта из ресурсов для использования в методе LoadFromFile.
SaveTo ClipboardFormat	void SaveToClipboardFormat(Word &AFormat, int &AData, HPALETTE &APalette) Сохраняет изображение в буфере обмена Clipboard в заданном формате
SaveToFile	void SaveToFile(const AnsiString Filename) Сохраняет изображение в файле FileName
Supports ClipboardFormat	bool SupportsClipboardFormat(System::TMetaClass* vmt, Word AFormat) Определяет, поддерживается ли указанный формат Clipboard графического объекта
Unregister GraphicClass	void UnregisterGraphicClass(System::TMetaClass* vmt, System::TMetaClass* AClass) Удаляет все ссылки на ранее зарегистрированный формат Clipboard или файла

Кроме того, наследуется множество методов от класса TObject.

События

Событие	Описание
OnChange	Событие при изменении графического объекта
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

TPoint — тип

Определяет точку координат в пикселах.

Заголовочный файл *Types.hpp.*

Определение

```
struct TPoint : public POINT
{
  TPoint() {}
  TPoint(int _x, int _y) { x=_x; y=_y; }
  TPoint(POINT& pt)
  {
    x = pt.x;
```

```
    y = pt.y;
  }
};

typedef TPoint tagPoint;
```

Описание

Тип TPoint определяет точку координат в пикселах. В частности, во многих свойствах этот тип используется для задания координат углов прямоугольников и других фигур. В этих случаях началом координат, в зависимости от применения, считается левый верхний угол экрана или окна.

Для задания параметров типа **TPoint** во многих функциях удобно использовать функцию **Point.** возвращающую структуру **TPoint.** Но можно для создания точки использовать и объявленные в структуре конструкторы.

Примеры

Операторы

```
TPoint Tp = Point(Left, Top);
TPoint * Ptp = new TPoint(Left, Top);
TPoint Tp2 = TPoint(Left, Top);
TPoint Tp3 = TPoint(*Ptp);
```

создают переменные **Tp,** Tp2 и Tp3 типа **TPoint,** а также указатель **Ptp** на тип **TPoint,** причем во всех точках задаются значения координат, соответствующие левому верхнему углу формы. Третий и четвертый операторы используют две объявленные в структуре формы конструктора.

Следующий оператор сдвигает точку Tp на 10 пикселов вправо:

```
Tp.x += 10;
```

TRect — тип

Определяет координаты прямоугольной области.

Заголовочный файл *Types.hpp.*

Определение

```
struct TRect : public RECT
{
  TRect() {}
  TRect (const TPoint& TL, const TPoint& BR)
              {left=TL.x; top=TL.y; right=BR.x; bottom=BR.y;}
  TRect(int 1, int t, int r, int b)
              {left=l;    top=t;    right=r;    bottom=b;}
  TRect(RECT& r)
  {
    left    = r.left;
    top     = r.top;
    right   = r.right;
    bottom  = r.bottom;
  }
  int Width () const { return right  - left; }
  int Height() const { return bottom - top ; }
  bool operator ==(const TRect& rc) const
  {
     return left ==  rc.left  && top==rc.top &&
            right == rc.right && bottom==rc.bottom;
  }
  bool operator != (const TRect& rc) const
  { return !(rc==*this); }
  __property LONG Left    = { read=left, write=left   };
  __property LONG Top     = { read=top, write=top };
```

```
    __property LONG Right   = { read=right,  write=right  };
    __property LONG Bottom  = { read=bottom, write=bottom };
};
```

Описание

Доступ к координатам осуществляется или через данные-элементы **left, top, right, bottom,** или через свойства **Left, Top, Right, Bottom.** Для типа определены функции-элементы **Width() и Height(),** возвращающие соответственно ширину и высоту. Определены также операции " " и "!=".

Задавать значения переменной типа **TRect** удобно функцией **Rect.** задающей координаты левой, верхней, правой и нижней границ — **Left, Top, Right, Bottom,** или функцией **Bounds.** задающей координаты левого верхнего угла **Left** и **Top,** ширину прямоугольника **AWidth** и его высоту **AHeight.** Но с равным успехом можно использовать и объявленные в структуре конструкторы.

Примеры

Операторы

```
TRect Tr = Rect(Left, Top, Left+Width, Top+Height);
TRect Tr1 = Bounds(Left, Top, Width, Height);
TRect * Ptr = new TRect(BoundsRect);
TPoint P1 = Point(Left, Top);
TPoint P2 = Point(BoundsRect.Right, BoundsRect.Bottom);
TRect Tr2 = TRect(P1, P2);
TRect Tr3 = TRect(*Ptr);
```

создают переменные **Tr, Tr1,** Тг2 и **Tr3** типа **TRect,** а также указатель **Ptr** на тип **TRect,** причем во всех структурах задаются значения координат, соответствующие прямоугольнику **BoundsRect** формы. Переменные **Tr** и **Tr1** создаются функциями **Rect и Bounds.** Переменная Тг2 формируется конструктором из двух точек типа **TPoint.** А переменная Тг3 создается конструктором, воспринимающим координаты области, на которую указывает **Ptr.**

Следующий оператор сдвигает область **Tr** на 10 пикселов вправо:

```
Tr.Left += 10;
```

TRegIniFile — класс

Тип объекта, позволяющего работать с реестром Windows так, как с файлом .INI.

Иерархия *TObiect — TRegistru — TRegIniFile*

Модуль *Registry.*

Описание

Тип TReglniFile введен в C++Builder прежде всего, чтобы обеспечить наиболее простой переход от 16-разрядных приложений для Windows 3.x к 32-разрядным приложениям Windows, а точнее от работы с файлами .INI к работе с реестром. В приложениях для Windows 3.x регистрация приложения и сохранение его настроек осуществлялось с помощью файлов .INI. Для работы с этими файлами использовались объекты типа **TIniFile.** В 32-разрядных Windows для тех же целей используется системный реестр — registry. Перейти в приложениях, работающих с файлами .INI, к работе с реестром очень просто: достаточно заменить объект типа **TIniFile** объектом типа **TReglniFile.** При этом вся логика работы и все методы, использованные в приложении, сохраняются прежними.

TReglniFile наследует свойства и методы класса **TRegistry.** который может использоваться самостоятельно, но добавляет свойства и методы, характерные для объекта **TIniFile.**

Свойство **FileName,** которое в классе TIniFile обозначало имя файла, в классе **TReglniFile** соответствует субключу корневого ключа реестра (по умолчанию -

HKEY_CURRENT_USER). Разделу файла .INI соответствует ключ реестра, а значению ключа в файле .INI — параметр ключа реестра.

Имеется еще один класс — **TRegistryIniFile**, который обеспечивает другой способ доступа к реестру. Но поскольку **TRegistrylniFile** наследует классу **TCustomIniFile**, от которого наследует и **TIniFile, то TRegistrylniFile** еще ближе к **TIniFile** и дает еще более простой способ перехода от файлов .INI к реестру.

Подробнее работа с реестром и файлами .INI описана в [1] и [3].

Свойства

Ниже приведены свойства класса **TRegIniFile.**

Свойство	Объявление / Описание
Access	long Access Множество флагов, определяющих уровень доступа к открываемому ключу
CurrentKey	HKEY CurrentKey Текущий открытый ключ. Все операции с TReglniFile влияют только на открытый ключ реестра. Если вы хотите открыть новый ключ, делайте это методами OpenKey или OpenKeyReadOnly. Свойство только для чтения
CurrentPath	AnsiString CurrentPath Путь по реестру от корня до текущего ключа включительно. Свойство только для чтения
FileName	AnsiString FileName Субключ корневого ключа реестра (по умолчанию – HKEY_CURRENT_USER)
LazyWrite	bool LazyWrite Определяет способ записи ключей в реестр при выполнении CloseKey. Если LazyWrite = true (по умолчанию), то процедура CloseKey может вернуться до завершения записи. В противном случае процедура ожидает завершения записи. Это снижает производительнось
RootKey	HKEY RootKey Определяет корневой ключ. По умолчанию — HKEY_CURRENT_USER. Если это значение требуется изменить (например, для процедур LoadKey или RegistryConnect), надо задать в качестве RootKey допустимое целое значение

Методы

Ниже приведен список основных методов, определенных в **TReglniFile.**

Метод	Объявление / Описание
CloseKey	void CloseKey(void) Записывает в реестр текущий ключ и закрывает его. Способ записи определяется свойством LazyWrite Не следует держать ключ открытым дольше, чем это необходимо. Многие методы вызывают CloseKey автоматически после чтения или записи в реестр

Метод	Объявление / Описание
CreateKey	bool CreateKey(const AnsiString Key) Добавляет в реестр новый ключ Key
DeleteKey	HIDESBASE void DeleteKey(const AnsiString Section, const AnsiString Ident) Удаляет значение параметра Ident ключа Section
DeleteValue	bool DeleteValue(const AnsiString Name) Удаляет параметр Name текущего ключа. В случае успешного удаления возвращает true
EraseSection	void EraseSection(const AnsiString Section) Удаляет ключ Section со всеми его субключами

GetDataInfo	enum TRegDataType { rdUnknown, rdString, rdExpandString, rdInteger, rdBinary };
	struct TRegDataInfo
	{
	TRegDataType RegData;
	int DataSize;
	};
	bool GetDataInfo(const AnsiString ValueName, TRegDataInfo &Value)
	Дает информацию о параметрах, заданных ValueName, связанных с текущим ключом реестра данного объекта типа TRegistry. Информация содержится в записи типа TRegDataInfo, возвращаемой как значение параметра Value

GetDataSize	int GetDataSize(const AnsiString ValueName) Возвращает размер в байтах значения параметра ValueName текущего ключа. Если параметр является строкой, то возвращенное значение учитывает нулевой завершающий символ. Если выполнение функции завершается ошибкой, возвращается -1
GetDataType	enum TRegDataType { rdUnknown, rdString, rdExpandString, rdInteger, rdBinary }; TRegDataType GetDataType(const AnsiString ValueName) Возвращает характеристику типа параметра ValueName текущего ключа. Информация возвращается как значение перечислимого типа TRegDataType

GetKeyInfo	struct TRegKeyInfo
	{
	int NumSubKeys;
	int MaxSubKeyLen;
	int NumValues;
	int MaxValueLen;
	int MaxDataLen;
	_FILETIME FileTime;
	};
	bool GetKeyInfo(TRegKeyInfo &Value)
	Возвращает информацию о текущем ключе в запись типа TRegKeyInfo

Метод	Объявление / Описание
GetValue Names	void GetValueNames(Classes::TStrings* Strings) Возвращает список Strings имен всех субключей текущего ключа
HasSubKeys	bool HasSubKeys(void) Возвращает true, если текущий ключ имеет хотя бы один субключ
KeyExists	bool KeyExists(const AnsiString Key) Определяет, имеется ли в реестре ключ Key
LoadKey	bool LoadKey(const AnsiString Key, const AnsiString FileName) Создает ключ Key и загружает в него данные из указанного файла FileName
MoveKey	void MoveKey(const AnsiString OldName, const AnsiString NewName, bool Delete) Копирует (при Delete = false) или перемещает (при Delete = true) ключ OldName, давая ему новое имя NewName. Копируются или перемещаются также все субключи и параметры ключа
OpenKey	bool OpenKey(const AnsiString Key, bool CanCreate) Открывает ключ Key, делая его текущим для объекта. Если Key равен NULL, то текущим делается корневой ключ, указанный свойством RootKey. Параметр CanCreate указывает, должен ли создаваться ключ Key, если его нет в реестре. Ключ открывается или создается с доступом KEY_ALL_ACCESS. Создаваемый ключ сохраняется в реестре при последующих запусках системы
ReadBinary Data	int ReadBinaryData(const AnsiString Name, void *Buffer, int BufSize) Читает двоичные данные параметра Name текущего ключа в буфер Buffer размера BufSize. Подобные данные обычно являются записями. Это может быть и графика, хотя Microsoft возражает против хранения графики в реестре. При ошибке чтения генерируется исключение
ReadBool	HIDESBASE bool ReadBool(const AnsiString Section, const AnsiString Ident, bool Default) Возвращает булево значение параметра Ident ключа Section. Default - значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
Read Currency	System::Currency ReadCurrency(const AnsiString Name) Возвращает значение типа Currency параметра Name текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadDate	System::TDateTime ReadDate(const AnsiString Name) Возвращает значение даты типа TDateTime параметра Name текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadDate Time	System::TDateTime ReadDateTime(const AnsiString Name) Возвращает значение даты и времени типа TDateTime параметра Name текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным

Метод	Объявление / Описание
ReadFloat	**double ReadFloat(const AnsiString Name)** Возвращает действительное значение параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadInteger	**HIDESBASE int ReadInteger(const AnsiString Section, const AnsiString Ident, int Default)** Возвращает целое значение параметра **Ident** ключа **Section. Default** - значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadSection	**void ReadSection(const AnsiString Section, Classes::TStrings* Strings)** Читает в **Strings** типа **TStrings** имена всех параметров ключа **Section**
ReadSections	**void ReadSections(Classes::TStrings* Strings)** Читает в **Strings** типа **TStrings** имена всех ключей
ReadSection Values	**void ReadSectionValues(const AnsiString Section, Classes::TStrings* Strings)** Читает в **Strings** типа **TStrings** значения всех параметров ключа **Section**
ReadString	**HIDESBASE AnsiString ReadString(const AnsiString Section, const AnsiString Ident, const AnsiString Default)** Возвращает строку значения параметра **Ident** ключа **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadTime	**TDateTime ReadTime(const AnsiString Name)** Возвращает значение времени типа **TDateTime** параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
Registry Connect	**bool RegistryConnect(const** AnsiString UNCName) Устанавливает связь с реестром другого компьютера. UNCName — имя удаленного компьютера в форме "\\имя". Если UNCName = NULL, устанавливается связь с локальным компьютером. Перед вызовом свойство **RootKey** надо задать равным HKEY_USERS или HKEY_LOCAL_MACHINE. При успешном соединении возвращается **true и RootKey** устанавливается равным корневому ключу
RenameValue	**void RenameValue(const AnsiString OldName, const AnsiString NewName)** Изменяет имя параметра текущего ключа с **OldName** на **NewName.** Параметр **OldName** должен существовать, а параметра NewName не должно быть до выполнения этого метода. В противном случае параметр останется неизмененным

Метод	Объявление / Описание
ReplaceKey	bool ReplaceKey(const AnsiString Key, const AnsiString FileName, const AnsiString BackUpFileName) Заменяет хранящуюся в файле BackUpFileName информацию о ключе Key, его субключах и параметрах аналогичной информацией из другого файла FileName
RestoreKey	bool RestoreKey(const AnsiString Key, const AnsiString FileName) Открывает ключ Key и заменяет его содержимое информацией, хранящейся в файле FileName
SaveKey	bool SaveKey(const AnsiString Key, const AnsiString FileName) Открывает ключ Key и сохраняет параметры его и всех его субключей в файле FileName
TRegIniFile	TRegIniFile(const AnsiString FileName); TRegIniFile(const AnsiString FileName, unsigned AAccess); Конструкторы. Свойство FileName становится равным параметру FileName. Если задан параметр AAccess, это значение записывается в свойство Access; в противном случае Access = KEY_ALL_ACCESS
UnLoadKey	bool UnLoadKey(const AnsiString Key) Удаляет из реестра информацию о субключах и параметрах ключа Key. При успешном завершении возвращает true. Перед вызовом метода надо установить RootKey в HKEY_USERS, или в HKEY_LOCAL_MACHINE, или вызвать RegistryConnect
ValueExists	bool ValueExists(const AnsiString Name) Возвращает true, если в текущем ключе имеется параметр Name
WriteBinary Data	void WriteBinaryData(const AnsiString Name, void *Buffer, int BufSize) Записывает двоичные данные из буфера Buffer размера BufSize в параметр Name текущего ключа. Подобные данные обычно являются записями. Длинные данные (более 2048 байтов) должны храниться в отдельных файлах, а в реестр заноситься только имя файла. Это касается прежде всего графики, т.к. Microsoft возражает против хранения графики в реестре. Если параметр Name не существует, он создается. При ошибке записи генерируется исключение и записи не происходит
WriteBool	HIDESBASE void WriteBool(const AnsiString Section, const AnsiString Ident, bool Value) Записывает булево значение Value в параметр Ident ключа Section
Write Currency	void WriteCurrency(const AnsiString Name, System::Currency Value) Записывает значение Value типа Currency в параметр Name текущего ключа. Если нужного параметра нет, он создается. При ошибке записи генерируется исключение

Метод	Объявление / Описание
WriteDate	void **WriteDate(const AnsiString** Name, System::**TDateTime** Value) Записывает значение даты Value типа TDateTime в параметр Name текущего ключа. Если нужного параметра нет, он создается. При ошибке записи генерируется исключение
WriteDate Time	void WriteDateTime(const AnsiString Name, **System::TDateTime** Value) Записывает значение даты и времени Value типа TDateTime в параметр Name текущего ключа. Если параметра Name нет, он создается. При ошибке записи генерируется исключение
Write ExpandString	void **WriteExpandString(const** AnsiString Name, const AnsiString Value) Записывает значение строки, содержащей нерасширенную ссылку на переменную окружения (например, "%PATH%") в параметр Name. Если параметра Name нет, он создается. При ошибке записи генерируется исключение
WriteFloat	void **WriteFloat(const** AnsiString Name, double Value) Записывает действительное значение Value в параметр Name текущего ключа. Если параметра Name нет, он создается. При ошибке записи генерируется исключение
WriteInteger	HIDESBASE void **WriteInteger(const** AnsiString **Section,** const AnsiString **Ident,** int Value) Записывает целое значение Value в параметр Ident ключа Section
WriteString	HIDESBASE void **WriteString(const** AnsiString Section, const AnsiString Ident, const AnsiString Value) Записывает строку Value в параметр Ident ключа Section
WriteTime	void **WriteTime(const** AnsiString Name, **System::TDateTime** Value) Записывает значение времени Value типа TDateTime в параметр Name. Если параметра Name нет, он создается. При ошибке записи генерируется исключение

События

Никаких событий в классе **TRegIniFile** не объявлено.

Примеры

Следующие операторы создают объект Reg типа **TReglniFile** и создают в реестре ключ *My Registry:*

```
#include "registry.hpp";

TReglniFile * Reg = new TRegIniFile("My Registry");
```

Теперь созданный ключ *My Registry* стал текущим. Следующий оператор заносит в него параметр *Version* и его значение - номер версии 1:

```
Reg->WriteInteger("", "Version",1);
```

Следующий оператор создает в ключе *My Registry* субключ *Font* с параметром *Name,* в котором сохраняет имя шрифта, используемого в форме **Form1:**

```
Reg->WriteString("Font", "Font Name", Form1->Font->Name);
```

Следующий оператор уничтожает ранее созданный ключ *My Registry* со всеми его параметрами и субключами:

```
Reg->EraseSection("");
```

Следующий оператор уничтожает объект Reg:

```
delete Reg;
```

TRegistry — класс

Тип объекта, позволяющего работать с реестром Windows.

Иерархия *TObiect*

Модуль *Registry.*

Описание

Класс **TRegistry** инкапсулирует доступ к системному реестру Windows. Реестр, используемый в Windows 95/NT и старше вместо файлов .INI, характерных для Windows 3.x, хранит информацию о конфигурации в виде иерархического дерева, каждый узел которого называется ключом. Ключ может содержать субключи и значения параметров.

Все ключи в приложении создаются как субключи предопределенного корневого ключа (по умолчанию HKEY_CURRENT_USER).

Объект типа **TRegistry** имеет доступ только к одному текущему ключу, задаваемому свойством **CurrentKey.** Методы TRegistry позволяют открывать, закрывать, сохранять, перемещать, копировать и удалять ключи. Можно также узнавать, имеют ли ключи параметры, читать параметры и записывать параметры в ключ.

См. также **TRegIniFile и TRegistryIniFile**. Подробнее работа с реестром и файлами .INI описана в [1] и [3].

Свойства

Ниже приведены свойства класса **TRegistry.**

Свойство	Объявление / Описание
Access	long Access Множество флагов, определяющих уровень доступа к открываемому ключу
CurrentKey	HKEY CurrentKey Текущий открытый ключ. Все операции с TReglniFile влияют только на открытый ключ реестра. Если вы хотите открыть новый ключ, делайте это методами OpenKey или OpenKeyReadOnly. Свойство только для чтения
CurrentPath	AnsiString CurrentPath Путь по реестру от корня до текущего ключа включительно. Свойство только для чтения
LazyWrite	**bool** LazyWrite Определяет способ записи ключей в реестр при выполнении CloseKey. Если LazyWrite = **true** (по умолчанию), то процедура **CloseKey** может вернуться до завершения записи. В противном случае процедура ожидает завершения записи. Это снижает производительность

Свойство	Объявление / Описание
RootKey	HKEY RootKey Определяет корневой ключ. По умолчанию - HKEY_CURRENT_USER. Если это значение требуется изменить (например, для процедур **LoadKey** или **RegistryConnect**), надо задать в качестве RootKey допустимое целое значение

Методы

Ниже приведен список основных методов, определенных в **TRegistry**.

Метод	Объявление / Описание
CloseKey	void CloseKey(void) Записывает в реестр текущий ключ и закрывает его. Способ записи определяется свойством **LazyWrite** Не следует держать ключ открытым дольше, чем это необходимо. Многие методы вызывают CloseKey автоматически после чтения или записи в реестр
CreateKey	**bool CreateKey(const** AnsiString Key) Добавляет в реестр новый ключ Key
DeleteKey	HIDESBASE void **DeleteKey(const** AnsiString Section, const AnsiString **Ident)** Удаляет значение параметра **Ident** ключа Section
DeleteValue	bool **DeleteValue(const** AnsiString Name) Удаляет параметр Name текущего ключа. В случае успешного удаления возвращает **true**
GetDataInfo	enum TRegDataType { rdUnknown, rdString, rdExpandString, **rdInteger,** rdBinary **};** struct **TRegDataInfo** **{** TRegDataType RegData; int DataSize; **};** bool GetDataInfo(const AnsiString **ValueName,** TRegDatalnfo & Value) Дает информацию о параметрах, заданных ValueName, связанных с текущим ключом реестра данного объекта типа TRegistry. Информация содержится в записи типа TRegDatalnfo, возвращаемой как значение параметра Value
GetDataSize	int GetDataSize(const AnsiString ValueName) Возвращает размер в байтах значения параметра ValueName текущего ключа. Если параметр является строкой, то возвращенное значение учитывает нулевой завершающий символ. Если выполнение функции завершается ошибкой, возвращается -1
GetData Type	enum TRegDataType { rdUnknown, rdString, rdExpandString, rdlnteger, rdBinary }; TRegDataType GetDataType(const AnsiString ValueName) Возвращает характеристику типа параметра ValueName текущего ключа. Информация возвращается как значение перечислимого типа TRegDataType

Метод	Объявление / Описание

GetKeyInfo	struct **TRegKeyInfo**
	{
	int **NumSubKeys;**
	int MaxSubKeyLen;
	int **NumValues;**
	int MaxValueLen;
	int MaxDataLen;
	_FILETIME **FileTime;**
	};
	bool GetKeyInfo(TRegKeyInfo &Value)
	Возвращает информацию о текущем ключе в запись типа TRegKeyInfo

GetKey Names	void **GetKeyNames(Classes::TStrings*** Strings) Возвращает список Strings имен всех субключей текущего ключа
GetValue Names	void **GetValueNames(Classes::TStrings*** Strings) Возвращает список Strings имен всех субключей текущего ключа
HasSubKeys	bool HasSubKeys(void) Возвращает true, если текущий ключ имеет хотя бы один субключ
KeyExists	bool KeyExists(const AnsiString Key) Определяет, имеется ли в реестре ключ Key
LoadKey	bool LoadKey(const AnsiString Key, const AnsiString **FileName)** Создает ключ Key и загружает в него данные из указанного файла FileName
MoveKey	void MoveKey(const **AnsiString** OldName, const AnsiString **NewName,** bool Delete) Копирует (при Delete = false) или перемещает (при Delete = true) ключ OldName, давая ему новое имя NewName. Копируются или перемещаются также все субключи и параметры ключа
OpenKey	bool OpenKey(const AnsiString Key, bool CanCreate) Открывает ключ Key, делая его текущим для объекта. Если Key равен NULL, то текущим делается корневой ключ, указанный свойством RootKey. Параметр CanCreate указывает, должен ли создаваться ключ Key, если его нет в реестре. Ключ открывается или создается с доступом KEY_ALL_ACCESS. Создаваемый ключ сохраняется в реестре при последующих запусках системы
OpenKey Readonly	bool OpenKeyReadOnly(const AnsiString Key) Открывает ключ Key только для чтения, делая его текущим для объекта. Если Key равен NULL, то текущим делается корневой ключ, указанный свойством RootKey. Ключ открывается с доступом KEY_READ. Создаваемый ключ сохраняется в реестре при последующих запусках системы
ReadBinary Data	int ReadBinaryData(const AnsiString Name, void *Buffer, int BufSize) Читает двоичные данные параметра Name текущего ключа в буфер Buffer размера BufSize. Подобные данные обычно являются записями. Это может быть и графика, хотя Microsoft возражает против хранения графики в реестре. При ошибке чтения генерируется исключение

Метод	Объявление / Описание
ReadBool	**bool ReadBool(const AnsiString Name)** Возвращает булево значение параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
Read Currency	**System::Currency ReadCurrency(const AnsiString Name)** Возвращает значение типа **Currency** параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadDate	**System::TDateTime ReadDate(const AnsiString Name)** Возвращает значение даты типа **TDateTime** параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadDate Time	**System::TDateTime ReadDateTime(const AnsiString Name)** Возвращает значение даты и времени типа **TDateTime** параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadFIoat	**double ReadFloat(const AnsiString Name)** Возвращает действительное значение параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadInteger	**int ReadInteger(const AnsiString Name)** Возвращает целое значение параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadString	**AnsiString ReadString(const AnsiString Name)** Возвращает строку значения параметра **Name** текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
ReadTime	**TDateTime ReadTime(const AnsiString Name)** Возвращает значение времени типа **TDateTime** параметра Name текущего ключа. При ошибке чтения генерируется исключение и возвращенное значение является ошибочным
Registry Connect	**bool RegistryConnect(const AnsiString UNCName)** Устанавливает связь с реестром другого компьютера. UNCName — имя удаленного компьютера в форме "\\имя". Если UNCName = **NULL,** устанавливается связь с локальным компьютером. Перед вызовом свойство **RootKey** надо задать равным HKEY_USERS или HKEY_LOCAL_MACHINE. При успешном соединении возвращается **true и RootKey** устанавливается равным корневому ключу
Rename Value	**void RenameValue(const AnsiString OldName, const AnsiString NewName)** Изменяет имя параметра текущего ключа с **OldName** на NewName. Параметр **OldName** должен существовать, а параметра New**Name** не должно быть до выполнения этого метода. В противном случае параметр останется неизмененным

Метод	Объявление / Описание
ReplaceKey	**bool ReplaceKey(const AnsiString** Key, const AnsiString FileName, const AnsiString BackUpFileName) Заменяет хранящуюся в файле BackUpFileName информацию о ключе Key, его субключах и параметрах аналогичной информацией из другого файла FileName
RestoreKey	bool RestoreKey(const AnsiString Key, const AnsiString FileName) Открывает ключ Key и заменяет его содержимое информацией, хранящейся в файле FileName
SaveKey	bool SaveKey(const AnsiString Key, const AnsiString FileName) Открывает ключ Key и сохраняет параметры его и всех его субключей в файле FileName
TRegistry	TRegistry(void); **TRegistry(long AAccess);** Конструкторы. Устанавливают RootKey в HKEY_CURRENT_USER и Lazy Write в true. Если задан параметр AAccess, это значение записывается в свойство Access; в противном случае Access = KEY_ALL_ACCESS
UnLoadKey	bool **UnLoadKey(const** AnsiString Key) Удаляет из реестра информацию о субключах и параметрах ключа Key. При успешном завершении возвращает **true**. Перед вызовом метода надо установить RootKey в HKEY_USERS, или в HKEY_LOCAL_MACHINE, или вызвать RegistryConnect
ValueExists	bool ValueExists(const AnsiString Name) Возвращает true, если в текущем ключе имеется параметр Name
Write **BinaryData**	void WriteBinaryData(const AnsiString Name, void *Buffer, int BufSize) Записывает двоичные данные из буфера Buffer размера BufSize в параметр Name текущего ключа. Подобные данные обычно являются записями. Длинные данные (более 2048 байтов) должны храниться в отдельных файлах, а в реестр заноситься только имя файла. Это касается прежде всего графики, т.к. Microsoft возражает против хранения графики в реестре. Если параметр Name не существует, он создается. При ошибке записи генерируется исключение и записи не происходит
WriteBool	void WriteBool(const AnsiString Name, bool Value) Записывает булево значение Value в параметр Name текущего ключа. Если параметр Name не существует, он создается. При ошибке чтения генерируется исключение и записи не происходит
Write Currency	void WriteCurrency(const AnsiString Name, System::Currency Value) Записывает значение Value типа Currency в параметр Name текущего ключа. Если нужного параметра нет, он создается. При ошибке записи генерируется исключение

Метод	Объявление / Описание
WriteDate	**void WriteDate(const AnsiString Name, System::TDateTime Value)** Записывает значение даты **Value** типа **TDateTime** в параметр **Name** текущего ключа. Если нужного параметра нет, он создается. При ошибке записи генерируется исключение
WriteDateTime	**void WriteDateTime(const AnsiString Name, System::TDateTime Value)** Записывает значение даты и времени **Value** типа **TDateTime** в параметр **Name** текущего ключа. Если параметра **Name нет, он** создается. При ошибке записи генерируется исключение
WriteExpandString	**void WriteExpandString(const AnsiString Name, const AnsiString Value)** Записывает значение строки, содержащей нерасширенную ссылку на переменную окружения (например, "%PATH%") в параметр **Name.** Если параметра Name нет, он создается. При ошибке записи генерируется исключение
WriteFloat	**void WriteFloat(const AnsiString Name, double Value)** Записывает действительное значение **Value** в параметр **Name** текущего ключа. Если параметра Name нет, он создается. При ошибке записи генерируется исключение
WriteInteger	**void WriteInteger(const AnsiString Name, int Value)** Записывает целое значение **Value** в параметр **Name** текущего ключа. Если параметра **Name** нет, он создается. При ошибке записи генерируется исключение
WriteString	**void WriteString(const AnsiString Name, const AnsiString Value)** Записывает строку **Value** в параметр **Name** текущего ключа. Если параметра **Name** нет, он создается. При ошибке записи генерируется исключение
WriteTime	**void WriteTime(const AnsiString Name, System::TDateTime Value)** Записывает значение времени **Value** типа **TDateTime** в параметр **Name.** Если параметра **Name** нет, он создается. При ошибке записи генерируется исключение

События

Никаких событий в классе **TRegistry** не объявлено.

Примеры

Следующие операторы создают объект Reg типа **TRegistry** и создают в реестре ключ *My Registry:*

```
#include "registry.hpp";
TRegistry *Reg = new TRegistry;
```

Корневым и текущим ключом созданного объекта является HKEY_CURRENT_USER. Если надо работать с другим корнем, например, с HKEY_LOCAL_MACHINE, это можно сделать оператором:

```
Reg->RootKey = HKEY_LOCAL_MACHINE;
```

Следующий оператор создает в реестре и открывает (делает текущим) ключ *My Registry,* являющийся субключом текущего ключа:

```
Reg->OpenKey("My Registry",true);
```

Такой оператор создаст субключ корня, если есть уверенность, что в данный момент текущим является корневой ключ. Если такой уверенности нет, можно изменить оператор следующим образом:

```
Reg->OpenKey("\My Registry",true);
```

В этом случае отсчет ведется от корня и *My Registry* будет создан как субключ корневого ключа.

Следующий оператор заносит в ключ *My Registry* параметр *Version и* его значение — номер версии 1.0:

```
Reg->WriteString("Version", "1.0");
```

Следующие операторы создают в текущем ключе (после выполнения предыдущих операторов это ключ *My Registry)* субключ *Font* с параметром *Name,* в котором сохраняется имя шрифта, используемого в форме **Form1**:

```
Reg->OpenKey("Font",true);
Reg->WriteString("Name",Form1->Font->Name);
```

Приведенный выше оператор, открывающий ключ, может быть заменен на следующий:

```
Reg->OpenKey("\My Registry\Font",true);
```

Следующий оператор удаляет ранее созданный субключ корня *My Registry:*

```
Reg->DeleteKey("\My Registry");
```

TRegistryIniFile — класс

Тип объекта, позволяющего работать с реестром Windows так, как с файлом .INI.

Иерархия *TObiect — TCustomIniFile*

Модуль *Registry*.

Описание

Класс **TRegistrylniFile** введен в C++Builder прежде всего, чтобы обеспечить наиболее простой переход от 16-разрядных приложений для Windows 3.x к приложениям Windows 95/NT и старше, а также от работы с файлами .INI к работе с реестром. В приложениях для Windows 3.x регистрация приложения и сохранение его настроек осуществлялось с помощью файлов .INI. Для работы с этими файлами использовались объекты типа **TIniFile.** В 32-разрядных Windows для тех же целей используется системный реестр.

TRegistrylniFile наследует свойства и методы класса **TCustomlniFile,** которому наследует и **TIniFile.** Поэтому перейти в приложениях, работающих с файлами .INI, к работе с реестром очень просто: достаточно заменить объект типа **TIniFile** объектом типа **TRegistrylniFile.** При этом логика работы и все методы, использованные в приложении, сохраняются прежними.

В конструктор объекта типа **TRegistrylniFile** передается имя создаваемого ключа. Он создается как субключ системного ключа Hkey_Current_User. В последующих операторах чтения и записи на этот ключ следует ссылаться в параметре **Section** просто пустой строкой. Если же в операторе записи указать другое значение **Section,** то тем самым будет создан субключ того ключа, для которого создавался объект. Если *в* параметре Section перечислить несколько имен, разделяя их символом "\", то будет создана цепочка вложенных друг в друга ключей.

Имеется еще один класс — **TReglniFile.** который обеспечивает другой способ доступа к реестру. Но поскольку **TReglniFile** наследует классу **TRegistry, а не TCustomIniFile,** то переход от работы с файлами .INI к работе с реестром при использовании **TReglniFile** несколько менее формальный и простой. Впрочем, он открывает более широкие возможности использования реестра. Но полностью использовать все возможности реестра можно только с помощью объекта типа **TRegistry.**

Подробнее работа с реестром и файлами .INI описана в [1] и [3].

Свойства

Ниже приведены свойства класса **TRegistryIniFile.**

Свойство	Объявление / Описание
FileName	AnsiString FileName Имя ключа системного реестра, который открывается или создается и является корневым для объекта TRegistrylniFile. Свойство только для чтения
RegIniFile	TReglniFile* ReglniFile Это свойство используется только внутри объекта для создания интерфейса между методами TRegistrylniFile и реестром. Свойство только для чтения

Методы

Ниже приведен список основных методов, определенных в **TRegistrylniFile.**

Метод	**Объявление / Описание**
DeleteKey	**void DeleteKey(const AnsiString Section, const AnsiString Ident)** Удаляет значение параметра **Ident** ключа **Section**
EraseSection	**void EraseSection(const AnsiString Section)** Удаляет ключ **Section** со всеми его субключами
ReadBool	**bool ReadBool(const AnsiString Section, const AnsiString Ident, bool Default)** Возвращает булево значение параметра **Ident** ключа **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadDate	**System::TDateTime ReadDate(const AnsiString Section, const AnsiString Name, System::TDateTime Default)** Возвращает значение даты типа **TDateTime** параметра **Name** ключа **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadDate Time	**System::TDateTime ReadDateTime(const AnsiString Section, const : AnsiString Name, System:: TDateTime Default)** Возвращает значение даты и времени типа **TDateTime** параметра **Name** ключа **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadFloat	**double ReadFloat(const AnsiString Section, const AnsiString Name, double Default)** Возвращает действительное значение параметра **Name** ключа **Section. Default** — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения

Метод	Объявление / Описание
ReadInteger	int **ReadInteger(const** AnsiString **Section,** const AnsiString **Ident,** int Default) Возвращает целое значение параметра Ident ключа Section. Default — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadSection	void ReadSection(const AnsiString Section, **Classes::TStrings*** Strings) Читает в Strings типа TStrings имена всех параметров ключа Section
ReadSections	void **ReadSections(Classes::TStrings*** Strings) Читает в Strings типа TStrings имена всех ключей
ReadSection Values	void ReadSectionValues(const AnsiString Section, Classes::TStrings* Strings) Читает в Strings типа TStrings значения всех параметров ключа Section
ReadString	AnsiString **ReadString(const AnsiString** Section, const AnsiString Ident, const AnsiString Default) Возвращает строку значения параметра Ident ключа Section. Default — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
ReadTime	System::TDateTime ReadTime(const AnsiString Section, const AnsiString Name, System::TDateTime Default) Возвращает значение времени типа TDateTime параметра Name ключа Section. Default — значение по умолчанию, возвращаемое, если не нашлось ключа, параметра или значения
SectionExists	**bool** SectionExists(const AnsiString Section) Возвращает **true,** если в реестре существует ключ Section
TRegistryIni File	TRegistryIniFile(const AnsiString **FileName);** TRegistryIniFile(const AnsiString FileName, long AAccess); Конструкторы объекта, создающие внутренний компонент класса **TRegIniFile.** Параметр FileName задает имя, передаваемое в свойство FileName этого внутреннего компонента, а AAccess передается в его свойство Access
UpdateFile	void **UpdateFile(void)** Очищает буфер и записывает файл на диск. Действует только на Windows 95, т.к. 32-разрядные Windows не используют буфер
ValueExists	bool ValueExists(const AnsiString Section, const AnsiString Ident) Возвращает **true,** если в реестре существует параметр Ident ключа Section
WriteBool	void **WriteBool(const** AnsiString Section, const AnsiString Ident, bool Value) Записывает булево значение Value в параметр Ident текущего ключа. Если параметр Ident не существует, он создается

Метод	Объявление / Описание
WriteDate	void WriteDate(const AnsiString Section, const AnsiString Name, System;:TDateTime Value) Записывает значение даты Value типа TDateTime в параметр Name ключа Section. Если нужного параметра нет, он создается
WriteDate Time	void WriteDateTime(const AnsiString Section, const AnsiString Name, System::TDateTime Value) Записывает значение даты и времени Value типа TDateTime в параметр Name ключа Section. Если параметра Name нет, он создается
WriteFloat	void WriteFloat(const AnsiString Section, const AnsiString Name, double Value) Записывает действительное значение Value в параметр Name ключа Section. Если параметра Name нет, он создается
WriteInteger	void WriteInteger(const AnsiString Section, const AnsiString Ident, int Value) Записывает целое значение Value в параметр Ident ключа Section. Если параметра Ident нет, он создается
WriteString	void WriteString(const AnsiString Section, const AnsiString Ident, const AnsiString Value) Записывает строку Value в параметр Ident ключа Section. Если параметра Ident нет, он создается
WriteTime	void WriteTime(const AnsiString Section, const AnsiString Name, System::TDateTime Value) Записывает значение времени Value типа TDateTime в параметр Name. Если параметра Name нет, он создается

События

Никаких событий в классе **TRegistryIniFile** не объявлено.

Примеры

Следующие операторы создают объект Reg типа **TRegistrylniFile** и создают в реестре субключ *My Registry* ключа HKEY_CURRENT_USER:

```
#include "registry.hpp";
...
TRegistrylniFile * Reg = new TRegistryIniFile("My Registry");
```

Следующий оператор заносит в ключ *My Registry* параметр *Version* и его значение — номер версии 1.0:

```
Reg->WriteString("","Version","1.0");
```

Следующий оператор создает в ключе *My Registry* субключ *Font* с параметром *Name,* в котором сохраняется имя шрифта, используемого в форме **Form1**:

```
Reg->WriteString("Font", "Font Name", Form1->Font->Name);
```

Следующий оператор удаляет в ключе *My Registry* параметр *Version:*

```
Reg->DeleteKey("", "Version");
```

Следующий оператор удаляет ранее созданный ключ *My Registry:*

```
Reg->EraseSection("");
```

TShiftState — тип

Определяет множество нажатых вспомогательных клавиш.

Заголовочный файл *Classes.hpp.*

Определение

```
enum Classes__1 (ssShift, ssAlt, ssCtrl, ssLeft, ssRight,
                 ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble>  TShiftState;
```

Описание

Тип **TShiftState** используется в обработчиках различных событий, связанных нажатием пользователем клавиш или кнопок мыши. Множество может содержать следующие значения, определяющие нажатые и удерживаемые в данный момент вспомогательные клавиши клавиатуры и кнопки мыши:

ssShift	нажата клавиша Shift
ssAlt	нажата клавиша Alt
ssCtrl	нажата клавиша Ctrl
ssLeft	нажата левая кнопка мыши
ssRight	нажата правая кнопка мыши
ssMiddle	нажата средняя кнопка мыши
ssDouble	произведен двойной щелчок

TSQLTimeStamp — тип записи даты и времени

Тип записи, используемый драйверами dbExpress для работы с датами и временем.

Заголовочный файл *SqlTimSt.hpp.*

Определение

```
struct TSQLTimeStamp
{
  short Year;
  Word Month;
  Word Day;
  Word Hour;
  Word Minute;
  Word Second;
  long Fractions;
};
```

Описание

Тип TSQLTimeStamp используют драйверы баз данных dbExpress для работы с датами и временем. Поле **Year** — год, который может принимать значения от 1 до 9999. Поле **Month** — месяц, от 1 до 12. Поле Day — день месяца (от 1 до 28, 29, 30 или 31 в зависимости от месяца). Поле **Hour** — час (от 0 до 23). Поля **Minute и Second** — минуты и секунды (от 0 до 59). Поле **Fractions** — миллисекунды (от О до 999).

Переменные типа **TSQLTimeStamp** можно создавать нецосредственно, занося в их поля требуемые значения. Например:

```
#include <SqlTimSt.hpp>
...
TSQLTimeStamp TSQL;
TSQL.Year = 2002;
```

```
TSQL.Month = 1;
TSQL.Day = 1;
```

Можно для занесения значений использовать многочисленные функции преобразования дат и времени. Например:

```
TSQL = DateTimeToSQLTimeStamp(Now());
```

Можно также с помощью функции **VarSQLTimeStampCreate** создавать объекты **Variant,** содержащие записи типа **TSQLTimeStamp.** Это наиболее простой способ манипулирования с записями типа **TSQLTimeStamp,** поскольку при этом можно использовать встроенные операции **Variant.**

TStringFloatFormat — тип

В ряде методов тип **TStringFloatFormat** определяет формат представления чисел строкой.

Определение

```
enum TStringFloatFormat {sffGeneral, sffExponent, sffFixed,
                         sffNumber, sffCurrency};
```

Различные значения формата означают следующее:

Значение	Описание
sffGeneral	Значение преобразуется в наиболее компактное из двух форматов: с фиксированной точкой или научного формата. Младшие нулевые разряды усекаются. Десятичная точка появляется только при необходимости. Формат с фиксированной точкой используется только при числе цифр целой части большем не больше указанной точности и при значениях не меньше 0.00001. В остальных случаях используется научный формат с минимальным числом цифр в степени порядка (от 0 до 4).
sffExponent	Научный формат. Значение преобразуется в строку вида "−d.ddd...E+dddd". Символ '−' записывается только для отрицательных чисел. Перед десятичной точкой записывается всегда одна цифра. Общее число цифр (включая цифру Перед точкой) определяется заданной точностью. После символа 'E' всегда ставится знак + или −. Число цифр в степени (порядок числа) лежит в пределах от 0 до 4.
sffFixed	Формат с фиксированной точкой. Значение преобразуется в строку вида "−ddd.ddd...". Символ '−' записывается только для отрицательных чисел. Перед десятичной точкой записывается пот крайней мере одна цифра. Число цифр поле точки определяется заданным числом разрядов (от 0 до 18). Если число цифр слева от точки должно быть больше заданной точности, используется научный формат.
sffNumber	Числовой формат. Значение преобразуется в строку вида "−d,ddd,ddd.ddd...". Совпадает с форматом **sffFixed** за исключением наличия разделителей после каждых трех разрядов в целой части.
sffCurrency	Монетарный формат для представления чисел, отображающих денежные суммы. Определяется установками Windows (глобальными переменными **CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator**). Число цифр после десятичной точки определяется заданным числом разрядов (от 0 до 18).

Для всех форматов действительные символы, используемые в качестве десятичной точки и разделителя тысяч определяются глобальными переменными **DecimalSeparator** и **ThousandSeparator.**

TStringList — класс

Список строк с расширенными возможностями манипулирования ими.

Иерархия *TObiect — TPersistent — TStrings*

Модуль *classes.*

Описание

Класс **TStringList** наследует классу **TStrings.** реализуя многие его абстрактные свойства и методы и вводя некоторые новые возможности:

- сортировку строк в списке
- запрещение хранения дубликатов строк
- реакцию на изменения содержания списка

Свойства

Ниже приведен список основных свойств, определенных в **TStringList.**

Свойство	Объявление / Описание
Capacity	int Capacity Указывает число строк, которые может содержать список, позволяет заранее выделить память для добавления нескольких строк
CaseSensitive	**bool** CaseSensitive Устанавливает чувствительность или нечувствительность к регистру при поиске, сортировке и т.п.
Count	int Count Число строк в списке. Свойство только для чтения
Duplicates	enum TDuplicates { **dupIgnore,** dupAccept, dupError }; TDuplicates Duplicates; Указывает, могут ли добавляться в сортированный список дубликаты строк. Значение duplgnore — игнорирование добавления дубликата, dupAccept — разрешение добавления дубликата, dupError — генерация исключения **EListError** при попытке добавления дубликата строки. Значения dupAccept и dupError никак не реагируют на уже имеющиеся в списке дубликаты. На несортированный список свойство Duplicates не оказывает никакого влияния
Objects	System::TObject* Objects[int Index] Возвращает объект, связанный со строкой с индексом Index свойства Strings
Sorted	bool Sorted Указывает, должны ли строки в списке автоматически сортироваться по алфавиту
Strings	AnsiString Stringsfint Index] Текст строки с указанным индексом. Индекс первой строки — 0

Кроме того, **TStringList** наследует от **TStrings** такие свойства, как **CommaText, DelimitedText, Delimiter, Names, StringsAdapter, QuoteChar, Text, Values.**

Методы

В классе **TStringList** объявлены или переопределены следующие методы:

Метод	Объявление / Описание
Add	int Add(const AnsiString S) Добавляет строку в конец списка. Возвращает индекс добавленной строки. В окнах редактирования добавляемая строка может оказаться разбитой на несколько, так что в этих случаях возвращаемый индекс ни о чем не говорит
Clear	void Clear(void) Очищает список
CustomSort	typedef int (CALLBACK *TStringListSortCompare) (TStringList* List, int Indexl, int Index2); fastcall CustomSort (TStringListSortCompare Compare) Обеспечивает нестандартную сортировку списка с помощью функции сравнения Compare
Delete	void Delete(int Index) Удаляет из списка строку с индексом Index
Exchange	void Exchange(int Indexl, int Index2) Переставляет местами строки списка с индексами Indexl и Index2
Find	bool Find(const AnsiString *S,* int &Index) Определяет, имеется ли заданная строка S в сортированном списке, и, если имеется, то возвращает в параметр Index индекс этой строки. Для несортированных списков следует использовать метод IndexOf
IndexOf	int IndexOf(const AnsiString S) Возвращает индекс указанной строки S. Если такой строки нет в списке, возвращается -1
Insert	void Insert(int Index, const AnsiString S) Вставляет указанную строку S в заданную позицию Index. Если Index = 0, строка вставляется на первую позицию
Sort	void Sort(void) Сортирует строки списка, свойство Sorted которого установлено в false, в возрастающей алфавитной последовательности. Если Sorted = true, то список сортируется автоматически

От класса TStrings наследуются методы **AddObject, AddStrings, Append, Assign, BeginUpdate, EndUpdate, Equals, GetText, IndexOfName, IndexOfObject, InsertObject, LoadFromFile, LoadFromStream, Move, SaveToFile, SaveToStream, SetText.** Кроме того, наследуется много методов от **TObject.**

Примеры

```
// Создание экземпляра списка
TStringList * TL = new TStringList;
TL->Sorted = true;          // Список сортированный
TL->Duplicates = dupError; // Запрет дубликатов
TL->Add("Петров");
int i = TL->Add("Иванов"); // i=0, т.к. список сортированный
TL->Add("Иванов");          // Генерация исключения из-за дубликата
```

Оператор

```
int i = TL->Add("Иванов");
```

присваивает переменной i значение 0, поскольку список сортированный и фамилия "Иванов" должна размещаться первой, раньше фамилии "Петров". Добавление в сортированный список дубликата вызывает генерацию исключения и сообщение: "String list does not allow dublicates".

TStrings — класс

Абстрактный класс объектов, представляющих собой списки строк и используемых во многих компонентах C++Builder в качестве различных свойств.

Иерархия *TObject — TPersistent*

Модуль *classes.*

Описание

Класс **TStrings** содержит методы и свойства, позволяющие манипулировать со списками строк:

- Добавлять и удалять строки в указанных позициях
- Перестраивать и упорядочивать последовательность строк
- Получать доступ к конкретным строкам
- Читать и записывать списки строк в файлы и потоки
- Связывать с каждой строкой некоторый объект

Свойства

Ниже приведен список свойств, определенных в **TStrings.**

Свойство	Объявление / Описание
Capacity	int Capacity Указывает число строк, которые может содержать список. В классе TStrings чтение Capacity возвращает значение Count, а запись значения Capacity ничего не изменяет в списке. Но в некоторых классах, производных от TStrings, свойство Capacity позволяет заранее выделить память для добавления нескольких строк
CommaText	AnsiString CommaText Возвращает текст, в котором отдельные строки объединены в одну строку формата SDF (system data format)
Count	int Count Число строк в списке. Свойство только для чтения
DelimitedText	AnsiString DelimitedText Возвращает текст, в котором отдельные строки выделены кавычками, определенными свойством QuoteChar, и, отделены друг от друга символом, указанным в свойстве Delimiter
Delimiter	char Delimiter Разделитель, используемый в свойстве DelimitedText

Свойство	Объявление / Описание
Names	AnsiString Names[int Index] Применяется для списков, имеющих структуру "Имя = Значение". Такую структуру имеют, например, файлы .ini. Свойство Names возвращает Имя, использованное в строке с указанным индексом. Если строка не имеет форму "Имя = Значение", возвращается пустая строка. Свойство только для чтения
Objects	System::TObject* Objects[int Index] Возвращает объект, связанный с указанной строкой свойства Strings. В классе TStrings свойство Objects не используется, но может использоваться в некоторых классах, производных от TStrings
QuoteChar	char QuoteChar Символ кавычек, используемый в свойстве DelimitedText
Strings	AnsiString Strings[int Index] Текст строки с указанным индексом. Индекс первой строки — 0
StringsAdapter	_di_IStringsAdapterStringsAdapter Реализует интерфейс IStrings, позволяющий объектам TStrings связываться с объектами OLE
Text	AnsiString Text Представляет весь список как одну строку, внутри которой используются разделители типа символов возврата каретки и перевода строки
Values	AnsiString Values[AnsiString Name] Применяется для списков, имеющих структуру "Имя = Значение". Такую структуру имеют, например, файлы .ini. Свойство Values возвращает Значение, в строке с указанным именем Name. Если заданное имя Name не найдено, возвращается пустая строка.

Методы

Ниже приведены основные методы, объявленные в классе **TStrings.**

Метод	Объявление / Описание
Add	int Add(const AnsiString S) Добавляет строку в конец списка. Возвращает индекс добавленной строки. В окнах редактирования добавляемая строка может оказаться разбитой на несколько, так что в этих случаях возвращаемый индекс ни о чем не говорит
AddObject	int AddObject(const AnsiString S, System::TObject* AObject) Добавляет в список строку и связанный с ней объект. Возвращает индекс добавленной строки и объекта
AddStrings	void AddStrings(TStrings* Strings) Добавляет в список группу строк из другого объекта Strings типа TStrings

Метод	Объявление / Описание
Append	void Append(const AnsiString S) Добавляет строку в конец списка. Метод аналогичен Add, но не возвращает индекс строки
Assign	void Assign(TPersistent* Source) Переносит строки из указанного объекта в данный
BeginUpdate	void BeginUpdate(void) Предотвращает перерисовку объекта до выполнения метода EndUpdate. Позволяет избежать мерцания при перестроении списка
Clear	void Clear(void) = 0 Очищает список
Delete	void Delete(int Index) = 0 Удаляет из списка указанную строку
EndUpdate	void EndUpdate(void) Завершает блокировку перерисовки объекта, заданную методом BeginUpdate
Equals	bool Equals(TStrings* Strings) Сравнивает данный список с заданным списком Strings. Возвращает true при идентичности списков
Exchange	void Exchange(int Indexl, int Index2) Переставляет местами строки списка с индексами Indexl и Index2
GetText	char * GetText(void) Возвращает буфер, под который выделяет память, и заполняет его значением свойства Text
IndexOf	int IndexOf(const AnsiString S) Возвращает индекс указанной строки S. Если такой строки нет в списке, возвращается –1
IndexOfName	int IndexOfName(const AnsiString Name) Применяется для списков, имеющих структуру "Имя = Значение". Такую структуру имеют, например, файлы .ini. Возвращается индекс строки, в которой имя равно заданному значению Name. Если такой строки нет в списке, возвращается -1
IndexOfObject	int IndexOfObiect(Svstem::TObiect* AObject) Возвращает индекс первой строки, связанной с заданным объектом. Если такой строки нет в списке, возвращается –1
Insert	void Insert(int Index, const AnsiString S) = 0 Вставляет указанную строку S в заданную позицию Index. Если Index = 0, строка вставляется на первую позицию

Метод	Объявление / Описание
InsertObject	void InsertObject(int Index, const AnsiString S, System::TObject* AObject) Вставляет указанную строку S в заданную позицию Index и связывает с ней объект AObject. Если Index = 0, строка вставляется на первую позицию
LoadFromFile	void LoadFromFile(const AnsiString FileName) Заполняет список строками текста из указанного файла FileName
LoadFrom Stream	void LoadFromStream(TStream* Stream) Заполняет список строками текста из указанного потока Stream
Move	void Move(int CurIndex, int NewIndex) Изменяет позицию строки с индексом CurIndex, давая ей индекс NewIndex
SaveToFile	void SaveToFile(const AnsiString FileName) Сохраняет строки списка в файле с указанным именем FileName
SaveToStream	void SaveToStream(TStream* Stream) Сохраняет значение свойства Text в указанном потоке Stream
SetText	void SetText(char * Text) Задает значение свойства Text

Кроме того, наследуется множество методов от класса **TObject.**

TSystemTime — тип записи даты и времени

Тип структуры, используемой в API Windows 32 для хранения данных о датах и времени.

Заголовочные файлы *Windows.hpp, winbase.h.*

Определение

```
typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME, *LPSYSTEMTIME;

typedef _SYSTEMTIME *PSystemTime;
typedef _SYSTEMTIME  TSystemTime;
```

Описание

Тип структуры **TSystemTime** используется в API Windows 32 для хранения данных о датах и времени. Этот тип, а также тип указателя **PSystemTime** на эту структуру используется при вызове функций API Windows для представления в них значения **SYSTEMTIME.** Для взаимного преобразования типов **TSystemTime и TDateTime** используются функции **DateTimeToSystemTime и SystemTimeToDateTime**.

TTimeStamp — тип записи даты и времени

Тип структуры, используемой для хранения данных о датах и времени.

Заголовочный файл *SysUtils.hpp.*

Определение

```
struct TTimeStamp
{
  int Time; // Число миллисекунд с начала дня
  int Date; // 1 + число дней с 1/1/0001
};
```

Описание

Тип структуры **TTimeStamp** используется в случаях, когда требуется повышенная точность отображения времени. Если не нужна точность на уровне миллисекунд, лучше использовать более компактный тип **TDateTime.** А если наоборот, требуется более высокая точность, то лучше использовать TSQLTimeStamp, но без присваивания переменной **Variant.**

В записи **TTimeStamp** поле **Time** содержит число миллисекунд текущего дня, отсчитанное от полуночи, а поле **Date** содержит число дней, отсчитанное с начала Григорианского календаря.

TWinControl — базовый класс оконных компонентов

Абстрактный базовый класс всех оконных компонентов.

Иерархия *TObiect — TPersistent — TComponent — TControl*

Модуль *controls.*

Описание

Класс **TWinControl** является базовым абстрактным классом для всех оконных компонентов Delphi, т.е. для компонентов, которые:

- Могут получать фокус во время выполнения приложения. Другие компоненты могут отображать данные, но пользователь не может общаться с компонентом с помощью клавиатуры, если это не оконный компонент.
- Могут содержать другие компоненты, т.е. быть компонентами-контейнерами, компонентами-родителями других, дочерних компонентов.
- Имеют дескрипторы окна.

Новые компоненты редко создаются непосредственно на основе **TWinControl.** Обычно они основываются на производных классах, таких, как TCustomControl, имеющий канву и обработку сообщений прорисовки, или на более специализированных классах типа **TButtonControl, TCustomComboBox, TCustomEdit** или TCustomListBox.

Свойства

Ниже приведен список основных свойств, определенных или переопределенных в **TWinControl.** Некоторые методы, используемые в основном при разработке сложных новых классов, в него не включены.

Свойство	Объявление / Описание
BevelEdges	enum TBevelEdge { **beLef t,** beTop, beRight, beBottom } ; typedef Set<TBevelEdge, beLeft, **beBottom**> TBevelEdges; TBevelEdges BevelEdges Определяет, какая граница компонента будет иметь обрамление. Является множеством, которое может содержать beLeft — левая, **beTop** — верхняя, beRight — правая, beBottom · — нижняя
BevelInner	enum TBevelCut { **bvNone,** bvLowered, bvRaised, bvSpace }; TBevelCut Bevellnner Определяет выпуклый (bvRaised), утопленный (bvLowered) или плоский (bvSpace) вид внутренней части компонента
BevelKind	enum TBevelKind { bkNone, bkTile, bkSoft, bkFlat }; **TBevelKind** BevelKind Определяет в комбинации с BevelWidth, Bevellnner и BevelOuter тип обрамления компонента
BevelOuter	enum TBevelCut { **bvNone,** bvLowered, bvRaised, bvSpace }; TBevelCut BevelOuter Определяет выпуклый (bvRaised), утопленный (bvLowered) или плоский (bvSpace) вид обрамления компонента
BevelWidth	typedef unsigned **TBevelWidth;** **TBevelWidth** BevelWidth Определяет ширину обрамления компонента в пикселах
BorderWidth	typedef unsigned **TBorderWidth;** TBorderWidth BorderWidth Определяет ширину бордюра компонента в пикселах
Brush	**Graphics::TBrush*** Brush Определяет цвет и стиль заполнения фона окна. Свойство только для чтения
ClientOrigin	Types::TPoint ClientOrigin Экранные координаты левого верхнего угла клиентской области компонента. Свойство только для чтения
ClientRect	Types::TRect ClientRect Содержит размер клиентской области компонента. Свойство только для чтения
ControlCount	int ControlCount Число дочерних компонентов данного оконного элемента. Свойство только для чтения
Controls	TControl* **Controls**[**int** Index] Массив дочерних компонентов оконного элемента. Свойство только для чтения
CtI3D	**bool** Ctl3D Определяет, будет ли компонент выглядеть объемным или плоским

Свойство	Объявление / Описание
DockClientCount	int DockClientCount Число компонентов, встроенных в данное окно
DockClients	TControl* DockClients[int Index] Массив компонентов, встроенных в данное окно
DockManager	_di_IDockManager DockManager Указывает на интерфейс диспетчера встраивания, который управляет процессами Drag&Doc. Диспетчер создается методом CreateDockManager. DockManager быть NULL, если DockSite или UseDockManager равно false
DockSite	bool DockSite Определяет, может ли компонент быть приемником в операциях Drag&Doc
DoubleBuffered	bool DoubleBuffered Определяет, будет ли изображение компонента рисоваться непосредственно в окне (при DoubleBuffered = false), или сначала заноситься в битовую матрицу в памяти. Последнее уменьшает эффекты мерцания при перерисовке изображения, но требует дополнительных затрат памяти
Handle	HWND Handle Дескриптор оконного элемента, используемый при вызове функций API Windows. Свойство только для чтения
ImeMode	TImeMode ImeMode Определяет режим работы входного редактора метода IME — (input method editor). IME — внешний входной процессор для Азиатских языковых символов IME — иероглифов, преобразующий их в код, пригодный для использования в приложениях C++Builder
ImeName	AnsiString ImeName Указывает входной редактор метода (IME) для преобразования данных, вводимых с клавиатуры в Азиатские языковые символы. IME — внешний входной процессор для Азиатских языковых символов. Значение ImeName определяет, какой IME используется для преобразования нажатий клавиши. IME перехватывает нажатия клавиш клавиатуры, преобразует их в Азиатские символы в конверсионном окне и посылает преобразованные символы в приложения C++Builder
ParentCtl3D	bool ParentCtl3D Управляет наследованием родительского свойства Ctl3D
ParentWindow	HWND ParentWindow Дескриптор родительского окна, не являющегося визуальным компонентом, например TActiveXControl. Если свойство Parent не NULL, то значение ParentWindow игнорируется
Showing	bool Showing Определяет, виден ли компонент в данный момент. Свойство только для чтения

Свойство	Объявление / Описание
TabOrder	typedef short TTabOrder; TTabOrder TabOrder Указывает позицию компонента в последовательности табуляции
TabStop	bool TabStop Определяет, может ли пользователь перевести фокус на компонент клавишей табуляции
UseDockManager	bool UseDockManager Указывает, используется ли диспетчер встраивания DockManager в процессе Drag&Doc
VisibleDock ClientCount	int VisibleDockClientCount Число видимых компонентов, встроенных в данное окно
WindowHandle	HWND WindowHandle То же, что Handle, но это свойство можно читать и изменять

Класс **TWinControl** наследует также много свойств своих предшественников **TControl и TComponent.**

Методы

Ниже приведены основные методы, наследуемые от **TWinControl** и используемые в компонентах — потомках этого класса.

Метод	Объявление / Описание
Broadcast	void Broadcast(void *Message) Посылает сообщение Message всем своим дочерним компонентам
CanFocus	bool CanFocus(void) Определяет, может ли компонент получать фокус, т.е. получать сообщения пользователя
CanResize	bool CanResize(int &NewWidth, int &NewHeight) Задает новые значения ширины и высоты компонента NewHeight и New Width и генерирует событие OnCanResize, в обработчике которого эти значения могут быть скорректированы
ChangeScale	void ChangeScale(int M, int D) Изменяет масштаб компонента и его дочерних компонентов
Constrained Resize	void ConstrainedResize(int &MinWidth, int &MinHeight, int &MaxWidth, int &MaxHeight) Вызывается автоматически для генерации события OnConstrainedResize, в которое передаются новые ограничения ширины и высоты компонента
ContainsControl	bool ContainsControl(TControl* Control) Определяет, является ли указанный компонент Control прямым или косвенным наследником данного оконного элемента

Метод	Объявление / Описание
ControlAtPos	TControl* ControlAtPos(const Types::TPoint &Pos, bool AllowDisabled, bool AllowWinControls = false) Возвращает дочерний компонент, находящийся в указанной позиции Pos
DisableAlign	void DisableAlign(void) Временно запрещает выравнивание компонентов в оконном элементе
DockDrop	void DockDrop(TDragDockObject* Source, int X, int Y) Генерирует событие OnDockDrop
EnableAlign	void EnableAlign(void) Отменяет действие предварительно вызванного метода DisableAlign и вызывает Realign для выравнивания компонентов
FindChildControl	TControl* FindChildControl(const AnsiString ControlName) Возвращает дочерний компонент, указанный его именем ControlName
FindNextControl	TWinControl* fastcall FindNextControl(TWinControl* CurControl, bool GoForward, bool CheckTabStop, bool CheckParent) Возвращает очередной оконный компонент в последовательности табуляции
Focused	bool Focused(void) Определяет, находится ли оконный элемент в фокусе
GetTabOrderList	void GetTabOrderList(Classes::TList* List) Строит список дочерних компонентов в последовательности табуляции
HandleAllocated	bool HandleAllocated(void) Проверяет наличие дескриптора окна компонента
HandleNeeded	void HandleNeeded(void) Создает дескриптор окна, если он до этого не существовал
Invalidate	void Invalidate(void) Сообщает о необходимости перерисовки компонентов
PaintTo	void PaintTo(HDC DC, int X, int Y) Рисует компонент в заданное устройство DC
Realign	void Realign(void) Выравнивает компоненты в оконном элементе
RemoveControl	void RemoveControl(TControl* AControl) Удаляет указанный компонент из списка Controls. Лучше использовать для этих целей само свойство Controls
Repaint	void Repaint(void) Перерисовывает изображение компонента на экране с помощью Invalidate

Метод	Объявление / Описание
ScaleBy	void ScaleBy(int M, int D) Масштабирует оконный элемент и все содержащиеся в нем компоненты в M / D раз
ScaleControls	void ScaleControls(int M, int D) Изменяет масштаб компонентов в оконном элемент в M / D раз, не изменяя масштаба самого оконного элемента
ScrollBy	void ScrollBy(int DeltaX, int DeltaY) Сдвигает содержимое оконного элемента
SelectFirst	void SelectFirst(void) Передает фокус дочернему компоненту, первому в последовательности табуляции
SelectNext	void SelectNext(TWinControl* CurControl, bool GoForward, bool CheckTabStop) Передает фокус компоненту, следующему или предыдущему в последовательности табуляции
SetBounds	void SetBounds(int ALeft, int ATop, int AWidth, int AHeight) Задает координаты и размеры элемента
SetChildOrder	void SetChildOrder(Classes::TComponent* Child, int Order) Изменяет позицию компонента в списке дочерних компонентов
SetFocus	void SetFocus(void) Передает фокус элементу (активизирует его)
SetZOrder ,	void SetZOrder(bool TopMost) Перемещает компонент на верх или в низ Z-последовательности
Update	void Update(void) Немедленная перерисовка компонента

Помимо перечисленных методов имеется еще немало методов, наследуемых от **TControl**.

События

Событие	Описание
OnEnter	Событие при получении элементом фокуса
OnExit	Событие при потере элементом фокуса
OnKeyDown	Событие при нажатии любой клавиши или кнопки мыши
OnKeyPress	Событие при нажатии клавиши символа
OnKeyUp	Событие при отпускании клавиши

WM_USER — константа

Константа используется приложениями для определения своих частных сообщений.

Описание

Константа **WM_USER** используется для разграничения номеров сообщений, резервированных для Windows, и частных сообщений оконных компонентов. Все возможные номера сообщений разделены на пять диапазонов:

от 0 до **WM_USER** - 1	Номера сообщений, используемые Windows
от **WM_USER** до **0x7FFF**	Номера частных сообщений внутри данного класса оконных компонентов
от 0x8000 до OxBFFF	Номера, зарезервированные для будущего использования в Windows
от OxCOOO до OxFFFF	Номера, соответствующие строкам сообщений, используемым для обмена между приложениями и зарегистрированным функцией **RegisterWindowMessage**
свыше OxFFFF	Номера, зарезервированные для будущего использования в Windows

Номера второго диапазона от **WM_USER** до **0x7FFF** могут использоваться для определения и посылки сообщений внутри данного класса оконных компонентов. Их нельзя использовать для определения сообщений, предназначенных для обмена между приложениями, поскольку некоторые предопределенные классы оконных компонентов (например, **TButton, TEdit, TListBox и TComboBox)** уже используют этот диапазон. Сообщения другим приложениям в этом диапазоне могут посылаться только в случае, если приложения спроектированы с учетом обмена данными сообщениями и одинаково понимают номера этих сообщений.

Глава 2

Компоненты C++Builder

В данной главе содержатся справочные сведения по компонентам C++Builder. В двух первых разделах рассматриваются вопросы взаимодействия некоторых видов компонентов. А последующие разделы, размещенные в алфавитном порядке, посвящены некоторым наиболее часто используемым компонентам C++Builder. В описании их свойств, методов и событий вы часто можете встретить идентификаторы, выделенные подчеркиванием. Например, **Caption.** Это означает, что в данной, предыдущей или последующих главах имеется подробное описание этого термина. Определения свойств и методов даются, как и в предыдущей главе, в сокращенной нотации.

В отношении идентификаторов, используемых для обозначения компонентов, данная книга следует правилу, принятому в C++Builder. Имена компонентов в данной и последующих главах даются без префикса "T", хотя все идентификаторы классов, соответствующих компонентам, начинаются с "T". Так что читатель должен учитывать, что, например, компонент **Label и TLabel** — это фактически одно и то же. И если надо найти описание класса **TLabel,** то следует смотреть раздел, посвященный компоненту **Label.**

В рамках данной книги, конечно, невозможно привести все справочные данные по всем компонентам C++Builder. Отобраны только основные, часто используемые компоненты. Значительно большее число компонентов вы найдете в [3], а подробная методика работы с компонентами рассмотрена в [1].

Организация взаимодействия компонентов в приложении

Хорошо структурированное приложение, обеспечивающее простоту его модернизации и сопровождения, должно строиться на основе диспетчеризации действий — основных операций, предусмотренных в нем. Одним из компонентов, осуществляющих диспетчеризацию действий, является компонент **ActionList.** Все действия, которые должны осуществляться в ответ на те или иные действия пользователя (выбор разделов меню, щелчки на кнопках и т.п.), могут быть занесены в список **ActionList** под некоторыми именами. Этим действиям могут быть приписаны различные характеристики: надписи, пиктограммы, тексты подсказок, быстрые клавиши, пиктограммы. Каждому действию соответствует некоторый обработчик события **OnExecute.** В C++Builder предусмотрено также множество стандартных действий, для которых даже не надо писать обработчики событий, поскольку соответствующие обработчики заложены в объектах этих действий.

В C++Builder 6 появилась группа гораздо более мощных компонентов, предназначенных для управления действиями: **ActionManager, ActionMainMenuBar, ActionToolBar, CustomizeDlg. Они** не только обеспечивают новые возможности визуального проектирования, но и решают, например, такую задачу, как настройка меню и инструментальных панелей пользователем. Правда, при этом возникают проблемы с русификацией диалогов.

Все управляющие элементы — кнопки, разделы меню ссылаются на соответствующее действие своим свойством **Action.** При этом характеристики действия автоматически переносятся в эти компоненты.

Пиктограммы, используемые в **ActionList,** в кнопках, меню, списках собираются в списке изображений — компоненте **ImageList.** На этот компонент ссылаются все другие компоненты, для которых требуются пиктограммы, своим свойством **Images.**

Еще одним полезным компонентом, который осуществляет перехват событий приложения (переменной **Application)** и тем самым упрощает многие операции, является **ApplicationEvents.**

Использование всех этих компонентов позволяет разрабатывать хорошо структурированные приложения, которые в дальнейшем легко сопровождать и модернизировать.

Каждое серьезное приложение должно иметь:

- главное меню **MainMenu**
- инструментальную панель быстрых кнопок (например, **ToolBar),** дублирующих основные разделы меню
- контекстные меню **PopupMenu,** всплывающие при щелчке пользователя правой кнопкой мыши на том или ином компоненте
- ярлычки подсказок (реализуются свойством **Hint),** всплывающие при перемещении курсора мыши над быстрыми кнопками и иными компонентами
- полосу состояния **StatusBar,** используемую часто для развернутых подсказок
- файл справки, темы которого отображаются при нажатии клавиши F1 и при выборе пользователем соответствующего раздела меню.

Методика разработки такого приложения в общих чертах сводится к следующему:

1. Продумывается и составляется список действий, которые должны быть доступны будущему пользователю через разделы меню, инструментальные панели, кнопки и другие элементы управления.
2. На форму переносится список изображений **ImageList** и заполняется пиктограммами, которые будут использоваться для тех нестандартных действий, которые должны быть доступны из быстрых кнопок инструментальной панели.
3. На главную форму приложения переносится компонент диспетчеризации действий: **ActionList или ActionManager.** Компонент связывается с **ImageList.** Двойным щелчком на диспетчере действий вызывается редактор действий, с помощью которого формируется список стандартных и нестандартных действий.
4. Каждому действию задается набор характеристик: Name (имя) **Caption** (надпись, в которой выделяется символ быстрого доступа), **Shortcut** (горячие клавиши), **ImageIndex** (номер изображения в **ImageList), Hint** (тексты подсказок), **HelpContext** или **HelpKeyword** (ссылка на тему справки) и др. Для нестандартных действий все эти характеристики вы записываете сами. Для стандартных действий они заносятся автоматически. Вам надо только перевести надписи и подсказки на русский язык и, может быть, исправить ссылки на не устраивающие вас стандартные изображения и комбинации горячих клавиш. А если у вас предусмотрена в приложении контекстная справка, то надо задать ссылки на соответствующие темы. Впрочем, в начале проектирования справки, конечно, еще нет. Так что свойства **HelpContext и HelpKeyword** вы можете задать позднее.
5. Записываются обработчики событий выполнения для всех нестандартных действий. Стандартные действия обрабатываются автоматически и для многих из них достаточно задать некоторые свойства обработки. Впрочем, как будет видно позднее, иногда надо писать обработчики и для стандартных действий.

Дальнейшие шаги зависят от того, используете ли вы компонент **ActionList, или ActionManager. Для ActionList** далее надо сделать следующее:

6. На форму переносится компонент **MainMenu** — главное меню, связывается с **ImageList,** в компоненте формируется меню, и в его разделах даются ссылки на действия, описанные в **ActionList.**

7. На форме создается инструментальная панель (обычно, компонент **ToolBar**). Панель связывается с ImageList, а в ее кнопках даются ссылки на действия, описанные в **ActionList**.

Если вы используете компонент **ActionManager,** то эти шаги выглядят иначе:

6. На форму переносится компонент **ActionMainMenuBar** - - полоса главного меню. Она связывается с диспетчером **ActionManager.** Затем из редактора **ActionManager** перетаскиваются мышью на полосу меню категории разделов, которые должны входить в меню как головные разделы, или отдельные действия.
7. В редакторе **ActionManager** создается новая инструментальная панель, или несколько панелей. На них перетаскиваются мышью необходимые действия.

Дальнейшие операции во всех случаях сводятся к следующему:

8. Если необходимо обеспечить в приложении контекстные всплывающие меню, то на форму переносится один или несколько (по числу различных контекстных меню) компонентов **PopupMenu.** Они связываются с компонентом **ImageList** своим свойством **Images.** Далее контекстные меню заполняются аналогично главному меню, причем обычно большинство разделов просто копируются из главного меню.
9. При переносе на форму компонентов, с которыми должны быть связаны контекстные меню, в их свойствах **PopupMenu** даются ссылки на соответствующие компоненты **PopupMenu.**
10. На форму переносится компонент **ApplicationEvents,** и с его помощью записываются обработчики событий приложения, в частности, отображение развернутых подсказок в полосе состояния и т.п.
11. Создается файл справок **.hlp** и связывается с приложением.

Конечно, это очень схематичное описание методики проектирования. Детальное рассмотрение этой методики проведено в [1].

Взаимодействие компонентов, работающих с базами данных

Каждое приложение, использующее базы данных, обычно имеет по крайней мере по одному компоненту следующих трех типов:

- Компонент — набор данных (data set), непосредственно связывающийся с базой данных. Это такие компоненты, как **Table, Query, StoredProc** и ряд других.
- Компонент — источник данных (data source), осуществляющий обмен информацией между компонентами первого типа и компонентами визуализации и управления данными. Таким компонентом является **DataSource.**
- Компоненты отображения и ввода данных.

В качестве наборов данных могут использоваться компоненты со страницы BDE (Data Access в версиях, младше C++Builder 6), **Table, Query, StoredProc.** Наиболее просто связать с базой данных компонент **Table.** Для этого прежде всего надо задать свойство **DatabaseName,** выбрав базу данных из выпадающего списка, находящегося рядом с этим свойством в окне Инспектора Объектов. Затем надо задать свойство **TableName,** выбрав требуемую таблицу базы данных из соответствующего выпадающего списка. После этого можно установить в **true** свойство **Active.** Это свойство обеспечивает соединение с базой данных.

Перечисленные наборы данных связываются с базами данных посредством BDE. В C++Builder 6 имеется несколько страницы библиотеки, содержащих альтернативные наборы данных, для которых не требуется BDE. Это страница ADO (наборы данных **ADOTable, ADODataSet, ADOQuery, ADOStoredProc),** страница InterBase (наборы данных **IBTable, IBQuery, IBStoredProc, IBDataSet),** страница dbExpress **(SQLTable, SQLQuery, SQLStoredProc).**

В качестве источника данных всегда используется размещенный на странице Data Access компонент **DataSource.** Он связывается с набором данных своим свойством **DataSet.**

Подавляющее большинство компонентов, используемых для связи с конкретным полем таблицы данных, для отображения и редактирования данных, имеют два свойства **DataSource и DataField.** Первое из них указывает на компонент – источник данных, а второе — на поле таблицы. Впрочем, имеется компонент **DBGrid,** который представляет собой таблицу, в которой могут отображаться поля строк, числовые и булевы поля. Для связи этого компонента с набором данных используется только одно свойство — **DataSource.**

Помимо указанных компонентов в приложении может размещаться компонент **Database.** Этот компонент в основном используется в приложениях, работающих на платформе клиент/сервер. Его задачи связаны с общением с удаленным сервером, реализацией транзакций, работой с паролями. Компонент **Database** целесообразно вводить в приложение только в сравнительно редких случаях. Если он не введен явно, C++Builder автоматически создает его для каждой используемой в приложении базы данных.

Еще один компонент, который тоже автоматически создается C++Builder – компонент Session. Это главный компонент любого приложения, работающего с базами данных. Но в явном виде эти компоненты имеет смысл вводить только в многозадачные приложения, в которых параллельно обрабатывается несколько потоков информации.

ActionList — диспетчер действий

Невизуальный компонент, обеспечивает диспетчеризацию действий разработчика и, соответственно, событий компонентов.

Страница библиотеки *Standard*

Класс *TActionList*

Иерархия *TObiect — TPersistent — TComponent — TCustomActionList*

Модуль*ActnList*

Описание

Компонент **ActionList,** не добавляя никаких принципиально новых возможностей, позволяет систематизировать и упорядочить разработку объектно-ориентированных приложений путем организации связи между действиями и их инициаторами, такими как щелчок на кнопке или элементе меню. Компонент представляет собой интерфейс разработчика, позволяющий ему упорядочить свою работу с действиями в процессе проектирования путем создания списков действий.

В начале проектирования приложения разработчик должен представить себе список тех действий, которые может осуществлять пользователь. Практическая реализация составленного списка действий начинается с переноса на проектируемую форму компонента **ActionList.** Сделав на этом компоненте двойной щелчок, вы попадаете в редактор действий (см. рис. 2.1), позволяющий вводить и упорядочивать действия.

Щелчок правой кнопкой мыши или щелчок на маленькой кнопочке со стрелкой вниз правее первой быстрой кнопки окна редактирования позволит вам выбрать одну из команд: New Action (новое действие) или New Standard Action... (новое стандартное действие). Первая из них относится к вводу нового действия любого типа. По умолчанию эти действия будут получать имена **Actionl, Action2** и т.д. Вторая команда открывает окно, в котором вы можете выбрать необходимое вам стандартное действие (или сразу несколько действий). После этого в правом окне (Actions) редактора появятся имена выбранных действий, а в левом (Categories) – категории действий.

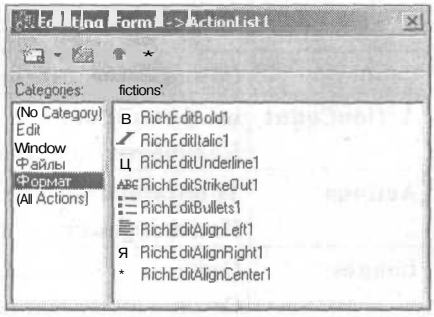


Рис. 2.1
Окно редактора действий

Каждое действие, которое вы внесли в список — это объект типа **TAction** для нестандартных действий или других производных типов для стандартных. Выбрав в окнах редактора ту или иную категорию или [AllActions] (все категории), а в правом — конкретное действие, вы можете увидеть в Инспекторе Объектов его свойства и события. В Инспекторе Объектов для каждого действия вы можете установить свойство **Name** — имя, а также ряд свойств, которые автоматически перенесутся затем во все компоненты, ссылающиеся на данное действие. Это **Caption** — надпись, **Shortcut** — «горячие» клавиши, **Hint** — подсказки в ярлычках и панели состояния, **HelpContext. HelpKeyword, HelpType** — связь с контекстной справкой, **Enabled. Visible. Checked.**

Можно для каждого или некоторых действий указать свойство **ImageIndex,** которое является индексом (начиная с 0) изображения, соответствующего данному действию в отдельном компоненте списка изображений **ImageList.** Этот индекс передастся в дальнейшем компонентам, связанным с данным событием — разделам меню, кнопкам. Если в свойстве **Images** компонента **ActionList** указать имя списка, размещенного на форме и заполненного изображениями, то эти изображения появятся также в окне редактора действий.

Свойство **Category** (категория) не имеет отношения к выполнению приложения. Задание категории просто позволяет в процессе проектирования сгруппировать действия по их назначению. Можно назвать их русскими именами (категории Файлы и Формат на рис. 2.1).

На странице событий Инспектора Объектов для каждого действия определено три события:

- **OnExecute** — возникает в момент, когда пользователь инициализировал действие, например, щелкнув на компоненте (разделе меню, кнопке), связанном с данным действием.
- **OnUpdate** — периодически возникает в промежутках между действиями. Возникновение этих событий прекращается только во время реализации события или во время, когда пользователь ничего не делает и компьютер находится в состоянии ожидания действий. Обработчик события может содержать какие-то настройки, подготовку ожидаемых дальнейших действий или выполнение каких-то фоновых операций.
- **OnHint** — возникает в момент, когда на экране отображается ярлычок подсказки в результате того, что пользователь задержал курсор мыши над компонентом, инициализирующим событие.

Связь объектов действий с конкретными инициализаторами действий — управляющими элементами типа кнопок, разделов меню и т.д., осуществляется через свойство **Action,** имеющееся у всех управляющих элементов. Достаточно в объекте раздела меню или в копке **SpeedButton** установить соответствующее значение **Action,** и все заданные атрибуты этого действия, включая обработчик события, перенесутся в раздел меню и кнопку.

Основные свойства

Свойство	Объявление / Описание
ActionCount	**int ActionCount** **Количество действий в списке. Только для чтения**
Actions	**TContainedAction* Actions[int Index]** **Индексированный список действий**
Images	**Controls::TImageList* Images** **Ссылка на список изображений — обычно на компонент ImageList**

Остальные свойства наследуются от **TComponent.**

Основные методы

Метод	Объявление / Описание
ExecuteAction	**bool ExecuteAction(Classes::TBasicAction* Action)** **Вызывается из метода Execute указанного действия Action. Генерирует событие OnExecute компонента ActionList**
IsShortCut	**bool IsShortCut(Messages::TWMKey &Message)** **Распознает «горячие» клавиши, связанные с действием. Явным образом из приложения не вызывается**
UpdateAction	**bool UpdateAction(Classes::TBasicAction* Action)** **Вызывается при обновлении списка действий**

События

Событие	Описание
OnChange	Наступает, когда изменяется список действий
OnExecute	Наступает, когда действие генерирует событие OnExecute
OnUpdate	Наступает, когда изменяется действие

Все эти события используются редко. Поэтому больший интерес представляют события действий. Для каждого действия, входящего в массив Actions, основными событиями являются:

Событие	Описание
OnExecute	Реализация действия
OnHint	Возникает в момент, когда на экране должен отобразиться ярлычок подсказки
OnUpdate	События происходят между выполнениями действий. Обработчики событий могут использоваться для каких-то настроек, для подготовки ожидаемых дальнейших действий или выполнения каких-то фоновых операций

ActionManager — диспетчер действий

Невизуальный компонент, обеспечивает диспетчеризацию действий разработчика и, соответственно, событий компонентов, с возможностью настройки пользователем инструментальных панелей и меню.

Страница библиотеки *Additional*

Класс *TActionManager*

Иерархия *TObject — TPersistent — TComponent — TCustomActionList — TCustomActionManager*

Модуль *ActnMan*

Описание

Компонент **ActionManager** введен в C++Builder 6 для создания дополнительных возможностей диспетчеризации действий. Методика работы с этим компонентом коротко изложена в данной главе, в разд. «Организация взаимодействия компонентов в приложени», а подробно рассмотрена в книге [1].

Диспетчер действий **ActionManager** создает список стандартных и нестандартных действий и в этом отношении подобен **ActionList.** Но возможности **ActionManager** несравненно шире. Он не только хранит набор действий. Он управляет также полосами действий — визуальными компонентами, на которых располагаются элементы пользовательского интерфейса. К таким компонентам относятся **ActionMainMenuBar** — полоса главного меню, и **ActionTooIBar** — инструментальная панель. Эти компоненты могут вводиться в приложение непосредственно из палитры компонентов, а могут создаваться Редактором Действий **ActionManager.** Из окна Редактора Действий можно формировать полосы действий простым перетаскиванием на них необходимых действий.

Компонент **ActionManager** запоминает информацию о составе набора действий и конфигурации полос действий в текстовом или двоичном файле на диске. При этом можно предоставить пользователю возможность настройки меню и инструментальных полос во время выполнения. Эта настройка сохранится в файле и в следующем сеансе работы автоматически загрузится в приложение.

Настройка во время выполнения может осуществляться или вызовом соответствующего стандартного действия, или обращением к специальному компоненту **CustomizeDlg,** перенесенному на форму. В обоих случаях во время выполнения открывается то же диалоговое окно Редактора Действий, которое используется во время проектирования, но в несколько упрощенном варианте. Это окно позволяет пользователю в процессе выполнения приложения настраивать меню и инструментальные панели, перенося на них новые действия, убирая прежние, делая те или иные инструментальные панели видимыми или невидимыми. Компонент **ActionManager** обеспечивает сохранение в файле на диске этих пользовательских настроек и загрузку их в следующем сеансе работы с приложением. К сожалению, диалог Редактора Действий, позволяющий пользователю проводить настройки, реализован, естественно, на английском языке.

Свойство **State** компонента **ActionManager** определяет реакцию на действия пользователя. Значение **asNormal** соответствует нормальной рабочей реакции: при щелчке пользователя на доступных интерфейсных компонентах действий выполняются соответствующие действия. Два других возможных значения — **asSuspended и asSuspendedEnabled** отключают возможность выполнения действий. Щелчок пользователя не приводит ни к каким результатам. Эти значения State используются при настройке пользователем меню и инструментальных панелей. Различие этих двух значений в том, что первое не изменяет свойства **Enabled** действий, а второе переводит во всех действиях **Enabled в true.**

Таким образом, если вы в некоторый момент хотите, чтобы полосы действий, управляемые компонентом **ActionManager**1, перешли в состояние настройки, надо выполнить оператор

```
ActionManager1->State = asSuspended;
```

Тогда нажатие пользователем кнопки инструментальной панели или раздела меню не будет вызывать выполнение соответствующего действия и может интерпретироваться программой как-то иначе.

Свойство **FileName** задает имя файла, в котором **ActionManager** хранит информацию о составе связанных с ним полос действий. В начале выполнения приложения **ActionManager** читает информацию из этого файла и в соответствии с ней формирует полосы действий. А при любых изменениях настройки в процессе выполнения компонент записывает в этот файл проведенные изменения. Так что при следующем сеансе работы состав полос действий будет таким, каким сделал его пользователь в предыдущем сеансе.

Если вы в процессе проектирования впервые задаете значение **FileName,** вам надо просто записать в этом свойстве имя файла с путем к нему. При отсутствии пути файл будет создан в том каталоге, в котором расположен ваш проект. В случае, если вы хотите задать в качестве значения **FileName** имя уже существующего файла, можете воспользоваться для его выбора кнопкой с многоточием около свойства **FileName** в окне Инспектора Объектов.

Свойство Images компонента ActionManager указывает на компонент ImageList, содержащий пиктограммы, используемые для обозначения действий.

Основной инструмент проектирования - - Редактор Действий компонента **ActionManager.** Он вызывается двойным щелчком на **ActionManager.** Вы попадете в окно Редактора Действий на страницу Actions. Щелкнув в ней правой кнопкой мыши, вы можете ввести новое нестандартное или стандартное действий, выбрав из контекстного меню соответственно команду New Action или New Standard Action.

После того как вы выбрали некоторые действия, в панели Actions появятся имена объектов этих действий, а в панели Category — их категории. Если вы выделите какое-то действие, в Инспекторе Объектов вы можете увидеть и изменить его свойства: **Caption, Hint, Shortcut** и другие. В частности, вы можете изменить его категорию (свойство Category). В отличие от компонента **ActionList. в ActionManager** понятие категории имеет вполне определенный смысл. При создании меню названия категорий станут надписями головных разделов меню. Так что имеет смысл оформить их сразу так, как положено в меню, переведя на русский язык и введя символ амперсанда.

Страница Toolbars (инструментальные панели) окна Редактора Действий компонента **ActionManager** содержит список управляемых диспетчером ActionManager инструментальных панелей (компонентов **ActionToolBar)** и меню (компонентов **ActionMainMenuBar).** Компонент **ActionMainMenuBar** (полосу главного меню) надо добавлять в приложение обычным способом, перенося его из палитры компонентов. А добавить на форму компонент **ActionToolBar** можно, просто нажав в окне Редактора Действий на странице Toolbars кнопку New.

После того как вы добавили кнопкой New новую инструментальную панель или перенесли на форму меню **ActionMainMenuBar,** можно вернуться на страницу Actions и перетащить с нее мышью на панель или в полосу меню требуемые действия или целиком категории. Если в меню перетаскивается категория, ее надпись становится головным разделом меню.

Индикаторы на странице Toolbars около названий полос действий управляют их видимостью. Если выключить такой индикатор, то во время выполнения соответствующая панель будет невидимой.

Если вы выделили на странице Toolbars одну из полос действий, вы можете увидеть в Инспекторе Объектов ее свойства. Большинство из них обычны для любых панелей. В частности, имеет смысл изменить задаваемую по умолчанию надпись (Caption) на что-то более понятное пользователю, так как при настройках во время выполнения пользователь будет видеть эти надписи. Стоит также обратить внимание на свойство — **AllowHiding.** Оно позволяет (при значении **true)** или запрещает делать полосу невидимой в процессе выполнения. Если задать **AllowHiding = false,** то в Редакторе Действий такая полоса отображается серой. Для такой полосы ни во время проектирования, ни во время выполнения невозможно переключить индикатор видимости. Полоса всегда будет видна.

Выпадающий список Caption Options управляет способом отображения надписей **(Caption)** действий в выделенной в окне Toolbars инструментальной панели. Значение **None** соответствует отсутствию надписей. Отображаются только пиктограммы. Впрочем, если действие не имеет пиктограммы, то надпись все равно отображается. Значение **Selective** позволяет для каждого действия задать видимость надписи индивидуально. Значение **AH** обеспечивает отображение надписей для всех действий. Индикатор Apply caption options to all toolbars распространяет выбранную опцию на все инструментальные панели.

Страница Options Редактора Действий позволяет задать некоторые опции отображения. Индикатор Menus show recently used items first делает меню перестраиваемым, как это предусмотрено в Windows 2000/XP, причем первыми располагаются разделы, которые недавно использовались. Если этот индикатор включен, то диспетчер действий при каждом очередном выполнении приложение проверяет, давно ли пользователь обращался к тому или иному разделу меню. Если на протяжении нескольких сеансов работы какие-то разделы не использовались, они делаются невидимыми. Точнее, их можно увидеть, только развернув меню полностью. Таким образом, меню перестраивается от сеанса к сеансу, делая видимыми и легко доступными те разделы, к которым пользователь обращается чаще всего.

Кнопка Reset Usage Data восстанавливает первоначальные установки полос действий. Индикатор Large icons приводит к отображению в полосах действий больших пиктограмм. Индикатор Show tips on toolbars управляет появлением всплывающих ярлычков у элементов инструментальных панелей. А индикатор Show shortcut keys in tips определяет включение в тексты этих ярлычков обозначений «горячих» клавиш. Выпадающий список Menu animation определяет форму отображения меню.

Основные свойства

Свойство	Объявление / Описание
ActionBars	TActionBars* ActionBars Коллекция объектов полос действий, управляемых компонентом
ActionCount	int ActionCount Количество действий в списке. Только для чтения
Actions	TContainedAction* Actions[int Index] Индексированный список действий
FileName	AnsiString FileName Имя файла, в котором хранятся текущие настройки полос действий
Images	Imglist::TCustomImageList* Images Ссылка на список изображений — обычно на компонент ImageList
Linked ActionLists	TActionListCollection* LinkedActionLists Коллекция списов действий, доступных из полос действий (инструментальных панелей и меню), управляемых данным компонентом
Priority Schedule	Classes::TStringList*PrioritySchedule Список, управляющий числом обращений к действиям в управляемом меню
State	enum TActionListState {asNormal, asSuspended, asSuspendedEnabled}; bool State Определяет реакцию на действия пользователя (см. выше в описании компонента)

Основные методы

Метод	Объявление / Описание
ExecuteAction	bool ExecuteAction(Classes::TBasicAction* Action) Вызывается из метода Execute указанного действия Action. Генерирует событие OnExecute диспетчера действий
LoadFromFile	void LoadFromFile(const AnsiString FileName) Загружает из файла, указанного свойством FileName, информацию о натройках полос действий
ResetActionBar	void ResetActionBar(Index: Integer) Восстанавливает состояние по умолчанию полосы действий с индексом Index
ResetUseageData	void ResetUsageData Удаляет файл, указанный свойством FileName, восстанавливая тем самым начальное состояние всех полос действий
SaveToFile	void SaveToFile(const AnsiString Filename) Сохраняет в файле, указанном свойством FileName, состояние всех полос действий

События

Событие	Описание
OnChange	Наступает, когда изменяется список действий
OnExecute	Наступает, когда действие генерирует событие OnExecute
OnUpdate	Наступает, когда изменяется действие

Все эти события используются редко. Поэтому больший интерес представляют события действий, которые вы можете посмотреть в разд. «ActionList».

ActionMainMenuBar — настраиваемая полоса состояния главного меню

Компонент главного меню, работающий совместно с диспетчером действий ActionManager.

Страница библиотеки *Additional*

Класс *TActionMainMenuBar*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TToolWindow — TCustomActionBar — TCustomActionDockBar — TCustomActionMenuBar — TCustomActionMainMenuBar*

Модуль *ActnMenus*

Описание

Компонент **ActionMainMenuBar** введен в C++Builder 6 для создания совместно с **ActionManager** перестраиваемой полосы главного меню. Методика работы с этим компонентом коротко изложена в данной главе, в разделах «Организация взаимодействия компонентов в приложении» и «ActionManager», а подробно рассмотрена в книге [1].

Компонент должен вводится в приложение обычным способом переносом его на форму из палитры компонентов. А формирование разделов меню, как рассмотрено в указанных разделах, призводится из редактора компонента **ActionManager.**

Основные свойства

Свойство	Объявление / Описание
ActionControls	**TCustomActionControl* ActionControls[const** int Index] Список объектов разделов меню
ActionManager	TActionManager* ActionManager Указатель на компонент ActionManager, управляющий данным меню
AnimationStyle	**enum TAnimationStyle** {asNone, asDefault, **asUnFold,** asSlide, **asFade};** TAnimationStyle AnimationStyle Определяет стиль анимации при показе разделов меню
ExpandDelay	int ExpandDelay Определяет задержку в миллисекундах (по умолчанию 4000) пред показом невидимых или неиспользуемых разделов
Inactive	**bool** Inactive Определяет доступность данного меню
InMenuLoop	**bool** InMenuLoop Указывает, может ли меню получать фокус, обеспечивая доступ к его разделам
ParentControl	Actnman::TCustomActionControl* ParentControl Указатель на родительский компонент. Используется в контекстных меню, состав котолрых может изменяться в зависимости объекта, к которому такое меню относится
RootMenu	TCustomActionMenuBar* RootMenu Указатель на головное меню из объекта вспомогательного выпадающего меню

Основные методы

Метод	**Объявление / Описание**
CloscMenu	**voidCloseMenu(void)** **Сворачивает меню**
FindFirst	**TActionClientItem* FindFirst(void)** **Возвращает объект первого раздела меню**

Основные события

Событие	**Описание**
OnPopup	**Наступает перед показом всплывающего меню**
OnPaint	**Наступает перед прорисовкой меню на экране**

ActionToolBar — настраиваемая инструментальная панель

Настраиваемая инструментальная панель, работающая совместно с диспетчером действий ActionManager.

Страница библиотеки *Additional*

Класс *TActionToolBar*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TToolWindow — TCustomActionBar — TCustomActionDockBar — TCustomActionToolBar*

Модуль *ActnCtrls*

Описание

Компонент **ActionToolBar** введен в C++Builder 6 для создания совместно с **ActionManager** перестраиваемых инструментальных панелей. Методика работы с этим компонентом коротко изложена в данной главе в разделах «Организация взаимодействия компонентов в приложении» и «ActionManager», а подробно рассмотрена в книге [1].

Компонент может вводиться в приложение обычным способом — переносом его на форму из палитры компонентов, а может содаваться из редактора компонента **ActionManager.** Формирование кнопок инструментальной панели, как рассмотрено в указанных разделах данной главы, призводится из редактора компонента **ActionManager.**

Основные свойства

Свойство	Объявление / Описание
ActionClient	TActionCIient* ActionClient Список клинтов, размещенных на панели
ActionControIs	TCustomActionControl* ActionControls[const int Index] Список управляющих компонентов на панели
ActionManager	TActionManager* ActionManager Указатель на компонент ActionManager, управляющий данной панелью
Align	enum TAlign { alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom } TAlign Align Определяет выравнивание панели в родительском контейнере
AllowHiding	bool AllowHiding Разрешает делать панель невидимой
HiddenCount	int HiddenCount Число невидимых управляющих компонентов на панели
Orientation	enum TBarOrientation {boLeftToRight, boRightToLeft, boTopToBottom, boBottomToTop}; TBarOrientation Orientation Определяет ориентацию размещения управляющих компонентов: слева направо, справа налево, сверху вниз, снизу вверх
Vert S eparator	bool HorzSeparator Определяет появление разделителя между рядами компонентов

Основные методы

Метод	Объявление / Описание
FindFirst	TActionClientItem* FindFirst(void) Возвращает объект первого управляющего элемента панели
FindFirst VisibleItem	TActionClientItem* FindFirstVisibleItem(void) Возвращает объект первого видимого управляющего элемента панели
FindLast VisibleItem	TActionClientItem* FindLastVisibleItem(void) Возвращает объект последнего видимого управляющего элемента панели

Основные события

Событие	Описание
OnControl Created	Наступает в момент появления на панели нового компонента
OnPaint	Наступает перед прорисовкой панели на экране

Кроме того, наследуется множество событий класса **TWinControl**, связанных с перетаскиванием и встраиванием, а также с манипуляциями мышью

Animate — клипы Windows

Используется для воспроизведения немых клипов AVI, подобных используемым в Windows изображениям копирования файлов и т.п.

Страница библиотеки *Win32*

Класс *TAnimate*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl*

Модуль *comctrls*

Описание

Компонент **Animate** позволяет воспроизводить на форме стандартные видео клипы Windows (типа копирования файлов, поиска файлов и т.п.) и немые видео файлы .avi — Audio Video Interleaved. Эти файлы представляют собой последовательность кадров битовых матриц. Они могут содержать и звуковую дорожку, но компонент **Animate** воспроизводит только немые клипы AVI. Он работает только с неуплотненными файлами AVI или с клипами AVI, уплотненными с использованием RLE — run-length encoding.

Animate может воспроизводить клипы AVI из ресурсов, из файлов или из библиотеки Shell32.dll, если приложение работает с Windows 95/98/2000 или NT.

Во время проектирования в компонент **Animate** можно загрузить кадры клипа AVI, указав такие свойства, как **FileName** — имя файла и **CommonAVI** — стандартный клип Windows. Можно также просматривать клип по кадрам, щелкнув на компоненте правой кнопкой мыши и выбирая разделы всплывающего меню Next Frame (Следующий кадр) или Previous Frame (Предыдущий кадр). Это позволит вам выбрать фрагмент клипа и воспроизводить его методом **Play,** если вы не хотите воспроизводить клип полностью.

В компоненте **Animate** предусмотрены события **OnClose, OnOpen, OnStart и OnStop,** генерируемые соответственно в моменты закрытия и открытия компонента, начала и окончания воспроизведения.

Основные свойства

Свойство	Объявление / Описание
Active	**bool Active** Указывает, должен ли компонент воспроизводить клип. Установка в **true** запускает воспроизведение кадров, начиная со **StartFrame** и кончая **StopFrame**, столько раз, сколько указано в свойстве **Repetitions.** Установка в **false** прерывает воспроизведение. Перед активацией компонент должен быть открыт
AutoSize	**bool AutoSize** Указывает, что размер компонента автоматически устанавливается равным размерам кадров
Center	**bool Center** Определяет центровку изображения в клиентской области компонента, если размеры клиентской области **ClientHeight** или **ClientWidth** не совпадают с соответствующими размерами кадра **FrameHeight и FrameWidth.** Если центровка не задана, изображение позиционируется в верхнем левом углу клиентской области
CommonAVI	**enum TCommonAVI {aviNone, aviFindFolder, aviFindFile, aviFindComputer, aviCopyFiles, aviCopyFile, aviRecycleFile, aviEmptyRecycle, aviDeleteFile};** **TCommonAVI CommonAVI** Указывает клип Windows из библиотеки Shell32.dll
FileName	**AnsiString FileName** Указывает имя воспроизводимого файла. Установка **FileName** автоматически приводит к установке **CommonAVI = aviNone,** очищает **ResName** и сбрасывает на 0 **ResHandle и ResID**
FrameCount	**int FrameCount** Определяет общее количество кадров в клипе AVI. Свойство только для чтения
FrameHeight	**int FrameHeight** Высота, необходимая для полного отображения кадров. Если свойство **AutoSize** установлено в **true,** высота клиентской области компонента автоматически устанавливается равной **FrameHeight.** Свойство только для чтения
FrameWidth	**int FrameWidth** Ширина, необходимая для полного отображения кадров. Если свойство **AutoSize** установлено в **true,** ширина клиентской области компонента автоматически устанавливается равной **FrameWidth.** Свойство только для чтения
Open	**bool Open** Указывает, открыт ли компонент, т.е. загружен ли клип. При задании клипа свойствами **CommonAVI, FileName, ResName** или **ResID,** компонент открывается автоматически. Установка в **false** освобождает ресурсы, используемые клипом. При установке в **true** свойство **StartFrame** устанавливается в 1, а свойство **StopFrame — во FrameCount**

компонентов **ApplicationEvents.** Если же вы при этом не хотите, чтобы другие компоненты **ApplicationEvents** получали события, примените к привилегированному компоненту метод **CancelDispatch.** Тогда после обработки события в данном компоненте другие компоненты **ApplicationEvents** вообще не будут реагировать на эти события.

Во многие обработчики событий компонента **ApplicationEvents** передается по ссылке параметр **Handled.** По умолчанию его значение равно **false.** Если вы обработали соответствующее событие и не хотите, чтобы оно далее обрабатывалось другими компонентами **ApplicationEvents,** надо в обработчике установить **Handled = true.** Если же вы оставите **Handled = false,** то событие будут пытаться обрабатывать другие компоненты **ApplicationEvents** (если они есть). Если событие так и останется необработанным, то его будет пытаться обработать активный компонент, а если не обработает — то активная форма. Предотвратить обработку события другими компонентами можно, используя описанный ранее метод **CancelDispatch.**

Приведем примеры использования описаннных далее событий компонента **ApplicationEvents.**

Обработчик события **OnHint:**

```
void TForm1::ApplicationEvents1Hint(TObject *Sender)
{
  StatusBar1->SimpleText = Application->Hint;
}
```

отображает в полосе состояния **StatusBarl** вторую часть свойства **Hint** любого компонента, в котором определено это свойство и над которым перемещается курсор мыши. Отображение происходит независимо от значения свойства **ShowHint** компонента.

Обработчик события **OnShowHint:**

```
void TForm1::ApplicationEvents1ShowHint(AnsiString &HintStr,
                                        bool &CanShow, THintInfo &HintInfo)
{
 if (HintInfo.HintControl->ClassNameIs("TEdit"))
  if (Canvas->TextWidth(Edit1->Text) > Edit1->ClientWidth)
  {
   HintStr = Edit1->Text;
   ApplicationEvents1->CancelDispatch();
  }
}
```

проверяет класс источника события и, если это окно редактирования, то с помощью функции **TextWidth** проверяет, не превышает ли длина текста ширины клиентской области. Если превышает, то текст ярлычка подменяется текстом, содержащимся в окне редактирования. Такой прием позволяет пользователю, подведя курсор мыши к окну редактирования, увидеть во всплывающем окне полный текст, который может не быть виден обычным образом, если он длинный и не помещается целиком в окне редактирования. Только учтите, что событие **OnShowHint** будет наступать при перемещении курсора только над теми компонентами, в которых свойство **ShowHint** установлено в **true.**

Обработчик события **OnHelp:**

```
bool TForm1::ApplicationEvents1Help(
                    WORD Command, int Data,bool &CallHelp)
{
 if((Command == HELP_CONTEXT) && (Data < 10))
 {
   Application->HelpCommand(HELP_CONTEXTPOPUP,Data);
   CallHelp = false;
 }
 return true;
}
```

обеспечивает отображение всех контекстных справок с номерами иденификаторов тем, меньшими 10, во всплывающем окне, не вызывая при этом WinHelp. Это дает возможность отобразить многострочные (в отличие от ярлычков) всплывающие окна, поясняющие назначение тех или иных элементов приложения.

Обработчик события On**ShortCut:**

```
void TForm1::ApplicationEvents1ShortCut(TWMKey &Msg, bool &Handled)
{
if (Msg.CharCode  == 'Q')
 if  (Application->MessageBox(
      "Действительно хотите завершить работу?",
      "Подтвердите завершение",
      MB_YESNOCANCEL+MB_ICONQUESTION) == IDYES)
  Application->Terminate();
)
```

перехватывает нажатие пользователем клавиш и, если нажата клавиша с символом "Q" (в любом регистре и независимо от установки русского или английского языка), то пользователю методом **Application->MessageBox** предлагается диалоговое окно с текстом "Действительно хотите завершить работу?". Если пользователь в нем нажмет кнопку Да, то приложение закрывается методом Application- >Terminate.

Обработчики событий OnActivate и OnDeactivate :

```
void TForm1::ApplicationEvents1Activate(TObject *Sender)
(
 Label1->Caption = "Ура! Я работаю!";
}

void TForm1::ApplicationEvents1Deactivate(TObject *Sender)
{
 Label1->Caption = "Увы! Меня покинули!";
}
```

обеспечивают появление в метке Label1 соответствующей надписи каждый раз, когда пользователь переключается из данного приложение в другое и возвращается назад.

Основные свойства

Никаких специальных свойств в компоненте не определено. Компонент наследует ряд свойств базового класса **TComponent**.

Основные методы

Метод	Объявление / Описание
Activate	void Activate(void) Ставит данный компонент в начало очереди компонентов ApplicationEvents. В результате все события передаются прежде всего этому компоненту
CancelDispatch	void CancelDispatch(void) Предотвращает обработку текущего события другими объектами **ApplicationEvents**

Остальные методы наследуются от базового класса **TControl**.

События

Событие	Описание
OnAction Execute	Возникает при выполнении действия, объявленного в компоненте **ActionList,** но не обработанного им (не написан соответствующий обработчик). В обработчик передается параметр **Action** — действие и по ссылке передается параметр **Handled**
OnAction Update	Возникает при обновлении (Update) некоторого действия, объявленного в компоненте **ActionList,** но не обработанного им (не написан соответствующий обработчик). В обработчик передается параметр **Action** — действие и по ссылке передается параметр **Handled**
OnActivate	Возникает, когда приложение становится активным (при начале выполнения и в случаях, когда пользователь, перейдя к другим приложениям, вернулся в данное). Если это событие обработано в **ApplicationEvents,** то предотвратить дальнейшую его обработку можно методом **CancelDispatch**
OnDeactivate	Возникает перед тем моментом, когда приложение перестает быть активным (пользователь переключается на другое приложение). Если событие обработано в **ApplicationEvents,** то предотвратить дальнейшую его обработку можно методом **CancelDispatch**
OnExeption	Возникает, когда в приложении сгенерировано исключение, которое нигде не перехвачено. В обработчик передается параметр **Sender** — источник исключения, и параметр Е типа **Exception** — объект исключения. В обработчике можно предусмотреть нестандартную обработку исключений на уровне приложения, например, русифицировать стандартные сообщения об исключениях и дать пользователю какие-то рекомендации
OnHelp	Возникает при запросе приложением справки. Это событие возникает, в частности, при выполнении методов приложения **HelpContext, Help Jump и HelpCommand.** Обработчик может использоваться для каких-то подготовительных операций. В обработчик передаются параметры — **Command** — команда API WinHelp, **Data** — параметр этой команды и по ссылке передается булев параметр **CallHelp.** Если его установить в **true,** то после завершения обработчика будет вызван WinHelp, в противном случае вызова WinHelp не будет
OnHint	Возникает в момент, когда курсор мыши начинает перемещаться над компонентом или разделом меню, в котором определено свойство **Hint.** При этом свойство **Hint** компонента переносится в свойство **Hint** приложения **(Application.Hint)** и в обработчике данного события может отображаться, например, в строке состояния
OnIdle	Возникает, когда приложение начинает простаивать, ожидая, например, действий пользователя. В обработчик передается по ссылке булев параметр **Done,** который по умолчанию равен **true.** Если оставить его без изменения, то по окончании работы обработчика будет вызвана функция **WaitMessage** API Windows, которая займется в период ожидания другими приложениями. Если задать **Done = false,** то эта функция вызываться не будет. Это может снизить производительность Windows

Событие	Описание
OnMessage	Возникает, когда приложение получает сообщение Windows. В обработчике можно предусмотреть нестандартную обработку сообщения. В обработчик передается параметр Msg — полученное сообщение и по ссылке передается параметр Handled. Учтите, что в Windows передаются тысячи сообщений в секунду, так что обработчик OnMessage может серьезно снизить производительность приложения
OnMinimize	Возникает при сворачивании приложения
OnRestore	Возникает при восстановлении ранее свернутого приложения
OnShortCut	Возникает при нажатии пользователем клавиши, до того, как возникло стандартное событие **OnKeyDown**. Обработчик позволяет предусмотреть нестандартную реакцию на нажатие какой-то клавиши. В него передается параметр сообщения Windows Msg, поле **CharCode** которого **(Msg.CharCode)** содержит виртуальный код нажатой клавиши. Передается также по ссылке параметр Handled. Если задать ему значение true, то стандартные события **OnKeyDown**, **OnKeyPress**, OnKeyUD не наступят
OnShowHint	Возникает, когда приложение собирается отобразить ярлычок с текстом подсказки Hint. В обработчик передается по ссылке параметр HintStr — первая часть свойства Hint компонента, ярлычок которого должен отображаться. В обработчике этот текст можно изменить. Так же по ссылке передается параметр CanShow. Если в обработчике установить его равным false, то ярлычок отображаться не будет. Третий параметр, передаваемый по ссылке — **HintInfo**. Это структура, поля которой содержат информацию о ярлычке. В частности, имеется поле HintControl — компонент, сообщение которого должно отображаться в ярлычке, и поле HintStr — отображаемое сообщение. В обработчике это значение можно изменить

BatchMove — перенос данных из одного набора в другой

Невизуальный компонент, предназначен для групповых операций переноса данных из одного набора в другой.

Страница библиотеки *BDE*, в версиях младше C++Builder 6 — *Data Access*

Класс *TBatchMove*

Иерархия *TObject* — *TPersistent* — *TComponent*

Модуль *Dbtables*

Описание

Компонент **BatchMove** предназначен для групповых операций переноса данных из одного набора в другой. Основные свойства компонента: **Source** — источник данных и **Destination** — приемник данных типа **Ttable.** Свойство **Mode** (см. в гл. 3) определяет режим переноса данных.

Основной метод компонента — **Execute** выполняет операцию переноса данных.

При использовании компонента **BatchMove** надо иметь в виду, что обычно возникают проблемы с переносом русских текстов.

Основные свойства

Свойство	Объявление / Описание
AbortOnKey Viol	**bool** AbortOnKeyViol Указывает, должна ли немедленно прекращаться операция, вызвавшая в таблице-приемнике нарушение целостности или дублирование первичного ключа. При задании false желательно одновременно задать KeyViolTableName
AbortOn Problem	**bool AbortOnProblem** Указывает, должна ли немедленно прекращаться операция, вызвавшая несоответствие типов полей в таблице-источнике и таблице-приемнике. При задании false желательно одновременно задать **ProblemTableName**
ChangedCount	**int** ChangedCount Число записей, измененных или добавленных в таблице-приемнике (при Mode = batUpdate или batAppendUpdate), или удаленных из нее (при Mode = batDelete). Копии этих записей хранятся в таблице ChangedTableName
ChangedTable Name	**AnsiString** ChangedTableName Определяет имя таблицы Paradox, создаваемой для сохранения копий всех изменяемых записей таблицы-приемника
CommitCount	**int** CommitCount Объем перемещаемых данных в байтах, при превышении которого данные фиксируются в базе данных
Destination	TTable* Destination Объект Table, являющийся приемником данных
KeyViolCount	int KeyViolCount Число записей, нарушающих целостность таблицы-приемника и поэтому не помещенных в приемник
KeyViolTable Name	AnsiString **KeyViolTableName** Указывает имя таблицы Paradox, в которую будут помещаться записи, нарушающие целостность таблицы-приемника и поэтому не помещенные в приемник
Mappings	**Classes::TStrings*** Mappings Список, позволяющий задать таблицу соответствия полей источника и приемника
Mode	enum TBatchMode { batAppend, batUpdate, batAppendUpdate, batDelete, batCopy }; TBatchMode Mode Режим переноса данных
MovedCount	int MovedCount Число записей таблицы-источника, участвующих в операции переноса данных
ProblemCount	int ProblemCount Число записей с несоответствием типов полей, которые поэтому не помещены в приемник

Свойство	Объявление / Описание
ProblemTable Name	AnsiString ProblemTableName Указывает имя таблицы Paradox, в которую будут помещаться записи с несоответствием типов полей, которые поэтому не помещены в приемник
RecordCount	int RecordCount Число записей, успешно перенесенных в таблицу-приемник
Source	TBDEDataSet* Source Объект Table, являющийся источником данных
Transliterate	bool Transliterate Указывает, требуется ли трансляция множества символов при переносе данных из источника в приемник

Основной метод

Метод	Объявление / Описание
Execute	void Execute(void) Выполняет операцию, заданную свойством Mode

Остальные методы наследуются от TComponent.

BDEClientDataSet — клиентский набор данных BDE

Клиентский набор данных, использующий BDE.

Страница библиотеки *BDE*

Класс *TBDEClientDataSet*

Иерархия *TObiect — TPersistent— TComponent — TDataSet — TCustomClientDataSet, TCustomCachedDataSet*

Модуль *DBClient*

Описание

Компонент **BDEClientDataSet** используется для создания в приложении клиентского набора данных. С его помощью можно создавать автономные наборы данных, портфельные наборы данных, интерфейсы к другим наборам данных, обладающие расширенными возможностями по индексации и фильтрации данных, по расчету совокупных характеристик.

Все вопросы, связанные с клиентскими наборами данных, см. в описании компонента ClientDataSet. Отличие **BDEClientDataSet от ClientDataSet** заключается в том, что **BDEClientDataSet** использует внутренние компоненты **Query и DataSetProvider.** Так что внешний провайдер для **BDEClientDataSet** не требуется, как он требуется для **ClientDataSet.** Вместо этого для связи с базой данных используется компонент **Database.** На этот компонент дается ссылка в свойстве **DBConnection.** А запрос SQL записывается в свойстве CommandText.

Таким образом, в простейшем случае, когда базовый и клиентский наборы данных располагаются в одном приложении, для связи этих наборов надо сделать следующее:

- Ввести в приложение компонент **Database** и установить его свойства, обеспечивающие связь с базой данных.

- Ввести в приложение компонент BDEClientDataSet и в его свойстве **DBConnection** сослаться на компонент Database.
- Записать в свойстве **CommandText** текст запроса. Например, «Select * from Pers».

А все остальное программирование и работа с клиентским набором осуществляется так же, как описано для компонента **ClientDataSet**.

BitBtn — кнопка с пиктограммой

Управляющая кнопка с пиктограммой.

Страница библиотеки *Additional*

Класс *TBitBtn*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TButtonControl — TButton*

Модуль *stdctrls*

Примеры изображения

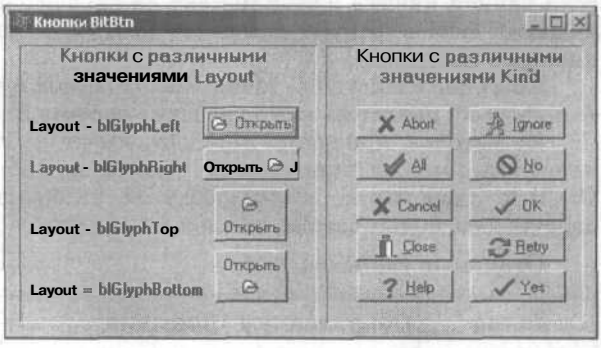


Рис. 2.2
Примеры кнопок BitBtn

Описание

Компонент BitBtn — это управляющая кнопка, на поверхности которой можно располагать изображение. Изображение на кнопке задается ее свойством Glyph. Оно представляет собой битовую матрицу, содержащую до четырех изображений размером 16 на 16. Самое левое соответствует отжатой кнопке. Второе слева соответствует недоступной кнопке, когда ее свойство Enabled равно false. Третье слева изображение используется при нажатии пользователя на кнопку при ее включении. Четвертое слева изображение используется в кнопках с фиксацией SpeedButton, а не в BitBtn. Изображение во время проектирования загружается в Glyph с помощью редактора, вызываемого из Инспектора Объектов. Число пиктограмм в изображении отображается автоматически в свойстве **NumGlyphs**.

Расположение изображения и надписи (свойство Caption) на кнопке определяется свойствами Margin, Layout и Spacing. Если свойство Margin равно -1 (значение по умолчанию), то изображение и надпись размещаются в центре кнопки. При этом положение изображения по отношению к надписи определяется свойством Layout, которое может принимать значения (см. рис. 2.2): **blGlyphLeft** (слева, это значение принято по умолчанию), **blGlyphRight** (справа), blGlyphTop (вверху), **blGlyphBottom** (внизу). Если же Margin > 0, то в зависимости от значения Layout изображение и надпись смещаются к той или иной кромке кнопки, отступая от нее на число пикселов, заданное значением Margin.

Свойство Spacing задает число пикселов, разделяющих изображение и надпись на поверхности кнопки. По умолчанию Spacing = 4. Если задать Spacing = O,

изображение и надпись будут размещены вплотную друг к другу. Если задать Spacing = –1, то текст появится посередине между изображением и краем кнопки.

Еще одно свойство **BitBtn** — свойство **Kind** определяет тип кнопки. По умолчанию значение этого свойства равно **bkCustom** — заказная. Но можно установить и множество других предопределенных типов (см. рис. 2.2): **bkOK, bkCancel, bkHelp, bkYes, bkNo, bkClose, bkAbort, bkRetry, bkIgnore, bkAll.** В этих типах уже сделаны соответствующие надписи, введены пиктограммы, заданы свойства **ModalResult и др.**

Основное событие кнопки — **OnClick,** возникающее при щелчке на ней. В обработчике этого события записываются операторы, которые должны выполняться при щелчке пользователя на кнопке.

Свойство **Cancel,** если его установить в **true,** определяет, что нажатие пользователем клавиши Esc будет эквивалентно нажатию на данную кнопку. Это свойство целесообразно задавать равным **true** для кнопок Отменить в различных диалоговых окнах, чтобы можно было выйти из диалога, нажав на эту кнопку или нажав клавишу Esc.

Свойство **Default,** если его установить в **true,** определяет, что нажатие пользователем клавиши ввода Enter будет эквивалентно нажатию на данную кнопку, даже если данная кнопка в этот момент не находится в фокусе. Правда, если в момент нажатия Enter в фокусе находится другая кнопка, то все-таки сработает именно кнопка в фокусе.

Еще одно свойство — **ModalResult** используется в модальных формах. В обычных формах значение этого свойства должно быть равно **mrNone.**

Из методов, присущих кнопкам, имеет смысл отметить один — **Click.** Выполнение этого метода эквивалентно щелчку на кнопке, т.е. вызывает событие кнопки **OnClick.** Этим можно воспользоваться, чтобы продублировать какими-то другими действиями пользователя щелчок на кнопке.

Основные свойства

Свойство	Объявление / Описание
Action	Classes: **:TBasicAction*** Action Определяет действие, связанное с данной кнопкой
Cancel	**bool** Cancel Определяет, будет ли обрабатываться событие кнопки OnClick при нажатии клавиши Esc
Caption	**AnsiString** Caption Надпись на кнопке
Default	bool Default Определяет, что нажатие пользователем клавиши ввода Enter будет эквивалентно нажатию на данную кнопку, даже если данная кнопка в этот момент не находится в фокусе
Glyph	**Graphics::TBitmap*** Glyph Определяет битовую матрицу, которая появляется на кнопке
Kind	enum TBitBtnKind **{bkCustom,** bkOK, bkCancel, bkHelp, bkYes, bkNo, bkClose, bkAbort, bkRetry, bklgnore, **bkAll};** TBitBtnKind Kind Определяет тип кнопки с заранее заданными надписями, пиктограммами и др. (см. выше в описании BitBtn). По умолчанию тип кнопки — заказная (bkCustom)

Свойство	Объявление / Описание
Layout	enum TButtonLayout {**blGlyphLeft, blGlyphRight,** blGlyphTop, blGlyphBottom }; TButtonLayout Layout Определяет, к какому краю кнопки смещается изображение
Margin	int Margin Определяет количество пикселов между краем изображения и краем кнопки, указанным свойством Layout. При -1 изображение и надпись размещаются в центре кнопки
ModalResult	**typedef** int **TModalResult;** Forms::TModalResult ModalResult Определяет значение свойства модальной формы ModalResult, задаваемое автоматически при щелчке на данной кнопке
NumGliphs	typedef Shortint **TNumGlyphs;** **TNumGlyphs** NumGlyphs Указывает количество изображений в свойстве Glyph
Spacing	int Spacing Число пикселов, разделяющих изображение и надпись на поверхности кнопки
Style	enum **TButtonStyle** { bsAutoDetect, **bsWin31, bsNew** }; TButtonStyle Style Определяет внешний вид кнопки. По умолчанию bsAutoDetect – автоматически изменяет вид, подстраиваясь под Windows 3.x или 32-разрядные Windows
TabOrder	typedef short **TTabOrder;** TTabOrder TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	**bool** TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Метод	Объявление / Описание
Click	void Click(void) Имитирует щелчок мышью, как если бы пользователь щелкнул на кнопке
ExecuteAction	bool ExecuteAction(TBasicAction* Action) Вызывает указанное действие Action, связанное с данной кнопкой
SetFocus	void SetFocus(void) Передает фокус элементу, активизирует его

Событие

Событие	Описание
OnClick	Соответствует щелчку мыши на кнопке или нажатию клавиш быстрого доступа

Button — кнопка

Кнопка для выполнения пользователем каких-то команд.

Страница библиотеки *Standard*

Класс *TButton*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TButtonControl*

Модуль *stdctrls*

Пример изображения

Рис. 2.3
Пример кнопки Button

Описание

Компонент **Button** представляет собой стандартную кнопку Windows, инициирующую какое-то действие. Основное с точки зрения внешнего вида свойство кнопки — **Caption** (надпись). В надписях кнопок можно предусматривать использование клавиш ускоренного доступа, выделяя для этого один из символов надписи — ставя перед ним символ амперсанда "&". Этот символ не появляется в надписи, а следующий за ним символ оказывается подчеркнутым. Тогда пользователь может вместо щелчка на кнопке нажать в любой момент клавишу Alt совместно с клавишей выделенного символа.

Основное событие кнопки — **OnClick,** возникающее при щелчке на ней. В обработчике этого события записываются операторы, которые должны выполняться при щелчке пользователя на кнопке.

Свойство **Cancel,** если его установить в **true,** определяет, что нажатие пользователем клавиши Esc будет эквивалентно нажатию на данную кнопку. Это свойство целесообразно задавать равным **true** для кнопок Отменить в различных диалоговых окнах, чтобы можно было выйти из диалога, нажав на эту кнопку или нажав клавишу Esc.

Свойство **Default,** если его установить в **true,** определяет, что нажатие пользователем клавиши ввода Enter будет эквивалентно нажатию на данную кнопку, даже если данная кнопка в этот момент не находится в фокусе. Правда, если в момент нажатия Enter в фокусе находится другая кнопка, то все-таки сработает именно кнопка в фокусе.

Еще одно свойство — **ModalResult** используется в модальных формах. В обычных формах значение этого свойства должно быть равно **mrNone.** Но в модальных формах использование этого свойства позволяет в ряде случаев вообще не писать обработчик щелчка на кнопке.

Из методов, присущих кнопкам, имеет смысл отметить один — **Click.** Выполнение этого метода эквивалентно щелчку на кнопке, т.е. вызывает событие кнопки

OnClick. Этим можно воспользоваться, чтобы продублировать какими-то другими действиями пользователя щелчок на кнопке.

Основные свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данной кнопкой
Cancel	**bool** Cancel Определяет, будет ли обрабатываться событие кнопки OnClick при нажатии клавиши Esc
Caption	AnsiString Caption property Caption: TCaption; Надпись на кнопке
Default	bool Default Определяет, что нажатие пользователем клавиши ввода Enter будет эквивалентно нажатию на данную кнопку, даже если данная кнопка в этот момент не находится в фокусе
ModalResult	**typedef int TModalResult;** **Forms::TModalResult** ModalResult Определяет значение свойства модальной формы ModalResult, задаваемое автоматически при щелчке на данной кнопке
TabOrder	typedef short **TTabOrder;** **TTabOrder** TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	bool TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Метод	Объявление / Описание
Click	void Click(void) Имитирует щелчок мышью, как если бы пользователь щелкнул на кнопке
Execute Action	bool ExecuteAction(TBasicAction* Action) Вызывает указанное действие Action, связанное с данной кнопкой
SetFocus	void SetFocus(void) Передает фокус элементу, активизирует его

Событие

Событие	Описание
OnClick	Соответствует щелчку мыши на кнопке или нажатию клавиш быстрого доступа

Chart — графики и диаграммы

Компонент принадлежит к семейству компонентов **TChart,** которые используются для создания диаграмм и графиков.

Страница библиотеки *Additional*

Класс *TChart*

Модуль *Chart*

Примеры изображения

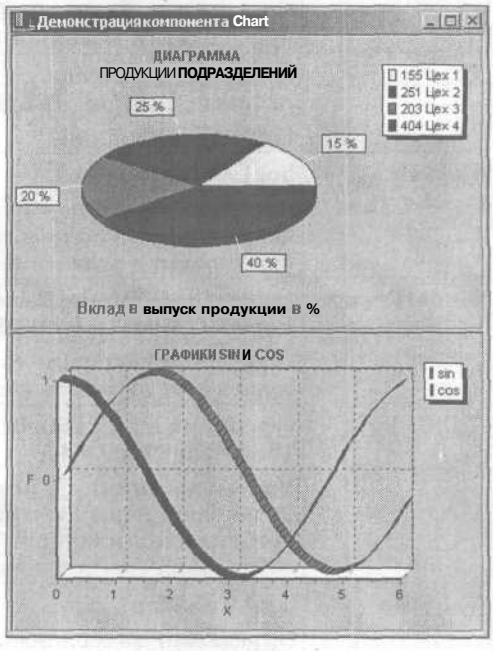


Рис. 2.4
Примеры компонента Chart: вверху отображение диаграммы, внизу - графиков

Описание

Компонент **Chart** является панелью, на которой можно создавать диаграммы и графики различных типов.

Компонент является контейнером объектов Series типа **TChartSeries** — серий данных, характеризующихся различными стилями отображения. Каждый компонент может включать несколько серий. Свойства серий устанавливаются с помощью Редактора Диаграмм или программно. Редактор Диаграмм вызывается из Инспектора Объектов нажатием кнопки с многоточием около соответствующего свойства или двойным щелчком на компоненте **Chart.** Вы попадаете в окно Редактора Диаграмм, показанное на рис. 2.5, на страницу Chart, которая имеет несколько закладок. Прежде всего, вас будет интересовать на ней закладка Series. Щелкните на кнопке Add — добавить серию. Вы попадете в окно, в котором вы можете выбрать тип диаграммы или графика. Воспользовавшись закладкой Titles, вы можете задать заголовок диаграммы, закладка Legend позволяет задать параметры отображения легенды диаграммы (списка обозначений) или вообще убрать ее с экрана, закладка Panel определяет вид панели, на которой отображается диаграмма, закладка 3D дает вам возможность изменить внешний вид вашей диаграммы: наклон, сдвиг, толщину и т.д.

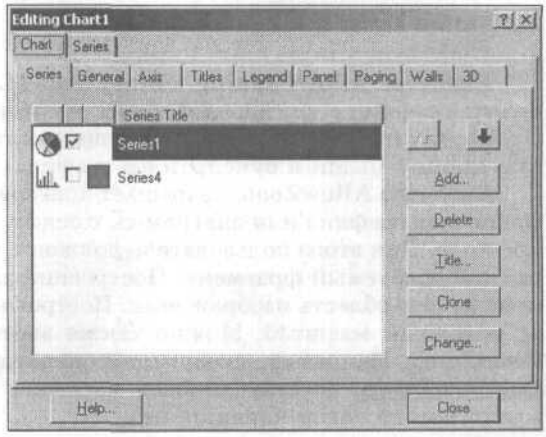


Рис. 2.5
Окно Редактора Диаграмм компонента Chart

Когда вы работаете с Редактором Диаграмм и выбрали тип диаграммы, в компоненте Chart на вашей форме отображается ее вид с занесенными в нее условными данными. Поэтому вы сразу можете наблюдать результат применения различных опций к вашему приложению, что очень удобно.

Страница Series, также имеющая ряд закладок, дает вам возможность выбрать дополнительные характеристики отображения серии. В частности, для круговой диаграммы на закладке Format полезно включить опцию Circled Pie, которая обеспечит при любом размере компонента Chart отображение диаграммы в виде круга. На закладке Marks кнопки группы Style определяют, что будет написано на ярлычках, относящихся к отдельным сегментам диаграммы: Value — значение, Percent — проценты, Label — названия данных и т.д.

Для программного задания отображаемых в диаграммах и графиках значений используются методы серий Series. Основные из них:

- Clear — очищает серию от занесенных ранее данных
- Add — позволяет добавить в диаграмму новую точку
- AddXY — позволяет добавить новую точку в график функции

Например, следующие операторы очищают серию Seriesl и заносят в нее для отображения диаграммы четыре значения, задавая отображающие их цвета:

```
int A1=155;
int A2=251;
int A3=203;
int A4=404;
Series1->Clear ();
Series1->Add(A1,"Цех 1",clYellow);
Series1->Add(A2,"Цех 2",clBlue);
Series1->Add(A3,"Цех 3",clRed);
Series1->Add(A4,"Цех 4",clPurple);
```

Следующие операторы заносят в серию Series2 значения, предназначенные для отображения графика синуса:

```
Series2->Clear ();
for (int i = 0; i <= 100; i++)
  Series2->AddXY(0.02*Pi*i,sin(0.02*Pi*i),"",clRed);
```

Следующий оператор переносит данные серии Seriesl в серию **Series3**, свойства которой, например, определяют отличный от Seriesl тип диаграммы:

```
Series3->Assign(Series1);
```

Следующие операторы обеспечивают смену типа диаграммы, переключая видимость серий Seriesl и **Series3**:

```
Series1->Active = ! Series1->Active;
Series3->Active = ! Series3->Active;
```

Свойство **AllowPanning** компонента **TChart** разрешает пользователю прокручивать графики и те типы диаграмм, в которых предусмотрены координатные оси. Прокрутку пользователь может осуществлять во время выполнения, нажимая правую кнопку мыши и буксируя ею график.

Свойство **AllowZoom** разрешает пользователю увеличивать размер выбранного фрагмента графика или диаграммы с осями координат, растягивая его на все видимое поле. Для этого пользователь должен с помощью левой кнопки мыши обвести рамкой требуемый фрагмент. Построение рамки вниз и вправо растягивает фрагмент на всю область изображения. Построение рамки вверх и влево восстанавливает исходный масштаб. Можно также восстановить исходный масштаб методом **UndoZoom**. Например, следующий оператор, вставленный в обработчик события **OnMouseDown**, восстанавливает масштаб, если пользователь нажимает кнопку мыши при нажатой клавише Alt:

```
if (Shift.Contains(ssAlt)) Chart1->UndoZoom();
```

Можно также изменять масштаб методами **ZoomPercent и ZoomRect.**

По умолчанию весь график или диаграмма размещаются на одной странице, которая видна целиком. Но если задать **MaxPointPerPage** — максимальное число точек на страницу, то изображение будет автоматически разбито на несколько страниц (если, конечно, число точек серий больше, чем **MaxPointPerPage).** Последняя страница может оказаться неполной. На экране одновременно можно видеть одну страницу (она определяется свойством **Page).** Перемещение по страницам возможно с помощью прокрутки графика пользователем (если оно разрешено свойством **AllowPanning)** или с помощью свойств **Page, NumPages** (число страниц) и методов **PreviousPage и NextPage.** Отображение последней страницы определяется свойством **ScaleLastPage.**

Среди множества свойств серий можно отметить **Mark** — ярлычки, отображающие численные значения точек серии.

Множество свойств **Chart** определяет оформление графика — оси координат (они могут быть со всех 4-ех сторон), трехмерную имитацию отображения и т.п.

Работа с компонентом **Chart** подробно рассмотрена в [3].

Компонент реализован на Object Pascal, так что не удивляйтесь, что приведенные ниже определения свойств и методов не похожи на C++.

Основные свойства

Свойство	Объявление / Описание
AllowPanning	**property AllowPanning : TPanningMode;** Разрешение пользователю прокручивать графики и некоторые типы диаграмм (диаграммы с осями координат), нажимая правую кнопку мыши: **pmNone** — запрет прокрутки, **pmHorizontal** — горизонтальная прокрутка, **pmVertical** — вертикальная прокрутка, **pmBoth** — прокрутка в любых направлениях
AllowZoom	**property AllowZoom : Boolean;** Разрешение пользователю увеличивать размер фрагмента графика и некоторых типов диаграмм (диаграммы с осями координат), обводя рамкой с помощью мыши требуемую область увеличения. Построение рамки вниз и вправо растягивает фрагмент на всю область изображения. Построение рамки вверх и влево восстанавливает исходный масштаб

Свойство	Объявление / Описание
Animated Zoom	property **AnimatedZoom** : Boolean; Определяет, будет ли при **AllowZoom** = true увеличение размера выполняться плавно (при значении **true**), или скачком
Animated ZoomSteps	property **AnimatedZoomSteps** : Integer; Задает число шагов, используемых при плавном изменении масштаба (при AnimatedZoom = true)
AxisVisible	property Axis**Visible** : Boolean; При значении true видны те координатные оси, в которых свойство Visible = true. При значении AxisVisible = false все оси не видны
BackWall	**property** BackWall : **TChartWall;** Определяет множество атрибутов задней стенки трехмерного изображения осей координат (пространства между осями): видимость, цвет, размер, штриховку и т.п.
BottomAxis	property BottomAxis : **TChartAxis;** Множество атрибутов, определяющих нижнюю координатную ось
BottomWall	property BottomWall : TChartWall; Определяет множество атрибутов трехмерного изображения нижней оси координат: видимость, цвет, размер, штриховку и т.п.
ClipPoints	property ClipPoints : Boolean; Определяет рисование последовательности точек в пределах координатных осей, предохраняя другие области от затирания. При значении false сдвиг графика приводит к тому, что его изображение выходит за пределы координатных осей
DepthAxis	property DepthAxis : TChartAxis; Множество атрибутов, определяющих координатную ось, направленную в глубину
Foot	property Foot : TChartTitle; Оформление нижней части диаграммы: текст (Text) и атрибуты его форматирования
LeftAxis	property LeftAxis : TChartAxis; Множество атрибутов, определяющих левую координатную ось
LeftWall	property LeftWall : TChartWall; Определяет множество атрибутов трехмерного изображения левой оси координат: видимость, цвет, размер, штриховку и т.п.
Legend	property Legend : **TChartLegend;** Определяет текст и атрибуты изображения легенды — списка обозначений данных
MarginBottom	property MarginBottom : Integer; Размер нижнего поля, свободного от изображения
MarginLeft	property MarginLeft : Integer; Размер левого поля, свободного от изображения

Свойство	Объявление / Описание
MarginRight	property MarginRight : Integer; Размер правого поля, свободного от изображения
MarginTop	property MarginTop : Integer; Размер верхнего поля, свободного от изображения
MaxPoints **PerPage**	property MaxPointsPerPage : **LongInt;** По умолчанию 0. Задание значения > 0 определяет число точек на страницу. Если в серии больше точек, изображение автоматически размещается на нескольких страницах. Перемещение по страницам возможно с помощью прокрутки графика пользователем (при **AllowPanning** = pmHorizontal или pmBoth) или с помощью свойств Page, NumPages и методов PreviousPage и NextPage
NumPages	function NumPages : Longint; Число страниц графика. Отлично от 1 только при MaxPoints-PerPage > 0. Свойство времени выполнения
Page	property Page : Longlnt; Текущая видимая страница. Отлична от 1 при MaxPointsPer-Page > 0. Свойство времени выполнения
RightAxis	property RightAxis : **TChartAxis;** Множество атрибутов, определяющих правую координатную ось
ScaleLast Page	property ScaleLastPage : Boolean; Управляет способом отображения последней страницы при MaxPointsPerPage > 0. Значение true означает отображение в том же масштабе, как и другие страницы. При значении false масштаб выбирается исходя из числа точек (их на последней странице может быть меньше, чем на предыдущих)
Series	property **Series[Index:Longint]:TChartSeries;** Индексированный список серий — объектов с множеством своих свойств и методов. Одно из основных из них — Active определяет активность серии
SeriesCount	function SeriesCount : Longint ; Число серий в массивах Series и SeriesList
Title	property Title : TChartTitle; Определяет текст надписи и атрибуты форматирования в верхней части компонента
TopAxis	property TopAxis : TChartAxis; Множество атрибутов, определяющих верхнюю координатную ось
View3D	property **View3d** : Boolean; Задает трехмерный характер изображения
View3DWalls	property **View3dWalls** : Boolean; Определяет видимость левой и правой стенок объемного изображения координатных осей

Свойство	Объявление / Описание
Width3D	property Width3D : Longint; Определяет глубину при трехмерном изображении

Основные методы

Метод	Объявление / Описание
ActiveSeries Legend	function ActiveSeriesLegend(SeriesIndex : Longint) : TChartSeries; Возвращает активную серию с указанным индексом
AddSeries	procedure AddSeries(ASeries : TChartSeries); Добавляет новую серию
Assign	procedure Assign(Source : TPersistent); Копирует все свойства указанной серии в данную
ChartXCenter	function ChartXCenter : Longint; Средняя горизонтальная координата изображения
ChartYCenter	function ChartYCenter : Longint; Средняя вертикальная координата изображения
CopyTo ClipBoard Metafile	procedure CopyToClipboardMetafile(Enhanced:Boolean); Копирует всю область диаграммы в буфер Clipboard в формате метафайла
CopyTo Clipboard Bitmap	procedure CopyToClipboardBitmap; Копирует всю область диаграммы в буфер в формате битовой карты
GetASeries	function GetASeries : TChartSeries; Возвращает первую активную серию
GetAxis Series	function GetAxisSeries(Axis : TChartAxis) : TChartSeries; Возвращает первую серию, связанную с указанной осью
LoadChart FromFile	procedure LoadChartFromFile(Var AChart:TCustomChart; Const AName:String); Загружает изображение из указанного файла. Это может быть файл, ранее сохраненный методом SaveChartToFile
MaxXValue	function MaxXValue(AAxis : TChartAxis): Double; Возвращает максимальное значение, отображаемое на горизонтальной оси
MaxYValue	function MinYValue(AAxis: TChartAxis): Double; Возвращает максимальное значение, отображаемое на вертикальной оси
MinXValue	function MinXValue(AAxis: TChartAxis): Double; Возвращает минимальное значение, отображаемое на горизонтальной оси
MinYValue	function MinYValue(AAxis: TChartAxis): Double; , Возвращает минимальное значение, отображаемое на вертикальной оси

Метод	Объявление / Описание
NextPage	procedure NextPage; Переходит к следующей странице (идентично выражению Page = Page+1)
NumPages	function NumPages : Longint; Возвращает количество страниц
PreviousPage	procedure PreviousPage; Переходит к следующей странице (идентично выражению Page = Page −1)
Print	procedure Print; Посылает изображение на печать
Print Landscape	procedure PrintLandscape; Задает альбомную (Landscape) ориентацию принтера.
Print Orientation	procedure PrintPortrait; Посылает изображение на печать с заданной ориентацией. После печати прежняя ориентация восстанавливается
PrintPortrait	procedure PrintPortrait; Устанавливает книжную (вертикальную) ориентацию принтера
Remove AllSeries	procedure RemoveAllSeries; Удаляет все серии из SeriesList
RemoveSeries	procedure RemoveSeries(ASeries : TChartSeries); Удаляет указанную серию из SeriesList
RotatcLabel	procedure RotateLabel(x,y: Integer; Const St: String; RotDegree: Integer); Рисует указанную строку текста, повернутую под указанным углом
SaveChart ToFile	procedure SaveChartToFile(AChart : TCustomChart; Const AName : String); Сохраняет изображение в файле с указанным именем. В дальнейшем он может быть загружен методом LoadChartFromFile
SaveTo BitmapFile	procedure SaveToBitmapFile(Const FileName : String); Сохраняет изображение в файле .bmp с указанным именем
SaveTo Metafile	procedure SaveToMetafile(Const FileName : String); Сохраняет изображение в метафайле с указанным именем
SaveTo MetafileEnh	procedure SaveToMetafileEnh(Const FileName : String); Сохраняет изображение в метафайле Enhanced WMF с указанным именем
SeriesCount	function SeriesCount : Longint; Количество серий в компоненте (активных и неактивных)
SeriesDown	procedure SeriesDown(ASeries : TChartSeries); Пересылает серию на «задний план» — она будет рисоваться последней

Метод	Объявление / Описание
SeriesTitle Legend	function SeriesTitleLegend(SeriesIndex : Longint; ActiveOnly : Boolean) : String; Возвращает заголовок указанной серии
SeriesUp	procedure SeriesUp (ASeries : TChartSeries); Пересылает серию на «передний план» — она будет рисоваться первой
UndoZoom	procedure UndoZoom; Восстанавливает исходный масштаб после изменения его пользователем
ZoomPercent	procedure ZoomPercent(Const PercentZoom : Double); Осуществляет изменение (увеличение или уменьшение) масштаба заданием его в процентах. На характер процесса изменения оказывает влияние свойство AnimatedZoom
ZoomRect	procedure ZoomRect(Const Rect : TRect); Осуществляет изменение (увеличение или уменьшение) масштаба заданием координат новой отображаемой области. На характер процесса изменения оказывает влияние свойство AnimatedZoom

События

Событие	Описание
OnAfterDraw	Наступает после прорисовки всех серий. В обработчике нельзя изменять какие-то свойства, способные вызвать повторную прорисовку серий
OnAllowScroll	Наступает перед прокруткой изображения. В обработчике можно указать величину прокрутки, а можно запретить прокрутку
OnClick	Наступает при щелчке на точке какой-то серии
OnClickAxis	Наступает при щелчке на оси координат
OnClick Background	Наступает при щелчке в точке фона, не относящейся к оси, серии или легенде
OnClickLegend	Наступает при щелчке на легенде — списке обозначений
OnClickSeries	Наступает при щелчке на точке какой-то серии. В обработчике можно узнать серию и точку, на которой был щелчок
OnGetAxisLabel	Наступает при рисовании метки оси
OnGetLegendPos	Наступает перед отображением легенды. В обработчике можно определить и изменить координаты и цвет отображения
OnGetLegend Rect	Наступает перед отображением легенды. В обработчике можно определить и изменить координаты и размеры отображения
OnGetLegend Text	Наступает перед отображением легенды. В обработчике можно определить и изменить отображаемый текст
OnGetNextAxis Label	Обработчик используется для неавтоматического изображения осей

Событие	Описание
OnPageChange	Наступает перед изменением свойства Page перед прорисовкой изображения новой страницы
OnScroll	Наступает перед очередной перерисовкой при прокрутке пользователем изображения правой кнопкой мыши
OnUndoZoom	Наступает при восстановлении масштаба с помощью метода UndoZoom и соответствующем изменении диапазона отображаемых значений
OnZoom	Наступает при увеличении изображения пользователем или методами ZoomRect и **ZoomPercent**

CheckBox — индикатор

Индикатор с флажком, используемый для включения и выключения каких-то опций или для индикации состояния.

Страница библиотеки *Standard*

Класс *TCheckBox*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TButtonControl — TCustomCheckBox*

Модуль *stdctrls*

Примеры изображения

Рис. 2.6
Примеры индикаторов

Описание

Индикаторы с флажком **CheckBox** используются в приложениях в основном для того, чтобы пользователь мог включать и выключать какие-то опции, или для индикации состояния. При каждом щелчке пользователя на индикаторе его состояние изменяется, проходя в общем случае последовательно через три значения: выделение (появление черной галочки — **CheckBoxl** на рис. 2.6), промежуточное (серое окно индикатора и серая галочка — **CheckBox2** на рис. 2.6) и не выделенное (пустое окно индикатора — **CheckBox3** на рис. 2.6). Этим трем состояниям соответствуют три значения свойства компонента **State: cbChecked, cbGrayed, cbUnchecked.** Все три состояния допускаются только при значении другого свойства **AllowGrayed** равном **true.** Если же **AllowGrayed = false** (значение по умолчанию), то допускается только два состояния: выделенное и не выделенное. И **State,** и **AllowGrayed** можно устанавливать во время проектирования или программно во время выполнения.

Промежуточное состояние обычно используется, если индикатор применяется для отображения какой-то характеристики объекта. Например, если индикатор призван показать, какой регистр использовался при написании некоторого фрагмента текста, то в случае, если весь текст написан в верхнем регистре, индикатор может принимать выделенное состояние, если в нижнем — не выделенное, а если использовались оба регистра — промежуточное.

Проверять состояние индикатора можно не только по значению **State,** но и по значению свойства **Checked.** Если **Checked** равно **true,** то индикатор выбран, т.е. **State = cbChecked.** Если **Checked** равно **false, то State** равно **cbUnchecked** или **cbGrayed.** Установка **Checked в true** во время проектирования или выполнения автоматически переключает **State в cbChecked.**

Надпись в индикаторе задается свойством **Caption,** а ее размещение по отношению к индикатору — свойством **Alignment** (на рис. 2.6 в нижнем индикаторе **Alignment = taLeftJustify** и надпись расположена слева).

Основное событие индикатора — **OnClick,** наступающее при щелчке на компоненте. В обработчике этого события можно анализировать свойства **Checked** и **State,** которые уже приняли новое значение.

Основные свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данным индикатором
Alignment	**enum** TAlignment { **taLeftJustify,** taRightJustify, taCenter **};** **typedef** TAlignment TLeftRight; Classes::TLeftRight Alignment Определяет положение надписи (слева или справа) по отношению к индикатору
AllowGrayed	**bool** AllowGrayed Разрешает или запрещает появление в индикаторе третьего состояния cbGrayed
Caption	AnsiString Caption Надпись индикатора
Checked	bool Checked Указывает, выбран ли индикатор (содержит ли он флажок)
State	**enum** TCheckBoxState **{cbUnchecked,** cbChecked, **cbGrayed};** TCheckBoxState State Определяет состояние индикатора: выключен (cbUnchecked), включен (cbChecked), в третьем состоянии (cbGrayed)
TabOrder	typedef short **TTabOrder;** TTabOrder TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	bool TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Метод	**Объявление / Описание**
ExecuteAction	**bool ExecuteAction(TBasicAction* Action)** **Вызывает указанное действие Action, связанное с данным индикатором**

Метод	Объявление / Описание
Hide	void Hide(void) Делает индикатор невидимым
SetFocus	void SetFocus(void) Передает фокус элементу, активизирует его
Show	void SetFocus(void) Делает видимым невидимый индикатор

Основные события

Событие	Описание
OnClick	Наступает при щелчке на компоненте. В обработчике этого события можно анализировать свойства Checked и State, которые уже приняли новое значение
OnContextPopup	Наступает при вызове пользователем контекстного меню, связанного с компонентом (щелчком правой кнопкой мыши или иным способом)

CheckListBox — список строк с индикаторами

Отображает список строк с индикаторами.

Страница библиотеки *Additional*

Класс *TCheckListBox*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TCustomListBox*

Модуль *checklst*

Примеры изображения

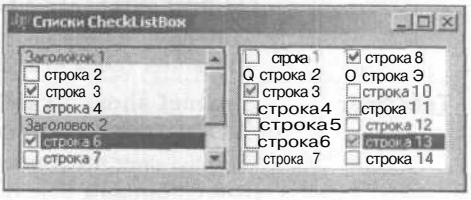


Рис. 2.7
Компоненты CheckListBox: слева - окно списка с одним столбцом, справа — с двумя

Описание

Компонент **CheckListBox** аналогичен компоненту списка строк **ListBox.** за исключением того, что рядом с каждый элементом находится окно с флажком — индикатор, который пользователь может включать и выключать, помечая элементы списока. Состояние индикатора изменяется при каждом щелчке пользователя на нем. Если свойство **AllowGrayed** установленно **true,** то при переключении индикатора возможно, наряду с включенным и выключенным состояниями, третье промежуточное состояние **cbGrayed** (см. строку 3 в правом списке на рис. 2.7).

Основное свойство компонента, содержащее список строк, — **Items,** имеющее тип **TStrings.** Заполнить его во время проектирования можно, нажав кнопку с многоточием около этого свойства в окне Инспектора Объектов. Во время выполнения работать с этим свойством можно, пользуясь свойствами и методами класса **TStrings — Clear, Add** и другими.

Индекс выбранной пользователем строки определяется свойством **ItemIndex,** доступным только во время выполнения. Если ни одна строка не выбрана, то **Itemlndex** = -1. Начальное значение **Itemlndex** невозможно задать во время проектирования. По умолчанию **Itemlndex** = -1. Это означает, что ни один элемент списка не выбран. Если вы хотите задать этому свойству какое-то другое значение, т.е. установить выбор по умолчанию, который будет показан в момент начала работы приложения, то сделать это можно, например, в обработчике события **OnCreate** формы, введя в него оператор вида

```
ChecListBox1.ItemIndex:=0;
```

Свойство **Columns** определяет число столбцов, в которых будет отображаться список, если он не помещается целиком в окне компонента CheckListBox (в правом списке на приведенном выше рисунке **Columns = 2).**

Свойство **Sorted** позволяет упорядочить список по алфавиту. При **Sorted** = **true** новые строки в список добавляются не в конец, а по алфавиту.

Свойство Style, установленное в **lbStandard** (значение по умолчанию) соответствует списку строк. Другие значения **Style** позволяют отображать в списке не только текст, но и изображения (см. об этом подробнее в описании компонента **ListBox).**

Состояния индикаторов определяют два свойства: **State и Checked.** Оба эти свойства можно рассматривать как индексированные массивы, каждый элемент которого соответствует индексу строки. Эти свойства можно устанавливать программно или читать, определяя установки пользователя. Например, операторы

```
CheckListBox1->Checked[1]= true;
CheckListBox1->State[2] = cbGrayed;
```

устанавливают индикатор второй строки списка CheckListBoxl в состояние выбранного, а индикатор третьей строки — в промежуточное состояние (вспомним, что индексы начинаются с 0).

Оператор

```
for (int i=0; i < CheckListBox1->Items->Count; i++)
  if( CheckListBox1->Selected[i] ) ...
```

проверяет состояние всех индикаторов списка и для выбранных пользователем строк осуществляет какие-то действия (в приведенном операторе на месте этих действий просто поставлено многоточие).

В компоненте **CheckListBox** имеется событие **OnClickCheck,** возникающее при каждом изменении пользователем состояния индикатора. Его можно использовать для обработки результатов изменения.

В C++Builder 6 в **CheckListBox** появились новые свойства, позволяющие зрительно разбить список на несколько разделов с помощью заголовков. Свойство **Header** представляет собой индексированный массив булевых значений, определяющих, является ли соответствующая строка заголовком (значение **true),** или это обычная строка с индикатором (значение **false).** Свойство **Header** — только времени выполнения и должно заполняться программно (по умолчанию все значения равны **false).** Например, операторы

```
CheckListBox1->Header[0] = true;
CheckListBox1->Header[4] = true;
```

задают в качестве заголовков первую и пятую строки (см. рис. 2.7).

Заголовки отображаются в строках с цветом фона, определяемым свойством **HeaderBackgroundColor,** и цветом надписи, задаваемым свойством **HeaderColor.**

См. также описание компонента **ListBox,** в котором описаны многие общие характеристики компонентов **CheckListBox и ListBox.**

Основные свойства

Свойство	Объявление / Описание
Action	Classes: **:TBasicAction*** Action Определяет действие, связанное с данным компонентом
Align	enum TAlign {alNone, **alTop,** alBottom, alLeft, alRight, alClient, **alCustom};** TAlign Align Определяет способ выравнивания компонента в контейнере (родительском компоненте)
AllowGrayed	**bool** AllowGrayed Разрешает или запрещает появление в индикаторе третьего промежуточного состояния **cbGrayed**
Anchors	enum **TAnchorKind** { **akLeft,** akTop, akRight, akBottom }; typedef Set<TAnchorKind, akLeft, akBottom> TAnchors; TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
Checked	bool Checked[int Index] Индексированный массив, каждый элемент которого соответствует индексу строки и определяет, включен ли в ней индикатор (имеет ли он флажок)
Columns	**int** Columns Определяет число столбцов, в которых будет отображаться список, если он не помещается целиком в окне компонента
Count	int Count Определяет число строк в списке. Может (и должно) задаваться только при значениях Style, равных **lbVirtual** или **lbVirtualOwnerDraw**
Extended Select	bool ExtendedSelect Определяет, может ли пользователь при MultiSelect = true выбрать несколько последовательно расположенных элементов, держа нажатой клавишу Shift
Header	bool Header[int Index] Индексированный массив булевых значений, определяющих, является ли соответствующая строка заголовком (значение true), или это обычная строка с индикатором (значение false). Свойство только времени выполнения
Header Background Color	**Graphics::TColor** HeaderBackgroundColor Определяет цвет фона строк заголовков, заданных свойством Header
HeaderColor	Graphics::TColor HeaderColor Определяет цвет надписей строк заголовков, заданных свойством Header

Свойство	Объявление / Описание
ItemEnabled	**bool ItemEnabled[int** Index] Индексированный массив, определяющий доступность или недоступность каждого элемента списка. Индикатор недоступного элемента закрашивается серым цветом и не может быть переключен пользователем
ItemIndex	int ItemIndex Указывает порядковый номер элемента, выделенного в списке. Свойство только времени выполнения
Items	**Classes::TStrings*** Items Массив строк списка — объект класса **TStrings.** Свойства этого класса позволяют формировать и изменять список
MultiSelect	bool MultiSelect Разрешает выбирать в списке несколько строк
SelCount	int SelCount Указывает количество выделенных элементов при MultiSelect = true. Доступ только для чтения
Selected	bool Selected[int Index] Индексированный массив, определяющий, какие элементы списка выделены
Sorted	bool Sorted Указывает, должны ли строки в списке автоматически сортироваться в алфавитном порядке
State	enum TCheckBoxState **{cbUnchecked,** cbChecked, cbGrayed}; Stdctrls::TCheckBoxState State[int Index] Индексированный массив, определяющий состояния всех индикаторов: **cbUnchecked** (не включен), cbChecked (включен), cbGrayed (в третьем состоянии)
Style	enum TListBoxStyle **{lbStandard, lbOwnerDrawFixed, lbOwnerDrawVariable, lbVirtual, lbVirtualOwnerDraw};** TListBoxStyle Style Определяет, будет ли окно списка стандартным, отображающим только текст, или будет позволять отображение также графических образов, а также определяет виртуальные списки (см. в описании компонента ListBox)
TopIndex	int TopIndex Указывает индекс элемента, видимого вверху списка. Изменение этого индекса соответственно сдвигает видимую часть списка

Основные методы

Метод	**Объявление / Описание**
Clear	**void Clear(void)** **Удаляет все элементы списка**

Метод	Объявление / Описание
ItemAtPos	int ItemAtPos(TPoint &Pos, **bool** Existing) Возвращает индекс элемента списка, соответствующего указанным координатам Pos. Если позиция Pos расположена после последнего элемента, то при Existing = true ItemAtPos возвращает −1, а при Existing = false — последний элемент списка
ItemRect	**Types::TRect ItemRect(int** Index) Возвращает прямоугольник, описывающий указанный элемент Item списка
SetFocus	Types::TRect **ItemRect(int** Index) Передает фокус элементу, активизирует его

Основные события

Событие	Описание
OnClick	Наступает при щелчке на элементе списка
OnClickCheck	Наступает, когда пользователь изменяет состояние индикатора элемента списка
OnData	Наступает в виртуальных списках, когда приложению надо отобразить очередную строку списка
OnDataFind	Обработчик события пишется для возможности управления виртуальным списком, например, для поиска строки по первым символам или для упорядочивания строк
OnDataObject	Наступает в виртуальных списках, когда со строками виртуального списка надо связать какие-то объекты
OnDrawItem	Наступает при необходимости перерисовать элемент списка (см. выше описание CheckListBox)
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKevUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу
OnMeasure Item	Наступает при необходимости перерисовать элемент в списке с изменяемой высотой элементов

ClientDataSet — клиентский набор данных

Клиентский набор данных.

Страница библиотеки *Data Access*

Класс *TClientDataSet*

Иерархия *TObject — TPersistent — TComponent — TDataSet — TCustomClientDataSet*

Модуль *DBClient*

Описание

Компонент **ClientDataSet** используется для создания в приложении клиентского набора данных. С его помощью можно создавать автономные наборы данных, портфельные наборы данных, интерфейсы к другим наборам данных, обладающие расширенными возможностями по индексации и фильтрации данных, по расчету совокупных характеристик.

Все свойства и методы компонент **ClientDataSet** наследует от своего базового класса **TCustomClientDataSet.** рассмотренного в гл. 1. Основные свойства, методы и события рассмотрены также в главах 3, 4 и 5, где можно найти соответствующие примеры. Клиентские наборы обладают также всеми свойствами и методами, наследуемыми ими от предшественника **TCustomClientDataSet** — класса **TDataSet.** и многими возможностями, присущими наборам данных, связанным с таблицами. Иначе говоря, упорядочивание данных с помощью индексов, задание ограничений, создание и модификация таблиц может осуществляться в клиентских наборах данных так же, как и в других наследниках **TDataSet.** Но во всех этих операциях добавляется немало новых возможностей.

Особенность клиентских наборов данных заключается в том, что данные хранятся в памяти и могут сохраняться в файле на диске и читаться из этого файла. Таким образом, клиентские наборы данных могут выступать как автономные основанные на файлах MyBase наборы данных в однопоточных приложениях. Другая немаловажная функция клиентских наборов — создание так называемой «портфельной» базы данных, в которую первоначально записываются данные сервера или какой-то иной базы данных, а затем вся работа проходит с файлом на компьютере клиента. Пользователь может изменять эти данные, причем при закрытии клиентского набора все изменения запоминаются в файле. А при открытии набора данные из файла записываются в клиентский набор. Таким образом, «портфельная» база данных — альтернатива кэширования. Преимуществом является то, что изменения могут производиться не обязательно в одном сеансе работы, а в нескольких. И когда пользователь решит, что изменения заслуживают пересылки в базу данных, он может занести в нее все изменения, сделанные на протяжении этих сеансов.

Клиентские наборы данных обладают еще одной полезной особенностью. С их помощью наиболее просто осуществлять взаимную трансляцию таблиц баз данных, созданных в разных СУБД и работающих на разных платформах.

Рассмотрим несколько подробнее основные области применения клиентских наборов данных. Начнем с автономных наборов данных.

Клиентские наборы данных могут выступать как автономные основанные на файлах MyBase наборы данных в однопоточных приложениях. Для работы с уже имеющимся автономным набором, хранящемся в виде файла, достаточно разместить на форме компонент клиентского набора **ClientDataSet** и указать в его свойстве **FileName** имя файла. При начале выполнения приложения данные из указанного файла будут читаться, а при завершении выполнения будут сохраняться в файле. Управление фиксацией результатов редактирования данных и отказом от проведенного редактирования осуществляется свойством **SavePoint (см. в гл.** 3).

Можно создавать новый пустой набор данных во время выполнения методом **CreateDataSet.** При этом, как показано в гл. 4 в примере, посвященном методу **CreateDataSet,** можно обеспечить пользователю возможность в диалоге создавать набор данных, задавать имена полей, индексы, файл, в котором будет храниться набор.

Новый пустой набор данных можно создавать и во время проектирования. Для этого сначала с помощью Инспектора Объектов надо заполнить свойство **FieldDefs.** Тогда при щелчке правой кнопкой мыши на компоненте клиентского набора данных в меню появится раздел Create Data Set. Выбор этого раздела обеспечит создание набора данных.

Теперь рассмотрим работу с портфельной базой данных. Для создания портфельного набора в приложении, кроме компонент клиентского набора **ClientDataSet,** должен располагаться провайдер (например, **DataSetProvider**) или брокер, обеспечивающие связь с основной базой данных. Набор основной базы данных может размещаться как в том же приложении, так и на удаленном сервере. Простейший вариант — размещение набора **(Table. Query** или любого другого) в том же приложении. Этот набор должен обычным образом быть связанным с базой данных.

В свойстве **DataSet** провайдера указывается основной набор данных. В свойстве **ProviderName** клиентского набора указывается провайдер. Если связь клиентского набора с сервером устанавливается брокером, то этот брокер должен быть указан в свойстве **ConnectionBroker.** В многопоточных приложениях в свойстве **RemoteServer** должен указываться компонент, используемый для связи с сервером приложений.

Так же, как в автономных базах данных, в свойстве **FileName** компонента клиентского набора указывается имя файла, в котором хранится база данных. Свойство **FileName** задается, если компонент должен всегда читать и записывать данные одного определенного файла. Если файл, указанный в **FileName,** существует, то при каждом открытии набора данных в него будут загружаться данные из этого файла, а при каждом закрытии набора данных в него будут записываться данные из набора. Можно также осуществлять чтение данных из файла методом **LoadFromFile.**

Фиксация исправлений в портфельном наборе данных или отказ от сделанных исправлений осуществляется с помощью свойства **SavePoint.** Пересылка исправлений в основную базу данных осуществляется методом **ApplyUpdates.**

Если в некоторый момент надо прочитать опять в портфельный набор всю базу данных, это можно сделать, удалив файл базы, а затем закрыв и открыв клиентский набор данных.

Имеется возможность создавать портфельный набор, перекачивая в него не всю основную базу данных, а только записи, удовлетворяющие некоторым критериям. Это делается с помощью фильтрации, как показано в гл. 4 в примере раздела «DataRequest, OnDataRequest — методы и событие».

Клиентские наборы данных могут выступать в качестве удобного интерфейса к другими наборами данных, обеспечивая пользователю расширенные возможности по упорядочиванию данных, их фильтрации, расчету совокупных характеристик по всем или по отдельным группам записей. Создание интерфейсного набора данных осуществляется так же, как создание портфельного набора с единственным отличием: не надо задавать имя файла, в котором хранятся данные. Кроме того, надо позаботиться, чтобы перед окончанием сеанса работы все незафиксированные изменения были переданы в основную базу данных или отменены.

Теперь рассмотрим, в чем состоят особенности клиентских наборов данных при выполнении традиционных операций: индексация данных и их фильтраци. Клиентские наборы данных обладают расширенными возможностями по упорядочиванию их отображения. В клиентских наборах можно осуществлять сортировку заданием списка полей в свойстве **IndexFieldNames.** Причем в этом свойстве можно указывать список любых полей, а не только тех, которые имеются в предопределенных индексах. Но существенно более широкие возможности открывает метод **AddIndex.** Он позволяет добавлять индекс, задавая при этом для каждого поля направление сортировки: нарастающее или убывающее. Индексы, добавляемые методом **Addlndex,** поддерживают группировку и вычисление совокупных характеристик. Все это дает возможность пользователю задавать различные способы индексации непосредственно во время выполнения, как показано в гл. 4, в примерах, приведенных при описании метода **Addlndex.**

Клиентские наборы данных обладают также расширенными возможностями фильтрации записей. Свойство **Filter** поддерживает в них множество операций

и функций (см. в гл. 3), недоступных, например, в таком традиционном наборе данных, как **TTable.** При использовании клиентского набора данных как портфельного можно также отфильтровать записи, записываемые в набор из основной базы данных. Эта возможность рассмотрена в гл. 4, в примере раздела «DataRequest, OnDataRequest — методы и событие».

Клиентские наборы данных позволяют вычислять совокупные характеристики. Под совокупными характеристиками понимается число записей, сумма значений какого-то поля, среднее значение поля и т.п. Вычисление может проводиться по всем записям набора данных, или по некоторой их совокупности.

Имеется два способа вычисления совокупных характеристик: использование свойства **Aggregates** и создание полей совокупных характеристик. Оба способа рассмотрены в гл. 3 в описании свойства Aggregates. Вычисление совокупных характеристик клиентского набора данных может осуществляться по всем записям набора, или по некоторой совокупности записей. В последнем случае требуемые записи должны быть выделены в некоторый уровень группирования (см. в [1] и [3]).

CoIorBox — выпадающий список для выбора цвета

Выпадающий список для выбора цвета.

Страница библиотеки *Additional*

Класс *TColorBox*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TCustomComboBox — TCustomColorBox*

Модуль *ExtCtrls*

Пример изображения

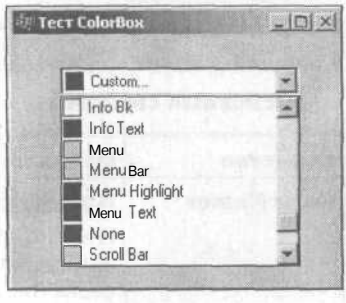


Рис. 2.8
Пример компонента, содержащего строки «Custom» и «None» при включенной опции cbPrettyNames

Описание

Компонент **CoIorBox,** введенный в C++Builder 6, представляет собой выпадающий список цветов. Может использоваться как простой и удобный способ выбора цвета пользователем.

Свойство **Style** является множеством, элементы которого определяют, какие именно категории цвета представлены в списке: стандартные, системные, дополнительные. Этим свойством может быть также задано наличие строки с заказным цветом. Подобная строка с надписью «Custom...» появляется первой в списке. При выборе ее открывается стандартный диалог Windows выбора цвета, в котором пользователь может определить заказной (нестандартный) цвет.

Свойство **Style** позволяет также включить в список цветов строки «clDefault» — цвет компонента по умолчанию, и «clNone» — цвет, зависящий от версии Windows — белый для Windows 98, черный для Windows NT/2000/XP. Если присвоить цвет **clDefault** какому-то компоненту, то компонент будет рисоваться цветом, который заложен в него по умолчанию. Аналогично, присваивание **clNone** тоже приведет к тому, что истинный цвет будет определяться самим компонентом.

Еще одна опция свойства **Style** — **cbPrettyNames** обеспечивает отображение имен цветов без префикса «cl». Например, «Black» (при **cbPrettyNames = true**) вместо «clBlack» (при **cbPrettyNames = false).**

Свойство **DefaultColorColor** определяет, квадратиком какого цвета будет помечена в списке строка «clDefault». Свойство **NoneColorColor** определяет, квадратиком какого цвета будет помечена в списке строка «clNone». При этом, как сказано выше, в действительности присваиваемые цвета будут определяться теми компонентами, в которые они передаются.

Узнать цвет, выбранный пользователем в списке, позволяет свойство **Selected.** Для этого можно воспользоваться, например, событием компонента **OnSelect,** наступающим в момент выбора пользователем цвета. Например, оператор

```
Memo1->Color = ColorBox1->Selected;
```

помещенный в обработчик этого события, задаст фону окна **Memo1** цвет, выбранный пользователем.

Свойство **Colors** является индексированным массивом цветов в списке (индексы начинаются с 0). Свойство **ColorNames** — аналогичный массив строк с именами цветов.

Большинство остальных свойств, методов, событий подобны компоненту **ComboBox.** В частности, список всех строк содержится в свойстве **Items** типа TStrings. Индекс строки цвета, которая будет показана пользователю в момент начала выполнения приложения, определяется свойством только времени выполнения **ItemIndex.** Если вам желательно в первый момент показать пользователю определенный цвет, это можно сделать в обработчике события формы **OnCreate,** определив в нем **ItemIndex** с помощью метода **IndexOf.** Например, оператор:

```
ColorBox1->ItemIndex = ColorBox1->Items->IndexOf("clDefault");
```

или (при **cbPrettyNames = true**):

```
ColorBox1->ItemIndex = ColorBox1->Items->IndexOf("Default");
```

в первый момент показывает пользователю строку цвета по умолчанию.

Основные свойства

Свойство	Объявление / Описание
ColorNames	**AnsiString ColorNames[int Index]** Индексированный массив строк с именами цветов. Индексы начинаются с 0. В зависимости от опции cbPrettyNames свойства Style имена начинаются с префикса «cl» («clBlack») или записываются без этого префикса («Black»)
Colors	Graphics::TColor **Colors[int** Index] Индексированный массив строк цветов. Индексы начинаются с 0. Опции свойства Style определяют, какие цвета содержатся в списке
DefaultColor Color	Graphics::TCoIor DefaultColorColor Определяет, квадратиком какого цвета будет помечена в списке строка «clDefault». Истинный цвет определяется компонентом, воспринимающим этот цвет (см. описание компонента ColorBox). Свойство работает, только если в Style заданы опции cbSystemColors и **cbIncludeDefault**
DropDown Count	int DropDownCount Определяет максимальное число строк, отображаемое в раскрывающемся списке

Свойство	Объявление / Описание
ItemIndex	int ItemIndex Указывает порядковый номер элемента, выделенного в выпадающем списке
Items	Classes::TStrings* Items Массив строк списка — объект класса TStrings. Свойства этого класса позволяют формировать и изменять список
NoneColorColor	Graphics::TColor NoneColorColor Определяет, квадратиком какого цвета будет помечена в списке строка «clNone». Истинный цвет определяется компонентом, воспринимающим этот цвет (см. описание ColorBox). Свойство работает, только если в Style заданы опции cbSystemColors и cbIncludeNone
Selected	TColor Selected Выбранный пользователем цвет. Его удобно определять в обработчике события OnSelect
Style	enum TColorBoxStyles {cbStandardColors, cbExtendedColors, cbSystemColors, cbIncludeNone, cbIncludeDefault, cbCustomColor, cbPrettyNames}; typedef Set<TColorBoxStyles, cbStandardColors, cbPrettyNames> TColorBoxStyle; TColorBoxStyle Style Множество опций, определяющих состав цветов в списке

ComboBox — выпадающий список строк

Отображает список строк в развернутом виде или в виде выпадающего списка, позволяет пользователю выбрать из списка необходимую строку или задать в качестве выбора собственный текст.

Страница библиотеки *Standard*

Класс *TComboBox*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomComboBox*

Модуль *stdctrls*

Примеры изображения

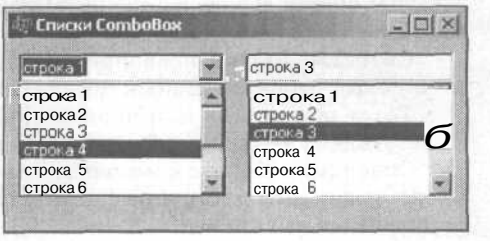


Рис. 2.9
Компоненты ComboBox: слева список в стиле csDropDown, справа — csSimple

Описание

Компонент **ComboBox** объединяет функции компонентов **ListBox** — списка, и **Edit** — окна редактирования. Компонент позволяет пользователю выбрать из

списка необходимую строку или задать в качестве выбора собственный текст. Список может отображаться в развернутом виде или как выпадающий список.

Отличие **ComboBox** от схожего по функциям компонента **ListBox** заключается в следующем:

- **ComboBox** разрешает пользователю редактировать список, а **ListBox** не разрешает
- **в ComboBox** список может быть развернут или свернут, а в **ListBox** он всегда развернут
- **ListBox** может допускать множественный выбор, а в **ComboBox** пользователь всегда должен выбрать только один элемент

Основное свойство компонента, содержащее список строк, — Items, имеющее тип **TStrings.** Заполнить его во время проектирования можно, нажав кнопку с многоточием около этого свойства в окне Инспектора Объектов. Во время выполнения работать с этим свойством можно, пользуясь свойствами и методами класса **TStrings — Clear, Add** и другими.

Стиль изображения списка определяется свойством **Style:**

csDropDown	Выпадающий список со строками одинаковой высоты и с окном редактирования, позволяющим пользователю вводить или редактировать текст (слева на рисунке)
csSimple	Развернутый список со строками одинаковой высоты и с окном редактирования, позволяющим пользователю вводить или редактировать текст (справа на рисунке)
csDropDownList	Выпадающий список со строками одинаковой высоты, не содержащий окна редактирования
csOwnerDrawFixed	Выпадающий список типа csDropDown с графической прорисовкой элементов одинаковой высоты, задаваемой свойством ItemHeight
csOwnerDrawVariable	Выпадающий список типа csDropDown с графической прорисовкой элементов, которые могут иметь различную высоту

Выбор пользователя или введенный им текст можно определить по значению свойства **Text.** Индекс выбранного пользователем элемента списка можно определить по свойству **ItemIndex.** Если в окне проводилось редактирование данных, то **ItemIndex** = -1. По этому признаку можно определить, что редактирование проводилось. Начальное значение **Itemlndex** можно задать во время проектирования, только если список (свойство Items) заполнен.

Свойство **MaxLength** определяет максимальное число символов, которые пользователь может ввести в окно редактирования. Если **MaxLength** = 0, то число вводимых символов не ограничено.

Свойство **Sorted** позволяет упорядочить список по алфавиту. При **Sorted = true** новые строки в список добавляются не в конец, а по алфавиту.

Текст выбранной или написанной пользователем строки находится в свойстве **Text.** Индекс выбранной строки можно узнать из свойства **Itemlndex.**

Основное событие компонента — **OnChange** наступает при изменении текста в окне редактирования в результате прямого редактирования текста или в результате выбора из списка.

При значениях стиля **Style,** равных **csOwnerDrawFixed и csOwnerDrawVariable** редактирование текста в окне списка невозможно. Работа с графикой производится в обработчиках событий **OnDrawItem и OnMeasureItem.** Событие

OnDrawItem наступает, когда должна рисоваться какая-то строка списка. Заголовок обработчика этого события имеет вид:

```
void TForm1::ComboBox1DrawItem(TWinControl *Control,
                    int Index, TRect &Rect, TOwnerDrawState State)
```

Параметр **Control** является указателем на список, в котором происходит событие. Параметр **Index** указывает индекс элемента, который должен быть перерисован. Параметр **Rect** типа **TRect** (см. в гл. 1) указывает область канвы списка, соответствующую рисуемому элементу списка. Параметр **State** типа **TOwnerDrawState** является множеством, элементами которого могут быть значения **odSelected** — строка выделена, **odFocused** строка находится в фокусе и ряд других.

В обработчике события **OnDrawItem** надо методами работы на канве (см. в гл. 1 описание типа **TCanvas**) нарисовать изображение элемента.

При значении **Style,** равном **lbOwnerDrawFixed,** перед прорисовкой наступает только событие **OnDrawItem. При Style = lbOwnerDrawVariable** перед этим событием наступает другое — **OnMeasureItem,** в котором надо указать высоту элемента. Заголовок обработчика этого события имеет вид:

```
void TForm1::ListBox1MeasureItem(TWinControl *Control,
                                 int Index, int &Height)
```

Параметры **Control и Index** имеют тот же смысл, что и в обработчике **OnDrawItem,** а значение параметра **Height** надо задать равным высоте данного элемента списка.

Подробно работа с графикой в списках рассмотрена в [1].

Основные свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данным компонентом
Align	**enum** TAlign { alNone, **alTop, alBottom,** alLeft, alRight, alClient, **alCustom** } TAlign Align Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Anchors	enum TAnchorKind { akLeft, akTop, akRight, **akBottom** }; typedef **Set<TAnchorKind,** akLeft, **akBottom>** TAnchors; TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
DropDown Count	int **DropDownCount** Определяет максимальное число элементов, отображаемое в раскрывающемся списке без появления полосы прокрутки. Действует при всех значениях Style, кроме csSimple
ItemHeight	int ItemHeight Указывает высоту элементов, в пикселах, в выпадающем списке
ItemIndex	int ItemIndex Указывает порядковый номер элемента, выделенного в списке
Items	**Classes::TStrings*** Items Массив строк списка — объект класса TStrings. Свойства этого класса позволяют формировать и изменять список

Свойство	Объявление / Описание
SelLength	int SelLength Определяет количество выделенных символов в окне редактирования
SelStart	int SelStart Указывает позицию первого выделенного символа в окне редактирования или, если выделения нет, то позицию курсора в нем
SelText	AnsiString SelText Текст, выделенный в окне редактирования
Sorted	bool Sorted Указывает, должны ли строки в списке автоматически сортироваться в алфавитном порядке
Style	enum TComboBoxStyle {csDropDown, csSimple, csDropDownList, csOwnerDrawFixed, csOwnerDrawVariable}; TComboBoxStyle Style Определяет стиль отображения списка (см. выше в описании ComboBox)

Основные методы

Метод	Объявление / Описание
Clear	void Clear(void) Удаляет все элементы списка
SelectAll	void SelectAll(void) Выделяет весь текст в окне редактирования
SetFocus	void SetFocus(void) Передает фокус элементу, активизирует его

Основные события

Событие	Описание
OnChange	Наступает при изменении текста в окне редактирования в результате прямого редактирования текста или в результате выбора из списка. В обработчике можно прочитать текст Text и индекс выбранного элемента ItemIndex (-1, если был не выбор, а редактирование)
OnDrawItem	Наступает при необходимости перерисовать элемент списка (см. выше описание ComboBox)
OnDropDown	Событие происходит, когда пользователь открывает раскрывающийся список, щелкая на стрелке справа от компонента
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод

Событие	Описание
OnKeyUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу
OnMeasure Item	Наступает при необходимости перерисовать элемент в списке с изменяемой высотой элементов (см. выше описание ComboBox)

ComboBoxEx — выпадающий список строк текста с изображениями

Отображает список строк с изображениями в развернутом виде или в виде выпадающего списка, позволяет пользователю выбрать из списка необходимую строку или задать в качестве выбора собственный текст.

Страница библиотеки *Win32*

Класс *TComboBoxEx*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TCustomListControl — TCustomCombo — TCustomComboBoxEx — TBiDiComboBoxEx*

Модуль *StdCtrls*

Пример изображения

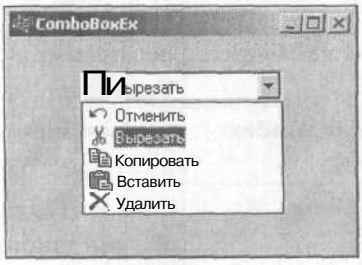


Рис. 2.10
Пример компонента ComboBoxEx

Описание

Компонент ComboBoxEx во многом подобен **ComboBox.** Различие, прежде всего, заключается в том, что в **ComboBoxEx** легче, чем в **ComboBox,** вводить изображения в элементы списка. С другой стороны, некоторые возможности **ComboBox,** например, возможность сортировки, в **ComboBoxEx** не поддерживаются. Невозможно также в этом списке заказное рисование на канве.

Изображения, отображаемые в элементах списка, должны содержаться в компоненте **ImageList.** Ссылка на этот компонент задается в свойстве **Images.** Ввод элементов списка во время проектирования осуществляется редактором коллекций, вызываемым шелчком на кнопке с многоточием в окне Инспектора Объектов около свойства **ItemsEx.** Перед вами откроется окно, в котором кнопка Add New позволяет ввести новый элемент. Если вы ввели элемент и выделили его, то в окне Инспектора Объектов увидите его свойства. Основные из них: **Caption** — текстовая строка, **ImageIndex** — индекс соответствующей пиктограммы в списке **ImageList, Indent** — отступ от левого поля элемента списка.

Свойство **Style** определяет стиль элемента и может быть равным **csExDropDown** — выпадающий список с окном редактирования, **csExSimple** — развернутый список с окном редактирования, **csExDropDownList** — выпадающий список без окна редактирования, так что пользователь не может ввести свой текст.

Более подробное описание работы со списком см. в разделе, посвященном компоненту **ComboBox.**

Основные свойства

Свойство	Объявление / Описание
Align	enum TAlign { alNone, **alTop,** alBottom, alLeft, alRight, alCIient, **alCustom** } TAlign Align Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Anchors	enum TAnchorKind { **akLeft,** akTop, akRight, **akBottom** }; typedef **Set<TAnchorKind,** akLeft, akBottom> TAnchors; **TAnchors** Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
DropDown Count	int DropDownCount Определяет максимальное число элементов, отображаемое в раскрывающемся списке без появления полосы прокрутки. Действует при всех значениях Style, кроме csSimple
Images	**TCustomImageList*** Images Ссылка на компонент ImageList, содержащий список изображений
ItemHeight	int ItemHeight Указывает высоту элементов, в пикселах, в выпадающем списке
ItemIndex	int Itemlndex Указывает порядковый номер элемента, выделенного в списке
Items	**Classes::TStrings*** Items Массив текстовых строк списка — объект класса TStrings. Свойства этого класса позволяют формировать и изменять список
ItemsEx	**TComboExItems*** Items Коллекция объектов элементов списка, основные свойства которых описаны выше
SelLength	int SelLength Определяет количество выделенных символов в окне редактирования
SelStart	int SelStart Указывает позицию первого выделенного символа в окне редактирования или, если выделения нет, то позицию курсора в нем
SelText	AnsiString SelText Текст, выделенный в окне редактирования
Style	enum **TComboBoxExStyle** {csExDropDown, csExSimple, **csExDropDownList};** TComboBoxExStyleStyle Определяет стиль отображения списка (см. в описании компонента)

Свойство	Объявление / Описание
StyleEx	enum TComboBoxExStyleEx {csExCaseSensitive, csExNoEditImage, csExNoEditImageIndent, csExNoSizeLimit, csExPathWordBreak}; typedef Set<TComboBoxExStyleEx, csExCaseSensitive, csExPathWordBreak> TComboBoxExStyles; TComboBoxExStyles StyleEx Множество флагов, определяющих стиль изображения: **csExCaseSensitive** — чувствительность к регистру, **csExNoEditImage** — строки появляются без изображений, **csExNoEditImageIndent** — отсутствие отступов в строках, **csExNoSizeLimit** — управляет перестроением высоты элементов, **csExPathWordBreak** — символы "\", "/" и "." трактуются как разделители. Это облегчает навигацию с использованием путей и URL (в Windows 9.x не поддерживается)

Основные методы

Метод	Объявление / Описание
Clear	void Clear(void) Удаляет все элементы списка
SelectAll	void SelectAll(void) Выделяет весь текст в окне редактирования
SetFocus	void SetFocus(void) Передает фокус элементу, активизирует его

Наследуется также много методов классов-предшественников.

Основные события

Событие	Описание
OnBeginEdit	Наступает в начале редактирования пользователем текста в окне редактирования
OnChange	Наступает при изменении текста в окне редактирования в результате прямого редактирования текста или в результате выбора из списка. В обработчике можно прочитать текст Text и индекс выбранного элемента ItemIndex (-1, если был не выбор, а редактирование)
OnEndEdit	Наступает после редактирования пользователем текста в окне редактирования
OnDropDown	Событие происходит, когда пользователь открывает раскрывающийся список, щелкая на стрелке справа от компонента
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKeyUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу

ControlBar — контейнер инструментальных панелей

Используется в качестве контейнера перестраиваемых инструментальных панелей с использованием технологии Drag&Doc.

Страница библиотеки *Additional*

Класс *TControlBar*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomControl — TCustomControlBar*

Модуль *Extctrls*

Пример изображения

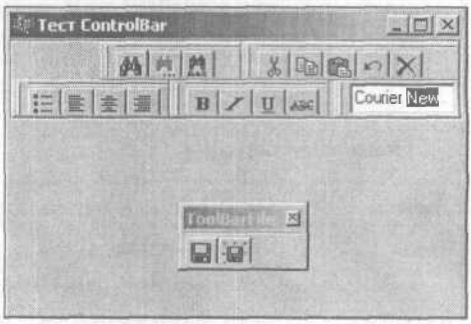


Рис. 2.11
Пример ControlBar. В середине - панель ToolBar, вынутая из ControlBar и ставшая плавающим окном

Описание

Компонент **ControlBar,** как и компонент **CoolBar.** относится к числу перестраиваемых панелей и служит для составления сложных инструментальных панелей, состоящих из ряда других панелей, размещенных на полосах **ControlBar.** Отличие от **CoolBar** заключается в широком применении техники перетаскивания и встраивания Drag&Doc, которую вы можете видеть в инструментальных панелях C++Builder **6.** На **ControlBar** можно поместить различные компоненты, например, инструментальные панели **ToolBar,** окна редактирования **Edit** и др. Каждый компонент, попадая на **ControlBar,** получает полосу захвата (см. на рис. 2.11), свойственную технологии Drag&Doc. В дальнейшем пользователь сможет все эти компоненты перемещать по **ControlBar и** даже вынимать из **ControlBar,** превращая в самостоятельные плавающие окна (см. рис. 2.11).

Свойство AutoDrag компонента **ControlBar** определяет, можно (при значении **true),** или нельзя простым перетаскиванием вынести полосу за пределы **ControlBar.** Но чтобы воспользоваться этой возможностью надо у компонентов, размещенных на **ControlBar,** установить свойства **DragMode = dmAutomatic и DragKind = dkDock.** Это будет означать автоматическое выполнение операций Drag&Doc. Если этого не сделать, допускается только перемещение компонентов по полосам **ControlBar.**

Свойства **RowSize и RowSnap** компонента **ControlBar** определяют процедуру встраивания. Свойство **RowSize** задает размеры полос, в которые могут встраиваться компоненты, а **RowSnap** определяет захват полосами встраиваемых компонентов.

Свойство **AutoDock,** установленное в **true,** обеспечивает временное встраивание компонента, перетаскиваемого над панелью, в **ControlBar.** Это позволяет пользователю наглядно представлять результат перетаскивания. Причем это относится не только к компонентам, первоначально находившимся на **ControlBar,** но и к любому перетаскиваемому и встраиваемому компоненту.

Основные свойства

Свойство	Объявление / Описание
Align	enum TAlign {alNone, **alTop,** alBottom, alLeft, alRight, alClient, **alCustom};** TAlign Align Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Anchors	enum **TAnchorKind** { **akLeft,** akTop, akRight, akBottom } ; typedef **Set<TAnchorKind,** akLef t, akBottom> **TAnchors;** TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
AutoDock	**bool** AutoDock Обеспечивает временное встраивание компонента, перетаскиваемого над панелью, в ControlBar
AutoDrag	bool AutoDrag Определяет, можно (при значении true), или нельзя простым перетаскиванием вынести полосу за пределы ControlBar
AutoSize	bool AutoSize Определяет, будет ли высота панели автоматически подгоняться под размеры расположенных на ней и перетаскиваемых компонентов
BevelEdges	enum TBevelEdge { beLeft, beTop, beRight, beBottom }; typedef Set<TBevelEdge, beLeft, beBottom> TBevelEdges; TBevelEdges BevelEdges Определяет, какая граница компонента будет иметь обрамление. Является множеством, которое может содержать beLeft — левая, **beTop** — верхняя, beRight — правая, beBottom — нижняя
BevelInner	enum TBevelCut {bvNone, **bvLowered,** bvRaised, **bvSpace};** TBevelCut Bevellnner Определяет выпуклый, утопленный или плоский вид внутренней части компонента
BevelKind	enum TBevelKind {bkNone, bkTile, bkSoft, **bkFlat};** TBevelKind BevelKind Определяет в комбинации с BevelWidth, Bevellnner и BevelOuter тип обрамления компонента
BevelOuter	enum TBevelCut **{bvNone,** bvLowered, bvRaised, **bvSpace};** TBevelCut BevelOuter Определяет выпуклый, утопленный или плоский вид обрамления компонента
BevelWidth	typedef int **TBevelWidth;** **TBevelWidth** BevelWidth Определяет ширину обрамления компонента в пикселах
BorderWidth	typedef unsigned **TBorderWidth;** **TBorderWidth** BorderWidth Расстояние в пикселах между наружной и внутренней кромками обрамления

Свойство	Объявление / Описание
Picture	Graphics::**TPicture*** Picture Определяет рисунок — шаблон, заполняющий фон панели
RowSize	typedef unsigned TRowSize; **TRowSize** RowSize Определяет высоту полос, на которых располагаются компоненты
RowSnap	**bool** RowSnap Определяет захват полосами встраиваемых компонентов. Если RowSnap = **true**, то высота встраиваемых компонентов становится равной высоте полос RowSize

Методы

Никаких специальных методов, вызываемых из приложения, в компоненте не объявлено

Основные события

Событие	Описание
OnBandDrag	Наступает, когда пользователь начинает перетаскивание указанного компонента. В обработчике можно запретить перетаскивание
OnBandInfo	Наступает при согласовании позиции встраиваемого в панель компонента
OnBandMove	Наступает при перемещении панели
OnBandPaint	Наступает при прорисовке полос панели
OnPaint	Наступает при прорисовке панели

Помимо этого наследуется множество событий, свойственных всем оконным компонентам.

CoolBar — контейнер инструментальных панелей

Позволяет строить перестраиваемые панели, состоящие из полос, в которые могут включаться инструментальные панели и любые другие оконные компоненты.

Страница библиотеки *Win32*

Класс *TCoolBar*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TToolWindow*

Модуль *Comctrls*

Пример изображения

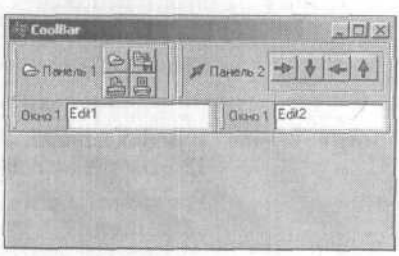


Рис. 2.12
Пример перестраиваемой панели на основе компонента CoolBar

Описание

Компонент **CoolBar** позволяет строить перестраиваемые панели, состоящие из полос. В полосы могут включаться инструментальные панели **ToolBar** и любые другие оконные компоненты: окна редактирования, панели и т.п. Каждый из этих компонентов автоматически снабжается средствами перемещения его пользователем в пределах окна CoolBar. В полосы могут вставляться и не оконные компоненты, например, метки. Но они не будут перемещаемыми.

Свойства полос вы можете задавать редактором полос, который можно вызвать тремя способами: из Инспектора Объектов кнопкой с многоточием около свойства **Bands,** двойным щелчком на компоненте **CoolBar** или из контекстного меню, выбрав команду Bands Editor. В окне этого редактора вы можете перемещаться по полосам, добавлять новые полосы или уничтожать существующие. При перемещении по полосам в окне Инспектора Объектов вы будете видеть свойства полос. Свойство **Control** определяет размещенный на полосе компонент. Свойство **Break** определяет, занимает ли полоса весь соответствующий размер контейнера **CoolBar,** или обрывается. Свойство **Text** задает текст, который может появиться в начале соответствующей полосы (см. надписи «Панель ...» и «Окно ...» на рис. 2.12). Вместо свойства **Text** (или наряду с ним) можно задать свойство **ImageIndex** — индекс списка изображений **ImageList.** ссылка на который задается свойством Images. Указанные таким образом изображения появятся в начале соответствующих полос (см. верхние полосы на рис. 2.12).

Свойства **MinHeight и MinWidth** определяют минимальную высоту и ширину полосы при перестроениях пользователем полос панели. Свойство **FixedSize** определяет, фиксирован ли размер данной полосы, или он может изменяться пользователем. По умолчанию для всех полос **FixedSize = false,** т.е. все полосы перестраиваются. Но при желании размеры некоторых полос можно зафиксировать, задав для них **FixedSize = true.**

Для компонента **CoolBar** в целом, помимо обычных для других панелей свойств, надо обратить внимание на свойство **BandMaximize.** Оно определяет действие, которым пользователь может установить максимальный размер полосы, не перетаскивая ее границу: **bmNone** — такое действие не предусмотрено, **bmClick** — щелчком мыши, **bmDblClick** — двойным щелчком.

Свойство **FixedOrder,** если его установить в **true,** не разрешит пользователю в процессе перемещений полос изменять их последовательность. Свойство **Vertical** указывает вертикальное или горизонтальное расположение полос.

Основные свойства

Свойство	Объявление / Описание
Align	**enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom}** **TAlign Align** Определяет способ выравнивания панели (по умолчанию **alTop)**
BandBorder Style	**enum TFormBorderStyle {bsNone, bsSingle, bsSizeable, bsDialog, bsToolWindow, bsSizeToolWin};** **typedef TFormBorderStyle TBorderStyle;** **Forms::TBorderStyle BandBorderStyle** Определяет наличие (bsSingle) или отсутствие (bsNone) рамки, окружающей панель
Band Maximize	**enum TCoolBandMaximize {bmNone, bmClick, bmDblClick};** **TCoolBandMaximize BandMaximize** Определяет действие, которым пользователь может установить максимальный размер полосы, не перетаскивая ее границу: **bmNone** — такое действие не предусмотрено, **bmClick** — щелчком мыши, **bmDblClick** — двойным щелчком

Свойство	Объявление / Описание
Bands	TCoolBands* Bands Собрание полос панели — объектов TCoolBand
BitMap	Graphics::TBitmap* Bitmap Изображение, заполняющее фон полос
EdgeBorders	enum TEdgeBorder {ebLeft, ebTop, ebRight, ebBottom}; typedef Set<TEdgeBorder, ebLeft, ebBottom> TEdgeBorders; TEdgeBorders EdgeBorders Определяет изображение отдельных сторон панели (левой, правой, верхней, нижней)
EdgeInner	enum TEdgeStyle {esNone, esRaised, esLowered}; TEdgeStyle EdgeInner Стиль изображения внутренней части панели (утопленный, выступающий или плоский)
EdgeOuter	enum TEdgeStyle {esNone, esRaised, esLowered}; TEdgeStyle EdgeOuter Стиль изображения внешней части панели (утопленный, выступающий или плоский)
FixedOrder	bool FixedOrder Разрешает (true) или не разрешает (false) пользователю в процессе перемещения полос изменять их последовательность
FixedSize	bool FixedSize Фиксирует размеры полос, не разрешая пользователю изменять их
Images	Imglist::TCustomImageList* Images Список изображений ImageList, которые появятся в начале соответствующих полос (свойство полосы ImageIndex задает для нее изображение из этого списка)
ShowText	bool ShowText Определяет, отображаются ли в полосах надписи, заданные свойством Text полос
Vertical	bool Vertical Указывает вертикальное или горизонтальное расположение полос

Методы

Никаких специальных методов, вызываемых из приложения, *в* компоненте не объявлено.

Основные события

Событие	Описание
OnChange	Наступает при изменении размера или местоположения полосы: ее свойств Break, Index или Width

Помимо этого наследуется множество событий, свойственных всем оконным компонентам.

Database — компонент базы данных

Невизуальный компонент, осуществляет управление транзакциями.

Страница библиотеки *BDE*, в версиях младше C++Builder 6 — *Data Access*

Класс *TDatabase*

Иерархия *TObject* — *T Persistent* — *TComponent* — *TCustomConnection*

Модуль *Dbtables*

Описание

Компонент базы данных типа **Database** автоматически включается в любое приложение C++Builder, работающее с базами данных. Если же вы хотите сознательно управлять транзакциями, вы должны явным образом включить компонент **Database** в свое приложение. Этот компонент решает следующие задачи:

- Создание соединения с удаленным сервером
- Регистрация пользователя при первом обращении к серверу
- Создание локальных псевдонимов приложений
- Управление транзакциями
- Определение уровня изоляции транзакции (регулирование одновременных транзакций к одним и тем же таблицам)

Database связывается с компонентами наборов данных **Table, Query** и другими через имя базы данных, к которой он подключается. Это имя задается в свойстве **DatabaseName.** Может быть задан псевдоним базы данных или полный путь к ней. Если задается база данных, имеющая псевдоним BDE, то свойства **AliasName, DriverName и Params** можно не задавать. В противном случае надо задать или свойство **AliasName,** или свойства **DriverName и Params.** Однако и при задании **AliasName** или **DriverName и Params** надо задать какое-то, в данном случае произвольное имя **DatabaseName.**

Установку значений всех этих свойств во время проектирования можно проводить непосредственно в Инспекторе Объектов, но удобнее воспользоваться специальным редактором, который вызывается двойным щелчком на **Database (см. рис. 2.13).**

Как уже говорилось, вы можете ограничиться заданием только имени базы данных (окно Name). Но если ваша база данных имеет псевдоним, вы можете указать и его (окно Alias Name). В этом случае можно щелкнуть на кнопке Defaults (умолчание) и в окне Parameter overrides появятся значения параметров по умолчанию. Аналогичные данные появятся, если в выпадающем списке Driver Name вы выберете драйвер, а затем щелкнете на кнопке Defaults. Вы можете внести в эти данные какие-то изменения. Например, при работе с базой данных, защищенной паролем, вы можете указать в параметрах этот пароль и сбросить индикатор Login prompt — приглашение к соединению, которое запрашивает пароль. Тогда при запуске вашего приложения не будет каждый раз запрашиваться пароль. Это облегчит работу пользователей, но, конечно, снимет защиту вашей базы данных.

Индикатор Keep inactive connection устанавливает свойство **KeepConnection,** о котором будет сказано ниже.

После всех установок в редакторе щелкните на OK и введенные вами установки заполнят значения свойств **DatabaseName, AliasName, DriverName, Params и KeepConnection.**

Свойство **Connected** (соединение) совместно со свойством **KeepConnection** управляют процессом соединения компонентов с базой данных. Если **KeepConnection** равно **true,** то соединение с базой данных постоянное даже при отсутствии открытых наборов данных. Если же **KeepConnection** равно **false,** то для регистрации на сервере надо устанавливать **Connected в true** при каждом открытии таблицы.

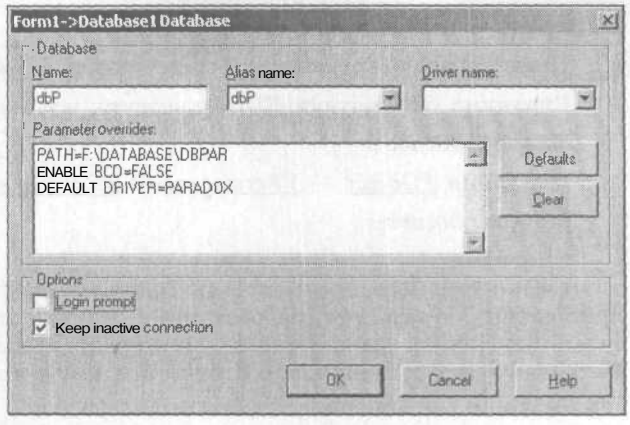


Рис. 2.13
Окно редактора компонента Database

Свойство **TransIsolation** определяет уровень изоляции транзакции. Это свойство может иметь значения:

tiDirtyRead	Позволяет читать все текущие изменения, проводимые другими транзакциями до их фиксации
tiReadCommit	Позволяет читать только зафиксированные изменения, проводимые другими транзакциями. Это значение принято по умолчанию
tiRepeatableRead	После начала транзакции не позволяет читать даже подтвержденные изменения, проводимые другими транзакциями в прочитанных данных. Следовательно, при повторном прочтении на протяжении данной транзакции той же записи будут получены прежние результаты, даже если другие транзакции их уже изменили

Указанные выше значения свойства **Translsolation** могут не поддерживаться на конкретном сервере, с которым вы работаете. В этом случае сервер переходит на доступный ему более высокий, чем запрошенный, уровень изоляции. Ниже приведена таблица уровней для различных систем.

Translsolation	Interbase	Oracle	Sybase & Microsoft
tiDirtyRead	Read **commited**	Read commited commited	Read
tiReadCommit	Read commited	Read commited	Read commited
tiRepeatableRead	Repeatable Read	Repeatable Read (READ ONLY)	Error (не поддерживается)

Основные методы компонента **Database: StartTransaction** — начало транзакции, **Commit** — фиксация изменений в базе данных, **Rollback** — откат, отмена всех изменений. Программа работы с данными должна строиться по следующей схеме:

```
Database1->StartTransaction();

Группа операторов изменения данных (ExecSQL и др.)

Проверка результатов: Если успешно - Database1->Commit();
                      Если неудача - Database1->Rollback();
```

Основные свойства

Свойство	Объявление / Описание
AliasName	AnsiString AliasName Псевдоним BDE базы данных. При установке AliasName свойство DriverName автоматически очищается
Connected	bool Connected Указывает, активно ли соединение с базой данных
DatabaseName	AnsiString DatabaseName Указывает имя базы данных, с которой связан компонент
DataSetCount	int DataSetCount Число активных наборов данных, связанных с компонентом
DataSets	TDBDataSet* DataSets[int Index] Индексированный массив активных наборов данных, связанных с компонентом
Directory	AnsiString Directory Рабочий каталог для баз данных Paradox и dBASE
DriverName	AnsiString DriverName Имя используемого драйвера BDE
Exclusive	bool Exclusive Запрещает или разрешает доступ к базе данных другим приложениям
KeepConnection	bool KeepConnection Указывает, остается ли приложение соединенным с базой данных даже если нет отрытой базы данных
Locale	void * Locale Драйвер языка BDE, Используется только при явном обращении к функциям BDE. Только для чтения
LoginPrompt	bool LoginPrompt Указывает, должен ли появляться диалог соединения (имя пользователя, пароль) при открытии базы данных
Session	TSession* Session Указывает компонент Session, связанный с базой данных
SessionName	AnsiString SessionName Имя компонента сеанса сетевого соединения Session, связанного с базой данных
Readonly	bool Readonly Устанавливает данные только для чтения (при значении true)
TransIsolation	enum TTransIsolation {tiDirtyRead, tiReadCommitted, tiRepeatableRead}; TTransIsolation TransIsolation Определяет уровень изоляции транзакций (см. пояснения выше в описании Database)

Основные методы

Метод	Объявление / Описание
ApplyUpdates	void ApplyUpdates(const TDBPataSet* const * DataSets, const int DataSets_Size) Переносит кэшированные изменения в базу данных
Close	void Close(void) Закрывает соединение
CloseDataSets	void CloseDataSets(void) Закрывает все наборы данных, связанные с компонентом
Commit	void Commit(void) Фиксирует в базе данных все изменения, произведенные транзакцией, и завершает транзакцию
Open	void Open(void) Открывает соединение
Rollback	void Rollback(void) Производит откат назад, аннулируя все изменения данных, произведенные на протяжении транзакции
Start Transaction	void StartTransaction(void) Начинает новую транзакцию

События

Событие	Описание
OnLogin	Наступает при соединении приложения с базой данных
AfterConnect	Наступает после установки соединения с базой данных
AfterDisconnect	Наступает после закрытия соединения с базой данных
BeforeConnect	Наступает перед установкой соединения с базой данных
BeforeDisconnect	Наступает перед закрытием соединения с базой данных

DataSource — источник данных

Невизуальный компонент источника данных — обеспечивает интерфейс между компонентом набора данных и средствами визуализации.

Страница библиотеки *Data Access*

Класс *TDataSource*

Иерархия *TObiect — TPersistent — TComponent*

Модуль *Db*

Описание

Компонент **DataSource** представляет собой источник данных, который обеспечивает связь между набором данных и компонентами отображения и редактирования данных.

Все наборы данных должны быть связаны с компонентом источника данных, если требуется редактирование данных.

Основное свойство источника данных — **DataSet.** Оно указывает на компонент набора данных **(Table, Query** и др.), с которым связан источник. Свойство **State** дает информацию о текущем состоянии набора данных: находится ли он в состоянии просмотра, редактирования, вставки данных и т.п.

Основные свойства

Свойство	Объявление / Описание
AutoEdit	**bool AutoEdit** **Определяет, вызывается ли автоматически метод редактирования данных Edit**
DataSet	**TDataSet* DataSet** **Набор данных, с которым связан компонент**
Enabled	**bool Enabled** **Определяет, отображаются ли данные в компонентах отображения, связанных с этим источником данных**
State	**enum TDataSetState {dsInactive, dsBrowse, dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc, dsOpening};** **TDataSetState State** **Указывает текущее состояние набора данных, связанного с источником данных**

Основные методы

Метод	Объявление / Описание
Edit	void Edit(void) Подтверждает возможность редактирования набора данных (нельзя путать этот метод с методом Edit набора данных TDataSet)
IsLinkedTo	bool IsLinkedTo(TDataSet* DataSet) Проверяет связь источника с указанным набором данных. Обычно напрямую не используется, а применяется при разработке новых компонентов

События

Событие	Описание
OnDataChange	Наступает при перемещении на новое поле или новую запись, если текущая запись редактировалась
OnStateChange	Наступает при изменении состояния набора данных, связанного с данным источником
OnUpdateData	Наступает, когда намечается обновление текущей записи

DBCheckBox — индикатор, связанный с данными

Позволяет отображать и редактировать данные поля булева типа и некоторых других типов.

Страница библиотеки *DataAccess*

Класс *TDBCheckBox*

Иерархия *TObiect — TPersistent— TComponent— TControl— TWincontrol— TButtonControl— TCustomCheckBox*

Модуль *Dbctrls*

Описание

DBCheckBox — связанный с данными аналог обычного индикатора **CheckBox.** Он позволяет отображать и редактировать данные поля булева типа, а также символьного и числового типа. Если при выводе данных булево поле имеет значение **true,** то индикатор включается. А если в процессе редактирования пользователь включит или выключит индикатор, то соответственно значение **true или false** запишется в поле. Это один из способов обеспечить пользователю безошибочный ввод значений в булево поле.

Если поле символьное или числовое, то в свойство **ValueChecked** заносится строка, перечисляющая значения поля, при которых индикатор включается, а в свойстве **ValueUnchecked** перечисляются значения, при которых индикатор выключается. При значениях, не перечисленных ни в **ValueChecked, ни в ValueUnchecked,** индикатор переходит в третье состояние, отображая серый флажок (даже если нет разрешения на третье состояние — свойство **AllowGrayed = false).**

Включен или выключен индикатор можно определить по значению свойства **Checked,** но только во время выполнения. Более детально состояние индикатора можно определить по свойству **State: cbChecked** — включен, **cbUnchecked** — выключен, **cbGrayed** — третье состояние.

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

При переключении пользователем индикатора в поле заносится значение, определенное свойствами **ValueChecked и ValueUnchecked** (если в них записано йо нескольку значений, то заносится соответствующее первое значение).

Подробнее о свойствах, методах, событиях см. в описании **CheckBox.**

DBEdit — окно редактирования, связанное с данными

Представляет собой окно редактирования, которое может отображать и редактировать поле набора данных.

Страница библиотеки *DataAccess*

Класс *TDBEdit*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWincontrol — TCustomEdit*

Модуль *Dbctrls*

Описание

DBEdit — связанный с данными аналог обычного окна редактирования Edit. Он позволяет отображать и редактировать данные полей различных типов: строка, число, булева величина. Преобразование значения поля в строку текста, отображаемую в **DBEdit,** производится автоматически. Если задать в компоненте **ReadOnly = true,** то он, **как и DBText.** будет служить элементом отображения, но несколько более изящным, чем **DBText.**

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**

- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Большинство остальных свойств **(AutoSelect, AutoSize, CharCase** и др.) аналогичны свойствам компонента **Edit. Но в DBEdit** главное свойство окна — **Text** недоступно во время проектирования. Текст в окне определяется соответствующим полем текущей записи и может изменяться пользователем или программно во время выполнения. Отредактированное значение в окне помещается в соответствующее поле набора данных.

Подробнее о свойствах, методах, событиях см. в описании **Edit.**

DBImage — отображение графического поля данных

Компонент позволяет отображать графические поля.

Страница библиотеки *DataAccess*

Класс *TDBImage*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWincontrol — TCustomControl*

Модуль *Dbctrls*

Описание

DBImage — связанный с данными аналог обычного компонента Image. Компонент позволяет отображать графические поля, например, фотографии сотрудников.

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Большинство свойств и методов **DBImage** подобны **Image.** Однако основное свойство компонента — **Picture** недоступно во время проектирования, поскольку связано с данными, хранящимися в соответствующем поле базы данных. Во время выполнения оно может изменяться программно или действиями пользователя, например, загрузкой изображения из буфера обмена Clipboard.

Подробнее о свойствах, методах, событиях см. в описании **Image.**

DBMemo — отображение данных типа многострочных текстов

Позволяет отображать и редактировать данные поля, в частности, типа MEMO и BLOB.

Страница библиотеки *DataAccess*

Класс *TDBMemo*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWincontrol — TCustomEdit — TCustomMemo*

Модуль *Dbctrls*

Описание

DBMemo — связанный с данными аналог обычного многострочного окна редактирования **Memo.** Он позволяет отображать и редактировать данные полей разных типов и, прежде всего, типов MEMO и BLOB.

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Большинство остальных свойств и методов аналогичны свойствам и методам компонента **Memo. Но в DBMemo** главное свойство окна — **Lines** недоступно во время проектирования. Текст в окне определяется соответствующим полем текущей записи и может изменяться пользователем или программно во время выполнения. Отредактированное значение в окне помещается в соответствующее поле набора данных.

Свойство **Lines,** доступное во время выполнения, имеет множество свойств и методов типа **TStrings.** которые обычно используются для формирования и редактирования текста.

Подробнее о свойствах, методах, событиях см. в описании **Memo.**

DBRadioGroup — группа радиокнопок, связанная с данными

Представляет группу радиокнопок, связанных с базой данных.

Страница библиотеки *DataAccess*

Класс *TDBRadioGroup*

Иерархия *TObject — TPersistent — TComponent — TControl — TWincontrol — TCustomControl — TCustomGroupBox — TCustomRadioGroup*

Модуль *Dbctrls*

Описание

DBRadioGroup — связанный с данными аналог группы радиокнопок **RadioGroup.** Компонент позволяет отображать и редактировать поля с ограниченным множеством возможных значений.

Количество радиокнопок в группе и надписи около них определяются свойством **Items.** Это свойство во время проектирования удобно заполнять редактором, вызываемым из Инспектора Объектов. Значения полей, соответствующие кнопкам, заносятся в свойство **Values** в той же последовательности, в которой заносились кнопки в Items. Это свойство во время проектирования также удобно заполнять редактором, вызываемым из Инспектора Объектов.

Свойство **Columns** определяет число столбцов, в которых отображаются радиокнопки группы.

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Значение поля в текущей записи можно найти в свойстве **Value.** При переключении пользователем кнопок в поле заносится значение из списка **Values,** соответствующее включенной кнопке. Определить во время выполнения, какая кнопка включена в данный момент, можно по индексу **ItemIndex** (0 — первая кнопка, -1 — ни одна кнопка не включена).

Подробнее о свойствах, методах, событиях см. в описании **RadioGroup.**

DBRichEdit — отображение полей текстовых данных в обогащенном формате

Представляет собой многострочное окно редактирования — аналог **RichEdit,** которое может отображать и редактировать значение поля, содержащее текст в обогащенном формате.

Страница библиотеки *DataAccess*

Класс *TDBRichEdit*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWincontrol — TCustomEdit — TCustomMemo*

Модуль *Dbctrls*

Описание

Компонент **DBRichEdit** дает возможность отображать и редактировать поле базы данных, содержащее текст в обогащенном формате RTF. Основные свойства, позволяющие осуществлять выборочное форматирование фрагментов текста — **SelAttributes** — объект типа **TTextAttributes**, форматирующий выделенный или вновь вводимый текст, и свойство **Paragraph** — объект типа **TParaAttributes**, форматирующий текущий абзац.

DBRichEdit имеет свойство **Text,** в котором содержится текст окна редактирования. Свойство **Lines** типа **TStrings** представляет тот же текст, но разбитый на строки. Свойства и методы **Lines** позволяют редактировать текст поля. Однако в отличие от **RichEdit,** в **DBRichEdit** это свойство недоступно во время проектирования, поскольку его содержание связано с данными, хранящимися в соответствующем поле базы данных.

Свойства компонента, обеспечивающие связь с данными:

- DataSource — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Подробнее о свойствах, методах, событиях см. в описании **RichEdit.**

DBText — метка, связанная с данными

Аналог обычной метки **Label,** связанный с данными.

Страница библиотеки *DataAccess*

Класс *TDBText*

Иерархия *TObject — TPersistent — TComponent — TControl — TGraphicControl — TCustomLable*

Модуль *Dbctrls*

Описание

DBText — аналог обычной метки **Label,** связанный с данными. Он позволяет отображать данные некоторого поля текущей записи набора данных, но не дает возможности их редактировать. Поле может быть различного типа: символьное, числовое, булево. Преобразование значения поля в строку текста, отображаемую в **DBText,** производится автоматически.

Свойства компонента, обеспечивающие связь с данными:

- **DataSource** — источник данных типа **TDataSource**
- **DataField** — имя поля, с которым связан компонент
- **Field** — объект этого поля (только для чтения)

Большинство остальных свойств **(Alignment, AutoSize, Anchors** и др.) аналогичны свойствам компонента **Label. Но в DBText** отсутствует главное свойство метки — **Caption** (текст). Текст целиком и полностью определяется соответствующим полем текущей записи.

Подробнее о свойствах, методах, событиях см. в описании **Label.**

Edit — однострочное окно редактирования

Окно редактирования для ввода пользователем однострочных текстов. Может использоваться для отображения текста.

Страница библиотеки *Standard*

Класс *TEdit*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomEdit*

Модуль *stdctrls*

Примеры изображения

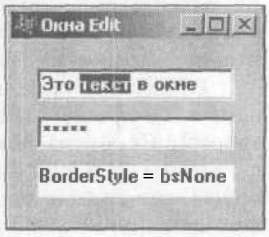


Рис. 2.14
Примеры окон Edit

Описание

В компоненте **Edit** вводимый и выводимый текст содержится в свойстве **Text.** Это свойство можно устанавливать в процессе проектирования или задавать программно. Выравнивание текста невозможно. Перенос строк тоже невозможен. Текст, не помещающийся по длине в окно, просто сдвигается, и пользователь может перемещаться по нему с помощью курсора. Свойство **AutoSize** позволяет автоматически подстраивать высоту (но не ширину) окна под размер текста.

Окно редактирования снабжено многими функциями, свойственными большинству редакторов. Например, в нем предусмотрены типичные комбинации «горячих» клавиш: Ctrl-C — копирование выделенного текста в буфер обмена Clipboard (команда Copy), Ctrl-X — вырезание выделенного текста в буфер Clipboard (команда Cut), Ctrl-V — вставка текста из буфера Clipboard в позицию курсора (команда Paste), Ctrl-Z — отмена последней команды редактирования.

Свойство **AutoSelect** определяет, будет ли автоматически выделяться весь текст при передаче фокуса в окно редактирования. Его имеет смысл задавать равным **true** в случаях, когда при переключении в данное окно пользователь будет скорее заменять текущий текст, чем исправлять его. Имеются также свойства только времени выполнения **SelLength, SelStart, SelText,** определяющие соответственно длину выделенного текста, позицию перед первым символом выделенного текста и сам выделенный текст. В примере, показанном на рис. 2.14, **SelStart = 4, SelLength = 5, SelText** = "текст". Если выделенного текста нет, то свойство **SelStart** просто определяет текущее положение курсора.

Свойство **MaxLength** определяет максимальную длину вводимого текста. Если **MaxLength** = 0, то длина текста не ограничена. В противном случае значение **MaxLength** указывает максимальное число символов, которое может ввести пользователь.

Свойство **Modified,** доступное только во время выполнения, показывает, проводилось ли редактирование текста в окне. Если вы хотите использовать это свойство, то в момент начала работы пользователя с текстом **Modified** надо установить в **false.** Тогда при последующем обращении к этому свойству можно по его значению **(true** или **false)** установить, было или не было произведено редактирование.

Свойство **PasswordChar** позволяет превращать окно редактирования в окно ввода пароля. По умолчанию значение **PasswordChar** равно #0 — нулевому символу. В этом случае это обычное окно редактирования. Но если в свойстве указать иной символ (например, символ звездочки "*"), то при вводе пользователем текста в окне будут появляться именно эти символы, а не те, которые вводит пользователь (см. среднее окно на рис. 2.14). Тем самым обеспечивается секретность ввода пароля.

Свойство **BorderStyle** определяет, ограничена ли клиентская область компонента одинарной бордюрной линией (см. на рис. 2.14 нижнее окно без бордюра).

Основные свойства

Свойство	Объявление / Описание
AutoSelect	**bool** AutoSelect Определяет, будет ли автоматически выделяться весь текст при передаче фокуса в окно редактирования
AutoSize	**bool** AutoSize Указывает, изменяется ли автоматически высота компонента, подстраиваясь под размер текста. По умолчанию false — не подстраивается
BorderStyle	enum **TFormBorderStyle** {bsNone, bsSingle, **bsSizeable,** bsDialog, bsToolWindow, **bsSizeToolWin**}; **typedef** TFormBorderStyle TBorderStyle; TBorderStyle BorderStyle Указывает, ограничена ли клиентская область компонента одинарной бордюрной линией
CanUndo	bool CanUndo Указывает, содержит ли компонент изменения, которые можно отменить. Доступ только для чтения
CharCase	enum TEditCharCase { ecNormal, ecUpperCase, **ecLowerCase** }; TEditCharCase CharCase Указывает, приводится ли принудительно текст к верхнему или нижнему регистрам
Color	**Graphics::TColor** Color Указывает цвет фона компонента
Font	**Graphics::TFont*** Font Определяет атрибуты шрифта
HideSelection	bool HideSelection Определяет, видно ли (при значении false) выделение текста при переходе фокуса к другому компоненту
MaxLength	Classes::TStrings* Lines Указывает максимальное количество символов, которое пользователь может вводить в компонент. При значении 0 длина текста неограничена
Modified	bool Modified Указывает, редактировался ли пользователем текст в компоненте
PasswordChar	char PasswordChar Указывает символ, замещающий фактические символы, вводимые в компонент. При значении '0' символы не заменяются. Используется для ввода паролей
PopupMenu	**Menus::TPopupMenu*** PopupMenu Идентифицирует всплывающее меню, связанное с данным компонентом

Свойство	Объявление / Описание
Readonly	**bool** Readonly Указывает, может ли пользователь изменять тест в компоненте
SelLength	**int** SelLength Определяет количество выделенных символов в строке
SelStart	int SelStart Указывает позицию первого выделенного символа в тексте или, если выделения нет, то позицию курсора
SelText	**AnsiString** SelText Текст, выделенный в окне
Text	**AnsiString** Text Текст в окне редактирования

Основные методы

Метод	Объявление / Описание
Clear	void Clear(void) Удаляет текст из окна
Clear Selection	void ClearSelection(void) Удаляет текст, выделенный в окне
ClearUndo	void ClearUndo(void) Очищает буфер отмены команд редактирования, так что никакие изменения в тексте после этого не могут быть отменены
CopyTo **Clipboard**	void CopyToClipboard(void) Копирует выделенный текст в компоненте редактирования в Clipboard в формате CF_TEXT
CutTo Clipboard	void CutToClipboard(void) Переносит выделенный текст в Clipboard в формате CF_TEXT и уничтожает его в окне
PasteFrom Clipboard	void **PasteFromClipboard(void)** Переносит в окно текст из буфера Clipboard
SelectAll	void S**electAll(void)** Выделяет весь текст в окне редактирования

Основные события

Событие	Описание
OnChange	Наступает, когда текст в окне может быть изменился. Свойство Modified показывает, действительно ли произошло изменение. Свойство Text отображает измененный текст
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу

Событие	Описание
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKeyUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу

FontDialog — диалог выбора шрифта

Невизуальный компонент вызова стандартного диалога Windows для выбора шрифта.

Страница библиотеки *Dialogs*

Класс *TFontDialog*

Иерархия *TObject — TPersistent — TGraphicsObject*

Модуль *dialogs*

Описание

Компонент **FontDialog** вызывает стандартный диалог Windows для выбора шрифта, показанный на рис. 2.15.

Отрытие диалога осуществляется методом **Execute.** Если в диалоге пользователь нажмет кнопку OK, диалог закрывается, метод **Execute** возвращает **true** и выбранные атрибуты шрифта передаются в свойство **Font** компонента-диалога. Если же пользователь отказался от диалога (нажал кнопку Отмена или клавишу Esc), то метод **Execute** возвращает **false.**

Значение свойства **Font** можно задать и перед обращением к диалогу. Тогда оно определит значения атрибутов шрифта по умолчанию, которые увидит пользователь в момент открытия диалога. Таким образом, например, выполнение команды Шрифт, по которой пользователь может задать текущее значение шрифта для компонента **RichEditl,** может иметь вид:

```
// Задание в качестве атрибутов по умолчанию
// атрибутов шрифта текущей позиции курсора в тексте
FontDialog1->Font->Assign(RichEdit1->SelAttributes);
// Открытие диалога
if (FontDialog1->Execute())
   RichEdit1->SelAttributes->Assign(FontDialog1->Font);
```

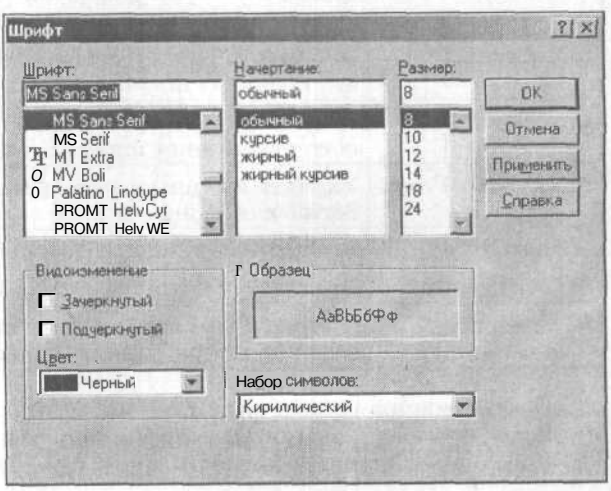


Рис. 2.15
Диалоговое окно выбора шрифта

Свойства **MaxFontSize и MinFontSize** устанавливают ограничения на максимальный и минимальный размеры шрифта. Если значения этих свойств равны О (по умолчанию), то никакие ограничения на размер не накладываются. Если же значения свойств заданы (обычно это целесообразно делать, исходя из размеров компонента приложения, для которого выбирается шрифт), то в списке Размер диалогового окна (см. рис. 2.15) появляются только размеры, укладывающиеся в заданный диапазон. При попытке пользователя задать недопустимый размер ему будет выдано предупреждение вида «Размер должен лежать в интервале ...» и выбор пользователя отменится. Свойства **MaxFontSize и MinFontSize** действуют только при включенной опции **fdLimitSize** (см. ниже).

Свойство **Device** определяет, из какого списка возможных шрифтов будет предложен выбор в диалоговом окне: fdScreen — из списка экрана (по умолчанию), **fdPrinter** — из списка принтера, **fdBoth** — из обоих.

Свойство **Options** содержит множество опций:

fdAnsiOnly	Отображать только множество шрифтов символов Windows, не отображать шрифтов со специальными символами
fdApplyButton	Отображать в диалоге кнопку Применить независимо от того, предусмотрен ли обработчик события OnApply
fdEffects	Отображать в диалоге индикаторы специальных эффектов (подчеркивание и др.) и список Цвет
fdFixedPitchOnly	Отображать только шрифты с постоянной шириной символов
fdForceFontExist	Позволять пользователю выбирать шрифты только из списка, запрещать ему вводить другие имена
fdLimitSize	Разрешить использовать свойства MaxFontSize и MinFontSize, ограничивающие размеры шрифта
fdNoFaceSel	Открывать диалоговое окно без предварительно установленного имени шрифта
fdNoOEMFonts	Удалять из списка шрифтов шрифты OEM
fdScalableOnly	Отображать только масштабируемые шрифты, удалять из списка не масштабируемые (шрифты bitmap)
fdNoSimulations	Отображать только шрифты и их начертания, напрямую поддерживаемые файлами, не отображая шрифты, в которых жирный стиль и курсив синтезируется
fdNoSizeSel	Открывать диалоговое окно без предварительно установленного размера шрифта
fdNoStyleSel	Открывать диалоговое окно без предварительно установленного начертания шрифта
fdNoVectorFonts	Удалять из списка векторные шрифты (типа Roman или Script для Windows 1.0)
fdShowHelp	Отображать в диалоговом окне кнопку Справка
fdTrueTypeOnly	Предлагать в списке только шрифты TrueType
fdWysiwyg	Предлагать в списке только шрифты, доступные и для экрана, и для принтера, удаляя из него аппаратно зависимые шрифты

По умолчанию все эти опции, кроме **fdEffects,** отключены.

Если установить опцию **fdApplyButton,** то при нажатии пользователем кнопки Применить возникает событие **OnApply,** в обработчике которого вы можете напи-

сать код, который применит выбранные пользователем атрибуты, не закрывая диалогового окна. Например:

```
RichEdit1->SelAttributes->Assign(FontDialog1->Font);
```

Тогда пользователь может наблюдать изменения в окне **RichEditl,** нажимая в диалоговом окне кнопку Применить и не прерывая диалога. Это очень удобно, так как позволяет пользователю правильно подобрать атрибуты шрифта.

При работе с окном редактирования Memo аналогичный оператор может иметь вид:

```
Memo1->Font->Assign(FontDialog1->Font);
```

Основные свойства

Свойство	Объявление / Описание
Device	**enum TFontDialogDevice { fdScreen, fdPrinter, fdBoth }; TFontDialogDevice Device** **Определяет, из какого списка возможных шрифтов будет предложен выбор в диалоговом окне: fdScreen — из списка экрана (по умолчанию), fdPrinter — из списка принтера, fdBoth — из обоих**
Font	**Graphics::TFont* Font** **Определяет атрибуты шрифта**
MaxFontSize	**int MaxFontSize** **Устанавливает ограничение на максимальный размер шрифта**
MinFontSize	**int MinFontSize** **Устанавливает ограничение минимальный размер шрифта**
Options	**enum TFontDialogOption {fdAnsiOnly, fdTrueTypeOnly, fdEffects, fdFixedPitchOnly, fdForceFontExist, fdNoFaceSel, fdNoOEMFonts, fdNoSimulations, fdNoSizeSel, fdNoStyleSel, fdNoVectorFonts, fdShowHelp, fdWysiwyg, fdLimitSize, fdScalableOnly, fdApplyButton};** **typedef Set<TFontDialogOption, fdAnsiOnly, fdApplyButton> TFontDialogOptions;** **TFontDialogOptions Options** **Различные опции диалога (см. выше в описании FontDialog)**

Основные методы

Метод	Объявление / Описание
Execute	**bool** Execute(void) Вызывает диалог, возвращает true, если пользователь произвел выбор в диалоге

Остальные методы наследуются от классов — предшественников.

События

Событие	Описание
OnApply	Наступает, когда пользователь в диалоге нажимает кнопку Apply (см. выше в описании FontDialog)
OnClose	Событие наступает при закрытии диалога
OnShow	Событие наступает при открытии диалога

GroupBox — групповая панель

Панель — контейнер с рамкой и надписью, объединяющий группу связанных органов управления.

Страница библиотеки *Standard*

Класс *TGroupBox*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TCustomControl — TCustomGroupBox*

Модуль *stdctrls*

Примеры изображения

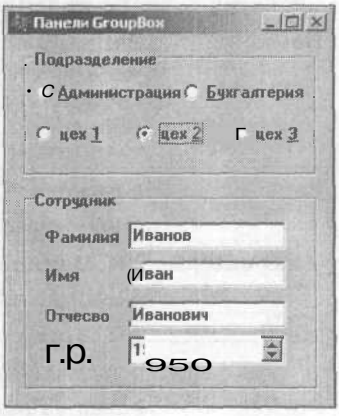


Рис. 2.16
Примеры групповой панели GroupBox, размещающей радиокнопки (вверху) и окна редактирования (внизу)

Описание

Панель **GroupBox** -- это контейнер с рамкой и надписью, объединяющий группу связанных органов управления, таких как радиокнопки **RadioButton,** индикаторы **CheckBox** и т.д. В отличие от других панелей (например, **Panel)** не имеет широких возможностей задания различных стилей оформления. Но **GroupBox** имеет встроенную рамку с надписью, которая обычно используется для выделения на форме группы функционально объединенных компонентов.

Основные свойства

Все свойства панели наследуются от **TWinControl и TComponent.** Основные из них:

Свойство	Объявление / Описание
Align	**enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom};** **TAlign Align** **Определяет способ выравнивания компонента в контейнере (родительском компоненте)**
Anchors	**enum TAnchorKind { akLeft, akTop, akRight, akBottom };** **typedef Set<TAnchorKind, akLeft, akBottom> TAnchors;** **TAnchors Anchors** **Определяет привязку данного компонента к родительскому при изменении размеров последнего**
Brush	**Graphics::TBrush* Brush** **Кисть, определяющая заполнение фона**

Свойство	Объявление / Описание
Caption	**AnsiString** Caption Надпись в углу рамки панели
Font	**Graphics::TFont*** Font Определяет атрибуты шрифта
TabOrder	**typedef** short TTabOrder; **TTabOrder** TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	**bool** TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Никаких специальных методов в панели не объявлено. Методы наследуются от классов-предков **TWinControl и TControl.**

Основные события

Никаких специальных событий в панели не объявлено. События наследуются от классов-предков **TWinControl и TControl.**

Image — контейнер графического изображения

Отображает графическое изображение и обеспечивает работу с ним.

Страница библиотеки *Additional*

Класс *TImage*

Иерархия *TObject — TPersistent — TComponent — TControl — TGraphicControl*

Модуль *extctrls*

Пример изображения

Рис. 2.17
Пример отображения изображения графического файла в компоненте Image

Описание

Компонент **Image** отображает на форме графическое изображение (рис. 2.17). Свойство **Picture** типа **TPicture** содержит отображаемый графический объект типа битовой матрицы, пиктограммы, метафайла или определенного пользователем типа. Свойство **Canvas** позволяет создавать и редактировать изображения.

Во время проектирования загрузить в свойство **Picture** графический файл можно щелкнув на кнопке с многоточием около свойства **Picture** в окне Инспекто-

ра Объектов или просто двойным щелчком на Image. Перед вами откроется окно Picture Editor (см. рис. 2.18), позволяющее загрузить в свойство **Picture** какой-нибудь графический файл (кнопка Load), а также сохранить открытый файл под новым именем или в новом каталоге.

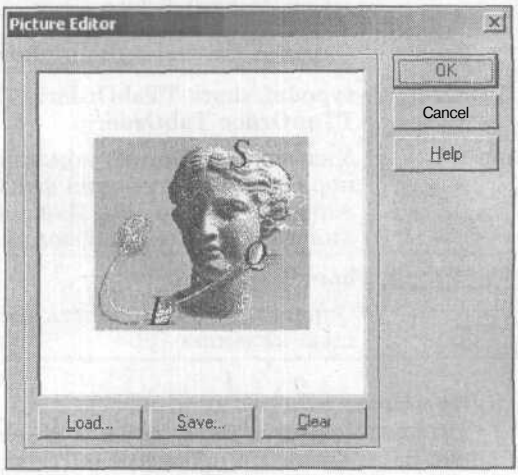


Рис. 2.18
Окно загрузки графического файла в свойство Picture компонента Image

Когда вы в процессе проектирования загрузили изображение из файла в компонент **Image,** он не просто отображает его, но и сохраняет в приложении. Это дает вам возможность поставлять ваше приложение без отдельного графического файла.

Если установить свойство **AutoSize в true,** то размер компонента **Image** будет автоматически подгоняться под размер помещенной в него картинки. Если же свойство **AutoSize** установлено в **false,** то изображение может не поместиться в компонент или, наоборот, площадь компонента может оказаться много больше площади изображения.

Другое свойство — Stretch позволяет подгонять не компонент под размер рисунка, а рисунок под размер компонента. Но поскольку вряд ли реально установить размеры Image точно пропорциональными размеру рисунка, то изображение исказится. Устанавливать **Stretch в true** может иметь смысл только для каких-то узоров, но не для картинок. Свойство **Stretch** не действует на изображения пиктограмм, которые не могут изменять своих размеров.

Свойство — **Center,** установленное в **true,** центрирует изображение на площади **Image,** если размер компонента больше размера рисунка.

Рассмотрим еще одно свойство -- **Transparent** (прозрачность). Если **Transparent** равно **true,** то изображение в **Image** становится прозрачным. Это можно использовать для наложения изображений друг на друга. Учтите, что свойство **Transparent** действует только на битовые матрицы. При этом прозрачным (т.е. заменяемым на цвет расположенного под ним изображения) делается по умолчанию цвет левого нижнего пиксела битовой матрицы.

Свойство **Picture** позволяет легко организовать обмен с графическими файлами любых типов в процессе выполнения приложения. Это свойство является объектом, который имеет в свою очередь подсвойства, указывающие на хранящийся графический объект. Если в **Picture** хранится битовая матрица, на нее указывает свойство **Picture.Bitmap.** Если хранится пиктограмма, на нее указывает свойство **Picture.Icon.** На хранящийся метафайл указывает свойство **Picture.Metafile.** Наконец, на графический объект произвольного типа указывает свойство **Picture. Graphic.**

Объект **Picture** и его свойства **Bitmap, Icon, Metafile и Graphic** имеют методы файлового чтения и записи **LoadFromFile и SaveToFile.** Для свойств **Pictu-**

re.Bitmap, Picture.Icon и Picture.Metafile формат файла должен соответствовать классу объекта: битовой матрице, пиктограмме, метафайлу. При чтении файла в свойство **Picture.Graphic** файл должен иметь формат метафайла. А для самого объекта **Picture** методы чтения и записи автоматически подстраиваются под тип файла.

Например, если вы имеете в приложении компонент-диалог **OpenPictureDialog,** то загрузка в **Image** выбираемого пользователем графического фала может быть организована оператором

```
if (OpenPictureDialog1->Execute())
  Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
```

Загружаемый этим оператором файл может быть любого типа: битовая матрица, пиктограмма или метафайл. Если будут открываться только файлы битовых матриц, то оператор загрузки файла можно заменить на

```
Image1->Picture->Bitmap->LoadFromFile(
                                OpenPictureDialog1->FileName);
```

Для пиктограмм можно было бы использовать оператор

```
Image1->Picture->Icon->LoadFromFile(
                                OpenPictureDialog1->FileName);
```

а для метафайлов — оператор

```
Image1->Picture->Metafile->LoadFromFile(
                                OpenPictureDialog1->FileName);
```

или

```
Image1->Picture->Graphic->LoadFromFile(
                                OpenPictureDialog1->FileName);
```

Но во всех этих случаях, если формат файла не совпадет с предполагаемым, возникнет ошибка.

Аналогично работает и метод **SaveToFile** с тем отличием, что примененный к **Picture или к Picture->Graphic** он сохраняет в файле изображение любого формата. Например, если вы дополните свое приложение диалогом **SavePictureDialog,** то сохранение изображения в файле можно организовать оператором

```
if (SavePictureDialog1->Execute())
  Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
```

В этом случае пользователь получит возможность сохранить изображение любого формата в файле с новым именем. Только при этом, чтобы не возникало в дальнейшем путаницы, расширение сохраняемого файла все-таки должно соответствовать формату сохраняемого изображения.

Абсолютно идентично для изображений любого формата будет работать программа, если оператор сохранения вы замените на

```
Image1->Picture->Graphic->SaveToFile(SavePictureDialog1->FileName);
```

использующий свойство **Picture->Graphic.** А если вам известен формат хранимого в компоненте **Image** изображения, то вы можете применить метод **SaveToFile** к свойствам **Picture->Bitmap, Picture->Icon, и Picture->Metafile.**

Для всех рассмотренных объектов **Picture, Picture->Bitmap, Picture->Icon, и Picture->Metafile** определены методы присваивания значений объектов Assign. Однако для **BitMap, Icon и Metafile** присваивать можно только значения однородных объектов: соответственно битовых матриц, пиктограмм, метафайлов. При попытке присвоить значения разнородных объектов генерируется исключение **EConvertError.** Объект **Picture** — универсальный, ему можно присваивать значения объектов любых из остальных трех классов. А значение **Picture** можно присваивать только тому объекту, тип которого совпадает с типом объекта, хранящегося в нем.

Метод Assign можно использовать и для обмена изображениями с буфером Clipboard. Например, оператор

```
Clipboard()->Assign(Image1->Picture);
```

занесет в буфер обмена изображение, хранящееся в **Image**1. Аналогично оператор

```
Image1->Picture->Assign(Clipboard());
```

прочитает в **Image1** изображение, находящееся в буфере обмена. Причем это может быть любое изображение и даже текст.

Надо только не забыть при работе с буфером обмена вставить в модуль директиву:

```
#include<Clipbrd.hpp>
```

Автоматически C++Builder эту директиву не вставляет.

Благодаря наличию канвы **Canvas** компонент **Image** широко используется не только для хранения готовых изображений, но и для построения различных графических редакторов (это подробно рассмотрено в [1]).

Основные свойства

Свойство	Объявление / Описание
AutoSize	**bool AutoSize** Указывает, изменяется ли автоматически размер компонента, подстраиваясь под размер изображения. По умолчанию **false** — не подстраивается
Canvas	**Graphics::TCanvas* Canvas** Определяет поверхность (холст, канву) для рисования пером Реп и кистью **Brush**, для наложения друг на друга нескольких изображений. Доступ только для чтения. Доступно только, если в свойстве **Picture** хранится битовая матрица
Center	**bool Center** Указывает, должно ли изображение центрироваться в поле компонента, если его размеры меньше размеров поля. При значении **false** изображение располагается в верхнем левом углу поля. Свойство не действует, если **AutoSize** установлено в **true** или если **Stretch** установлено в **true и Picture** содержит не пиктограмму
Incremental Display	**bool IncrementalDisplay** Указывает, должно ли изображение частично рисоваться во время медленных операций с большими изображениями. Вместо такого рисования часто можно использовать индикацию процесса обработкой событий **OnProgress**
Picture	**Graphics::TPicture* Picture** Определяет отображаемый графический объект типа **TPicture.** Может загружаться программно или во время проектирования с помощью Picture Editor
Stretch	**bool Stretch** Указывает, должны ли изменяться размеры изображения, подгоняясь под размеры компонента. Учтите, что изменение размеров изображения приведет к его искажению, если соотношение сторон графического объекта и компонента **Image** не одинаково

Свойство	Объявление / Описание
Transparent	**bool** Transparent Указывает, должен ли быть цвет фона изображения прозрачным, чтобы сквозь него было видно нижележащее изображение

Основные методы

Никаких специальных методов в компоненте не объявлено. Компонет наследует множество методы от базового класса **TControl.**

Основные события

Событие	Описание
OnProgress	События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса

Остальные события наследуются от класса **TControl.**

ImageList — список изображений

Список, использующийся для хранения набора изображений одинаковых размеров, на которые можно ссылаться по индексам. Невизуальный компонент.

Страница библиотеки *Additional*

Класс *Win32*

Иерархия *TObiect — TPersistent — TComponent — TCustomImageList — TDragImageList*

Модуль *controls*

Описание

ImageList используется и как компонент, и как свойство других компонентов, например, свойство **Images** компонента **MainMenu.** Представляет собой набор изображений одинаковых размеров, на которые можно ссылаться по индексам, начинающимся с 0. Используется для эффективного управления множеством пиктограмм и битовых матриц. Может включать в себя монохромные битовые матрицы, содержащие маски для прозрачности рисуемых изображений.

Изображения в компонент ImageList могут быть загружены в процессе проектирования с помощью редактора списков изображений. Окно редактора (рис. 2.19), вызывается двойным щелчком на компоненте **ImageList** или щелчком правой кнопки мыши и выбором команды контекстного меню ImageList Editor.

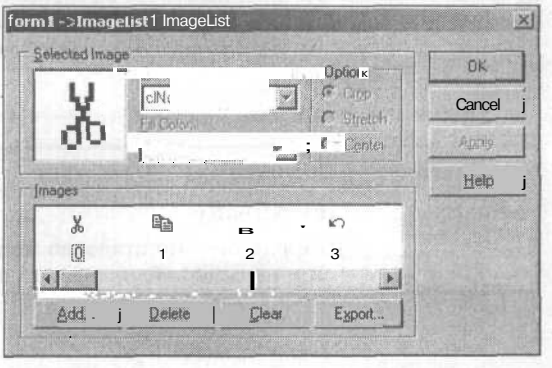


Рис. 2.19
Окно редактора списка изображений

В окне редактора списков изображений вы можете добавить в списки изображения, пользуясь кнопкой Add, удалить изображение из списка кнопкой Delete, очистить весь список кнопкой Clear. При добавлении изображения в список открывается обычное окно открытия файлов изображений, в котором вы можете выбрать интересующий вас файл. Только учтите, что размер всех изображений в списке должен быть одинаковым. Как правило, это размер, используемый для пиктограмм в меню, списках, кнопках. При добавлении в список изображений для кнопок надо иметь в виду, что они часто содержат не одно, а два и более изображений. В этих случаях при попытке добавить изображение задается вопрос: "Bitmap dimensions for ... are greater then imagelist dimensions. Separate into ... separate bitmaps?" (Размерность изображения ... больше размерности списка. Разделить на ... отдельных битовых матрицы?). Если вы ответите отрицательно, то все изображения уменьшатся в горизонтальном размере и лягут как одно изображение. Использовать его в дальнейшем будет невозможно. Поэтому на заданный вопрос надо отвечать положительно. Тогда загружаемая битовая матрица автоматически разделится на отдельные изображения, и потом вы можете удалить те из них, которые вам не нужны, кнопкой Delete.

Как видно из рис. 2.19, каждое загруженное в список изображение получает индекс. Именно на эти индексы впоследствии вы можете ссылаться в соответствующих свойствах разделов меню, списков, кнопок и т.д., когда вам надо загрузить в них то или иное изображение. Изменить последовательность изображений в списке вы можете, просто перетащив изображение мышью на новое место.

В редакторе списков изображений вы можете, выделив то или иное изображение, установить его свойства: Transparent Color и Fill Color. Но это можно делать только в том сеансе работы с редактором списков изображений, в котором загружено данное изображение. Для изображений, загруженных в предыдущих сеансах, изменение этих свойств невозможно.

Свойство Transparent Color определяет цвет, который используется в маске для прозрачного рисования Изображения. По умолчанию это цвет левого нижнего пиксела изображения. Для пиктограмм данное свойство устанавливается в **clNone,** поскольку пиктограммы уже маскированы.

Свойство Fill Color определяет цвет, используемый для заполнения пустого пространства при перемещении и центрировании изображения. Для пиктограмм данное свойство устанавливается в clNone.

Группа радиокнопок Options определяет способ размещения изображения битовой матрицы с размерами, не соответствующими размерам, принятым в списке:

Crop	Отображается часть изображения, помещающаяся в размер списка, начиная с левого верхнего угла
Stretch	Размеры изображения изменяются, становясь равными размерам списка. При этом возможны искажения
Center	Изображение центрируется, а если его размер больше размера списка, то не помещающиеся области отсекаются

Основные свойства

Свойство	Объявление / Описание
AllocBy	**int** AllocBy Определяет количество изображений, на которое увеличивается список для добавления новых изображений

Свойство	Объявление / Описание
BkColor	**Graphics::TColor** BkColor Определяет цвет фона, используемый при маскировании области изображений. Если BkColor равен **clNone**, изображение рисуется прозрачным с использованием маски. В противном случае BkColor определяет цвет вне маски. BkColor не влияет на изображение, если свойство Masked установлено в false
BlendColor	Graphics::TColor BlendColor Определяет цвет фона, используемый при рисовании изображения. Это цвет, который комбинируется с указанным цветом при значениях **DrawingStyle**, равных dsFocus и dsSelected. Значение clNone соответствует отсутствию комбинируемого цвета, а значение clDefauIt означает системный цвет выделения
Count	**int** Count Указывает число изображений в списке. Свойство только для чтения
DrawingStyle	**enum TDrawingStyle {dsFocus,** dsSelected, **dsNormal, dsTransparent};** TDrawingStyle DrawingStyle Указывает стиль, используемый при рисовании
Height	**int** Height Высота изображений в списке. При изменении этого свойства список очищается
ImageType	enum **TImageType {itImage, itMask};** **TImageType** ImageType Определяет, использует ли список изображение (itlmage) или маску (itMask)
Masked	**bool** Masked Определяет, содержит ли список маски, комбинируемые с изображениями
Width	int Width Ширина изображений в списке. При изменении этого свойства список очищается

Основные методы

Все методы **TImageList** наследует от своих предков, прежде всего от **TDragImageList** и **TCustomImageList.**

Метод	Объявление / Описание
Add	int **Add(Graphics::TBitmap*** Image, Graphics::**TBitmap*** Mask) Добавляет в компонент изображение и его маску. Возвращает индекс изображения или -1
AddIcon	int AddIcon(Graphics::TIcon* Image) Добавляет пиктограмму в компонент. Возвращает индекс изображения или −1
AddImages	void **AddImages(TCustomImageList* Value)** Добавляет в компонент изображения из другого списка изображений Value

Метод	Объявление / Описание
AddMasked	int AddMasked(Graphics::TBitmap* Image, Graphics::TColor MaskColor) Добавляет в компонент изображение вместе с цветом, который используется, чтобы сгенерировать прозрачную маску
DragLock	bool DragLock(HWND Window, int XPos, int YPos) Заменяет изображение перетаскиваемого окна изображением из данного списка, рисует его в позиции, определяемой параметрами XPos и YPos
DragMove	bool DragMove(int X, int Y) Заменяет изображение перетаскиваемого окна изображением из данного списка, определяя новую позицию координатами X и Y
DragUnlock	void DragUnlock(void) Устраняет связь между изображением данного списка и перетаскиваемым окном, которая использовалась в методах DragLock и DragMove

Draw	enum TImageType {itImage, itMask};
	void Draw(Graphics::TCanvas* Canvas, int X, int Y, int Index, bool Enabled = true);
	void Draw(Graphics::TCanvas* Canvas, int X, int Y, int Index, TDrawingStyle ADrawingStyle, TImageType AImageType, bool Enabled = true);
	Рисует изображение, указанное индексом Index, в указанной позиции (X, Y) на канве Canvas. При Enabled = false изображение рисуется как недоступное

DrawOverlay	typedef Shortint TOverlay;
	enum TDrawingStyle {dsFocus, dsSelected, dsNormal, dsTransparent};
	enum TImageType {itImage, itMask};
	void DrawOverlay(Graphics::TCanvas* Canvas, int X, int Y, int ImageIndex, TOverlay Overlay, bool Enabled = true);
	void DrawOverlay(Graphics::TCanvas* Canvas, int X, int Y, int ImageIndex, TOverlay Overlay, TDrawingStyle ADrawingStyle, TImageType AImageType, bool Enabled = true);
	Рисует изображение ImageIndex и находящуюся над ним прозрачную оверлейную маску с индексом Overlay на канве Canvas в позиции (X, Y)

FileLoad	enum TResType { rtBitmap, rtCursor, rtIcon }; bool FileLoad(TResType ResType, const AnsiString Name, Graphics::TColor MaskColor) Загружает в список изображение из ресурсов (битовую матрицу, курсор или пиктограмму)
GetBitmap	void GetBitmap(int Index, Graphics::TBitmap* Image) Возвращает по указанному индексу указатель на битовую матрицу, хранящуюся в списке
GetHotSpot	Types::TPoint GetHotSpot() Возвращает позицию рабочей точки (текущую позицию мыши), связанную с перетаскиваемым изображением

Метод	Объявление / Описание

GetIcon	enum TDrawingStyle {dsFocus, dsSelected, dsNormal, dsTransparent};
	enum TImageType {itImage, itMask};
	void GetIcon(int Index, Graphics::TIcon* Image, TDrawingStyle ADrawingStyle, TImageType AImageType);
	void GetIcon(int Index, Graphics::TIcon* Image);
	Возвращает по указанному индексу указатель на пиктограмму, хранящуюся в списке

GetImage Bitmap	HBITMAP GetImageBitmap(void) Возвращает ссылку на битовую матрицу, содержащую все изображения из списка изображений
GetMask Bitmap	HBITMAP GetMaskBitmap(void) Возвращает ссылку на битовую матрицу, содержащую все маски изображений из списка изображений
GetResource	enum TResType { rtBitmap, rtCursor, rtIcon }; enum TLoadResource {lrDefaultColor, lrDefaultSize, lrFromFile, lrMap3DColors, lrTransparent, lrMonoChrome}; typedef Set<TLoadResource, lrDefaultColor, lrMonoChrome> TLoadResources; bool GetResource(TResType ResType, const AnsiString Name, int Width, TLoadResources LoadFlags, Graphics::TColor MaskColor); Загружает битовую матрицу, курсор или пиктограмму из ресурса в список изображений (для загрузки ресурса из пакета нужно использовать GetInstRes)
HideDrag Image	void HideDragImage(void) Скрывает текущее перетаскиваемое изображение, если перед этим оно было видимым
Insert	HIDESBASE fastcall Insert(int Index, Graphics::TBitmap* Image, Graphics::TBitmap* Mask) Вставляет битовую матрицу Image и ее маску Mask в указанную позицию Index списка изображений
InsertIcon	void InsertIcon(int Index, Graphics::TIcon* Image) Вставляет пиктограмму Image в указанную позицию Index списка изображений
InsertMasked	void InsertMasked(int Index, Graphics::TBitmap* Image, Graphics::TColor MaskColor) Вставляет битовую матрицу Image в указанную позицию Index списка изображений, создавая маску на основе цвета MaskColor
Overlay	typedef Shortint TOverlay; bool Overlay(int ImageIndex, TOverlay Overlay) Включает индекс изображения в список изображений для использования в качестве оверлейной маски
Register Changes	void RegisterChanges(TChangeLink* Value) Регистрирует объект для последующих извещений об изменениях в списке изображений

Метод	Объявление / Описание
Replace	void Replace(int Index, Graphics::TBitmap* Image, Graphics::TBitmap* Mask) Заменяет изображение в списке с индексом Index новым изображением Image и маской Mask
ReplaceIcon	void ReplaceIcon(int Index, Graphics::TIcon* Image) Заменяет изображение в списке с индексом Index новой пиктограммой Image
Replace Masked	void ReplaceMasked(int Index, Graphics::TBitmap* NewImage, Graphics::TColor MaskColor) Заменяет изображение в списке с индексом Index новым изображением Newlmage и маской цвета MaskColor
ResInstLoad	enum TResType { rtBitmap, rtCursor, rtIcon } bool ResInstLoad(int Instance, TResType ResType, const AnsiString Name, Graphics::TColor MaskColor) Загружает ресурс из пакета в список изображений
ResourceLoad	bool ResourceLoad(TResType ResType, const AnsiString Name, Graphics::TColor MaskColor) Загружает ресурс в список изображений (для загрузки ресурса из пакета нужно использовать RestInstLoad)
SetDrag Image	bool SetDragImage(int Index, int HotSpotX, int HotSpotY) Устанавливает изображение Index в списке изображений, которое должно быть показано во время операции перетаскивания Drag&Drop. HotSpotX и HotSpotY — точка привязки
ShowDrag Image	void ShowDragImage(void) Делает видимым текущее перетаскиваемое изображение, если до этого оно было невидимо
UnRegister Changes	void UnRegisterChanges(TChangeLink* Value) Удаляет объект из списка зарегистрированных объектов изображений

События
В ImageList определено только одно событие:

Событие	Описание
OnChange	Происходит при изменении списка

Label — метка

Метка, используемая для отображения текста.

Страница библиотеки *Standard*

Класс *TLabel*

Иерархия *TObject — TPersistent — TComponent — TControl — TGraphicControl — TCustomLabel*

Модуль *stdctrls*

Примеры изображения

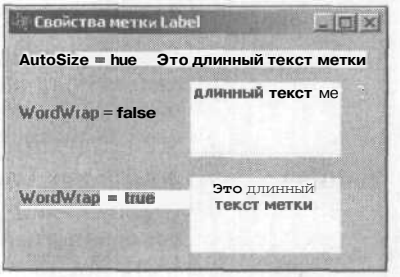


Рис. 2.20
Примеры меток. В левых метках Color = clBtnFace, в правых — clWhite. Левые метки поясняют свойства, установленные в правых метках

Описание

Компонент **Label** используется для отображения текста, который играет роль метки и не изменяется пользователем. Текст метки задается свойством **Caption.** Шрифт надписи определяется свойством **Font,** цвет фона - - свойством **Color,** а цвет надписи — подсвойством **Color** свойства **Font.** Размер меток **Label** определяется свойством **AutoSize.** Если это свойство установлено в **true,** то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же **AutoSize** равно **false,** то выравнивание текста внутри компонента определяется свойством **Alignment,** которое позволяет выравнивать текст по левому краю, правому краю или центру клиентской области метки.

Свойство **Wordwrap** определяет допустимость переноса слов длинной надписи, превышающей длину компонента, на новую строчку. Чтобы такой перенос мог осуществляться, надо установить свойство **Word Wrap в true,** свойство **AutoSize в false** (чтобы размер компонента не определялся размером надписи) и сделать высоту компонента такой, чтобы в нем могло поместиться несколько строк. Если **Wordwrap** не установлено в **true при AutoSize** равном **false,** то длинный текст, не помещающийся в рамке метки, просто обрезается.

Влияние свойств **AutoSize и Wordwrap** вы можете видеть на рис. 2.20. На нем представлены три метки с одинаковой надписью "Это длинный текст метки". Цвет фона **Color** выбран белым, чтобы был виден размер меток. В верхней метке **AutoSize = true** и размер метки определяется размером надписи. В нижних метках одинакового размера **AutoSize = false,** а выравнивание Alignment = **taCenter** (по центру). Но в средней метке **Wordwrap = false** и видна только середина надписи. А в нижней метке **WordWrap = true** и текст переносится.

Метки могут обеспечить клавишами ускоренного доступа элементы, в которых такие клавиши не предусмотрены, например, окна редактирования. Компонент, на который должен переключаться фокус при нажатии клавиш ускоренного доступа, задается свойством **FocusControl.** Чтобы доступ осуществлялся, надо установить в **true** свойство **ShowAccelChar.** В надписи **Caption** перед соответствующим символом надо поставить символ амперсанда - "&". Следующий за амперсандом символ будет отображаться в надписи подчеркнутым и будет являться символом быстрого доступа: при выполнении приложения нажатие клавиши Alt + клавиши данного символа будет эквивалентно переключению фокуса на компонент, указанный свойством **FocusControl.**

Основные свойства

Свойство	Объявление / Описание
Alignment	**enum TAlignment { taLeftJustify, taRightJustify, taCenter }; Classes::TAlignment Alignment** **Управляет горизонтальным выравниванием текста в пределах метки, если свойство AutoSize установлено в false: taLeftJustify — влево, taRightJustify — вправо, taCenter — по центру**

Свойство	Объявление / Описание
AutoSize	**bool** AutoSize Если это свойство установлено в **true**, то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же AutoSize равно false, то выравнивание текста внутри компонента определяется свойством Alignment
Caption	**AnsiString** Caption Строка текста, отображаемая меткой. Может содержать символ ускоренного доступа к элементу, указанному свойством FocusControl
Color	Graphics::TColor Color Определяет цвет фона метки
FocusControl	Controls: **:TWinControl*** FocusControl Определяет оконный компонент, получающий фокус при нажатии пользователем клавиши быстрого доступа метки (см. пояснения выше в описании Label)
Font	Graphics: **:TFont*** Font Определяет атрибуты шрифта
Layout	enum TTextLayout { **tlTop**, tlCenter, **tlBottom** }; TTextLayout Layout Определяет выравнивание текста в поле метки по вертикали
ParentColor	bool ParentColor Определяет (при значении true), что для фона метки будет заимствован цвет родительского компонента. В этом случае фон метки не заметен и видна только ее надпись — Caption
ShowAccel Char	bool ShowAccelChar Определяет, как амперсанд отображается в тексте метки (см. пояснения выше в описании Label)
Wordwrap	bool Wordwrap Указывает, переносится ли текст на новую строку, если он превышает ширину метки, а высота метки позволяет разместить в ней несколько строк

Основные методы

Никаких специальных методов в компоненте не объявлено. Метка наследует множество методов от своих предшественников, в основном, от базового класса **TControl.**

Основные события

В C++Builder 6 в метке определены новые события:

Событие	Описание
OnMouseEnter	Наступает в начале прохождения курсора мыши над меткой
OnMouseLeave	Наступает в конце прохождения курсора мыши над меткой

Кроме того, метка наследует множество событий от класса **TControl.**

LabeledEdit — окно редактирования с меткой

Окно редактирования для ввода пользователем однострочных текстов с привязанной к окну меткой.

Страница библиотеки *Additional*

Класс *TLabeledEdit*

Иерархия *TObiect — TPersistent— TComponent— TControl— TWinControl— TCustomEdit— TCustomLabeledEdit*

Модуль *ExtCtrls*

Примеры изображения

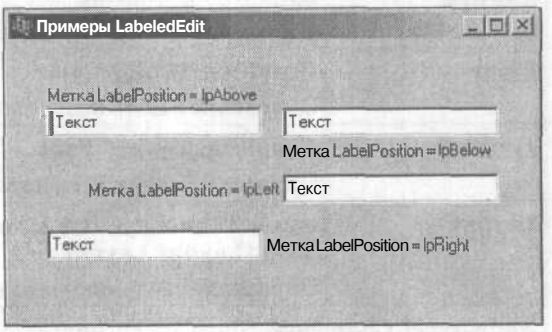


Рис. 2.21
Примеры LabeledEdit. В текстах меток указаны значения свойства LabelPosition компонентов

Описание

Компонент **LabeledEdit** является почти полным аналогом окна редактирования **Edit** с единственным отличием — в нем имеется привязанная к окну метка. Наличие метки — свойства **EditLabel** класса **TBoundLabel,** имеющего свои подсвойства, методы, события, создает ряд удобств в использовании компонента. Во-первых, практически любое окно редактирования в приложении все равно имеет связанную с ним метку, поясняющую назначение окна. Но в данном случае не приходится заботиться о ее размещении: с помощью свойства **LabelPosition** метку можно разместить сверху, слева, справа или внизу окна (см. рис. 2.21) и она будет привязана к нему при любых перемещениях. Во-вторых, выделение символом амперсанда "&" какого-то символа надписи в свойстве **EditLabel->Caption** позволяет задать для окна клавиши быстрого доступа — при нажатии пользователем клавиши Alt и клавиши выделенного символа фокус автоматически будет передаваться данному окну. В-третьих, облегчается управление доступностью окна. Например, оператор

```
LabeledEdit1->Enabled = ! LabeledEdit1->Enabled;
```

изменяет доступность окна. Причем, если окно сделано недоступным, то недоступной выглядит и связанная с ним метка, извещая пользователя, что данным окном пользоваться нельзя.

Большинство свойств, методов и событий **LabeledEdit,** кроме связанных со свойством **EditLabel,** идентично компоненту **Edit.** Вводимый и выводимый текст содержится в свойстве **Text.** Свойство **AutoSize** позволяет автоматически подстраивать высоту (но не ширину) окна под размер текста. Свойство **AutoSelect** определяет, будет ли автоматически выделяться весь текст при передаче фокуса в окно редактирования. Имеются также свойства только времени выполнения **SelLength, SelStart, SelText,** определяющие соответственно длину выделенного текста, позицию перед первым символом выделенного текста и сам выделенный текст. Свойство **MaxLength** определяет максимальную длину вводимого текста. Свойство **Modified,** доступное только во время выполнения, показывает, проводилось ли редактирование текста в окне. Свойство **PasswordChar** позволяет превра-

щать окно редактирования в окно ввода пароля. См. пояснения всех этих свойств в описании компонента **Edit**.

Из свойств объекта, содержащегося в свойстве **EditLabel** и характеризующего метку, надо отметить следующие:

Свойство	Объявление / Описание
Canvas	Graphics::TCanvas* Canvas Определяет поверхность (холст, канву) для рисования пером Реп и кистью Brush,для наложения друг на друга нескольких изображений. Доступ только для чтения
Caption	AnsiString Caption Строка текста, отображаемая меткой
Color	**Graphics::TColor** Color Определяет цвет фона метки
Font	**Graphics::TFont* Font** Определяет атрибуты шрифта
Layout	enum TTextLayout { **tlTop, tlCenter,** tlBottom }; TTextLayout Layout Определяет выравнивание текста в поле метки по вертикали
ShowAccel Char	**bool** Show Accel Char Определяет, как амперсанд отображается в тексте метки
Wordwrap	bool WordWrap Указывает, переносится ли текст на новую строку, если он превышает ширину метки, а высота метки позволяет разместить в ней несколько строк

Свойства компонента LabeledEdit, связанные с меткой

Свойство	**Объявление / Описание**
EditLabel	**property EditLabel: TBoundLabel;** **Объект класса TBoundLabel привязанной к окну метки (см. выше описание свойств этого класса)**
LabelPosition	**type TLabelPosition = (lpAbove, lpBelow, lpLeft, IpRight);** **property LabelPosition: TLabelPosition;** **Определяет положение метки относительно окна: IpAbove — сверху, IpBelow снизу, IpLeft — слева, lpRight** -- справа (см. **рисунок)**
LabelSpacing	**property LabelSpacing: Integer;** **Расстояние в пикселах между меткой и окном**

Все остальные свойства см. в описании компонента **Edit.**

Методы

Все методы см. в описании компонента **Edit.**

События

Все события см. в описании компонента **Edit.**

ListBox — список строк

Отображает список строк и позволяет пользователю выбрать из него необходимые строки.

Страница библиотеки *Standard*

Класс *TListBox*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomListBox*

Модуль *buttons*

Примеры изображения

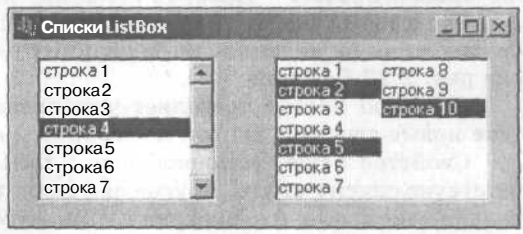


Рис. 2.22
Примеры списков ListBox

Описание

Компонент **ListBox** отображает список строк и позволяет пользователю выбрать из него необходимые строки. В список автоматически добавляются полосы прокрутки, если все строки не помещаются в окне компонента.

Отличие **ListBox** от схожего по функциям компонента **ComboBox** заключается в следующем:

- ComboBox разрешает пользователю редактировать список, а **ListBox** не разрешает
- в ComboBox список может быть развернут или свернут, а в **ListBox** он всегда развернут
- **ListBox** может допускать множественный выбор, а в **ComboBox** пользователь всегда должен выбрать только один элемент

Основное свойство компонента, содержащее список строк, — **Items,** имеющее тип **TStrings**. Заполнить его во время проектирования можно, нажав кнопку с многоточием около этого свойства в окне Инспектора Объектов. Во время выполнения работать с этим свойством можно, пользуясь свойствами и методами класса **TStrings — Clear, Add** и другими.

В компоненте **ListBox** имеется свойство **MultiSelect,** разрешающее пользователю множественный выбор в списке. Если **MultiSelect = false** (значение по умолчанию), то пользователь может выбрать только один элемент списка. В этом случае можно узнать индекс выбранной строки из свойства **ItemIndex,** доступного только во время выполнения. Если ни одна строка не выбрана, то **ItemIndex** = −1. Начальное значение **ItemIndex** невозможно задать во время проектирования. По умолчанию **ItemIndex** = −1. Это означает, что ни один элемент списка не выбран. Если вы хотите задать этому свойству какое-то другое значение, т.е. установить выбор по умолчанию, который будет показан в момент начала работы приложения, то сделать это можно, например, в обработчике события **OnCreate** формы, введя в него оператор вида

```
ListBox1->ItemIndex = 0;
```

Если допускается множественный выбор **(MultiSelect = true** — на рис. 2.22 множественный выбор задан в правом списке), то значение **ItemIndex** соответствует тому элементу списка, который находится в фокусе. При множественном выбо-

ре проверить, выбран ли данный элемент, можно проверив свойство **Selected[Index: Integer]** типа **Boolean.** Например, следующий код для каждой из выбранных строк выдает сообщение вида «Выбрана строка ...»:

```
for (int i=0; i < ListBox1->Items->Count; i++)
  if( ListBox2->Selected[i])
    ShowMessage("Выбрана строка "+ListBox1->Items->Strings[i]);
```

На способ множественного выбора при **MultiSelect = true** влияет свойство **ExtendedSelect.** Если **ExtendedSelect = true,** то пользователь может выделить интервал элементов, выделив один из них, затем нажав клавишу Shift и переведя курсор к другому элементу. Выделить не прилегающие друг к другу элементы пользователь может, если будет удерживать во время выбора нажатой клавишу Ctrl. Если же **ExtendedSelect = false,** то клавиши Shift и Ctrl при выборе не работают.

Свойство **Columns** определяет число столбцов, в которых будет отображаться список, если он не помещается целиком в окне компонента **ListBox** (в правом списке рис. 2.22 **Columns = 2).**

Свойство **Sorted** позволяет упорядочить список по алфавиту. При **Sorted = true** новые строки в список добавляются не в конец, а по алфавиту.

Свойство **Style,** установленное в **lbStandard** (значение по умолчанию), соответствует списку строк. Другие значения **Style** позволяют отображать в списке не только текст, но и изображения. При этом значение **lbOwnerDrawFixed** означает, что высота всех элементов списка одинакова, а значение **lbOwnerDrawVariable** означает, что высота элементов может быть различной.

При значении Style, отличном от **IbStandard,** в момент, когда должна рисоваться какая-то строка списка, наступает событие **OnDrawItem.** Заголовок обработчика этого события имеет вид:

```
void TForm1::ListBox1DrawItem(TWinControl *Control,
                              int Index, TRect &Rect, TOwnerDrawState State)
```

Параметр **Control** является указателем на список, в котором происходит событие. Параметр **Index** указывает индекс элемента, который должен быть перерисован. Параметр **Rect** типа **TRect** (см. **в** гл. 6) указывает область канвы списка, соответствующую рисуемому элементу списка. Параметр **State** типа **TOwnerDrawState** является множеством, элементами которого могут быть значения **odSelected** — строка выделена, **odFocused** строка находится в фокусе и ряд других.

В обработчике события **OnDrawItem** надо методами работы на канве (см. в гл. 1 описание типа **TCanvas)** нарисовать изображение элемента.

При значении **Style** равном **IbOwnerDrawFixed** перед прорисовкой наступает только событие **OnDrawItem. При Style = IbOwnerDrawVariable** перед этим событием наступает другое — **OnMeasureItem,** в котором надо указать высоту элемента. Заголовок обработчика этого события имеет вид:

```
void TForm1::ListBox1MeasureItem(TWinControl *Control,
                                 int Index, int &Height)
```

Параметры **Control и Index** имеют тот же смысл, что и в обработчике **OnDrawItem,** а значение параметра Height надо задать равным высоте данного элемента списка.

Примеры отображения в списках изображений рассмотрены в [1] и [3].

В C++Builder 6 введены значения **Style,** равные **lbVirtual и lbVirtualOwnerDraw.** Они соответствуют виртуальным спискам, формируемым программно во время выполнения. Для этого используются события компонента **OnData, OnDataFind, OnDataObject.** Чтобы эти события наступили, надо задать свойство списка **Count** — число строк. Это свойство всегда присутствует в списке **ListBox,** но только в виртуальных списках допускается задание его значения. В остальных случаях это свойство только для чтения.

Событие **OnData** наступает в тот момент, когда приложению надо отобразить очередную строку списка. Заголовок обработчика этого события имеет вид:

```
void TForm1::ListBox1Data(TWinControl *Control,
                          int Index, AnsiString &Data)
```

Параметр **Control** — список, в котором происходит событие. Параметр **Index** — это индекс строки, которая должна отображаться. А в параметр **Data** надо задать отображаемый текст.

Если со строками виртуального списка надо связать какие-то объекты, это делается в обработчике события **OnDataObject,** заголовок которого имеет вид:

```
void TForm1::ListBox1DataObject(TWinControl *Control,
                                int Index, TObject *&DataObject)
```

Параметр **Index** указывает индекс строки, а в параметр **DataObject** заносится связываемый со строкой объект.

Для возможности управления виртуальным списком, например, для поиска строки по первым символам или для упорядочивания строк, надо написать обработчик события **OnDataFind.** Его заголовок имеет вид:

```
int TForm1::ListBox1DataFind(TWinControl *Control,
                             AnsiString FindString)
```

Параметр **FindString** — это искомая строка, индекс которой надо вернуть как **Result** — результат, возвращаемый функцией.

Примеры создания виртуальных списков рассмотрены в [1] и [3].

Имеется еще один компонент, очень похожий на **ListBox** — это список с индикаторами **CheckListBox.** Выглядит он так же, как **ListBox,** но около каждой строки имеется индикатор, который пользователь может переключать.

Основные свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данным компонентом
Align	enum **TAlign** {alNone, **alTop,** alBottom, alLeft, alRight, **alClient, alCustom};** **TAlign** Align Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Anchors	enum **TAnchorKind** { akLeft, akTop, akRight, akBottom }; **typedef** Set<T Anchor Kind, akLeft, akBottom> TAnchors; TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
Columns	int Columns Определяет число столбцов, в которых будет отображаться список, если он не помещается целиком в окне компонента
Count	int Count Определяет число строк в списке. Может (и должно) задаваться только при значениях Style, равных **lb Virtual** или **lbVirtualOwnerDraw**
ExtendedSelect	**bool** ExtendedSelect Определяет, может ли пользователь при MultiSelect = true выбрать несколько последовательно расположенных элементов, держа нажатой клавишу Shift

Свойство	Объявление / Описание
ItemIndex	int ItemIndex Указывает порядковый номер элемента, выделенного в списке
Items	Classes::TStrings* Items Массив строк списка — объект класса TStrings. Свойства этого класса позволяют формировать и изменять список
MultiSelect	**bool** MultiSelect Указывает, можно ли выбрать в окне списка несколько элементов одновременно
SelCount	int SelCount Указывает количество выделенных элементов при MultiSelect = **true**. Доступ только для чтения
Selected	**bool Selected[int** Index] Индексированный массив, определяющий, какие элементы списка выделены
Sorted	**bool** Sorted Указывает, должны ли строки в списке автоматически сортироваться в алфавитном порядке
Style	enum TListBoxStyle **{lbStandard, lbOwnerDrawFixed, lbOwnerDrawVariable, lbVirtual, lbVirtualOwnerDraw};** TListBoxStyle Style Определяет, будет ли окно списка стандартным, отображающим только текст, или будет позволять отображение также графических образов, а также определяет виртуальные списки (см. выше в описании компонента)
TopIndex	int TopIndex Указывает индекс элемента, видимого вверху списка. Изменение этого индекса соответственно сдвигает видимую часть списка

Основные методы

Метод	Объявление / Описание
Clear	void **Clear(void)** Удаляет все элементы списка
ItemAtPos	int ItemAtPos(TPoint &Pos, bool Existing) Возвращает индекс элемента списка, соответствующего указанным координатам Pos. Если позиция Pos расположена после последнего элемента, то при Existing = **true** ItemAtPos возвращает -1, а при Existing = false — последний элемент списка
ItemRect	**Types::TRect** ItemRect(int Index) Возвращает прямоугольник, описывающий указанный элемент Item списка
SetFocus	**Types::TRect ItemRect(int** Index) Передает фокус элементу, активизирует его

Основные события

Событие	Описание
OnClick	Наступает при щелчке на элементе списка
OnData	Наступает в виртуальных списках, когда приложению надо отобразить очередную строку списка
OnDataFind	Обработчик события пишется для возможности управления виртуальным списком, например, для поиска строки по первым символам или для упорядочивания строк
OnDataObject	Наступает в виртуальных списках, когда со строками виртуального списка надо связать какие-то объекты
OnDrawItem	Наступает при необходимости перерисовать элемент списка (см. выше описание ListBox)
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKeyUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу
OnMeasure Item	Наступает при необходимости перерисовать элемент в списке с изменяемой высотой элементов

MainMenu — главное меню

Невизуальный компонент, позволяет конструировать и создавать на форме полосу главного меню, а также сопутствующие выпадающие меню.

Страница библиотеки *Standard*

Класс *TMainMenu*

Иерархия *TObject — TPersistent — TComponent — TMenu*

Модуль *menus*

Описание

Компонент **MainMenu** отображает на форме главное меню. Обычно на форму помещается один компонент **MainMenu.** В этом случае его имя автоматически заносится в свойство формы **Menu.** Но можно поместить на форму и несколько компонентов **MainMenu** с разными наборами разделов, соответствующими различным режимам работы приложения. В этом случае во время проектирования свойству **Menu** формы присваивается ссылка на один из этих компонентов. А в процессе выполнения в нужные моменты это свойство можно изменять, меняя соответственно состав главного меню приложения.

Основное свойство компонента — **Items.** Его заполнение производится с помощью конструктора меню, вызываемого двойным щелчком на компоненте **MainMenu** или нажатием кнопки с многоточием рядом со свойством **Items** в окне Инспектора Объектов. Окно конструктра показано на рис. 2.23 а. На рис. 2.23 б показано сконструированное меню в работе.

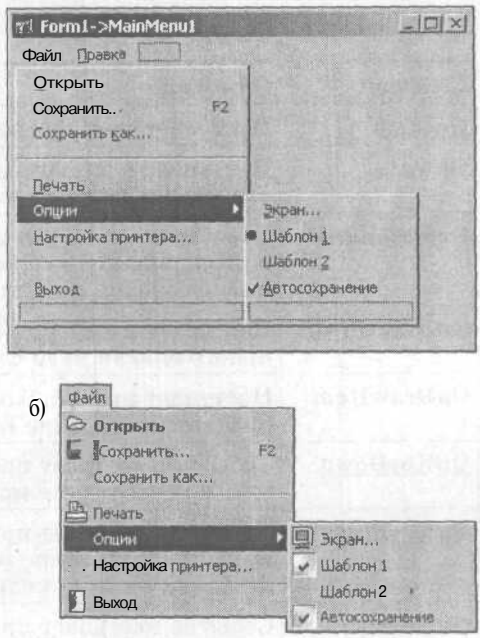


Рис. 2.23
Окно конструктора меню (а)
и сконструированное меню в работе (б)

При работе в конструкторе меню новые разделы можно вводить, помещая курсор в рамку из точек, обозначающую место расположения нового раздела (см. рис. 2.23 а). Если при этом раздел ввелся не на нужном вам месте, вы можете отбуксировать его мышью туда, куда вам надо. Другой путь ввода нового раздела – использование контекстного меню, всплывающего при щелчке правой кнопкой мыши. Если вы предварительно выделите какой-то раздел меню и выберете из контекстного меню команду Insert, то рамка нового раздела вставится перед ранее выделенным. Из контекстного меню вы можете также выполнить команду Create Submenu, позволяющую ввести подменю в выделенный раздел (см. на рис. 2.23 подменю раздела Опции).

При выборе нового раздела вы увидите в Инспекторе Объектов множество свойств данного раздела. Дело в том, что каждый раздел меню, т.е. каждый элемент свойства **Items,** является объектом типа **TMenuItem,** обладающим своими свойствами, методами, событиями.

Свойство **Caption** обозначает надпись раздела. Заполнение этого свойства подчиняется тем же правилам, что и заполнение аналогичного свойства в кнопках, включая использование символа амперсанда для обозначения клавиш быстрого доступа. Если вы в качестве значения **Caption** очередного раздела введете символ минус "-", то вместо раздела в меню появится разделитель (см. на рис. 2.23 разделители после разделов Сохранить как, Настройка принтера и Опции).

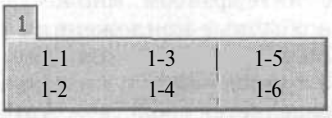
Свойство **Name** задает имя объекта, соответствующего разделу меню.

Свойство **Shortcut** определяет клавиши быстрого доступа к разделу меню – «горячие» клавиши, с помощью которых пользователь, даже не заходя в меню, может в любой момент вызвать выполнение процедуры, связанной с данным разделом. Чтобы определить клавиши быстрого доступа, надо открыть выпадающий список свойства **Shortcut** в окне Инспектора Объектов и выбрать из него нужную комбинацию клавиш. Эта комбинация появится в строке раздела меню (см. на рис. 2.23 команду Сохранить).

Свойство **Default** определяет, является ли данный раздел разделом по умолчанию своего подменю, т.е. разделом, выполняемым при двойном щелчке пользователя на родительском разделе. Подменю может содержать только один раздел по умолчанию, выделяемый жирным шрифтом (см. на рис. 2.23 раздел Открыть).

Свойство **Break** используется в длинных меню, чтобы разбить список разделов на несколько столбцов. Возможные значение **Break: mbNone** — отсутствие разбиения меню (это значение принято по умолчанию), **mbBarBreak и mbBreak** — в меню вводится новый столбец разделов, отделенный от предыдущего полосой **(mbBarBreak)** или пробелами **(mbBreak).** На приведенном ниже рис. 2.24 показан пример, в котором в разделе 1-3 установлено значение **Break = mbBreak,** а в разделе 1-5 — **Break = mbBarBreak.**

Рис. 2.24
Пример меню с разбиением на столбцы



Свойство **Checked,** установленное в **true,** указывает, что в разделе меню будет отображаться маркер флажка, показывающий, что данный раздел выбран — см. на рис. 2.23 раздел «Автосохранение». В C++Builder 6 в разделах меню имеется свойство **AutoCheck.** Если установить его в **true,** то при каждом щелчке пользователя на этом разделе (перед каждым событием **OnClick)** значение маркера переключается на противоположное. В версиях младше C++Builder 6 сам по себе этот маркер не изменяется и в обработчик события **OnClick** такого раздела надо вставлять оператор типа

```
MAutoSave->Checked = ! MAutoSave->Checked;
```

(в приведенном операторе подразумевается, что раздел меню назван MAutoSave).

Еще одним свойством, позволяющим вводить маркеры в разделы меню, является **RadioItem.** Это свойство, установленное в **true,** определяет, что данный раздел должен работать в режиме радиокнопки совместно с другими разделами, имеющими то же значение свойства **GroupIndex.** По умолчанию значение **GroupIndex** равно 0. Но можно задать его большим нуля и тогда, если имеется несколько разделов с одинаковым значением **GroupIndex и с RadioItem = true,** то в них могут появляться маркеры флажков, причем только в одном из них. На рис. 2.23 свойство **RadioItem** установлено в **true** в разделах Шаблон 1 и Шаблон 2, имеющих одинаковое значение **GroupIndex.** Если вы зададите программно в одном из этих разделов **Checked = true,** то в остальных разделах **Checked** автоматически сбросится в **false.** Впрочем, установка **Checked = true** лежит на программе; эта установка может выполняться аналогично приведенному выше оператору.

Описанные маркеры флажков в режиме радиокнопок и в обычном режиме используются для разделов меню, представляющих собой различные опции, взаимоисключающие или совместимые.

Для каждого раздела могут быть установлены во время проектирования или программно во время выполнения свойства **Enabled** (доступен) и **Visible** (видимый). Если установить **Enabled = false,** то раздел будет изображаться серой надписью и не будет реагировать на щелчок пользователя. Если же задать **Visible = false,** то раздел вообще не будет виден, а остальные разделы сомкнутся, заняв место невидимого. Свойства **Enabled и Visible** используются для того, чтобы изменять состав доступных пользователю разделов в зависимости от режима работы приложения.

В C++Builder предусмотрена возможность ввода в разделы меню изображений. За это ответственны свойства разделов **Bitmap и ImageIndex.** Первое из них позволяет непосредственно ввести изображение в раздел, выбрав его из указанного файла. Второе позволяет указать индекс изображения, хранящегося во внешнем компоненте **ImageList.** Указание на этот компонент вы можете задать в свойстве Images компонента **MainMenu.**

Основное событие раздела меню — **OnClick,** возникающее при щелчке пользователя на разделе или при нажатии «горячих» клавиш и клавиш быстрого доступа.

Рассмотрим теперь вопросы объединения главных меню вторичных форм с меню главной формы. Речь идет о приложениях с несколькими формами, в которых и главная, и вспомогательные формы имеют свои главные меню — компоненты **MainMenu.** Конечно, пользователю неудобно работать одновременно с несколькими окнами, каждое из которых имеет свое меню. Обычно надо, чтобы эти меню сливались в одно меню главной формы.

Приложения с несколькими формами могут быть двух видов: приложения с интерфейсом множества документов - - так называемые приложения MDI, и обычные приложения с главной и вспомогательными формами. В приложениях MDI меню дочерних форм всегда объединяются с меню родительской формы. А в приложениях с несколькими формами наличие или отсутствие объединения определяется свойством **AutoMerge** компонентов **TMainMenu.** Если требуется, чтобы меню вторичных форм объединялись с меню главной формы, то в каждой такой вторичной форме надо установить **AutoMerge в true. При** этом свойство **AutoMerge** главной формы должно оставаться в **false.**

Способ объединения меню определяется свойством разделов **GroupIndex.** По умолчанию все разделы меню имеют одинаковое значение **GroupIndex,** равное нулю. Если требуется объединение меню, то разделам надо задать неубывающие номера свойств **GroupIndex.** Тогда, если разделы встраиваемого меню имеют те же значения **GroupIndex,** что и какие-то разделы меню основной формы, то эти разделы заменяют соответствующие разделы основного меню. В противном случае разделы вспомогательного меню встраиваются между элементами основного меню в соответствии с номерами **GroupIndex.** Если встраиваемый раздел имеет **GroupIndex** меньший, чем любой из разделов основного меню, то разделы встраиваются в начало.

Теперь остановимся на одном из вопросов, связанных с меню в упоминавшихся выше приложениях MDI. В них пользователь может открывать сколько ему требуется окон документов. Обычно в подобных приложениях имеется меню Окно (см. рис. 2.25), которое содержит такие разделы, как Новое, Упорядочить и т.п. Последним идет обычно список открытых окон документов, в который заносятся названия открытых пользователем окон. Выбирая в этом списке, пользователь может переключаться между окнами документов.

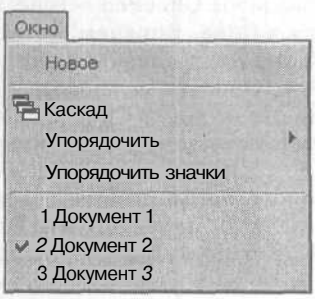


Рис. 2.25
Меню «Окно» в приложении MDI со списком открытых документов

Для включения в меню раздела списка открытых окон, надо в свойстве **WindowMenu** главной формы приложения MDI указать имя меню, в конец которого должен помещаться список. Указывается именно имя меню, а не разделов выпадающего списка. Для приведенного примера должно быть указано имя элемента меню, соответствующего команде Окно.

Одним из безусловных требований, предъявляемых к меню приложений для Windows, является стандартизация меню и их разделов. Этому помогает команда Save As Template в контекстном меню, всплывающем при щелчке правой кнопкой мыши в окне Конструктора Меню. Эта команда вызывает диалог, в котором вы можете указать описание (заголовок), под которым хотите сохранить ваше меню.

Впоследствии в любом вашем новом приложении вы можете загрузить этот шаблон в меню, выбирая из всплывающего меню в окне Конструктора Меню команду Insert From Template.

Основные свойства

Свойство	Объявление / Описание
AutoHotKeys	enum **TMenuItemAutoFlag** { ma **Automatic, maManual,** maParent }; typedef TMenuItemAutoFlag TMenuAutoFlag; TMenuAutoFlag AutoHotkeys Определяет, могут ли «горячие» клавиши элементов меню устанавливаться автоматически
AutoLine Reduction	**bool** AutoLineReduction Указывает, проверяет ли и исправляет ли C++Builder появление в начале или конце подменю разделителей или двух разделителей подряд
AutoMerge	bool **AutoMerge** Определяет, объединятся ли главные меню вспомогательных форм с главным меню основной формы
Images	**Imglist::TCustomImageList*** Images Определяет список изображений, которые могут отображаться в разделах меню слева от надписи. Изображения конкретных разделов указываются свойством **ImageIndex** (индексом массива Images) разделов меню Items типа TMenuItem
Items	TMenuItem* Items Список элементов (разделов) меню типа TMenuItem

Основные методы

Свойство	Объявление / Описание
FindItem	enum **TFindItemKind {fkCommand, fkHandle, fkShortCut};** TMenuItem* **FindItem(int** Value, TFindItemKind Kind) Ищет и возвращает раздел меню, индентифицируемый значением Value, которое равно: при Kind = fkCommand — идентификатору ID, используемому в сообщении Windows **WM_COMMAND;** при Kind = fkHandle — дескриптору всплывающего меню; при Kind = fkShortCut — коду «горячих» клавиш
IsShortCut	bool **IsShortCut(Messages::TWMKey** &Message) Распознает «горячие» клавиши, связанные с разделом меню, и выполняет этот раздел меню
Merge	void **Merge(TMainMenu*** Menu) Объединяет данное меню с указанным меню Menu вспомогательной формы
Unmerge	void **Unmerge(TMainMenu*** Menu) Уничтожает объединение указанного меню Menu вспомогательной формы с данным меню

События

Событие	Описание
OnChange	Наступает при изменении меню

При работе с меню основные события связаны не с самим объектом меню, а с объектами его разделов типа **TMenuItem**, из которых главное — **OnClick**, наступающее при щелчке на разделе или при нажатии «горячих» клавиш или клавиш быстрого доступа.

MaskEdit — окно редактирования с шаблонами

Окно редактирования для ввода пользователем текстов с возможностью использовать шаблоны.

Страница библиотеки *Additional*

Класс *TMaskEdit*

Иерархия *TObfect — TPersistent — TComponent — TControl — TWinControl — TCustomEdit — TCustomMaskEdit*

Модуль *buttons*

Примеры изображения

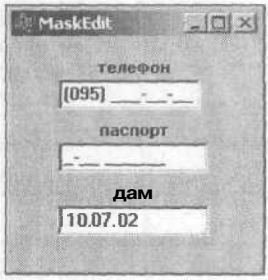


Рис. 2.26
Примеры компонентов MaskEdit

Описание

Компонент **MaskEdit** аналогичен по своим свойствам компоненту **Edit**, позволяет редактировать в окне однострочные тексты без возможности их выравнивания и переноса на следующую строку. Компонент отличается от **Edit** возможностью задавать строку маски (свойство **EditMask),** в каждой позиции которой записываются условные символы, буквы или цифры, кодирующие возможности записи в этих позициях определенных знаков, например, только цифр или только букв. На рис. 2.26 вверху показано окно, в которое вводится по шаблону телефон с кодом страны, в середине — шаблон ввода паспорта, внизу — введенная по шаблону дата

Окно редактирования снабжено многими функциями, свойственными большинству редакторов. Например, в нем предусмотрены типичные комбинации «горячих» клавиш: Ctrl-C — копирование выделенного текста в буфер обмена Clipboard (команда Copy), Ctrl-X — вырезание выделенного текста в буфер Clipboard (команда Cut), Ctrl-V — вставка текста из буфера Clipboard в позицию курсора (команда Paste), Ctrl-Z — отмена последней команды редактирования.

Основное свойство, отличающее компонент **MaskEdit от Edit** — строка маски **EditMask.** Маска состоит из трех разделов, между которыми ставится точка с запятой (;). В первом разделе — шаблоне записываются специальным образом символы, которые можно вводить в каждой позиции, и символы, добавляемые самой

маской; во втором разделе записывается 1 или 0 в зависимости от того, надо или нет, чтобы символы, добавляемые маской, включались в свойство Text компонента; в третьем разделе указывается символ, используемый для обозначения позиций, в которых еще не осуществлен ввод. Подробное описание формата маски см. в гл. 3, в разд. «EditMask и EditMaskPtr».

Прочитать результат ввода можно или в свойстве Text, которое в зависимости от вида второго раздела маски включает или не включает в себя символы маски, или в свойстве **EditText**, содержащем введенный текст вместе с символами маски.

Вводить маску можно непосредственно в свойство EditMask. Но удобнее пользоваться специальным редактором масок, вызываемым при нажатии кнопки с многоточием в строке свойства EditMask в Инспекторе Объектов. Окно редактора масок имеет вид, показанный на рис. 2.27.

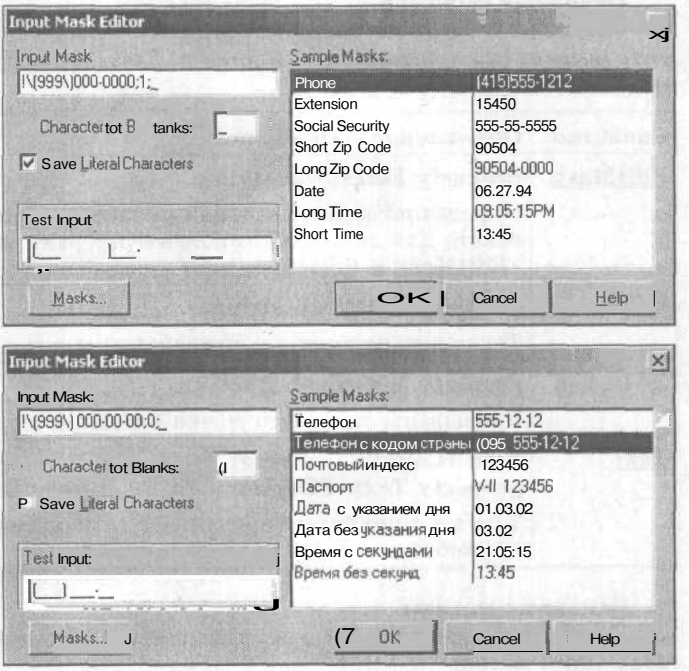


Рис. 2.27
Окно редактора масок с загруженными файлами, стандартных масок: американским (а) и российским (б)

В редакторе масок окно Sample Masks содержит наименования стандартных масок и примеры ввода с их помощью. В окно Input Mask надо ввести маску. Если вы выбираете одну из стандартных масок, то окно Input Mask автоматически заполняется и вы можете, если хотите, отредактировать эту маску.

Окно Character for Blanks определяет символ, используемый для обозначения позиций, в которых еще не осуществлен ввод (третий раздел маски). Индикатор Save Literal Characters определяет второй раздел маски: установлен, если второй раздел равен 1, и не установлен, если второй раздел равен 0.

Кнопка Masks позволяет выбрать и загрузить какой-либо другой файл стандартных масок. Среди файлов стандартных масок, поставляемых с C++Builder, отсутствует маска, соответствующая российским стандартам. Но вы легко можете сами сделать себе такой файл стандартных масок. Он делается в обычном текстовом редакторе и должен сохраняться как «только текст» с расширением .dem. Чтобы редактор масок C++Builder видел этот файл, его надо сохранить в каталоге C++Builder BIN. Каждая строка файла состоит из трех частей, разделяемых символом вертикальной черты. Первая часть состоит из пояснительного текста, появляющегося в левой панели окна Sample Masks редактора масок. Вторая часть –

пример, который появляется в правой панели окна Sample Masks редактора масок. А третья часть — сама маска. Пример подобного файла:

```
Телефон | 5551212 | !000-00-00;0;_
Телефон с кодом страны | 0955551212 | !\(999\) 000-00-00;0;_
Почтовый индекс | 123456 I !0000000;1;_
Паспорт| VII123456 | !L-LL 999999;0;_
Дата с указанием дня | 270694 | !99/99/00;1;_
Дата без указания дня | 0694 | !99/00;1;_
Время с секундами I 210515 | !90:00:00;1;_
Время без секунд I 1345 I !90:00;1;_
```

На показанном выше рис. 2.27 б) вы можете видеть его загруженным в окно редактора, а на рис. 2.26 вы видите окна, использующие его шаблоны при вводе данных.

Основные свойства

Компонент имеет все свойства обычного окна редактирования, которые вы можете посмотреть в описании компонента **Edit,** плюс следующие свойства, связанные с масками:

Свойство	Объявление / Описание
EditMask	property EditMask: string; Определяет маску, которая указывает допустимую форму записи текста для данного компонента (формат маски см. в гл. 3, в разд. «EditMask и EditMaskPtr»)
EditText	property EditText: string; Текст в окне вместе с символами маски
IsMasked	property IsMasked: Boolean; Указывает, задана ли маска EditMask
Text	type TCaption = string; property Text: TCaption; Текст в окне редактирования с символами маски или без них в зависимости от второго раздела маски

Основные методы

Компонент имеет все методы, указанные в разделе, описывающем **Edit,** плюс следующий метод:

Метод	Объявление / Описание
ValidateEdit	void ValidateEdit(void) Проверяет соответствие маске текста в окне. Если нет соответствия, генерируется исключение и курсор устанавливается на первую позицию, не соответствующую маске

События

Все события см. в описании компонента Edit.

Memo — многострочное окно редактирования

Многострочное окно редактирования, используется для ввода, отображения и редактирования многострочных текстов.

Страница библиотеки *Standard*

Класс *TMemo*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWinControl — TCustomEdit — TCustomMemo*

Модуль *stdctrls*

Пример изображения

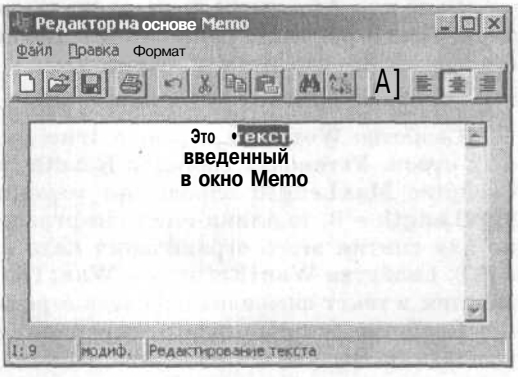


Рис. 2.28
Текстовый редактор на основе Memo

Описание

Memo — многострочный текстовый редактор, позволяющий редактировать текст окна, в которое можно вводить в отличие от компонента **Edit** не одну, а множество строк. Выполняет функции большинства редакторов, имеет "горячие" клавиши для быстрого редактирования. Формат всего текста одинаков и определяется свойством **Font.** В этом его отличие от многострочного окна редактирования **RichEdit.** которое допускает раздельное форматирование отдельных фрагментов текста. Если вы сохраните в файле текст, введенный или отредактированный в **Memo** пользователем, то будет создан текстовый файл, содержащий только символы и не содержащий элементов форматирования. При последующем чтении этого файла в **Memo** формат будет определяться текущим состоянием свойства **Font** компонента **Memo,** а не тем, в каком формате ранее вводился текст.

Изменять атрибуты шрифта можно во время проектирования или программно. Например, операторы

```
FontDialog1->Font->Assign(Memo1->Font);
if  (FontDialog1->Execute())
  Memo1->Font->Assign(FontDialog1->Font);
```

загружают в диалог выбора шрифта **FontDialogl** текущий шрифт **Memo1** как значение по умолчанию, вызывают диалог выбора шрифта и, если пользователь выбрал в нем шрифт, копируют этот шрифт в окно **Memol.**

Свойство **Lines,** доступное как во время проектирования, так и во время выполнения, имеет множество свойств и методов типа TStrings. которые обычно используются для формирования и редактирования текста (см. в гл. 1). Весь текст содержится в свойстве **Text.** Имеются также свойства только времени выполнения **SelLength, SelStart, SelText,** определяющие соответственно длину выделенного текста, позицию перед первым символом выделенного текста и сам выделенный текст. Если выделенного текста нет, то свойство **SelStart** просто определяет текущее положение курсора.

Окно редактирования снабжено многими функциями, свойственными большинству редакторов. Например, в нем предусмотрены типичные комбинации «горячих» клавиш: Ctrl-C — копирование выделенного текста в буфер обмена Clipboard (команда Copy), Ctrl-X — вырезание выделенного текста в буфер Clipboard (команда Cut), Ctrl-V — вставка текста из буфера Clipboard в позицию курсора (команда Paste), Ctrl-Z — отмена последней команды редактирования.

Свойство **Modified,** доступное только во время выполнения, показывает, проводилось ли редактирование текста в окне. Если вы хотите использовать это свойство, то в момент начала работы пользователя с текстом **Modified** надо установить в **false.** Тогда при последующем обращении к этому свойству можно по его значению **(true или false)** установить, было или не было произведено редактирование.

Свойство **Alignment** определяет выравнивание текста (влево, вправо, по центру). Таким образом, например, выполнение команды выравнивания по центру (см. рис. 2.28) сводится к оператору

```
Memo1->Alignment = taCenter;
```

Свойство **Wordwrap,** равное **true,** указывает на допустимость переноса длинных строк. Установка свойства **ReadOnly в true** задает текст только для чтения. Свойство **MaxLength** определяет максимальную длину вводимого текста. Если **MaxLength** = 0, то длина текста не ограничена (реально она все равно ограничена, но для снятия этого ограничения надо использовать сообщения Windows — см. в [3]). Свойства **WantReturns и WantTab** определяют допустимость ввода пользователем в текст символов перевода строки и табуляции.

Свойство ScrollBars определяет наличие полос прокрутки текста в окне. По умолчанию **ScrollBars = ssNone,** что означает их отсутствие. Пользователь может в этом случае перемещаться по тексту только с помощью курсора. Можно задать свойству **ScrollBars** значения **ssHorizontal, ssVertical** (это задано в примере на рис. 2.28) или **ssBoth,** что будет соответственно означать наличие горизонтальной, вертикальной или обеих полос прокрутки.

Свойство **CaretPos** указывает на запись, поле X которой содержит индекс символа в строке, перед которым расположен курсор, а поле Y — индекс строки, в которой находится курсор. Таким образом, учитывая, что индексы начинаются с О, значения **Memo1->CaretPos.Y+1 и Memo1->CaretPos.X+l** определяют соответственно номер строки и символа в ней, перед которым расположен курсор. В редакторе на рис. 2.28 именно эти значения использованы, чтобы отображать в строке состояния **StatusBar** позицию курсора.

Свойства Align и Anchors позволяют адаптировать размер окна **Memo** к размеру окна приложения, выбранного пользователем.

Основные свойства

Свойство	Объявление / Описание
Align	**enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom};** **TAlign Align** Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Alignment	**enum T Alignment { taLeft Justify, taRight Justify, taCenter };** **Classes::TAlignment Alignment** Управляет выравниванием текста: **taLeftJustify** — влево, **taRightJustify** — вправо, **taCenter** — по центру. Значение по умолчанию — влево
Anchors	**enum TAnchorKind { akLeft, akTop, akRight, akBottom };** **typedef Set<TAnchorKind, akLeft, akBottom> TAnchors;** **TAnchors Anchors** Определяет привязку данного компонента к родительскому при изменении размеров последнего

Свойство	Объявление / Описание
CanUndo	**bool** CanUndo Указывает, содержит ли компонент изменения, которые можно отменить. Доступ только для чтения
CaretPos	TPoint CaretPos Указывает строку и символ расположения курсора (см. в приведенном ранее описании Memo). Доступ только для чтения
Font	Graphics: :TFont* Font Определяет атрибуты шрифта
Lines	**Classes::TStrings*** Lines Строки текста — объект типа **TStrings**
MaxLength	int MaxLength Указывает максимальное количество символов, которое пользователь может вводить в компонент. При значении 0 длина текста неограничена
Modified	bool **Modified** Указывает, редактировался ли пользователем текст в компоненте
PopupMenu	**Menus::TPopupMenu*** PopupMenu Определяет всплывающее меню, связанное с данным компонентом
Readonly	bool ReadOnly Указывает, может ли пользователь изменять текст в компоненте
SelLength	int SelLength Определяет количество выделенных символов в тексте
SelStart	int SelStart Указывает позицию первого выделенного символа в тексте или, если выделения нет, то позицию курсора
SelText	**AnsiString** SelText Текст, выделенный в окне
Text	AnsiString Text Текст окна в виде одной строки
WantReturns	bool WantReturns Указывает, можно ли вставить в текст символы возврата каретки
WantTabs	bool WantTabs Указывает, можно ли вставить в текст символы табуляции
Wordwrap	bool Wordwrap Указывает, переносится ли текст на новую строку, если он превышает ширину компонента

Основные методы

Метод	**Объявление / Описание**
Clear	**void Clear(void)** **Удаляет текст из окна**

Метод	Объявление / Описание
ClearSelection	void **ClearSelection(void)** Удаляет текст, выделенный в окне
ClearUndo	void ClearUndo(void) Очищает буфер отмены команд редактирования, так что никакие изменения в тексте после этого не могут быть отменены
CopyTo Clipboard	void CopyToClipboard(void) Копирует выделенный текст в компоненте редактирования в Clipboard в формате CF_TEXT
CutTo Clipboard	void CutToClipboard(void) Переносит выделенный текст в Clipboard в формате CF_TEXT и уничтожает его в окне
PasteFrom Clipboard	void PasteFromClipboard(void) Переносит в окно в позицию SelStart текст из буфера Clipboard
Perform	int **Perform(unsigned** Msg, int **WParam,** int LParam) Передает окну сообщение Windows Msg с параметрами WParam и LParam
SelectAll	void **SelectAll(void)** Выделяет весь текст в окне редактирования
Undo	void **Undo(void)** Отменяет все изменения, хранившиеся в буфере отмены результатов редактирования с момента последнего вызова ClearUndo

Основные события

Событие	Описание
OnChange	Наступает, когда текст в окне может быть изменился. Свойство Modified показывает, действительно ли произошло изменение
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKevUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу

OpenDialog, OpenPictureDialog, SaveDialog, SavePictureDialog — диалоги работы с файлами

Невизуальные компоненты вызова стандартных диалогов Windows открытия и сохранения файлов.

Страница библиотеки *Dialogs*

Классы *TOpenDialog*, *TOpenPictureDialog*, *TSaveDialog*, *TSavePictureDialog*

Иерархия *TObiect* — *TPersistent* — *TComponent* — *TCommonDialog*

Модуль *dialogs*

Описание

Компоненты **OpenDialog и SaveDialog** вызывают стандартные диалоги Windows открытия и сохранения файлов, показанные на рис. 2.29 а) и б). Компоненты **OpenPictureDialog и SavePictureDialog** вызывают аналогичные стандартные диалоги Windows открытия и сохранения файлов изображений, показанные на рис. 2.29 в) и г).

Отрытие соответствующего диалога осуществляется методом **Execute.** Если в диалоге пользователь нажмет кнопку Открыть (Сохранить), диалог закрывается, метод **Execute** возвращает **true** и выбранный файл отображается в свойстве компонента-диалога **FileName.** Если же пользователь отказался от диалога (нажал кнопку Отмена или клавишу Esc), то метод **Execute** возвращает **false.**

Значение свойства **FileName** можно задать и перед обращением к диалогу. Тогда оно появится в диалоге как значение по умолчанию в окне Имя файла (см. рис. 2.29). Таким образом, например, выполнение команды Сохранить как, по которой в файле с выбранным пользователем именем надо сохранить текст окна редактирования **RichEditl,** может иметь вид:

```
SaveDialog1->FileName = FName; // Задание имени по умолчанию
if  (SaveDialog1->Execute())
{
 FName = OpenDialog1->FileName;
 RichEdit1->Lines->SaveToFile(FName);
}
```

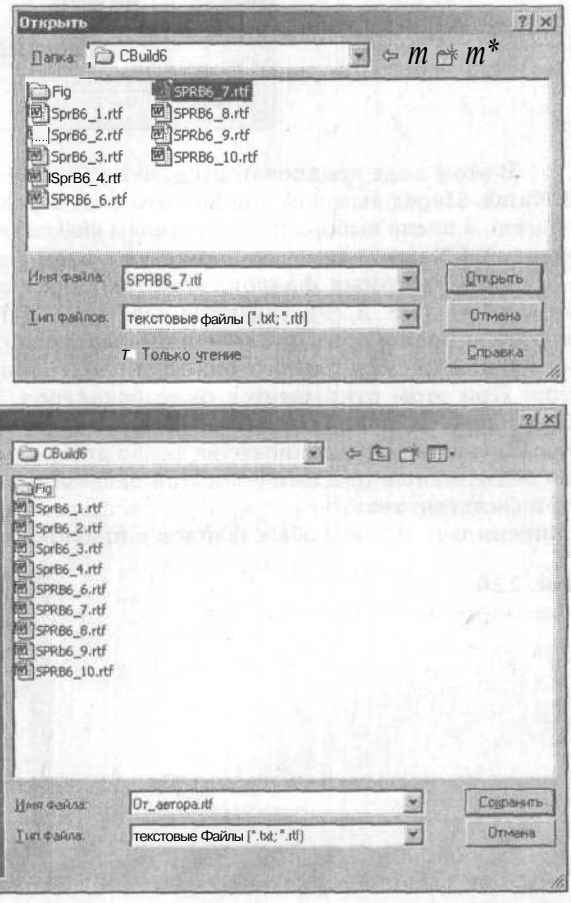


Рис. 2.29
Диалоговые окна открытия файла (а), сохранения файла (б)

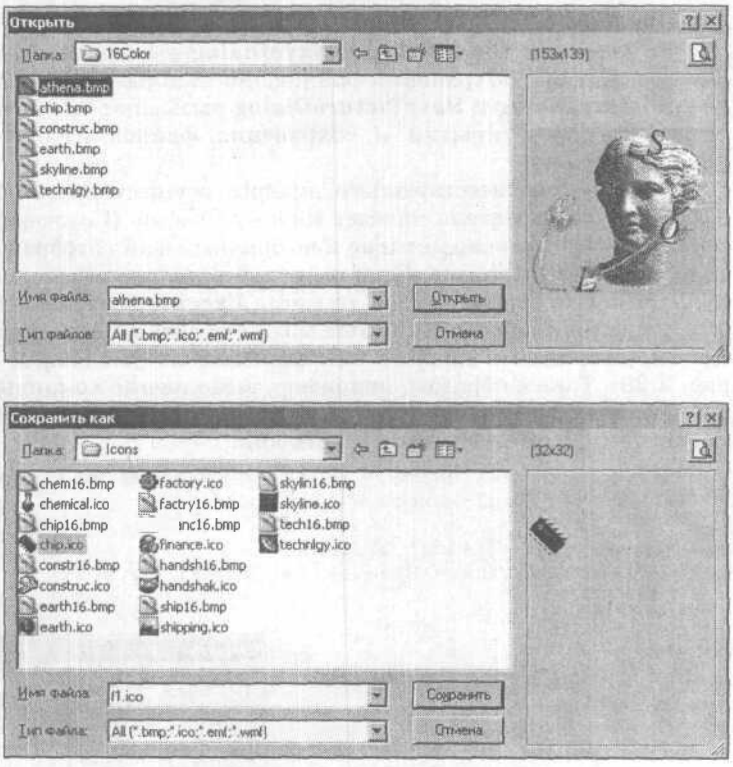


Рис. 2.29
Диалоговые окна открытия файла изображения (в), сохранения файла изображения (г)

В этом коде предполагается, что имя файла хранится в строковой переменной **FName.** Перед вызовом диалога это имя передается в него как имя файла по умолчанию, а после выбора пользователем файла его выбор запоминается в той же переменной **FName** и текст сохраняется в этом файле методом **SaveToFile.**

Типы искомых файлов, появляющиеся в диалоге в выпадающем списке Тип файла (см. рис. 2.29), задаются свойством **Filter.** В процессе проектирования это свойство проще всего задать с помощью редактора фильтров, который вызывается нажатием кнопки с многоточием около имени этого свойства в Инспекторе Объектов. При этом открывается окно редактора, вид которого показан на рис. 2.30. В его левой панели Filter Name вы записываете тот текст, который увидит пользователь в выпадающем списке Тип файла диалога. А в правой панели Filter записываются разделенные точками с запятой шаблоны фильтра. В примере на рисунке задано три фильтра: текстовых файлов с расширениями .txt и .rtf, файлов документов с расширениями .doc и любых файлов с шаблоном *.*.

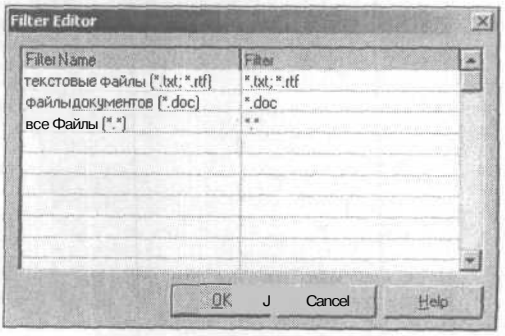


Рис. 2.30
Диалоговое окно задания фильтра

После выхода из окна редактирования фильтров заданные вами шаблоны появятся в свойстве **Filter** в виде строки вида:

```
текстовые файлы (*.txt; *.rtf)|*.txt;*.rtf|
                файлы документов (*.doc)|*.doc|все файлы|*.*
```

В этой строке тексты и шаблоны разделяются вертикальными линиями. В аналогичном виде, если требуется, можно задавать свойство **Filter** программно во время выполнения приложения.

Свойство **FilterIndex** определяет номер фильтра, который будет по умолчанию показан пользователю в момент открытия диалога. Например, значение **FilterIndex** = 1 задает по умолчанию первый фильтр.

Свойство **InitialDir** определяет начальный каталог, который будет открыт в момент начала работы пользователя с диалогом. Если значение этого свойства не задано, то открывается текущий каталог или тот, который был открыт при последнем обращении пользователя к соответствующему диалогу в процессе выполнения данного приложения.

Свойство **DefaultExt** определяет значение расширения файла по умолчанию. Если значение этого свойства не задано, пользователь должен указать в диалоге полное имя файла с расширением. Если же задать значение **DefaultExt** (например, "txt"), то пользователь может писать в диалоге имя без расширения. В этом случае будет принято заданное расширение.

Свойство **Title** позволяет вам задать заголовок диалогового окна. Если это свойство не задано, окно открывается с заголовком, определенным в системе. Но вы можете задать и свой заголовок, подсказывающий пользователю ожидаемые действия. Например, «Укажите имя открываемого файла».

Свойство **Options** определяет условия выбора файла. Множество опций, которые вы можете установить программно или во время проектирования, включает:

ofAllowMultiSelect	Позволяет пользователю выбирать несколько файлов
ofCreatePrompt	В случае, если пользователь написал имя несуществующего файла, появляется замечание и запрос, надо ли создать файл с заданным именем
ofEnableIncludeNotify	Посылает сообщение CDN_INCLUDEITE каждому элементу открывающейся папки. Может использоваться для управления тем, какие элементы отображаются в папке. Начиная с Windows 2000
ofEnableSizing	Разрешает пользователю изменять размер диалогового окна
ofExtensionDifferent	Этот флаг, который можно прочитать после выполнения диалога, показывает, что расширение файла, выбранного пользователем, отличается от DefaultExt
ofFileMustExist	В случае, если пользователь написал имя несуществующего файла, появляется сообщение об ошибке
ofHideReadOnly	Удаляет из диалога индикатор Открыть только для чтения (выключена на рис. 2.29 а)
ofNoChangeDir	После щелчка пользователя на кнопке OK восстанавливает текущий каталог, независимо от того, какой каталог был открыть при поиске файла
ofNoDereferenceLinks	Запрещает переназначать клавиши быстрого доступа в диалоговом окне

ofNoLongNames	Отображаются только не более 8 символов имени и трех символов расширения
ofNoNetworkButton	Убирает из диалогового окна кнопку поиска в сети. Действует только если флаг ofOldStyleDialog включен
ofNoReadOnlyReturn	Если пользователь выбрал файл только для чтения, то генерируется сообщение об ошибке
ofNoTestFileCreate	Запрещает выбор в сети защищенных файлов и не доступных дисков при сохранении файла
ofNoValidate	Не позволяет писать в именах файлов неразрешенные символы, но не мешает выбирать файлы с неразрешенными символами
ofOldStyleDialog	Создает диалог выбора файла в старом стиле
ofOverwritePrompt	В случае, если при сохранении файла пользователь написал имя существующего файла, появляется замечание, что файл с таким именем существует, и запрашивается желание пользователя переписать существующий файл
ofPathMustExist	Генерирует сообщение об ошибке, если пользователь указал в имени файла несуществующий каталог
ofReadOnly	По умолчанию устанавливает индикатор Открыть только для чтения при открытии диалога
ofShareAware	Игнорирует ошибки нарушения условий коллективного доступа и разрешает, несмотря на них, производить выбор файла
ofShowHelp	Отображает в диалоговом окне кнопку Справка (включена на рис. 2.29 а)
ofDontAddToRecent	Предотвращает включение файла в списрк недавно открывавшихся файлов
ofShowHidden	Делает невидимые файлы видимыми в диалоге

По умолчанию все перечисленные опции, кроме **ofHideReadOnly**,выключены. Но, как видно из их описания, многие из них полезно включить перед вызовом диалогов.

Имеется еще одно свойство — **OptionsEx.** Это расширенное множество опций, правда, содержащее пока только одну выключенную по умолчанию опцию:

ofExNoPlacesBar	Запрещает отображение панели быстрых кнопок Рабочий стол, Мои документы и др. (включена на рис. 2.29 б и выключена на остальных)

Если вы разрешаете с помощью опции **ofAllowMultiSelect** множественный выбор файлов, то список выбранных файлов можно прочитать в свойстве **Files** типа **TStrings.**

Свойства компонентов **OpenPictureDialog и SavePictureDialog** ничем не отличаются от свойств компонентов **OpenDialog** и SaveDialog. Единственное отличие — заданное значение по умолчанию свойства **Filter в OpenPictureDialog и SavePictureDialog.** В этих компонентах заданы следующие фильтры:

All (*.jpg;*.jpeg;*.bmp;*.ico;*.emf;*.wmf)	*.jpg;*.jpeg;*.bmp;*.ico;*.emf;*.wmf
JPEG Image File (*.jpg)	*.jpg
JPEG Image File (*.jpeg)	*.jpeg
Bitmaps (*.bmp)	*.bmp
Icons (*.ico)	*.ico
Enhanced Metafiles (*.emf)	*.emf
Metafiles (*.wmf)	*.wmf

В этих фильтрах перечислены все типы графических файлов, с которыми может работать диалог. Так что вам остается удалить, если хотите, фильтры тех файлов, с которыми вы не хотите работать, добавить, может быть, фильтр "Все файлы (*.*)" и перевести на русский язык названия типов.

В компонентах диалогов открытия и сохранения файлов предусмотрена возможность обработки ряда событий. Такая обработка может потребоваться, если рассмотренных опций, несмотря на их количество, не хватает, чтобы установить все диктуемые конкретным приложением ограничения на выбор файлов. Событие **OnCanClose** возникает при нормальном закрытии пользователем диалогового окна после выбора файла. При отказе пользователя от диалога — нажатии кнопки Отмена, клавиши Esc и т.д. событие **OnCanClose** не наступает. В обработке события **OnCanClose** вы можете произвести дополнительные проверки выбранного пользователем файла и, если по условиям вашей задачи этот выбор недопустим, вы можете известить об этом пользователя и задать значение **false** передаваемому в обработчик параметру **CanClose.** Это не позволит пользователю закрыть диалоговое окно.

Можно также написать обработчики событий **OnFolderChange** — изменение каталога, **OnSelectionChange** — изменение имени файла, **OnTypeChange** — изменение типа файла. В этих обработчиках вы можете предусмотреть какие-то сообщения пользователю.

Основные свойства

Свойство	Объявление / Описание
DefaultExt	**AnsiString** DefaultExt Определяет заданное по умолчанию расширение файла
FileName	AnsiString FileName Имя выбранного файла
Files	Classes::TStrings* Files Список выбранных файлов, если пользователю разрешен множественный выбор (включена опция of **AllowMultiSelect**)
Filter	AnsiString Filter Определяет маски файла (фильтры), доступные в диалоге
FilterIndex	AnsiString **InitialDir** Определяет индекс фильтра по умолчанию при открытии диалога
InitialDir	AnsiString InitialDir Определяет каталог при открытии диалога

Свойство	Объявление / Описание
Options	enum TOpenOption {**ofReadOnly, ofOverwritePrompt, ofHideReadOnly, ofNoChangeDir, ofShowHelp, ofNoValidate, ofAllowMultiSelect, ofExtensionDifferent,** ofPathMustExist, **ofFileMustExist, ofCreatePrompt,** ofShareAware, **ofNoReadOnlyReturn, ofNoTestFileCreate, ofNoNetworkButton,** ofNoLongNames, **ofOldStyleDialog, ofNoDereferenceLinks, ofEnableIncludeNotify,ofEnableSizing**}; typedef Set<TOpenOption, ofReadOnly, **ofNoDereferenceLinks**> TOpenOptions; TOpenOptions Options Различные опции компонента (см. приведенное выше описание диалогов)
OptionsEx	enum TOpenOptionEx { **ofExNoPlacesBar** } typedef Set<TOpenOptionEx, ofExNoPIacesBar, **ofExNoPlacesBar**> TOpenOptionsEx; TOpenOptionsEx OptionsEx Расширенное множество опций (см. приведенное выше описание диалогов)
Title	AnsiString Title Заголовок диалогового окна

Основные методы

Метод	Объявление / Описание
Execute	**bool** Execute(void) Вызывает диалог, возвращает true, если пользователь произвел выбор в диалоге

Остальные методы наследуются от классов-предшественников.

Основные события

Событие	Описание
OnCanCIose	Событие наступает при нормальном закрытии пользователем диалогового окна после выбора файла
OnFolder Change	Событие наступает при изменении пользователем каталога
OnInclude Item	Событие наступает перед добавлением файла в список
OnSelection Change	Событие наступает перед изменением пользователем (любым способом) списка отображаемых файлов
OnType Change	Событие наступает перед изменением пользователем фильтра, задающего типы файлов
OnCIose	Событие наступает при закрытии диалога
OnShow	Событие наступает при открытии диалога

OpenPictureDialog — диалога открытия файла изображения

Невизуальный компонент вызова диалога открытия файла изображения.

См. раздел «OpenDialog, OpenPictureDialog, SaveDialog, SavePictureDialog – диалоги работы с файлами».

Panel — панель

Панель — контейнер для группировки других компонентов. Может использоваться также как компонент отображения текста.

Страница библиотеки *Standard*

Класс *TPanel*

Иерархия *TObiect — TPersistent — TComponent — TControl – TWinControl — TCustomPanel*

Модуль *extctrls*

Примеры изображения

Рис. 2.31
Примеры оформления панелей Panel

Описание

Компонент **Panel** представляет собой панель, которая служит контейнером, объединяющим группу управляющих компонентов, компонентов ввода и отображения информации, других, меньших контейнеров. Панель можно использовать также для построения полос состояния и инструментальных панелей.

Свойства **BorderStyle, BevelInner, BevelOuter, BevelWidth, BorderWidth** определяют обрамление — бордюр панели, предоставляя богатые возможности ее оформления (см. рис. 2.31). Свойства **Aline** (выравнивание компонента по всей верхней, левой, правой, нижней частям контейнера или по всей его клиентской области), **Anchors** (привязка сторон компонента к сторонам контейнера), **Constraints** (ограничения допустимых изменений размеров) определяют изменение размеров панели при изменении контейнера, в котором она размещена, например, при изменении пользователем размеров окна приложения.

Свойство **Caption** — текст, отображаемый в панели. Свойство **Alignment** определяет выравнивание этого текста.

Основное назначение панелей — визуальное объединение различных элементов интерфейса (кнопок, окон редактирования, списков), функционально связанных друг с другом. Такая функциональная связь должна поддерживаться и зри-

тельной связью — объединением соответствующих элементов в рамках одной панели. Свойство AutoSize определяет, будут ли размеры панели автоматически подстраиваться под размещенные в ней компоненты.

Объединение панелью таких компонентов, как радиокнопок RadioButton, обеспечивает их функционирование как единой группы: включение одной из таких радиокнопок выключает остальные.

Основные свойства

Свойство	Объявление / Описание
Align	enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom}; TAlign Align Определяет способ выравнивания компонента в контейнере (родительскомкомпоненте)
Alignment	enum TAlignment { taLeftJustify, taRightJustify, taCenter }; Classes::TAlignment Alignment Определяет способ выравнивания текста внутри компонента — по левому краю, правому краю, центру. Значение по умолчанию — taCenter — по центру
Anchors	enum TAnchorKind { akLeft, akTop, akRight, akBottom }; typedef Set<TAnchorKind, akLeft, akBottom> TAnchors; TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
AutoSize	bool AutoSize Определяет, будут ли размеры панели автоматически подстраиваться под размещенные в ней компоненты
BevelInner	enum TBevelCut {bvNone, bvLowered, bvRaised, bvSpace}; TBevelCut BevelInner Определяет выпуклый, утопленный или плоский вид внутренней части компонента
BevelOuter	enum TBevelCut {bvNone, bvLowered, bvRaised, bvSpace}; TBevelCut BevelOuter Определяет выпуклый, утопленный или плоский вид обрамления компонента
BevelWidth	typedef int TBevelWidth; TBevelWidth BevelWidth Определяет ширину обрамления компонента в пикселах
BorderStyle	enum TFormBorderStyle { bsNone, bsSingle, bsSizeable, bsDialog, bsToolWindow, bsSizeToolWin }; typedef TFormBorderStyle TBorderStyle; Forms::TBorderStyle BorderStyle Указывает, ограничена ли клиентская область компонента одинарной бордюрной линией
Border Width	TBorderWidth BorderWidth Расстояние в пикселах между наружной и внутренней кромками обрамления

Свойство	Объявление / Описание
Brush	**Graphics::TBrush*** Brush Кисть, определяющая заполнение фона
Caption	AnsiString Caption Текст, отображаемый в панели
Font	Graphics::TFont* Font Определяет атрибуты шрифта
Constraints	**TSizeConstraints*** Constraints Позволяет задавать ограничения размера (максимально допустимую ширину и высоту) компонента. Во избежание неопределенности нельзя устанавливать ограничения, конфликтующие со свойствами Align и Anchors. По умолчанию ограничения отсутствуют
TabOrder	typedef short **TTabOrder;** TTabOrder TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	**bool** TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Никаких специальных методов в панели не объявлено. Методы наследуются от классов-предков **TWinControl и TControl.**

Основные события

Никаких специальных событий в панели не объявлено. События наследуются от классов-предков **TWinControl и TControl.**

PopUpMenu — всплывающее контекстное меню

Невизуальный компонент, позволяет создавать всплывающие контекстные меню, возникающие при нажатии пользователем правой кнопки мыши.

Страница библиотеки *Standard*

Класс *TPopupMenu*

Иерархия *TObject — TPersistent — TComponent — TMenu*

Модуль *menus*

Описание

Компонент **PopupMenu** определяет всплывающее контекстное меню, появляющееся на экране при щелчке пользователя правой кнопкой мыши в поле компонента, который связан с данным меню своим свойством **PopupMenu.**

Проектирование меню производится с помощью конструктора меню точно так же, как описано для компонента главного меню MainMenu (см. соответствующий раздел). Обратите только внимание на возможность упрощения этой работы. Поскольку разделы контекстного меню обычно повторяют некоторые разделы уже сформированного главного меню, то можно обойтись копированием соответствующих разделов. Для этого, войдя в конструктор меню из компонента **PopupMenu,** щелкните правой кнопкой мыши и из всплывшего меню выберите команду Select

Menu (выбрать меню). Вам будет предложено диалоговое окно, в котором вы можете перейти в главное меню. В нем вы можете выделить нужный вам раздел или разделы (при нажатой клавише Shift выделяются разделы в заданном диапазоне, при нажатой клавише Ctrl можно выделить совокупность разделов, не являющихся соседними). Затем выполните копирование их в буфер обмена, нажав клавиши Ctrl-C. После этого опять щелкните правой кнопкой мыши, выберите команду Select Menu и вернитесь в контекстное меню. Укажите курсором место, в которое хотите вставить скопированные разделы, и нажмите клавиши чтения из буфера обмена -- Ctrl-V. Разделы меню вместе со всеми их свойствами будут скопированы в создаваемое вами контекстное меню.

В остальном работа с **PopupMenu** не отличается от работы **с MainMenu.** Только не возникает вопросов объединения меню разных форм: контекстные меню не объединяются.

Основные свойства

Свойство	Объявление / Описание
Alignment	**enum TPopupAlignment { paLeft, paRight, paCenter }; TPopupAlignment Alignment** Определяет, какая точка меню появится под курсором мыши: **paLeft** — левый верхний угол, **paRight** — правый верхний угол, **paCenter** — центр. Значение по умолчанию — **paLeft**
AutoHotKeys	**enum TMenuItemAutoFlag { maAutomatic, maManual, maParent }; typedef TMenuItemAutoFlag TMenuAutoFlag; TMenuAutoFlag AutoHotkeys** Определяет, могут ли «горячие» клавиши элементов меню устанавливаться автоматически
AutoLine Reduction	**bool AutoLineReduction** Указывает, проверяет ли и исправляет ли C++Builder появление в начале или конце подменю разделителей или двух разделителей подряд
AutoPopup	**bool AutoPopup** Определяет автоматическое появление меню при щелчке правой кнопки мыши или определенной в Windows комбинации клавиш. Если **AutoPopup** равно **false,** то для отображения меню надо использовать метод **Popup**
HelpContext	**typedef int THelpContext;** **Classes::THelpContext HelpContext** Определяет идентификатор контекстной справки, связанной со всем меню в целом. Для задания контекстной справки отдельным разделам меню надо использовать свойство **HelpContext** объектов разделов **TMenuItem**
Images	**Imglist::TCustomImageList* Images** Определяет список изображений, которые могут отображаться в разделах меню слева от надписи. Изображения конкретных разделов указываются свойством **ImageIndex** (индексом массива Images) разделов меню **Items** типа **TMenuItem**
Items	**TMenuItem* Items** Список элементов (разделов) меню типа **TMenuItem**

Свойство	Объявление / Описание
MenuAnimation	enum **TMenuAnimations** {maLeftToRight, maRightToLeft, maTopToBottom, maBottomToTop, **maNone};** typedef Set<TMenuAnimations, maLeftToRight, maBottomToTop> TMenuAnimation; TMenuAnimation MenuAnimation Определяет способ появления меню на экране (только начиная с Windows 98 и NT 5.0): постепенное появление слева направо (maLeftToRight), справа налево (maRightToLeft), сверху вниз (maTopToBottom), снизу вверх (maBottomToTop) или мгновенное появление (maNone)
Popup Component	Classes::TComponent* PopupComponent Определяет компонент, который последним вызвал данное меню. Используется в меню, с которым связано несколько компонентов, чтобы определить, из какого именно компонента поступил вызов
PopupPoint	TPoint* PopupPoint Защищенное свойство. Указывает место появления всплывающего меню. Значение свойства устанавливается методом Popup
TrackButton	enum TTrackButton { tbRightButton, tbLeftButton **};** **TTrackButton** TrackButton Указывает, какая кнопка мыши активизирует всплывающее меню, если оно связано *с* кнопкой панели инструментов

Основные методы

Метод	Объявление / Описание
DoPopup	void DoPopup(TObject* Sender) Генерирует событие OnPopup
FindItem	enum **TFindItemKind {fkCommand, fkHandle, fkShortCut};** TMenuItem* **FindItem(int** Value, TFindItemKind Kind) Ищет и возвращает раздел меню, индентифицируемый значением Value, которое равно: при Kind = fkCommand — идентификатору ID, используемому в сообщении Windows **WM_COMMAND;** при Kind = fkHandle — дескриптору всплывающего меню; при Kind = fkShortCut — коду «горячих» клавиш
Popup	void **Popup(int** X, int Y) Отображает всплывающее меню в указанной позиции экрана

События

Событие	Описание
OnChange	Наступает при изменении меню
OnPopup	Событие происходит непосредственно перед появлением всплывающего меню

При работе с меню основные события связаны не с самим объектом меню, а с объектами его разделов типа **TMenuItem,** из которых главное — **OnClick.** наступающее при щелчке на разделе или при нажатии «горячих» клавиш или клавиш быстрого доступа.

Query — набор данных, использующий SQL

Невизуальный компонент набора данных, выполняющий запросы SQL.

Страница библиотеки *BDE,* до C++Builder 6 — *Data Access*

Класс *TQuery*

Иерархия *TObiect — TPersistent — TComponent — TDataSet — TBDEDataSet — TDBDataSet*

Модуль *Dbtables*

Описание

Компонент **Query** может во многих случаях включаться в приложения вместо **Table.** Преимущества **Query** по сравнению с **Table:**

- Возможность формирования набора данных из полей нескольких таблиц базы данных
- Формирование запросов на языке SQL, что обеспечивает большую гибкость, чем в **Table**
- При работе в сети с удаленным сервером (Sybase, SQL Server, Oracle, Informix, DB2, InterBase)

Query более эффективен, поскольку **Table** создает на компьютере пользователя временную копию серверной базы данных, что требует больших ресурсов и существенно загружает сеть. **Query** размещает на компьютере пользователя только результат запроса.

Основное свойство компонента **Query — SQL,** имеющее тип TStrings. Это список строк, содержащих запросы SQL (справочные данные по SQL даны в [1]). В процессе проектирования приложения обычно необходимо сформировать в этом свойстве некоторый предварительный запрос SQL, который показал бы, с какой таблицей или таблицами будет проводиться работа. Но далее во время выполнения приложения свойство SQL может формироваться программно методами, обычными для класса **TStrings: Clear** — очистка, Add — добавление строки и т.д.

Большинство свойств, методов и событий **Query** наследует от классов-предшественников, сведения о которых приводятся в гл. 1.

Для связи **Query** с необходимой базой данных служит свойство **DatabaseName.** В выпадающем списке этого свойства в Инспекторе Объектов вы можете видеть все доступные BDE псевдонимы баз данных и выбрать необходимый вам.

После того как указана база данных, можно устанавливать значение свойства SQL, содержащее запрос. Этот запрос обычно содержит оператор **Select** со списком всех полей таблиц и необходим для настройки компонента. В дальнейшем он может изменяться во время выполнения.

Соединение с выбранной таблицей базы данных осуществляется свойством **Active.** По умолчанию оно равно **false.** Если установить его в **true** во время проектирования или программно во время выполнения, то компонент соединится с базой данных.

Объекты полей, экспонируемых компонентом **Query,** могут создаваться автоматически. Но тогда их характеристики (надписи, число отводимых под них символов и т.п.) будут приняты по умолчанию и вряд ли устроят пользователя. Можно создавать и редактировать объесты полей с помощью специального Редактора Полей. Вызвать его проще всего двойным щелчком на компоненте **Query.** См. о работе с Редактором Полей в описании компонента **Table.**

Имеется множество свойств, методов и событий как объектов полей — наследников **TField.** так и базовых классов наборов данных, которым наследует **Query: TDataSet. TBDEDataSet. TDBDataSet.** Описание базовых классов вы найдете в гл. 1, а подробные описания свойств, методов, событий — в гл. 3, 4, 5. Но детальное рассмотрение методики работы с базами данных далеко выходит за рамки этой книги. Вы можете найти эту методику в [1].

Основные свойства

Свойство	Объявление / Описание
Active	**bool** Active Определяет, открыта база данных, или нет
Constrained	bool Constrained Указывает совместимость ограничений в предложении SELECT с операциями редактирования и вставки записей в таблицах Paradox и dBASE
Constraints	TCheckConstraints* Constraints Ограничения на допустимые значения параметров на уровне записи
Database	TDatabase Database Указатель на связанный с базой данных компонент Database. Обеспечивает доступ к его свойствам и методам. Автоматически устанавливается при открытии базы данных, указанной свойством DatabaseName
DatabaseName	**AnsiString** DatabaseName Определяет имя базы данных, связанной с набором данных
DataSource	Db::TDataSource* DataSource Указывает компонент **DataSource.** относящийся к другой, головной таблице, из которой берется значение ключа, которому должны соответствовать записи данной таблицы
Local	bool Local Определяет, относится ли запрос к локальным таблицам Paradox и dBASE, или к таблицам SQL и удаленного сервера. Устанавливается автоматически при вызове Prepare
ParamCheck	bool ParamCheck Определяет, должно ли свойство Params автоматически обновляться при изменении запроса в свойстве SQL во время выполнения
ParamCount	Word ParamCount Число параметров в свойстве Params
Params	TParams* Params Массив параметров запроса, содержащегося в SQL
Prepared	bool Prepared Определяет, подготовлен ли запрос к выполнению. Установка в true вызывает метод Prepare, установка в false — Unprepare. Но лучше вызывать Prepare и **Unprepare** непосредственно

Свойство	Объявление / Описание
RequestLive	**bool** RequestLive Позволяет попытаться возвращать результат запроса как «живой» набор данных, вместо таблицы только для чтения
RowsAffected	**int** RowsAffected Взращает число обновленных или удаленных записей в результате последнего запроса. Возвращает — 1, если запрос не удалось выполнить
SQL	**Classes::TStrings***SQL Основное свойство компонента, содержащее запрос SQL
SQLBinary	char * SQLBinary Используется внутри класса для указания на двоичный поток запроса SQL или результата запроса
StmtHandle	Bde::hDBIStmt StmtHandle Идентификатор прямого запроса к Borland Database Engine (BDE)
Text	**AnsiString** Text Текст запроса SQL, передаваемый в Borland Database Engine (BDE)
UniDirectional	bool UniDirectional Определяет доступность двунаправленного курсора Borland Database Engine (BDE)

Компонент наследует также много свойств классов **TDataSet. TBDEDataSet. TDBDataSet.**

Основные методы

Метод	Объявление / Описание
Append	void Append(void) Добавляет новую пустую запись в конец набора данных
AppendRecord	void AppendRecord(const System::TVarRec * Values, const int Values_Size) Добавляет в набор данных новую запись, заполняет ее и пересылает в базу данных
ApplyUpdates	void ApplyUpdates(void) Записывает кэшированные изменения в базу данных
Cancel	void Cancel(void) Отменяет результаты редактирования
CancelUDdates	void **CancelUpdates(void)** Отменяет все кэшированные изменения и восстанавливает исходное состояние набора данных
Close	void Close(void) Закрывает набор данных
CloseDatabase	void CIoseDatabase(TDatabase* Database) Закрывает соединение с базой данных Database

Метод	Объявление / Описание
Commit Updates	void **CommitUpdates(void)** Очищает буфер кэшированных изменений
Delete	void Delete(void) Удаляет активную запись и позиционирует курсор на следующую запись
Edit	void Edit(void) Переводит набор данных в режим редактирования
ExecSQL	void ExecSQL(void) Выполняет запрос, не связанный с SELECT (запросы INSERT, UPDATE, DELETE, CREATE TABLE). Для запроса SELECT используется метод Open
FieldByName	TField* FieldByName(const AnsiString FieldName) Находит поле по его имени FieldName. При неверном имени генерирует исключение
FindField	TField* FindField(const AnsiString FieldName) Находит поле по его имени. При неверном имени возвращает nil
FindFirst	**bool** FindFirst(void) Перемещает курсор к первой записи и возвращает **true** в случае успеха
FindLast	bool **FindLast(void)** Перемещает курсор к последней записи и возвращает **true** в случае успеха
FindNext	bool FindNext(void) Перемещает курсор к следующей записи и возвращает true в случае успеха
FindPrior	bool FindPrior(void) Перемещает курсор к предыдущей записи и возвращает **true** в случае успеха
First	void First(void) Перемещает курсор к первой записи
GetFieldNames	void **GetFieldNames(Classes::TStrings*** List) Выдает список имен всех полей набора данных
Insert	HIDESBASE void Insert(void) Вставляет новую пустую запись в набор данных
InsertRecord	void InsertRecord(const **System::TVarRec** * Values, const int Values_Size) Вставляет новую заполненную запись в набор данных
Last	void Last(void) Перемещает курсор к последней записи

Метод	Объявление / Описание
Locate	**bool** Locate(const AnsiString **KevFields,** const **System::Variant** &KeyValues, **TLocateOptions** Options) Осуществляет поиск записи в наборе данных
Lookup	**System::Variant Lookup(const** AnsiString KeyFields, const Variant **&KeyValues,** const AnsiString ResultFields) Осуществляет поиск записи в наборе данных и возвращает значения указанных полей этой записи
MoveBy	int **MoveBy(int** Distance) Перемещает курсор на заданное число записей
Next	void Next(void) Перемещает курсор к следующей записи
Open	void Open(void) Открывает соединение с базой данных и выполняет запрос SELECT, содержащийся в свойстве SQL. Для запросов иных типов используется метод ExecSQL
OpenDatabase	TDatabase* OpenDatabase(void) Открывает базу данных
ParamByName	TParam* **ParamByName(const** AnsiString Value) Возвращает объект параметра запроса с указанным именем
Post	void Post(void) Пересылает отредактированную запись в базу данных
Prepare	void Prepare(void) Подготавливает BDE и удаленный сервер к выполнению запроса. Подготовка ускоряет последующую обработку запроса
Prior	void Prior(void) Перемещает курсор к предыдущей записи
RevertRecord	void RevertRecord(void) Отменяет исправления текущей записи
UnPrepare	void UnPrepare(void) Освобождает ресурсы, выделенные ранее при выполнении Prepare для подготовки к выполнению запроса

Основные события

Событие	Описание
OnUpdateError	Наступает при генерации исключения в процессе пересылки в базу данных измененной записи.
OnUpdateRecord	Наступает при пересылке кэшированной записи в базу данных

Кроме того, наследуется множество событий класса **TDataSet**.

RadioButton — радиокнопка

Радиокнопка — компонент, используемый в совокупности с другими радиокнопками для выбора одной из взаимоисключающих альтернатив.

Страница библиотеки *Standard*

Класс *TRadioButton*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TButtonControl*

Модуль *stdctrls*

Пример изображения

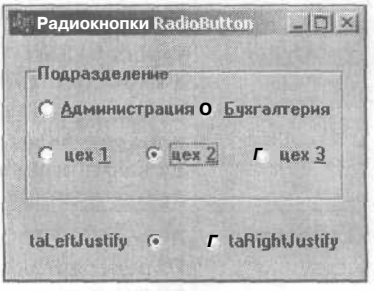


Рис. 2.32
Пример радиокнопок RadioButton

Описание

Компонент **RadioButton** используется, как правило, в группе других радиокнопок для выбора одной из взаимоисключающих альтернатив. Из кнопок, объединенных в группу, включена может быть только одна. При включении одной кнопки группы остальные кнопки автоматически выключаются. Объединение радиокнопок в группу осуществляется обычно панелями: **GroupBox** (см. верхнюю часть рис. 2.23), **Panel** и др. Радиокнопки, размещенные непосредственно на форме (внизу на рис. 2.32), тоже образуют группу. Имеется также компонент группы радиокнопок **RadioGroup.** в который не надо переносить кнопки — они присутствуют там изначально и располагаются регулярными рядами. Отдельные компоненты **RadioButton,** объединенные контейнерами, имеет смысл использовать только при необходимости нерегулярного размещения кнопок (сравните рис. 2.32 с аналогичной группой радиокнопок, показанной в описании компонента **RadioGroup).**

Свойство **Caption** компонента **RadioButton** содержит надпись, появляющуюся около кнопки. Как и в других кнопках, надпись может содержать символ амперсанда "&", вызывающий подчеркивание следующего символа, соответствующего клавише быстрого доступа (см. на рис. 2.32 верхнюю группу радиокнопок). Значение свойства **Alignment** определяет, с какой стороны от кнопки появится надпись (см. нижние радиокнопки на рис. 2.32): **taLeftJustify** — слева, **taRightJustify** – справа (это значение принято по умолчанию). Свойство **Checked** определяет, выбрана ли данная кнопка пользователем, или нет. Поскольку в начале выполнения приложения обычно надо, чтобы одна из кнопок группы была выбрана по умолчанию, ее свойство **Checked** надо установить в **true** в процессе проектирования.

Основные свойства

Свойство	Объявление / Описание
Action	**Classes::TBasicAction* Action** **Определяет действие, связанное с данной кнопкой**

Свойство	Объявление / Описание
Alignment	enum TAlignment { taLeft Justify, taRight Justify, taCenter }; typedef TAlignment TLeftRight; Classes::TLeftRight Alignment Определяет положение надписи (справа или слева), относящейся к радиокнопке
Caption	AnsiString Caption property Caption: TCaption; Надпись на кнопке
Checked	bool Checked Указывает, выбрана ли радиокнопка
TabOrder	typedef short TTabOrder; TTabOrder TabOrder Указывает позицию компонента в списке табуляции. Определяет порядок переключения фокуса между компонентами окна при нажатии клавиши Tab. Изначально соответствует порядку добавления компонентов на форму
TabStop	bool TabStop Определяет возможность доступа пользователя к кнопке с помощью клавиши Tab

Основные методы

Метод	**Объявление / Описание**
ExecuteAction	**bool ExecuteAction(TBasicAction* Action)** **Вызывает указанное действие Action, связанное с данной кнопкой**
Hide	**void Hide(void)** **Делает кнопку невидимой**
SetFocus	**void SetFocus(void)** **Передает фокус элементу, активизирует его**
Show	**void SetFocus(void)** **Делает видимой невидимую кнопку**

Основные события

Событие	Описание
OnClick	Наступает при щелчке на компоненте. В обработчике этого события можно анализировать свойств Checked, которое уже приняло новое значение
OnContextPopup	Наступает при вызове пользователем контекстного меню, связанного с компонентом (щелчком правой кнопкой мыши или иным способом)

RadioGroup — группа радиокнопок

Применяется для формирования группы регулярно размещенных радиокнопок, из которых в любой момент времени может быть включена только одна.

Страница библиотеки *Standard*

Класс *TRadioGroup*

Иерархия *TObiect — TPersistent — TComponent — TControl — TWincontrol — TCustomControl — TCustomGroupBox — TCustomRadioGroup*

Модуль *extctrls*

Примеры изображения

Рис. 2.33
Примеры групп радиокнопок RadioGroup

Описание

Компонент **RadioGroup** — это панель, которая содержит радиокнопки, регулярно расположенные столбцами и строками. Из радиокнопок группы может быть включена только одна. При включении какой-то кнопки все остальные выключаются. Надпись в левом верхнем углу панели определяется свойством **Caption.** Надписи кнопок и их количество определяются свойством **Items,** имеющим тип TStrings. Во время проектирования задание свойства **Items** осуществляется вызываемым из Инспектора Объектов редактором списков строк. Сколько строчек вы запишете в нем, столько и будет кнопок. Во время выполнения формировать список Items можно, используя методы и свойства класса **TStrings.** Как и в других управляющих компонентах (см., например, **Button),** надпись каждой кнопки может содержать символ амперсанда "&", вызывающий подчеркивание следующего символа, соответствующего клавише быстрого доступа. На рис. 2.33 вы можете это видеть в нижней граппе радиокнопок.

Кнопки можно разместить в несколько столбцов (не более 17), задав свойство **Columns.** По умолчанию **Columns** = 1, т.е. кнопки размещаются друг под другом. На рис. 2.33 в левой верхней группе **Columns** = 1, в остальных — **Columns = 2.**

Определить, какую из кнопок выбрал пользователь, можно по свойству **ItemIndex,** которое показывает индекс выбранной кнопки (начинаются с 0). По умолчанию **Itemlndex** = **–1,** что означает отсутствие выбранной кнопки. Если вы хотите, чтобы в момент начала выполнения приложения какая-то из кнопок была выбрана (это практически всегда необходимо), то надо установить соответствующее значение **Itemlndex** во время проектирования. Если вы используете радиокнопки не для ввода, а для отображения данных, устанавливать значение **ItemIndex** можно программно во время выполнения приложения.

Определенным недостатком **RadioGroup** является регулярное расположение кнопок, причем при расположении их в несколько столбцов расстояния между столбцами ориенируются на наиболее длинную надпись кнопки. Поэтому в ряде случаев (см. нижнюю группу на рис. 2.33) размещение кнопок получается некомпактным и некрасивым (сравните с аналогичной группой радиокнопок, показанной в описании компонента **RadioButton).** При необходимости нерегулярного расположения кнопок надо использовать компоненты **RadioButton и GroupBox.**

Основные свойства

Свойство	Объявление / Описание
Caption	AnsiString Caption Надпись в левом верхнем углу панели кнопке
Columns	int Columns Определяет количество столбцов кнопок в радиогруппе
ItemIndex	int Itemlndex Указывает, какая из радиокнопок выбрана в данный момент
Items	Classes::TStrings* Items Список радиокнопок группы

Основные методы

Никаких специальных методов в компоненте не объявлено. Методы наследуются от **TWinControl и TControl.**

События

Событие	Описание
OnClick	Событие соответствует щелчку мыши на кнопке. В обработчике этого события можно определить включенную кнопку по свойству Itemlndex

Остальные события наследуются от **TWinControl и TControl.**

RichEdit — многострочное окно редактирования в обогащенном формате

Многострочное окно редактирования текстов в обогащенном формате .rtf, позволяющее производить выбор цвета, шрифта, поиск текста и т.д.

Страница библиотеки *Win32*

Класс *TRichEdit*

Иерархия *TObiect — TPersistent— TComponent — TControl — TWinControl — TCustomEdit— TCustomMemo— TCustomRichEdit*

Модуль *stdctrls*

Пример изображения

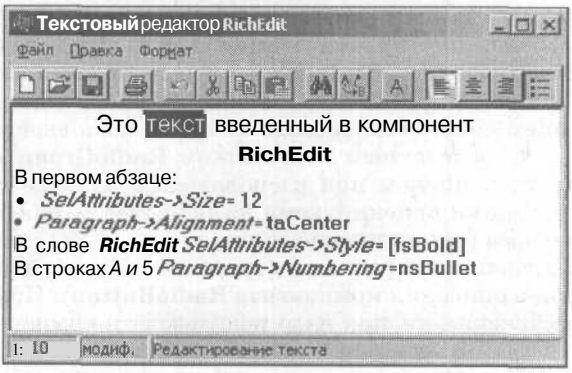


Рис. 2.34
Текстовый редактор на основе RichEdit

Описание

Компонент **RichEdit** представляет собой средство редактирования текстов, позволяющее работать с обогащенным форматом **.rtf,** т.е. выбирать различные атрибуты форматирования для разных фрагментов текста. В этом основное отличие **RichEdit** от более простого компонента **Memo,** в котором атрибуты форматирования одинаковы для всего текста.

Окно редактирования снабжено многими функциями, свойственными большинству редакторов. Например, в нем предусмотрены типичные комбинации «горячих» клавиш: Ctrl-C — копирование выделенного текста в буфер обмена Clipboard (команда Copy), Ctrl-X — вырезание выделенного текста в буфер Clipboard (команда Cut), Ctrl-V — вставка текста из буфера Clipboard в позицию курсора (команда Paste), Ctrl-Z — отмена последней команды редактирования.

Свойство **Lines,** доступное как во время проектирования, так и во время выполнения, имеет множество свойств и методов типа **TStrings,** которые обычно используются для формирования и редактирования текста (см. в гл. 1). Весь текст содержится в свойстве **Text.** Имеются также свойства только времени выполнения **SelLength, SelStart, SelText,** определяющие соответственно длину выделенного текста, позицию перед первым символом выделенного текста и сам выделенный текст. Если выделенного текста нет, то свойство **SelStart** просто определяет текущее положение курсора.

При желании изменить атрибуты вновь вводимого фрагмента текста вы можете задать свойство **SelAttributes.** Это свойство типа **TTextAttributes,** которое в свою очередь имеет подсвойства: **Color** (цвет), **Name** (имя шрифта), **Size** (размер), **Style** (стиль) и ряд других. Объекты **SelAttributes и Font** совместимы по типу. Так что значения объектов этих типов можно присваивать друг другу. Например, если приложение имеет компонент **RichEdit** и диалог выбора шрифта **FontDialogl,** то следующий код позволит пользователю менять атрибуты вновь вводимого или выделенного текста:

```
FontDialog1->Font->Assign(RichEdit1->SelAttributes);
if (FontDialog1->Execute())
  RichEdit1->SelAttributes->Assign(FontDialog1->Font);
```

Первый оператор заносит атрибуты, соответствующие текущей позиции курсора, как начальные значения в диалог выбора шрифта. А второй оператор заносит указанные пользователем атрибуты в свойство **SelAttributes** окна **RichEdit**1.

В компоненте имеется также свойство **Def Attributes,** содержащее атрибуты по умолчанию. Эти атрибуты действуют до того момента, когда изменяются атрибуты в свойстве **SelAttributes.** Но значения атрибутов в **Def Attributes** сохраняются и в любой момент эти значения могут быть методом **Assign** присвоены атрибутам свойства **SelAttributes,** чтобы вернуться к прежнему стилю.

За выравнивание, отступы и т.д. в пределах текущего абзаца отвечает свойство **Paragraph** типа **TParaAttributes.** Этот тип в свою очередь имеет ряд свойств:

Alignment	Определяет выравнивание текста
FirstIndent	Число пикселов отступа красной строки
Numbering	Управляет вставкой маркеров, как в списках
LeftIndent	Отступ в пикселах от левого поля
RightIndent	Отступ в пикселах от правого поля
TabCount	Количество позиций табуляции
Tab	Значения позиций табуляции в пикселах

Значения подсвойств свойства **Paragraph** можно задавать только в процессе выполнения приложения, например, в событии создания формы или при нажатии какой-нибудь кнопки. Значения подсвойств свойства **Paragraph** относятся к тому абзацу, в котором находится курсор. Например, каждый из следующих операторов осуществит соответственное выравнивание текущего абзаца:

```
RichEdit1->Paragraph->Alignment = taLeftJustify;  // Влево
RichEdit1->Paragraph->Alignment = taCenter;       // По центру
RichEdit1->Paragraph->Alignment = taRightJustify; // Вправо
```

Следующий оператор приведет к тому, что текущий абзац будет отображаться как список, т.е. с маркерами (см. рис. 2.34):

```
RichEdit1->Paragraph->Numbering = nsBullet;
```

Уничтожение списка в текущем абзаце осуществляется оператором

```
RichEdit1->Paragraph->Numbering = nsNone;
```

Свойство **Modified,** доступное только во время выполнения, показывает, проводилось ли редактирование текста в окне. Если вы хотите использовать это свойство, то в момент начала работы пользователя с текстом **Modified** надо установить в **false.** Тогда при последующем обращении к этому свойству можно по его значению (true или false) установить, было или не было произведено редактирование.

Свойство **WordWrap,** равное **true,** указывает на допустимость переноса длинных строк. Установка свойства **ReadOnly в true** задает текст только для чтения. Свойство **MaxLength** определяет максимальную длину вводимого текста. Если **MaxLength** = 0, то длина текста не ограничена. Свойства **WantReturns и WantTab** определяют допустимость ввода пользователем в текст символов перевода строки и табуляции.

Свойство **ScrollBars** определяет наличие полос прокрутки текста в окне. По умолчанию **ScrollBars** = **ssNone,** что означает их отсутствие. Пользователь может в этом случае перемещаться по тексту только с помощью курсора. Можно задать свойству **ScrollBars** значения **ssHorizontal, ssVertical** или **ssBoth,** что будет соответственно означать наличие горизонтальной, вертикальной или обеих полос прокрутки.

Свойство **CaretPos** указывает на запись, поле X которой содержит индекс символа в строке, перед которым расположен курсор, а поле Y — индекс строки, в которой находится курсор. Таким образом, учитывая, что индексы начинаются с О, значения **Memo1.CaretPos.Y+1 и Memo1.CaretPos.X+1** определяют соответственно номер строки и символа в ней, перед которым расположен курсор. В редакторе на приведенном выше рисунке именно эти значения использованы, чтобы отображать в строке состояния **StatusBar** позицию курсора.

Свойства **Align и Anchors** позволяют адаптировать размер окна **RichEdit** к размеру окна приложения, выбранного пользователем.

Основные свойства

Свойство	Объявление / Описание
Align	**enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom};** **TAlign Align** Определяет способ выравнивания компонента в контейнере (родительском компоненте)
Alignment	**enum TAlignment { taLeft Justify, taRight Justify, taCenter };** **Classes::TAlignment Alignment** Управляет выравниванием текста: **taLeftJustify** — влево, **taRightJustify** — вправо, **taCenter** — по центру. Значение по умолчанию — влево

Свойство	Объявление / **Описание**
Anchors	enum TAnchorKind { akLeft, akTop, akRight, **akBottom** }; typedef Set<T Anchor Kind, akLeft, akBottom> **TAnchors;** TAnchors Anchors Определяет привязку данного компонента к родительскому при изменении размеров последнего
CanUndo	**bool** CanUndo Указывает, содержит ли компонент изменения, которые можно отменить. Доступ только для чтения
CaretPos	TPoint CaretPos Указывает строку и символ расположения курсора (см. в приведенном ранее описании компонента). Доступ только для чтения
DefAttributes	**TTextAttributes* DefAttributes** Атрибуты форматирования, используемые по умолчанию
Default Converter	class PACKAGE TMetaClass; typedef TMetaClass* TClass; TMetaClass* DefaultConverter Определяет класс объекта TConversion, который используется для преобразования исходного формата файла во внутренний формат компонента при работе с файлами, не имеющими зарегистрированного расширения
Font	**Graphics::TFont*** Font Определяет атрибуты шрифта
HideScroll Bars	bool HideScrollBars Определяет, должны ли полосы прокрутки делаться невидимыми, когда в них нет необходимости, т.е. когда весь текст и без них виден в окне (по умолчанию true)
HideSelection	bool HideSelection Определяет, сохраняется ли выделение текста видимым при переходе фокуса к другому компоненту
Lines	Classes::TStrings* Lines Строки текста — объект типа TStrings
MaxLength	int MaxLength Указывает максимальное количество символов, которое пользователь может вводить в компонент. При значении 0 длина текста неограничена
Modified	bool Modified Указывает, редактировался ли пользователем текст в компоненте
PageRect	property PageRect: TRect; Определяет в твипсах (твипс — 1/20 пункта, пункт — 1/72 дюйма) размеры логической страницы при печати содержимого компонента.
Paragraph	TParaAttributes* Paragraph Объект, определяющий форматирование текущего абзаца

Свойство	Объявление / Описание
PlainText	**bool** PlainText При обмене с файлом определяет, является ли текст простым, или соответствует формату .rtf
PopupMenu	**Menus::TPopupMenu*** PopupMenu Определяет всплывающее меню, связанное с данным компонентом
Readonly	bool ReadOnly Указывает, может ли пользователь изменять текст в компоненте
SelAttributes	TTextAttributes* SelAttributes Описывает характеристики выделенного текста
SelLength	int SelLength Определяет количество выделенных символов в тексте
SelStart	int SelStart Указывает позицию первого выделенного символа в тексте или, если выделения нет, то позицию курсора
SelText	**AnsiString** SelText Текст, выделенный в окне
Text	AnsiString Text Текст окна в виде одной строки
WantReturns	bool WantReturns Указывает, можно ли вставить в текст символы возврата каретки
WantTabs	bool WantTabs Указывает, можно ли вставить в текст символы табуляции
WordWrap	bool Wordwrap Указывает, переносится ли текст на новую строку, если он превышает ширину компонента

Основные методы

Метод	Объявление / Описание
Clear	void **Clear(void)** Удаляет текст из окна
Clear Selection	void ClearSelection(void) Удаляет текст, выделенный в окне
ClearUndo	void **ClearUndo(void)** Очищает буфер отмены команд редактирования, так что никакие изменения в тексте после этого не могут быть отменены
CopyTo Clipboard	voidCopyToClipboard(void) Копирует выделенный текст в компоненте редактирования в Clipboard в формате CF_TEXT

Метод	Объявление / Описание
CutTo Clipboard	void CutToClipboard(void) Переносит выделенный текст в Clipboard в формате CF_TEXT и уничтожает его в окне
FindText	enum TSearchType { stWholeWord, stMatchCase }; typedef Set<TSearchType, stWholeWord, stMatchCase> TSearchTypes; int FindText(const AnsiString SearchStr, int StartPos, int Length, TSearchTypes Options) Ищет в тексте заданный фрагмент SearchStr, начиная с позиции StartPos на протяжении Length символов. Может искать фрагмент как целое слово (опция stWholeWord) и с учетом (опция stMatchCase) или без учета регистра
PasteFrom Clipboard	void PasteFromClipboard(void) Переносит в окно в позицию SelStart текст из буфера Clipboard
Perform	int Perform(unsigned Msg, int WParam, int LParam) Передает окну сообщение Windows Msg с параметрами WParam и LParam
Print	void Print(const AnsiString Caption) Форматирует и печатает текст компонента с заголовком Caption
SelectAll	void SelectAll(void) Выделяет весь текст в окне редактирования
Undo	void Undo(void) Отменяет все изменения, хранившиеся в буфере отмены результатов редактирования с момента последнего вызова ClearUndo

Основные события

Событие	Описание
OnChange	Наступает, когда текст в окне может быть изменился. Свойство Modified показывает, действительно ли произошло изменение
OnKeyDown	Событие наступает при нажатии пользователем любой клавиши. В обработчике можно распознать нажатую клавишу
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить его ввод
OnKevUp	Событие наступает при отпускании пользователем любой клавиши. В обработчике можно распознать отпускаемую клавишу
OnProtectChange	Событие происходит при попытке пользователя изменить текст, маркированный как защищенный
OnResizeRequest	Событие происходит, когда текст становится либо меньше, либо больше размеров окна компонентов

Событие	Описание
OnSaveClipboard	Событие происходит при намерении ликвидировать окно компонента в условиях, когда компонент копирует текст в Clipboard
OnSelectionChange	Событие происходит при изменении выделенного текста

SaveDialog — диалог сохранения файла

Невизуальный компонент вызова диалога сохранения файла.

См. раздел «OpenDialog, OpenPictureDialog, SaveDialog, SavePictureDialog – диалоги работы с файлами».

SavePictureDialog — диалог сохранения файла изображения

Невизуальный компонент вызова диалога сохранения файла изображения.

См. раздел «OpenDialog, OpenPictureDialog, SaveDialog, SavePictureDialog – диалоги работы с файлами».

Session — сеанс связи с базами данных

Невизуальный компонент, осуществляет общее управление связыванием приложения с базами данных.

Страница библиотеки *Data Access*

Класс *TSession*

Иерархия *TObject — TPersistent — TComponent*

Модуль *Dbtables*

Описание

Компонент **Session** осуществляют общее управление связыванием приложения с базами данных. Обычно пользователю не приходится заботиться о компоненте Session, поскольку C++Builder автоматически генерирует объект **Session** в каждом приложении, работающем с базами данных. На этот объект можно ссылаться через глобальную переменную Session.

Компонент **Session** имеет много полезных методов и позволяет легко работать с BDE. Например, метод **GetAliasNames** позволяет получить список всех псевдонимов баз данных, зарегистрированных в BDE. Так что оператор

```
Session->GetAliasNames(ComboBox1->Items);
```

заполнит выпадающий список **ComboBox1** перечнем псевдонимов баз данных, зарегистрированных в BDE. Пользователь может затем выбрать любую базу данных из этого списка. В списке **ComboBoxl** можно включить в обработчик события **OnChange** оператор

```
Session->GetTableNames(ComboBox1->Text,"",true,false,
                       ComboBox2->Items);
```

Тогда при выборе пользователем базы данных в списке **ComboBoxl** список **ComboBox2** загрузится перечнем таблиц базы данных, выбранной в **ComboBoxl.** Это делается методом **GetTableNames.** В результате пользователь сможет, производя соответствующий выбор в **ComboBoxl и ComboBox2,** соединяться с любой таблицей любой базы данных.

Вводить явным образом компоненты **Session** приходится только в многозадачных приложениях, в которых предусмотрено несколько параллельных процессов со своими потоками обмена информацией с базой данных. В таких многопоточных

приложениях обычно вводится явным образом по одному компоненту Session на каждый поток, чтобы исключить влияние потоков друг на друга.

При явном вводе компонента Session в приложение следует установить его свойство SessionName — имя сеанса сетевого соединения, задав в нем произвольный идентификатор. После этого в выпадающих списках свойств SessionName компонентов типа **Database,** Table, Query и т.п. появится введенное вами имя. Выбор соответствующего имени сеанса сетевого соединения из этих списков свяжет эти компоненты с соответствующим компонентом Session.

Основные свойства

Свойство	Объявление / Описание
Active	**bool** Active Определяет, активен сеанс сетевого соединения, или нет
AutoSession Name	bool **AutoSessionName** Указывает, является ли компонент автоматически сгенерированным с уникальным именем
ConfigMode	**enum** TConfigModes **{cfmVirtual, cfmPersistent, cfmSession};** TConfigMode ConfigMode Указывает, как BDE должна управлять псевдонимами
DatabaseCount	int DatabaseCount Число компонентов баз данных Database, связанных с данным сеансом и содержащихся в Databases
Databases	**TDatabase*** Databases[int Index] Индексированный массив компонентов баз данных Database, связанных с данным сеансом
Handle	Bde::hDBISes Handle Дескриптор BDE данного сеанса
KeepConnections	bool KeepConnections Указывает, должны ли поддерживаться соединения даже при неактивных наборах данных
Locale	void * Locale Идентификатор локализации BDE сеанса
NetFileDir	AnsiString NetFileDir Определяет каталог, содержащий файл управления сетью BDE — PDOXUSRS.NET
PrivateDir	AnsiString PrivateDir Определяет каталог временных файлов, генерируемых BDE для компонентов баз данных Database, связанных с данным компонентом
SessionName	AnsiString SessionName Имя сеанса, используемое связанными с Session компонентами. Если AutoSessionName = true, то SessionName явно задаваться не может

Свойство	Объявление / Описание
SQLHourGlass	bool SQLHourGlass Определяет, должен ли курсор приобретать вид песочных часов SQL во время операций BDE
TraceFlags	enum TTraceFlag {tfQPrepare, tfQExecute, tfError, tfStmt, tfConnect, tfTransact, tfBlob, tfMisc, tfVendor, tfDataIn, tfDataOut}; typedef Set<TTraceFlag, tfQPrepare, tfDataOut> TTraceFlags; TTraceFlags TraceFlags Определяет операции с базой данных, отображаемые во время выполнения в SQL Monitor

Основные методы

Метод	Объявление / Описание
AddAlias	void AddAlias(const AnsiString Name, const AnsiString Driver, Classes::TStrings* List) Добавляет указанный псевдоним BDE Name с драйвером Driver и параметрами List в сеанс работы с сервером SQL
AddDriver	void AddDriver(const AnsiString Name, Classes::TStrings* List) Добавляет указанный драйвер BDE Name с параметрами List в сеанс работы с сервером SQL
AddPassword	void AddPassword(const AnsiString Password) Добавляет в текущий сеанс пароль доступа к таблицам Paradox
AddStandard Alias	void AddStandardAlias(const AnsiString Name, const AnsiString Path, const AnsiString DefaultDriver) Добавляет в сеанс псевдоним типа Standard для связи с таблицами Paradox, dBASE, ASCII
Close	void Close(void) Закрывает соединения всех баз данных и закрывает сеанс
CloseDatabase	void CloseDatabase(TDatabase* Database) Закрывает соединения всех баз данных
DeleteAlias	void DeleteAlias(const AnsiString Name) Удаляет указанный псевдоним BDE из сеанса
DeleteDriver	void DeleteDriver(const AnsiString Name) Удаляет указанный драйвер BDE из сеанса
DropConnections	voidDropConnections(void) Уничтожает все неактивные временные компоненты баз данных Database, связанные с сеансом
FindDatabase	TDatabase* FindDatabase(const AnsiString DatabaseName) Возвращает компонент Database, связанный с указанной базой данных

Метод	Объявление / Описание
GetAliasDriver Name	AnsiString GetAliasDriverName(const AnsiString AliasName) Возвращает имя драйвера, используемого указанным псевдонимом BDE
GetAliasNames	void GetAliasNames(Classes::TStrings* List) Возвращает список всех псевдонимов баз данных, зарегистрированных в BDE
GetAliasParams	void GetAliasParams(const AnsiString AliasName, Classes::TStrings* List) Обеспечивает доступ к параметрам, связанным с указанным псевдонимом BDE
GetConfig Params	void GetConfigParams(const AnsiString Path, const AnsiString Section, Classes::TStrings* List) Обеспечивает доступ к параметрам конфигурации BDE
GetDatabase Names	void GetDatabaseNames(Classes::TStrings* List) Возвращает список всех псевдонимов баз данных, зарегистрированных в BDE, и всех компонентов баз данных сеанса
GetDriverNames	void GetDriverNames(Classes::TStrings* List) Возвращает список всех доступных драйверов BDE
GetDriverParams	void GetDriverParams(const AnsiString DriverName, Classes::TStrings* List) Возвращает список параметров указанного драйвера BDE
GetPassword	bool GetPassword(void) Генерирует событие OnPassword — запрос пароля
GetStoredProc Names	void GetStoredProcNames(const AnsiString DatabaseName, Classes::TStrings* List) Возвращает список имен всех хранимых на SQL сервере процедур, связанных с указанным компонентом базы данных Database
GetTableNames	void GetTableNames(const AnsiString DatabaseName, const AnsiString Pattern, bool Extensions, bool SystemTables, Classes::TStrings* List) Возвращает список имен всех таблиц, связанных с указанным компонентом базы данных Database
IsAlias	bool IsAlias(const AnsiString Name) Определяет, является ли указанная строка псевдонимом BDE
ModifyAlias	void ModifyAlias(AnsiString Name, Classes::TStrings* List) Добавляет или изменяет параметры List псевдонима Name BDE
ModifyDriver	void ModifyDriver(AnsiString Name, Classes::TStrings* List) Добавляет или изменяет параметры List драйвера Name BDE
Open	void Open(void) Открывает сеанс и делает его текущим

Метод	**Объявление** / Описание
OpenDatabase	TDatabase* **OpenDatabase(const** AnsiString DatabaseName) Открывает существующую базу данных, создает для нее временный компонент Database и открывает его
RemoveAll Passwords	void **RemoveAllPasswords(void)** Удаляет все ранее добавленные в текущий сеанс пароли доступа к таблицам Paradox
RemovePassword	void **RemovePassword(const** AnsiString Password) Удаляет добавленный в текущем сеансе пароль доступа к таблицам Paradox
SaveConfigFile	void **SaveConfigFile(void)** Переписывает текущую информацию BDE из памяти и сохраняет ее в файле конфигурации BDE

События

Событие	Описание
OnUpdateData	Наступает, когда намечается обновление текущей записи
OnPassword	Наступает при первой попытке приложения открыть таблицу Paradox, если BDE фиксирует отсутствие соответствующих прав доступа
OnStartup	Наступает в момент активизации сеанса

SpeedButton — кнопка с пиктограммой и фиксацией

Используется для создания инструментальных панелей и в других случаях, когда требуется кнопка с фиксацией нажатого состояния.

Страница библиотеки *Additional*

Класс *TSpeedButton*

Иерархия *TObiect — TPersistent — TComponent — TControl — TGraphicControl*

Модуль *buttons*

Примеры изображения

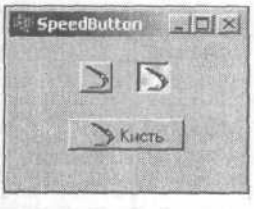


Рис. 2.35
Примеры кнопок SpeedButton

Описание

Компонент **SpeedButton** используется для кнопок с фиксацией. Имеет возможность отображения пиктограмм и может использоваться как обычная управляющая кнопка или как кнопка с фиксацией нажатого состояния. Обычно используется в качестве быстрых кнопок, дублирующих различные команды меню, и в инструментальных панелях, в которых требуется фиксация нажатого состояния.

У кнопок **SpeedButton,** как и у других кнопок, имеется свойство **Caption** – надпись, но в этих кнопках оно обычно оставляется пустым, так как вместо надписи используется пиктограмма. Изображение на кнопке задается свойством **Glyph** точно так же, как для кнопок **BitBtn.** И точно так же свойство **NumGlyphs** определяет число используемых пиктограмм, свойства **Layout и Margin** определяют расположение изображения, а свойство **Spacing** — расстояние между изображением и надписью (если, конечно, вы все-таки хотите использовать надпись на кнопке).

Особенностью кнопок **SpeedButton** являются свойства **GroupIndex** (индекс группы), **AllowAllUp** (разрешение отжатого состояния всех кнопок группы) и **Down** (исходное состояние — нажатое). Если **GroupIndex** = 0, то кнопка ведет себя так же, как **Button и BitBtn.** При нажатии пользователем кнопки она погружается, а при отпускании возвращается в нормальное состояние. В этом случае свойства **AllowAHUp и Down** не влияют на поведение кнопки.

Если **GroupIndex > 0 и AllowAHUp = true,** то кнопка при щелчке пользователя на ней погружается и остается в нажатом состоянии. При повторном щелчке пользователя на кнопке она освобождается и переходит в нормальное состояние (именно для того, чтобы освобождение кнопки состоялось, необходимо задать **AllowAHUp = true).** Если свойство **Down** во время проектирования установлено равным **true,** то исходное состояние кнопки — нажатое.

Если есть несколько кнопок, имеющих одинаковое ненулевое значение **GroupIndex,** то они образуют группу взаимосвязанных кнопок, из которых нажатой может быть только одна. Если одна кнопка находится в нажатом состоянии и пользователь щелкает на другой, то первая кнопка освобождается, а вторая фиксируется в нажатом состоянии. Поведение нажатой кнопки при щелчке на ней зависти от значения свойства **AllowAHUp.** Если оно равно **true,** то кнопка освободится, поскольку в этом случае возможно состояние, когда все кнопки группы отжаты. Если же **AllowAHUp** равно **false,** то щелчок на нажатой кнопке не приведет к изменению вида кнопки. Впрочем, и в этом случае, как и при любом щелчке на кнопке, возникает событие **OnClick,** которое может быть обработано.

Состояние кнопки во время выполнения можно определить по значению свойства Down: если значение равно true, то кнопка нажата. Во время события **OnClick** значение **Down** уже равно тому состоянию, которое примет кнопка в результате щелчка на ней.

Основные свойства

Свойство	Объявление / Описание
Action	Classes::TBasicAction* Action Определяет действие, связанное с данной кнопкой
AllowAHUp	bool AllowAHUp Определяет, могут ли все кнопки с одинаковым значением GroupIndex быть одновременно отжатыми
Caption	AnsiString Caption Надпись на кнопке
Down	bool Down Указывает, нажата кнопка или отжата
Glyph	Graphics::TBitmap* Glyph Определяет битовую матрицу, которая появляется на кнопке

Свойство	Объявление / Описание
GroupIndex	int GroupIndex Определяет произвольный заданный номер группы кнопок. Позволяет кнопке работать в составе группы
Layout	enum TButtonLayout {blGlyphLeft, **blGlyphRight,** blGlyphTop, **blGlyphBottom};** TButtonLayout Layout Определяет, к какому краю кнопки смещается изображение
Margin	int Margin Определяет количество пикселов между краем изображения и краем кнопки, указанным свойством Layout. При -1 изображение и надпись размещаются в центре кнопки
NumGliphs	typedef Shortint TNumGlyphs; TNumGlyphs NumGlyphs Указывает количество изображений в свойстве Glyph
Spacing	int Spacing Число пикселов, разделяющих изображение и надпись на поверхности кнопки

Основные методы

Метод	Объявление / Описание
Click	void Click(void) Имитирует щелчок мышью, как если бы пользователь щелкнул на кнопке
Execute Action	**bool** ExecuteAction(TBasicAction* Action) Вызывает указанное действие Action, связанное с данной кнопкой

События

Событие	Описание
OnClick	Соответствует щелчку мыши на кнопке или нажатию клавиш быстрого доступа
OnDblClick	Наступает при двойном щелчке
OnMouseDown	Событие при нажатии кнопки мыши над кнопкой
OnMouseMove	Событие при перемещении указателя мыши над кнопкой
OnMouseUD	Событие при отпускании нажатой кнопки мыши над кнопкой

StaticText — метка с бордюром

Метка, используемая для отображения текста.

Страница библиотеки *Additional*

Класс *TStaticText*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomStaticText*

Модуль *stdctrls*

Примеры изображения

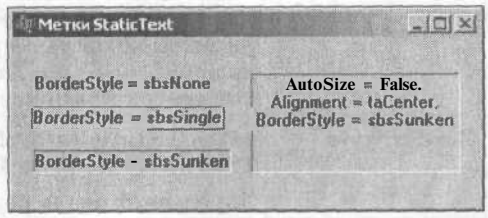


Рис. 2.36
Примеры StatText. Надписи в метках поясняют установленные в них значения свойств

Описание

StaticText — оконный компонент, отображающий однострочный текст на форме, не подлежащий редактированию. **StaticText** функционирует подобно метке **Label**, за исключением того, что он потомок **TWinControl** и поэтому имеет вид и многие свойства окна, включая возможность оформления более красивого обрамления надписи, чем в **Label.**

Текст метки задается свойством **Caption.** Шрифт надписи определяется свойством **Font,** цвет фона — свойством **Color,** а цвет надписи — подсвойством **Color** свойства **Font.** Размер меток **StaticText** определяется свойством **AutoSize.** Если это свойство установлено в **true,** то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же **AutoSize** равно **false,** то выравнивание текста внутри компонента определяется свойством **Alignment,** которое позволяет выравнивать текст по левому краю, правому краю или центру клиентской области метки.

В метке **StaticText** автоматически осуществляется перенос длинного текста по словам, если значение **AutoSize** установлено в **false** и размер компонента достаточен для размещения нескольких строк (см. рис. 2.36). Для того чтобы в **StaticText** осуществлялся перенос при изменении пользователем размеров окна, надо осуществлять перерисовку компонента методом **Repaint** в обработчике события формы **OnResize.**

StaticText имеет свойство **BorderStyle,** определяющее рамку текста — бордюр (см. рис. 2.36). При стиле **BorderStyle** = **sbsNone** метка **StaticText** по виду не отличается от метки **Label.** Вероятно, если уж использовать бордюр, то наиболее приятный стиль **BorderStyle = sbsSunken.**

Метки могут обеспечить клавишами ускоренного доступа элементы, в которых такие клавиши не предусмотрены, например, окна редактирования **Edit.** Компонент, на который должен переключаться фокус при нажатии клавиш ускоренного доступа, задается свойством **FocusControl.** Чтобы доступ осуществлялся, надо установить в **true** свойство **ShowAccelChar.** В надписи **Caption** перед соответствующим символом надо поставить символ амперсанда — "&". Следующий за амперсандом символ будет отображаться в надписи подчеркнутым и будет являться символом быстрого доступа: при выполнении приложения нажатие клавиши Alt + клавиши данного символа будет эквивалентно переключению фокуса на компонент, указанный свойством **FocusControl.**

Основные свойства

Свойство	Объявление / Описание
Alignment	**enum TAlignment { taLeftJustify, taRightJustify, taCenter } ; Classes::TAlignment Alignment** **Управляет горизонтальным выравниванием текста в пределах метки, если свойство AutoSize установлено в false: taLeftJustify — влево, taRightJustify — вправо, taCenter — по центру**

Свойство	Объявление / Описание
AutoSize	**bool** AutoSize Если это свойство установлено в **true**, то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же AutoSize равно false, то выравнивание текста внутри компонента определяется свойством Alignment
BorderStyle	enum TStaticBorderStyle { sbsNone, sbsSingle, sbsSunken }; TStaticBorderStyle BorderStyle Определяет стиль бордюра надписи (см. рис. 2.36)
Brush	**Graphics::TBrush*** Brush Кисть — определяет цвет и стиль заполнения фона окна. Свойство только для чтения
Caption	**AnsiString** Caption Строка текста, отображаемая меткой. Может содержать символ ускоренного доступа к элементу, указанному свойством FocusControl
Color	**Graphics::TColor** Color Определяет цвет фона метки
FocusControl	**Controls::TWinControl*** FocusControl Определяет оконный компонент, получающий фокус при нажатии пользователем клавиши быстрого доступа метки (см. пояснения выше *в* описании StaticText)
Font	**Graphics::TFont*** Font Определяет атрибуты шрифта
ParentColor	bool ParentColor Определяет (при значении true), что для фона метки будет заимствован цвет родительского компонента. В этом случае фон метки не заметен и видна только ее надпись — Caption
ShowAccelChar	bool ShowAccelChar Определяет, как амперсанд отображается в тексте метки (см. пояснения выше в описании StaticText)

Основные методы

Никаких специальных методов в компоненте не объявлено. Метка наследует множество методов от своих предшественников, в основном, от базового класса **TControl.**

Основные события

Никаких специальных событий в компоненте не объявлено. Метка наследует множество событий от класса **TControl.**

StatusBar — полоса состояния

Представляет собой ряд панелей, отображающих полосу состояния в стиле Windows.

Страница библиотеки *Win32*

Класс *TStatusBar*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl*

Модуль *comctrls*

Пример изображения

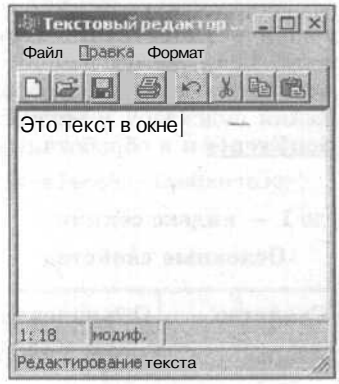


Рис. 2.37
Пример приложения с двумя полосами состояния: верхняя секционирована, нижняя - простая

Описание

Компонент **StatusBar** размещается обычно внизу окна и представляет собой полосу состояния из одной или нескольких панелей. В этих панелях пользователю сообщается какая-то текущая информация о режиме работы, даются подсказки и т.п.

Свойство **SimplePanel** компонента **StatusBar** определяет, включает ли полоса состояния одну или множество панелей. Если **SimplePanel = true,** то вся полоса состояния представляет собой единственную панель, текст которой задается свойством **SimpleText** (на рис. 2.37 такой стиль имеет нижняя полоса). Если же **SimplePanel = false,** то полоса состояния является набором панелей, задаваемых свойством **Panels.**

Каждая панель полосы состояния является объектом типа **StatusPanels.** Свойства панелей вы можете задавать специальным редактором наборов, который можно вызвать тремя способами: из Инспектора Объектов кнопкой с многоточием около свойства **Panels,** двойным щелчком на компоненте **StatusBar** или из контекстного меню, выбрав команду Panels Editor. В окне редактора вы можете перемещаться по панелям, добавлять новые или уничтожать существующие. При перемещении по панелям в окне Инспектора Объектов вы будете видеть их свойства.

Основное свойство каждой панели — **Text,** в который заносится отображаемый в панели текст. Другое существенное свойство панели -- **Width** (ширина). Свойство **Style** определяет стиль панели: **psText** — текстовая, **psOwnerDraw** — предназначена для изображений на канве.

Программный доступ к текстам отдельных панелей можно осуществлять через свойство **Panels** и его индексированное подсвойство **Items.** Например, оператор:

```
StatusBar1->Panels->Items[0]->Text = "текст 1";
```

напечатает текст "текст 1" в первой панели.

Количество панелей полосы состояния можно определить из подсвойства **Count** свойства **Panels.** Например, следующий оператор очищает тексты всех панелей:

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

Свойство **SizeGrip** определяет наличие в нижнем правом углу полосы состояния захвата, позволяющего пользователю изменять размер полосы. Если при этом полоса выровнена по нижнему краю формы **(Align = alBottom),** то одновременно

будет изменяться и размер окна приложения (на рис. 2.37 в верхней полосе **SizeGrip = false,** а в нижней — **true**).

Свойство **AutoHint** обеспечивает автоматическое отображение в строке состояния подсказок — вторых частей свойства **Hint** тех компонентов, над которыми перемещается курсор мыши (строка "Редактирование текста" на рис. 2.37). Если в панели **SimpleText = true,** то подсказки отобразятся в единственной секции панели. Если же вы используете многосекционную панель состояния, то свойство **AutoHint** обеспечит отображение подсказок только в первой секции. Для отображения подсказок в другой секции надо перенести на форму компонент **ApplicationEvents** и в обработчик его события **OnHint** компонента вставить оператор

```
StatusBar1->Panels->Items[I]->Text = Application->Hint;
```

где I — индекс секции.

Основные свойства

Свойство	Объявление / Описание
Align	enum TAlign {alNone, **alTop,** alBottom, alLeft, alRight, alCIient, **alCustom**}; TAlign Align Определяет способ выравнивания полосы состояния на форме. По умолчанию alBottom
AutoHint	**bool** AutoHint Определяет автоматическое отображение в первой панели полосы состояния вторых частей подсказок Hint тех компонентов, над которыми перемещается курсор мыши
Canvas	**Graphics::TCanvas*** Canvas Канва, позволяющая рисовать на панели в обработчике события **OnDrawPanel**
Panels	TStatusPanels* Panels Собрание панелей — объектов типа TStatusPanel
SimplePanel	bool SimplePanel Определяет, отображает ли полоса состояния одну (при значении **true**) или несколько панелей
SimpleText	**AnsiString** SimpleText Содержит строку, которая отображается в полосе состояния при SimplePanel = true
SizeGrip	bool SizeGrip Определяет наличие в нижнем правом углу полосы состояния захвата, позволяющего пользователю изменять размер полосы, а при Align = alBottom — и размер формы
UseSystem Font	bool **UseSystemFont** Определяет, используется ли в полосе состояния системный шрифт

Основные методы

Никаких специальных методов в компоненте не объявлено. Компонент наследует множество методов от класса **TWinControl.**

Основные события

Событие	Описание
OnDrawPanel	Событие происходит при необходимости перерисовать панель состояния, например при изменении пользователем ее размеров. Наступает только если стиль панели **psOwnerDraw**
OnHint	Наступает перед тем, как в полосе состояния должна отображаться подсказка. Обработчик отменяет действие свойства **AutoHint** и должен сам обеспечивать отображение подсказки

Table — набор данных, связанный с одной таблицей

Невизуальный компонент набора данных, связанный с одной таблицей.

Страница библиотеки *BDE,* в версиях, младше C++Builder 6 — *Data Access*

Класс *TTable*

Иерархия *TObject — TPersistent — TComponent — TDataSet — TBDEDataSet — TDBDataSet*

Модуль *Dbtables*

Описание

Компонент **Table** обеспечивает прямой доступ к каждой записи и полю в одной указанной таблице базы данных. Компонент может также работать с подмножеством записей внутри таблицы базы данных. Во время проектирования вы можете создавать, удалять, модифицировать, или переименовывать таблицу базы данных, связанную с **Table.**

Большинство свойств, методов и событий **Table** наследует от классов-предшественников, сведения о которых приводятся в гл. 1.

Для связи **Table** с необходимой таблицей базы данных служат два свойствае: **DatabaseName и TableName.** Прежде всего, надо установить свойство **DatabaseName.** В выпадающем списке этого свойства в Инспекторе Объектов вы можете видеть все доступные BDE псевдонимы баз данных.

После того как указана база данных, можно устанавливать значение свойства **TableName.** В выпадающем списке этого свойства перечислены таблицы, доступные в выбранной базе данных.

Соединение с выбранной таблицей базы данных осуществляется свойством **Active.** По умолчанию оно равно **false.** Если установить его в **true** во время проектирования или программно во время выполнения, то компонент соединится с базой данных.

Свойство **Exclusive** компонента **Table** определяет доступ к используемой таблице при одновременном обращении к ней нескольких приложений (например, при работе в сети или в многозадачном режиме). Если задать значение этого свойства **true,** то таблица будет закрыта для других приложений. Свойство можно изменять только при **Active = false.**

Свойства **IndexName и IndexFieldName** позволяют управлять упорядочиванием записей в наборе. Свойство **IndexName** упорядочивает представление записей в соответствии с имеющимися в базе данных индексами. Выпадающий список индексов можно открыть, нажав в Инспекторе Объектов кнопку с многоточием рядом со свойством **IndexName.** Альтернативный вариант индексации предоставляет свойство **IndexFieldName.** В его выпадающем списке просто перечислены предусмотренные в индексах комбинации полей, и вы можете выбрать необходимую, если забыли, что обозначают имена индексов.

Объекты полей, экспонируемых компонентом **Table,** могут создаваться автоматически. Но тогда их характеристики (надписи, число отводимых под них символов и т.п.) будут приняты по умолчанию и вряд ли устроят пользователя. Можно создавать и редактировать объекты полей с помощью специального Редактора Полей. Вызвать его проще всего двойным щелчком на компоненте **Table.** Вы увидите поле этого редактора (см. рис. 2.38) сначала пустое. Щелкните на нем правой кнопкой мыши и из всплывающего меню выберите раздел Add fields (добавить поля). Вы увидите окно, в котором содержится список всех полей таблицы. Выберите из него курсором мыши интересующие вас поля, щелкните на OK и вы вернетесь к основному окну Редактору Полей, но в нем уже будет содержаться список добавленных полей. Имейте в виду, что только к тем полям, которые вы добавите, вы сможете в дальнейшем обращаться.

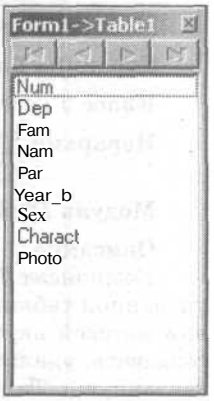


Рис. 2.38
Окно Редактора Полей набора данных

Выделив в списке окна Редактора Полей какое-то поле, вы можете увидеть его свойства в Инспекторе Объектов. Каждое поле является объектом класса, производного от **TField** — базового класса полей (см. подробнее в гл. 1) и обладает множеством свойств. Рассмотрим основные из них, которые чаще всего необходимо задавать.

Свойство Alignment определяет выравнивание отображаемого текста внутри колонки таблицы: влево, вправо или по центру.

Свойство **DisplayLabel** соответствует заголовку столбца данного поля. Заголовок можно писать по-русски. Свойство **DisplayWidth** определяет ширину колонки — число символов, отводимых под заголовок.

Свойства **EditMask** для строк и **EditFormat** для чисел определяют форматы отображения данных.

Для логических полей очень важным является свойство **Display Values.** Это свойство определяет, какие значения должны отображаться, если поле имеет значение **true** или **false.** Отображаемые значения разделяются точкой с запятой. Первым пишется значение, соответствующее **true.** Например, если вы имеете булево поле Sex (пол) то в свойстве **DisplayValues** можете написать: "м;ж" или "мужской;женский".

Свойство **Readonly,** установленное в **true,** запрещает пользователю вводить в данное поле значения. Свойство **Visible** определяет, будет ли видно пользователю соответствующее поле.

Имеется еще множество свойств, методов и событий как объектов полей — наследников **TField.** так и базовых классов наборов данных, которым наследует **Table: TDataSet. TBDEDataSet. TDBDataSet.** Вы найдете описания базовых классов в гл. 1, а подробные описания свойств, методов и событий — в гл. 3, 4, 5. Но детальное рассмотрение методики работы с базами данных далеко выходит за рамки этой книги (см. источники [1] и [3]).

Основные свойства

Свойство	Объявление / Описание
Active	**bool** Active Определяет, открыта база данных, или нет
Database Name	AnsiString DatabaseName Указывает имя базы данных, с которой связан набор данных
Exclusive	bool Exclusive Указывает, открывается ли таблица с блокировкой доступа со стороны других приложений
Exists	bool Exists Показывает, имеется ли таблица данных, связанная с Table
IndexDefs	Db::TIndexDefs* IndexDefs Содержит информацию об индексах таблицы
IndexField Names	**System::AnsiString IndexFieldNames** Список полей, используемых для индексации таблицы
IndexFields	**Db::TField*** IndexFields[int Index] Индексированный список объектов полей текущего индекса
IndexFiles	Classes::TStrings* IndexFiles Список индексных файлов таблицы dBASE
IndexName	AnsiString IndexName Имя используемого вторичного индекса таблицы
MasterFields	AnsiString MasterFields Во вспомогательной таблице определяет ключевые поля головной таблицы, используемые для связи со вспомогательной таблицей
MasterSource	Db::TDataSource* MasterSource Во вспомогательной таблице определяет источник данных головной таблицы
Readonly	bool Readonly Устанавливает данные только для чтения (при значении true)
TableName	AnsiString TableName Указывает имя таблицы базы данных

Имеется также множество наследуемых свойств, переопределенных в Table.

Основные методы

Метод	Объявление / Описание
AddIndex	**enum TIndexOption { ixPrimary, ixUnique, ixDescending, ixCaseInsensitive, ixExpression, ixNonMaintained };** **typedefSet<TIndexOption, ixPrimary, ixNonMaintained> TIndexOptions;** **void AddIndex(const AnsiString Name, const AnsiString Fields, Db::TIndexOptions Options, const AnsiString DescFields = "")** Создает новый индекс таблицы

Метод	Объявление / Описание
Append	void Append(void) Добавляет новую пустую запись в конец набора данных
Append Record	void AppendRecord(const System::TVarRec * Values, const int Values_Size) Добавляет в набор данных новую запись, заполняет ее и пересылает в базу данных
ApplyRange	void ApplyRange(void) Применяет диапазон значений, установленный методами SetRangeStart и SetRangeEnd
AppIvUpdates	void ApplyUpdates(void) Записывает кэшированные изменения в базу данных
BatchMove	enum TBatchMode {batAppend, batUpdate, batAppendUpdate, batDelete, batCopy}; int BatchMove(TBDEDataSet* ASource, TBatchMode AMode) Перемещает записи из набора данных Asource в таблицу данного компонента. Параметр AMode типа TBatchMode определяет режим перемещения: добавление, замена, копирование. Метод возвращает число записей, с которыми проводилась работа. При переносе полей с русскими текстами возникают проблемы
Cancel	void Cancel(void) Отменяет результаты редактирования
CancelRange	void CancelRange(void) Снимает ограничения диапазона, введенные методами SetRangeStart, SetRangeEnd, SetRange
Cancel Updates	void CancelUpdates(void) Отменяет все кэшированные изменения и восстанавливает исходное состояние набора данных
Close	void Close(void) Закрывает набор данных
Close Database	void CloseDatabase(TDatabase* Database) Закрывает соединение с базой данных Database
Commit Updates	void CommitUpdates(void) Очищает буфер кэшированных изменений
CreateTable	void CreateTable(void) Создает новую таблицу базы данных
Delete	void Delete(void) Удаляет активную запись и позиционирует курсор на следующую запись
DeleteTable	void DeleteTable(void) Удаляет таблицу базы данных
Edit	void Edit(void) Переводит набор данных в режим редактирования

Метод	Объявление / Описание
EditKey	void EditKey(void) Переводит набор данных в режим поиска с частичным изменением ключей
EditRange End	void EditRangeEnd(void) Подготавливает изменение верхней границы вводимого значения
EditRange Start	void EditRangeStart(void) Подготавливает изменение нижней границы вводимого значения
FieldByName	TField* **FieldByName(const** AnsiString **FieldName)** Находит поле по его имени FieldName. При неверном имени генерирует исключение
FindField	TField* **FindField(const** AnsiString **FieldName)** Находит поле по его имени. При неверном имени возвращает nil
FindFirst	**bool** FindFirst(void) Перемещает курсор к первой записи и возвращает **true** в случае успеха
FindKey	bool FindKey(const System::TVarRec * KeyValues, const int KeyValues_Size) Ищет запись по заданным ключам KeyValues
FindLast	bool FindLast(void) Перемещает курсор к последней записи и возвращает **true** в случае успеха
FindNearest	void FindNearest(const System::**TVarRec** * KeyValues, const int KeyValues_Size) Перемещает курсор на запись, ближайшую к заданным ключам
FindNext	bool FindNext(void) Перемещает курсор к следующей записи и возвращает true в случае успеха
FindPrior	bool FindPrior(void) Перемещает курсор к предыдущей записи и возвращает true в случае успеха
First	void First(void) Перемещает курсор к первой записи
FlushBuffers	void **FlushBuffers(void)** Пересылает в базу данных все изменения, сохраненные в буфере
GetDetail DataSets	void GetDetailDataSets(Classes::TList* List) Заполняет список данными о вспомогательном наборе данных, находящемся с данным набором в соотношении master/detail
GetDetail LinkFields	void **GetDetailLinkFields(Classes::TList*** MasterFields, Classes::TList* DetailFields) Во вспомогательном наборе выдает списки ключевых полей этого и головного наборов данных

Метод	Объявление / Описание
GetFieldNames	void **GetFieldNames(Classes::TStrings*** List) Выдает список имен всех полей набора данных
GotoCurrent	void GotoCurrent(TTable* Table) Синхронизирует текущую запись таблицы с текущей записью другого набора данных
GotoKey	**bool** GotoKey(void) - Перемещает курсор на запись, найденную с помощью методов SetKey и EditKey
GotoNearest	void GotoNearest(void) Перемещает курсор на запись, примерно соответствующую ключам, заданным методами SetKey и EditKey
Insert	HIDESBASE void **Insert(void)** Вставляет новую пустую запись в набор данных
InsertRecord	void **InsertRecord(const System::TVarRec** * Values, const int Values_Size) Вставляет новую заполненную запись в набор данных
Last	void Last(void) Перемещает курсор к последней записи
Locate	bool Locate(const AnsiString KeyFields, const System::Variant &KeyValues, TLocateOptions Options) Осуществляет поиск записи в наборе данных
Lookup	**System::Variant Lookup(const** AnsiString KeyFields, const Variant **&KeyValues,** const AnsiString ResultFields) Осуществляет поиск записи в наборе данных и возвращает значения указанных полей этой записи
MoveBy	int MoveBy(int Distance) Перемещает курсор на заданное число записей
Next	void Next(void) Перемещает курсор к следующей записи
Open	void **Open(void)** Открывает набор данных
OpenDatabase	TDatabase* OpenDatabase(void) Открывает базу данных
Post	void Post(void) Пересылает отредактированную запись в базу данных
Prior	void Prior(void) Перемещает курсор к предыдущей записи
RevertRecord	void RevertRecord(void) Отменяет исправления текущей записи

Метод	Объявление / Описание
SetFields	void SetFields(const System::TVarRec * Values, const int Values_Size) Устанавливает значения всех полей записи
SetKey	void SetKey(void) Переводит набор данных в режим поиска по ключам
SetRange	void SetRange(const System: :TVarRec * Start Values, const int StartValues_Size, const System::TVarRec * EndValues, const int EndValues_Size) Устанавливает диапазон допустимых значений
SetRangeEnd	void SetRangeEnd(void) Устанавливает верхнюю границу диапазона допустимых значений
SetRange Start	void SetRangeStart(void) Устанавливает нижнюю границу диапазона допустимых значений

Основные события

Событие	Описание
OnUpdateError	Наступает при генерации исключения в процессе пересылки в базу данных измененной записи.
OnUpdateRecord	Наступает при пересылке кэшированной записи в базу данных.

Кроме того наследуется множество событий класса **TDataSet.**

Timer — таймер

Невизуальный компонент, позволяющий задавать в приложении интервалы времени.

Страница библиотеки *System*

Класс *TTimer*

Иерархия *TObiect — TPersistent — TComponent*

Модуль *extctrls*

Описание

Компонент **Timer** позволяет задавать в приложении интервалы времени. Таймер находит многочисленные применения: синхронизация мультипликации, закрытие каких-то окон, с которыми пользователь долгое время не работает, включение хранителя экрана или закрытие связей с удаленным сервером при отсутствии действий пользователя, регулярный опрос каких-то источников информации, задание времени на ответ в обучающих программах — все это множество задач, в которых требуется задавать интервалы времени, решается с помощью таймера.

Таймер имеет два свойства, позволяющие им управлять: **Interval** — интервал времени в миллисекундах и **Enabled** — доступность. Свойство **Interval** задает период срабатывания таймера. Через заданный интервал времени после предыдущего срабатывания, или после программной установки свойства **Interval,** или после запуска приложения, если значение **Interval** установлено во время проектирования, таймер срабатывает, вызывая событие **OnTimer. В** обработчике этого события записываются необходимые операции.

Если задать **Interval = 0 или Enabled = false,** то таймер перестает работать. Чтобы запустить отсчет времени надо или задать **Enabled = true,** если установлено положительное значение **Interval,** или задать положительное значение **Interval,** если **Enabled = false.**

Например, если требуется, чтобы через 5 секунд после запуска приложения закрылась форма-заставка, отображающая логотип приложения, на ней надо разместить таймер, задать в нем интервал **Interval** = 5000, а в обработчик события **OnTimer** вставить оператор **Close,** закрывающий окно формы.

Если необходимо в некоторой процедуре запустить таймер, который отсчитал бы заданный интервал, например, 5 секунд, после чего надо выполнить некоторые операции и отключить таймер, это можно сделать следующим образом. При проектировании таймер делается доступным **(Enabled = true),** но свойство **Interval** задается равным 0. Таймер не будет работать, пока в момент, когда нужно запустить таймер, не выполнится оператор

```
Timer1->Interval = 5000;
```

Через 5 секунд после этого наступит событие **OnTimer.** В его обработчике надо задать оператор

```
Timer1->Interval = 0;
```

который отключит таймер, после чего можно выполнять требуемые операции.

Другой эквивалентный способ решения задачи — использование свойства **Enabled.** Во время проектирования задается значение **Interval** = 5000 и значение **Enabled = false.** В момент, когда надо запустить таймер выполняется оператор

```
Timer1->Enabled = true;
```

В обработчик события **OnTimer,** которое наступит через 5 секунд после запуска таймера, можно вставить оператор

```
Timer1->Enabled = false;
```

который отключит таймер.

Таймер точно выдерживает заданные интервалы **Interval,** если они достаточно велики — сотни и тысячи миллисекунд. Если же задавать интервалы длительностью десятки или единицы миллисекунд, то реальные интервалы времени оказываются заметно больше вследствие различных накладных расходов, связанных с вызовами функций и иными вычислительными аспектами.

Основные свойства

Свойство	Объявление / Описание
Enabled	**bool** Enabled Определяет способность таймера отмерять отрезки времени
Interval	Cardinal Interval Интервал времени в миллисекундах, отмеряемый таймером. При значении 0 таймер не реагирует на время

Основные методы

Никаких специальных методов в компоненте не объявлено. Компонент наследует множество методов от класса **TComponent.**

Событие

Событие	Описание
OnTimer	Событие происходит, когда истек очередной отрезок времени Interval

ToolBar — инструментальная панель

Инструментальная панель для быстрого доступа к часто используемым функциям приложения с помощью инструментальных быстрых кнопок.

Страница библиотеки *Win32*

Класс *TToolBar*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomStaticText*

Модуль *comctrls*

Пример изображения

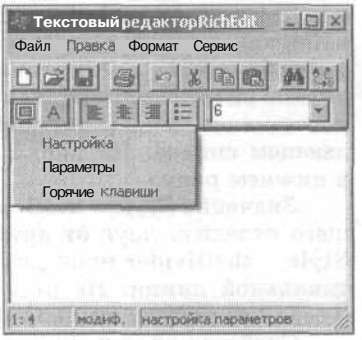


Рис. 2.39
Пример приложения с инструментальной панелью ToolBar

Описание

Компонент **ToolBar** является инструментальной панелью и управляет компоновкой инструментальных быстрых кнопок и других компонентов. Размещаемые на панели компоненты автоматически располагаются рядами и упорядочиваются по размерам.

Для занесения на панель **ToolBar** кнопок надо щелкнуть на **ToolBar** правой кнопкой мыши и выбрать из всплывшего меню команду New Button. На форме появится очередная кнопка — объект типа **ToolButton**. Это не совсем обычная кнопка, так как ее вид и поведение определяется ее свойством Style, которое по умолчанию равно **tbsButton** — кнопка. Кнопка этого стиля похожа на кнопку **SpeedButton.** Изображение на кнопке определяется свойством **ImageIndex**. Оно задает индекс изображения, хранящегося во внешнем компоненте ImageList. Указание на этот компонент может задаваться такими свойствами компонента **ToolBar, как Images, DisabledImages** (указывает на список изображений кнопок в недоступном состоянии) и **HotImages** (указывает на список изображений кнопок в моменты, когда над ними перемещается курсор мыши).

Свойство **MenuItem** позволяет задать раздел главного или контекстного меню, который дублируется данной кнопкой. При установке этого свойства, если в соответствующем разделе меню было задано изображение и установлен текст подсказок (свойство **Hint),** то это же изображение появится на кнопке и тот же текст появится в свойстве **Hint** кнопки. Передадутся из раздела меню в кнопку также значения свойств **Enabled и Visible.** Свойство **Wrap,** установленное в **true,** приводит к тому, что после этой кнопки ряд кнопок на панели прерывается и следующие кнопки размещаются в следующем ряду. Надо только установить в **ToolBar** свойство **AutoSize** раным **false,** чтобы высота панели не подстраивалась автоматически под размер кнопок, и установить высоту панели достаточную для размещения двух рядов кнопок. На рис. 2.39 **Wrap = true** задано в кнопке Заменить — последней в верхнем ряду.

Свойство **Wrap = true** действует только в случае, если свойство **Wrapable** компонента **ToolBar** установлено в **false.** Это свойство (если оно установлено в true)

обеспечивает автоматический перенос кнопок в следующий ряд панели во время выполнения, если они не помещаются в предыдущем ряду. Так что при **Wrapable = true** прерывать ряд свойством Wrap не имеет смысла.

Теперь вернемся к свойству **Style,** задающему стиль кнопки. Значение **Style = tbsCheck** определяет, что после щелчка пользователя на кнопке она остается в нажатом состоянии. Повторный щелчок на кнопке возвращает ее в отжатое состояние. Поведение такой кнопки подобно кнопкам **SpeedButton** и определяется аналогичными свойствами **AllowAllUp** и **Down.** Если при этом в нескольких кнопках установлено свойство **Grouped = true,** то эти кнопки образуют группу, из которой только одна кнопка может находиться в нажатом состоянии. На рис. 2.39 такой стиль установлен в кнопках выравнивания (3-5 слева в нижнем ряду).

Значение **Style = tbsDropDown** соответствует кнопке в виде выпадающего списка. Этот стиль удобен для воспроизведения выпадающего меню. Если для подобной кнопки задать в качестве свойства **Menultem** головной раздел меню, то в выпадающем списке автоматически будут появляться разделы выпадающего меню. Можно вместо свойства **Menultem** задать свойство **DropDownMenu,** определяющее контекстное меню (компонент **TPopupMenu),** которое будет отображаться в выпадающем списке. На рис. 2.39 стиль **tbsDropDown** использован в левой кнопке в нижнем ряду.

Значение **Style = tbsSeparator** приводит к появлению разделителя, позволяющего отделить друг от друга кнопки разных функциональных групп. Значение **Style = tbsDivider** приводит к появлению разделителя другого типа — в виде вертикальной линии. На рис. 2.39 стиль **tbsSeparator** установлен у разделителей в верхнем ряду, а стиль **tbsDivider** — у разделителей в нижнем ряду.

Свойство кнопки **Indeterminate** задает ее третье состояние — не нажатая и не отпущенная. Это свойство можно устанавливать в **true** во время выполнения, если в данном режиме кнопка не доступна.

Свойство **Marked** выделяет кнопку.

Мы рассмотрели занесение на панель кнопок. Но в инструментальных панелях нередко используются и выпадающие списки. Например, для задания размера шрифта. Не представляет труда перенести на панель **ToolBar** такие компоненты, как **ComboBox** (см. рис. **2.39), CSpinEdit и др.**

Из общих свойств компонента **ToolBar** следует отметить **ButtonHeight и ButtonWidth** — высота и ширина кнопок в пикселах.

Свойства, определяющие вид панели: **Border Width** — ширина бордюра, **EdgeInner и EdgeOuter** — стиль изображения внутренней и внешней части панели (утопленный или выступающий), **EdgeBorders** — определяет изображение отдельных сторон панели (левой, правой, верхней, нижней).

Основные свойства

Свойство	Объявление / Описание
Align	**enum TAlign {alNone, alTop, alBottom, alLeft, alRight, alClient, alCustom};** **TAlign Align** **Определяет способ выравнивания панели (по умолчанию alTop)**
ButtonCount	**int ButtonCount** **Указывает количество кнопок, в компоненте**
ButtonHeight	**int ButtonHeight** **Высота кнопок**
Buttons	**TToolButton* Buttons[int Index]** **Индексированный список объектов кнопок типа TToolButton**

Свойство	Объявление / Описание
ButtonWidth	int ButtonWidth Ширина кнопок
Canvas	TCanvas Canvas Канва, используемая для нестандартного изображения панели в обработчиках событий OnCustomDraw и **OnCustomDrawItem**
Disabled Images	**TCustomImageList* DisabledImages** Список изображений ImageList, используемых для недоступных кнопок
EdgeBorders	enum **TEdgeBorder** {ebLeft, ebTop, ebRight, ebBottom}; typedef **Set<TEdgeBorder,** ebLeft, **ebBottom> TEdgeBorders;** **TEdgeBorders** EdgeBorders Определяет изображение отдельных сторон панели (левой, правой, верхней, нижней)
EdgeInner	enum TEdgeStyle {esNone, esRaised, esLowered}; TEdgeStyle Edgelnner Стиль изображения внутренней части панели (плоский, выступающий или утопленный)
EdgeOuter	enum TEdgeStyle **{esNone,** esRaised, **esLowered};** TEdgeStyle EdgeOuter Стиль изображения внешней части панели (плоский, выступающий или утопленный)
Flat	**bool** Flat Определяет плоский (true) или объемный вид кнопок
HotImages	TCustomlmageList* Hotlmages Список изображений ImageList, используемых для кнопок в моменты, когда над ними перемещается курсор мыши
Images	TCustomlmageList* Images Список изображений ImageList, используемых для кнопок
Indent	int Indent Определяет поле, отделяющее левый край компонента от расположенных в нем кнопок
List	bool List При значении **true** выравнивает надписи (если они видимы) вправо, а изображения влево. При значении false — выравнивание по центру
RowCount	int RowCount Определяет количество рядов кнопок на инструментальной панели. Свойство только для чтения
ShowCaptions	bool ShowCaptions Включает или отключает отображение надписей на кнопках
Wrapable	bool Wrapable Определяет возможность автоматического образования новых рядов инструментальных кнопок, если они не помещаются на компоненте в один ряд

Основные методы

Никаких специальных методов, кроме защищенных, предназначенных для построения классов-наследников, в компоненте не объявлено. Компонент наследует множество методов от класса **TWinControl**.

Основные события

Событие	Описание
OnAdvancedCustomDraw	Событие происходит перед прорисовкой панели. Обработчик может использоваться для нестандартного изображения фона панели на канве. Нестандартная прорисовка отдельных кнопок осуществляется в обработчиках OnCustomDrawButton или OnAdvancedCustomDrawButton. Отличается от OnCustomDraw организацией прорисовки
OnAdvancedCustomDrawButton	Событие происходит перед прорисовкой кнопок панели. Обработчик может использоваться для нестандартного изображения кнопок. Отличается от **OnCustomDrawButton** организацией прорисовки
OnCustomDraw	Событие происходит перед прорисовкой панели. Обработчик может использоваться для нестандартного изображения фона панели на канве. Нестандартная прорисовка отдельных кнопок осуществляется в обработчиках **OnCustomDrawButton** или OnAdvancedCustomDrawButton. Отличается от OnAdvancedCustomDraw организацией прорисовки
OnCustomDrawButton	Событие происходит перед прорисовкой кнопок панели. Обработчик может использоваться для нестандартного изображения кнопок. Отличается от OnAdvancedCustomDrawButton организацией прорисовки

TreeView — иерархические данные в виде дерева

Служит для отображения иерархических данных в виде дерева, в котором пользователь может выбрать нужный ему узел или узлы.

Страница библиотеки *Win32*

Класс *TTreeView*

Иерархия *TObject — TPersistent — TComponent — TControl — TWinControl — TCustomTreeView*

Модуль *comctrls*

Пример изображения

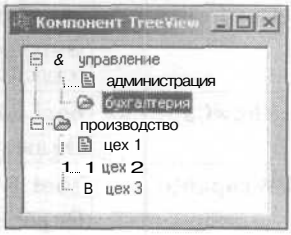


Рис. 2.40
Пример компонента TreeView

Описание

Компонент **Tree View** представляет собой окно для отображения иерархических данных в виде дерева, в котором пользователь может выбрать нужный ему узел или узлы. Иерархическая информация может быть самой разной: структура некоторого предприятия, структура документации учреждения, структура отчета и т.п. С каждым узлом дерева могут быть связаны некоторые данные.

Основным свойством **TreeView,** содержащим информацию об узлах дерева, является индексированный список узлов **Items.** Каждый узел является объектом типа **TTreeNodes,** обладающим своими свойствами и методами.

Во время проектирования формирование дерева осуществляется в окне редактора узлов дерева (см. рис. 2.41), которое вызывается двойным щелчком на компоненте **TreeView** или нажатием кнопки с многоточием около свойства **Items** в окне Инспектора Объектов.

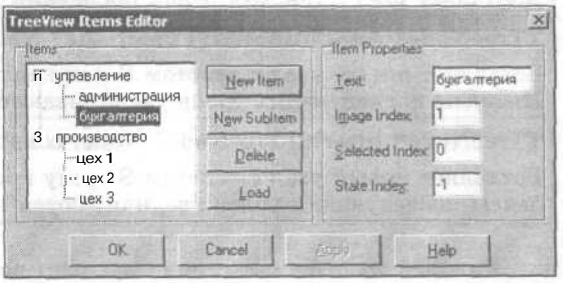


Рис. 2.41
Окно редактора узлов дерева

Кнопка New Item (новый узел) позволяет добавить в дерево новый узел. Он будет расположен на том же уровне, на котором расположен узел, выделенный курсором в момент щелчка на кнопке New Item.

Кнопка New SubItem (новый дочерний узел) позволяет добавить в дерево дочерний узел. Он будет расположен на уровень ниже уровня того узла, который выделен курсором в момент щелчка на кнопке New SubItem.

Кнопка Delete (удалить) удаляет выделенный узел дерева. Кнопка Load позволяет загрузить структуру дерева из файла. Файл, хранящий структуру дерева – это обычный текстовый файл, содержащий тексты узлов. Уровни узлов обозначаются отступами. Например, файл дерева, изображенного на приведенных рисунках, может иметь вид:

```
производство
    цех 1
    цех 2
    цех 3
управление
    администрация
    бухгалтерия
```

Для каждого нового узла дерева можно указать ряд свойств: **Text** — надпись, появляющаяся в дереве около данного узла, **ImageIndex и SelectedIndex** — индексы пиктограмм, отображаемых для узла, который соответственно не выделен и выделен пользователем в данный момент. Эти индексы соответствуют списку изображений, хранящихся в отдельном компоненте **ImageList,** на который указывает свойство **Images** компонента **TreeView.** Свойство узла — **StateIndex** позволяет добавить вторую пиктограмму в данный узел, не зависящую от состояния узла. Подобная пиктограмма может просто служить дополнительной характеристикой узла. Индекс, указываемый как **StateIndex,** соответствует списку изображений, хранящихся в отдельном компоненте **ImageList,** указанном в свойстве **StateImages** компонента **TreeView.**

Множество методов объектов типа **TTreeNodes** позволяет перестраивать дерево во время выполнения приложения. Следующие методы позволяют вставлять в дерево новые узлы:

TTreeNode* Add(TTreeNode* Node, const AnsiString S) Добавляет новый узел с текстом S как последний узел уровня, на котором расположен **Node**
TTreeNode* AddFirst(TTreeNode* Node, const AnsiString S) Вставляет новый узел с текстом S как первый из узлов уровня, на котором находится **Node.** Индексы последующих узлов увеличиваются на 1
TTreeNode* AddChild(TTreeNode* Node, const AnsiString S) Добавляет узел с текстом S как последний дочерний узла **Node**
TTreeNode* AddChildFirst(TTreeNode* Node, const AnsiString S) Вставляет новый узел с текстом S как первый из дочерних узлов узла **Node.** Индексы последующих узлов увеличиваются на 1
TTreeNode* Insert(TTreeNode* Node, const AnsiString S) Вставляет новый узел с текстом S сразу после узла **Node** на то же уровень. Индексы последующих узлов увеличиваются на 1

Каждый из этих методов возвращает вставленный узел. В качестве примера их применения ниже приведен код, формирующий то же дерево, которое вы можете видеть на рис. 2.40 и 2.41.

```
TreeView1->Items->Clear(); // очистка списка
// добавление корневого узла "производство" (индекс 0)
TreeView1->Items->Add(NULL, "производство");

/* добавление дочених узлов "цех 1" - "цех 3"
   (индексы 1 - 3) */
TreeView1->Items->AddChild(TreeView1->Items->Item[0],"цех 1");
TreeView1->Items->AddChild(TreeView1->Items->Item[0],"цех 2");
TreeView1->Items->AddChild(TreeView1->Items->Item[0],"цех 3");

/* добавление корневого узла "управление" после узла
    "производство" (индекс 4) */
TreeView1->Items->Add(TreeView1->Items->Item[0], "управление");

/* добавление дочених узлов "администрация" и "бухгалтерия" узла
"управление"*/
TreeView1->Items->AddChild(
                  TreeView1->Items->Item[4],"администрация");
TreeView1->Items->AddChild(
                  TreeView1->Items->Item[4],"бухгалтерия");
```

Дерево может быть сколь угодно разветвленным. Например, следующие операторы добавляют дочерние узлы "бригада 1" и "бригада 2" в сформированный ранее узел "цех 1":

```
TreeView1->Items->AddChild(TreeView1->Items->Item[1],
                                                "бригада 1");
TreeView1->Items->AddChild(TreeView1->Items->Item[1],
                                                "бригада 2");
```

Текст, связанный с некоторым узлом, можно найти с помощью его свойства Text. Например, **TreeView1->Items->Item[1]->Text** — это надпись "цех 1".

С каждым узлом может быть связан некоторый объект. Добавление таких узлов осуществляется методами **AddObject, AddObjectFirst, InsertObject, AddChildObject, AddChildObjectFirst,** аналогичными приведенным выше, но содержащими в качестве параметра еще указатель на объект:

TTreeNode* AddObject(TTreeNode* Node, const AnsiString S, void * Ptr) Добавляет новый узел с текстом S и объектом **Ptr** как последний узел уровня, на котором расположен **Node**
TTreeNode* AddChildObjectFirst(TTreeNode* Node, const AnsiString S, void * Ptr) Вставляет новый узел с текстом S и объектом **Ptr** как первый из узлов уровня, на котором находится **Node.** Индексы последующих узлов увеличиваются на 1
TTreeNode* AddChildObject(TTreeNode* Node, const AnsiString S, void * Ptr) Добавляет узел с текстом S и объектом **Ptr** как последний дочерний узла **Node**
TTreeNode* AddChildObjectFirst(TTreeNode* Node, const AnsiString S, void * Ptr) Вставляет новый узел с текстом S и объектом **Ptr** как первый из дочерних узлов узла **Node.** Индексы последующих узлов увеличиваются на 1
TTreeNode* InsertObject(TTreeNode* Node, const AnsiString S, void * Ptr) Вставляет новый узел с текстом S и объектом **Ptr** сразу после узла **Node** на то же уровень. Индексы последующих узлов увеличиваются на 1

Объект, связанный с некоторым узлом, можно найти с помощью его свойства **Data.** Например, **TreeView1.Items.Item[1].Data.**

Для удаления узлов имеется два метода: **Clear,** очищающий все дерево, и **Delete(Node: TTreeNode),** удаляющий указанный узел **Node** и все его узлы — потомки. Например, оператор

```
TreeView1.Items.Clear;
```

Объект, связанный с некоторым узлом, можно найти с помощью его свойства **Data.** Например, **TreeView1->Items->Item[1]->Data.**

Для удаления узлов имеется два метода: **Clear(void),** очищающий все дерево, и **Delete(TTreeNode* Node),** удаляющий указанный узел **Node** и все его узлы – потомки. Например, оператор

```
TreeView1->Items->Clear();
```

удалит все узлы, а оператор

```
TreeView1->Items->Delete(TreeView1->Items->Item[1]);
```

удалит узел "цех 1" и его дочерние узлы (если они имеются).

При удалении узлов, связанных с объектами, сами эти объекты не удаляются.

Реорганизация дерева, связанная с созданием или удалением многих узлов, может вызывать неприятное мерцание изображения. Избежать этого можно с помощью методов **BeginUpdate** и **EndUpdate.** Первый из них запрещает перерисовку дерева, а второй — разрешает. Таким образом, изменение структуры дерева может осуществляться по следующей схеме:

```
TreeView1->Items->BeginUpdate();
<операторы изменения дерева>
TreeView1->Items->EndUpdate();
```

Если метод **BeginUpdate** применен подряд несколько раз, то перерисовка дерева произойдет только после того, как столько же раз будет применен метод **EndUpdate.**

Среди свойств узлов следует отметить **Count** — число узлов, управляемых данным, т.е. дочерних узлов, их дочерних узлов и т.п. Если значение **Count** узла равно нулю, значит у него нет дочерних узлов, т.е. он является листом дерева.

Вернемся к свойствам компонента **TreeView.** Важным свойством компонента **TreeView** является **Selected.** Это свойство указывает узел, который выделен пользователем. Пользуясь этим свойством можно запрограммировать операции, которые надо выполнить для выбранного пользователем узла. Если ни один узел не выбран, значение **Selected** равно **NULL.** При выделении пользователем нового узла происходят события **OnChanging** (перед изменением выделения) и **OnChanged** — после выделения. В обработчик события **OnChanging** передаются параметры **TTreeNode *Node** — узел, который выделен в данный момент, и **bool &AllowChange** — разрешение на перенос выделения. Если в обработчике задать **AllowChange = false,** то переключение выделения не произойдет. В обработчик события **OnChanged** передается только параметр **TTreeNode *Node** — выделенный узел. В этом обработчике можно предусмотреть действия, которые должны производится при выделении узла.

У компонента **TreeView** имеется свойство **RightClickSelect,** разрешающее (при значении равном **true**) выделение узла щелчком как левой, так и правой кнопкой мыши. Но и в этом случае событие **OnChanged** наступает только при выделении узла левой кнопкой мыши.

Ряд событий компонента **TreeView** связан с развертыванием и свертыванием узлов. При развертывании узла происходят события **OnExpanding** (перед развертыванием) и **OnExpanded** (после развертывания). В обработчики обоих событий передается параметр **TTreeNode *Node** — развертываемый узел. Кроме того в обработчик **OnExpanding** передается параметр **bool &AllowExpansion,** который можно задать равным **false,** если желательно запретить развертывание. При свертывании узла происходят события OnCollapsing (перед свертыванием) и **OnCollapsed (после** свертывания). Так же, как и в событиях, связанных с развертыванием, в обработчики передается параметр **TTreeNode *Node** — свертываемый узел, а в обработчик **OnCollapsing** дополнительно передается параметр **bool &AllowCollapse,** разрешающий или запрещающий свертывание.

Свойство **ReadOnly** компонента **TreeView** позволяет запретить пользователю редактировать отображаемые данные. Если редактирование разрешено, то при редактировании возникают события **OnEditing и OnEdited,** аналогичные рассмотренным ранее (в обработчике **OnEditing** параметр **bool &AllowEdit** позволяет запретить редактирование).

Ряд свойств компонента **TreeView: ShowButtons, ShowLines, ShowRoot** отвечают за изображение дерева и позволяют отображать или убирать из него кнопки, показывающие раскрытия узла, линии, связывающие узлы, и корневой узел. Поэкспериментируйте с этими свойствами и увидите, как они влияют на изображение.

Свойство **SortType** позволяет автоматически сортировать ветви и узлы дерева. По умолчанию это свойство равно **stNone,** что означает, что дерево не сортируется. Если установить **SortType** равным **stText,** то узлы будут автоматически сортироваться по алфавиту. Возможно также проводить сортировку по связанным с узлами объектам **Data** (значение **SortType** равно **stData),** одновременно по тексту и объектам **Data** (значение **SortType равно stBoth)** или любым иным способом. Для использования этих возможностей сортировки надо написать обработчик события **OnCompare,** в который передаются, в частности, параметры **TTreeNode *Nodel и TTreeNode *Node2** — сравниваемые узлы, и параметр **int &Compare,** в который надо заносить результат сравнения: отрицательное число, если узел **Node1** должен располагаться ранее **Node2, 0,** если эти узлы считаются эквивалентными, и положительное число, если узел **Nodel** должен располагаться в дереве после **Node2.** Например, следующий оператор в обработчике события **OnCompare** обеспечивает обратный алфавитный порядок расположения узлов:

```
Compare = - AnsiStrIComp(Node1->Text.c_str(), Node2->Text.c_str());
```

События **OnCompare** наступают после задания любого значения **SortType,** отличного от **stNone,** и при изменении пользователем свойств узла (например, при редактировании им надписи узла), если значение **SortType** не равно **stNone.** После сортировки первоначальная последовательность узлов в дереве теряется. Поэтому последующее задание **SortType = stNone** не восстанавливает начальное расположение узлов, но исключает дальнейшую генерацию событий **OnCompare,** т.е. автоматическую перестановку узлов, например, при редактировании их надписей. Если же требуется изменить характер сортировки или провести сортировку с учетом новых созданных узлов, то надо сначала задать значение **SortType = stNone,** а затем задать любое значение **SortType,** отличное от **stNone.** При этом будут сгенерированы новые обращения к обработчику событий **OnCompare.**

Имеются и другие возможности сортировки. Например, метод **AlphaSort(void)** обеспечивает алфавитную последовательность узлов независимо от значения **SortType,** но при отсутствии обработчика событий **OnCompare** (если обработчик есть, то при выполнении метода AlphaSort происходит обращение к этому обработчику). Отличие метода **AlphaSort** от задания значения **SortType = stText** заключается в том, что изменение надписей узлов приводит к автоматической пересортировке дерева только при **SortType = stText.**

Основные свойства

Свойство	Объявление / Описание
AutoExpand	**bool** AutoExpand Определяет, раскрываются ли и свертываются ли узлы автоматически в зависимости от выбора в просматриваемом дереве
Canvas	Graphics::TCanvas* Canvas Канва дерева. Ее свойства и метода можно использовать в обработчиках событий OnCustomDraw и OnCustomDrawItem для рисования собственных изображений
ChangeDelay	int ChangeDelay Определяет задержку между моментом выбора узла и моментом события OnChange
DropTarget	TTreeNode* DropTarget Определяет, какой элемент дерева появляется в результате операции перетаскивания Drag&Drop
HotTrack	**bool** HotTrack Определяет, выделяются ли цветом элементы, над которыми проходит курсор мыши
Images	**Imglist::TCustomImageList*** Images Определяет, какой список изображений ImageList, связан с деревом
Indent	int Indent Определяет отступ, в пикселах, отделяющий родительские узлы от дочерних при раскрытии списка дочерних узлов
Items	TTreeNodes* Items Индекисрованный список узлов дерева

Свойство	Объявление / Описание
Readonly	**bool** ReadOnly Указывает, может ли пользователь изменять текст надписей узлов дерева
RightClick Select	bool RightClickSelect Определяет, возвращает ли свойство Selected узлы, выбранные с помощью правой кнопки мыши
RowSelect	bool RowSelect Определяет, выделяется ли цветом ряд выделенного элемента
Selected	**TTreeNode*** Selected ' Выбранный узел дерева
ShowButtons	bool ShowButtons Определяет, отображаются ли кнопки со знаками "+" (раскрыть узел) и " " (свернуть узел) с левой стороны каждого родительского элемента
ShowLines	bool ShowLines Определяет, видны ли линии, связывающие дочерние узлы с соответствующими родительскими узлами
ShowRoot	bool ShowRoot Определяет, отображаются ли линии, соединяющие узлы верхнего уровня дерева с его корнем
SortType	enum **TSortType** { stNone, stData, stText, stBoth }; TSortType SortType Позволяет автоматически сортировать ветви и узлы дерева: stNone — дерево не сортируется, stText — сортировка по алфавиту, stData — сортировка по связанным с узлами объектам Data, stBoth — сортировка одновременно по тексту и объектам Data или любым иным способом. Для использования этих возможностей сортировки надо написать обработчик события ОпСотраге
StateImages	**Imglist::TCustomImageList* StateImages** Список изображений ImageList, используемый для добавления вторых пиктограмм в узлы. Подобные пиктограммы могут служить дополнительной характеристикой отдельных узлов
ToolTips	bool ToolTips Определяет, снабжены ли элементы дерева подсказками. Сами тексты подсказок надо задавать в обработчике события OnHint, используя свойство Hint
TopItem	TTreeNode* Topltem Определяет самый верхний видимый узел

Основные методы

Метод	Объявление / Описание
AlphaSort	bool AlphaSort(bool ARecurse = true) Обеспечивает алфавитную последовательность узлов независимо от значения SortType, но при отсутствии обработчика событий OnCompare (если обработчик есть, то при выполнении метода AlphaSort происходит обращение к этому обработчику). Отличие метода от задания значения SortType = stText заключается в том, что изменение надписей узлов приводит к автоматической пересортировке дерева только при SortType = stText
CustomSort	typedefint (CALLBACK *PFNTVCOMPARE)(LPARAM lParam1, LPARAM lParam2, LPARAM lParamSort); bool CustomSort(PFNTVCOMPARE SortProc, int Data; bool ARecurse = true) Сортирует узлы в последовательности, определяемой указанной функцией SortProc
FullCollapse	void FullCollapse(void) Свертывает все узлы дерева
FullExpand	void FullExpand(void) Раскрывает все узлы дерева
GetNodeAt	TTreeNode* GetNodeAt(int X, int Y) Возвращает узел, найденный в указанной позиции
IsEditing	bool IsEditing(void) Указывает, редактируется ли в данный момент пользователем надпись узла
LoadFrom File	void LoadFromFile(const AnsiString FileName) Считывает указанный и загружает данные в дерево
LoadFrom Stream	void LoadFromStream(Classes::TStream* Stream) Загружает в дерево данные из потока
SaveToFile	void SaveToFile(const AnsiString FileName) Сохраняет дерево в файле с указанным именем
SaveTo Stream	void SaveToStream(Classes::TStream* Stream) Записывает данные дерева в поток

Основные события

Событие	Описание
OnAdvanced CustomDraw	Событие происходит в процессе прорисовки дерева. Обработчик может использоваться для нестандартного рисования на канве
OnAdvanced CustomDraw Item	Событие происходит в процессе рисования узлов дерева. Обработчик может использоваться для нестандартного рисования узлов на канве

Событие	Описание
OnChange	Событие происходит после перемещения выделения на другой узел
OnChanging	Событие происходит перед перемещением выделения на другой узел. Обработчик позволяет запретить перемещение
OnCollapsed	Событие происходит после свертывания узла
OnCollapsing	Событие происходит перед свертыванием узла. Обработчик позволяет запретить перемещение
OnCompare	Событие происходит во время сравнения двух узлов в процессе сортировки узлов дерева. Обработчик должен сообщить, какой из двух переданных в него узлов должен быть расположен выше другого
OnCustomDraw	Событие происходит перед прорисовкой дерева. Обработчик может использоваться для нестандартного изображения дерева на канве
OnCustomDraw Item	Событие происходит перед прорисовкой узла. Обработчик может использоваться для нестандартного изображения узла на канве
OnDeletion	Событие происходит при удалении узла из дерева
OnEdited	Событие происходит после редактирования пользователем надписи узла
OnEditing	Событие происходит, когда пользователь начинает редактировать надпись узла. Обработчик позволяет запретить редактирование
OnExpanded	Событие происходит после раскрытия узла
OnExpanding	Событие происходит перед раскрытием узла. Обработчик позволяет запретить раскрытие
OnGetImage Index	Событие происходит, когда отыскивается индекс изображения узла перед тем, как его нарисовать. Обработчик может использоваться для изменения изображения
OnGetSelected Index	Событие происходит, когда отыскивается индекс изображения выбранного узла перед тем, как его нарисовать. Обработчик может использоваться для изменения изображения

Глава 3

Свойства компонентов и классов C++Builder

В этой главе приведены развернутые описания более 200 свойств компонентов и классов C++Builder. Для многих свойств даются примеры их применения. Конечно, это далеко не все свойства. Для многих других свойств вы можете найти краткие описания в главах 1 и 2.

Определения свойств в данной главе даны несколько упрощенно, чтобы более наглядно была видна их суть и для экономии места. В частности, не приводятся их функции чтения и записи, которые могут представлять интерес только при проектировании новых компонентов с переопределением свойств родительских компонентов. Но в этом случае не представляет труда найти имена функций чтения и записи во встроенной справке C++Builder.

В описаниях вы можете встретить идентификаторы, выделенные подчеркиванием. Например, **TCanvas.** Это означает, что в этой или других главах в соответствующем разделе вы сможете найти развернутое пояснение, комментарий или примеры, связанные с данным термином.

Значительно большее количество описаний различных свойств вы можете найти в источнике [3].

Action

Определяет действие, связанное с данным управляющим элементом — разделом меню, кнопкой и др.

Класс *TControl*

Определение

```
__property Classes::TBasicAction* Action
```

Описание

Значение свойства **Action** выбирается во время проектирования из выпадающего списка предусмотренных действий в Инспекторе Объектов. Этот список формируется в процессе проектирования размещением на форме компонента **ActionList** и заданием его свойств.

Подробнее о диспетчеризации действий см. в разд. «Организация взаимодействия компонентов в приложении» в гл. 2.

Active

Определяет открытие и закрытие базы данных.

Класс *TDataSet*

Определение

```
__property bool Active
```

Описание

Свойство **Active** определяет наличие соединения с базой данных и управляет этим соединением. Если значение **Active** равно **false** — база данных закрыта и чтение или запись данных невозможны.

Установка Active в **true** вызывает следующее:

- Генерацию события BeforeOpen.
- Установку набора данных (свойства State) в состояние **dsBrowse.**
- Открытие курсора в наборе данных.
- Генерацию события **AfterOpen.**

Если во время открытия набора данных произошла ошибка, набор данных (свойство **State**) переводится в состояние **dsInactive** и курсор закрывается.

Приложение должно устанавливать Active в false перед изменением свойств набора данных, влияющих на его состояние или на отображение данных. Если этого не сделать, то будет сгенерировано исключение EDataBaseError с сообщением: «Cannot performe this operation on an open dataset» — «Невозможно выполнить эту операцию на открытом наборе данных».

Свойство Active может устанавливаться непосредственно или методами Open — установка в true, и Close — установка в false.

Примеры

```
Table1->Active = false;
Table1->TableName = "Pers.db";
Table1->Active = true;
```

Первый из приведенных операторов закрывает ранее открытый набор данных. Второй — изменяет таблицу, с которой далее будет работать компонент. Последний оператор открывает эту таблицу.

Аналогичных результатов достигает следующий код:

```
Table1->Close();
Table1->TableName = "Pers.db";
Table1->Open();
```

Aggregates

Список объектов совокупных (агрегированных) характеристик клиентского набора данных.

Класс *TCustomClientDataSet*

Объявление

```
__property TAggregates* Aggregates
```

Описание

Свойство **Aggregates** представляет собой коллекцию типа **TAggregates** объектов совокупных (агрегированных) характеристик типа **TAggregate.** Добавлять в Aggregates новые элементы во время проектирования можно двумя путями: щелчком на кнопке с многоточием около этого свойства в Инспекторе Объектов, или созданием нового поля совокупной характеристики с помощью Редактора Полей, вызываемого двойным щелчком на компоненте. В последнем случае одновременно с созданием поля автоматически добавляется соответствующий элемент в **Aggregates.**

При добавлении элемента в Aggregates с помощью Инспектора Объектов открывается обычное окно редактора коллекций, в котором кнопкой Add New вы можете добавить новый элемент. Выделив его, вы увидите в окне Инспектора Объектов его свойства. Основное из них — **Expression.** В нем вы должны записать выражение, задающее значение поля. Помимо обычных арифметических операций, скобок, имен полей и констант, выражение может содержать следующие функции:

Sum	Сумма значений числового поля или арифметического выражения
Avg	Среднее значение числового поля, или поля даты и времени, или арифметического выражения

Count	Число непустых значений указанного поля или выражения. Функция **Count(*)** возвращает общее число записей, независимо от значений полей
Min	Возвращает минимальное значение числового поля, строкового поля, поля дат и времени или соответствующего выражения
Max	Возвращает максимальное значение числового поля, строкового поля, поля дат и времени или соответствующего выражения

В записи выражения нельзя смешивать совокупные характеристики и значения полей, так как это принципиально различные объекты: совокупная характеристика — это одно значение, а поле — это множество значений. Приведем примеры выражений. Следующее выражение:

```
Max(Year_b) - Min(Year_b)
```

вычисляет диапазон значений поля **Year_b.** Выражение

```
Sum(Year_b) / Count(Year_b)
```

вычисляет среднее значение поля **Year_b,** то же, что вычислит и более простое выражение

```
Avg(Year_b)
```

А выражение

```
Avg(2002 - Year_b)
```

вычислит средний возраст, правда, только в 2002 году.

Помимо задания выражения, вычисляющего характеристику, надо установить в **true** свойство **Active** данного элемента. В этом случае значение будет вычисляться, но только если значение свойства **AggregatesActive** компонента клиентского набора будет установлено в **true.** Можете также задать имя элемента **AggregateName.**

Характеристики могут вычисляться по всем записям или по некоторой их совокупности. При вычислениях по всем записям свойство **GroupingLevel** элемента должно быть равно 0, а свойство **IndexName** не задается. В остальных случаях эти свойства задаются (подробнее о группировании записей см. в [1] и [3]).

Для добавления элемента в **Aggregates** во время выполнения используется метод **Add** класса **TAggregates.** После создания элемента надо задать его свойство **Expression** и можно указать имя **AggregateName.** Например:

```
TAggregate* A = ClientDataSet1->Aggregates->Add();
A->AggregateName = "AvgYear";
A->Expression = "Avg(Year_b)";
```

Значение Совокупной характеристики читается методом **Value.** Найти соответствующий объект в **Aggregates** можно или методом **Find,** или по его индексу в подсвойстве Items свойства **Aggregates.** Например, отобразить введенную выше характеристику можно оператором

```
ShowMessage(ClientDataSet1->Aggregates->Find("AvgYear"->Value());
```

или оператором

```
ShowMessage(ClientDataSet1->Aggregates->Items[0]->Value());
```

Если вы хотите ввести во время проектирования поле совокупной характеристики, сделайте двойной щелчок на компоненте клиентского набора данных, в открывшемся окне Редактора Полей щелкните правой кнопкой мыши и выберите команду New. Откроется окно задания нового поля. В этом окне, в частности, будет радиокнопка Aggregate. Ее надо включить и задать имя поля Name. Тип поля авто-

матически установится равным Aggregate. Щелкнув на OK, вы вернетесь в окно Редактора Полей, в нижней части которого появится новое поле. Выделив его, вы увидите в окне Инспектора Объектов его свойства. Вам надо задать **Expression, GroupingLevel и IndexName** так же, как описано выше для элемента типа **TAggregate.**

Отображать значения характеристик введенных полей можно обычными компонентами, используемыми для обычных полей набора данных.

Align

Определяет способ выравнивания компонента внутри контейнера (родительского компонента).

Класс *TControl*

Определение

```
enum TAlign ( alNone, alTop, alBottom, alLeft, alRight,
              alClient, alCustom };
typedef Set<TAlign, alNone, alClient> TAlignSet;

__property TAlign Align
```

Описание

Свойство Align определяет, остается ли компонент неизменным при изменении размеров содержащей его формы, панели, другого компонента, или он изменяется, занимая всю доступную площадь, ее верхнюю, нижнюю, левую или правую часть.

Возможные значения свойства:

Значение	Описание
alNone	Компонент остается там, где он размещен во время проектирования. Размеры его не изменяются. Это значение Align по умолчанию.
alTop	Компонент занимает всю верхнюю часть контейнера и во время выполнения приложения его ширина изменяется при изменении ширины контейнера. Высота компонента остается неизменной.
alBottom	Компонент занимает всю нижнюю часть контейнера и во время выполнения приложения его ширина изменяется при изменении ширины контейнера. Высота компонента остается неизменной.
alLeft	Компонент занимает всю левую часть контейнера и во время выполнения приложения его высота изменяется при изменении высоты контейнера. Ширина компонента остается неизменной.
alRight	Компонент занимает всю правую часть контейнера и во время выполнения приложения его высота изменяется при изменении высоты контейнера. Ширина компонента остается неизменной.
alClient	Компонент занимает всю клиентскую область контейнера и во время выполнения приложения его размеры изменяются при изменении размеров контейнера. Если в контейнере часть клиентской области уже занята, компонент занимает всю ее оставшуюся часть.
alCustom	Введено в C++Builder 6. Позиция компонента определяется вызовами функций CustomAlignPosition и CustomAlignInsertBefore.

Значение **Align** по умолчанию — **alNone.** В приложениях, в которых пользователь может изменять размер формы, а сама форма разбита панелями или другими компонентами на ряд областей, необходимо изменять это значение **Align.**

Если компонент имеет значение **Align,** равное alClient, то в процессе проектирования невозможно добраться до содержащего его контейнера и щелкнуть на нем, чтобы получить в Инспекторе Объектов его свойства и события. В этом случае возможны два решения: щелкнуть на компоненте и нажать клавишу Esc или осуществить выбор компонента-контейнера с помощью выпадающего списка в верхней части Инспектора Объектов.

Значения **Align alTop** и **alBottom** имеют приоритет перед **alLeft и alRight.** Поэтому, если вы, например, ввели на форму две панели, одной задали значение **alLeft,** а второй задаете значение **alTop,** то вторая панель вытеснит верхнюю часть первой панели, которая первоначально заняла всю левую часть клиентской области. Если это нежелательно, приходится вводить дополнительные панели, являющиеся контейнерами для других панелей.

В C++Builder 6 введено значение выравнивания **alCustom** (заказное) и тип **TAlignSet** — множество значений выравнивания. Если какие-то дочерние компоненты контейнера имеют значение **Align = alCustom,** то в компоненте-контейнере можно перегрузить виртуальный метод **CustomAlignPosition,** который будет автоматически вызываться каждый раз при изменении размеров контейнера для каждого дочернего компонента, имеющего **Align = alCustom и Visible = true.** Этот метод объявлен следующим образом (учтите, что во встроенной справке C++Builder 6 его объявление неверное):

```
virtual void__fastcall CustomAlignPosition(TControl* Control,
                       int &NewLeft, int &NewTop, int &NewWidth,
                       int &NewHeight, Types::TRect &AlignRect,
                       const TAlignInfo &AlignInfo);
```

Параметр **Control** -- это выравниваемый компонент. Параметры **NewLeft, NewTop, NewWidth, NewHeight** можно изменять, задавая новые координаты левого верхнего угла, ширину и высоту. Параметр **AlignRect** определяет область выравнивания, но не для данного компонента, а для следующего. Параметр **AlignInfo** определяет структуру типа **TAlignInfo,** содержащую сведения о выравнивании дочерних компонентов контейнера:

```
struct TAlignInfo
{
   Classes::TList* AlignList;
   int ControlIndex;
   TAlign Align;
   int Scratch;
};
```

Имеется еще один метод, который может быть переопределен при заказном выравнивании:

```
virtual bool__fastcall CustomAlignInsertBefore(TControl* C1,
                                               TControl* C2);
```

Функция **CustomAlignInsertBefore** поочередно вызывается для каждой пары C1 и C2 дочерних компонентов контейнера, в которых **Align = alCustom и Visible = true,** и определяет, в какой последовательности должны выравниваться компоненты, т.е. в какой последовательности вызывается процедура **CustomAlignPosition** для различных компонентов.

При вызовах **CustomAlignInsertBefore** пары перечисляются в последовательности их расположения в свойстве **Controls** контейнера, причем C1 — второй компонент в паре, а C2 - - первый. Например, если форма содержит три панели **Panell, Panel2 и Panel3,** размещенных в **Controls** именно в этой последовательности, то при первом вызове **C1 = Panel2, C2 = Panell,** при втором **C1 = Panel3, C2 = Panel2,** при третьем **C1 = Panel3, C2 = Panell.** Функция должна возвращать **true,** если компонент C1 должен выравниваться прежде, чем C2 (здесь в справке C++Builder 6 тоже ошибка). В классе **TWinControl** функция **CustomAlignInsert-**

Before всегда возвращает **false,** так что по умолчанию выравнивание компонентов происходит в той последовательности, в которой они расположены в свойстве **Controls.** Но можно переписать эту виртуальную функцию. Например, если в переписанном варианте всегда возвращать **true,** то последовательность выравнивания изменится на противоположную.

Пример

Пусть вы проектируете форму, которая содержит:

1. панель **Panell,** на которой размещается, в частности, список **ListBox1**
2. панель **Pane12,** на которой размещаются какие-то надписи вверху окна формы
3. компонент **Memo1,** в котором будут редактироваться тексты

Так как размер списка **ListBoxl** может возрастать по мере его заполнения, желательно дать пользователю возможность увеличивать его высоту, изменяя высоту окна формы. Панель **Pane12** всегда должна заполнять верхнюю часть окна формы. А площадь окна **Memol** должна изменяться пропорционально изменению окна формы, чтобы пользователь мог увеличить область редактирования.

В этом примере логично для панели **Pane12** задать значение **Align,** равным **alTop,** чтобы ширина панели автоматически менялась с изменением ширины окна. Панель займет всю верхнюю часть формы, а ее высота будет такой, какую вы установите. Для панели **Panell** следует задать значение **Align,** равным **alLeft,** так как увеличение ее ширины не имеет смысла, а увеличение высоты позволяет увидеть большую часть списка **ListBox1.** Компонент **Panell** займет всю левую часть клиентской области, оставшейся свободной после размещения **Pane12.** Для компонента **Memol** следует задать значение **Align,** равным **alClient,** что позволит компоненту увеличиваться и в высоту, и в ширину, увеличивая площадь редактирования. При этом компонент **Memol** займет всю площадь клиентской области, оставшуюся после размещения **Panell и Pane12.**

Подобное традиционное использование свойства **Align** вряд ли может вызвать затруднения. Поэтому рассмотрим далее только пример заказного выравнивания.

Пусть у вас имеются на форме две панели **Panell и Pane12,** и вы хотите, чтобы при любых изменениях размеров окна панель **Panell** занимала всю левую половину формы, а **Panel2** — всю правую. Вы можете это сделать следующим образом. Расположите панели на форме любым образом. Задайте в обеих панелях **Align = alCustom.** Далее напишите следующий код:

заголовочный файл:

```
...
class TForm1 : public TForm
{
_ published:        // IDE-managed Components
    TPanel *Panel1;
    TPanel *Panel2;
  void __fastcall CustomAlignPosition(TControl* Control,
                    int &NewLeft, int &NewTop, int &NewWidth,
                    int &NewHeight, Types::TRect &AlignRect,
                    const TAlignInfo &AlignInfo);
private:  // User declarations
public:           // User declarations
    __fastcall TForm1(TComponent* Owner);
};
...
```

файл реализации:

```
...

void __fastcall TForm1::CustomAlignPosition(TControl* Control,
                    int &NewLeft, int &NewTop, int &NewWidth,
```

```
                        int &NewHeight, Types::TRect &AlignRect,
                        const TAlignInfo &AlignInfo)
{
 Panel1->Left = 0;
 Panell->Top = 0;
 Panel2->Top = 0;
 NewWidth = ClientWidth / 2;
 NewHeight = ClientHeight;
 if (Control == Panel2)
   NewLeft = ClientWidth / 2;
}
```

Вы ввели в класс формы объявление перегруженной функции **CustomAlignPosition.** В ее реализации вы задали для каждой панели ширину, равную половине ширины **ClientWidth** клиентской области формы, и высоту, равную высоте **ClientHeight** клиентской области формы. Кроме того, вы сдвинули на половину ширины **ClientWidth** левую координату панели **Panel2.**

Впрочем, рассмотренный пример можно и, возможно, проще решить обработчиком события формы **OnResize** (см. соответствующий раздел в гл. 5).

Alignment — свойство TField

Определяет выравнивание значений полей при их отображении в компонентах, связанных с данными.

Класс *TField*

Определение

```
enum TAlignment { taLeftJustify, taRightJustify, taCenter };
__property Classes::TAlignment Alignment
```

Описание

Свойство **Alignment** определяет выравнивание значений полей при их отображении в компонентах, связанных с данными. Возможные значения **Alignment:**

taCenter	Выравнивание в горизонтальном направлении по центру компонента
taLeftJustify	Выравнивание по левому краю компонента
taRightJustify	Выравнивание по правому краю компонента

Anchors

Определяет привязку данного компонента к родительскому при изменении размеров последнего.

Класс *TControl*

Определение

```
enum TAnchorKind { akLeft, akTop, akRight, akBottom };

typedef Set<TAnchorKind, akLeft, akBottom> TAnchors;

__property TAnchors Anchors
```

Описание

Свойство **Anchors** введено начиная с C++Builder 4. Оно определяет привязку данного компонента к родительскому при изменении размеров последнего. Свойство представляет собой множество типа **Set**, которое может содержать следующие элементы:

akTop	Компонент привязан к верхнему краю родительского
akLeft	Компонент привязан к левому краю родительского
akRight	Компонент привязан к правому краю родительского
akBottom	Компонент привязан к нижнему краю родительского

Если в множестве **Anchors** присутствуют привязки к противоположным сторонам родительского компонента, то при изменении родительского компонента происходит растяжение или сжатие данного компонента, поскольку расстояния от сторон родительского компонента выдерживаются. Сжатие может происходить вплоть до полного уничтожения изображения данного компонента. Для компонента **TPaintBox,** привязанного к противоположным сторонам родительского компонента, при изменении размеров родительского компонента изображение стирается и наступает событие **OnPaint.**

Примеры

1.
```
Button1->Anchors.Clear();
Button1->Anchors << akLeft << akBottom;
```

Первый из этих операторов очищает множество **Anchors** кнопки **Button1,** удаляя из него первоначальные установки. Второй оператор привязывает кнопку **Button1** к левому и нижнему краям окна. При изменении размеров окна кнопка будет перемещаться, сохраняя установленное расстояние от левого и нижнего краев формы.

2. Задание компоненту списку **TListBox** свойства **Anchors,** равного **[akLeft, akTop, akBottom],** приведет к тому, что при изменении высоты окна будут поддерживаться постоянными расстояния верхнего и нижнего краев компонента соответственно от верхнего и нижнего краев окна. Таким образом, увеличивая высоту окна пользователь может увеличивать число строк, видимых в списке без прокрутки. При этом необходимые с точки зрения эстетики расстояния до краев окна будут поддерживаться автоматически.

3.
```
Button1->Anchors.Clear();
Button1->Anchors << akLeft << akBottom << akRight;
```

В этом случае кнопка привязана к левому, нижнему и правому краям формы. При изменении высоты окна кнопка будет перемещаться синхронно с нижним краем окна. А при изменении горизонтального размера окна кнопка будет растягиваться или сжиматься по горизонтали, что, конечно, будет создавать ужасный зрительный эффект. При сжатии кнопка может сжаться до нуля, после чего ее изображение исчезнет.

AsBoolean, AsCurrency, AsDateTime, AsFloat, AsInteger, AsString, AsVariant

Свойства, осуществляющие взаимное преобразование различных типов значений полей.

Класс *TField*

Определения

```
__property bool AsBoolean
__property System::Currency AsCurrency
__property System::TDateTime AsDateTime
__property double AsFloat;
__property int AsInteger
__property AnsiString AsString
__property System::Variant AsVariant
```

Описание

Свойства **AsBoolean, AsCurrency, AsDateTime, AsFloat, Aslnteger, AsString, AsVariant** осуществляют в классах-наследниках **TField** перевод одного типа значений в другой. Например, свойство **AsString** переводит тип любого поля в строку при чтении значения поля и осуществляет обратный перевод строки в тип поля при записи значения поля. Аналогичные преобразования осуществляют другие свойства для булевых значений **(AsBoolean),** монетарных **(AsCurrency),** для дат и времени **(AsDateTime),** для действительных (AsFloat) и целых (Aslnteger) чисел, для значений типа **Variant (AsVariant).**

Примеры

Если компонент **Tablel** связан с таблицей, содержащей строковое поле Dep, целочисленное поле **Year_b** и булево поле Sex, вы можете написать

```
EDep->Text = Table1->FieldByName("Dep")->AsString;
EYear->Text = Table1->FieldByName("Year_b")->AsString;
ESex->Text = Table1->FieldByName("Sex")->AsString;
```

и в окна редактирования **EDep, EYear и ESex** будут занесены в текстовом виде значения в текущей записи полей **Dep, Year_b** и **Sex,** хотя поле Dep имеет тип строки, поле **Year_b** — целое значение, а поле Sex — булево. Если для поля Sex вы не задавали значений **DisplayValues,** то в окне редактирования **ESex** будут отображены значения "true" или "false". Если же вы задали значения свойства **DispIayValues,** например "м;ж" или "мужск;женск", то отобразятся именно эти заданные значения.

То же свойство **AsString** работает и как обратное преобразование типов. Продолжая предыдущий пример вы можете после того, как пользователь отредактировал значения в полях редактирования **EDep, EYear** и **ESex,** внести эти значения в текущую запись, например, следующим кодом:

```
Table1->Edit();
Table1->FieldByName("Dep")->AsString = EDep->Text;
Table1->FieldByName("Year_b")->AsString = EYear->Text;
Table1->FieldByName("Sex")->AsString = ESex->Text;
Table1->Post();
```

Для полей **Year_b** и Sex текст будет преобразован соответственно в целое и булево значение. При этом не обязательно, чтобы пользователь в окне **ESex** писал полностью обозначение пола сотрудника. Ему достаточно написать только первую букву: "t" или "f", если отображаемые значения **true и false,** и "м" или "ж", если отображаемые значения "мужск" и "женск".

Attributes

Атрибуты описания поля.

Класс *TFieldDef*

Определение

```
enum TFieldAttribute {faHiddenCol, faReadonly, faRequired,
                      faLink, faUnNamed, faFixed};

typedef Set<TFieldAttribute, faHiddenCol, faFixed> TFieldAttributes;

__property TFieldAttributes Attributes
```

Описание

Свойство **Attributes** содержит атрибуты объекта **TFieldDef.** Является множеством, которое может содержать следующие значения:

faHiddenCol	Внутренний флаг невидимых столбцов
faReadonly	Определяет свойство «только для чтения»

faRequired	Определяет поле, которое является обязательным в каждой записи
faLink	Внутренний флаг вложенного набора данных
faUnNamed	Указывает объект TObjectField без имени во вложенной таблице. Значение только для внутреннего использования
faFixed	Указывает, что поле имеет фиксированный размер

AttributeSet

Имя множества атрибутов словаря данных, применяемого к данному полю.

Класс *TField*

Определение

```
__property System::AnsiString AttributeSet
```

Описание

Свойство **AttributeSet** определяет имя множества атрибутов словаря данных, применяемого к данному полю. Множество атрибутов определяет формат и отображение поля в компонентах, связанных с данными, в процессе проектирования, а также устанавливают ограничения значений поля, значения по умолчанию и многое другое.

Множества атрибутов хранятся в словаре атрибутов. Когда во время проектирования объект поля связывается с конкретным множеством, его имя заносится в свойство **AttributeSet.**

AutoCalcFields

Определяет режим автоматической генерации событий **OnCalcFields.**

Класс *TDataSet*

Определение

```
__property bool AutoCalcFields
```

Описание

Свойство **AutoCalcFields,** установленное в **true** (значение по умолчанию), определяет, что события **OnCalcFields.** связанные с пересчетом вычисляемых полей, генерируются каждый раз, когда:

- Открывается набор данных.
- Набор данных (его свойство **State**) переводится в состояние **dsEdit.**
- Фокус перемещается между визуальными компонентами, связанными с данными, или перемещается по столбцам связанной с данными таблицы, если в записи были сделаны изменения.
- Отыскивается запись в базе данных.

Чтобы сократить частоту пересчетов по событиям **OnCalcFields,** можно установить **AutoCalcFields в false.**

AutoGenerateValue

Показывает, может ли значение поля генерироваться автоматически в базе данных на сервере.

Класс *TField*

Определение

```
enum TAutoRefreshFlag {arNone, arAutoInc, arDefault};
__property TAutoRefreshFlag AutoGenerateValue
```

Описание

Свойство **AutoGenerateValue** определяет, может ли значение поля генерироваться и обновляться автоматически в базе данных на сервере. Такая автоматическая генерация значений присуща полям с автоматическим нарастанием значений (autoincrement) и полям со значением по умолчанию.

Обычно при пересылке записи в базу данных значения таких полей не обновляются автоматически. Для изменения значения **Value** надо применять к набору данных метод **Refresh.** Если же в наборе данных свойство **AutoRefresh** установлено в **true,** то обновление данных происходит автоматически. Однако некоторые драйверы баз данных не могут определить, значения каких полей могут автоматически создаваться сервером. Свойство **AutoGenerateValue** позволяет снабдить подобные драйверы необходимой информацией.

Это свойство может принимать значения:

arNone	Поле не имеет значения по умолчанию и не является автоматически нарастающим. Такие поля автоматически не обновляются
arAutoInc	Поле является автоматически нарастающим
arDefault	Поле имеет значение по умолчанию, задаваемое сервером

Задание **AutoGenerateValue** необходимо не всегда. Многие драйверы, использующие метаданные, нормально работают с любыми полями, даже если **AutoGenerateValue = arNone.** Но для надежности и облегчения перехода с одной платформы баз данных на другую лучше все-таки задавать соответствующие значения **AutoGenerateValue.**

Изменение значения **AutoGenerateValue** при открытом наборе данных вызывает исключение.

При использовании наборов данных, не основанных на BDE, а также при **AutoRefresh = false** значение **AutoGenerateValue** игнорируется.

AutoMerge

Определяет, должно ли главное меню вторичной формы объединяться с меню главной формы.

Класс *TMainMenu*

Определение

```
__property bool AutoMerge
```

Описание

Если требуется, чтобы меню вторичных форм объединялись с меню главной формы, то в каждой такой вторичной форме надо установить **AutoMerge в true.** При этом свойство главной формы должно оставаться в **false.** Способ объединения меню определяется свойствами **GroupIndex** элементов меню **Items** типа **TMenuItem.**

В приложениях MDI объединение меню осуществляется автоматически независимо от значения свойства **AutoMerge.**

AutoRefresh

Определяет, обновляются ли автоматически значения полей, вычисляемые сервером.

Класс *TDBDataSet*

Определение

```
__property bool AutoRefresh
```

Описание

Свойство **AutoRefresh** определяет, обновляются ли автоматически значения полей, вычисляемые сервером. К таким полям относятся поля с автоматически нарастающим значением (autoincrement) и поля со значениями по умолчанию. Если **AutoRefresh = false,** то значения таких полей при пересылке записи в базу данных не обновляются. Для их обновления надо явным образом вызывать метод **Refresh.** Если **AutoRefresh = true,** то обновление таких полей производится автоматически и вызов **Refresh** не требуется.

Некоторые драйверы баз данных не могут сами распознать поля с автоматическим нарастанием значения и поля, имеющие значения по умолчанию. В подобных случаях надо задать для всех полей, значения которых должны обновляться, свойство **AutoGenerateValue.** Это свойство снабдит драйверы необходимой информацией. Свойство **AutoGenerateValue** надо задавать также для полей наборов данных, отображающих запросы SQL и хранимые процедуры.

AutoSelect

Указывает, будет ли выделяться весь текст, когда элемент получает фокус.

Класс *TCustomEdit*

Определение

```
__property bool AutoSelect
```

Описание

Свойство **AutoSelect,** установленное в **true,** показывает, что при получении элементом фокуса весь текст окажется выделенным. Это свойство относится только к элементам редактирования одной строки текста.

Свойство **AutoSelect** имеет смысл устанавливать в **true,** если по условиям работы пользователь будет скорее заменять текст, имеющийся в окне, чем дополнять его.

AutoSize

Определяет, будет ли высота элемента автоматически адаптироваться к размеру символов текста.

Классы *TCustomEdit, TCustomLabel, TImage и др.*

Определение

```
__property bool AutoSize
```

Описание

При свойстве **AutoSize,** установленном в **false,** размеры компонента фиксированы. При **AutoSize,** установленном в **true,** в классах-наследниках **TCustomLabel** (метках), в **TImage** и ряде других компонентов др. ширина и высота компонента автоматически адаптируется к размерам текста или изображения. В классах-наследниках **TCustomEdit** (окнах редактирования) ширина компонента остается фиксированной, а высота изменяется так, чтобы высота клиентской области соответствовала высоте текста. Например, высота элемента меняется при изменении шрифта или стиля бордюра.

Bitmap

Определяет внешний нестандартный шаблон размером 8 на 8 пикселов, который использует для заполнения кисть **Brush.**

Класс *TBrush*

Определение

```
__property TBitmap* Bitmap
```

Описание

Свойство кисти Bitmap указывает на объект типа TBitmap, в который загружен шаблон размером 8 на 8 пикселов, используемый для заполнения кистью **Brush.**

Если для кисти задан шаблон **BitMap,** то заполнение производится именно этим шаблоном, независимо от значения свойства кисти Style. Шаблон **BitMap** может создаваться в процессе выполнения приложения или, например, загружаться из файла.

Если размер изображения превышает 8 на 8 пикселов, то в качестве шаблона будет использоваться его левая верхняя часть размером 8 на 8.

Изменение изображения в объекте **TBitmap** не влияет на шаблон, пока не произведено повторное присваивание свойству **Bitmap.**

После окончания работы с шаблоном объект **TBitmap** следует удалить из памяти, так как автоматически это не делается.

Пример

```
Graphics::TBitmap *MyBitmap = new Graphics::TBitmap;

try
{
 MyBitmap->LoadFromFile("MyBitmap.bmp");
 Image1->Canvas->Brush->Bitmap = MyBitmap;
 ...
}
__finally
{
 Image1->Canvas->Brush->Bitmap = NULL;
 delete MyBitmap;
}
```

В этом примере создается объект **MyBitmap** типа **TBitmap** и в него загружается битовая матрица из файла с именем «MyBitmap.bmp». Затем свойству **Image1->Canvas->Brush->Bitmap** присваивается указатель на этот объект. После этого загруженный шаблон можно использовать для заполнения фигур на канве **Imagel.** В конце кода свойству **BitMap** присваивается значение **NULL,** после чего заполнение опять начинает определяться свойством **Style.** Затем объект **MyBitmap** уничтожается, чтобы освободить занимаемую им память.

BlockReadSize

Свойство, определяющее число записей, помещаемых при чтении в буфер.

Класс *TBDEDataSet*

Определение

```
__property int BlockReadSize
```

Описание

Свойство **BlockReadSize** определяет число записей, помещаемых при чтении в буфер. Заданное число записей читается, заносится в буфер, и, пока указатель таблицы не вышел за пределы прочитанных записей, новое чтение не производится. Соответственно, не осуществляется и повторное отображение в компонентах отображения данных, что ускорят работу и позволяет избежать неприятных мерцаний. Установка **BlockReadSize** в 0 запрещает чтение блоков записей. Задание **BlockReadSize** > 0 переводит состояние набора данных **State** в **dsBlockRead** и ускоряет работу приложения.

Bof

Свойство указывает, находится ли курсор на первой записи.

Класс *TDataSet*

Определение

```
__property bool Bof
```

Описание

Свойство Bof определяет, находится ли курсор на первой записи. Значение Bof устанавливается в **true:**

- При открытии набора данных
- В результате выполнения метода First
- При ошибке выполнения метода **Prior** из-за того, что курсор уже расположен на первой записи
- В остальных случаях **Bof = false.**

Пример

Следующий пример показывает типичный способ организации цикла по записям набора данных. Пусть в вашем приложении имеется выпадающий список с именем **CBdep,** который вы хотите заполнить данными, содержащимися в полях Dep всех записей таблицы, соединенной с компонентом **Table1.** Причем заполняться он должен в обратном порядке — от последней записи к первой. Это можно сделать следующим кодом:

```
CBdep->Clear();
Table1->Last();
while (! Table1->Bof)
{
 CBdep->Items->Add(Table1->FieldByName("Dep")->AsString);
 Table1->Prior();
}
```

Первый оператор кода очищает список **CBdep.** Второй — устанавливает курсор таблицы на последнюю запись. Далее следует цикл по всем записям, пока не достигнута первая, что проверяется выражением **Table1->BOF.** Для каждой записи в список заносится значение поля Dep, после чего методом **Prior** курсор перемещается к предыдущей записи.

Bookmark

Свойство определяет текущую закладку набора данных.

Класс *TDataSet*

Определение

```
__property AnsiString Bookmark;
```

Описание

Свойство **Bookmark** определяет текущую закладку набора данных. Свойство **Bookmark** можно устанавливать, для запоминания текущей записи, а затем после перемещения по набору данных читать **Bookmark** для быстрого возврата к той же записи.

О методах работы с закладками см. в гл. 4 раздел «BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark».

Пример

В следующем примере делается закладка на текущей записи и после цикла по набору данных осуществляется возврат на запись, которая до этого была текущей.

```
TBookmark SavePlace;
// Закладка на текущей записи
 SavePlace = Query1->GetBookmark();
// Цикл по записям
```

Описание

Свойство **Break** используется в длинных меню, чтобы разбить список разделов на несколько столбцов. Возможные значение **Break:**

mbNone	Отсутствие разбиения меню. Это значение принято по умолчанию
mbBarBreak	В меню вводится новый столбец разделов и данный раздел является верхним в новом столбце. Столбец отделяется от предыдущего полосой
mbBreak	В меню вводится новый столбец разделов и данный раздел является верхним в новом столбце. Столбец отделяется от предыдущего пробелами

Brush

Определяет цвет и стиль заполнения фона окна.

Класс *TWinControl*

Доступ *только для чтения*

Определение

```
__property Graphics::TBrush* Brush
```

Описание

Свойство **Brush** (кисть) присуще многим оконным объектам, включая **Canvas.** Его можно читать, чтобы определить цвет и стиль заполнения фона окна. Это свойство только для чтения. Однако атрибуты объекта **Brush** можно изменять, используя свойства **Color** и **Style**. Кроме того, все свойства объекта могут быть изменены методом **Assign.**

Тип TBrush определяет свойства и методы объекта **Brush.** См. информацию о свойствах и методах, а также примеры в разд. «TBrush» гл. 1.

CacheBlobs

Показывает, кэшируются ли поля BLOB в памяти.

Класс *TBDEDataSet*

Определение

```
__property bool CacheBlobs
```

Описание

Свойство **CacheBlobs** показывает, кэшируются ли поля BLOB в памяти. Кэширование позволяет повысить производительность при перемещении по записям, когда приложение отображает образы полей BLOB. По умолчанию **CacheBlobs = true,** т.е. кэширование производится. Если приложение не предусматривает постоянное отображение полей BLOB, то целесообразно установить **CacheBlobs = false,** чтобы сэкономить ресурсы памяти.

CachedUpdates

Определяет, кэшируются ли изменения набора данных.

Класс *TBDEDataSet*

Определение

```
__property bool CachedUpdates
```

Описание

Свойство **CachedUpdates** определяет, кэшируются ли изменения набора данных (при значении **true),** или кэширование не производится. При кэшировании все изменения данных, вставка новых записей, удаление существующих записей, т.е. все манипуляции с данными, проводимые пользователем, сначала делаются не в самой базе данных, а запоминаются в памяти во временной, виртуальной таблице. И только по особой команде после всех проверок правильности вносимых в таблицу данных пользователю предоставляется возможность или зафиксировать все эти изменения в базе данных в рамках одной транзакции, или отказаться от этого и вернуться к тому состоянию, которое было до начала редактирования.

При работе в сети кэширование позволяет уменьшить число транзакций, сократить их длительность и разгрузить сеть. Дополнительным преимуществом кэширования является невозможность других приложений увидеть изменения данных, происходящие в процессе их редактирования данным приложением. С другой стороны, если пока шло редактирование, другие приложения внесли изменения в базу данных, то при переносе кэшированных изменений в базу данных возможны конфликты.

Альтернативой кэширования при работе в сети является использование клиентского набора данных и компонента-провайдера.

Calculated

Определяет, является ли поле вычисляемым в обработчике события **OnCalcFields.**

Класс *TField*

Определение

```
__property bool Calculated
```

Описание

Свойство **Calculated** определяет, является ли поле вычисляемым в обработчике события **OnCalcFields.** По умолчанию **Calculated = false. При Calculated = true** значение поля определяется обработчиком события **OnCalcFields.** Значения таких полей не сохраняются в базе данных.

Значение **Calculated** устанавливается во время проектирования в Редакторе Полей и не может изменяться во время выполнения.

Надо представлять себе различие между свойствами **Calculated и InternalCalcField.** Свойство **Calculated** указывает на расчет значения поля в обработчике события **OnCalcFields.** А свойство **InternalCalcField** указывает на то, что значение поля вычисляется или сервером SQL, или Borland Database Engine (BDE).

CanModify — свойство TDataSet

Определяет возможность редактирования набора данных.

Класс *TDataSet*

Доступ *только для чтения*

Определение

```
__property bool CanModify
```

Описание

Свойство **CanModify** определяет, возможно ли редактирование набора данных, и управляет этой возможностью. Если значение **CanModify** равно **true,** то редактирование (т.е. изменение, добавление, удаление записей) возможно. Если значение **CanModify** равно **true,** то набор данных является просто таблицей только для чтения.

Значение **CanModify** автоматически устанавливается в момент открытия таблицы в зависимости от значения свойства **ReadOnly**. Если значение **ReadOnly** равно **true,** то значение **CanModify** устанавливается в **false.** Но значение **CanModify** автоматически устанавливается в **false** также в случаях, когда:

- Другое приложение заблокировало набор данных.
- Таблица базы данных спроектирована только для чтения.

Однако значение **CanModify,** равное **true,** еще не гарантирует возможность вставлять или изменять записи. Это может зависеть еще от ряда факторов, например, от привилегий доступа SQL.

CanModify — свойство TField

Определяет, может ли данное поле модифицироваться.

Класс *TField*

Доступ *только для чтения*

Определение

```
__property bool CanModify
```

Описание

Свойство **CanModify** определяет, может ли данное поле модифицироваться. Проверка этого свойства позволяет узнать, можно ли редактировать его значение. **CanModify** равно **true,** только если **ReadOnly = false** и набор данных имеет доступ для записи этого поля базы данных.

Canvas

Поверхность (холст, канва) для рисования во многих компонентах.

Классы *TForm, TImage, TBitMap, TPaintBox* и другие

Доступ *только для чтения*

Определение

```
__property Graphics::TCanvas* Canvas
```

Описание

Свойство **Canvas** типа **TCanvas** используется для рисования пером Реп и кистью **Brush** для модификации изображения, наложения друг на друга нескольких изображений.

В компоненте типа **TImage** канва может использоваться только в случае, если в свойство **Picture** загружена битовая матрица или ничего не загружено. Если в **Picture** находится объект типа, отличного от **TBitMap,** то при обращении к **Canvas** генерируется исключение **EInvalidOperation.** Если же в **Picture** находится битовая матрица, то **Canvas** можно использовать для редактирования изображения, например, для добавления в изображение надписей методом **TextOut.**

Capacity

Количество элементов массива указателей, которые могут храниться в объектах списков.

Классы *TList, TStringList*

Определение

```
__property int Capacity
```

Описание

Свойство **Capacity** можно задавать, чтобы определить число указателей, которые могут храниться в объектах списков классов **TList** и **TStringList**. Если при увеличении **Capacity** оказывается, что ресурсы памяти исчерпаны, генерируется исключение **EOutOfMemory**.

Свойство **Capacity** отличается от близкого ему свойства **Count. Capacity** показывает, сколько указателей может храниться в массиве, т.е. емкость списка, а **Count** показывает, сколько указателей в действительности хранится в массиве. Поэтому значение **Capacity** всегда не меньше значения **Count.**

При добавлении в массив нового элемента **Count** автоматически увеличивается на 1. Если при этом значение **Count** превышает значение **Capacity, то Capacity** тоже автоматически увеличивается, причем с запасом (обычно запас равен 3). Так что свойство **Capacity** вообще можно нигде в приложении не задавать. Смысл задания **Capacity** заключается только в том, чтобы предотвратить слишком частое перераспределение памяти при добавлении новых элементов списка.

При удалении элементов списка **Count** автоматически уменьшается, а значение **Capacity** остается неизменным. Таким образом, при сокращении длины списка получаются излишние затраты памяти. В эти случаях целесообразно уменьшить Capacity до значения **Count** (если не планируется в ближайшее время нового увеличения числа элементов).

Если задать значение **Capacity** меньшее, чем **Count**, генерируется исключение **EListError** с сообщением: «List capacity out of bounds [...]» («Емкость списка вне допустимых пределов»). В квадратных скобках указывается значение **Count**.

Очистка списка методом **Clear** сбрасывает **Capacity** на нуль.

Значение **Capacity** может быть увеличено методом **Expand.**

Пример

```
TList *List = new TList ();
...
List->Clear();
List->Capacity = 5;
// Тут List->Count = 0 и List->Capacity = 5
...
for(int I = 1; I <= 5; I++)
  List->Add(...);
// Тут List->Count = 5 и List->Capacity = 5
...
for (int I = 0; I <= 2; I++)
 List->Delete(I);
// Тут List->Count = 2 и List->Capacity = 5
...
List->Capacity = List->Capacity - 3;
// Тут List->Count = 2 и List->Capacity = 2
...
```

Этот пример будет выполнять те же самые функции и без операторов

```
List->Capacity = 5;
```

И

```
List->Capacity = List->Capacity - 3;
```

Но первый из этих операторов экономит время при увеличении длины списка, а второй — сокращает затраты памяти.

Caption

Определяет строку текста, идентифицирующую компонент для пользователя.

Класс *TControl*

Определение

```
__property System::AnsiString Caption;
```

Описание

Свойство **Caption** связывает с компонентом некоторую строку текста, поясняющую его назначение. Чаще всего это надписи на кнопках, метках, панелях, тексты разделов меню и т.д.

По умолчанию **Caption** совпадает с именем компонента — свойством Name и изменяется при его изменении.

Для разделов меню и кнопок можно в свойстве **Caption** указать символы быстрого доступа. Для этого перед соответствующим символом надо поставить символ амперсанда — &. Следующий за амперсандом символ будет отображаться подчеркнутым и будет являться символом быстрого доступа: при выполнении приложения нажатие клавиши Alt + клавиши данного символа будет эквивалентно выбору соответствующего раздела меню или нажатию соответствующей кнопки.

Если в текст строки **Caption** надо ввести символ амперсанда, его надо повторить дважды: &&.

Для разделов меню задание значения **Caption,** равного символу "-", означает разделительную черту.

Примеры

1. Примеры задания свойства **Caption** в процессе проектирования меню:

Задание	Отображение
&Файл	Файл
Фор&мат	Формат
Сохранить &как	Сохранить как

2. Пример использования метки для отображения результата ответа пользователя:

```
if (Edit1->Text = "1")
  Label1->Caption = "Ответ правильный";
else Label1->Caption = "Ответ ошибочный";
```

3. Примеры присваивания свойству **Caption** значений, отличных от строк:

```
Label1->Caption = 5;
Label2->Caption = 5.1;
Label3->Caption = 'A';
```

Charset

Свойство типа **TFont.** определяющее набор символов шрифта.

Класс *graphics*

Определение

```
__property TFontCharset Charset
```

Описание

Свойство **Charset** определяет набор символов шрифта — объекта типа **TFont.** Каждый вид шрифта, определяемый его именем, поддерживает один или более наборов символов. Какие именно значения **Charset** поддерживает тот или иной шрифт можно установить из документации на него или экспериментальным путем. Для шрифтов, поддерживающих несколько наборов символов, важно правильно задать **Charset.**

В C++Builder предопределено много констант, соответствующих стандартным наборам символов. Большинство из них, относящихся к японскому, корейскому, китайскому и другим языкам, вряд ли представляют интерес для наших читателей. Поэтому ниже приводится сокращенная таблица этих констант.

Константа	Значение	Описание
ANSI_CHARSET	0	Символы ANSI.
DEFAULT_CHARSET	1	Задается по умолчанию. Шрифт выбирается только по его имени Name и размеру Size. Если описанный шрифт недоступен в системе, то Windows заменит его другим шрифтом.
SYMBOL_CHARSET	2	Стандартный набор символов.
MAC_CHARSET	77	Символы Macintosh. Недоступны для NT 3.51.
GREEK_CHARSET	161	Греческие символы. Недоступны для NT 3.51.
RUSSIAN_CHARSET	204	Символы кириллицы. Недоступны для NT 3.51.
EASTEUROPE_CHARSET	238	Включает диакритические знаки (знаки, добавляемые к буквам и характеризующие их произношение) для восточно-европейских языков. Недоступны для NT 3.51.
OEM_CHARSET	255	Зависит от кодовой таблицы операционной системы.

По умолчанию в объекте типа **TFont** задается значение **Charset,** равное **DEFAULT_CHARSET.** Для имен шрифтов, принятых в C++Builder по умолчанию, это обычно нормальный вариант. Но в ряде случаев полезно для отображения русских текстов с другими шрифтами заменить это значение на **RUSSIAN_CHARSE.** Это позволит отобразить символы кириллицы для тех шрифтов, для которых при **DEFAULT_CHARSET** символы кириллицы не отображаются нормально.

Пример
См. пример 4 в разд. **Font.**

ChildDefs

Объект **TFieldDef,** содержащий массив дочерних поле.

Класс *TFieldDef*

Определение

```
__property TFieldDefs * ChildDefs
```

Описание
Если поле является объектом, подобным **TADTField или TArrayField,** то дочерние поля хранятся как объекты массива **TFieldDefs,** на который указывает свойство **ChildDefs.**

ClientHeight

Высота клиентской области в пикселах.

Класс *TControl*

Определение

```
__property int ClientHeight
```

Описание

Свойство **ClientHeight** может использоваться для чтения или изменения высоты клиентской области. Чтение **ClientHeight** необходимо, чтобы изменять местоположение или размеры компонентов, расположенных в клиентской области, при изменении размеров компонента-контейнера (см. примеры в разд. **BoundsRect**).

Эквивалентно по значению свойству **ClientRect.Bottom (см. ClientRect).**

В классе **TControl** значение ClientHeight равно значению **Height.** Но в производных классах оно обычно меньше **Height.** Например, для формы значение **ClientHeight** может уменьшаться из-за бордюра, полосы заголовка, полосы меню, полосы прокрутки.

Пример

Пусть вы проектируете форму, содержащую панель **Panell,** на которой размещается список **ListBoxl,** причем во время выполнения высота этой формы может изменяться. Тогда обеспечить синхронное изменение размера окна списка можно, вставив в событие **OnResize** формы или этой панели оператор:

```
ListBoxl->Height = Panel1->ClientHeight - ListBox1->Top - 20;
```

ClientOrigin

Координаты положения на экране левого верхнего угла клиентской области компонента.

Класс *TControl*

Доступ *только для чтения*

Определение

```
struct TPoint // полное определение см. в гл. 1
{
 int  x;
 int  y;
};

__property Windows::TPoint ClientOrigin
```

Описание

Свойство **ClientOrigin** возвращает экранные координаты X и Y левого верхнего угла клиентской области компонента. Горизонтальная координата X и вертикальная координата Y хранятся в структуре типа **TPoint.** Началом координат считается левый верхний угол экрана.

Экранные координаты **ClientOrigin** для компонентов, не являющихся потомками класса **TWinControl** (т.е. не являющихся окнами), определяются как экранные координаты родительского компонента (компонента-контейнера), сложенные со значениями свойств Left и Top. Если компонент не имеет родителя, то при попытке чтения **ClientOrigin** генерируется исключение **EInvalidOperation.**

ClientRect

Определяет координаты углов клиентской области компонента.

Класс *TControl*

Доступ *только для чтения*

Определение

```
struct TRect // полное определение см. в гл. 1
{
 int Left;
 int Top;
 int Right;
 int Bottom;
};

__property Windows::TRect ClientRect
```

Описание

Свойство **ClientRect** возвращает структуру типа TRect. характеризующую прямоугольную клиентскую область компонента. При этом поля структуры Top и Left устанавливаются в нуль, а поля **Right и Bottom** определяются соответственно значениями свойств **ClientWidth и ClientHeight**. Так что **ClientRect** эквивалентна функции **Rect** вида **Rect(0, 0, ClientWidth, ClientHeight).**

ClientWidth

Горизонтальный размер клиентской области в пикселах.

Класс *TControl*

Определение

```
__property int ClientWidth
```

Описание

Свойство **ClientWidth** может использоваться для чтения или изменения горизонтального размера клиентской области. Чтение **ClientWidth** необходимо, чтобы изменять местоположение или размеры компонентов, расположенных в клиентской области, при изменении размеров компонента-контейнера (см. подобные примеры в разд. «BoundsRect»).

Эквивалентно по значению свойству **ClientRect.Right (см. ClientRect).**

В классе **TControl** значение **ClientWidth** равно значению **Width.** Но в производных классах оно обычно меньше **Width** за счет бордюра, полосы прокрутки и иных элементов оформления.

ClipRect

Определяет доступную область рисования на канве и область, подлежащую перерисовке при обработке события **OnPaint.**

Класс *TCanvas*

Доступ *только для чтения*

Определение

```
struct TRect // полное определение см. в гл. 1
{
 int Left;
 int Top;
 int Right;
 int Bottom;
};

__property Windows::TRect ClipRect
```

Описание

Свойство канвы **ClipRect** определяет доступную область рисования на канве и область, нуждающуюся в перерисовке. Вне области **ClipRect** рисовать невозможно.

При обработке события формы **OnPaint** это свойство определяет ту часть канвы, вне которой перерисовка не требуется. Использование этого свойства позволяет сократить затраты времени на перерисовку.

Color

Цвет фона компонента, цвет текста объекта **TFont и др.**

Класс *TControl*

Определение

```
__property Graphics::TColor Color
```

Описание

Свойство **Color** определяет цвет фона компонента или цвет текста объекта **TFont.** Значение цвета может задаваться как значение, определяющее интенсивности красного, зеленого и синего цветов в формате RGB, или равным одной из предопределенных в C++Builder констант (см. варианты задания цвета и некоторые рекомендации по выбору цвета в описании типа **TColor в гл.** 1).

CommandText — свойство TCustomClientDataSet

Строка запроса SQL.

Класс *TCustomClientDataSet*

Объявление

```
__property AnsiString CommandText
```

Описание

Свойство **CommandText** определяет данные, которые клиентский набор хочет получить от провайдера. Это или строка запроса SQL, который должен выполнить сервер, или имя таблицы, или имя хранимой процедуры.

Если клиентский набор использует внутренний провайдер, задание значения **CommandText** необходимо. При этом, например, в наборе **TSQLClientDataSet** свойство **CommandType** определяет, что именно задано в **CommandText:** запрос, имя таблицы, или имя хранимой процедуры.

При использовании внешнего провайдера, задание **CommandText** не обязательно. Это свойство будет срабатывать только в том случае, если в провайдере в свойстве **Options** включена опция **poAllowCommandText.** Тогда значение **CommandText** будет заменять то свойство, с помощью которого набор данных сервера связывается с данными.

Значение **CommandText** пересылается провайдеру при открытии набора данных или при выполнении метода **Execute.**

Возможные типы запроса **CommandText** определяются тем серверным компонентом, с которым связывается клиентский набор. Если, например, в качестве серверного набора используется **TTable,** то значение **CommandText** должно указывать таблицу, с которой осуществляется связь. Например, операторы

```
ClientDataSet1->Close();
ClientDataSet1->CommandText = "Dep";
ClientDataSet1->Open();
```

обеспечат связь с таблицей *Dep,* независимо от того, с какой таблицей был связан набор данных до этого.

Если в качестве серверного набора используется компонент, выполняющий запросы SQL (например, **TQuery),** то значение **CommandText** должно указывать запрос SQL. Например, операторы

```
ClientDataSet1->Close();
ClientDataSet1->CommandText = Edit1->Text;
ClientDataSet1->Open();
```

обеспечат выполнение запроса, содержащегося в окне **Edit1.**

Если запрос SQL содержит параметры, следите за правильной их последовательностью, так как провайдер распознает параметры в запросе **CommandText** только по индексам.

Примеры

Пусть в приложении имеется клиентский набор данных **ClientDataSetl,** связанный через провайдера с таблицей базы данных *Pers,* и имеется выпадающий список **ComboBoxl.** Компонент базы данных воспринимает запросы SQL (например, это компонент **TQuery).** Мы хотим заполнить **ComboBoxl** именами полей набора данных и предоставить пользователю возможность, выбрав в списке нужное поле, обеспечить индексацию данных по этому полю.

Для решения этой задачи в обработчик события формы **OnCreate** надо вставить операторы:

```
ComboBox1->Items->Assign(ClientDataSet1->FieldList);
ComboBoxl->ItemIndex = 0;
```

Первый из этих операторов заносит в **ComboBoxl** список полей, предоставляемый свойством **FicldList.**

Для обеспечения индексации по выбранному пользователем полю в обработчик события **OnChange** компонента **ComboBoxl** можно вставить операторы:

```
ClientDataSet1->Close();
ClientDataSet1->CommandText = "Select * from Pers order by " +
                              ComboBox1->Text;
ClientDataSet1->Open();
```

Первый оператор закрывает набор данных. Второй формирует текст **CommandText,** используя имя поля, выбранное пользователем. Последний оператор открывает набор данных. Поскольку в нем задано значение **CommandText,** то именно этот запрос будет выполняться. Но обязательное условие, при котором этот код работает - - в провайдере в свойстве **Options** должна быть задана опция **poAllowCommandText.**

CommaText, DelimitedText, Delimiter, QuoteChar

Определяют полный текст списка строк.

Класс *TStrings*

Определения

```
__property AnsiString CommaText
__property AnsiString DelimitedText
__property char Delimiter
__property char QuoteChar
```

Описание

Свойство **CommaText** возвращает текст списка строк, в котором отдельные строки объединены в одну строку формата SDF (system data format). Отдельные исходные строки разделяются в итоговой строке запятыми и каждая строка, если в ней имеются символы пробелов, заключается в двойные кавычки. Если в исходных строках использовались символы двойных кавычек, они дублируются (получается два следующих друг за другом символа).

Свойство **DelimitedText** — также полный текст списка, но в нем отдельные строки выделены кавычками, определенными свойством **QuoteChar,** и, кроме того, отделены друг от друга символом, указанным в свойстве **Delimiter.** Свойства **CommaText и DelimitedText** идентичны, если **Delimiter** = "," **и QuoteChar** = " ".

ComponentCount, ComponentIndex, Components

Определяют массив компонентов, которыми владеет данный компонент.

Класс *TComponent*

Определения

```
__property int ComponentCount
__property int ComponentIndex
__property TComponent* Components[int Index]
```

Описание

Свойство **Components** содержит массив компонентов, которыми владеет данный компонент. Параметр **Index** позволяет сослаться на любой компонент с помощью его свойства **ComponentIndex,** определенного в классе **TComponent**. Индексы отсчитываются от 0, т.е. индекс первого компонента равен 0. Общее число компонентов, содержащихся в массиве **Components,** определяется свойством **ComponentCount,** объявленным в классе **TComponent.** Значение **ComponentCount** на **1** меньше последнего индекса массива **Components.**

Свойство **Components** может использоваться вместе с **ComponentCount** в циклах, когда надо изменить какие-то свойства всех компонентов.

Примеры

1. В приведенном ниже примере все компоненты на данной форме, кроме компонента с именем **Button1,** смещаются вправо на 10 единиц.

```
for(int i = 0; i < ComponentCount; i++)
  if(Components[i]->Name != "Button1")
    ((TControl *)Components[i])->Left += 10;
```

2. Ниже приведен аналогичный пример, но используется свойство Tag и сдвигаются только компоненты, у которых Tag =1.

```
for(int i = 0; i < ComponentCount; i++)
  if(Components[i]->Tag == 1)
    ((TControl *)Components[i])->Left += 10;
```

Connected

Свойство, определяющее, активно ли соединение с базой данных.

Класс *TDatabase*

Определение

```
__property bool Connected
```

Описание

Свойство **Connected** (соединение) совместно со свойством **KeepConnection** управляют процессом соединения компонентов с базой данных. Если **KeepConnection** равно **true,** то соединение с базой данных постоянное даже при отсутствии открытых наборов данных. Если же **KeepConnection равно false,** то для регистрации на сервере надо устанавливать **Connected в true** при каждом открытии таблицы.

ConnectKind

Указывает способ соединения компонента **TOleServer** с сервером.

Класс *TOleServer*

Определение

```
enum TConnectKind {ckRunningOrNew, ckNewInstance,
                   ckRunningInstance, ckRemote, ckAttachToInterface};

__property TConnectKind ConnectKind;
```

Описание

Свойство **ConnectKind** показывает, как должно устанавливаться соединение компонента **TOleServer** с сервером OLE. Это свойство может принимать следующие значения:

ckRunningOrNew	Подсоединиться к выполняющемуся серверу или создать новый экземпляр сервера
ckNewInstance	Всегда создавать новый экземпляр сервера
ckRunningInstance	Только подсоединиться к выполняющемуся серверу
ckRemote	Подсоединиться к удаленному серверу. Эта опция должна сочетаться с заданием **RemoteMachineName**
ckAttachToInterface	Не подсоединяться к серверу. Вместо этого приложение обеспечивает интерфейс методом **ConnectTo**. Эта опция не может использоваться совместно с установкой свойства AutoConnect

ConstraintErrorMessage

См. раздел «CustomConstraint и ConstraintErrorMessage».

Constraints — свойство TControl

Позволяет задавать ограничения на допустимые изменения размеров компонента при изменениях размеров окна приложения.

Класс *TControl*

Определение

```
__property TSizeConstraints* Constraints
```

Описание

Свойство **Constraints** является объектом типа **TSizeConstraints**. Этот объект имеет четыре основных свойства: **MaxHeight, MaxWidth, MinHeight и MinWidth** — соответственно максимальная высота и ширина и минимальная высота и ширина компонента. Тип каждого из этих свойств — **TConstraintSize** целое без знака.

По умолчанию значения свойств **MaxHeight, MaxWidth, MinHeight и MinWidth** равны 0, что означает отсутствие ограничений. Но задание любому из этих свойств положительного значения приводит к соответствующему ограничению размера заданным числом пикселов.

Чтобы какие-то компоненты не исчезали из поля зрения при изменениях пользователем или программой размеров окна приложения, можно задать компонентам ограничения минимальной высоты и длины. Таким образом можно поддерживать нормальные пропорции отдельных частей окна. Можно задать ограничения на минимальные и максимальные размеры формы, т.е. всего окна. Например, если вы зададите для формы значения **MaxHeight = 500 и MaxWidth** = 500, то пользователь не сможет сделать окно большим, чем квадрат 500x500. Причем это ограничение будет действовать, даже если пользователь нажмет системную кнопку, разворачивающую окно на весь экран. Окно развернется, но его размеры не превысят заданных. Это иногда полезно делать, чтобы развернутое окно не заслонило какие-то другие нужные пользователю окна.

Constraints — свойство TDataSet

Ограничения на допустимые значения параметров на уровне записи.

Класс *TDataSet*

Определение

```
__property TCheckConstraints* Constraints
```

Описание

Свойство **Constraints** определяет ограничения на допустимые значения параметров на уровне записи. Его значение является объектом типа **TCheckConstraints,** содержащим ограничения и имеющим свои свойства и методы. Основное свойство — индексированный массив ограничений **Items:**

```
__property TCheckConstraint* Items[int Index]
```

Элементы этого массива являются объектами типа **TCheckConstraint.** Основные свойства этих объектов:

AnsiString CustomConstraint	Текст ограничения на языке SQL. Ограничения действуют только на уровне приложения и не влияют на запись данных на сервер в базу данных
AnsiString **ErrorMessage**	Строка, отображаемая в диалоговом окне, если пользователь ввел данные, не удовлетворяющие ограничению
bool FromDictionary	Определяет, должны ли искаться ограничения в словаре данных

Во время выполнения добавить новое ограничение можно методом **Add** класса **TCheckConstraints:**

```
HIDESBASE TCheckConstraint*__fastcall Add(void)
```

После этого, получив доступ к объекту ограничения через свойство Items класса **TCheckConstraints,** можно задавать свойства ограничения.

Во время проектирования свойство **Constraints** устанавливается нажатием кнопки в этом свойстве в Инспекторе Объектов. В результате открывается диалоговое окно списка ограничений. Щелкая в нем на кнопке Add вы можете занести в свойство **Constraints** набор ограничений. Выделив одно из ограничений, вы увидите в Инспекторе Объектов его свойства и можете установить их.

Например, можно написать в свойстве **CustomConstraint:**

```
((Sex=true) and (Year_b >1955))or((Sex=false) and (Year_b>1965))
```

а в свойстве **ErrorMessage:** «Принимаем только мужчин >1955 г.р. и женщин >1965 г.р.»

Это обеспечит вам различные границы отбора по возрасту для мужчин и женщин (предполагается, что Sex — булево поле, обозначающее пол, **a Year_b** — поле, содержащее год рождения).

ControlCount

Число дочерних компонентов оконного элемента.

Класс *TWinControl*

Доступ *только для чтения*

Определение

```
__property int ControlCount
```

Описание

Свойство **ControlCount** используется совместно со свойством **Controls** в итерациях по всем дочерним компонентам данного оконного элемента. Свойство **Controls,** с которым используется **ControlCount,** дает доступ к дочерним компонентам. Значение **ControlCount** всегда на 1 больше последнего индекса дочернего компонента, поскольку индексы считаются с 0.

См. примеры в описании **Controls.**

Controls

Список дочерних компонентов оконного элемента.

Класс *TWinControl*

Доступ *только для чтения*

Определение

```
__property TControl* Controls[int Index]
```

Описание

Свойство **Controls** является массивом всех дочерних компонентов данного оконного элемента. Дочерними являются те компоненты, которые расположены в клиентской области данного оконного элемента и в свойстве **Parent** которых указан как родитель данный элемент. Параметр **Index** определяет индекс соответствующего компонента. Индекс, начинающийся с 0, соответствует положению компонента в Z-последовательности данного родительского элемента.

Свойство **Controls** обычно используется в итеративных процедурах групповой обработки дочерних компонентов, когда на них неудобно ссылаться по имени. В подобных итеративных процедурах обычно используется также свойство **ControlCount.** определяющее число дочерних компонентов.

Надо четко представлять себе отличие свойства **Controls** от свойства **Components.** Свойство **Components** относится не к дочерним компонентам, а к тем, которыми владеет данный объект. В частности, всеми компонентами на форме владеет форма и они содержатся *в* ее списке **Components.**

Свойство **Controls** предназначено только для чтения. Оно изменяется (точнее меняются индексы компонентов) при изменении Z-последовательности.

Изменить Z-последовательность в процессе проектирования можно, выполнением команд Bring To Front или Send To Back. Первая из них пересылает выделенный компонент наверх, присваивая ему максимальный индекс, а вторая пересылает вниз, присваивая ему минимальный индекс (0 для неоконных компонентов и минимально возможный для оконных, поскольку всегда неоконные компоненты имеют индекс меньше оконных). Выполнить эти команды можно или из раздела меню Edit, или щелкнув правой кнопкой мыши и выбрав их из всплывающего меню.

Программно место компонента в Z-последовательности можно изменить методами **BringToFront и SendToBack.** На Z-последовательность влияют также методы **InsertControl и RemoveControl,** добавляющие и удаляющие дочерние компоненты, и изменение свойства компонентов **Parent.** меняющее родителя компонента.

Примеры

Пусть в приложении в классе формы определена некоторая функция

```
void__fastcall Func(TObject *Sender);
```

которая обрабатывает объект, передаваемый в нее через аргумент **Sender.** Это может быть какая-то процедура изменения размеров и места расположения, окрашивания, перестановок и т.д. Например, она может содержать оператор

```
((TControl *)Sender)->Left += 10;
```

сдвигающий объект на 10 пикселов вправо. Тогда обработать этой процедурой все дочерние компоненты, например, панели **Panell** можно с помощью оператора:

```
for(int i = 0; i < Panel1->ControlCount; i++)
  Func(Panel1->Controls[i]);
```

Если надо обработать только какую-то группу компонентов с идущими по порядку индексами и известны их начальный и конечный индексы **Ind1** и Ind2, то соответствующий оператор может иметь вид

```
for(int i = Ind1; i <= Ind2; i++)
  Func(Panel1->Controls[i]);
```

Задать нужную последовательность индексов можно описанными выше способами в процессе проектирования или во время выполнения приложения.

Выяснить истинную последовательность индексов для той же панели **Panell** можно, например, оператором:

```
for(int i = 0; i < Panel1->ControlCount; i++)
 ShowMessage("Ind = "+IntToStr(i) + ' ' +
             Panel1->Controls[i]->Name);
```

ControlState

Множество значений, характеризующих состояние компонента во время выполнения приложения.

Класс *TControl*

Определение

```
enum Controls__7 ( csLButtonDown, csClicked, csPalette,
                   csReadingState, csAlignmentNeeded,
                   csFocusing, csCreating, csPaintCopy,
                   csCustomPaint, csDestroyingHandle,
                   csDocking };

typedef Set<Controls__7, csLButtonDown, csDocking> TControlState;

__property TControlState ControlState
```

Описание

Свойство **ControlState** определяет различные условия, действующие на данный экземпляр компонента, например, щелчок мыши или необходимость выравнивания компонента. Свойство используется в основном при создании новых классов, производных от **TControl.** Свойство является множеством типа Set и может содержать следующие флаги:

Состояние	Пояснение
csLButtonDown	Левая кнопка мыши нажата, но еще не освобождена.
csClicked	То же самое, что csLButtonDown, но только в том случае, если свойство компонента ControlStyle содержит флаг csClickEvents, означающее, что событие, связанное с нажатием кнопки, интерпретируется как щелчок.
csPalette	Компонентом или одним из его родителей получено сообщение WM_PALETTCHANGED.
csReadingState	Компонент читает свое состояние из потока.
csAlignmentNeeded	Компонент должен осуществить выравнивание.
csFocusing	Приложение получило сообщение о переключении фокуса на данный компонент. Это не гарантирует, что компонент получит фокус, но позволяет предотвратить рекурсивные вызовы.

Состояние	Пояснение
csCreating	Создается данный компонент, или его владелец, или управляемый им компонент. Этот флаг очищается, когда создание компонента завершено.
csPaintCopy	Компонент должен быть перерисован. Это состояние возможно, если свойство **ControlStyle** содержит флаг csReplicatable.
csCustomPaint	Компонент обрабатывает сообщения перерисовки
csDestroyingHandle	Окно компонента разрушается.
csDocking	Компонент находится в процессе встраивания.

Свойство **ControlState** характеризует не класс в целом, а конкретный объект класса.

ControlStyle

Множество значений, характеризующих стиль компонента.

Класс *TControl*

Определение

```
enum Controls__8 { csAcceptsControls, csCaptureMouse,
                   csDesignInteractive, csClickEvents,
                   csFramed, csSetCaption, csOpaque,
                   csDoubleClicks, csFixedWidth, csFixedHeight,
                   csNoDesignVisible, csReplicatable,
                   csNoStdEvents, csDisplayDragImage,
                   csReflector, csActionClient, csMenuEvents };

typedef Set<Controls__8, csAcceptsControls, csMenuEvents>
                                                TControlStyle;

__property TControlStyle ControlStyle
```

Описание

Свойство **ControlStyle** определяет различные атрибуты компонента, например, может ли он быть захвачен мышью или имеет ли он фиксированные размеры. Свойство используется в основном при создании новых классов, производных от **TControl.** Свойство является множеством типа Set и может содержать следующие флаги:

Флаг	Пояснение
csAcceptsControls	Компонент становится родителем любого компонента, перенесенного на него *в* процессе проектирования.
csCaptureMouse	Компонент перехватывает события мыши после щелчка на нем.
csDesignInteractive	Компонент устанавливает соответствие во время проектирования щелчка правой кнопки мыши щелчку левой кнопки для манипуляций с компонентом.
csClickEvents	Компонент получает сообщение о щелчке мыши и реагирует на него.
csFramed	Компонент имеет объемную рамку.

Флаг	Пояснение
csSetCaption	Компонент должен изменять надпись на нем в соответствии со свойством **Name**, если только надпись не задана явным образом.
csOpaque	Компонент полностью заполняет свою клиентскую область.
csDoubleClicks	Компонент получает сообщение о двойном щелчке мыши и реагирует на него. Если флаг не установлен, то двойной щелчок интерпретируется как просто щелчок.
csFixedWidth	Ширина компонента не меняется и не масштабируется.
csFixedHeight	Высота компонента не меняется и не масштабируется.
csNoDesignVisible	Компонент невидим во время проектирования.
csReplicatable	Компонент может копироваться методом **PaintTo** для прорисовки произвольной канве.
csNoStdEvents	Игнорируются стандартные события, такие, как нажатие кнопок мыши, клавиш, щелчки. Этот флаг надо устанавливать, если ваш код не должен реагировать на эти события; в результате ваше приложение будет выполняться быстрее.
csDisplayDragImage	Компонент может отображать изображение из списка изображений, когда мышь перемещается на него. Этот флаг устанавливается, если компонент реализует список изображений для отображения при перемещении на него мыши.
csReflector	Компонент реагирует на сообщения Windows, поступающие из диалогов, сообщения о фокусировке, сообщения об изменении размеров. Этот флаг устанавливается, если компонент может использоваться как элемент ActiveX и должен реагировать на эти события.
csActionClient	Компонент связан с объектом действия. Этот флаг устанавливается при установке свойства Action и сбрасывается при очистке Action.
csMenuEvents	Компонент отвечает на команды главного меню.

Свойство **ControlStyle** описывает не свойства отдельных экземпляров класса, а класс в целом. Флаги не могут изменяться для различных экземпляров компонентов и не могут изменяться в процессе выполнения приложения. Изменяемые характеристики отображаются свойством **ControlState.**

Конструктор класса **TControl** инициализирует свойство **ControlStyle** значениями **[csCaptureMouse, csClickEvents, csSetCaption, csDoubleClicks].**

CopyMode

Определяет режим копирования графического изображения на канву.

Класс *TCanvas*

Определение

```
__property int CopyMode
```

Описание

Свойство канвы **CopyMode** определяет режим копирования графического изображения на канву методом **CopyRect** или при рисовании объекта **TBitmap.** Используя свойство можно достичь различных эффектов объединения изображений и их комбинирования.

Возможны следующие значения свойства **CopyMode** (используемые константы определены в *Windows.hpp*):

cmBlackness	Заполняет область канвы, в которую производится копирование, черным цветом. Собственное изображение на канве и копируемое изображение игнорируются
cmDstInvert	Инвертирует изображение на канве. Копируемое изображение игнорируется
cmMergeCopy	Комбинирует изображение канвы и копируемое изображение, используя булеву операцию and. То же, что cmSrcAnd
cmMergePaint	Комбинирует изображение канвы и инверсию копируемого изображения, используя булеву операцию or
cmNotSrcCopy	Копирует на канву инверсное изображение. Собственное изображение на канве игнорируется
cmNotSrcErase	Комбинирует изображения канвы и копируемого изображения, используя булеву операцию or, а затем инвертирует результат
cmPatCopy	Копирует шаблон источника на канву. Собственное изображение на канве игнорируется
cmPatInvert	Комбинирует изображение канвы и шаблон источника, используя булеву операцию xor
cmPatPaint	Комбинирует инверсное изображение источника и его шаблон, используя булеву операцию or. Затем этот результат комбинирует с изображением канвы, используя булеву операцию or
cmSrcAnd	Комбинирует изображения канвы и источника, используя булеву операцию and. То же, что cmMergeCopy
cmSrcCopy	Копирует изображение источника на канву. Собственное изображение на канве игнорируется. Этот режим принят по умолчанию
cmSrcErase	Инвертирует изображение на канве и комбинирует результат с изображением источника, используя булеву операцию and
cmSrcInvert	Комбинирует изображения канвы и источника, используя булеву операцию **xor.** Повторное копирование восстанавливает прежнее изображение на канве
cmSrcPaint	Комбинирует изображения канвы и источника, используя булеву операцию or
cmWhiteness	Заполняет область канвы, *в* которую производится копирование, белым цветом. Собственное изображение на канве и копируемое изображение игнорируются

Примеры

Операторы

```
Image1->Canvas->CopyMode = cmSrcCopy;
Image1->Canvas->CopyRect(Rect(0,0,100,100),Image2->Canvas,
                         Rect(0,0,100,100));
```

обеспечивают копирование области изображения канвы компонента **Image2** на канву компонента **Imagel.** Изображение, которое ранее было на канве компонента **Imagel,** в операциях не участвует.

Операторы

```
Image1->Canvas->CopyMode = cmSrcInvert;
Image1->Canvas->CopyRect(Rect(0,0,100,100),Image2->Canvas,
                         Rect (0,0,100,100));
...
Image1->Canvas->CopyRect(Rect(0,0,100,100),Image2->Canvas,
                         Rect(0,0,100,100));
```

обеспечивают копирование части изображения канвы компонента **Image2** на канву компонента **Imagel** в режиме **cmSrcInvert.** После выполнения функции **CopyRect** в первый раз изображения в компонентах **Imagel и Image2** налагаются друг на друга, а в результате выполнения функции **CopyRect** во второй раз исходное изображение на канве компонента **Imagel** восстанавливается.

Операторы

```
Image1->Canvas->CopyMode = cmWhiteness;
Image1->Canvas->CopyRect(Rect(0,0,100,100),Image2->Canvas,
                         Rect(0,0,100,100));
```

просто очищают указанную область канвы компонента **Imagel,** закрашивая ее белым цветом. При этом изображение в компоненте **Image2** никак не участвует в операциях копирования.

Count — свойство TCollection

Число объектов в контейнере **TCollection.**

Класс *TCollection*

Определение

```
__property int Count
```

Описание

Свойство **Count** определяет число объектов в контейнере **TCollection** — в массиве **Items.** Поскольку индексы, как всегда в C++Builder, начинаются с 0, то значение **Count** на единицу больше последнего индекса в **Items.**

Пример

См. пример в гл. 4, в разд. «BeginUpdate».

Count — свойство TFields

См. разд. «Fields».

Count — свойство списков

Количество элементов массива списка.

Классы *TList, TStringList, TStrings*

Определение

```
__property int Count
```

Описание

Свойство **Count** показывает число элементов массива указателей, хранящихся в объекте класса **TList. TStringList** или **TStrings. При** добавлении или удалении элементов методами **Add** и **Delete** значение **Count** увеличивается или уменьшается автоматически. Если намеренно увеличить значение **Count,** то в список добавится соответствующее число нулевых указателей **NULL.** Если намеренно уменьшить

значение **Count,** то соответствующее число последних элементов списка будет удалено.

Значение **Count** не всегда равно действительному числу указателей на объекты, хранящихся в списке, поскольку ряд хранящихся в нем указателей могут иметь значение **NULL.** Удалить эти элементы NULL можно методом Pack.

Свойство **Count** отличается от близкого ему свойства **Capacity. Capacity** показывает, сколько указателей может храниться в массиве, т.е. емкость списка, а **Count** показывает, сколько указателей в действительности хранится в массиве. Поэтому значение **Count** всегда не больше значения **Capacity.**

Пример

См. пример в разд. **Capacity.**

Ctl3D

Определяет, будет ли компонент выглядеть объемным или плоским.

Класс *TWinControl*

Определение

```
__property bool Ctl3D
```

Описание

Свойство **Ctl3D** определяет внешний вид элемента. Если **Ctl3D** установлено в **true,** элемент выглядит объемным. Если же **Ctl3D** установлено в **false,** то элемент выглядит плоским. По умолчанию **Ctl3D** равно **true.**

Если свойство дочерних компонентов **ParentCtl3D** установлено в **true,** то изменение свойства **Ctl3D** родительского компонента автоматически приводит к изменению свойств **Ctl3D** дочерних компонентов. Когда явным образом идет установка свойства **Ctl3D** компонента, его свойство **ParentCtl3D** автоматически устанавливается в **false.**

Cursor

Определяет изображение курсора мыши, когда он расположен в области компонента.

Класс *TControl*

Определение

```
enum TCursor {crMin=0x7fff-1, crMax=0x7fff};

__property TCursor Cursor
```

Описание

Изменение изображения курсора мыши при перемещении его в области компонента указывает пользователю на действия, которые он при этом может совершить. Имеется еще одно свойство, аналогичное **Cursor** — свойство **DragCursor.** Оно отвечает за изображение курсора при перемещении его в области компонента в процессе перетаскивания.

Значения свойств **Cursor** и **DragCursor** являются индексом списка возможных курсоров, управляемого глобальной переменной Screen. Кроме встроенных типов курсоров, обеспечиваемых в **TScreen,** приложение может добавить в список свои заказные изображения.

Таблицу встроенных в TScreen типов курсоров см. в гл. 1, в разд. «TCursor».

CurValue

Текущее значение поля с учетом изменений, внесенных другими пользователями.

Класс *TField*

Определение

```
__property System::Variant CurValue
```

Описание

Свойство **CurValue** определяет текущее значение поля с учетом изменений, внесенных другими пользователями. Это значение можно использовать, когда возникают проблемы с пересылкой значения в базу данных. При возникновении таких проблем в клиентском наборе данных генерируется событие **OnReconcileError.** На стороне сервера компонент-провайдер генерирует событие **OnUpdateError.** В обработчике этого события можно сравнить **NewValue** — новое значение, которое не удалось переслать, **OldValue** — прежнее значение до начала редактирования данного поля, и **CurValue** — текущее значение поля. Текущее значение может отличаться от **OldValue,** если другой пользователь изменил значение поля за время, которое прошло после чтения **OldValue.**

Свойство поддерживается только в наборах данных **TClientDataSet.**

CustomConstraint и ConstraintErrorMessage

Строка SQL, накладывающая ограничение на вводимое пользователем значение поля.

Класс *TField*

Определение

```
__property AnsiString CustomConstraint
__property AnsiString ConstraintErrorMessage
```

Описание

Свойство **CustomConstraint** позволяет написать ограничение на вводимое значение поля в виде строки SQL. Если вы задали свойство **CustomConstraint,** то необходимо задать и свойство **ConstraintErrorMessage.** Оно содержит строку текста, который будет показан пользователю в диалоговом окне в случае, если он вводит данные, не удовлетворяющие поставленным ограничениям.

Ограничение, введенное в **CustomConstraint,** действует одновременно с ограничением, которое может передаваться с сервера. Это внешнее ограничение вы можете прочитать в свойстве **ImportedConstraint.**

Применение свойства **CustomConstraint** удобнее задания **MinValue и MaxValue,** поскольку не требует перехвата исключений. К тому же ограничение, записанное в **CustomConstraint,** может применяться не только к числовым полям.

Пример

Для поля **Year_b,** определяющего год рождения, вы можете написать в свойстве **CustomConstraint:**

```
X < 1980 and X > 1950
```

Это ограничение требует, чтобы вводимый год рождения был меньше 1980 и больше 1950. Имя поля, использованное в выражении (в данном выражении оно обозначено как X), может быть произвольным идентификатором.

Одновременно необходимо задать соответствующую строку в свойстве **ConstraintErr or Message.** Например, «Возраст претендента на должность не подходит».

Data, XMLData — свойства TCustomClientDataSet

Данные клиентского набора.

Класс *TCustomClientDataSet*

Объявления

```
__property System::OleVariant Data
__property System::OleVariant XMLData
```

Описание

Свойство **Data** представляет все данные клиентского набора в виде пакета, используемого клиентским набором данных. В таком виде данные получаются от провайдера, из файла, из другого клиентского набора.

Свойство **XMLData** представляет те же данные, что **Data,** но не в двоичном виде, а как пакет XML.

Примеры

См. пример в гл. 4, в разд. «DataRequest, OnDataRequest — методы и событие».

Database

Свойство, определяющее объект базы данных.

Класс *TDBDataSet*

Доступ *только для чтения*

Определение

```
__property TDatabase Database
```

Описание

Свойство **Database** определяет компонент базы данных **TDatabase.** с которым связан набор данных. Этот объект создается автоматически, когда открывается база данных, указанная в свойстве **DatabaseName.** Объект **Database** дает доступ к свойствам, методам и событиям базы данных.

Пример

Следующий пример показывает организацию пересылки в базу данных отредактированной записи с помощью объекта **Database.**

```
Table1->Database->TransIsolation = tiDirtyRead;
Table1->Database->StartTransaction();
// Пересылка данных
Table1->Database->Commit();
```

DatabaseName — свойство TDBDataSet

Свойство, определяющее имя базы данных, связанной с набором данных.

Класс *TDBDataSet*

Определение

```
__property AnsiString DatabaseName
```

Описание

Свойство **DatabaseName** определяет имя базы данных. Оно соответствует объекту базы данных **TDatabase.**

Во время проектирования **DatabaseName** в компонентах наборов данных **(Table, Query** и др.) устанавливается в окне Инспектора Объектов выбором псевдонима из выпадающего списка или прямым вводом строки, определяющей путь к базе данных, если база данных не имеет псевдонима.

Во время выполнения свойство DatabaseName может устанавливаться программно.

При задании или изменении **DatabaseName** компонент набора данных должен находиться в неактивном состоянии (свойство **Active = false).** В противном случае будет сгенерировано исключение.

См. пример в разделе «TableName».

DatabaseName — свойство компонента типа TDatabase

Определяет имя базы данных.

Класс *TDatabase*

Определение

```
__property AnsiString DatabaseName
```

Описание

Свойство **DatabaseName** определяет имя базы данных, связанной с компонентом базы данных. Если значение **DatabaseName** является псевдонимом Borland Database Engine (BDE), то свойства **AliasName и DriverName** можно не устанавливать. Если же значение **DatabaseName** не соответствует псевдониму BDE, то надо или в дополнение к **DatabaseName** задать псевдоним в свойстве **AliasName,** или задать свойства **DriverName и Params.**

При работе с базами данных Paradox и dBASE значение **DatabaseName** может быть полным путем к базе данных.

Попытка установить **DatabaseName** при значении свойства **Connected,** равном **true,** приведет к генерации исключения.

Во время проектирования задание значения **DatabaseName** можно осуществлять с помощью редактора баз данных, вызываемого двойным щелчком на компоненте **TDatabase.**

Пример

В примере описывается приложение, позволяющее просматривать любую заданную пользователем таблицу в любой заданной им доступной базе данных. Приложение содержит два выпадающих списка типа **TComboBox,** названных **cbDatabase и cbTable.** Первый из них предназначен для выбора пользователем базы данных, а второй — для выбора таблицы.

При создании формы в обработчике ее события **OnCreate** надо загрузить список **cbDatabase** доступными BDE псевдонимами:

```
Session->GetAliasNames(cbAlias->Items);
```

Этот оператор использует метод **GetAliasNames** объекта Session, который передает в свой параметр типа **TStrings** (в данном примере это список **cbAlias-> Items)** перечень псевдонимов баз данных, зарегистрированных в BDE.

При выборе пользователем базы данных в списке **cbAlias** надо загрузить список **cbTable** перечнем таблиц выбранной базы данных. Это делается включением в обработчик события **OnChange** компонента **cbAlias** операторов:

```
Session->GetTableNames(cbAlias->Text,"",true,false,
                       cbTable->Items);
cbTable->ItemIndex = 0;
```

Первый из этих операторов использует метод **GetTableNames** для загрузки в свой последний параметр типа **TStrings** (в данном примере это список **cbTable->Items)** перечня таблиц базы данных, заданной своим первым параметром (в данном примере это **cbAlias->Text).**

Обработчик щелчка на копке, обеспечивающей просмотр таблицы, может иметь вид:

```
if  (cbTable->Text  ==  "")
 {
  ShowMessage("Не задана таблица");
  return;
 }
Table1->Active = false;
Table1->DatabaseName = cbAlias->Text;
Table1->TableName = cbTable->Text;
Table1->Active = true;
```

Этот обработчик просто задает свойства **DatabaseName и TableName** компонента **Tablel** равными выбранным пользователем значениям и переводит **Tablel** в активное состояние.

DataField

См. «DataSource, DataField, Field — свойства компонентов, связанных с данными».

DataSize

Размер в байтах, необходимый для хранения значения поля.

Класс *TField*

Доступ *только для чтения*

Определение

```
__property int DataSize
```

Описание

Свойство **DataSize** определяет объем памяти в байтах, необходимый для хранения значения поля. Значение может требоваться для назначения буфера при использовании, например, методов **GetData и SetData.** Значение **DataSize = 0** означает, что объем памяти неизвестен. Это характерно, например, для полей BLOB.

DataSource — свойство наборов данных

Указывает источник данных другого набора данных, снабжающего информацией данный набор.

Классы *TDataSet*. *TTable*. *TQuery*, *TADOQuery*

Определение

```
__property TDataSource* DataSource
```

Описание

Свойство **DataSource** указывает источник данных другого набора данных, снабжающего информацией данный набор.

В TDataSet свойство всегда NULL, но оно перегружено в классах-наследниках. В **TTaЬle DataSource** — просто вариант свойства **MasterSource** только для чтения.

Более содержательно это свойство в **TQuery и в TADOQuery.** Оно позволяет строить приложения, содержащие связанные друг с другом таблицы.

Пример

Пусть вы хотите построить приложение, работающее с таблицей Dep, содержащей список отделов (в поле **Dep)** и с таблицей персонала Pers, содержащей в поле Dep имя отдела, в котором работает каждый сотрудник. Надо, чтобы при выборе записи в таблице Dep в таблице Pers отбирались только записи, относящиеся к выбранному отделу. Тогда надо перенести на форму компоненты **Query1, DataSource1, DBGridl** и соединить их обычной цепочкой: в **DBGridl** задать свойство

DataSource равным **DataSource1, а в DataSourcel** задать свойство **DataSet** равным **Query1.** Компонент **Query1** настраивается на таблицу Dep. Создается также другая аналогичная цепочка, перенеся на форму компоненты **Query2, DataSource2, DBGrid2,** и связывается с таблицей Pers запросом

```
Select * from Pers where (Dep=:Dep)
```

В этом запросе указано условие отбора: значение поля **Dep** должно быть равно параметру :Dep. В данном случае не надо определять этот параметр с помощью редактора параметров, вызываемого из свойства **Params** компонента **Query2.** Вместо этого в свойстве **DataSource** компонента **Query2** надо сослаться на **DataSourcel** — источник данных, связанный с таблицей Dep. Это скажет приложению, что оно должно взять значения параметра :Dep из текущей записи этого источника данных. А поскольку имя параметра совпадает с именем поля в источнике данных, то в качестве значения параметра будет взято текущее значение этого поля. Таким образом, вспомогательная таблица, запрашиваемая в **Query2,** оказывается связанной с головной таблицей, запрашиваемой в **Queryl.**

DataSource, DataField, Field — свойства компонентов, связанных с данными

Определяют источник данных и поле, с которыми связан компонент.

Классы *TDBText, TDBEdit , TDBMemo, TDBImage*и многие другие

Определения

```
__property Db::TDataSource* DataSource
__property AnsiString DataField
__property Db::TField* Field        // только для чтения
```

Описание

В компонентах, связанных с данными, свойство **DataSource** определяет источник данных типа **TDataSource.** свойство **DataField** определяет имя поля в **DataSource,** с которым связан компонент, а свойство **Field** типа **TField** (только для чтения) определяет объект самого поля. Через этот объект можно получить доступ к данным, хранящимся в этом поле в текущей записи.

В компонентах **DBLookupListBox и DBLookupComboBox** свойство **DataField** указывает имя отображаемого просматриваемого поля, а не того, которое снабжает информацией это поле.

DataType

Определяет тип данных, хранящихся в поле.

Классы *TField. TFieldDef*

Доступ *только для чтения*

Определение

```
__property TFieldType DataType
```

Описание

Свойство **DataType** типа **TFieldTvpe** определяет тип данных поля. Для существующих полей свойство **DataType** можно использовать для проверки типа поля. При создании описаний **TFieldDef** новых полей это свойство должно задаваться.

Пример

См. пример в гл. 4, в описании метода **CreateTable.**

DBSession

Определяет связанный с набором данных компонент **Session.**

Класс *TDBDataSet*

Определение

```
__property TSession* DBSession
```

Описание

Свойство **DBSession** определяет связанный с набором данных компонент **Session.** По умолчанию связь осуществляется с генерируемым автоматически объектом **Session.** Компонент **Session** осуществляет общее управление связыванием приложения с базами данных. Он имеет много полезных методов и позволяет легко работать с BDE, например, через такие методы, как **GetAliasNames**, **GetTableNames** и др. (см. в гл. 1, в описании **Session**).

DefaultExpression

Выражение SQL, которое присваивает значение поля, если пользователь не задал никакого значения.

Класс *TField*

Определение

```
__property AnsiString DefaultExpression
```

Описание

Свойство **DefaultExpression** содержит выражение SQL, которое присваивает значение поля, если пользователь не задал никакого значения. Значение **DefaultExpression** может быть любым допустимым выражением SQL, не ссылающимся на значения полей. Если выражение содержит символы, отличные от цифр, оно должно заключаться в одинарные кавычки. Например: '12:00:00' или 'отсутствует'.

Свойство **DefaultExpression** имеет приоритет над значениями по умолчанию, предусмотренными в базе данных. Это объясняется тем, что данное свойство применяется при вводе данных в приложении, т.е. до того, как запись передается в базу данных.

DelimitedText

Полный текст списка строк.

См. раздел «CommaText, DelimitedText, Delimiter, QuoteChar».

Delimiter

Разделитель, используемый в свойстве **DelimitedText.**

См. раздел «CommaText, DelimitedText, Delimiter, QuoteChar».

Delta — свойство TCustomClientDataSet

Пакет изменений в данных, еще не зафиксированных в базе данных.

Класс *TCustomClientDataSet*

Объявление

```
__property System::OleVariant Delta
```

Описание

Свойство **Delta** (только для чтения) представляет собой пакет изменений, который может быть передан провайдеру. Он содержит информацию об измененных, вставленных и удаленных записях. Значение **Delta** передается как аргумент методов **ApplyUpdates** и **Reconcile**, обеспечивающих занесение результатов редактирования в базу данных. Если обновление базы данных прошло успешно, **Delta** очищается. Если при обновлении базы произошли ошибки, то в результате содержи-

мое **Delta** зависит от числа возникших ошибок. Если это число превысило максимум, указанный при вызове метода **ApplyUpdates**, то **Delta** сохраняет все изменения, а те изменения, которые не вызвали ошибок, отменяются. Так что в этом случае никаких изменений в базе данных не происходит. Если же число ошибок меньше допустимого, то в **Delta** остаются только изменения, вызвавшие ошибки.

В приложениях, использующих файловые наборы данных, очистка **Delta** происходит также при выполнении метода **MergeChangeLog**, объединяющего данные **Data и Delta.**

DesktopFont

Определяет, использует ли компонент для отображения текста изображение шрифта Windows.

Класс *TControl*

Определение

```
__property bool DesktopFont
```

Описание

Установка свойства **DesktopFont в true** определяет, что компонент должен использовать для свойства **Font** изображение шрифта Windows. Этот шрифт задается свойством **IconFont** глобальной переменной Screen.

Если **DesktopFont** установлено в **true**, то свойство **Font** будет изменяться всякий раз при изменении шрифта Windows. Это изменение может осуществляться заданием свойства **IconFont** глобальной переменной Screen или изменяться другими программами. Установка свойства **Font** напрямую равным **Screen::IconFont** не обеспечивает изменение шрифта компонента при изменении шрифта Windows.

DisplayLabel и DisplayName

Строка, отображаемая как заголовок столбца в таблице данных.

Класс *TField*

Определения

```
__property AnsiString DisplayLabel
__property AnsiString DisplayName  // только для чтения
```

Описание

Свойства **DisplayLabel и DisplayName** представляют собой строку, отображаемую как заголовок столбца в таблице данных. Свойство **DisplayName** — только для чтения, а значение **DisplayLabel** можно задавать. По умолчанию **DisplayName** совпадает с **FieldName**. Но если задать значение **DisplayLabel,** то **DisplayName** становится равным **DisplayLabel.**

Пример

Следующий пример отображает окно редактирования, содержащее имена всех полей таблицы, связанной с компонентом **Tablel,** и их отображаемые имена **DisplayName:**

```
String Info = "Поля таблицы " + Table1->TableName +" :\n\n";
 for (int i=0; i < Table1->FieldCount; i++)
   Info = Info + Table1->Fields->Fields[i]->FieldName +
         " - " + Table1->Fields->Fields[i]->DisplayName + '\n';
ShowMessage(Info);
```

DisplayName — свойство TField

См. раздел «DisplayLabel и DisplayName».

DisplayName — свойство TNamedItem

Имя, появляющееся в редакторе наборов (например, в Редакторе Полей) во время проектирования.

Класс *TNamedItem*

Определение

```
__property System::AnsiString DisplayName
```

Описание

Свойство **DisplayName** определяет имя, появляющееся в редакторе наборов (например, в Редакторе Полей) во время проектирования. Оно принудительно устанавливается равным свойству **Name.**

DisplayText и Text

Строка, отображающая значение поля в компонентах, связанных с данными.

Класс *TField*

Определения

```
__property AnsiString DisplayText // только для чтения
__property AnsiString Text
```

Описание

Свойства **Text и DisplayText** представляют собой строку, отображающую значение поля **Value** в компонентах, связанных с данными. Строка **DisplayText** соответствует отображению в режиме просмотра данных, а строка **Text** соответствует режиму редактирования. По умолчанию **Text и DisplayText** соответствуют свойству **AsString.**

Значения **Text** и **DisplayText** могут различаться, если требуется разное отображение в режимах просмотра и редактирования. Для обеспечения двух разных форматов отображения используется обработчик события **OnGetText.** При наличии такого обработчика в **Text или в DisplayText** заносится значение, возвращаемое как параметр **Text.** Параметр обработчика показывает, в какое именно свойство заносится строка.

DisplayText соответствует значению поля **Value,** если не проводится редактирование. Если проведено редактирование, его результаты отображаются в свойстве Text.

DragCursor

См. **Cursor.**

DragKind

Определяет, будет ли объект перетаскиваться по технологии Drag&Drop, или Drag&Doc.

Класс *TControl*

Определение

```
enum TDragKind { dkDrag, dkDock };
__property TDragKind DragKind
```

Описание

Свойство **DragKind** определяет, будет ли компонент перетаскиваться обычным образом (по технологии Drag&Drop) — значение **dkDrag,** или он будет перетаскиваться для встраивания (по технологии Drag&Doc) — значение **dkDock.**

DragMode

Определяет поведение компонента в процессе его перетаскивания.

Класс *TControl*

Определение

```
enum TDragMode ( dmManual, dmAutomatic };
__property TDragMode DragMode
```

Описание

Свойство **DragMode** определяет поведение компонента в процессе его перетаскивания. Свойство может принимать значения:

dmAutomatic	От программиста не требуется обработка каких-либо событий. Компонент готов к перетаскиванию. Пользователю достаточно щелкнуть на его изображении и начать перетаскивать.
dmManual	Компонент не может быть перетащен, пока приложение не вызовет метод **BeginDrag**.

Примеры использования свойства **DragMode** см. в гл. *5,* **в** разд. «OnDragDrop».

DrawingStyle

Определяет стиль, используемый при рисовании изображения.

Класс *TCustomImageList*

Определение

```
enum TDrawingStyle (dsFocus, dsSelected, dsNormal, dsTransparent};
__property TDrawingStyle DrawingStyle
```

Описание

Свойство **DrawingStyle** определяет стиль изображения. Возможные значения:

dsFocused	Изображение рисуется на 25% смешанное с цветом, указанным свойством BlendColor. Влияет только на список изображений, содержащий маски
dsSelected	Изображение рисуется на 50% смешанное с цветом, указанным свойством BlendColor. Влияет только на список изображений, содержащий маски
dsNormal	Изображение рисуется с использованием цвета, указанного свойством BkColor. Если BkColor задано равным clNone, изображение рисуется прозрачным с использованием маски
dsTransparent	Изображение рисуется с использованием маски, независимо от установки BkColor

EditMask и EditMaskPtr

Маска, используемая для ввода данных.

Классы *T Field. TMaskEdit*

Определения

```
__property AnsiString EditMask
__property AnsiString EditMaskPtr
```

Описание

Свойство **EditMask** содержит маску, используемую для ввода данных в поля набора данных и в компонент **MaskEdit.** Свойство **EditMaskPtr,** имеющееся только в **TField,** обеспечивает доступ к той же маске, но только для чтения, что позволяет избежать случайного ее изменения.

Применение маски позволяет ограничить возможность пользователя ввести синтаксически ошибочные данные. Свойство **EditMask** осуществляет проверку вводимых данных по символам. Для проверки в **TField** введенного значения в целом можно использовать обработчик события **OnValidate.**

Задание в качестве **EditMask** пустой строки удаляет маску.

Маска состоит из трех разделов, между которыми ставится точка с запятой (;). В первом разделе — шаблоне записываются специальным образом символы (см. приведенную ниже таблицу), которые можно вводить в каждой позиции, и символы, добавляемые самой маской; во втором разделе записывается 1 или 0 в зависимости от того, надо или нет, чтобы символы, добавляемые маской, включались в свойство **Text** компонента **MaskEdit** или в свойство **Value** поля; в третьем разделе указывается символ, используемый для обозначения позиций, в которых еще не осуществлен ввод.

Специальные символы маски:

!	Наличие символа "!" означает, что недостающие символы предваряются пробелами, а отсутствие символа "!" означает, что пробелы размещаются в конце.
>	Символ ">" означает, что все последующие за ним символы должны вводиться в верхнем регистре, пока не кончится маска или пока не встретится символ "<".
<	Символ "<" означает, что все последующие за ним символы должны вводиться в нижнем регистре, пока не кончится маска или пока не встретится символ ">".
o	Символы "o" означают, что анализ регистра не производится.
\	Символ "\" означает, что следующий за ним символ является буквенным, а не специальным, характерным для маски. Например, символ ">" после символа "\" воспримется как знак >, а не как символ, указывающий на верхний регистр.
L	Символ "L" означает, что в данной позиции должна быть буква.
l	Символ "l" означает, что в данной позиции может быть только буква или ничего.
A	Символ "A" означает, что в данной позиции должна быть буква или цифра.
a	Символ "a" означает, что в данной позиции может быть буква, или цифра, или ничего.
C	Символ "C" означает, что в данной позиции должен быть любой символ.
c	Символ "c" означает, что в данной позиции может быть любой символ или ничего.
0	Символ "O" означает, что в данной позиции должна быть цифра.
9	Символ "9" означает, что в данной позиции может быть цифра или ничего.
#	Символ "#" означает, что в данной позиции может быть цифра, знак "+", знак " " или ничего.

:	Символ ":" используется для разделения часов, минут и секунд.
/	Символ "/" используется для разделения месяцев, дней и годов в датах.
_	Символ "_" означает автоматическую вставку в текст пробела.

Примеры

Ниже приведены примеры масок.

Семизначный номер телефона	!000-00-00;0;_
Номер телефона с кодом страны	!\(999\) 000-00-00;0;_
Почтовый индекс	!0000000;1;_
Паспорт	!L-LL 999999;0;_
Дата с указанием дня	!99/99/00;1;_
Дата без указания дня	!99/00;1;_
Время с секундами	!90:00:00;1;_
Время без секунд	!90:00;1;_

EditMaskPtr

См. раздел «EditMask и EditMaskPtr».

Enabled

Определяет, реагирует ли компонент на события, связанные с мышью, клавиатурой и таймером.

Класс *TControl*

Определение

```
__property bool Enabled
```

Описание

Свойство **Enabled** определяет доступность компонента для пользователя. Чтобы сделать компонент недоступным, надо установить **Enabled в false.** Недоступный компонент отображается серым цветом. Он игнорирует события, связанные с мышью, клавиатурой и события таймера **OnTimer.**

Чтобы восстановить доступность компонента, надо установить **Enabled** в **true.** Компонент начинает нормально отображаться на экране и реагировать на действия пользователя.

Eof — свойство TDataSet

Свойство указывает, находится ли курсор на последней записи.

Класс *TDataSet*

Определение

```
__property bool Eof
```

Описание

Свойство Eof определяет, находится ли курсор на последней записи. Значение Eof устанавливается в **true:**

- При открытии пустого набора данных
- В результате выполнения метода Last

- При ошибке выполнения метода **Next** из-за того, что курсор уже расположен на последней записи
- При вызове метода **SetRange** с пустым диапазоном данных или на пустом наборе данных

В остальных случаях **Eof = false.**

Свойство **Eof** используется при организации циклов по набору данных.

Пример

Следующий пример показывает типичный способ организации цикла по записям набора данных. Пусть в вашем приложении имеется выпадающий список с именем **CBdep,** который вы хотите заполнить данными, содержащимися в полях Dep всех записей таблицы, соединенной с компонентом **Table1.** Это можно сделать следующим кодом:

```
CBdep->Clear();
Table1->First();
while (! Table1->Eof)
{
 CBdep->Items->Add(Table1->FieldByName("Dep")->AsString);
 Table1->Next();
}
```

Первый оператор кода очищает список CBdep. Второй — устанавливает курсор таблицы на первую запись. Далее следует цикл по всем записям, пока не достигнута последняя, что проверяется выражением **Table1**->Eof. Для каждой записи в список заносится значение поля Dep, после чего методом Next курсор перемещается к следующей записи.

Exists

Свойство, определяющее, существует ли данная таблица базы данных.

Класс *TTable*

Доступ *только для чтения*

Определение

```
__property bool Exists
```

Описание

Свойство **Exists** позволяет во время выполнения определить, существует ли таблица, установленная в компоненте **Table** свойством **TableName.** Если не существует, то она может создаваться методом **CreateTable.**

См. пример применения свойства **Exists** в гл. 4, **в** разд. «CreateTable».

Field

См. «DataSource, DataField, Field — свойства компонентов, связанных с данными».

FieldClass

Класс объекта поля, соответствующий описанию **TFieldDef.**

Класс *TFieldDef*

Доступ *только для чтения*

Определение

```
__property System::TMetaClass* FieldClass
```

Описание

Свойство **FieldClass** определяет класс объекта поля, соответствующий описанию **TFieldDef.** Объект поля создается методом **CreateField** (вызывается автоматически) из описания **TFieldDef** как объект класса **FieldClass.** Значение **FieldClass** однозначно определяется значением свойства поля **DataType** типа **TFieldType.** Например, если значение **DataType** равно **ftString,** то значение **FieldClass** равно **TStringField.**

FieldCount

Число объектов полей, связанных с набором данных.

Класс *TDataSet*

Определение

```
__property int FieldCount
```

Описание

Свойство **FieldCount** определяет число полей, содержащихся в свойстве **Fields** набора данных. В это число не входят поля совокупных характеристик, список которых содержится в свойстве **AggFields.** Если объекты полей создаются динамически (не с помощью Редактора полей), то значение **FieldCount** может меняться при каждом открытии набора данных. Если же объекты полей созданы заранее (например, Редактором Полей), то значение **FieldCount** постоянно.

Примеры

В следующем примере выпадающий список **ComboBoxl** заполняется именами полей таблицы, соединенной с компонентом **Table1:**

```
Table1->Active = true;
ComboBox1->Clear();
for (int i=0; i < Table1->FieldCount; i++)
  ComboBox1->Items->Add(Table1->Fields->Fields[i]->FieldName);
```

Почти аналогичного результата можно достичь с помощью списка описаний полей — **FieldDefs.** Для этого в приведенном выше примере надо заменить оператор **for** на:

```
for (int i=0; i < Table1->FieldDefs->Count; i++)
  ComboBox1->Items->Add(Table1->FieldDefs->Items[i]->Name);
```

или на тождественный ему:

```
for i:= 1 to Table1.FieldDefs.Count - 1 do
  ComboBox1.Items.Add(Table1.FieldDefs.Items[i].Name);
```

Отличие результата, полученного этими операторами, от рассмотренного ранее будет в том, что в списке не окажется имени поля первичного ключа,

Впрочем, получить список полей можно проще методом **GetFieldNames** набора данных:

```
Table1->GetFieldNames(ComboBox1->Items);
```

или методом **GetFieldNames** свойства **Fields:**

```
Table1->Fields->GetFieldNames(ComboBox1->Items);
```

или методом **GetItemNames** свойства **FieldDefs:**

```
Table1->FieldDefs->GetItemNames(ComboBox1->Items);
```

FieldDefs

Свойство, указывающее на список объектов, определяющих поля таблицы базы данных.

Класс *TDataSet*

Определение

```
__property TFieldDefs* FieldDefs
```

Описание

Свойство **FieldDefs** указывает на список типа **TFieldDefs** объектов полей типа **TFieldDef,** определяющих поля таблицы базы данных. Это свойство должно быть определено, прежде чем вызывается метод **CreateTable.** создающий таблицу.

Свойства и методы объекта **FieldDefs** позволяют

- Получить доступ к определениям полей
- Добавлять, изменять или удалять поля, создавая новую таблицу
- Определять, сколько полей содержит таблица
- Копировать определения полей из одной таблицы в другую

См. примеры применения свойства **FieldDefs** в разд. «CreateTable» и «FieldCount».

FieldKind

Указывает, является ли поле обычным полем базы данных, или вычисляемым полем, или полем просмотра.

Класс *TField*

Определение

```
enum TFieldKind { fkData, fkCalculated, fkLookup, fkInternalCalc };

__property TFieldKind FieldKind
```

Описание

Свойство **FieldKind** определяет тип поля, указывающий, является ли оно обычным полем базы данных, или вычисляемым полем, или полем просмотра. Значение **FieldKind** можно изменять программно во время выполнения, но обычно оно задается во время проектирования в Редакторе Полей.

Описание возможных значений **FieldKind** см. в гл. 1, в описании **TFieldKind.**

FieldList

Список имен полей набора данных.

Класс *TDataSet*

Объявление

```
__property TFieldList* FieldList
```

Описание

Свойство **FieldList** содержит список имен всех полей набора данных. Класс этого списка **TFieldList** является дальним потомкам класса TStringList и наследует все основные свойства и методы TStringList, такие как **Add. Delete, Insert** и др. Из новых свойств класса **TFieldList** важнейшим является **Fields:**

```
__property TField* Fields[int Index]
```

Это свойство — индексированный список объектов полей. Еще одно свойство **DataSet** — набор данных.

Среди методов класса **TFieldList** надо отметить методы **FieldByName(const** AnsiString **Name)** и Find(const **AnsiString Name),** возвращающие объект поля, заданного именем **Name.** Различие методов в том, что при отсутствии поля в списке **Find** возвращает NULL, а **FieldByName** генерирует исключение.

Типичный пример использования **FieldList:**

```
ComboBox1->Items->Assign(Table1->FieldList);
```

Этот оператор заносит в выпадающий список ComboBoxl список всех полей набора данных **Table1.**

FieldName

Имя поля набора данных.

Класс *TField*

Определение

__property AnsiString FieldName

Описание

Значение свойства FieldName задается автоматически равным имени поля, объект которого создается. Для вычисляемых полей имя **FieldName** задается при описании поля. Свойство **FieldName** используется для задания значений по умолчанию свойствам **DisplayLabel и DisplayName.**

Для не вычисляемых полей генерируется исключение **EDatabaseError,** если имя **FieldName** не совпадает с именем одного из полей таблицы данных.

Примеры

См. примеры в разд. «Fields» и «DisplayLabel и DisplayName».

FieldNo

Порядковый номер поля в таблице.

Класс *TFieldDef*

Определение

__property int FieldNo

Описание

Свойство **FieldNo** определяет порядковый номер поля в таблице. Например, если **FieldNo** равно **3,** то это третье поле таблицы. При добавлении определения поля в набор данных значение **FieldNo** определяет место создаваемого поля в таблице.

Fields — свойство TFields

Список ссылок на поля.

Класс *TFields*

Определение

__property TField* Fields[int Index]

Описание

Свойство **Fields** -- индексированный список указателей на объекты полей типа **TField.** Индексы **Index** начинаются с 0. Число полей в списке определяется свойством списка **Count.** С помощью **Fields** можно присвоить свойства какого-то другого объекта **TField** полю данного списка.

Пример

Следующий пример отображает окно редактирования, содержащее имена всех полей таблицы, связанной с компонентом **Table1:**

```
String Info = "Поля таблицы " +
      Table1->TableName +" :\n\n";
for (int i=0; i < Table1->Fields->Count; i++)
   Info = Info + Table1->Fields->Fields[i]->FieldName + '\n';
ShowMessage(Info);
```

В данном примере для итераций по полям использовано свойство **Fields->Count.** То *же* самое можно было бы сделать, используя свойство набора данных **FieldCount:**

```
for (int i=0; i < Table1->FieldCount; i++)
  ...
```

Fields и ObjectView — свойства TDataSet

Список всех объектов полей (кроме полей совокупных характеристик) набора данных и способ его организации.

Класс *TDataSet*

Определения

```
_ property TFields* Fields // только для чтения
_ property bool ObjectView
```

Описание

Свойство **Fields** — список всех объектов полей (кроме полей совокупных характеристик) набора данных. Поля совокупных характеристик размещены в **AggFields.**

Если объекты полей сгенерированы динамически во время выполнения, то их последовательность совпадает с последовательностью столбцов таблицы набора данных. Если же объекты полей созданы во время проектирования с помощью Редактора Полей, то их последовательность в массиве **Fields** та, которая была задана при проектировании.

Если значение **ObjectView** равно **true,** то поля хранятся в **Fields** иерархически: родительское поле ссылается на дочерние поля. Если значение **ObjectView** равно **false,** то поля хранятся в **Fields** «плоско», т.е. дочерние поля хранятся после родительского поля.

Свойство **Fields** позволяет организовывать циклы по полям и работать с данными, структура которых заранее не известна. Можно также читать и записывать значения отдельных полей. Например, следующий оператор читает в окно редактирования **Edit1** текст второго поля набора данных **Table1:**

```
Edit1->Text = Table1->Fields->Fields[1]->AsString;
```

Следующий оператор осуществляет обратное действие — заносит во второе поле значение, заданное в окне **Edit1:**

```
Table1->Fields->Fields[1]->AsString = Edit1->Text;
```

Для доступа к конкретному поля могут также использоваться методы класса **TFields: FieldByName, FindField** и другие.

Примеры

См. примеры в разд. «FieldCount».

FieldValues

Свойство обеспечивает доступ к значениям всех полей активной записи.

Класс *TDataSet*

Определение

```
_ property System::Variant FieldValues[AnsiString FieldName]
```

Описание

Свойство **FieldValues** обеспечивает чтение и запись значений любых полей активной записи набора данных. Параметр **FieldName** указывает имя поля, с которым ведется работа. Это может быть любое поле: обычное поле записи, дочернее поле в сложной структуре, поле с совокупной характеристикой, содержащееся

в свойстве **AggFields.** Благодаря такой универсальности свойство **FieldValues,** как и метод **FieldByName.** имеет преимущества по сравнению со свойствами **Fields. FieldList** и **AggFields,** относящимися только к ограниченному множеству полей.

Тип свойства **FieldValues — Variant,** так что это свойство обеспечивает преобразование значений любых типов данных. Например, оператор

```
Edit1->Text = Table1->FieldValues["Year_b"];
```

заносит в окно редактирования **Edit1** значение поля **Year_b** — поля целого типа, обозначающего год рождения. А операторы

```
Table1->Edit();
Table1->FieldValues["Year_b"] = Edit1->Text;
```

заносят значение, введенное пользователем в окне **Editl,** в поле **Year_b.**

Надо отметить, что работа с типом **Variant** в свойстве **FieldValues** требует несколько больших временных затрат при преобразовании типов, чем использование **AsString** и др. аналогичных свойств.

FileName — свойство диалогов

Свойство диалогов, определяющее имя файла.

Классы *TOpenDialog, TOpenPictureDialog, TSavePictureDialog, TSavePictureDialog*

Определение

```
__property AnsiString FileName
```

Описание

В диалогах открытия и сохранения файлов свойство **FileName** определяет имя открываемого или сохраняемого файла. Перед вызовом диалога можно задать значение **FileName** как имя файла по умолчанию. А если пользователь в диалоге выбрал имя файла, то это имя можно прочитать в свойстве **FileName.**

Пример

Следующие операторы задают имя сохраняемого файла по умолчанию «Неизвестный.rtf», вызывают диалог сохранения файла и, если пользователь в диалоге нажал кнопку Сохранить, то текст окна редактирования **RichEdit1** сохраняется в заданном пользователем файле:

```
SaveDialog1->FileName = "Неизвестный.rtf";
if (OpenDialog1->Execute())
  RichEdit1->Lines->SaveToFile(SaveDialog1->FileName);
```

Filter — свойство диалогов

Задает фильтр в диалогах открытия и сохранения файлов.

Классы *TOpenDialog, TOpenPictureDialog, TSavePictureDialog, TSavePictureDialog*

Определения

```
__property AnsiString Filter
```

Описание

Свойство **Filter** определяет типы искомых файлов, появляющиеся в диалоге в выпадающем списке Тип файла. В процессе проектирования это свойство проще всего задать с помощью редактора фильтров, который вызывается нажатием кнопки с многоточием около имени этого свойства в Инспекторе Объектов. При этом открывается окно редактора. В его левой панели Filter Name вы записываете тот текст, который увидит пользователь в выпадающем списке Тип файла диалога. А в правой

панели окна редактора записываются разделенные точками с запятой шаблоны фильтра.

После выхода из окна редактирования фильтров заданные вами шаблоны появятся в свойстве **Filter** в виде строки. Например:

```
текстовые (*.txt, *.doc)|*.txt; *.doc|Все файлы|*.*
```

В этой строке тексты и шаблоны разделяются вертикальными линиями. В аналогичном виде, если требуется, можно задавать свойство **Filter** программно во время выполнения приложения.

В диалогах **TOpenDialog и TSaveDialog** значение по умолчанию пусто. В диалогах **TOpenPictureDialog и TSavePictureDialog** значение **Filter** по умолчанию включает следующие фильтры:

All (*.jpg;*.jpeg;*.bmp;*.ico;*.emf;*.wmf)	*.jpg;*.jpeg;*.bmp; *.ico;*.emf;*.wmf
JPEG Image File (*.jpg)	*.jpg
JPEG Image File (*.jpeg)	*.jpeg
Bitmaps (*.bmp)	*.bmp
Icons (*.ico)	*.ico
Enhanced Metafiles (*.emf)	*.emf
Metafiles (*.wmf)	*.wmf

В этих фильтрах перечислены все типы графических файлов, с которыми можно работать. Так что вам остается удалить, если хотите, фильтры тех файлов, с которыми вы не хотите работать, добавить, может быть, фильтр "Все файлы (*.*)" и перевести на русский язык названия типов.

Filter и Filtered — свойства наборов данных

Свойства, обеспечивающие фильтрацию набора данных.

Классы *TDataSet*, *TCustomClientDataSet*

Определения

```
__property AnsiString Filter
__property bool Filtered
```

Описание

Фильтрация данных, извлекаемых из набора данных, задается свойствами **Filter, Filtered и FilterOptions.** При наличии фильтра доступными становятся только те записи, поля которых удовлетворяют условиям фильтрации. Свойство **Filtered** включает или выключает использование фильтра. Свойство **FilterOptions** задает опции фильтрации. А сам фильтр записывается в свойство **Filter** в виде строки, содержащей определенные ограничения на значения полей. Например, значение **Filter**

```
Dep='Цех 1'
```

при Filtered = true, обеспечивает отображение только тех записей, в которых поле Dep имеет значение "Цех 1". В условиях сравнения строк можно использовать символ звездочки "*", который как в обычных шаблонах означает: "любое количество любых символов". Например, фильтр

```
Dep='Цех*'
```

приведет к отображению всех записей, в которых значение поля Dep начинается с "Цех". Но для того, чтобы это сработало, надо, чтобы в опциях, содержащихся

в свойстве **FilterOptions**, была выключена опция **foNoPartialCompare**, запрещающая частичное совпадение при сравнении.

При записи условий можно использовать операции отношения =, >, >=, <, <=, <>, а также логические операции and, or и not. Например, вы можете написать фильтр

```
(Dep='Цех 1')and(Year_b<=1970)and(Year_b>=1940)
```

и отобразятся записи сотрудников цеха 1, чей год рождения лежит в заданных пределах. Но использовать в фильтре имена вычисляемых полей не разрешается.

При фильтрации пустые записи пропускаются, если только их появление не указано в фильтре явным образом, как в следующем примере:

```
(Dep='Цех*') or (Dep = NULL)
```

Фильтрация на удаленных таблицах SQL и в клиентских наборах данных поддерживает сравнение полей (например: Fieldl > Field2) и множество других операций и функций, не поддерживаемых в локальных таблицах (Paradox, dBASE, Access, FoxPro). В частности:

Операция, функция	Пример	Комментарий
IS NULL	Dep IS NULL	
IS NOT NULL	Dep IS NOT NULL	
+	Year_b + 5 > 1970	для чисел, строк, дат (времени) + число
–	2002 - Year_b <> 19	для чисел, дат, дат (времени) - число
*	Fieldl * 100 > 20	для чисел
/	Fieldl > Field2 / 5	для чисел
Upper	Upper(Dep) = 'ЦЕХ 1'	
Lower	Lower(Fam + Nam) = 'ивановиван'	
Substring	Substring(Dep,5) = '1' Substring(Dep,1,3) = 'Цех'	подстрока, начиная с 5-го символа подстрока с 1-го по 3-ий символ (могут быть проблемы с кириллицей)
Trim	Trim(Fam + Nam) Trim(Field1, '–')	удаляет пробелы удаляет символы '–' в начале и конце строки
TrimLeft	TrimLeft(StringFieid) TrimLeft(Fieldl, '$') <> ''	
TrimRight	TrimRight(StringField) TrimRight(Field1, '.') <> ''	
Year	Year(DateField) = 2000	выделяет из даты год,
Month	Month(DateField) <> 12	месяц,
Day	Day(DateField) = 1	день,
Hour	Hour(DateField) < 16	час,
Minute	Minute(DateField) = 0	минуту,

Операция, функция	Пример	Комментарий
Second	Second(DateField) = 30	секунду
GetDate	GetDate - DateField > 7	текущая дата и время
Date	DateField = Date(GetDate)	дата из значения даты и времени
Time	TimeField > Time(GetDate)	время из значения даты и времени
Like	Memo LIKE '%filters%'	при применении к полям BLOB, свойство FilterOptions определяет чувствительность к регистру
In	Fam in ('Иванов','Сидоров')	отберет всех Ивановых и Сидоровых (второй операнд — список отбираемых значений)

Filtered

См. раздел «Filter и Filtered — свойства наборов данных».

FilterOptions

Опции фильтрации набора данных.

Класс *TDataSet*

Определения

```
enum TFilterOption { foCaseInsensitive, foNoPartialCompare };
 typedef Set<TFilterOption, foCaseInsensitive,
             foNoPartialCompare> TFilterOptions;

__property TFilterOptions FilterOptions
```

Описание

Свойство **FilterOptions** определяет опции фильтрации. Сама фильтрация задается свойствами **Filter**, **Filtered**, а **FilterOptions** устанавливает чувствительность фильтрации к регистру и допустимость частичного совпадения при сравнении. **FilterOptions** представляет собой множество следующих опций:

foCaseInsensitive	Учет регистра при сравнении строк
foNoPartialCompare	При включении этой опции в FilterOptions символ звездочки (*) воспринимается именно как символ. А когда эта опция не включена в FilterOptions, символ звездочки воспринимается как в обычных шаблонах: любой число любых символов

Например, если в свойстве **Filter** задано

```
Dep='Цех*'
```

то при включении в **FilterOptions** опции **foCaseInsensitive** и отсутствии опции **foNoPartialCompare** под условия фильтрации подходят значения поля Dep, равные "Цех 1", "цех 2", "цеха" и т.п. Если исключить из **FilterOptions** опцию **foCaseInsensitive,** то под условие фильтрации подойдет только первое из этих значений.

А если включить в **FilterOptions** опцию **foNoPartialCompare**, то ни одно из перечисленных значений не удовлетворит условию фильтрации. Это условие будет удовлетворено, только если значение Dep окажется равным 'Цех*'.

Font

Определяет атрибуты шрифта.

Класс *TControl*

Определение

```
__property Graphics::TFont* Font
```

Описание

Свойство **Font** является объектом типа **TFont**. Изменение шрифта можно осуществить или созданием нового объекта типа **TFont.** или изменением свойств **Color. Height. Name, Pitch. Size. Style** существующего объекта. См. подробнее методы свойства и события объекта **Font** и его установки по умолчанию в разделе **TFont.**

Примеры

1. Пусть на форме имеется компонент **Memo1**, в котором расположен некоторый текст, компонент **FontDialogl** — диалог выбора шрифта, и меню с разделом выбора шрифта, названным **Font.** Для того чтобы пользователь мог выбрать имя и атрибуты шрифта текста, отображаемого в Memol, в обработчик события **OnClick** раздела меню **Font** надо вставить операторы:

```
void__fastcall TForm1::FontClick(TObject *Sender)
{
  if (FontDialog1->Execute())
    Memo1->Font->Assign(FontDialog1->Font);
}
```

Если пользователь сменил атрибуты в диалоговом окне выбора шрифта, то метод **FontDialog1->Execute()** возвращает **true** и атрибуты шрифта компонента **Memol** устанавливаются равными выбранным пользователем.

2. Для того чтобы продемонстрировать доступные в системе шрифты, можно построить форму, содержащую выпадающий список ComboBoxl и компонент **Memol** с некоторым текстом. Приведенная ниже программа загружает в список доступные имена шрифтов и отображает первую строку списка (этот код вставлен в обработчик события формы **OnCreate**). В обработчик события **OnClick** компонента **ComboBoxl** вставлен оператор, меняющий подсвойство **Name** в свойстве **Font** компонента **Memol.**

```
void__fastcall TForm1::FormCreate(TObject *Sender)
{
 ComboBox1->Items = Screen->Fonts;
 ComboBox1->ItemIndex = 0;
 Memo1->Font->Name = ComboBox1->Items->Strings[0];
}

void__fastcall TForm1::ComboBox1Click(TObject *Sender)
{
 Memo1->Font->Name = ComboBox1->Items->Strings[ComboBox1->ItemIndex];
}
```

3. Для того чтобы посмотреть влияние на отображаемый текст свойства **Charset**, определяющего набор символов шрифта, можно добавить на форму компонент редактирования (например, типа **TCSpinEdit**) и кнопку, в событие **OnClick** которой ввести оператор:

```
Memol->Font->Charset = CSpinEdit1->Value;
```

4. Чтобы посмотреть влияние на отображаемый текст свойства **Pitch**, можно добавить на форму еще один компонент ComboBox, задать в его свойстве Items строки "fpDefault", "fpFixed" и "fpVariable", а в событие OnClick вставить код:

```
switch (ComboBox2->ItemIndex)
{
  case 0: Memo1->Font->Pitch = fpDefault;
          break;
  case 1: Memo1->Font->Pitch = fpFixed;
          break;
  case 2: Memo1->Font->Pitch = fpVariable;
}
```

GroupIndex — свойство разделов меню

Определяет логическую группу, к которой относится раздел главного меню и которая используется при объединении меню нескольких форм.

Класс *TMenuItem*

Определение

```
__property Byte GroupIndex
```

Описание

Свойство **GroupIndex** определяет способ объединения меню. В приложениях MDI меню дочерних форм всегда объединяются с меню родительской формы. В приложениях с несколькими формами наличие или отсутствие объединение определяется свойством **AutoMerge** компонента типа **TMainMenu.**

По умолчанию все разделы меню имеют одинаковое значение **GroupIndex.** Если требуется объединение меню, то разделам надо задать не убывающие номера свойств **GroupIndex.** Тогда, если разделы встраиваемого меню имеют те же значения, что и какие-то разделы меню основной формы, то эти разделы заменяют соответствующие разделы основного меню. В противном случае разделы вспомогательного меню встраиваются между элементами основного меню в соответствии с номерами **GroupIndex.** Если встраиваемый раздел имеет **GroupIndex** меньший, чем любой из разделов основного меню, то разделы встраиваются в начало.

Когда активизирован объект, созданный сервером OLE 2.0, сервер может попытаться объединить свое меню с меню контейнера приложения. Тогда свойство **GroupIndex** используется для замены до трех разделов главного меню. Для замещения приложение сервера использует следующие предопределенные значения **GroupIndex:**

Группа	Индекс	Описание
Edit	1	Разделы меню сервера, связанные с редактированием активного объекта OLE
View	3	Разделы меню сервера, связанные с отображением активного объекта OLE
Help	5	Разделы меню сервера, связанные с доступом к встроенной справке

Свойство **GroupIndex** может использоваться и еще для одной цели: создания группы разделов, работающих по принципам радиокнопок. Для этого во всех разделах группы надо установить в **true** свойство **RadioItem** и задать всем разделам одинаковое значение **GroupIndex.** Выбор пользователем одного из таких разделов будет снимать выделение с остальных разделов.

Пример

Если в основной и вторичной формах структуры меню имеет следующие значения **GroupIndex:**

```
Форма 1        Форма 2
2  -  4        1  -  3
|     |        |     |
2     4        1     3
|     |        |
2     4        1
```

то в момент, когда активизируется вторая форма, в первой появляется меню со структурой:

```
1  -  2  -  3  -  4
|     |     |     |
1     2     3     4
|     |           |
1     2           4
```

Handle

Обеспечивает доступ к дескриптору окна.

Класс *TWinControl*

Доступ *только для чтения*

Определение

```
__property HWND Handle
```

Описание

Свойство **Handle** используется при обращении к функциям API Windows, требующим указания дескриптора окна.

Обращение к свойству **Handle** приводит к созданию дескриптора, если его не было до этого. Поэтому нельзя обращаться к этому свойству при создании компонента или чтении его из потока.

HasConstraints

Указывает, имеются ли ограничения на вводимые значения поля.

Класс *TFleld*

Доступ только *для чтения*

Определение

```
__property bool HasConstraints
```

Описание

Свойство **HasConstraints** определяет, имеет ли объект поля ограничения на вводимые значения в свойствах **CustomConstraint, DefaultExpression, ImportedConstraint.** Во всех этих случаях **HasConstraints = true.** Если же ни одно из этих свойств не установлено, то **HasConstraints = false,** что указывает на отсутствие ограничений со стороны сервера и BDE. Впрочем, значение **HasConstraints = false** еще не означает отсутствие каких-либо ограничений. Например, ограничения могут задаваться маской **EditMask** или в обработчике события **OnValidate.** Но это ограничения приложения, а не сервера или BDE.

Height — свойство компонента

Определяет высоту компонента или формы в пикселах.

Класс *TControl*

Определение

```
__property int Height
```

Описание

Свойство **Height** определяет вертикальный размер компонента или формы в пикселах. Используется для изменения высоты компонента при изменениях размеров окна приложения. См. разд. **ClientHeight, ClientRect.**

Height — свойство шрифта

Определяет высоту шрифта в пикселах.

Класс *TFont*

Определение

```
property Height: Integer;
```

Описание

Свойство **Height** определяет высоту шрифта в пикселах. Если значение **Height** задано отрицательным, то в размер не входит верхний пиксел каждой строки.

Обычно для задания размера используется не **Height,** а другое свойство: Size — размер шрифта в кеглях или в пунктах, принятых в Windows.

Значение **Height** связано со свойствами **Size и PixelsPerInch** (число пикселов на дюйм — см. **TFont)** соотношением:

```
Font->Height = -Font->Size * Font->PixelsPerInch / 72
```

Из этого соотношения, в частности, видно, что задание положительного значения **Size** ведет к отрицательному значению **Height.** Можно задавать Size отрицательным; тогда **Height** будет положительным.

HelpContext — свойство оконных компонентов

Определяет номер, используемый в контекстно-зависимой справке.

Класс *TControl*

Определение

```
typedef int THelpContext;

__property Classes::THelpContext HelpContext
```

Описание

Значение **HelpContext** задается для определения уникального номера темы контекстно-зависимой справки (этот номер определяется в файле проекта справки). В этом отличие свойства **HelpContext** от введенного в C++Builder 6 свойства **HelpKeyword,** задающего непосредственно идентификатор темы справки. Экран с темой, указанной одним из этих свойств, будет показан пользователю, если он нажмет клавишу F1, когда данный компонент в фокусе.

Начиная с C++Builder 6 свойство **HelpContext** работает только в том случае, если другое свойство компонента — **HelpType** равно **htContext.**

Если значение **HelpContext** равно 0, то компонент наследует значение **HelpContext** своего родительского элемента (элемента-контейнера, в котором он расположен). Например, если задано значение **HelpContext** для формы, то для всех компонентов, у которых не задано отличного от нуля значения **HelpContext,** будет показана тема справки, указанная для формы.

В версиях C++Builder, младше C++Builder 6, свойство **HelpContext** было объявлено в классе **TWinControl.**

HelpKeyword — свойство оконных компонентов

Определяет идентификатор темы контекстно-зависимой справки.

Класс *TControl*

Определение

```
__property AnsiString HelpKeyword
```

Описание

Значение **HelpKeyword** (свойство введено в C++Builder 6) определяет идентификатор темы контекстно-зависимой справки. В этом его отличие от свойства **HelpContext**, указывающего номер темы. Экран с указанной темой будет показан пользователю, если он нажмет клавишу F1, когда данный компонент в фокусе.

Свойство **HelpContext** работает только в том случае, если другое свойство компонента — **HelpType** равно **htKeyword**.

HelpType — свойство оконных компонентов

Определяет, какое свойство компонента указывает тему контекстно-зависимой справки.

Класс *TControl*

Определение

```
enum THelpType {htKeyword, htContext};
__property Classes::THelpType HelpType
```

Описание

Значение **HelpType** (свойство введено в C++Builder 6) определяет, какое из свойств компонента указывает тему контекстно-зависимой справки, связанной с данным компонентом. Экран с этой темой будет показан пользователю, если он нажмет клавишу F1, когда данный компонент в фокусе.

При значении **HelpType = htContext** тема определяется свойством **HelpContext** — номером темы в файле проекта справки. При значении **HelpType = htKeyword** тема определяется свойством **HelpKeyword** — идентификатором темы справки.

Hint

Содержит текст, отображаемый в окне подсказки или в строке состояния.

Классы *TControl, TApplication*

Определение

```
__property AnsiString Hint
```

Описание

Свойство **Hint** компонента обеспечивает текст подсказки, появляющийся в ярлычке (всплывающем окне подсказки) или в заданном месте окна, например, в строке состояния.

В общем случае **Hint** состоит из двух частей, разделенных символом вертикальной черты '|'. Первая часть отображается в ярлычке, если пользователь задержит курсор мыши над данным компонентом (это может быть любой компонент, включая разделы меню). Обычно первая часть содержит краткое пояснение компонента. В частности, такой подсказкой как правило снабжаются быстрые кнопки типа **TSpeedButton**. Вторая часть содержит текст, отображаемый в какой-то выделенной для этого части окна, например, в строке состояния. Это обычно развернутое пояснение. Например, свойство **Hint** для быстрой кнопки доступа к разделу меню сохранения файла может иметь вид: 'Сохранить|Сохранение текущего доку-

мента в файле'. Как частный случай, в свойстве **Hint** может быть задана только первая часть подсказки без символа '|'.

Для того чтобы первая часть подсказки появлялась в ярлычке, когда пользователь задержит курсор мыши над данным компонентом, надо сделать следующее:

1. Указать тексты свойства **Hint** для всех компонентов, для которых вы хотите обеспечить окно подсказки.
2. Установить свойства **ShowHint** (показать подсказку) этих компонентов в **true** или установить в **true** свойство **ParentShowHint (ShowHint** родителя) и установить в **true** свойство **ShowHint** контейнера, содержащего данные компоненты, или формы.

Конечно, вы можете устанавливать свойства *в* **true** или **false** программно, включая и отключая подсказки в различных режимах работы приложения.

При ShowHint. равном true, ярлычок будет всплывать даже если компонент в данный момент недоступен **(Enabled = false).**

Если вы не задали значение свойства компонента **Hint,** но установили в **true** свойство **ShowHint** или установили в **true** свойство ParentShowHint. а в родительском компоненте **ShowHint = true,** то в ярлычке будет отображаться текст **Hint** из родительского компонента.

Правда, все описанное выше справедливо при значении свойства **ShowHint** приложения (объекта **Application),** равном **true** (это значение задано по умолчанию). Если установить **Application.ShowHint в false,** то ярлычки не будут появляться, независимо от значений **ShowHint в** любых компонентах.

Для того чтобы вторая часть сообщения, записанного в **Hint,** отображалась в строке состояния в моменты, когда курсор мыши проходит над компонентом, надо использовать обработку события **OnHint.** Это событие именно приложения **Application,** а не того компонента, над которым проходит курсор мыши. Если обработчик этого события определен, то в момент прохождения курсора над компонентом, в котором задано свойство **Hint,** вторая часть сообщения компонента заносится в свойство **Hint** объекта **Application.** Если свойство **Hint** компонента содержит только одну часть, то в свойство **Hint** объекта **Application** заносится эта первая часть. Причем все это делается независимо от состояния свойства компонента **ShowHint.**

Чтобы отображать эти сообщения в строке состояния, надо определить и написать обработчик события приложения **OnHint,** как показано в приведенном ниже примере.

Третий способ использования свойства **Hint** компонента заключается в непосредственном отображении текста заключенного в нем сообщения в какой-то метке или панели с помощью функций **GetShortHint и GetLongHint.** первая из которых возвращает первую часть сообщения, а вторая — вторую (если второй части нет, то возвращается первая часть).

Примеры

Пусть вы хотите отображать вторую часть подсказок, введенных в свойство **Hint** компонентов, в полосе состояния **StatusBar.** Поместите на форму панель состояния — компонент **StatusBar** со страницы Win32 и установите его свойство **SimplePanel в true.** Тем самым вы превратите панель в односекционную. Если вы хотите использовать многосекционную панель, то в приведенном ниже операторе надо заменить свойство **SimpleText** ссылкой на панель, в которой вы хотите отображать подсказку. Перенесите на форму компонент **ApplicationEvents,** и в обработчик его события **OnHint** вставьте оператор

```
StatusBar1->SimpleText = Application->Hint;
```

Рассмотрим другой пример. Пусть вы хотите, чтобы при нажатии некоторой кнопки **Button1** вашего приложения в панели **Panell** высвечивалась подсказка

пользователю, например, «Укажите имя файла», а сама кнопка имела ярлычок с текстом «Ввод». Тогда вы можете задать свойству **Hint** этой кнопки значение 'Ввод|Укажите имя файла', задать значение **true** свойству **ShowHint**, а в обработчик события нажатия этой кнопки вставить оператор

```
Panel1->Caption = GetLongHint(Button1->Hint);
```

Если же вы не хотите отображать ярлычок для кнопки, то можете ограничиться значением **Hint,** равным "Укажите имя файла", а приведенный выше оператор оставить неизменным или заменить на эквивалентный ему в данном случае оператор

```
Panel1->Caption = GetShortHint(Button1->Hint);
```

или даже просто на оператор

```
Panel1->Caption = Button1->Hint;
```

HostDockSite

Определяет контейнер, в который встроен данный компонент.

Класс *TControl*

Определение

```
__property TWinControl* HostDockSite
```

Описание

Свойство **HostDockSite** определяет контейнер, в который встроен данный компонент. Если компонент находится в состоянии «плавающего» окна, то **HostDockSite** — временный объект типа **FloatingDockSiteClass** или **NULL.**

Обычно для программного встраивания компонента лучше использовать метод **ManualDock,** а не задание **HostDockSite.** Установка **HostDockSite** приводит к немедленному встраиванию компонента в указанный контейнер, но не устанавливает позицию компонента, его выравнивание и не воспроизводит событий встраивания.

Для встроенных компонентов значение **HostDockSite** совпадает со значением **Parent.** Но если компонент никуда не встроен, то **HostDockSite = NULL, a Parent** указывает на контейнер, содержащий данный компонент.

ГО

Уникальный идентификатор объекта в собрании **TCollection.**

Класс *TCollectionItem*
Доступ *только для чтения*

Определение

```
__property int ID
```

Описание

Свойство **ID** определяет уникальный идентификатор объекта **TCollectionItem.** Обычно значение **ID** совпадает с индексом (свойством **Index**) в момент включения объекта в собрание **TCollection.** Но индекс **Index** может изменяться при вставке или удалении других объектов собрания. А идентификатор **ID** остается неизменным. Если объект удален из собрания, его идентификатор в дальнейшем использовать невозможно.

Поиск объекта по его идентификатору осуществляется методом **FindItemID** того собрания **TCollection,** к которому относится объект.

ImageIndex — свойство раздела меню

Индекс изображения раздела в списке изображений родительского меню.

Класс *TMenuItem*

Определение

```
__property int ImageIndex
```

Описание

Свойство **ImageIndex** указывает индекс изображения, появляющегося левее надписи данного раздела меню. Индекс относится к списку изображений — свойству **Images** родительского меню типа **TMenu** или **TPopupMenu** и начинается с 0.

Если родительское меню не содержит списка изображений (свойство **Images** равно **NULL),** то для отображения изображения надо использовать свойство **Bitmap** компонента типа **TMenuItem. Но Bitmap** действует только при **Images** равном **NULL** и отрицательном значении **ImageIndex.** В противном случае значение **Bitmap** не принимается во внимание.

ImportedConstraint

Строка SQL, накладывающая ограничение со стороны сервера на значение поля.

Класс *TField*

Определение

```
__property AnsiString ImportedConstraint
```

Описание

Свойство **ImportedConstraint** позволяет прочитать ограничение в виде строки SQL на значение поля, переданное сервером. Например, оно может иметь вид

```
Value > 0 and Value < 100
```

что означает допустимый интервал значений 0-100. Значение **ImportedConstraint** не следует изменять. Для ввода дополнительных ограничений служит свойство **CustomConstraint.**

Принудительное удаление строки в свойстве **ImportedConstraint** не изменит ограничение, заданное сервером. Просто проверка ограничения будет проводиться не локально, а самим сервером. Это означает, что при нарушении ограничения вместо сообщения, заданного свойством **ConstraintErrorMessage,** будет генерироваться сообщение сервера об ошибке.

Index — свойство TCollectionItem

Индекс объекта в свойстве **Items** его контейнера **TCollection.**

Класс *TCollectionItem*

Определение

```
__property int Index
```

Описание

Свойство **Index** определяет позицию объекта в свойстве **Items** его контейнера (собрания) **TCollection.** Индексация начинается с 0. При вставке или удалении других объектов собрания **Index** может изменяться. В этом его отличие от идентификатор **ID.** который остается неизменным.

Index — свойство TField

Индекс объекта поля в свойстве **Fields** набора данных.

Класс *TField*

Определение

```
__property int Index
```

Описание

Свойство **Index** определяет индекс объекта поля в свойстве **Fields** набора данных. Это свойство позволяет определить или изменить индекс поля. Изменение индекса переместит колонку поля на новое место в таких компонентах отображения данных, как **TDBGrid.** При этом, конечно, физическое положение поля в таблице базы данных не изменится.

IndexDefs

Свойство, содержащее информацию об индексах таблицы.

Класс *TTable*

Определение

```
__property Db::TIndexDefs* IndexDefs
```

Описание

Свойство **IndexDefs** определяет объект типа **TIndexDefs,** свойства которого содержат информацию об индексах таблицы, связанной с компонентом типа TTaBle. Свойства и методы этого объекта позволяют:

- Получить доступ к определениям индексов
- Определить количество имеющихся индексов
- Добавлять или удалять индексы
- Копировать определения всех индексов из свойства **IndexDefs** одной таблицы в другую таблицу

См. пример применения свойства **IndexDefs** в гл. 4, **в** разд. «CreateTable».

IndexOf — метод TFields

Возвращает номер индекса объекта **TField** в списке **TFields.**

Класс *TFields*

Определение

```
int __fastcall IndexOf(TField* Field)
```

Описание

Метод **IndexOf** возвращает номер индекса объекта **Field** типа **TField** в списке **Fields** типа **TFields.** Индексы начинаются с 0. Если поле в списке не найдено, возвращается **-1.**

InternalCalcField

Определяет, является ли поле вычисляемым в источнике данных.

Класс *TFieldDef*

Определение

```
__property bool InternalCalcField
```

Описание

Свойство **InternalCalcField** определяет, является ли поле вычисляемым в источнике данных (при значении **true),** или оно имеется в таблице набора данных. Например, если выполняется запрос

```
SELECT Fam, Nam, (2002-Year_b) FROM Pers
```

и набор представляет собой «живые» данные (свойство набора **RequestLive** равно **true),** то описание поля, соответствующего выражению (2002–Year_b), имеет значение **InternalCalcField = true.**

Следует различать поля с **InternalCalcField = true** и вычисляемые поля, значения которых просчитываются в обработчиках событий **OnCalcFields.**

ItemClass

Класс объектов, содержащихся в контейнере **TCollection.**

Класс *TCollection*

Доступ *только для чтения*

Определение

```
type TCollectionItemClass = class of TCollectionItem;

__property System::TMetaClass* ItemClass
```

Описание

Свойство **ItemClass** определяет класс объектов, содержащихся в контейнере **TCollection** — в массиве **Items.** Каждому классу-наследнику **TCollection** соответствует свой класс-наследник **TCollectionItem.** Этот класс и указывается свойством **ItemClass.**

Items — свойство TCollection

Индексированный массив объектов, хранящихся в **TCollection.**

Класс *TCollection*

Определение

```
__property TCollectionItem* Items[int Index]
```

Описание

Свойство Items обеспечивает индексированный доступ к объектам, хранящимся в **TCollection** — объектам типа **TCollectionItem.** Индекс соответствует свойству **Index** соответствующего объекта **TCollectionItem.** Индексы начинаются с 0.

Items — свойство TFieldDefs

Индексированный список описаний полей.

Класс *TFieldDefs*

Определение

```
__property TFieldDef* Items[int Index]
```

Описание

Свойство **Items** позволяет получить доступ к описанию **TFieldDef** каждого поля набора данных. Параметр **Index** определяет индекс описания в списке (начинается с 0). Например, **Table1->FieldDefs->Items[0]->DisplayName** — имя первого поля набора данных компонента **Table1.**

Пример

В следующем примере выпадающий список **ComboBox1** заполняется именами полей таблицы, соединенной с компонентом **Table1.**

```
Table1->Active = true;
ComboBox1->Clear();
for (int i=0; i < Table1->FieldDefs->Count; i++)
  ComboBox1->Items->Add(Table1->FieldDefs->Items[i]->Name);
```

Items — свойство TList

Элементы массива указателей в объекте класса TList.

Класс *TList*

Определение

```
__property void * Items[int Index]
```

Описание

Свойство **Items** дает доступ к указателям, хранящимся в объекте класса **TList.** Параметр **Index** является индексом массива указателей. Индексы изменяются, начиная с 0 (0 — индекс первого указателя).

Свойство **Items** позволяет получить доступ и к объектам, на которые указывают элементы массива. Но при этом надо учитывать, что свойство **Items** имеет тип **pointer,** т.е. является нетипизированным указателем. Такой указатель нельзя разыменовать непосредственно. Надо использовать явное или неявное приведение типов. Примеры этого даны ниже, а также в разделе **TList.**

При организации циклов с просмотром указателей, содержащихся в **Items,** надо иметь в виду, что часть из них может быть равна **NULL.** Если это может нарушить работу алгоритма, то предварительно надо применить к объекту TList метод **Pack,** который сократит размер массива, удалив из него нулевые указатели.

Пример

См. пример в гл. 1, в описании класса TList.

KeyFields, Lookup, LookupKeyFields, LookupDataSet, LookupResultField

Свойства, характеризующие поле просмотра — т.е. поле, значение которого переносится в текущий набор данных из другого просматриваемого набора данных.

Класс *TField*

Определения

```
__property bool Lookup
__property AnsiString KeyFields
__property AnsiString LookupKeyFields
__property TDataSet* LookupDataSet
__property AnsiString LookupResultField
```

Описание

Поле просмотра — это поле, значение которого переносится в текущий набор данных, заданный свойством **DataSet,** из другого просматриваемого набора данных, заданного свойством **LookupDataSet.** Если данное поле является полем просмотра, его свойство **Lookup** устанавливается в **true.** Свойство **FieldKind** должно равняться **fkLookup.**

Имя поля в **LookupDataSet,** значение которого переносится в свойство **Value** данного поля, задается свойством **LookupResultField.** Запись в **LookupDataSet,** из которой берется значение **LookupResultField,** определяется следующим образом. В свойствах **KeyFields и LookupKeyFields** перечисляются ключевые поля наборов данных **DataSet и LookupDataSet.** Это те поля двух наборов данных, значения которые должны совпадать. Если имеется несколько ключевых полей, то они разделяются в списках **KeyFields и LookupKeyFields** точками с запятой. Последовательность перечисления полей в этих списках должна быть взаимно согласована. Т.е. сравниваются значения первых полей, значения вторых и т.д. Типы сравниваемых полей должны совпадать друг с другом. Если в **LookupDataSet** нашлась запись, в которой значения всех полей списка **LookupKeyFields** совпадают со значениями полей списка **KeyFields** текущей записи набора данных **DataSet,** то в данное поле заносится значение поля **LookupResultField.**

Left

Координата левого края компонента в пикселах.

Класс *TControl*

Определение

```
__property int Left
```

Описание

Свойство **Left** определяет координату левого края компонента в пикселах. Для компонентов за начало отсчета берется левая граница клиентской области родителя (например, панели, если данный компонент расположен на панели, или формы, если компонент расположен непосредственно на форме). Для формы координата **Left** представляет собой горизонтальную координату экрана, отсчитываемую от его левого края.

Свойство **Left** используется при перемещениях и изменениях размеров компонентов (см. примеры в разд. **BoundsRect и Components).**

List

Указатель на массив указателей в объекте класса **TList.**

Класс *TList*

Доступ *только для чтения*

Определение

```
typedef void *TPointerList[134217727];
typedef TPointerList *PPointerList;

__property PPointerList List
```

Описание

Свойство List дает непосредственный доступ к массиву **Items** — массиву указателей объекта типа **TList.** Например, **MyList->Items[0] и MyList->List[0]** указывают на один и тот же элемент массива.

Пример

См. пример в гл. 1, в описании класса TList.

Lookup

См. раздел «KeyFields, Lookup, LookupKeyFields, LookupDataSet, LookupResultField».

LookupCache

Определяет должны ли значения поля просмотра кэшироваться, или просмотр должен осуществляться при каждом изменении текущей записи.

Класс *TField*

Определение

```
__property bool LookupCache
```

Описание

Свойство **LookupCache** имеет смысл только для полей просмотра, в которых **FieldKind = fkLookup.** Задание **LookupCache** равного **true** определяет, что просмотр набора данных **LookupDataSet** производится однократно при открытии набора **PataSet.** и значения поле просмотра кэшируются. Этот кэш определяется свойством **LookupList** и содержит значения поля **LookupResultField** для каждого множества ключевых полей списка **LookupKeyFields**. При смене текущей записи набора **DataSet** значение поля просмотра берется из кэша. Это существенно повышает эффективность работы, особенно в случаях, когда доступ к просматриваемо-

му набору данных **LookupDataSet** должен осуществляться по сравнительно медленной сети. С другой стороны, кэширование требует затрат вычислительных ресурсов клиента. Эти затраты могут стать особенно непомерными, если в различных записях набора **DataSet** используются разные наборы ключевых полей **KeyFields.** Для сокращения затрат вычислительных ресурсов клиента имеет смысл задавать **LookupCache = false.**

При изменении во время выполнения приложения просматриваемого набора данных **LookupDataSet** просмотр в кэше может дать неправильные результаты. В подобных случаях надо обновлять **LookupList** с помощью метода **RefreshLookupList.** При установке в **true** свойства **LookupCache** во время выполнения необходимо также вызывать **RefreshLookupList.**

LookupDataSet

См. раздел «KeyFields, Lookup, LookupKeyFields, LookupDataSet, LookupResultField».

LookupKeyFields

См. раздел «KeyFields, Lookup, LookupKeyFields, LookupDataSet, LookupResultField».

LookupList и RefreshLookupList

Кэш значений набора данных **LookupDataSet,** индексированных множеством полей **KeyFields,** и метод обновления этого кэша.

Класс *TField*

Определения

```
__property TLookupList* LookupList // только для чтения
void __fastcall RefreshLookupList(void);
```

Описание

Свойство **LookupList** определяет (только при кэшировании полей просмотра, т.е. при **FieldKind = fkLookup и при LookupCache = true**) кэш значений набора данных **LookupDataSet,** индексированных множеством полей **KeyFields.**

Значения, хранящиеся в **LookupList,** устанавливаются при открытии **DataSet** и при выполнении метода **RefreshLookupList.** Заполнение производится в соответствии со списком **LookupKeyFields.** Этот метод надо вызывать явно, если во время выполнения производится установка свойства **LookupCache** и если изменяются значения **LookupResultField** или **LookupKeyFields в LookupDataSet.**

Приложение может задавать значения в **LookupList** программно, подменяя таким образом реальный просматриваемый набор данных (его может вообще не быть).

Свойство **LookupList** имеет тип **TLookupList.** Объект **TLookupList** генерируется программно, если задается значение свойства поля **LookupList** и устанавливается в **true** свойство **LookupCache.** Если при этом другие свойства, связанные с полями просмотра, не установлены, то кэш **LookupList** не заполняется автоматически значениями из просматриваемого набора данных и, значит, может заполняться программно.

В классе **TLookupList,** наследующем **TObject,** отсутствуют свойства, но определены следующие основные методы:

Метод	Описание
Add	Добавляет программно строку в список LookupList, не связанный с реальным просматриваемым набором данных

Метод	Описание
Clear	Очищает кэш, освобождая память. Этот метод надо вызывать перед программным заполнением списка LookupList
ValueOfKey	Возвращает значение, соответствующее указанному списку ключевых полей

Помимо этого наследуются все методы класса **TObject.**

Метод **Add** объявлен как

```
void__fastcall Add(const System::Variant &AKey,
                   const System::Variant &AValue);
```

Параметр **AKey** типа **Variant** или массива **Variant** представляет одно или более значений ключевых полей. Параметр **AValue** типа **Variant** задает значение поля просмотра, связанного с ключевыми полями.

Метод **ValueOfKey** объявлен как

```
System::Variant__fastcall ValueOfKey(const System::Variant &AKey);
```

Параметр AKey типа **Variant** или массива **Variant** представляет одно или более значений ключевых полей в просматриваемом наборе данных. Метод возвращает найденное значение поля **LookupKeyFields.** Таким образом, значение поля просмотра определяется выражением:

```
Value = LookupList->ValueOfKey(DataSet->FieldValues[KeyFields]);
```

LookupResultField

См. раздел «KeyFields, Lookup, LookupKeyFields, LookupDataSet, LookupResultField».

MasterSource

Во вспомогательной таблице определяет источник данных головной таблицы.

Класс *TTable*

Определение

```
__property Db::TDataSource* MasterSource
```

Описание

Свойство **MasterSource** во вспомогательной таблице определяет источник данных головной таблицы, т.е. компонент **DataSource.** в свойстве **DataSet** которого содержится ссылка на головной набор данных.

После установки **MasterSource** можно задавать ключевые поля **MasterFields,** совпадение которых обеспечивает связь головной и вспомогательной таблиц.

MinValue и MaxValue

Определяют ограничения на значение поля

Классы *TIntegerField, TLargeIntField, TFloatField, TBCDFleld*

Определения

```
__property int MinValue          // для TIntegerField
__property int MaxValue          // для TIntegerField

__property__int64 MinValue       // для TLargeIntField
__property__int64 MaxValue       // для TLargeIntField

__property double MinValue       // для TFloatField
```

```
__property double MaxValue          // для TFloatField

__property System::Currency MinValue // для TBCDField
__property System::Currency MaxValue // для TBCDField
```

Описание

Свойства **MinValue и MaxValue** определяют ограничения минимального и максимального значений, вводимых в поле. Если значения этих свойств равны нулю, значит ограничение не накладывается (кроме естественных ограничений, свойственных типу вводимых значений). При нарушении установленных ограничений генерируется исключение **EDatabaseError**.

Mode — свойство TBatchMove

Свойство, определяющее, что происходит при выполнении метода **Execute** объекта **TBatchMove**

Класс *TBatchMove*

Определение

```
enum TBatchMode { batAppend, batUpdate, batAppendUpdate,
                  batDelete, batCopy };
__property TBatchMode Mode
```

Описание

Свойство **Mode** определяет, что именно делает объект **BatchMove.** Свойство **Mode** может иметь следующие значения:

batAppend	Записи из источника данных добавляются в приемник, не изменяя существующих там записей. Таблица-приемник должна существовать до начала переноса данных
BatUpdate	Записи в таблице-приемнике с ключевыми полями, соответствующими полям в таблице-источнике, изменяются на записи из источника. Новые записи в приемник не добавляются. Таблица-приемник должна существовать до начала переноса данных и должна иметь индекс
BatAppend Update	Записи в таблице-приемнике с ключевыми полями, соответствующими полям в таблице-источнике, изменяются на записи из источника. Записи из таблицы источника, которые не имеют соответствия в приемнике, добавляются туда. Таблица-приемник должна существовать до начала переноса данных и должна иметь индекс
batDelete	Записи в таблице-приемнике, которым находится соответствие в источнике, удаляются из приемника. Таблица-приемник должна существовать до начала удаления данных и должна иметь индекс
batCopy	Таблица-приемник создается и заполняется записями источника. Если таблица-приемник уже существовала, то ее содержимое заменяется содержимым источника

Mode — свойство TPen

Определяет режим рисования пером на канве.

Класс *TPen*

Определение

```
enum TPenMode {pmBlack, pmWhite, pmNop, pmNot, pmCopy,
               pmNotCopy, pmMergePenNot, pmMaskPenNot,
```

```
pmMergeNotPen, pmMaskNotPen, pmMerge,
pmNotMerge, pmMask, pmNotMask, pmXor, pmNotXor};

__property TPenMode Mode
```

Описание

Свойство пера **Mode** определяет, каким образом взаимодействуют цвета пера и канвы. Выбор значения **Mode** позволяет получать различные эффекты.

Возможные значения **Mode:**

Режим	Цвет пиксела
pmBlack	Всегда черный
pmWhite	Всегда белый
pmNop	Неизменный
pmNot	Инверсный по отношению к цвету фона канвы
pmCopy	Цвет, указанный в свойстве Color пера Реп: это значение принято по умолчанию
pmNotCopy	Инверсия цвета пера
pmMergePenNot	Комбинация цвета пера и инверсного цвета фона канвы
pmMaskPenNot	Комбинация цветов, общих для цвета пера и инверсного цвета фона канвы
pmMergeNotPen	Комбинация цвета фона канвы и инверсного цвета пера
pmMaskNotPen	Комбинация цветов, общих для цвета фона канвы и инверсного цвета пера
pmMerge	Комбинация цвета пера и цвета фона канвы
pmNotMerge	Инверсия режима pmMerge: комбинации цвета пера и цвета фона канвы
pmMask	Комбинация цветов, общих для цвета фона канвы и цвета пера
pmNotMask	Инверсия режима pmMask: комбинации цветов, общих для цвета фона канвы и цвета пера
pmXor	Операция **xor**: комбинация цветов или пера, или фона канвы, но не обоих
pmNotXor	Инверсия режима pmXor: комбинации цветов или пера, или фона канвы, но не обоих

Name — свойство TComponent

Имя компонента, по которому на него ссылаются другие компоненты.

Класс *TComponent*

Определение

```
__property AnsiString Name
```

Описание

Свойство **Name** определяет имя компонента, которое используется в дальнейшем в программе. Задается только в процессе проектирования и не должно изменяться во время выполнения. Иначе операторы программы могут оказаться неправильными.

По умолчанию C++Builder сама присваивает имена компонентам, размещаемым на форме. Эти имена по умолчанию надо изменять в процессе проектирования на осмысленные. Иначе при сколько-нибудь сложной форме вы сами через некоторое время не сможете понять, что такое **Panel7** или **Button13.**

Пример использования свойства **Name** см. в разделе «Components».

Name — свойство TNamedItem

Имя входа в базу данных.

Класс *TNamedItem*

Определение

```
__property System::AnsiString Name
```

Описание

Свойство **Name** определяет имя объекта входа в базу данных. Если этот объект **TNamedItem** — описание поля, то **Name** — это имя поля таблицы.

Пример

См. пример в гл. 4, в разд. «CreateTable».

NewValue

См. раздел «Value, NewValue, OldValue».

ObjectView

См. раздел «Fields и ObjectView».

OldValue

См. раздел «Value, NewValue, OldValue».

Origin

Имя поля в наборе данных, включая имена базы данных и таблицы.

Класс *TField*

Определение

```
__property AnsiString Origin
```

Описание

Свойство **Origin** устанавливается только во время проектирования и только при использовании компонента **Query.** Значение **Origin** складывается из имени таблицы и имени поля. Например, при следующем запросе SQL к базе данных Paradox с именем **DBP:**

```
SELECT Fam AS F FROM PERS
```

значение **Origin** будет равно "DBP."PERS.DB".Fam", а значение **FieldName — "F".**

Owner

Определяет владельца данного компонента.

Класс *TComponent*

Доступ *только для чтения*

Определение

```
__property TComponent* Owner
```

Описание

Свойство Owner определяет владельца данного компонента. Форма является владельцем всех расположенных на ней компонентов. В свою очередь Application является владельцем всех форм. Когда освобождается память, занимавшаяся владельцем, автоматически освобождается память всех компонентов, которыми он владел.

Parent

Определяет родительский компонент, в площади которого располагается данный компонент.

Класс *TControl*

Определение

```
__property TWinControl* Parent
```

Описание

Свойство **Parent** определяет родительский компонент, т.е. компонент-контейнер, содержащий данный компонент. Контейнерами являются оконные компоненты, такие, как формы, панели и некоторые другие. Расположенные на них дочерние компоненты могут наследовать часть свойств содержащего их контейнера, например, шрифт, цвет, отображение ярлычков подсказки, трехмерность. Для этого должны быть установлены в **true** свойства дочерних компонентов **ParentFont, ParentColor, ParentShowHint, ParentCtl3D.**

Надо различать два похожих свойства: **Parent** — родительский компонент, и **Owner** — владелец компонента. Родительский компонент — это тот, на котором располагается данный компонент. А владелец — это компонент, который передается в качестве параметра в конструктор данного компонента и который владеет им. Форма является владельцем всех расположенных на ней компонентов. В свою очередь объект приложения **Application** является владельцем всех форм.

Изменение во время выполнения свойства **Parent** заставляет компонент перемещаться на экране в клиентскую область нового родителя.

Пример использования свойства **Parent** приведен в разд. «Visible».

ParentCtl3D

Определяет наследование свойства объемного изображения от родительского компонента.

Класс *TWinControl*

Определение

```
__property bool ParentCtl3D
```

Описание

Если свойство **ParentCtl3D** имеет значение **true,** то компонент наследует свойство **Ctl3D.** управляющее объемным или плоским изображением, от своего родительского элемента. Это способствует единообразию изображений. Если все компоненты на форме имеют значение **ParentCtl3D,** равное **true,** то их вид определяется значением **Ctl3D** формы.

Непосредственное задание свойства **Ctl3D** какому-либо компоненту автоматически приводит к сбросу на **false** его свойства **ParentCtl3D.**

ParentFont

Включает и выключает использование шрифта родительского компонента

Класс *TControl*

Определение

```
__property bool ParentFont
```

Описание

Свойство **ParentFont** определяет, будет ли для данного компонента использоваться шрифт (свойство **Font**) родительского компонента-контейнера. Если установить во всех компонентах, размещенных на панели, **ParentFont** в **true**, то во всех компонентах будет использоваться одинаковый шрифт с одинаковым размером, цветом, стилем. Если все компоненты на форме имеют **ParentFont,** равным **true,** то во всех них атрибуты шрифта определяются свойством **Font** формы. Тогда, например, увеличение размера шрифта у формы приведет к согласованному изменению размеров шрифта всех размещенных на ней компонентов.

Если у формы свойство **ParentFont** тоже установлено в **true**, то шрифт определяется свойством **Font** объекта приложения **Application.**

При изменении свойства **Font** в каком-то компоненте, его свойство **ParentFont** автоматически сбрасывается в **false.**

ParentShowHint

Включает и выключает использование родительского свойства **ShowHint**.

Класс *TControl*

Определение

```
__property bool ParentShowHint
```

Описание

Свойство **ParentShowHint** используется, чтобы иметь возможность одновременно всем компонентам некоторого контейнера или формы разрешать или запрещать отображение ярлычков (всплывающих окон подсказки) при задержке на них курсора мыши. Отображаемый в окнах текст определяется свойствами **ShowHint** компонентов.

Если свойство **ParentShowHint** установлено в **true**, то разрешение или запрет отображения ярлычков определяется свойством **ShowHint** родительского компонента. Если же свойство **ParentShowHint** установлено в **false,** то отображение окон подсказки определяется свойством **ShowHint** самого компонента.

При задании в компоненте значения **ShowHint,** равного **true**, его свойство **ParentShowHint** автоматически сбрасывается в **false.**

Управлять отображением окон подсказок всего приложения в целом можно также свойством **ShowHint** объекта **Application.**

Подробные пояснения и примеры см. в разд. «Hint».

Pen

Определяет атрибуты пера, используемого для рисования линий и фигур.

Класс *TCanvas*

Определение

```
__property TPen* Pen
```

Описание

Свойство канвы Pen определяет атрибуты пера, используемого для рисования линий и фигур. Это свойство является объектом типа **TPen** (см. описание этого типа для получения дополнительной информации). Атрибуты объекта типа **TPen** определяют цвет, ширину, стиль линий и режим рисования пера.

Присваивание свойства Pen может производиться методом **Assign.**

PenPos

Определяет положение пера на канве

Класс *TCanvas*

Определение

```
__property Windows::TPoint PenPos
```

Описание

Свойство канвы **PenPos** определяет переменной типа **TPoint** положение пера на канве. Координаты пера, определенные этим свойством, задают начальную точку рисования линии методом **LineTo.**

Свойство **PenPos** изменяется методом **MoveTo** и некоторыми методами рисования (например, методом **LineTo**). Непосредственная установка **PenPos** эквивалентна применению метода **MoveTo.**

Pitch

Определяет способ установки ширины символов шрифта.

Класс *TFont*

Определение

```
enum TFontPitch { fpDefault, fpVariable, fpFixed };

__property TFontPitch Pitch
```

Описание

Каждый вид шрифта имеет соответствующий способ определения ширины символов. Есть шрифты с одинаковой шириной всех символов. Есть шрифты, в которых разные символы имеют разную ширину. Шрифты с постоянной шириной используются для отображения исходных кодов программ, поскольку в них удобно делать фиксированные отступы. Но шрифты с различной шириной символов выглядят естественнее и более компактны.

Возможные значения свойства **Pitch:**

Значение	Описание
fpDefault	Ширина устанавливается равной по умолчанию, т.е. описанной в шрифте заданного вида Name.
fpFixed	Установка одинаковой ширины всех символов.
fpVariable	Установка различной ширины символов.

Установка значений **fpVariable** или **fpFixed** заставляет Windows искать наилучший способ удовлетворить всем заданным характеристикам шрифта. Иногда это может привести к замене шрифта на шрифт другого, близкого вида. Но иногда может вообще не повлиять на шрифт. Все зависит от конкретного вида шрифта и даже от версии этого шрифта.

См. пример 4 в разд. **Font.**

Pixels

Определяет цвета пикселов канвы в пределах текущей области **ClipRect.**

Класс *TCanvas*

Определение

```
enum TColor {clMin=-0x7fffffff-1, clMax=0x7fffffff};

__property TColor Pixels[int X][int Y]
```

Описание

Свойство канвы **Pixels** определяет цвет пиксела канвы с координатами X и Y в пределах текущей области **ClipRect.** Если заданы координаты пиксела вне области **ClipRect.** то при чтении свойства **Pixels** возвращается значение -1.

Задание значений пикселов позволяет рисовать по пикселам графики и линии. Определение цвета пиксела используется обычно в методе **FillRect.**

Не все устройства поддерживают свойство **Pixels.** Чтение **Pixels** для таких устройств возвращает −1. Установка **Pixels** для подобных устройств не дает никаких результатов.

PopupMenu

Определяет всплывающее меню, связанное с данным компонентом

Класс *TFont*

Определение

```
__property Menus::TPopupMenu* PopupMenu
```

Описание

Задание свойства **PopupMenu** обеспечивает появление всплывающего меню, если в то время, когда выбран данный компонент, пользователь щелкнул правой кнопкой мыши. Обычно в этом всплывающем меню задаются основные команды, относящиеся к данному компоненту. Объект меню имеет тип **TPopupMenu.** Если в этом объекте свойство **AutoPopup** установлено в **true,** то меню будет появляться автоматически. В противном случае надо отображать меню с помощью метода **Popup** класса **TPopupMenu.**

Precision

Определяет число разрядов, характеризующих точность поля BCD.

Класс *TFieldDef*

Определение

```
__property int Precision
```

Описание

Свойство **Precision** определяет общее число разрядов, используемых для хранения значения поля BCD. Если **DataType не ftBCD,** то значение **Precision** игнорируется.

QuoteChar

Символ кавычек, используемый в свойстве **DelimitedText.**

См. раздел «CommaText, DelimitedText, Delimiter, QuoteChar».

Readonly — свойство TField

Указывает, является ли значение поля значением только для чтения.

Класс *TField*

Определение

```
__property bool Readonly
```

Описание

Задание **Readonly** равным **true** запрещает модификацию значения данного поля. В таблице данных при табуляции пользователя по полям с помощью клавиши Tab поле, помеченное как **Readonly,** пропускается.

Чтобы определить, можно ли модифицировать значение поля, лучше использовать свойство **CanModify.** которое учитывает не только установку **ReadOnly** данного поля, но и иные ограничения на работу с набором данных.

RecNo

Номер записи.

Класс *TBDEDataSet*

Определение

```
_ property int RecNo
```

Описание

Свойство **RecNo** определяет номер активной записи в наборе данных с учетом применяемой индексации набора. Совместно с числом записей **RecordCount** свойство **RecNo** может использоваться для организации циклов по записям. Особенно это удобно делать для таблиц Paradox, в которых установка **RecNo** автоматически позиционирует курсор на указанную запись (см. пример). Впрочем, удобнее для навигации использовать методы **First, Last. MoveBy. Next. Prior.**

Свойство IsSequenced показывает, можно ли применять в данном наборе свойство **RecNo.**

Пример

Следующий код отображает список сотрудников, хранящийся в таблице Paradox.

```
void _fastcall TForm1::Button1Click(TObject *Sender)
{
 String info = "Список сотрудников\n\n";
 for (int i=1; i <= Table1->RecordCount; i++)
 {
  Table1->RecNo = i;
  info += Table1->FieldByName("Fam")->AsString + '\n';
 }
 ShowMessage(info);
}
```

Required — свойство TField

Указывает, должно ли поле обязательно иметь значение, или оно может оставаться пустым.

Класс *TField*

Определение

```
_ property bool Required
```

Описание

Свойство **Required** определяет, должно ли поле обязательно иметь значение, или оно может оставаться пустым. Если объект поля создан Редактором Полей, то свойство **Required** устанавливается в соответствии с аналогичным свойством поля в таблице данных. Если свойство помечено в таблице как обязательное, то при попытке приложения переслать в него пустое значение генерируется исключение **EDatabaseError.**

В приложении можно задать **Required = true** и для поля, которое в таблице данных не помечено как обязательное. Но в этом случае надо написать соответствующий обработчик события **On Validate,** в котором генерировать **EDatabaseError** при попытке переслать в базу данных пустое значение.

Required — свойство TFieldDef

Свойство указывает, обязательно ли должно быть задано значение описываемого поля.

Класс *TFieldDef*

Определение

```
__property bool Required
```

Описание

Свойство **Required** указывает, обязательно ли должно быть задано значение описываемого поля. Если **Required = false,** то значение поля при заполнении записи может оставаться пустым. Если **Required = true,** то значение поля обязательно должно быть задано. Свойство **Required** задается при создании нового поля в его описании.

Пример

См. пример в гл. 4, в разд. «CreateTable».

SavePoint

Определяет точку текущего состояния клиентского набора данных.

Класс *TCustomClientDataSet*

Объявление

```
__property int SavePoint
```

Описание

Свойство **SavePoint** определяет точку текущего состояния результатов редактирования. Целое число, возвращаемое этим свойством, можно запомнить. Если после этого проведен ряд операций редактирования, то вы можете отменить их все, задав значение **SavePoint,** равное сохраненному.

Например, вы можете ввести в приложение глобальную переменную:

```
int MySavePoint;
```

и в событии формы **OnCreate** после открытия набора данных записать оператор:

```
MySavePoint = ClientDataSet1->SavePoint;
```

Тем самым вы запомнили начальное состояние. Если после работы с набором данных пользователь захочет отменить все результаты редактирования и вернуться к этому начальному состоянию, достаточно выполнить оператор:

```
ClientDataSetl->SavePoint = MySavePoint;
```

Если же пользователь хочет зафиксировать в наборе все имеющиеся на данный момент результаты редактирования, достаточно выполнить уже приведенный выше оператор

```
MySavePoint = ClientDataSet1->SavePoint;
```

Аналогичный оператор надо выполнить после того, как изменения перенесены в основную базу данных методом **ApplyUpdates.**

Возврат к прежнему состоянию невозможен, если уже произошла отмена изменений или установкой свойства (т.е. нельзя вернуться на 2 шага назад, можно помнить только одно предыдущее состояние), или методами **RevertRecord** и **CancelUpdates.**

SessionName — свойство TDatabase

Имя сеанса сетевого соединения, с которым связан компонент базы данных.

Класс *TDatabase*

Определение

```
__property System::AnsiString SessionName
```

Описание

Свойство **SessionName** задает имя сеанса сетевого соединения — компонента **Session,** с которым связан компонент базы данных **Database.** На это имя должны ссылаться использующие эту базу данных компоненты наборов данных (наследники **TDBDataSet)** в своих свойствах **SessionName.**

Если свойство **SessionName** в компоненте **Database** оставляется пустым, то компонент автоматически связывается с компонентом по умолчанию **Session.** Во время проектирования свойство **SessionName** компонентов **Database** устанавливается выбором из выпадающего списка в Инспекторе Объектов. В этом списке всегда имеется раздел **Default** — по умолчанию. Если в приложение явным образом введены компоненты **Session,** в которых заданы свойства **SessionName,** то в выпадающем списке в Инспекторе Объектов содержатся также имена **SessionName** этих компонентов **Session.**

SessionName — свойство TDBDataSet

Имя компонента **Session,** связанного с набором данных.

Класс *TDBDataSet*

Определение

```
__property System::AnsiString SessionName
```

Описание

Свойство **SessionName** задает имя набора данных компонента **Session.** Это имя автоматически задается равным свойству SessionName компонента базы данных **TDatabase,** с которым связан набор данных.

Если свойство **SessionName** в компоненте набора данных оставляется пустым, то набор автоматически связывается с компонентом по умолчанию Session. Во время проектирования свойство **SessionName** компонентов-наследников **TDBDataSet** устанавливается выбором из выпадающего списка в Инспекторе Объектов. В этом списке всегда имеется раздел **Default** — по умолчанию. Если в приложение явным образом введены компоненты **Session,** в которых заданы свойства **SessionName,** то в выпадающем списке в Инспекторе Объектов содержатся также имена **SessionName** этих компонентов Session. Значение **SessionName,** устанавливаемое в компоненте набора данных, должно совпадать со значением **SessionName,** установленным в наборе данных **Database.** с которым связан набор.

SessionName — свойство TSession

Имя сеанса сетевого соединения, которое присваивается компоненту **TSession** для ссылок на него в компонентах наборов данных и баз данных.

Класс *TSession*

Определение

```
__property AnsiString SessionName
```

Описание

Свойство **SessionName** задает имя сеанса сетевого соединения, на которое могут ссылаться компоненты наборов данных **TDBDataSet** и баз данных **TDatabase.** Имя, указанное в свойстве SessionName компонента **TSession,** появится в Инспекторе Объектов в выпадающих списках свойств **SessionName** компонентов **TDatabase** и наборов данных-наследников **TDBDataSet.**

Если свойство **AutoSessionName** установлено в **true,** приложение не может явным образом установить значение **SessionName.**

Shortcut

Определяет комбинацию «горячих» клавиш, обеспечивающих быстрый доступ к разделу меню.

Класс *TMenuItem*

Определение

```
typedef Word TShortCut;
__property TShortCut Shortcut
```

Описание

Задание свойства **Shortcut** позволяет пользователю не выбирать данный раздел из меню, а просто нажать заданную комбинацию «горячих» клавиш. Эта комбинация при установке **Shortcut** автоматически появляется в надписи раздела.

При задании свойства **Shortcut** во время проектирования Инспектор Объектов предлагает длинный список возможных комбинаций клавиш. При задании значения **Shortcut** во время выполнения можно использовать функции **ShortCut. TextToShortCut. ShortCutToText.**

ShowHint

Включает или отключает показ ярлычка (всплывающего окна подсказки) при задержке курсора мыши над компонентом.

Классы *TControl. TApplication*

Определение

```
__property bool ShowHint
```

Описание

Ярлычок (всплывающее окно подсказки) при задержке курсора мыши над компонентом появляется, если свойство компонента **ShowHint** (показать подсказку) установлено в **true** и задан текст подсказки в свойстве **Hint.** Правда, и при **ShowHint = false** всплывающее окно подсказки может появляться, если установлено в **true** свойство **ParentShowHint** (взять **ShowHint** родительского компонента), а в родительском компоненте **ShowHint = true.**

Изменение **ShowHint** автоматически ставит **ParentShowHint в false.**

Свойство **ShowHint** приложения **Application** (по умолчанию равно **true)** определяет, могут ли появляться окна подсказки в каких-то компонентах. Если установить **Application->ShowHint в false,** то окна подсказки не будут появляться независимо от значений **ShowHint в** любых компонентах.

См. подробные пояснения и примеры в разделе «Hint».

Showing

Определяет, виден ли компонент в данный момент.

Класс *TWinControl*

Доступ *только для чтения*

Определение

```
__property bool Showing
```

Описание

Свойство **Showing** показывает, может ли пользователь в данный момент видеть компонент. Правда, при этом не учитывается, что другие компоненты могут

загородить данный. Поэтому значение **Showing,** равное **true,** еще не гарантирует, что пользователь действительно видит компонент. Если компонент накрыт другим видимым компонентом, то пользователь его все-таки не увидит.

Если свойство **Visible** компонента и всех его родителей (компонентов, содержащих его) равно **true, то и Showing** равно **true.** Если же свойство **Visible** компонента или какого-то из его родителей равно **false, то Showing** равно **false.**

Size — свойство TField

Размер, определенный в описании поля базы данных, для типов полей, поддерживающих различные размеры.

Класс *TField*

Определение

```
__property int Size
```

Описание

Свойство **Size** указывает размер, определенный в описании поля базы данных, для типов полей, поддерживающих различные размеры. Интерпретация значения Size зависит от типа поля:

ftBoolean	Size не используется; значение Size = 0
ftSmallInt	Size не используется; значение Size = 0
ftWord	Size не используется; значение Size = 0
ftAutoInc	Size не используется; значение Size = 0
ftDate	Size не используется; значение Size = 0
ftInteger	Size не используется; значение Size = 0
ftTime	Size не используется; значение Size = 0
ftCurrency	Size не используется; значение Size = 0
ftDateTime	Size не используется; значение Size = 0
ftFloat	Size не используется; значение Size = 0
ftBCD	Size — число цифр после десятичной запятой
ftString	Size — максимальное число символов в строке
ftVarBytes	Size — максимальное число байтов, не считая двух байтов, указывающих истинное использованное число байтов
ftBytes	Size — максимальное число байтов
ftBlob	Size — число байтов из поля BLOB, которые хранятся в буфере записи
ftDBaseOle	Size — число байтов из поля DBase OLE BLOB, которые хранятся в буфере записи
ftFmtMemo	Size — число байтов из поля Memo, которые хранятся в буфере записи
ftGraphic	Size — число байтов из поля изображения, которые хранятся в буфере записи
ftMemo	Size — число символов из поля Memo, которые хранятся в буфере записи Size

ftParadoxOle	Size — число байтов из поля Paradox OLE BLOB, которые хранятся в буфере записи
ftTypedBinary	Size — число байтов из поля BLOB, которые хранятся в буфере записи
ftUnknown	Size не используется; значение Size = 0
ftCursor	Size не используется; значение Size = 0
ftADT	Size — общее число полей, содержащихся в поле ADT, включая дочерние поля любых дочерних полей типа.
ftReference	Size не используется; значение Size = 0
ftDataSet	Size не используется; значение Size = 0

Size — свойство TFieldDef

Свойство определяет размер поля.

Класс *TFieldDef*

Определение

```
__property int Size
```

Описание

Свойство **Size** определяет размер поля одного из следующих типов **TFieldType: ftString, ftBCD, ftBytes, ftVarBytes, ftBlob, ftMemo, ftGraphic.** При создании в таблице полей этих типов для них должно быть указано значение **Size.**

Для строк и полей байтов значение **Size** — это число байтов, отведенных под хранение значения данного поля. Для полей BCD значение **Size** — это число разрядов после запятой. Для полей BLOB, Memo и Graphic значение **Size** — это число первых байтов значения поля, которые хранятся непосредственно в самой таблице набора данных.

Пример

См. пример в гл. 4, в разд. «CreateTable».

Size — свойство TFont

Размер шрифта в кеглях (пунктах).

Класс *TFont*

Определение

```
__property int Size
```

Описание

Свойство **Size** определяет размер шрифта в кеглях (пунктах, принятых в Windows). Если значение **Size** задано отрицательным, то в размер входит верхний пиксел каждой строки. Если значение **Size** задано положительным, то этот пиксел не учитывается.

Для задания размера шрифта может использоваться другое свойство: **Height** — размер шрифта в пикселах. Значение **Size** связано со свойствами **Height и PixelsPerInch** (число пикселов на дюйм — см. **TFont)** соотношением:

```
Font->Size = -Font->Height * 12 I Font->PixelsPerInch
```

Из соотношения, в частности, видно, что задание положительного значения **Size** ведет к отрицательному значению **Height.** Можно задавать **Size** отрицательным; тогда **Height** будет положительным.

State

Свойство определяет режим, в котором находится набор данных.

Класс *TDataSet*

Доступ *только для чтения*

Определение

```
enum TDataSetState {dsInactive, dsBrowse, dsEdit, dsInsert,
                    dsSetKey, dsCalcFields, dsFilter,
                    dsNewValue, dsOldValue, dsCurValue,
                    dsBlockRead, dsInternalCalc, dsOpening};

__property TDataSetState State
```

Описание

Свойство **State** определяет состояние набора данных. Это свойство доступно только во время выполнения и только для чтения. Набор данных может находиться в одном из следующих состояний:

dslnactive	Набор данных закрыт, данные недоступны
dsBrowse	Данные могут наблюдаться, но не могут изменяться. Это состояние по умолчанию после открытия набора данных
dsEdit	Текущая запись может редактироваться
dslnsert	Может вставляться новая запись
dsSetKey	Доступен режим поиска записи и операция задания диапазона изменений SetRange. Может наблюдаться только ограниченное множество данных и не может проводиться редактирование или вставка новой записи
dsCalcFields	Происходит обработка события OnCalcFields. Не может проводиться редактирование невычисляемых полей или вставка новой записи
dsFilter	Происходит обработка события **OnFilter Record.** Может наблюдаться только ограниченное множество данных и не может проводиться редактирование или вставка новой записи
dsNewValue	Внутреннее временное состояние, показывающее, что доступно свойство поля TField.NewValue
dsOldValue	Внутреннее временное состояние, показывающее, что доступно свойство поля TField.OldValue
dsCurValue	Внутреннее временное состояние, показывающее, что доступно свойство поля **TField.CurValue**
dsBlockRead	Компоненты, связанные с данными, еще не изменили значения и еще не генерировались события, связанные с перемещением курсора по таблице
dsInternalCalc	Временное состояние, используемое внутри компонента
dsOpening	Набор данных открывается, но этот процесс еще не закончен

Шесть последних состояний чисто служебные и для пользователя интереса не представляют. Они устанавливаются автоматически, когда приложение обращается к соответствующему свойству **TField,** и наблюдаться из приложения не могут. Состояния **dsCalcFields и dsFilter** также устанавливаются автоматически при воз-

никновении событий **OnCalcField** и **OnFilterRecord** и наблюдаться из приложения не могут. А остальные состояния могут устанавливаться в приложении во время выполнения, но не непосредственно, а с использованием различных методов.

Метод **Close** закрывает соединение с базой данных, устанавливая свойство **Active** набора данных в **false.** При этом **State** переводится в состояние **dsInactive.**

Метод **Open** открывает соединение с базой данных, устанавливая свойство **Active** набора данных в **true.** При этом State переводится в состояние **dsBrowse.**

Метод **Edit** переводит набор данных в состояние **dsEdit.**

Методы **Insert и InsertRecord** вставляют новую пустую запись в набор данных, выполняют еще ряд операций, о которых вы можете посмотреть в справке C++Builder, и переводят **State** в состояние **dsInsert.**

Методы **EditKey, SetRange, SetRangeStart, SetRangeEnd и ApplyRange,** связанные с поиском записи и с заданием допустимого диапазона изменения данных, переводят **State** в состояние **dsSetKey.**

Многие другие методы также устанавливают автоматически различные состояния набора данных. При программировании работы с базой данных надо следить за тем, чтобы набор данных вовремя был переведен в соответствующее состояние. Например, если вы стали редактировать запись, не переведя предварительно набор данных в состояние **dsEdit** методом **Edit,** то будет сгенерировано исключение **EDatabaseError** с сообщением «Dataset not in edit or insert mode» — «Набор данных не в режиме Edit или Insert».

Style — свойство TBrush

Определяет шаблон заполнения кисти **Brush.**

Класс *TBrush*

Определение

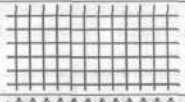
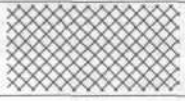
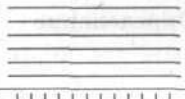
```
enum TBrushStyle {bsSolid, bsClear, bsHorizontal, bsVertical,
                  bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross};

__property TBrushStyle Style
```

Описание

Свойство кисти **Style** определяет шаблон, которым рисует кисть **Brush.** если для нее не задано значение свойства **Bitmap.**

Возможные значения **Style:**

Значение	Шаблон	Значение	Шаблон
bsSolid		**bsCross**	
bsClear		**bsDiagCross**	
bsBDiagonal		**bsHorizontal**	
bsFDiagonal		**bsVertical**	

Пример

Код

```
Image1->Canvas->Brush->Color = clRed;
Image1->Canvas->Brush->Style = bsDiagCross;
Image1->Canvas->Ellipse(0, 0, Image1->Width, Image1->Height);
```

строит на канве компонента **Image1** эллипс, заполненный красной штриховкой крест на крест.

Style — свойство TFont

Стиль шрифта.

Класс *TFont*

Определение

```
enum TFontStyle { fsBold, fsItalic, fsUnderline, fsStrikeOut };
typedef Set<TFontStyle, fsBold, fsStrikeOut> TFontStyles;

__property TFontStyles Style
```

Описание

Свойство **Style** задает стиль: характер начертания символов заданного шрифта. Свойство представляет собой множество или пустое, или содержащее одно и более следующих значений:

Значение	Описание
fsBold	Полужирный
fsItalic	Курсив
fsUnderline	Подчеркнутый
fsStrikeout	Перечеркнутый горизонтальной прямой

Примеры

```
// Обычный стиль
Label1->Font->Style = TFontStyles();
// Полужирный
Label1->Font->Style = TFontStyles()<< fsBold;
// Полужирный курсив
Label1->Font->Style = TFontStyles()<< fsBold << fsItalic;
// Задание курсива без изменения других установок
Label1->Font->Style = Label1->Font->Style << fsItalic;
```

Style — свойство TPen

Определяет стиль рисования линий пером.

Класс *TPen*

Определение

```
enum TPenStyle {psSolid, psDash, psDot, psDashDot,
                psDashDotDot, psClear, psInsideFrame};
__property TPenStyle Style
```

Описание

Свойство пера **Style** определяет вид линии. Это свойство может принимать следующие значения:

psSolid	Сплошная линия
psDash	Штриховая линия
psDot	Пунктирная линия
psDashDot	Штрих-пунктирная линия
psDashDotDot	Линия, чередующая штрих и два пунктира
psClear	Отсутствие линии
psInsideFrame	Сплошная линия, но при Width > 1 допускающая цвета, отличные от палитры Windows

Примеры линий всех стилей даны на рис. 3.1.

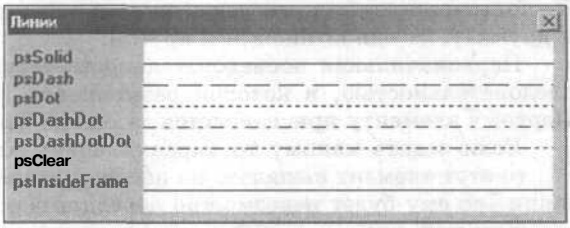


Рис. 3.1
Примеры линий различных стилей

Все стили со штрихами и пунктирами доступны только при **Width** = 1. В противном случае линии этих стилей рисуются как сплошные.

Стиль **psInsideFrame** — единственный, который допускает произвольные цвета. Цвет линии при остальных стилях округляется до ближайшего из палитры Windows.

TableName

Свойство, определяющее имя таблицы базы данных.

Класс *TTable*

Определение

```
__property AnsiString TableName
```

Описание

Свойство **TableName** определяет имя таблицы базы данных, которое предварительно должно быть задано свойством **DatabaseName** набора данных. Во время проектирования **TableName** в компонентах наборов данных **(Table. Query** и др.) устанавливается в окне Инспектора Объектов выбором из выпадающего списка. Во время выполнения свойство **TableName** может устанавливаться программно.

При задании или изменении **TableName** компонент набора данных должен находиться в неактивном состоянии (свойство **Active** = **false).** В противном случае будет сгенерировано исключение.

Пример

Следующий код завершает текущее соединение компонента **Table1,** задает новую базу данных и таблицу, беря значения из окон редактирования **Edit1** и **Edit2** и соединяет компонент **Tablel** с новой таблицей новой базы данных.

```
Tablel->Active = false;
Table1->DatabaseName = Edit1->Text;
Table1->TableName = Edit2->Text;
Tablel->Active = true;
```

См. также примеры в описании свойства **Active.**

TabOrder

Определяет позицию компонента в последовательности табуляции.

Класс *TWinControl*

Определение

```
typedef short TTabOrder;
__property TTabOrder TabOrder
```

Описание

Под последовательностью табуляции понимается последовательность, в которой переключается фокус между компонентами окна, когда пользователь последовательно нажимает клавишу табуляции Tab. Значение **TabOrder,** равное нулю, означает, что при первом появлении формы на экране в фокусе будет этот компонент.

Последовательность табуляции в каждом приложении надо продумывать, чтобы пользователю было легче работать и переходить от одного окна редактирования к другому, от одной кнопки к другой.

Первоначальная последовательность табуляции определяется просто той последовательностью, в которой размещались управляющие элементы на экране. Первому элементу присваивается значение **TabOrder,** равное 0, второму 1 и т.д.

Если задать какому то управляющему элементу значение **TabOrder,** равное -1, то этот элемент выпадает из последовательности табуляции и с помощью клавиши Tab ему будет невозможно передать фокус.

Каждый управляющий элемент имеет уникальный номер **TabOrder** внутри своего родительского компонента. Поэтому изменение значения **TabOrder** какого-то элемента на уже существующее у другого элемента значение приведет к тому, что значения **TabOrder** всех последующих элементов автоматически изменятся, чтобы не допустить дублирования. Если задать элементу значение **TabOrder,** большее, чем число элементов в родительском компоненте, он просто станет последним в последовательности табуляции.

В среде проектирования C++Builder имеется специальная команда Edit | Tab Order, позволяющая в режиме диалога задать последовательность табуляции всех элементов.

Значение свойства **TabOrder** играет роль только тогда, когда свойство компонента **TabStop** установлено в **True** и компонент имеет родителя. Например, для формы свойство **TabOrder** имеет смысл только в случае, если для формы задан родитель в виде другой формы.

TabStop

Определяет возможность передать фокус на элемент нажатием клавиши табуляции.

Класс *TWinControl*

Определение

```
__property bool TabStop
```

Описание

Свойство **TabStop** разрешает или запрещает останавливать фокус на данном элементе при нажатии пользователем клавиши Tab. Если значение **TabStop** равно **true,** то клавиша Tab будет передавать фокус на этот элемент в последовательности табуляции, определяемой значениями свойства элементов **TabOrder.** Если значение **TabStop** равно **false,** то элемент недостижим в последовательности табуляции независимо от значения **TabOrder.**

Для формы свойство **TabStop** имеет смысл только в случае, если форма имеет родителя в виде другой формы.

Tag

Свойство, используемое пользователем по своему усмотрению.

Класс *TComponent*

Определение

```
__property int Tag
```

Описание

Свойство Tag системой не используется. Его значение по умолчанию равно 0. Пользователь может определить и использовать Tag по своему усмотрению, помещая в него необходимую информацию. Например, можно в процессе проектирования или программно в процессе выполнения задать некоторое значение Tag группе компонентов и затем оперировать с этой группой.

Пример использования свойства Tag см. в разделе «Components».

Text — свойство TComponent

Текстовая строка, связанная с управляющим элементом.

Класс *TComponent*

Определение

```
__property AnsiString Text
```

Описание

Свойство **Text** позволяет прочесть или задать строку, связанную с данным управляющим элементом. По умолчанию значение **Text** равно имени компонента — его свойству **Name.** Применяется в основном в компонентах редактирования и в списках.

Text — свойство TField

См. раздел «DisplayText и Text».

TextFlags

Определяет способ вывода текста на канву.

Класс *TCanvas*

Определение

```
__property int TextFlags
```

Описание

Свойство канвы **TextFlags** определяет особенности вывода текста на канву методами **TextOut и TextRect.** Свойство **TextFlags** может формироваться как комбинация следующих констант:

Константа	Пояснение
ETO_CLIPPED	Выводится только текст, помещающийся в указанной прямоугольной области. В методе TextRect этот флаг устанавливается автоматически. На метод TextOut этот флаг не влияет.
ETO_OPAQUE	Текст выводится с непрозрачным цветом фона.
ETO_RTLREADING	Строка текста выводится справа налево. Доступно только с версией Windows Mideast (для стран Востока).

Константа	Пояснение
ETO_GLYPH_INDEX	Текст является массивом кодов символов, который непосредственно передается GDI Windows. Применимо только для шрифтов TrueType, но этот флаг можно применять и для других шрифтов, чтобы указать, что GDI должен обрабатывать текст напрямую, без языковой обработки. Подробнее см. в документации по GDI Windows.
ETO_IGNORELANGUAGE	Недокументированный пока флаг Microsoft.
ETO_ NUMERICSLOCAL	Недокументированный пока флаг Microsoft.
ETO_NUMERICSLATIN	Недокументированный пока флаг Microsoft.

Top

Координата верхнего края компонента в пикселах.

Класс *TControl*

Определение

```
__property int Top
```

Описание

Свойство Top определяет координату верхнего края компонента в пикселах. Для компонентов за начало отсчета берется верхняя граница клиентской области родителя (например, панели, если данный компонент расположен на панели, или формы, если компонент расположен непосредственно на форме). Отсчет координаты ведется сверху вниз. Для формы координата Top представляет собой вертикальную координату экрана, отсчитываемую от его верхнего края.

Свойство Top используется при перемещениях и изменениях размеров компонентов (см. примеры в разделах BoundsRect» и «Components»).

TransparentColor и TransparentMode

Определяет, какой цвет в битовой матрице будет прозрачным при ее рисовании.

Класс *TCanvas*

Определение

```
__property TColor TransparentColor

enum TTransparentMode {tmAuto, tmFixed};
__property TransparentMode
```

Описание

Значение свойства канвы **TransparentColor** имеет тип **TColor** и зависит от установки свойства **TransparentMode.** Если **TransparentMode** установлено в **tmAuto, то TransparentColor** возвращает цвет пиксела левого нижнего угла изображения. Если вы задаете значение свойства **TransparentColor, то TransparentMode** автоматически устанавливается в tmFixed. При этом новый цвет сохраняется вместе с объектом битовой матрицы и может быть использован позднее. Если вы хотите отменить заданное значение **TransparentColor,** установите **TransparentMode в tmAuto,** и тогда **TransparentColor** опять будет указывать на цвет левого нижнего пиксела.

TransparentMode

См. раздел «TransparentColor и TransparentMode».

UpdateObject

Определяет объект, используемый для обновления кэшируемых результатов «только для чтения».

Класс *TBDEDataSet*

Определение

```
__property TDataSetUpdateObject* UpdateObject
```

Описание

Объект **TDataSetUpdateObject,** определяемый свойством **UpdateObject,** используется для обновления результатов, возвращаемых Borland Database Engine (BDE) в режиме «только для чтения». BDE всегда пытается, если возможно, вернуть «живые» данные, которые можно редактировать. Но иногда это невозможно, если, например, делается запрос сразу к нескольким таблицам. В этих случаях требуется компонент **UpdateSQL,** который может справляться с обновлением подобных наборов данных. Этот объект и должен быть указан в свойстве **UpdateObject.**

UpdateRecordTypes

Определяет, какие записи должны быть видимы в наборе данных при кэшировании.

Класс *TBDEDataSet*

Определение

```
enum DBTables__9 {rtModified, rtInserted, rtDeleted,
                  rtUnmodified};
typedef Set<DBTables__9, rtModified, rtUnmodified>
TUpdateRecordTypes;

__property TUpdateRecordTypes UpdateRecordTypes
```

Описание

Свойство **UpdateRecordTypes** определяет, какие записи должны быть видимы в наборе данных при кэшировании. Свойство представляет собой множество следующих возможных значений:

rtModified	видимы модифицированные записи
rtInserted	видимы вставленные записи
rtDeleted	видимы удаленные записи.
rtUnmodified	видимы немодифицированные записи

По умолчанию **UpdateRecordTypes = [rtModified, rtInserted, rtUnmodified].** Если, например, добавить в **UpdateRecordTypes** значение **rtDeleted** и удалить все прочие значения, то пользователь увидит все удаленные записи и может решать, следует ли их действительно удалять при занесении результатов в базу данных. Аналогично можно предоставить пользователю возможность видеть, например, только все вставленные или все отредактированные записи.

UpdatesPending

Определяет, имеются ли в кэше обновленные записи, не отправленные в базу данных.

Класс *TBDEDataSet*

Определение

```
__property bool UpdatesPending
```

Описание

Свойство **UpdatesPending** определяет, имеются ли в кэше обновленные записи, не отправленные в базу данных. Если **UpdatesPending = true,** значит имеются отредактированные, удаленные или вставленные записи, причем эти изменения набора данных еще не перенесены в базу данных. В этом случае можно осуществить перенос исправлений в базу данных или спросить пользователя о необходимости такого переноса.

Пример

См. пример в гл. 5, в разд. «AfterClose и BeforeClose».

ValidChars

Определяет символы, которые могут использоваться при редактировании значения поля.

Класс *TField*

Определение

```
typedef Set<char, 0, 255> TFieldChars;
__property TFieldChars ValidChars
```

Описание

Свойство **ValidChars** задается, чтобы ограничить набор символов, используемых для редактирования значения поля. Значение **ValidChars** используется методом **IsValidChar** для проверки допустимости вводимых пользователем символов. В отличие от свойства **EditMask**, определяющего допустимые символы для каждой позиции, свойство **ValidChars** определяет символы, допустимые в любой позиции.

Значение **ValidChars** устанавливается по умолчанию автоматически в зависимости от типа поля (см. значения по умолчанию в разд. «IsValidChar» в гл. 4). Например, для строк допускаются любые символы, для целых типов — только цифры. Но вы можете заменить или дополнить эту установку по умолчанию.

Примеры

Пусть вы хотите, чтобы пользователь мог занести в объект поля **Fieldl** только цифры от 1 до 3 (может быть, это номер отдела или номер ответа на какой-то вопрос). Тогда задайте значение **ValidChars** этого поля равным **['1'..'3']**. Соответствующий оператор можно поместить, например, в обработчике события **AfterOpen** набора данных:

```
Fieldl->ValidChars = Field1->ValidChars.Clear() << '1' << '2' << '3';
```

Другой пример. Пусть вы имеете объект поля **TablelFam,** в значение которого пользователь должен занести фамилию. Тогда целесообразно запретить ввод в это поле цифр, знаков препинания и латинских символов. Для этого достаточно задать значение **ValidChars,** равное **['а'..'я', 'А'..'Я', ' ', '–']**. В этом множестве символы пробела и " " добавлены для возможности задания составных фамилий. Как и в предыдущем примере, задание этого значения **ValidChars** можно осуществить в обработчике события **AfterOpen.**

И еще один пример. Пусть вы задали необходимые значения **ValidChars** и имеете в приложении окно редактирования **Edit1**, в которое пользователь должен вводить данные, передаваемые в дальнейшем в объект поля **Fieldl.** Тогда целесообразно запретить пользователю вводить в окно символы, не входящие в множество допустимых для данного поля. Это можно сделать, написав следующий обработчик события **OnKeyPress** окна редактирования **Editl:**

```
void__fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
 (
 if ( ! Field1->IsValidChar(Key))
   Key = '\0';
}
```

В этом обработчике все недопустимые символы подменяются нулевым и не вводятся в окно.

Value, NewValue, OldValue

Текущее, новое и прежнее значения поля.

Класс *TField*

Определения

```
__property System::Variant Value
__property System::Variant NewValue
__property System::Variant OldValue // только для чтения
```

Описание

Свойства **Value, NewValue и OldValue,** а также свойство **CurValue** определяют значение объекта поля. Свойство **OldValue** соответствует тому начальному значению поля, которое было до какого-либо редактирования и занесения новых значений в буфер записи. **Value** — это текущее значение поля. **NewValue** — это значение, которое задается полю в случае возникновения проблем при пересылке записи методом **Post** в базу данных.

После чтения записи из базы данных **Value и OldValue** совпадают друг с другом. В дальнейшем значение **OldValue** остается неизменным, а значение **Value** может изменяться в процессе редактирования записи или программно.

При пересылке записи в базу данных значение **NewValue** является пересылаемым значением. Сначала оно совпадает с **Value,** Но при пересылке могут возникнуть проблемы. В этом случае в клиентском наборе данных генерируется событие **OnReconcileError.** На стороне сервера компонент-провайдер генерирует событие **OnUpdateError.** В обработчике этого события можно сравнить **NewValue** — новое значение, которое не удалось переслать, **OldValue** — прежнее значение до начала редактирования данного поля, и **CurValue** — текущее значение поля с учетом изменений, внесенных другими пользователями в базу данных. Значение **CurValue** может отличаться от **OldValue,** если другой пользователь изменил значение поля за время, которое прошло после чтения **OldValue.** В обработчиках этих событий можно задать новое значение **NewValue,** которое бы ликвидировало возникшие проблемы. Значение **NewValue** можно также задать отличающимся от **Value** в обработчике события **OnUpdateRecord.**

Свойства **NewValue и OldValue** можно использовать только при применении компонента **TClientDataSet** или при кэшировании.

Visible — свойство TControl

Определяет, видим или невидим компонент.

Класс *TControl*

Определение

```
__property bool Visible
```

Описание

Свойство **Visible** определяет видимость компонента во время выполнения. Если **Visible** делается равным **true,** то компонент становится видимым; если **Visible** делается равным **false,** то компонент становится невидимым, исчезает для пользователя. Если устанавливается в **false** свойство **Visible** компонента-контей-

нера, то становятся невидимыми и все расположенные на нем дочерние компоненты, независимо от значения их свойств **Visible.** Если свойство **Visible** ранее невидимого компонента-контейнера устанавливается в **true,** то становятся видимыми и все его дочерние компоненты, у которых **Visible = true.**

Свойство **Visible** позволяет проектировать на одном и том же месте формы несколько панелей, соответствующих различным режимам работы приложения, и в нужные моменты делать одну из них видимой, а остальные невидимыми, как в приведенном ниже примере.

Свойство **Visible** может также активно использоваться для разделов меню. Очевидно, что обычно не все разделы меню имеют смысл при любых режимах работы приложения. Ненужные разделы можно делать недоступными задавая значения **false** их свойствам **Enabled.** В этом случае они будут видны серыми и недоступными, но размер меню не изменится. А если их делать невидимыми, то они видны не будут, оставшиеся разделы меню сомкнутся и все будет выглядеть более компактно.

Прямое задание значений **true и false** свойству **Visible** можно заменить вызовами методов **Show и Hide.** Первый из них делает компонент видимым и устанавливает **Visible в true.** А второй делает компонент невидимым и устанавливает **Visible в false.**

Пример

Пусть в приложении в одном и том же месте формы друг на друге расположены две панели: **Panell и на** ней **Panel2,** содержащие какие-то управляющие компоненты для разных режимов работы. **Pane12** расположена на **Panell,** которая является, таким образом, ее родителем. В обработчик события формы **OnCreate** можно вставить операторы:

```
Panel2->Visible = false;
Panel1->Visible = true;
Panel2->Parent = Forml;
Panel2->BoundsRect = Panel1->BoundsRect;
```

Первый два из них делают панель **Pane12** невидимой, а **Panell** — видимой. Впрочем можно было бы обойтись и без этих операторов, если задать в процессе проектирования значения **Visible,** равными **true** для **Panell и false** для **Pane12.** Третий оператор делает родительским компонентом панели **Pane12** форму **Forml.** А четвертый оператор задает панели Pane12 то же местоположение и размеры, которые имеет панель **Panell.** Последнее необходимо, поскольку при проектировании ее координаты соответствовали координатному пространству контейнера — клиентской области панели **Panell.** А теперь ее родитель сменился на форму, и надо ее расположить в том же месте формы, в котором расположена **Panell.**

Приведенный код можно было бы сократить, если в процессе проектирования размещать панель **PaneI2** не на панели **Panell,** а в каком-то другом месте непосредственно на форме, и задать значения **Visible,** равными **true** для **Panell и false** для **Pane12.** Тогда в обработчике события формы **OnCreate** достаточно одного оператора:

```
Panel2->BoundsRect = Panel1->BoundsRect;
```

изменяющего положение **Panel**2.

Аналогичный оператор может быть также реализован методом **SetBounds:**

```
Panel2->SetBounds(Panel1->Left, Panell->Top,
                  Panel1->Width, Panel1->Height);
```

В результате работы одного из приведенных операторов в момент создания формы на ней будет видна панель **Panell.** В момент, когда ее надо заменить на **PaneI2,** можно выполнить операторы:

```
Panell->Visible = false;
Panel2->Visible = true;
```

делающие невидимой первую и видимой вторую панель. Когда надо вернуть на экран изображение **Panell,** можно выполнить операторы:

```
Panel2->Visible = false;
Panell->Visible = true;
```

Другой способ решения той же задачи приведен в гл. 4, в разд. «BringToFront».

Visible — свойство TField

Определяет видимость поля в таблицах данных.

Класс *TField*

Определение

```
__property bool Visible
```

Описание

Свойство Visible определяет видимость поля в таблицах данных **DBGrid.** Если **Visible = false,** то соответствующее поле в таблице не видно. Свойство **Visible** игнорируется, если в **DBGrid** явно задано значение Columns для данного поля, или если для отображения используется компонент, отличный от **DBGrid.**

Width

Определяет горизонтальный размер компонента или формы в пикселах.

Класс *TControl*

Определение

```
__property int Width
```

Описание

Свойство **Width** определяет горизонтальный размер компонента или формы в пикселах. Используется для изменения ширины компонента при изменениях размеров окна приложения. На компоненты-таблицы во время выполнения изменение **Width** не действует. См. разд. «Client Width» и «ClientRect».

WindowText

Содержит строку текста, связанного с компонентом.

Класс *TControl*

Определение

```
__property char* WindowText
```

Описание

Свойство **WindowText** используется, чтобы связать с компонентом некоторую строку текста, которая может заменяться во время выполнения. По умолчанию **WindowText** — та же самая строка, которая записана в свойстве **Text.** Однако в классах, производных от **TControl.** это может быть изменено. Для окон редактирования эта строка соответствует отображаемому в компоненте тексту. Для выпадающих списков это текст в окошке редактирования. Для кнопок это имя кнопки. Для остальных компонентов это строка заголовка окна.

XMLData

См. разд. «Data, XMLData — свойства TCustomClientDataSet».

Глава 4

Методы компонентов и классов C++Builder

В этой главе приведены развернутые описания около 200 методов компонентов и классов C++Builder. Для многих методов даются примеры их применения. Конечно, это далеко не все методы. Для многих других методов вы можете найти краткие описания в главах 1 и 2.

В описаниях вы можете встретить идентификаторы, выделенные подчеркиванием. Например, **Post.** Это означает, что в этой или других главах в соответствующем разделе вы сможете найти развернутое пояснение, комментарий или примеры, связанные с данным термином.

Значительно большее количество описаний различных методов вы можете найти в источнике [3].

Навигация по наборам данных

Методы перемещают курсор набора данных.

Классы *TDataSet*

Определения

```
void__fastcall  First(void);
void__fastcall  Last(void);
void__fastcall  Next(void);
void__fastcall  Prior(void);
int__fastcall MoveBy(int Distance);
bool__fastcall  FindFirst(void);
bool__fastcall FindLast(void);
bool__fastcall  FindNext(void);
bool__fastcall  FindPrior(void);
```

Описание

Методы навигации перемещают курсор набора данных на другую запись и делают эту запись активной. Перемещение происходит с учетом условий фильтрации и индексации. Прежде, чем переместить курсор, любой метод навигации выполняет метод **Post** для текущей записи, фиксируя тем самым в базе данных результаты ее редактирования. Буфер записи очищается.

Процедуры **First, Last,** Next, **Prior** перемещают курсор набора данных соответственно к первой, последней, следующей и предыдущей записям. Процедура **First** устанавливает в **true** свойство Bof — первая запись. Процедура **Prior** тоже устанавливает **Bof** в **true,** если текущая запись уже была первой. Процедура **Last** устанавливает в **true** свойство Eof — последняя запись. Процедура Next тоже устанавливает **Eof** в **true,** если текущая запись уже была последней.

Функция **MoveBy** перемещает курсор на **Distance** записей и возвращает число записей, на которые переместился курсор. При положительном значении **Distance** перемещение происходит по направлению к последней записи, при отрицательном — по направлению к первой записи. Если значение **Distance** требует выхода за пределы набора данных, то курсор останавливается на крайней (последней или первой) записи и свойство Eof или **Bof** устанавливается в **true.** При этом возвращенное функцией значение окажется меньше, чем модуль **Distance.**

Функции **FindFirst, FindLast, FindNext, FindPrior** перемещают курсор так же, как процедуры **First, Last, Next, Prior.** При этом они возвращают **true** при успешном перемещении. Свойства Eof и Bof не устанавливаются.

Примеры

Следующий пример показывает типичный способ организации цикла по записям набора данных. Пусть в вашем приложении имеется выпадающий список с именем CBdep, который вы хотите заполнить данными, содержащимися в полях Dep всех записей таблицы, соединенной с компонентом **Table1**. Это можно сделать следующим кодом:

```
CBdep->Clear();
Table1->First();
while (! Table1->Eof)
{
 CBdep->Items->Add(Table1->FieldByName("Dep")->AsString);
 Table1->Next();
}
```

Первый оператор кода очищает список **CBdep.** Второй — устанавливает курсор таблицы на первую запись. Далее следует цикл по всем записям, пока не достигнута последняя, что проверяется выражением **Table1->Eof.** Для каждой записи в список заносится значение поля Dep, после чего методом Next курсор перемещается к следующей записи.

Вместо метода **FieldByName** можно применить метод **FindField:**

```
CBdep->Items->Add(Table1->FindField("Dep")->AsString);
```

Отличие проявится только в случае, если поля Dep в наборе данных не окажется. Тогда метод **FieldByName** сгенерирует исключение, а метод **FindField** вернет **NULL.**

Ниже приведен аналогичный пример, но просматривающий записи в направлении от последней к первой.

```
CBdep->Clear();
Table1->Last();
while (! Table1->Bof)
{
 CBdep->Items->Add(Table1->FieldByName("Dep")->AsString);
 Table1->Prior();
};
```

В приведенных примерах вместо методов First, Next, **Last и Prior** можно использовать методы **FindFirst, FindNext, FindLast и FindPrior,** осуществляющие навигацию и возвращающие **true** при ее успешном завершении. Например:

```
CBdep->Clear();
Table1->FindFirst();
do
 CBdep->Items->Add(Table1->FindField("Dep")->AsString);
while (Table1->FindNext());
```

Add — метод TCollection

Создает новый объект **TCollectionItem** и добавляет его в массив **Items.**

Класс *TCollection*

Определение

```
TCollectionItem*__fastcall Add(void);
```

{

Описание

Метод **Add** создает новый объект **TCollectionItem** и добавляет его в конец массива **Items.** Возвращает созданный объект.

Пример

См. пример в разделе «BeginUpdate».

Add — метод TFieldDefs

Создает новый объект описания поля и добавляет его в свойство **Items** объекта **TFieldDefs**

Класс *TFieldDefs*

Определение

```
HIDESBASE void __fastcall Add(const AnsiString Name,
                              TFieldType DataType,
                              int Size, bool Required);
```

Описание

Метод **Add** оставлен только для обратной совместимости. В новых приложениях следует использовать метод **AddFieldDef.**

Метод создает новый объект описания поля и добавляет его в свойство Items объекта **TFieldDefs.** Параметры **Name, DataType, Size, Required** передаются в значения соответствующих свойств объекта описания поля. Если описание поля с тем же именем уже существует, то генерируется исключение **EDatabaseError.**

Пример

См. пример в разделе «CreateTable».

Add — метод списков

Функция добавляет новый элемент в список.

Классы *TList, TStringList, TStrings*

Определения

Для **TList:**

```
int __fastcall Add(void * Item);
```

для **TStrings и TStringList:**

```
virtual int __fastcall Add(const System::AnsiString S);
```

Описание

Функция **Add** добавляет новый элемент в список. Если список не сортированный, то элемент добавляется в конец списка. Если же список сортированный, то новый элемент добавляется в позицию, которая определяется сортировкой. Функция возвращает индекс добавленного элемента (индекс первого элемента — 0). Увеличивает значение свойства **Count** на 1. Если значение **Count** равно значению **Capacity** (емкости массива), то увеличивается значение **Capacity** (с запасом) и перераспределяется память под новые элементы.

Для сортированного списка **TStringList** при выполнении **Add** генерируется исключение **EListError,** если строка S уже имеется в списке и свойство **Duplicates** установлено в **dupError.**

Примеры

См. в гл. 1 примеры в разд. «TList — класс» и «TStrings — класс».

AddFieldDef — метод TFieldDefs

Создает новый объект описания поля и добавляет его в свойство Items объекта TFieldDefs.

Класс *TFieldDefs*

Определение

```
TFieldDef*__fastcall AddFieldDef(void);
```

Описание

Метод **AddFieldDef** создает новый объект описания поля типа **TFieldDef** и добавляет его в свойство **Items** объекта **TFieldDefs.** Возвращается созданный объект описания. После вызова **AddFieldDef** можно задать свойства созданного описания **Name, DataType, Size, Required.**

Метод **AddFieldDef** применяется только для последующего создания новой таблицы методами CreateTable или **CreateDataSet.** Для добавления поля в уже существующую таблицу метод использовать нельзя.

Пример

См. пример в разделе «CreateTable».

AddIndex — метод TCustomClientDataSet

Создает и добавляет новый индекс в клиентский набор данных.

Класс *TCustomClientDataSet*

Объявление

```
void__fastcall AddIndex(const AnsiString Name,
                        const AnsiString Fields,
                        Db::TIndexOptions Options,
                        const AnsiString DescFields = "",
                        const AnsiString CaseInsFields = "",
                        const int GroupingLevel = 0);
```

Описание

Метод **AddIndex** создает и добавляет в клиентский набор данных новый индекс. Параметр **Name** задает имя создаваемого индекса. Параметр **Fields** задает список полей, по которым проводится индексация. Имена полей отделяются друг от друга точками с запятой. Параметр **Options** задает множество опций сортировки, которое может включать значения **ixCaseInsensitive** — нечувствительность к регистру при сортировке строковых полей, и ixDescending — сортировка в порядке убывания значений полей. По умолчанию обе опции отсутствуют.

Параметры **DescFields и CaseInsFields** дают альтернативный вариант задания опций сортировки. Параметр **DescFields** может содержать строку, в которой перечисляются через точку с запятой те поля, по которым сортировка должна проводится в порядке убывания значений. А параметр **CaseInsFields** может содержать строку, в которой перечисляются через точку с запятой те строковые поля, по которым при сортировке не должен учитываться регистр. Таким образом, параметры **DescFields и CaseInsFields,** в отличие от параметра **Options,** позволяют задать разные опции сортировки по разным полям.

Параметр **GroupingLevel** задает уровень группирования по умолчанию и может принимать значения от 0 до числа полей в индексе. Этот уровень имеет смысл, если формируемый индекс будет использоваться при подсчете обобщенных (агрегированных) характеристик.

При вызове метода **AddIndex** набор данных должен быть открыт. Индекс добавляется только на данный сеанс связи с данными. Как только набор будет закрыт, при следующем его открытии индекс уже не будет существовать. Индекс не сохраняется также вместе с данными методом SaveToFile.

Примеры

Пусть в приложении имеется клиентский набор данных **ClientDataSetl** и выпадающий список **ComboBox1.** Мы хотим заполнить **ComboBoxl** именами полей набора данных и предоставить пользователю возможность, выбрав в списке нужное поле, обеспечить индексацию данных по этому полю.

Для решения этой задачи в обработчик события формы **OnCreate** надо вставить операторы:

```
ComboBox1->Items->Assign(ClientDataSet1->FieldList);
ComboBox1->ItemIndex = 0;
```

Первый из этих операторов заносит в **ComboBox1** список полей, предоставляемый свойством **FieldList.**

Для обеспечения индексации по выбранному пользователем полю в обработчик события **OnChange** компонента **ComboBoxl** можно вставить оператор, задающий значение **IndexFieldNames:**

```
ClientDataSet1->IndexFieldNames = ComboBox1->Text;
```

То же самое можно сделать, используя метод **AddIndex:**

```
ClientDataSet1->AddIndex(ComboBox1->Text + "Index",
                         ComboBox1->Text,
                         TIndexOptions() << ixCaseInsensitive,
                         "", "",0);
ClientDataSet1->IndexName = ComboBox1->Text + "Index";
```

Первый оператор формирует индекс с именем "*<имя поля>*Index". В список индекса заносится имя выбранного поля и задается опция, обеспечивающая нечувствительность сортировки к регистру. Второй оператор индексирует набор по сформированному индексу.

В приведенном примере сортировка всегда осуществляется в направлении увеличения значения выбранного пользователем поля. Добавим в приложение группу радиокнопок **RadioGroup1,** которая позволит пользователю выбирать направление сортировки: нарастание или убывание. Тогда приведенный выше обработчик события **OnChange** компонента **ComboBoxl,** использующий **Addlndex,** можно изменить так:

```
AnsiString SDesc = "";
// формирование строки параметра DescFields
if (RadioGroup1->ItemIndex == 1)
  SDesc = ComboBox1->Text;
ClientDataSet1->AddIndex(ComboBox1->Text + SDesc,
                         ComboBox1->Text,
                         TIndexOptions() << ixCaseInsensitive,
                         SDesc, "",0);
ClientDataSet1->IndexName = ComboBox1->Text + SDesc;
```

На этот же обработчик надо сослаться в событии **OnChange** компонента **RadioGroupl. В** приведенном коде несколько изменено имя индекса, но главное отличие от предыдущего варианта — задание направления сортировки не опциями, а параметром **DescFields,** в который заносится строка **SDesc:** пустая или с именем поля в зависимости от выбранной пользователем радиокнопки в компоненте **RadioGroupl.**

Теперь еще усложним задачу, предоставив пользователю возможность формировать индекс с двумя ключами, независимо выбирая для каждого ключа направление сортировки. Для этого добавим на форму еще один список **ComboBox2** и группу радиокнопок **RadioGroup2.** Второй список надо загрузить именами поле так же, как делали раньше.

При использовании свойства **IndexFieldNames** общий обработчик событий всех списков может иметь вид:

```
ClientDataSet1->IndexFieldNames = ComboBoxl->Text +
                                  ';' + ComboBox2->Text;
```

Но в этом варианте нельзя выбирать направление сортировки. Это можно сделать методом **AddIndex.** В этом случае общий обработчик событий всех списков и радиокнопок может иметь вид:

```
AnsiString SDesc = "", SName = ComboBox1->Text;
if (RadioGroup1->ItemIndex == 1)
{
// формирование имени и строки параметра DescFields
  SDesc = ComboBox1->Text + ";";
  SName += "_Desc";
}
if (ComboBox2->Text == ComboBox1->Text)
{
// Если ключи одинаковы, второй ключ игнорируется
  RadioGroup2->ItemIndex = RadioGroup1->ItemIndex;
  RadioGroup2->Enabled = false;
}
else
{
// формирование имени и строки параметра DescFields
  SName += ComboBox2->Text;
  RadioGroup2->Enabled = true;
  if (RadioGroup2->ItemIndex == 1)
  {
    SDesc += ComboBox2->Text;
    SName += "_Desc";
  }
}
ClientDataSet1->AddIndex(SName, ComboBox1->Text + ";" +
   ComboBox2->Text, TIndexOptions() << ixCaseInsensitive,
   SDesc, "",0);
ClientDataSet1->IndexName = SName;
```

Приведенные примеры можно дополнить выпадающим списком, содержащим все индексы, определенные пользователем в данном сеанса. С его помощью пользователь может легко переключаться между разными индексами. Введя такой список (назовем его **CBIndex),** надо добавить в конце обработчика события **OnChange** компонента **ComboBoxl** операторы:

```
ClientDataSet1->GetIndexNames(CBIndex->Items);
CBIndex->ItemIndex =
          CBIndex->Items->IndexOf(ClientDataSet1->IndexName);
```

В них используется метод **GetIndexNames** для получения списка индексов. В обработчик события **OnChange** компонента **CBIndex** надо вставить оператор:

```
ClientDataSet1->IndexName = CBIndex->Text;
```

который заменяет текущий индекс тем, который выбрал пользователь.

Append и AppendRecord — методы TDataSet

Добавляют новую запись в набор данных.

Класс *TDataSet*

Определения

```
void _fastcall Append(void);
void _fastcall AppendRecord(const System::TVarRec * Values,
                            const int Values_Size);
```

Описание

Метод **Append** добавляет новую пустую запись в конец набора данных и делает ее активной. После этого можно вводить данные в ее поля и пересылать эти данные в базу данных методами **Post или ApplvUpdates** (если производится кэширование).

Метод **AppendRecord** добавляет в набор данных новую запись, заполняет ее поля значениями, перечисленными в параметре Values, и пересылает в базу данных. Новая запись делается активной. В массиве Values заносимые значения перечисляются в той последовательности, в которой расположены поля таблицы. На месте значений тех полей, в которые данные не заносятся, пишется **NULL.** После записи последнего ненулевого значения список **Values** можно прервать, т.е. не указывать в нем **NULL** для остальных не заполняемых полей (см. примеры).

Если в записи имеются поля, являющиеся немодифицируемым, например, автоматически нарастающими **(AutoIncrement),** то вместо значения такого поля надо указывать **NULL.**

Место новой записи в базе данных определяется следующими правилами:

- Для индексированных таблиц Paradox и dBASE запись вставляется в позицию, соответствующую индексу
- Для неиндексированных таблиц Paradox и dBASE запись вставляется в конец набора данных
- Для баз данных SQL место расположения новой записи зависит от реализации базы данных. Для индексированных таблиц индекс обновляется с учетом новой записи

Имеется еще два метода внесения новой записи в набор данных — методы **Insert и InsertRecord.** Они отличаются от рассмотренных только позицией размещения записи в физической базе данных.

Примеры

Следующие операторы создают новую запись, заполняют ее поля Fam, Nam, Par (Фамилия, Имя, Отчество) значениями, введенными в окна редактирования **Edit1, Edit2, Edit3** и пересылают запись в базу данных.

```
Table1->Append();
Table1->FieldValues["Fam"] = Edit1->Text;
Table1->FieldValues["Nam"] = Edit2->Text;
Table1->FieldValues["Par"] = Edit3->Text;
Table1->Post();
Table1.Post;
```

В приведенном примере метод **Append** можно было бы заменить методом **Insert.** Никакой разницы в результатах не было бы, кроме, возможно, несколько иного размещения новой записи в базе данных.

Следующий оператор выполняет те же функции, что и в предыдущем примере. В первое поле (номер записи) заносится целое число, во второе поле (отдел работы сотрудника) не заносится ничего. В поля, следующие за полями фамилии, имени и отчества также ничего не заносится.

```
Table1->AppendRecord(ARRAYOFCONST(
                    (18, NULL, Edit1->Text, Edit2->Text, Edit3->Text)));
```

Этот оператор нельзя применить, например, если первое поле (номер записи) является автоматически нарастающим (AutoIncrement), т.е. не модифицируемым.

В приведенном примере метод **AppendRecord** можно было бы заменить методом **InsertRecord.** Никакой разницы в результатах не было бы, кроме, возможно, несколько иного размещения новой записи в базе данных.

AppendRecord

См. раздел «Append и AppendRecord — методы TDataSet».

ApplyUpdates — метод TBDEDataSet

Записывает кэшированные изменения в базу данных.

Класс *TBDEDataSet*

Объявление

```
void _fastcall ApplyUpdates(void);
```

Описание

Метод **ApplyUpdates** записывает кэшированные изменения в базу данных. Данные передаются в базу данных на хранение, но не фиксируются там. Для их фиксации в случае успешной пересылки в базу данных надо вызвать явным образом метод **Commit.** Если же обнаруживается, что фиксация не требуется, то можно отменить все изменения методом **Rollback.**

После успешной фиксации изменений в базе данных следует вызвать метод **CommitUpdates**, чтобы очистить буфер изменений.

Для записи и фиксации изменений в базе данных желательно использовать методы компонента базы данных **TDatabase.**

Примеры

В приведенном ниже обработчике события формы **OnCloseQuery** предполагается, что приложение содержит компонент **Query1** и изменения в этом наборе данных кэшируются. Обработчик выполняет следующие функции. При наличии не сохраненных изменений пользователю задается вопрос «Сохранить изменения в базе данных?». При положительном ответе изменения сохраняются методом **ApplyUpdates.** При отрицательном -- изменения отменяются методом **CancelUpdates.** Если при получении запроса пользователь нажал кнопку Отмена, то приложение не закрывается **(CanClose** задается равным **false).**

```
void _fastcall TForm1::FormCloseQuery(TObject *Sender,
                                      bool &CanClose)
{
 if (Query1->CachedUpdates)
 {
  switch (Application->MessageBox("Сохранить изменения в базе данных?",
                                  "Подтвердите сохранение изменений",
                                  MB_YESNOCANCEL+MB_ICONQUESTION))
  {
   case IDYES:   Query1->ApplyUpdates();
                 break;
   case IDCANCEL:CanClose = false;
                 break;
   case IDNO:    Query1->CancelUpdates();

  }
 }
}
```

См. также пример в гл. 5, в разд. «AfterClose и BeforeClose».

ApplyUpdates, Reconcile — методы TCustomClientDataSet

Передают все изменения данных провайдеру для записи в базу данных.

Класс *TCustomClientDataSet*

Объявления

```
virtual int _fastcall ApplyUpdates(int MaxErrors);
bool _fastcall Reconcile(const System::OleVariant &Results);
```

Описание

Метод ApplyUpdates вызывается для того, чтобы занести все изменения, вставки и удаления записей, сделанные в клиентском наборе данных, в базу данных или в источник данных. Параметр **MaxErrors** определяет максимальное число ошибок, после которого попытка изменения базы данных прерывается. Значение -1 указывает, что выполнение **ApplyUpdates** не прервется при любом числе ошибок. Метод возвращает число ошибок, возникших при обновлении базы данных.

При выполнении **ApplyUpdates** производятся следующие операции:

1. Генерируется событие **BeforeApplyUpdates.**
2. Происходит передача провайдеру значения **Delta,** содержащего все изменения. Те изменения, которые при попытке записи их в базу данных вызывают ошибки, запоминаются.
3. Генерируется событие **After ApplyUpdates.**
4. Вызывается метод **Reconcile.** Он очищает те кэшированные изменения клиентского набора, которые успешно передались в базу данных, и возвращает записи, вызвавшие ошибки и запомненные на шаге 2.

При вызове метода **Reconcile** записи, не внесенные в базу данных, вместе с сообщениями об ошибках возвращаются в параметр **Results.** Для каждой записи, содержащейся в **Results,** генерируется событие **OnReconcileError,** в обработчике которого можно принять решение, что делать с ошибочной записью. В результате все ошибочные записи с учетом действий, предпринятых в обработчике событий **OnReconcileError,** помещаются в свойство **Delta** клиентского набора данных. Метод возвращает **true,** если ошибок не возникало или они устранены.

См. в гл. 5 разд. «OnReconcileError, OnUpdateError — события TCustomClientDataSet».

Assign — метод TCollection

Копирует содержимое собрания объектов в другой контейнер.

Класс *TCollection*

Объявление

```
virtual void __fastcall Assign(TPersistent* Source);
```

Описание

Метод **Assign** копирует содержимое собрания объектов, указанного параметром **Source,** в то собрание, к которому применен метод. При этом в данном собрании удаляются все имеющиеся объекты в свойстве **Items** и для каждого элемента массива **Items** источника **Source** создается его копия в массиве **Items** данной коллекции.

Assign — метод TField

Копирует в свойство **Value** значение другого поля или другого объекта.

Класс *TField*

Объявление

```
virtual void __fastcall Assign(Classes::TPersistent* Source);
```

Описание

Метод Assign копирует в свойство **Value** значение другого поля **Source** или другого объекта. При копировании значений полей они должны быть совместимых типов. Если **Source** — другой объект поля, то Assign вызывает метод **Assign Value,** используя **Source** как значение **Value** поля. Если **Source = NULL,** то значение поля очищается.

Если **Source** — объект, отличный от поля, то вызывается метод **AssignTo.** позволяющий присваивание данного значения объекту **TField.**

Примеры

Следующий оператор копирует графическое поле в поле BLOB:

```
BlobField1->Assign(GraphicField1);
```

Следующий оператор копирует строки из компонента **Memo** в поле **TMetoField:**

```
MemoField1->Assign(Memo1->Lines);
```

Assign — метод TFieldDef

Копирует одно описание поля в другое.

Класс *TFieldDef*

Объявление

```
virtual void__fastcall Assign(Classes::TPersistent* Source);
```

Описание

Метод **Assign** копирует описание поля **Source** в данное описание типа **TFieldDef.** Копируются такие свойства, как **Name, DataType. Size. Precision. InternalCalcField.** Значение свойства **FieldNo** не изменяется.

Если **Source** не является описанием поля, то **Assign** вызывает наследуемые методы копирования, объявленные как копирование описания поля в их методах **AssignTo.**

Assign — метод графических объектов

Копирует изображение одного графического объекта в другой.

Классы *TGraphic*, *TBitMap*, *TIcon*. *TMetaFile*

Объявление

```
virtual void__fastcall Assign(Classes::TPersistent * Source);
```

Описание

Метод **Assign** копирует изображение, содержащееся в объекте **Source, в** данный объект. Типы объектов источника и приемника должны быть одинаковыми. Исключение составляет свойство **Graphic** объекта **TPicture. Graphic** может участвовать в обменах изображениями с объектами типов **TBitMap. TIcon. TMetaFile.**

Объектом копирования для классов **TBitMap. TIcon. TMetaFile** может быть также буфер обмена — объект **Clipboard.** При этом надо не забыть включить в оператор **uses** приложения ссылку на модуль *Clipbrd.* Свойство **Graphic** объекта **TPicture** может участвовать только в копировании в буфер обмена, но не в копировании из буфера.

Примеры

1. Все шесть приведенных ниже операторов делают одно и то же: копируют изображение из компонента **Image2** в компонент Image1. Но последний из них выполняется успешно только в том случае, если тип графического объекта в **Image2 — TBitMap.**

```
Image1->Picture->Bitmap->Assign(Image2->Picture->Bitmap);
Image1->Picture->Graphic->Assign(Image2->Picture->Bitmap);
Image1->Picture->Graphic->Assign(Image2->Picture->Icon);
Image1->Picture->Graphic->Assign(Image2->Picture->MetaFile);
Image1->Picture->Graphic->Assign(Image2->Picture->Graphic);
Image1->Picture->Bitmap->Assign(Image2->Picture->Graphic);
```

2. Каждый из приведенных ниже операторов копирует изображение из компонента **Imagel** в буфер обмена Clipboard.

```
#include <vcl\Clipbrd.hpp>

Clipboard()->Assign(Image1->Picture->Bitmap);
Clipboard()->Assign(Image1->Picture->Graphic);
```

3. Приведенный ниже оператор читает изображение из буфера обмена Clipboard в компонент **Image1**. Если в Clipboard хранится не битовая матрица, будет генерироваться исключение.

```
Image1->Picture->Bitmap->Assign(Clipboard());
```

Assign — метод копирования объектов

Копирует один объект в другой, создавая копию всех данных объекта.

Класс *TPersistent*

Объявление

```
virtual void__fastcall Assign(TPersistent* Source);
```

Описание

Метод **Assign** копирует данные одного объекта в другой. Объявлен в классе **TPersistent** и перегружен в классах, производных от него. Некоторое число классов C++Builder поддерживает присваивание объектов разных типов. Для большинства же классов, производных от **TPersistent,** применение Assign к несовпадающим типам объектов источника и назначения ведет к генерации исключения **EConvertError**.

Метод Assign отличается по результатам от операции присваивания

```
<объект-назначение> := <объект-источник>;
```

При присваивании указатель на <объект-назначение> начинает указывать на <объект-источник>. А метод **Assign** создает новую копию объекта. После применения **Assign** имеется два объекта с одинаковыми данными.

Если объекты разного типа, то при вызове **D.Assign(S)** тип **D** должен «знать», как скопировать в него тип S (тип S может ничего не знать о преобразовании типов). Если метод **Assign** не может осуществить преобразование типов, то он вызывает защищенный метод **AssignTo**, объявленный в классе **TPersistent** и перегруженный в классах, производных от него. Вызов имеет вид **S.AssignTo(D)**. Если и метод **AssignTo** не может осуществить преобразование или если он не перегружен, то вызывается **AssignTo** класса **TPersistent** и генерируется исключение.

Примеры

1. На форме имеется компонент **FontDiaolgl,** позволяющий пользователю выбрать вид шрифта для изображения надписей на форме. Тогда обработчик соответствующего события в разделе меню может иметь вид:

```
if (FontDialog1->Execute())
  Font->Assign(FontDialog1->Font);
```

2. В программе объявлено и заполнено два списка SL1 и SL2 типа **TStringList**. **На** форме имеется компонент **ComboBoxl,** в котором надо отображать один из списков **SL1** или SL2 в зависимости от того, какая кнопка с флажком нажата в группе радиокнопок **RadioGroupl.** Тогда в событие **OnClick** этой группы радиокнопок надо вставить обработчик, показанный ниже. Последний оператор **ComboBox1->ItemIndex = 0** необходим, чтобы изменение списка сразу отобразилось на экране.

```
if (RadioGroup1->ItemIndex == 0)
    ComboBox1->Items->Assign(SL1);
else ComboBox1->Items->Assign(SL2);
    ComboBox1->ItemIndex = 0;
```

3. Метод **Assign** позволяет проводить обмен данными между совершенно разнородными компонентами, например, компонентом буфера обмена **TClipboard** и графическим объектом **TBitmap.** Записать изображение в буфер можно оператором

```
#include <vcl\Clipbrd.hpp>

Clipboard()->Assign(Bitmap);
```

а прочитать из него изображение можно оператором

```
Bitmap->Assign(Clipboard());
```

AssignTo

Копирует один объект в другой, создавая копию всех данных объекта, является защищенным вариантом метода Assign.

Класс *TControl*

Объявление

```
virtual void __fastcall AssignTo(Classes::TPersistent* Dest);
```

где Dest — объект, в который производится копирование.

Пояснения см. в разделе **Assign.**

BeginDrag

Начало процесса перетаскивания компонента.

Класс *TControl*

Объявление

```
void __fastcall BeginDrag(bool Immediate, int Threshold);
```

Описание

Метод **BeginDrag** вызывается, когда начинается процесс перетаскивания (dragging) компонента. Его необходимо вызывать, только если значение свойства **DragMode** компонента равно **dmManual. В** противном случае процесс перетаскивания производится автоматически.

Вызов метода **BeginDrag** целесообразно вставлять в обработчик события **OnMouseDown.** Параметр **Immediate** (немедленно) определяет, сразу ли после нажатия кнопки мыши ее указатель изменит вид на тот, который задан свойством **DragCursor,** и сразу ли начнется процесс перетаскивания. Если параметр **Immediate** задан равным **false,** то указатель мыши не изменяет свой вид и процесс перетаскивания не начинается, пока пользователь не сместит указатель на небольшое расстояние в 5 пикселов. Это позволяет компоненту воспринимать щелчок мыши, не начиная операцию перетаскивания.

Пример

Пример применения метода см. в разд. «OnDragDrop» гл. 5.

BeginUpdate

Предотвращает перерисовку до выполнения метода **EndUpdate.**

Класс *TCollection*

Объявление

```
void__fastcall BeginUpdate(void);
```

Описание

Метод **BeginUpdate** предотвращает перерисовку экрана вплоть до выполнения метода **EndUpdate**. Метод имеет смысл применять перед началом изменения объектов, входящих в собрание некоторого визуального компонента. Тем самым ускоряется изменение перечня объектов собрания и предотвращается неприятное мерцание изображения в процессе изменения перечня объектов.

Пример

Следующий пример создает в компоненте CoolBar1 три новых полосы и дает им заголовки с номерами полос. Применение методов **BeginUpdate и EndUpdate** предотвращает мерцание изображения в процессе создания новых полос.

```
CoolBar1->Bands->BeginUpdate();
for (int i=1; i <= 3; i++)
{
 CoolBar1->Bands->Add();
 CoolBar1->Bands->Items[CoolBar1->Bands->Count-1]->Text =
              "Полоса " + IntToStr(CoolBar1->Bands->Count);
}
CoolBar1->Bands->EndUpdate ();
```

BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark

Обеспечивают работу с закладками наборов данных.

Классы *TDataSet, TBDEDataSet*

Определения

```
__property AnsiString Bookmark

virtual bool__fastcall BookmarkValid(void * Bookmark);
virtual int__fastcall CompareBookmarks(void * Bookmark1,
                                       void * Bookmark2);

virtual void *__fastcall GetBookmark(void);
void__fastcall GotoBookmark(void * Bookmark);
virtual void__fastcall FreeBookmark(void * Bookmark);
```

Описание

Закладка — это метка, которой можно пометить запись набора данных, чтобы впоследствии быстро переходить на нее (см. в гл. 3 свойство **Bookmark**). Закладки позволяют пометить одну или несколько записей набора данных и затем при необходимости быстро переходить на них. Свойство Bookmark — текущая закладка (ею становится каждая новая закладка) и методы работы с закладками **BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark** объявлены в классе **TDataSet** и переопределены в классе **TBDEDataSet.**

Чтобы создать новую закладку в приложении, надо объявить переменную типа **TBookmark** и присвоить ей функцией **GetBookmark** значение, соответствующее текущей записи. Процедура **GotoBookmark** позволяет в дальнейшем в любой момент перейти на запись, к которой относится введенная вами закладка. Метод **FreeBookmark** освобождает память, выделенную под закладку. Этот метод необходимо вызывать перед тем, как присвоить вашей переменной типа **TBookmark** новое значение.

Перед переходом на закладку можно проверить, доступна ли она, методом **BookmarkValid.** Он вернет **true,** если указанная в его параметре закладка существует. Метод **CompareBookmarks** позволяет сравнить две закладки **Bookmarkl**

и Bookmark2, в частности, установить, не указывают ли они на одну и ту же запись. Этот метод возвращает -1, если **Bookmarkl < Bookmark2,** возвращает 1, если **Bookmarkl > Bookmark2,** и возвращает 0, если закладки идентичны или равны **NULL.**

Пример

Пример работы с закладками см. в гл. 3, в разд. «Bookmark».

BringToFront

Перенос компонента наверх Z-последовательности.

Класс *TControl*

Объявление

```
void__fastcall BringToFront(void);
```

Описание

Метод **BringToFront** позволяет изменять последовательность перекрытия компонентов на форме и тем самым управлять видимостью компонентов.

Перекрывающиеся компоненты на форме размещаются поверх друг друга в последовательности (называемой Z-последовательностью), соответствующей порядку размещения компонентов в процессе проектирования. Например, если вы поместили в одно и то же место формы две кнопки одинаковых размеров, то видна будет только вторая из размещенных кнопок, поскольку она расположена в Z-последовательности выше. Применение во время выполнения приложения метода **BringToFront** к нижней кнопке переместит ее наверх в Z-последовательности и она станет видна пользователю.

Это справедливо по отношению к неоконным объектам, таким, как кнопки, метки, изображения и т.д., а также и к оконным компонентам, таким, как **Memo,** ComboBox и др. Но все неоконные компоненты всегда расположены в Z-последовательности ниже оконных и метод **BringToFront** не может изменить это правило. Например, попытка перенести наверх методом **BringToFront** метку, размещенную под оконным компонентом, ни к чему не приведет.

Примеры

1. Пусть вы хотите, чтобы в каком-то месте формы размещалась кнопка, которая в зависимости от текущего режима работы имела бы два различных набора свойств и выполняла бы различные функции. Вы можете разместить в нужном месте две кнопки друг на друге (пусть они имеют имена **Buttonl** и **Button2**), задать каждой нужные свойства и для каждой описать соответствующие обработчики событий. Тогда для смены этих кнопок вы в соответствующих местах кода программы пишете операторы

   ```
   Button1->BringToFront();
   ```

   или

   ```
   Button2->BringToFront ();
   ```

   и пользователь будет видеть то одну, то другую из этих кнопок.

2. Пусть в приложении в одном и том же месте формы друг на друге расположены две панели: **Panell и на** ней **Pane12,** содержащие какие-то управляющие компоненты для разных режимов работы. **Pane12** расположена на **Panel1,** которая является, таким образом, ее родителем. В обработчик события формы **OnCreate** можно вставить операторы:

   ```
   Panel2->Parent = Form1;
   Panel2->BoundsRect = Panel1->BoundsRect;
   Panell->BringToFront();
   ```

Первый оператор делает родительским компонентом панели Panel2 форму **Form1**. Второй оператор задает панели **Panel2** то же местоположение и размеры, которые имеет панель **Panel1**. Последнее необходимо, поскольку при проектировании ее координаты соответствовали координатному пространству контейнера – клиентской области панели **Panel1**. А теперь ее родитель сменился на форму, и надо ее расположить в том же месте формы, в котором расположена **Panel1**. Третий оператор перемещает наверх форму **Panel1**.

Приведенный код можно сократить, и убрать из него первый оператор, если в процессе проектирования размещать панель Panel2 не на панели **Panel1**, а в каком-то другом месте непосредственно на форме. Тогда в обработчике события формы **OnCreate** достаточно двух операторов:

```
Panel2->BoundsRect = Panel1->BoundsRect;
Panel1->BringToFront();
```

изменяющих положение **Panel2** и перемещающих наверх форму **Panel1**.

В результате работы приведенных операторов в момент создания формы на ней будет видна панель **Panel1**. В момент, когда ее надо заменить на **Panel2**, можно выполнить оператор:

```
Panel2->BringToFront();
```

выносящий наверх вторую панель. Когда надо вернуть на экран изображение **Panel1**, можно выполнить операторы:

```
Panel1->BringToFront();
```

Аналогичный пример приведен в разделе **Visible**, но использование метода **BringToFront** делает его более компактным.

BrushCopy

Копирует часть изображения битовой матрицы на данную канву, заменяя указанный цвет в изображении на значение, установленное для кисти канвы.

Класс *TCanvas*

Объявление

```
void __fastcall BrushCopy(const Windows::TRect &Dest, TBitmap* Bitmap,
                          const Windows::TRect &Source, TColor Color);
```

Описание

Метод **BrushCopy** сохраняется в C++Builder в основном для обратной совместимости с ранними версиями. Вместо **BrushCopy** лучше использовать класс **TImageList**.

Метод копирует часть изображения битовой матрицы компонента **Bitmap** на данную канву, заменяя указанный цвет **Color** в изображении на значение, установленное для кисти канвы **Brush**. Параметр **Source** указывает копируемую прямоугольную область в источнике изображения **Bitmap**. Параметр **Dest** указывает прямоугольную область на канве, в которую производится копирование.

Замена цвета делает изображение как бы частично прозрачным, если в параметре **Color** указать цвет фона изображения, а в параметре **Color** кисти **Brush** канвы указать цвет фона канвы.

Пример

Оператор

```
Form1->Canvas->BrushCopy(Rect(10,10,100,100),Bitmap1,
                         Rect(10,10,100,100),clBlack);
```

копирует прямоугольную область с координатами углов (10, 10) и (100, 100) из компонента **Bitmap1** в аналогичную область канвы формы **Form1** и заменяет в изображении черный цвет на цвет, установленный в свойстве **Form1.Canvas.Brush.Color**.

Cancel и другие методы отмены исправлений в наборах данных

Отменяют результаты редактирования.

Классы *TDataSet , TBDEDataSet, TCustomClientDataSet, TCustomADODataSet*

Объявления

Все классы:

```
virtual void__fastcall Cancel(void);
void__fastcall CancelUpdates(void);
```

TBDEDataSet и TCustomClientDataSet:

```
void__fastcall RevertRecord(void);
```

TCustomClientDataSet :

```
bool__fastcall UndoLastChange(bool FollowChange);
```

TCustomADODataSet:

```
enum TAffectRecords {arCurrent, arFiltered, arAll, arAllChapters};
void__fastcall CancelBatch(TAffectRecords AffectRecords = arAll);
```

Описание

Метод **CancelUpdates** отменяет все исправления, еще не зафиксированные в базе данных.

В классе **TBDEDataSet** применение метода **CancelUpdates** приводит к удалению всех кэшированных изменений и восстановлению состояния набора, соответствующего моменту открытия набора или моменту после последней фиксации изменений в базе данных. При закрытии набора данных или при установке свойства **CachedUpdates в false** метод **CancelUpdates** вызывается автоматически. Так что все не зафиксированные результаты редактирования пропадают.

В классе **TCustomClientDataSet** применение метода **CancelUpdates** приводит к полной очистке свойства **Delta.** В этом классе и в классе **TBDEDataSet** определен также метод **RevertRecord,** который удаляет результаты редактирования только текущей записи, если они еще не занесены в базу данных или не присоединены к **Data** (в клиентском наборе).

Метод **Cancel** так же, **как и RevertRecord,** отменяет результаты редактирования текущей записи, но только до тех пор, пока не был выполнен метод **Post.** Поэтому его целесообразно использовать перед выполнением Post (см. пример).

Метод **UndoLastChange** класса TCustomClientDataSet ликвидирует результаты последней операции редактирования. Это может быть операция изменения данных, или вставка новой записи, или удаление записи. Параметр **FollowChange** указывает положение курсора после восстановления записи. Если **FollowChange = true,** курсор перемещается на восстановленную запись. А если **FollowChange = false,** то курсор остается на текущей записи.

Необходимо отметить, что в наследниках **TCustomClientDataSet** имеется еще один инструмент отмены всех изменений и возврат к любому заранее определенному состоянию набора данных — свойство **SavePoint.**

В классе **TCustomADODataSet** применение метода **CancelUpdates** приводит к удалению всех кэшированных изменений и восстановлению прежнего состояния набора данных. В классе имеется еще один метод удаления результатов редактирования — **CancelBatch.** Он применим, если набор данных открыт в режиме пакетного обновления (свойство **CursorType** равно **ctKeySet или ctStatic,** свойство **LockType равно ltBatchOptimistic,** команда является запросом SELECT).

Параметр **AffectsRecords** указывает, исправления чего именно удаляются. Значение **arAll** (принято по умолчанию) указывает на отмену исправлений всех записей, так что **CancelBatch(arAll)** эквивалентно **CancelUpdates.** Значение **arCurrent** отменяет редактирование только текущей записи. Значение **arFiltered** отме-

няет изменения всех записей, соответствующих текущему фильтру. Значение **arAllChapters** действует на все разделы ADO.

Если набор данных ADO открыт в режиме немедленного обновления, то вызов CancclBatch со значением **AffectsRecords,** отличным от **arCurrent,** приводит к генерации исключения.

Примеры

Оператор

```
Table1->CancelUpdates();
```

отменит все кэшированные результаты редактирования набора данных **Tablel** (для кэширования в нем должно быть установлено в **true** свойство **CachedUpdates**). А оператор

```
Table1->RevertRecord();
```

отменит результат редактирования только текущей записи. Так что вы можете предоставить пользователю возможность перемещаться по набору данных и выборочно отменять результаты редактирования отдельных записей.

См. также примеры в разд. «Ap ply Updates» и «Post».

CancelBatch

Удаляет все кэшированные изменения и восстанавливает прежнее состояние набора данных.

См. разд. «Cancel и другие методы отмены исправлений в наборах данных».

CancelUpdates

Отменяет все кэшированные изменения и восстанавливает исходное состояние набора данных.

См. разд. «Cancel и другие методы отмены исправлений в наборах данных».

CanFocus

Определяет, может ли компонент получать сообщения пользователя.

Класс *TWinControl*

Объявление

```
bool __fastcall CanFocus(void);
```

Описание

Метод **CanFocus** определяет, может ли компонент получать сообщения пользователя, т.е. может ли он получать фокус. Функция возвращает **true,** если у компонента и всех его родителей свойства **Visible и Enabled** установлены в **true.** В противном случае возвращается **false.**

ChangeScale

Изменяет масштаб компонента и его дочерних компонентов.

Классы *TControl, TCustomForm, TScrollingWinControl, TWinControl*

Объявление

```
DYNAMIC void __fastcall ChangeScale(int M, int D);
```

Описание

Метод **ChangeScale** используется для изменения масштаба компонента. Масштабируются такие свойства компонента, как **Top** и **Left,** определяющие его местоположение, а также **Width и Height,** определяющие его размер. В классах, про-

изводных от **TControl**, *в* частности, в **TWinControl**, масштабируются также все компоненты, принадлежащие данному компоненту, и их шрифты.

Параметры M and D определяют соответственно множитель и делитель масштаба. Например, чтобы уменьшить размеры до 75% начального значения, можно задать M равным 75, а D равным 100 (75/100). То же самое можно сделать, задав M=3 и D=4 (3/4). Если вы хотите увеличить размер на 1/3, то можно задать M=133 и D=100 (133/100) или M=4 и D=3 (4/3).

CheckOpen

Проверяет, отсутствует ли ошибка при попытке доступа к данным.

Класс *TDBDataSet*

Объявление

```
bool __fastcall CheckOpen(Word Status);
```

Описание

Метод **CheckOpen** проверяет, не вернула ли BDE ошибку при попытке доступа к данным. Тип **DBIResult** — это объявленный в модуле *BDE.int* тип ошибок BDE.

CheckOpen возвращает **true,** если доступ к данным получен. Если Status указывает на ограничения прав доступа к таблицам Paradox или dBASE, **CheckOpen** вызывает метод **GetPassword** компонента Session, т.е. запрашивает пароль пользователя. Если пользователь введет правильный пароль, то **CheckOpen** вернет **true.**

Если попытка доступа к данным завершилась неудачей, **CheckOpen** возвращает **false.**

Chord

Рисует заполненную замкнутую фигуру, ограниченную дугой окружности или *эллипса* и хордой.

Класс *TCanvas*

Объявление

```
void __fastcall Chord(int X1, int Y1, int X2, int Y2,
                      int X3, int Y3, int X4, int Y4);
```

Описание

Метод **Chord** рисует замкнутую фигуру: дугу окружности или эллипса, замкнутую хордой, с помощью текущих параметров пера Реп. Фигура заполняется текущим значением **Brush**. Точки (X1, Y1) и (X2, Y2) определяют прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса *с* прямой, проходящей через его центр и точку (X3, Y3). Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (X4, Y4). Дуга рисуется против часовой стрелки от начальной до конечной точки. Хорда соединяет точки (X3, Y3) и (X4, Y4).

В Windows 95 суммы X1 + X2, Y1 + Y2 и X1 + X2 + Y1 + Y2 не должны превышать 32768.

В Windows NT направление рисования дуги можно изменить на направление по часовой стрелке вызовом функции **SetArcDirection.**

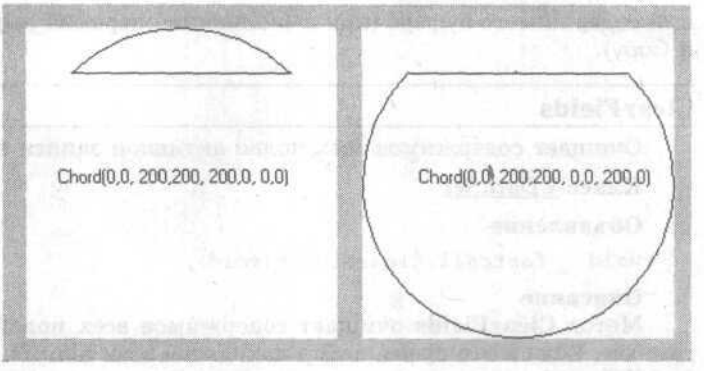
Примеры

Операторы

```
Image1->Canvas->Chord(0,0, 200,200, 200,0, 0,0);
Image2->Canvas->Chord(0,0, 200,200, 0,0, 200,0);
```

дают результат, показанный на рис. 4.1.

Рис. 4.1
Примеры применения метода Chord



ClassName

Возвращает имя типа объекта.

Класс *TObject*

Объявление

```
static ShortString __fastcall ClassName(TClass cls);
```

Описание

Метод **ClassName** возвращает имя действительного типа объекта. Например, переменная типа класса-предка может ссылаться на экземпляр любого типа-потомка. В этом случае **ClassName** возвращает имя реального типа объекта, а не того, которое было объявлено для этой ссылки. Например, оператор

```
catch(Exception & E)
{
 ShowMessage("Возникло исключение " + E.ClassName());
}
```

перехватит все исключения и отобразит сообщение с именем действительно сгенерированного исключения. Впрочем, в данной ситуации лучше применить оператор

```
catch(Exception & E)
{
 ShowMessage("Возникло исключение " + E.Message);
}
```

который отобразит пользователю не класс исключения, а сообщение этого класса.

Clear — метод списков

Очистка списков.

Классы *TClipboard, TList, TStringList, TStrings, TComboBox, TDBComboBox, TDBEdit, TDBListBox, TDBMemo, TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TListBox, TMaskEdit, TMemo, TOutline* и ряд других

Объявление

```
DYNAMIC void __fastcall Clear(void);
```

Описание

Для перечисленных выше объектов и компонентов процедура Clear удаляет все элементы списков или весь текст. Для некоторых других объектов аналогичная процедура действует несколько иначе.

Для объекта **Clipboard** процедура **Clear** удаляет все содержимое буфера Clipboard. Впрочем, то же самое происходит автоматически при каждом обновле-

нии содержимого буфера (при выполнении операций вырезать и копировать — Cut и Copy).

ClearFields

Очищает содержимое всех полей активной записи набора данных.

Класс *TDataSet*

Объявление

```
void __fastcall ClearFields(void);
```

Описание

Метод **ClearFields** очищает содержимое всех полей активной записи набора данных. Если в это время набор находится в состоянии **State.** отличном от **dsEdit** или **dsInsert,** генерируется исключение. Пересчитываются все вычисляемые поля и генерируется событие **OnDataChange** источника данных, связанного с данным набором данных (все это происходит, если в данный момент не выполняются операции **SetKey**).

ClientToParent

Пересчитывает координаты точки компонента в координаты указанного родительского компонента.

Класс *TControl*

Объявление

```
struct TPoint // полное определение см. в гл. 6
{
 int  x;
 int  y;
};

TPoint __fastcall ClientToParent(const TPoint Point,
                                 TWinControl *AParent = (TWinControl*) NULL);
```

Описание

Метод **ClientToParent** (введен в C++Builder 6) преобразует координаты точки **Point** данного компонента в систему координат родительского компонента **AParent.** Компонент **AParent** может быть непосредственным родителем — контейнером, содержащим данных компонент. А может быть одним из его предшественников — компонентом, который можно найти по цепочке ссылок свойств **Parent.**

Если параметр **AParent** равен **NULL** или просто не задан, подразумевается непосредственный родитель данного компонента.

Если параметр **AParent** не соответствует родительскому компоненту или если свойство **Parent** данного компонента равно **NULL,** генерируется исключение **EInvalidOperation.**

Пример

Пусть в вашем приложении имеется панель **Panell,** размещенная непосредственно на форме **Form1** или на какой-то другой панели. И пусть на панели **Panell** размещена панель **Pane12.** Вы хотите, чтобы при щелчке на некоторой кнопке панель **Pane12** размещалась непосредственно на форме **Forml,** не изменяя при этом своего положения. Это может потребоваться, например, чтобы **Pane12** не сдвигалась при перетаскивании панели **Panel1.**

Задача решается следующим кодом:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 TPoint Coord = Panel1->ClientToParent(
```

```
        Point(Panel2->BoundsRect.Left, Panel2->BoundsRect.Top), Form1);
  Panel2->SetBounds(Coord.x, Coord.y,
                    Panel2->Width, Panel2->Height);
  Panel2->Parent = Form1;
}
```

Первый оператор переводит координаты левого верхнего угла панели **Panel2,** которые заданы в системе координат контейнера **Panel 1,** в систему координат формы **Forml** и запоминает эти координаты в переменной **Coord.** Следующий оператор изменяет координаты левого верхнего угла панели **Panel2.** А последний оператор размещает панель **Panel2** непосредственно на форме **Forml.**

Если панель **Panell** размещается непосредственно на форме, то второй параметр **(Forml)** в вызове **ClientToParent** можно вообще не указывать или задать равным NULL.

ClientToScreen

Преобразует координаты клиентской области в координаты экрана.

Класс *TControl*

Объявление

```
struct TPoint // полное определение см. в гл. 6
{
 int  x;
 int  y;
};

Windows::TPoint __fastcall
            ClientToScreen(const Windows::TPoint &Point);
```

Описание

Метод **ClientToScreen** преобразует координаты точки в системе координат клиентской области компонента (начало координат — левый верхний угол клиентской области) в систему координат экрана (начало координат — левый верхний угол экрана).

Совместно с обратной функцией **ScreenToClient** метод может использоваться для пересчета координат точки экрана из системы координат клиентской области одного компонента в систему координат клиентской области другого компонента.

Пример

```
P = Comp2->ScreenToClient(Comp1->ClientToScreen(P));
```

Оператор пересчитывает координату точки P из системы координат компонента **Comp1** в систему координат компонента Comp2.

Close

Метод закрывает набор данных.

Класс *TDataSet*

Объявление

```
void __fastcall Close(void);
```

Описание

Метод **Close** закрывает набор данных, устанавливая его свойство **Active в false.** Состояние набора данных **State** переводится в **dsInactive** и закрывается курсор BDE. После этого ни читать из данного набора данных, ни записывать в него невозможно.

Закрывать набор данных методом **Close** или непосредственным заданием **false** свойству **Active** надо перед изменением любых свойств, оказывающих влияние на состояние набора данных.

CloseDatabase

Закрывает соединение с базой данных.

Класс *TDBDataSet*

Объявление

```
void __fastcall CloseDatabase(TDatabase* Database);
```

Описание

Метод **CloseDatabase** закрывает соединение с базой данных. Параметр **Database** определяет временный (созданный ранее методом OpenDatabase) или постоянный компонент типа **TDatabase,** соединение которого закрывается. Вызов **CloseDatabase** уменьшает на 1 число ссылок этого компонента. Если число ссылок стало равно нулю и свойство **KeepConnection** (поддерживать соединение) этого компонента равно **false,** то временный компонент **Database** уничтожается, освобождая память. Временный компонент **Database** закрывается также автоматически, если закрылся последний связанный с ним набор данных. Метод **CloseDatabase** просто форсирует это закрывание. Если в компоненте **Session**, с которым связан временный компонент **Database,** свойство **KeepConnection = true,** то при закрытии соединения **Database** не уничтожается. Для его уничтожения после закрытия соединения надо вызвать метод **DropConnections** компонента Session.

Для постоянного компонента **Database**, чтобы закрыть соединение, надо явным образом вызвать его метод **Close.**

CommitUpdates

Очищает буфер кэшированных изменений.

Класс *TBDEDataSet*

Объявление

```
void __fastcall CommitUpdates(void);
```

Описание

Метод **CommitUpdates** очищает буфер кэшированных изменений после успешного вызова метода **ApplvUpdates** или метода **Commit** базы данных. Очистка кэша гарантирует, что в нем не осталось записей, кроме необработанных или пропущенных обработчиками событий **OnUpdateRecord** или **OnUpdateError**. Оставшиеся записи еще можно пытаться модифицировать.

Модификации оставшихся записей опять кэшируются и в дальнейшем должны пересылаться в базу данных методом **Apply Updates.**

Пример

См. пример в гл. 5, в разд. «AfterClose и BeforeClose».

CompareBookmarks

См. в разд. «BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark».

ConstraintsPisabled, DisableConstraints, EnableConstraints

Методы, управляющие доступностью ограничений сервера.

Класс *TBDEDataSet*

Объявления

```
bool __fastcall ConstraintsDisabled(void);
void __fastcall DisableConstraints(void);
void __fastcall EnableConstraints(void);
```

Описание

Метод **DisableConstraints** временно блокирует применение ограничений сервера к набору данных. Например, при организации циклов по записям отключение ограничений ускоряет работу. Каждый вызов **DisableConstraints** увеличивает число блокировок на 1.

Метод **EnableConstraints** снимает введенную методом **DisableConstraints** блокировку. Точнее, вызов этого метода уменьшает на 1 число блокировок. Когда это число станет равно 0, блокировка снимется.

Функция **ConstraintsDisabled** проверяет наличие блокировки ограничений. Она возвращает **true,** если блокировка имеется. Совместное применение **ConstraintsDisabled и EnableConstraints** позволяет снять многократную блокировку. Например:

```
while (Table1->ConstraintsDisabled())
  Table1->EnableConstraints();
```

ContainsControl

Определяет, является ли указанный компонент прямым или косвенным наследником данного оконного компонента.

Класс *TWinControl*

Объявление

```
bool __fastcall ContainsControl(TControl* Control);
```

Описание

Метод **ContainsControl** позволяет определить, является ли компонент, указанный параметром **Control,** наследником данного оконного компонента. Метод возвращает **true** не только, если в свойстве **Controls** компонента **Control** указан в качестве родителя данный компонент, но и если он является прямым или косвенным потомком какого-то из дочерних компонентов данного оконного компонента.

ControlAtPos

Определяет, какой дочерний компонент имеется в указанной позиции.

Класс *TWinControl*

Объявление

```
struct TPoint // полное определение см. в гл. 6
{
 int  x;
 int  y;
};

TControl* __fastcall ControlAtPos(const Windows::TPoint &Pos,
                                  bool AllowDisabled);
```

Описание

Метод **ControlAtPos** позволяет определить, какой дочерний компонент данного оконного элемента имеется в позиции с координатами, указанными параметром **Pos.** Возвращается только непосредственно дочерний компонент, т.е. такой, в чьем свойстве **Parent** указан данный оконный элемент и который поэтому входит в список дочерних компонентов, содержащийся в свойстве **Controls** оконного элемента.

Позиция **Pos** может находиться в любом месте внутри дочернего компонента. Если заданная позиция не соответствует никакому дочернему компоненту, то функция **ControlAtPos** возвращает **NULL.**

Параметр **AllowDisabled** определяет, учитываются ли при поиске компоненты, которые находятся в недоступном состоянии.

ControlsDisabled

См. раздел «DisableControls, EnableControls, ControlsDisabled».

CopyRect

Копирует часть изображения с другой канвы на данную.

Класс *TCanvas*

Объявление

```
void __fastcall CopyRect(const Windows::TRect SDest,
                         TCanvas* Canvas,
                         const Windows::TRect &Source);
```

Описание

Метод **CopyRect** переносит указанную параметром **Source** область изображения в канве источника изображения **Canvas** в указанную параметром **Dest** область данного объекта **TCanvas.** Копирование производится в режиме, установленном свойством **CopyMode.**

Пример

Оператор

```
Image1->Canvas->CopyRect(MyRect2,Bitmap->Canvas,MyRect1);
```

копирует на канву компонента Image1 в область **MyRect2** изображение из области **MyRectl** канвы компонента **Bitmap.**

См. также примеры в гл. 3, в описании свойства **CopyMode.**

CreateBlobStream

Создает поток для чтения и записи данных поля BLOB.

Класс *TDataSet*

Объявление

```
virtual Classes::TStream* __fastcall CreateBlobStream(
                          TField* Field, TBlobStreamMode Mode);
```

Описание

Метод **CreateBlobStream** создает поток для чтения и записи данных поля **Field** типа BLOB. Параметр **Mode** определяет режим создаваемого потока: **bmRead** — чтение данных, **bmWrite** — запись данных, **bmReadWrite** — и чтение, и запись.

Поток создается для конкретной записи и конкретного поля. Поэтому при переходе к другой записи поток нельзя использовать повторно. Он должен создаваться заново.

Создание потока методом **CreateBlobStream** предпочтительнее непосредственного создания объекта **TBlobStream** операцией new.

Пример

Приведенный ниже код обеспечивает копирование фотографии из поля **Photo** текущей записи набора данных **Tablel** в поле **Photo** текущей записи набора данных **Table2.**

```
// это пример создания потока операцией new
TBlobStream * Streaml = new TBlobStream(Table1Photo, bmRead);
try
{
  Table2->Edit();
// это пример создания потока методом CreateBlobStream
 TStream * Stream2 =
   Table2->CreateBlobStream(Table2->FieldByName("Photo"),
                            bmReadWrite);
  try
  {
   Stream2->CopyFrom(Stream1, Stream1->Size);
   Table2->Post();
  }
  __finally
  {
   delete Stream2;
  }
 }
 __finally
 {
  delete Stream1;
 }
}
```

CreateDataSet — метод TCustomClientDataSet

Создает новый набор данных.

Класс *TCustomClientDataSet*

Объявление

```
void __fastcall CreateDataSet(void);
```

Описание

Метод **CreateDataSet** позволяет во время выполнения создавать пустой набор данных, который в дальнейшем пользователь может редактировать и сохранять. Структура набора данных, т.е. список поле, берется из свойства **FieldDefs**, если оно заполнено, или из свойства **Fields.** Если оба эти свойства пустые, генерируется исключение.

С помощью метода **CreateDataSet** можно обеспечить пользователю возможность создавать новый набор данных во время выполнения (см. пример). Метод можно использовать также неявно во время проектирования. Для этого сначала с помощью Инспектора Объектов надо заполнить свойство **FieldDefs.** Тогда при щелчке правой кнопкой мыши на компоненте клиентского набора данных в меню появится раздел Create Data Set. Выбор этого раздела обеспечит создание набора данных.

Пример

Пусть мы хотим предоставить пользователю возможность во время выполнения создавать набор данных, указывая имена и типы полей. Этот набор данных должен сохраняться после того, как пользователь провел его редактирование, в указанном пользователем файле XML.

Разместите на форме компонент **ClientDataSetl** типа **TClientDataSet.** источник данных **DataSource** и любые компоненты отображения данных, дающие пользователю возможность редактировать набор (например, компонент **DBGrid**). Никаких свойств компонента **ClientDataSetl** задавать не надо.

А для диалога создания набора данных добавьте на форму окно редактирования **Edit** (его имя в приведенном ниже коде **EFile**), в котором пользователь может задать имя файла. Добавьте также окно редактирования, в котором пользователь

сможет задавать имя поля (имя этого окна EField). Отметим, что имена можно задавать русские и состоящие из нескольких слов. Для задания типа поля введите в приложение выпадающий список **ComboBoxl** и задайте в нем строки "int" и "string" (можете, конечно, добавить другие типы полей). Наконец, для задания размера строкового поля добавьте компонент CSpinEditl. Управлять всем этим будут две кнопки: Add — добавление описания нового поля, и Create — создание набора данных.

Введите в раздел **private** описания класса формы две переменные:

```
TFieldDefs *pDefs;
TFieldDef *pDef;
```

Обработчик щелчка на кнопке Add может иметь вид:

```
pDefs = Form1->ClientDataSet1->FieldDefs;
pDef = pDefs->AddFieldDef();
pDef->DisplayName = EField->Text;
if (ComboBox1->ItemIndex == 0)
   pDef->DataType=ftInteger;
else
{
 pDef->DataType=ftString;
 pDef->Size = CSpinEdit1->Value;
}
```

Первый оператор устанавливает указатель **pDefs** на набор описаний полей **FieldDefs.** Второй добавляет в набор новое описание. Имя нового поля берется из окна **EField.** В зависимости от выбора пользователя в списке **ComboBoxl** задается тип и размер поля.

Обработчик щелчка на кнопке Create может иметь вид:

```
ClientDataSetl->CreateDataSet();
ClientDataSet1->FileName = EFile->Text;
ClientDataSet1->Open();
```

После щелчка на этой кнопке пользователь увидит в компонентах отображения данных записи созданного набора и может вводить данные. По окончании работы приложения данные сохранятся в файле, указанном пользователем в окне **FileName,** и могут быть прочитаны в следующем сеансе работы.

В приведенном примере не создавались индексы набора данных. Представляется, что проще предоставить пользователю возможность самому выбирать любую индексацию, как показано в примере, приведенном в описании метода **AddIndex.**

CreateTable

Метод создает новую таблицу данных.

Класс *TTable*

Объявление

```
void__fastcall CreateTable(void);
```

Описание

Метод **CreateTable** создает новую таблицу данных на основании указанной информации о ее структуре: свойств **FieldDefs** или **Fields.** характеризующих поля, и свойства **IndexDefs.** описывающего индексы. Чтобы избежать уничтожения уже существующей таблицы с тем же именем, перед вызовом **CreateTable** полезно проверить существование таблицы, обратившись к свойству **Exists.**

Пример

Приведенный ниже код создает в базе данных, с которой связан компонент **Table1,** таблицу Paradox с именем «Dep.db» с полями **Dep, Proisv** и заполняет ее.

```
// Компонент Table1 делается неактивным
Table1->Active = False;
// Указывается имя таблицы
Table1->TableName = "Dep.db";
// Таблица создается, если ее нет в базе данных
if(! Table1->Exists)
{
// Указывается тип таблицы
 Table1->TableType = ttParadox;
// Начало описания полей таблицы
 Table1->FieldDefs->Clear();
// Создается указатель на объект описания поля
 TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
// Описание первого поля
 pNewDef->Name = "Dep";
 pNewDef->DataType = ftString;
 pNewDef->Size = 20;
 pNewDef->Required = true;
// Описание второго поля
 pNewDef = Table1->FieldDefs->AddFieldDef();
 pNewDef->Name = "Proisv";
 pNewDef->DataType = ftBoolean;

// Описание индекса
 Table1->IndexDefs->Clear();
// Индекс без имени - первичный ключ таблицы
 Table1->IndexDefs->Add("","Dep",
            TIndexOptions() <<ixPrimary << ixUnique);

// Создание таблицы методом CreateTable
 Table1->CreateTable();
 Table1->Open();
// Вставка первой записи
 Table1->Insert();
 Table1->FieldByName("Dep")->AsString = "Бухгалтерия";
 Table1->FieldByName("Proisv")->AsBoolean = false;
 Table1->Post ();
...
```

CustomAlignInsertBefore, CustomAlignPosition

Заказное выравнивание.
См. разд. «Align» в гл. 3.

DataRequest, OnDataRequest — методы и событие

Обеспечивают запрос провайдеру с дополнительными условиями.

Классы *TCustomClientDataSet*, *TDataSetProvider*

Объявления
В TCustomClientDataSet:

```
OleVariant __fastcall DataRequest(OleVariant Data);
```

В TCustomProvider:

```
virtual System::OleVariant __fastcall
            DataRequest(const System::OleVariant &Input);

typedef System::OleVariant __fastcall
        (__closure *TDataRequestEvent)
        (System::TObject *Sender, System::OleVariant Input);

__property TDataRequestEvent OnDataRequest
```

Описание

Метод DataRequest клиентских наборов данных (объявлен в классе TCustomClientDataSet) позволяет послать провайдеру запрос, передав также некоторые условия. При вызове метода в аргумент Data передаются некоторые данные, которые провайдер должен использовать при выполнении запроса. Например, это может быть фильтр, по которому отбираются записи.

Вызов метода **DataRequest** клиентского набора автоматически приводит к вызову метода **DataRequest** того провайдера, с которым связан данный набор своим свойством **ProviderName.** В качестве значения параметра **Input** передается значение **Data,** заданное при вызове метода клиента. В результате этого автоматического вызова метода **DataRequest** провайдера, генерируется событие провайдера **OnDataRequest.** В обработчик этого события передается **Sender** — указатель на объект провайдера (не клиента, а провайдера), и **Input** — значение **Data,** переданное в первичном вызове. По умолчанию событие **OnDataRequest** не имеет обработчика. Так что если вы не напишете соответствующий обработчик, метод **DataRequest** вернет **NULL.** Но, написав этот обработчик, вы можете вернуть в нем данные, которые отображают результат запроса с учетом переданных условий.

Примеры

Ниже приведен пример, в котором пользователь клиентского приложения может, нажав некоторую кнопку, получить из базы данных набор записей, удовлетворяющих фильтру, введенному в окне редактирования **FilterEdit.** Обработчик щелчка на этой кнопке имеет вид:

```
ClientDataSetl->Data = ClientDataSet1->DataRequest(FilterEdit->Text);
```

Этот оператор заносит в свойство **Data** клиентского набора **ClientDataSet1** результаты, отфильтрованные фильтром **FilterEdit->Text.** Надо только отметить, что если клиентский набор использует свойство **FileName.** определяющее файл хранения данных, надо предварительно стереть этот файл, чтобы в набор занеслись новые данные. Для этого перед указанным выше оператором надо добавить оператор:

```
DeleteFile(ClientDataSet1->FileName);
```

Приведенные операторы только осуществляют запрос. А фильтрация проводится в обработчике события **OnDataRequest** провайдера:

```
OleVariant___fastcallTForm1::DataSetProvider1DataRequest(
                              TObject *Sender, OleVariant SInput)
{
 OleVariant NewData;
 DataSetProvider1->DataSet->Filtered = false;
 DataSetProviderl->DataSet->Filter = (AnsiString)Input;
 DataSetProviderl->DataSet->Filtered=true;
 NewData = DataSetProvider1->Data;
 DataSetProviderl->DataSet->Filtered=false;
 return NewData;
}
```

В этом обработчике вводится локальная переменная **NewData,** содержащая результаты запроса. Далее фильтрация базы данных отключается (свойство **DataSetProvider1->DataSet** — это набор, из которого черпаются данные). Фильтр заменяется заданным, фильтрация опять включается, и отфильтрованные данные передаются в переменную **NewData.** Затем фильтрация отключается, чтобы не мешать другим клиентам, которые, может быть, используют тот же провайдер и ту же базу данных. После этого отфильтрованные данные возвращаются как результат работы функции.

Рассмотренная выше фильтрация отличается от фильтрации, осуществляемой свойствами **Filter и Filtered** тем, что клиентский набор получает только данные,

удовлетворяющие фильтру. А свойства **Filter и Filtered** могут осуществлять только фильтрацию записей, уже имеющихся в наборе. Т.е. если применена описанная выше фильтрация, то **Filter и Filtered** смогут осуществить только дополнительную фильтрацию переданных в набор записей. Например, операторы

```
ClientDataSetl->Filtered = false;
ClientDataSetl->Filter = FilterEdit->Text;
ClientDataSetl->Filtered = true;
```

отберут из имеющихся записей те, которые удовлетворяют фильтру, записанному в окне **FilterEdit.**

Delete — метод TDataSet

Удаляет активную запись и позиционирует курсор на следующую запись.

Класс *TDataSet*

Объявление

```
void__fastcall Delete(void);
```

Описание

Метод **Delete** удаляет активную запись и позиционирует курсор на следующую запись. Если набор данных не активен, то генерируется исключение. При выполнении метода **Delete** осуществляются следующие операции:

- Проверяется, не является ли набор данных пустой. Если пустой, генерируется исключение
- Вызывается **CheckBrowseMode** для завершения обработки предыдущей записи
- Вызывается обработчик события **BeforeDelete**
- Запись удаляется
- Освобождаются буферы, связанные с данной записью
- Набор данных переводится в состояние State равное **dsBrowse**
- Курсор позиционируется на следующую неудаленную запись
- Вызывается обработчик события **AfterDelete**

Delete — метод списков и меню

Удаление элемента с указанным индексом из списка.

Классы *TList*, *TStringList*, *TStrings*, *TMenuItem*

Объявление

```
void__fastcall Delete(int Index);
```

Описание

Процедура **Delete** удаляет из списка элемент с указанным индексом. Во всех случаях индексы считаются, начиная с О (О — индекс первого элемента).

Из списка строк строка удаляется вместе со ссылкой на связанный с ней объект.

Удаление элемента меню ведет к удалению и связанного с ним подменю (если таковое имеется).

После удаления элемента список перестраивается. Это надо учитывать при удалении элементов в цикле, как указано в приведенном примере.

Пример

```
for (int i = 0; i < 2; i++)
  List->Delete(0);
```

В этом примере удаляются два первых элемента списка List. Неверно было бы написать:

```
for (int i = 0; i < 2; i++)
  List->Delete(i);
```

Так как после первого удаления список перестроится, то элементом с индексом 1 станет тот, который ранее имел индекс 2. Таким образом, в результате оказались бы удаленными элементы, имевшие в первоначальном списке индексы 0 и 2.

DisableAlign и другие методы выравнивания

Временно запрещают и разрешают выравнивание дочерних компонентов.

Класс *TWinControl*

Объявления

```
void__fastcall DisableAlign(void);
void__fastcall EnableAlign(void);
void__fastcall Realign(void);
```

Описание

Свойство **DisableAlign** временно запрещает выравнивание дочерних компонентов внутри оконного элемента. Метод применяется совместно с методом **EnableAlign,** отменяющим действие **DisableAlign** и разрешающим выравнивание.

Эти методы целесообразно использовать, если проводится перестроение ряда дочерних компонентов, например, при чтении их из файла формы, при масштабировании, при изменении взаимного расположения.

Каждому вызову **DisableAlign** должен соответствовать вызов **EnableAlign.** Очередной вызов **DisableAlign** увеличивает на единицу число запретов выравнивания, а вызов **EnableAlign** уменьшает это число. Как только при очередном вызове **EnableAlign** число запретов станет равным нулю, произойдет выравнивание. Это производится вызовом метода **Realign.**

Впрочем, надо иметь в виду, что при изменении свойства **Align** дочерних компонентов после выполнения метода **DisableAlign** некоторые изменения размеров компонентов все равно происходят, хотя выравнивание не производится.

Пример

```
Form1->DisableAlign();
  <операторы перестроения дочерних компонентов>
Form1->EnableAlign();;
```

DisableConstraints

См. раздел «ConstraintsDisabled, DisableConstraints, EnableConstraints».

DisableControls, EnableControls, ControlsDisabled

Методы запрещают и разрешают в компонентах, связанных с данными, отображение во время длинных операция перемещения по набору данных.

Класс *TDataSet*

Определения

```
void__fastcall  DisableControls(void);
void__fastcall  EnableControls(void);
bool__fastcall  ControlsDisabled(void);
```

Описание

При длительных итерациях по многим записям набора данных в компонентах, отображающих данные, могут появляться неприятные мерцания. К тому же, отображение данных при перемещениях к каждой новой записи замедляет работу приложения. Чтобы избежать этих неприятностей, полезно заблокировать отобра-

жение данных перед началом итераций. Это делает метод **DisableControls.** Разблокирует компоненты, отображающие данные, метод **EnableControls.** Каждый вызов **DisableControls** увеличивает число блокировок на 1. Поэтому, если блокировка была сделана несколькими вызовами **DisableControls,** то компоненты разблокируются только после такого же числа вызовов **EnableControls,** каждый из которых уменьшает число блокировок на 1. Функция **ControlsDisabled** позволяет определить наличие блокировки. Она возвращает **true,** если блокировка не снята. Применение **ControlsDisabled** в сочетании с **EnableControls** позволяет в цикле снять многократную блокировку.

Если набор данных является головным для другого вспомогательного набора данных, то метод **DisableControls** блокирует связь со вспомогательной таблицей. Для блокировки отображения вспомогательного набора данных лучше использовать свойство **BlockReadSize.**

Примеры

Следующий пример иллюстрирует блокировку связанных с набором данных компонентов отображения на время циклической обработки записей и разблокирование их после окончания цикла.

```
Table1->DisableControls();
try
{
  Table1->First();
  while (! Table1->Eof)
  {
   // Обработка очередной записи
   Table1->Next();
  }
}
__finally
{
 Table1->EnableControls();
}
```

В сложных приложениях не всегда можно определить, сколько раз была проведена блокировка. В этих случаях блокировка может быть снята следующим оператором:

```
while (Table1->ControlsDisabled())
 Table1->EnableControls();
```

Dormant

Создает изображение битовой матрицы в памяти, чтобы освободить дескриптор матрицы и сэкономить ресурсы.

Класс *TBitmap*

Объявление

```
void __fastcall Dormant(void);
```

Описание

Использование метода **Dormant** сокращает ресурсы GDI, используемые в приложении. Метод создает изображение матрицы в памяти, используя объект потока памяти. В дальнейшем через свойство **Handle** можно освободить связанный с матрицей HBITMAP.

Дополнительную экономию памяти можно получить методом **FreeImage,** освобождающим память, занятую кэшированием изображения. Но этот метод может приводить к потере глубины цвета.

Пример

```
void__fastcall TForm1::Button1Click(TObject *Sender)
{
 Graphics::TBitmap *BitMap1 = new Graphics::TBitmap();
 Graphics::TBitmap *BitMap2 = new Graphics::TBitmap();

try
{
// загрузка изображения из файла в BitMap1
 BitMap1->LoadFromFile("...");
// копирование в BitMap2 из BitMap1
 BitMap2->Assign(BitMap1);
// освобождение ресурсов GDI
 BitMap2->Dormant();
// освобождение памяти, изображение не теряется
 BitMap2->FreeImage();
// изображение из BitMap2 рисуется на канве
 Canvas->Draw(20,20,BitMap2);
// устанавливается монохромный режим BitMap2
// BitMap2->Monochrome = true;
// рисуется монохромный вариант
 Canvas->Draw(250,20,BitMap2);
// теперь изображение действительно теряется
 BitMap2->ReleaseHandle();
}
catch (...)
{
 MessageBeep(0);
}
 delete BitMap1;
 delete BitMap2;
}
```

Draw

Рисует графическое изображение в указанную позицию канвы.

Класс *TCanvas*

Объявление

```
void__fastcall Draw(int X, int Y, TGraphic* Graphic);
```

Описание

Метод **Draw** рисует изображение, содержащееся в объекте, указанном параметром **Graphic,** сохраняя исходный размер изображения в его источнике и перенося изображение в область канвы объекта **TCanvas.** Верхний левый угол этой области определяется параметрами X и Y. Источник изображения может быть битовой матрицей, пиктограммой или метафайлом. Если источник -- объект типа **TBitmap.** то перенос изображения производится в режиме, установленном свойством канвы **CopyMode.**

Пример

Оператор

```
Image1->Canvas->Draw(10,10,Bitmap1);
```

рисует на канве компонента **Image1** изображение из компонента **Bitmapl** в область с координатами левого верхнего угла (10, 10).

DrawFocusRect

Рисует изображение прямоугольника в виде, используемом для отображения рамки фокуса, операцией хог.

Класс *TCanvas*

Объявление

```
void _fastcall DrawFocusRect (const Windows: :TRect &Rect);;
```

Описание

Метод **DrawFocusRect** рисует на канве в области **Rect** изображение прямоугольника в виде, используемом обычно для отображения рамки фокуса, т.е. точками. При рисовании используется операция хог, что позволяет удалить изображение прямоугольника его повторной прорисовкой.

Пример

Следующая совокупность обработчиков событий, связанных с мышью, рисует на канве компонента **Imag1** прямоугольную рамку размером 10 на 10 вокруг курсора и перетаскивает ее при перемещении мыши с нажатой кнопкой:

```
int X0,Y0;
bool drag = false;

void _fastcall TForm1 : : Image1MouseDown (
                        TObject *Sender, TMouseButton Button,
                        TShiftState Shift, int X, int Y)
 (
// рисование рамки
 Image1->Canvas->DrawFocusRect(Rect(X-5,Y-5,X+5,Y+5));
// запоминание координат курсора
 XO = X;
 YO = Y;
// включение флажка режима перемещения рамки
 drag = true;
}
//---------------------------------------
void _fastcall TForm1 ::Image1MouseMove (
                          TObject *Sender, TShiftState Shift,
                          int X, int Y)
 (
 if( !drag) return;
// стирание рамки
 Image1->Canvas->DrawFocusRect(Rect(X0-5,Y0-5,XO+5,YO+5));
// рисование рамки
 Image1->Canvas->DrawFocusRect(Rect(X-5,Y-5,X+5,Y+5));
// запоминание координат курсора
 XO = X;
 YO = Y;
}
//,          ,                  ----------------------------
void _fastcall TForm1 ::Image1MouseUp (TObject *Sender,
                        TMouseButton Button, TShiftState Shift,
                        int X, int Y)
<
 if( !drag) return;
// стирание рамки
 Image1->Canvas->DrawFocusRect(
                    Rect(X0-5,Y0-5,X0+5,Y0+5));
// выключение флажка режима перемещения рамки
 drag = false;
}
```

При нажатии кнопки мыши рисуется первая рамка, запоминаются координаты курсора и включается режим перемещения рамки (переменная **drag = true**). При перемещении мыши в режиме перемещения рамки стирается прежняя рамка, рисуется рамка в новой позиции и запоминаются новые координаты курсора. При отпускании кнопки мыши стирается рамка и выключается режим перемещения рамки.

Edit

Переводит набор данных в режим редактирования.

Класс *TDataSet*

Объявление

```
void__fastcall Edit(void);
```

Описание

Метод **Edit** переводит набор данных в режим редактирования **(State = dsEdit)**. Предварительно проверяется набор данных и, если он пуст, то вызывается метод **Insert.** В противном случае выполняются операции:

- Вызывается **CheckBrowseMode** для завершения обработки предыдущей записи
- Вызывается обработчик события **BeforeEdit**
- Набор данных переводится в состояние **State** равное **dsEdit,** что позволяет редактировать поля текущей записи
- Вызывается обработчик события **AfterEdit**

EditKey

См. раздел «SetKey, EditKey, GotoKey, GotoNearest».

Ellipse

Рисует заполненную окружность или эллипс.

Класс *TCanvas*

Объявление

```
void__fastcall Ellipse(int X1, int Y1, int X2, int Y2);
```

Описание

Метод **Ellipse** рисует окружность или эллипс с помощью текущих параметров пера Реп. Фигура заполняется текущим значением **Brush.** Точки **(X1,** Y1) и (X2, Y2) определяют прямоугольник, описывающий эллипс.

В Windows 95 суммы **X1** + X2, Y1 + Y2 и **X1** + X2 + **Y1** + Y2 не должны превышать 32768.

Пример

Оператор

```
Image1->Canvas->Brush->Color = clRed;
Image1->Canvas->Brush->Style = bsDiagCross;
Image1->Canvas->Ellipse(0, 0, Image1->Width, Image1->Height);
```

рисует эллипс, вписанный в компонент Imagel и заполненный красной штриховкой.

EnableAlign

См. раздел «DisableAlign».

EnableConstraints

См. раздел «ConstraintsDisabled, DisableConstraints, EnableConstraints».

EnableControls

См. раздел «DisableControls, EnableControIs, ControlsDisabled».

EndUpdate

Снятие запрета перерисовки, введенного методом **BeginUpdate.**

Класс *TCollection*

Объявление

```
void __fastcall EndUpdate(void);
```

Описание

Метод снимает запрет перерисовки, введенный ранее методом **BeginUpdate.** Совместное применение и **EndUpdate** ускоряет реорганизацию перечня объектов собрания **TCollection** и предотвращается неприятное мерцание изображения в процессе изменения перечня объектов.

Пример

См. пример в разделе «BeginUpdate».

Exchange

Меняет позиции двух элементов списка.

Классы *TList*. *TStringList*. *TStrinss*

Объявления

Для TList:

```
void __fastcall Exchange(int Indexl, int Index2);
```

для TStrings и TStringList:

```
virtual void __fastcall Exchange (int Indexl, int Index2);
```

Описание

При вызове **Exchange** два элемента списка с позициями **Indexl и Index2** обмениваются местами. Индексы позиций начинаются с О (О — первый элемент).

Если в списках строк со строками связаны объекты, они остаются связанными с теми же строками в их новых позициях.

Не применяйте метод **Exchange** к сортированным спискам, за исключением случая взаимного перемещения двух одинаковых строк, связанных с разными объектами. Дело в том, что метод **Exchange** не проверяет условий сортировки и может нарушить упорядоченность списка.

Пример

Пусть список **List** типа **TList** содержит указатели на целые числа. Тогда приведенная ниже программа перемещает на первое место указатель на минимальное число (это один шаг пузырьковой сортировки).

```
for (int i=1; i < List->Count; i++)
  if (Mint *) (List->Items[0] ) > Mint *) (List->Items[i]) )
    List->Exchange(0, i);
```

См. также пример в гл. 1, в описании класса **TList.**

Expand

Увеличивает емкость списка типа **TList.**

Класс *TList*

Объявление

```
TList* _fastcall Expand(void);
```

Описание

Функция **Expand** увеличивает емкость списка типа **TList.** выделяя память для быстрого размещения новых элементов. Тем самым экономится время при добавлении в дальнейшем новых элементов списка.

Поскольку функция возвращает расширенный список, то ее можно также использовать для создания дубликата имеющегося списка.

Если список не заполнен, т.е. количество элементов **Count** меньше емкости списка **Capacity,** то список не расширяется. Если же **Count = Capacity,** то функция **Expand** увеличивает емкость **Capacity** согласно следующему алгоритму. Если значение **Capacity** меньше 4, то **Capacity** увеличивается на 4. Если значение **Capacity** больше 4, но меньше 9, то **Capacity** увеличивается на 8. Если значение **Capacity** больше 8, то **Capacity** увеличивается на 16.

Примеры

```
List->Expand();
```

Если список **List** заполнен не до конца, он остается неизменным. Если же он заполнен, то его емкость расширяется.

Следующий оператор создает объект **List1,** являющийся копией списка List:

```
TList * List1 = List->Expand();
```

Если исходный список был заполнен, то оба списка **List и List1** оказываются расширенными.

FetchAll

Считывает с сервера и сохраняет локально все записи, начиная с текущей.

Класс *TBDEDataSet*

Определение

```
void _fastcall FetchAll(void);
```

Описание

Метод **FetchAll** считывает с сервера и сохраняет локально все записи, начиная с текущей и до конца файла. Предварительно вызывается **CheckBrowseMode,** чтобы переслать в базу данных все оставшиеся незафиксированными изменения данных.

При обычном кэшировании транзакции читают столько данных, сколько необходимо для их отображения в приложении. При необходимости новых данных выполняется новая транзакция. Метод FetchAll позволяет сократить нагрузку на сеть, считывая все данные одной транзакцией. Но применение этого метода сопряжено с определенными проблемами. Если базу данных одновременно использует несколько клиентов, то есть опасность, что какие-то из прочитанных данных будут изменены другими приложениями. Тогда при возврате ранее прочитанных записей в базу данных могут возникнуть конфликты.

FieldByName, FindField

Методы ищут и возвращают поле по его имени.

Классы *TDataSet, TFields*

Определения

```
TField* _fastcall FieldByName(const AnsiString FieldName);
TField* _fastcall FindField(const AnsiString FieldName);
```

Описание

Методы **FieldByName** и **FindField** ищут поле, имя которого указано параметром **FieldName.** Если поле найдено, то возвращается объект **TField** этого поля. Если поле не найдено, то метод **FindField** возвращает **NULL,** а метод **FieldByName** генерирует исключение **EDatabaseError.**

Параметр **FieldName** может быть именем обычного поля, или именем дочернего поля, или именем поля совокупной характеристики. Благодаря подобной гибкости методы **FieldByName и FindField** часто предпочтительнее использования свойств **Fields или AggFields.**

Пример

Следующий пример показывает типичный способ организации цикла по записям набора данных. Пусть в вашем приложении имеется выпадающий список с именем **CBdep,** который вы хотите заполнить данными, содержащимися в полях Dep всех записей таблицы, соединенной с компонентом **Table1.** Это можно сделать следующим кодом:

```
CBdep->Clear();
Table1->First();
while (! Table1->Eof)
{
 CBdep->Items->Add(Table1->FieldByName("Dep")->AsString);
 Table1->Next();
}
```

Первый оператор кода очищает список CBdep. Второй — устанавливает курсор таблицы на первую запись. Далее следует цикл по всем записям, пока не достигнута последняя, что проверяется выражением **Table1->Eof.** Для каждой записи в список заносится значение поля Dep, после чего методом Next курсор перемещается к следующей записи.

Вместо метода **FieldByName** можно применить метод **FindField:**

```
CBdep->Items->Add(Table1->FindField("Dep")->AsString);
```

Отличие проявится только в случае, если поля Dep в наборе данных не окажется. Тогда метод **FieldByName** сгенерирует исключение, а метод **FindField** вернет **NULL.**

FillRect

Заполняет указанный прямоугольник канвы, используя текущее значение **Brush.**

Класс *TCanvas*

Объявление

```
void __fastcall FillRect(const Windows::TRect &Rect);
```

Описание

Метод **FillRect** заполняет прямоугольник канвы, указанный параметром **Rect,** используя текущее значение **Brush.** Заполняемая область включает верхнюю и левую стороны прямоугольника, но не включает правую и нижнюю стороны. При использовании **FillRect** параметр **Rect** часто задается функцией **Rect.**

Пример

Оператор

```
Image1->Canvas->FillRect(Rect(0, 0, Image1->Width,
                              Image1->Height));
```

очищает всю канву компонента **Image1,** заполняя ее фоном, если он установлен в свойстве **Brush.**

Find

Ищет в **Items** описание поля по его имени.

Класс *TFieldDefs*

Объявление

```
HIDESBASE TFieldDef*__fastcall Find(const AnsiString Name);
```

Описание

Метод **Find** ищет в Items описание поля по его имени Name. В результате поиска можно получить доступ к атрибутам указанного поля. Если требуемое описание не найдено, генерируется исключение. Чтобы избежать генерации исключения, можно перед вызовом **Find** удостовериться в наличии описания методом **IndexOf.** Если **IndexOf** вернет значение большее −1, значит описание поля существует и можно применять **Find.**

FindField

См. раздел «FieldByName, FindField».

FindFirst

См. раздел «Навигация по наборам данных».

FindKey, FindNearest

Обеспечивают поиск записей по ключу.

Классы *TCustomClientDataSet*, *TTable*

Объявления

```
bool__fastcall FindKey(const System::TVarRec * KeyValues,
                       const int KeyValues_Size);
void__fastcall  FindNearest(
                       const System::TVarRec * KeyValues,
                       const  int  KeyValues_Size);
```

Описание

Методы **FindKey и FindNearest** обеспечивают один из альтернативных (см. также **SetKey. Lookup** и **Locate)** способов поиска записи по ключу.

Перед началом поиска таблица Paradox или dBASE должна быть индексирована по тем полям, которые будут ключевыми для поиска и которые указаны в свойстве **IndexName.** В противном случае поиск можно проводить только по первичному индексу. Для таблиц SQL ключ может соответствовать индексу в свойстве **IndexName** или списку полей в свойстве **IndexFieldNames.**

В методы передается массив **KeyValues** значений ключевых полей в той последовательности, в которой они входят в индекс. Элементами массива могут быть строки, переменные или **NULL.** При этом не обязательно перечислять все поля — достаточно перечислить первое или несколько первых. Для остальных полей будет предполагаться значение **NULL.** Параметр **KeyValues_Size** определяет индекс последнего элемента в массиве **KeyValues,** т.е. он на 1 меньше числа элементов массива.

Если найдена запись, в которой значения ключевых полей совпадают с заданными ключами, то функция **FindKey** перемещает курсор на эту запись и возвращает **true.** Если же такая запись не найдена, то курсор останется на месте, а **FindKey** вернет **false.** Вместо **FindKey** часто удобнее использовать метод **FindNearest,** особенно для полей строкового типа. Метод обеспечивает переход к наиболее совпадающей записи, если полного совпадения не получено. Это позволяет проводить, например, ускоренный поиск записей со строковыми полями при посимвольным

вводе искомого значения, как показано в примере). Значение свойства **KeyExclusive** определяет, на какую именно запись переместится курсор при частичном совпадением значений полей и ключей.

Примеры

В приведенных ниже примерах предполагается поиск в таблице записи о сотруднике, работающем в отделе (поле **Dep),** название которого указано пользователем в окне **Edit1**. Фамилия сотрудника (поле **Fam)** указана пользователем в окне **Edit2**.

Следующий код обеспечивает переход к найденной записи или сообщение пользователю об отсутствии записи:

```
Table1->IndexFieldNames = "Dep;Fam";
if ( ! Table1->FindKey(OPENARRAY(TVarRec,
                                 (Edit1->Text,Edit2->Text))))
  ShowMessage("Запись не найдена");
```

Если приведенный ниже код вставить в обработчик события **OnChange** окна **Edit2,** то будет обеспечен ускоренный поиск: по мере ввода символов в Edit2 курсор набора данных будет перемещаться к записи, в которой начало фамилии наиболее близко к вводимым символам.

```
Table1->IndexFieldNames = "Dep;Fam";
Table1->FindNearest(OPENARRAY(TVarRec,  (Edit1->Text,Edit2->Text)));
```

FindLast

См. раздел «Навигация по наборам данных».

FindNearest

См. раздел «FindKey, FindNearest».

FindNext

См. раздел «Навигация по наборам данных».

FindNextControl

Возвращает следующий в последовательности табуляции оконный дочерний компонент.

Класс *TWinControl*

Объявление

```
TWinControl*__fastcall FindNextControl(
                                TWinControl* CurControl, bool GoForward,
                                bool CheckTabStop, bool CheckParent);
```

Описание

Метод **FindNextControl** находит и возвращает следующий за указанным в параметре **CurControl** дочерний оконный компонент в соответствии с последовательностью табуляции. Если **CurControl** не является дочерним компонентом данного оконного элемента, то возвращается компонент, первый в последовательности табуляции. То же самое происходит, если **CurControl** является последним компонентом в последовательности табуляции.

Параметр **GoForward** определяет направление поиска. Если он равен **true,** то поиск проводится вперед и возвращается компонент, следующий за **CurControl.** Если же параметр **GoForward** равен **false,** то возвращается предшествующий компонент.

Параметры **CheckTabStop и CheckParent** определяют условия поиска. Если **CheckTabStop** равен **true,** то просматриваются только компоненты, в которых

свойство **TabStop** установлено в **true. При CheckTabStop** равном **false** значение **TabStop** не принимается во внимание. Если параметр **CheckParent** равен **true,** то просматриваются только компоненты, в свойстве **Parent** которых указан данный оконный элемент, т.е. просматриваются только прямые потомки. Если **Check-Parent** равен **false,** то просматриваются все, даже косвенные потомки данного элемента.

Метод **FindNextControl** вызывает метод **GetTabOrderList** и из полученного таким способом списка черпает последовательность компонентов.

Примеры

```
TWinControl * obj;
.....
obj = Form1->FindNextControl(obj,true, true, true);
ShowMessage(obj->Name);
```

В этом примере переменная **obj** поочередно принимает значение всех компонентов, размещенных непосредственно на форме **Form1** и включенных в последовательность табуляции, т.е. имеющих свойство **TabStop,** равное **true.** Например, в эту последовательность войдут окна редактирования, кнопки, панели, расположенные непосредственно на форме и имеющие **TabStop = true,** но не войдут кнопки и окна редактирования, расположенные на панелях.

Если в приведенном операторе изменить параметр **CheckParent** на **false:**

```
obj = Form1->FindNextControl(obj,true, true, false);
```

то в последовательность войдут и непрямые наследники, имеющие **TabStop = true,** в частности, компоненты, содержащиеся в панелях, расположенных на форме, причем независимо от значения **TabStop** этих панелей.

Если в приведенном операторе изменить параметр **CheckTabStop на false:**

```
obj = Form1->FindNextControl(obj,true, false, false);
```

то в последовательность войдут компоненты, независимо от значения их свойства **TabStop.**

FindPrior

См. раздел «Навигация по наборам данных».

First

См. раздел «Навигация по наборам данных».

FloodFill

Закрашивает текущей кистью замкнутую область канвы, определенную указанным цветом.

Класс *TCanvas*

Объявление

```
enum TFillStyle {fsSurface, fsBorder};
void__fastcall FloodFill(int X, int Y, TColor Color,
                         TFillStyle FillStyle);
```

Описание

Метод **FloodFill** закрашивает текущей кистью **Brush** замкнутую область канвы, определенную цветом и начальной точкой закрашивания (X, Y). Точка с координатами X и Y является произвольной внутренней точкой заполняемой области, которая может иметь произвольную форму. Граница этой области определяется сочетанием параметров **Color и FillStyle.** Параметр **Color** типа **TColor** указывает цвет, который используется при определении границы закрашиваемой области,

а параметр **FillStyle** определяет, как именно по этому цвету определяется граница. Если **FillStyle = fsSurface,** то заполняется область, окрашенная цветом **Color,** а на других цветах метод останавливается. Если **FillStyle „= fsBorder,** то наоборот, заполняется область окрашенная любыми цветами, не равными **Color,** а на цвете **Color** метод останавливается.

Примеры

```
1.  Image1->Canvas->Brush->Color = clWhite;
    Image1->Canvas->FloodFill(X,Y, Image1->Canvas->Pixels[X][Y],
                              fsSurface);
```

Приведенные операторы закрашивают белым цветом на канве компонента **Imagel** все пикселы, прилегающие к пикселу с координатами (X, Y) и имеющие тот же цвет, что и этот пиксел.

```
2.  Image1->Canvas->Brush->Color = clWhite;
    Image1->Canvas->FloodFill(X, Y, clBlack, fsBorder);
```

Приведенные операторы закрашивают белым цветом на канве компонента **Imagel** все пикселы, прилегающие к пикселу с координатами (X, Y) и имеющие цвет, отличный от черного. При достижении черной границы области закраска останавливается.

FlushBuffers

Пересылает в базу данных все изменения, сохраненные в буфере.

Класс *TBDEDataSet*

Объявление

```
void __fastcall FlushBuffers(void);
```

Описание

Метод **FlushBuffers** пересылает в базу данных все оставшиеся изменения набора данных, включая кэшированные изменения, сохраненные в буфере. Метод можно использовать вместо **CheckBrowseMode,** если важно, чтобы было послано в базу данных содержимое буферов записей.

Focused

Определяет, находится ли оконный элемент в фокусе.

Класс *TWinControl*

Объявление

```
DYNAMIC bool __fastcall Focused(void);
```

Описание

Метод **Focused** определяет, является ли оконный элемент активным, т.е. находится ли он в фокусе. Возвращает **true,** если элемент находится в фокусе, и **false** — если элемент не в фокусе и пользователь в данный момент не может с ним взаимодействовать.

FrameRect

Рисует на канве текущей кистью прямоугольную рамку.

Класс *TCanvas*

Объявление

```
void __fastcall FrameRect(const Windows::TRect &Rect);
```

Описание

Метод **FrameRect** рисует на канве прямоугольную рамку вокруг области **Rect,** используя установку текущей кисти **Brush.** Толщина рамки — 1 пиксел. Область внутри рамки кистью не заполняется. Отличается от метода **Rectangle** тем, что рамка рисуется цветом кисти (в методе **Rectangle** — цветом пера Реп) и область не закрашивается (в методе **Rectangle** закрашивается).

Пример

Операторы

```
Image1->Canvas->Brush->Color = clBlack;
Image1->Canvas->FrameRect(Rect(10,10,100,100));
```

рисуют на канве компонента **Imagel** черную рамку.

FreeBookmark

См. в разделе «Bookmark Valid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark».

get — функция-элемент ifstream

Вводит символы из входного потока.

Класс *ifstream*

Объявления

```
char get();
bool get(char);
void get(char *, int n, char delim);
```

Описание

Метод **get** представляет собой функции-элементы класса входного потока **ifstream.** Он вводит символы из файла, связанного с потоком. Метод имеет три приведенные выше модификации.

Первая модификация

Функция get без аргументов вводит одиночный символ из указанного потока (даже, если это символ разделитель) и возвращает этот символ в качестве значения вызова функции. Этот вариант функции get возвращает EOF, когда в потоке встречается признак конца файла.

Следующий код использует функцию get без аргумента, чтобы построчно читать и обрабатывать весь текст файла:

```
char s[80], c;
ifstream infile("Test.dat");
if(!infile)
{
 ShowMessage("Файл не удается открыть");
 return;
}
int i = 0;
while((c = infile.get()) != EOF)
{
 if(c == '\n')
 {
  // занесение нулевого символа в конец строки
  s[i] = 0;
  // обработка строки
  ...
  i = 0;
 }
 // формирование строки
```

```
 else s[i++] = c;
}
// закрытие файла
 infile.close();
```

Здесь символы файла поочередно читаются в символьную переменную c. Если прочитанный символ не является символом перевода строки "\n", то символ добавляется в строку s. Если же символ равен "\n", то в конец строки заносится нулевой символ, строка подвергается какой-то обработке, после чего начинает формироваться следующая строка. Отметим, что этот код имеет один недостаток: если символу конца файла не предшествует символ перевода строки, то последняя строка оказывается без завершающего нулевого символа и остается необработанной. Нетрудно придумать дополнение кода, которое ликвидировало бы этот недостаток.

Функцию **get()** удобно использовать для поиска в файле какого-то ключевого символа. Например, цикл поиска в файле символа "$" можно организовать следующим образом:

```
while((c = infile.get()) != EOF)
  {
  if(c == '$') break;
  if(c == '$') ...
  }
```

Вторая модификация

Второй вариант функции-элемента get с символьным аргументом вводит очередной символ из входного потока (даже, если этот символ разделитель) и сохраняет его в символьном аргументе. Этот вариант функции **get** возвращает ложь, когда встречается признак конца файла; в остальных случаях этот вариант функции **get** возвращает ссылку на тот объект потока, для которого вызывалась функция-элемент get.

При использовании этого варианта функции get приведенные ранее примеры можно оставить практически без изменений, переписав только заголовки структур **while:**

```
while(infile.get(c))
```

Третья модификация

Третий вариант функции-элемента get принимает три параметра: символьный массив s, максимальное число символов n и ограничитель **delim** (по умолчанию символ перевода строки "\n"). Этот вариант читает символы из входного потока до тех пор, пока не достигается число символов, на 1 меньше указанного максимального числа n, или пока не считывается ограничитель. Затем для завершения введенной строки в символьный массив, используемый в качестве буфера программы, помещается нулевой символ. Ограничитель в символьный массив не помещается, а остается во входном потоке (он будет следующим считываемым символом). Таким образом, результатом второго подряд использования функции **get** явится пустая строка, если только ограничитель не удалить из входного потока.

Приведенный ранее пример чтения всего файла по строкам в данном случае реализуется проще:

```
char s[80];
 ifstream infile("Test.dat");
 if(!infile)
 {
  ShowMessage("Файл не удается открыть");
  return;
 }
while(!infile.eof())
 {
  infile.get(s,80);
  infile.get();
```

```
    // обработка строки
    ...
  }
// закрытие файла
 infile.close();
```

В данном случае третий аргумент в вызове get не указан. Значит подразумевается по умолчанию ограничитель "\n" и каждый вызов get читает одну строку (подразумевается, что ее длина не более 80 символов). Обратите внимание на то, что после оператора

```
infile.get(s,80);
```

добавлен оператор

```
infile.get();
```

Этот оператор удаляет из потока ограничитель. Если этого не сделать, программа зациклится.

Функция get с тремя параметрами не всегда удобна, поскольку оставляет ограничитель в потоке, и для повторного вызова функции его приходится убирать отдельным оператором. Часто более удобна другая функция — **getline.**

GetBookmark

См. в разделе «BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark».

GetData

Возвращает неформатированное («сырое») значение поля.

Класс *TField*

Объявление

```
bool __fastcall GetData(void * Buffer, bool NativeFormat);
```

Описание

Метод **GetData** возвращает неформатированное («сырое») значение поля в буфер **Buffer.** В отличие от свойств **DisplayText. Text. AsString** и т.п., метод **GetData** не осуществляет преобразование данных к какому-то типу. Буфер **Buffer** должен быть достаточного размера. Для определения этого размера можно использовать свойство **DataSize.**

Параметр **NativeFormat** указывает, не должны ли «сырые» данные транслироваться в тип поля набора данных: если **NativeFormat = true** — не должны, если **false** — должны.

Если значение поля пустое (NULL), то **GetData** возвращает **false** и в буфер ничего не заносится. В остальных случаях возвращается **true.**

Метод **GetData** не может применяться к полям BLOB и Memo.

GetDetailDataSets

Заполняет список вспомогательных наборов данных, управляемых головным набором.

Класс *TDataSet*

Объявление

```
void __fastcall GetDetailDataSets(Classes::TList* List);
```

Описание

Метод **GetDetailDataSets** применяется в головном наборе данных типа master/detail — головной/вспомогательный (детализирующий). Метод заполняет спи-

сок List типа **TList** всех вспомогательных наборов данных, подключенных к головному. Список **List** должен быть создан до обращения к **GetDetailDataSets.** Метод **GetDetailDataSets** только заполняет его указателями на объекты **TDataSet.** Пользуясь заполненным списком **List,** можно получить доступ и ко всем полям этих вспомогательных наборов данных. Речь идет только о связях типа master/detail. Для доступа ко вложенным наборам данных надо использовать свойство **NestedDataScts.**

Пример

Пусть в вашем приложении имеется головной набор данных **Tablel** и несколько вспомогательных (детализирующих) наборов данных, один из которых — **Table2.** Введите глобальные переменные:

```
TList * MasterFields = new TList;
TList * DetailFields = new TList;
```

Тогда следующий код с помощью метода **GetDetailDataSets,** примененного к головной таблице **Tablel,** позволяет отобразить список всех вспомогательных (детализирующих) таблиц и списки всех их полей:

```
Table1->GetDetailDataSets(DetailFields);
info = "Таблицы детализации\n\n";
for (int i=0; i < DetailFields->Count; i++)
{
 info = "поля таблицы " +
        ((TDataSet*)DetailFields->Items[i]) ->Name + '\n';
 for (int j=0;
      j < ((TDataSet*)DetailFields->Items[i]) ->FieldCount;
      j++)
   info += ((TDataSet *)DetailFields->Items[i])->
                  Fields->Fields[j]->FieldName + '\n';
}
ShowMessage(info);
```

В результате выполнения этого кода отобразится окно с сообщением вида:

```
Таблицы детализации

поля таблицы Table3
Dep
Fam
Nam
...

поля таблицы Table2
...
```

GetDetailLinkFields

Заполняет список ключевых полей, связывающих вспомогательный набор данных с головным.

Класс *TDataSet*

Объявление

```
virtual void__fastcall GetDetailLinkFields(
                              Classes::TList* MasterFields,
                              Classes::TList* DetailFields);
```

Описание

Метод **GetDetailLinkFields** применяется во вспомогательном наборе данных типа master/detail — головной/вспомогательный (детализирующий). Метод позволяет получить доступ к ключевым полям, совпадение значений которых обеспечивает связь головного и вспомогательного наборов. Списки, передаваемые в метод

как параметры **MasterFields и DetailFields** (должны быть созданы до вызова метода), заполняются объектами **TField** ключевых полей соответственно головного и вспомогательного наборов.

Пример

Пусть в вашем приложении имеется головной набор данных **Tablel** и несколько вспомогательных (детализирующих) наборов данных, один из которых — **Table2.** Введите глобальные переменные:

```
TList * MasterFields = new TList;
TList * DetailFields = new TList;
```

Тогда следующий код позволяет отобразить список ключевых полей головной и детализирующей таблиц (метод **GetDetailLinkFields** применяется ко вспомогательной таблице **Table2):**

```
Table2->GetDetailLinkFields(MasterFields, DetailFields);
String info = "Ключевые поля таблиц"
              "\nголовной\tвспомогательной\n\n";
for (int i=0; i < MasterFields->Count; i++)
  info +=
     ((TField *)MasterFields->Items[i])->FieldName + "\t\t" +
     ((TField*)DetailFields->Items[i])->FieldName + '\n';
ShowMessage(info);
```

В результате выполнения этого кода отобразится окно с сообщением вида:

```
Ключевые поля таблиц
головной вспомогательной

Dep        Dep
...        ...
```

GetFieldNames — метод TFields и TDataSet

Возвращает список имен всех полей объекта **TFields.**

Классы *TFields, TDataSet*

Определение

```
void__fastcall GetFieldNames(Classes::TStrings * List);
```

Описание

В классе **TFields** метод **GetFieldNames** заполняет список List типа TStrings именами всех полей объекта **TFields.**

В классе **TDataSet** метод **GetFieldNames** заполняет аналогичный список **List** именами всех полей набора данных, кроме полей совокупных характеристик. Тип списка **TStrings** удобен, для ввода его в такие компоненты, как **TListBox, TComboBox, TMemo и др.**

См. также метод **GetItemNames.**

Пример

Каждый из приведенных ниже операторов заполняет выпадающий список **ComboBoxl** именами полей таблицы, соединенной с компонентом **Tablel:**

```
Table1->GetFieldNames(ComboBox1->Items);
Table1->Fields->GetFieldNames(ComboBox1->Items);
```

GetGroupState

Определяет положение текущей записи в указанной группе.

Класс *TCustomClientDataSet*

Объявления

```
enum TGroupPosInd { gbFirst, gbMiddle, gbLast };
typedef Set<TGroupPosInd, gbFirst, gbLast> TGroupPosInds;

virtual TGroupPosInds__fastcall GetGroupState(int Level);
```

Описание

Метод **GetGroupState** позволяет определить положение текущей записи в группе записей, указанной параметром **Level.** Этот параметр указывает уровень группы в текущем индексе. Уровень 1 — это множество записей с одинаковым значением первого поля, указанного в индексе, уровень 2 — то же самое по отношению ко второму полю индекса и т.д. Уровень 0 — это множество всех записей. Если значение **Level** превышает значение свойства **GroupingLevel**, генерируется исключение.

Функция **GetGroupState** возвращает множество значений, пустое, или содержащее элементы:

[gbFirst]	Текущая запись первая, но не единственная в группе
[gbLast]	Текущая запись последняя, но не единственная в группе
[gbMiddle]	Текущая запись не первая и не последняя в группе
[gbFirst, gbLast]	Текущая запись единственная в группе

Если свойство **AggregatesActive** равно **false** или текущий индекс не поддерживает группирования, функция **GetGroupState** возвращает пустое множество.

GetItemNames

Возвращает имена всех описаний в объекте **TDefCollection.**

Класс *TDefCollection*

Объявление

```
void__fastcall GetItemNames(Classes::TStrings* List);
```

Описание

Метод **GetItemNames** возвращает в виде списка строк **List** типа **TStrings** имена всех описаний в объекте **TDefCollection.** При вызове метода параметр **List** может быть пустым списком строк. Этот список в результате вызова **GetItemNames** заполнится именами всех описаний.

См. также метод **GetFieldNames.**

Пример

Следующий оператор заполняет выпадающий список ComboBoxl именами полей таблицы, соединенной с компонентом **Tablel:**

```
Table1->FieldDefs->GetItemNames(ComboBox1->Items);
```

getline — функция-элемент ifstream

Вводит строку символов из потока.

Класс *ifstream*

Объявления

```
void getline(char *s, int n) ;
void getline(char *s, int n, char delim);
```

Описание

Метод **getline** представляет собой функцию-элемент класса входного потока **ifstream**. Он вводит строку символов из файла, связанного с потоком.

Функции **getline** принимает три параметра: символьный массив s, максимальное число символов n и ограничитель **delim** (по умолчанию символ перевода строки "\n"). Функция читает символы из входного потока до тех пор, пока не достигается число символов, на 1 меньше указанного максимального числа n, или пока не считывается ограничитель. Затем для завершения введенной строки в символьный массив, используемый в качестве буфера программы, помещается нулевой символ. Ограничитель в символьный массив не помещается и удаляется из входного потока. В этом основное отличие функции **getline** от варианта функции **get**, читающего строки символов. Функция get оставляет разделитель во входном потоке и его приходится удалять из него, чтобы прочитать следующую строку. Так что в этом отношении функция **getline** удобнее.

Пример

Ниже приведен пример чтения файла по строкам с помощью функции **getline.** Сравнив его с аналогичным примером, приведенным в описании функции get, вы можете увидеть преимущества функции **getline.**

```
char s[80], c;
ifstream infile("Test.dat");
if(!infile)
{
 ShowMessage("Файл не удается открыть");
 return;
}
while(!infile.eof())
 {
  infile.getline(s,80);
  // обработка строки
  if (s[0] != 0) Label1->Caption = s;
 }
// закрытие файла
 infile.close();
```

GetTabOrderList

Строит список дочерних оконных компонентов в последовательности табуляции.

Класс *TWinControl*

Объявление

```
DYNAMIC void __fastcall GetTabOrderList(Classes::TList* List);
```

Описание

Метод **GetTabOrderList** строит список типа TList дочерних оконных компонентов в последовательности табуляции. В список входят не только непосредственные потомки данного элемента, но и косвенные потомки, включенные в дочерние контейнеры. При этом не обращается внимание на значение свойства **TabStop** включаемых компонентов.

Метод **GetTabOrderList** вызывается методом **FindNextControl**, определяющим последующий или предшествующий компонент в списке табуляции.

Пример

Приведенные ниже операторы обеспечивают поочередный доступ ко всем дочерним оконным компонентам в последовательности табуляции.

```
TWinControl * obj;
...
```

```
TList * List = new TList;
GetTabOrderList(List);
for (int i=0; i < List->Count; i++)
{
 //Поочередный доступ к объектам
 obj = (TWinControl *) (List->Items[i]) ;
 ShowMessage(obj->Name);
}
```

GotoBookmark

См. в разделе «BookmarkValid, CompareBookmarks, GetBookmark, GotoBookmark, FreeBookmark».

GotoKey

См. раздел «SetKey, EditKey, GotoKey, GotoNearest».

GotoNearest

См. раздел «SetKey, EditKey, GotoKey, GotoNearest».

HandleAllocated

Проверяет наличие дескриптора окна компонента.

Класс *TWinControl*

Объявление

```
bool __fastcall HandleAllocated(void);
```

Описание

Метод **HandleAllocated** используется для определения, имеется ли дескриптор окна у данного элемента. Если дескриптор имеется, то возвращается **true.**

Непосредственная проверка свойства **Handle** приводит к тому, что даже если дескриптора не было, он создается. Применение **HandleAllocated** позволяет определить наличие дескриптора без этого побочного эффекта.

HandleNeeded

Создает дескриптор окна, если до этого он не существовал.

Класс *TWinControl*

Объявление

```
void __fastcall HandleNeeded(void);
```

Описание

Метод **HandleNeeded** создает дескриптор окна, если до этого его не было. При создании дескриптора он прежде всего вызывает метод **CreateHandle** родительского элемента, а уже затем создает дескриптор данного элемента.

Hide

Делает компонент невидимым.

Класс *TControl*

Объявление

```
void __fastcall Hide(void);
```

Описание

Метод **Hide** делает компонент невидимым, задавая значение **false** его свойству **Visible.** Если компонент является контейнером для других компонентов, то эти дочерние компоненты также делаются невидимыми.

Хотя компонент становится невидимым, его свойства и методы остаются доступными.

IndexOf — метод TDefCollection

Определение первого вхождения в список заданного элемента.

Класс *TDefCollection*

Объявление

```
int__fastcall IndexOf(const AnsiString AName);
```

Описание

Метод **IndexOf** возвращает индекс описания в массиве **Items** по его имени **AName.** Индексы начинаются с 0. Если описания с именем **AName** нет, то возвращается **-1.** Это можно использовать для определения наличия описания перед применением метода **Find.**

IndexOf — метод TFields

Возвращает номер индекса объекта **TField** в списке **TFields.**

Класс *TFields*

Объявление

```
int__fastcall IndexOf(TField* Field);
```

Описание

Метод **IndexOf** возвращает номер индекса объекта **Field** типа **TField** в списке **TFields.** Индексы начинаются с 0. Если поле в списке не найдено, возвращается **-1.**

IndexOf — метод списков

Определение первого вхождения в список заданного элемента.

Классы *TList. TStringList. TStrings*

Объявления

Для TList:

```
int__fastcall IndexOf(void * Item);
```

для TStrings и TStringList:

```
virtual int__fastcall IndexOf(const AnsiString S) ;
```

Описание

Вызов **IndexOf** возвращает индекс первого вхождения в массив списка заданного элемента (указателя **Item для TList** или строки S для **TStringList и TStrings).** Индексация начинается с О (О — первый элемент массива). Если заданного элемента в списке нет, возвращается **-1.**

Пример

В приведенном ниже примере определяется, есть ли в списке сотрудник, фамилия которого задана пользователем в окне **Edit1.**

```
TStringList *LPerson = new TStringList();
...
if  (LPerson->IndexOf(Edit1->Text)<0)
   ShowMessage("Сотрудника " + Edit1->Text + " в списке нет");
```

Insert — метод TCollection*

Создает новый объект и вставляет его в массив **Items.**

Класс *TCollection*

Определение

```
TCollectionItem*__fastcall Insert(int Index);
```

Описание

Метод **Insert** создает новый объект **TCollectionItem** и вставляет его в массив Items в позицию, указанную параметром **Index.** Индексы прежних объектов, начиная с **Index,** увеличиваются на 1.

Insert — метод TDataSet

Вставляют новую пустую запись в набор данных.

Класс *TDataSet*

Объявление

```
HIDESBASE void__fastcall Insert(void);
```

Описание

Метод **Insert** вставляет новую пустую запись в набор данных и делает ее активной. После этого можно предоставить пользователю возможность заполнить поля этой записи и затем занести запись в базу данных методом **Post** (или методом **ApplyUpdates**, если осуществляется кэширование). Место размещения новой записи в наборе определяется следующим образом:

- Для таблиц Paradox с первичным индексом запись вставляется в позицию, соответствующую индексу
- Для таблиц Paradox без первичного индекса запись вставляется в текущую позицию курсора индексу
- Для таблиц dBASE, FoxPro и Access запись вставляется в конец набора данных. Если в данный момент активен индекс, то отображение вставляемой записи может появиться в позиции, соответствующей индексу, но физически запись все равно вставляется в конец таблицы
- Для баз данных SQL место расположения новой записи зависит от реализации базы данных. Для индексированных таблиц индекс обновляется с учетом новой записи

См. также методы **InsertRecord, Append и AppendRecord.**

Пример

Следующие операторы создают новую запись, заполняют ее поля **Fam, Nam, Par** (Фамилия, Имя, Отчество) значениями, введенными в окна редактирования **Edit1, Edit2,** Edit3 и пересылают запись в базу данных.

```
Table1->Insert();
Table1->FieldValues["Fam"] = Edit1->Text;
Table1->FieldValues["Nam"] = Edit2->Text;
Table1->FieldValues["Par"] = Edit3->Text;
Table1->Post();
```

В приведенном примере метод **Insert** можно было бы заменить методом **Append.** Никакой разницы в результатах не было бы, кроме, возможно, несколько иного размещения новой записи в базе данных.

Insert — метод списков

Процедура вставляет элемент в список в заданную позицию.

Классы *TList, TStringList, TStrings*

Определения

Для **TList:**

```
void__fastcall Insert(int Index, void * Item);
```

для **TStrings:**

```
virtual void__fastcall Insert(int Index, const AnsiString S) = 0;
```

для **TStringList:**

```
virtual void__fastcall Insert(int Index, const AnsiString S);
```

Описание

Процедура **Insert** вставляет элемент (указатель **Item** или строку S) в список в позицию, индекс которой задан параметром **Index.** Если задан **Index** = 0, элемент вставляется в первую позицию. При вставке элемента все имеющиеся в списке элементы с индексами, равными и большими **Index,** сдвигаются, т.е. их индексы увеличиваются на 1.

Если список сортирован, то вызов **Insert** приводит к генерации исключения **EListError.** Для сортированных списков следует использовать метод **Add.**

Если в списки **TStringList и TStrings** надо вставить строку, связанную с объектом, то вместо метода **Insert** следует использовать метод **InsertObject.**

InsertRecord

Вставляют новую запись в набор данных и заполняет ее поля.

Класс *TDataSet*

Объявление

```
void__fastcall InsertRecord(const System::TVarRec * Values,
                            const int Values_Size);
```

Описание

Метод **InsertRecord** вставляет в набор данных новую запись, заполняет ее поля значениями, перечисленными в параметре **Values и** пересылает в базу данных. Новая запись делается активной. В массиве **Values** заносимые значения перечисляются в той последовательности, в которой расположены поля таблицы. На месте значений тех полей, в которые данные не заносятся, пишется **NULL.** После записи последнего ненулевого значения список **Values** можно прервать, т.е. не указывать в нем **NULL** для остальных не заполняемых полей.

Метод **InsertRecord** не удается применить, если какое-то поле, предшествующее последнему вводимому значению, является немодифицируемым, например, автоматически нарастающим **(AutoIncrement).** Поскольку в такое поле невозможно ввести значение, а оно должно присутствовать в массиве **Values,** то при применении **InsertRecord** будет генерироваться исключение,

Место размещения новой записи в базе данных определяется следующими правилами:

- Для индексированных таблиц Paradox и dBASE запись вставляется в позицию, соответствующую индексу
- Для неиндексированных таблиц Paradox запись вставляется в текущую позицию курсора
- Для неиндексированных таблиц dBASE, FoxPro и Access запись вставляется в конец набора данных
- Для баз данных SQL место расположения новой записи зависит от реализации базы данных. Для индексированных таблиц индекс обновляется с учетом новой записи

Имеется еще метода внесения новой записи в набор данных **AppendRecord**. Он отличается от рассмотренных только позицией размещения записи в физической базе данных.

См. методы **Insert. Append и AppendRecord**.

Пример

Следующие операторы создают новую запись, заполняют ее поля **Fam, Nam, Par** (Фамилия, Имя, Отчество) значениями, введенными в окна редактирования **Edit1, Edit2, Edit3**. В первое поле (номер записи) заносится целое число, во второе поле (отдел работы сотрудника) не заносится ничего. В поля, следующие за полями фамилии, имени и отчества также ничего не заносится.

```
Table1->InsertRecord(ARRAYOFCONST(
          (18, NULL, Edit1->Text, Edit2->Text, Edit3->Text)));
Table1->Post();
```

Приведенный выше оператор **InsertRecord** нельзя было бы применить, если, например, первое поле (номер записи) является автоматически нарастающим **(AutoIncrement)**, т.е. не модифицируемым.

В приведенном примере метод **InsertRecord** можно было бы заменить методом **AppendRecord.** Никакой разницы в результатах не было бы, кроме, возможно, несколько иного размещения новой записи в базе данных.

Invalidate

Сообщает Windows о необходимости полностью перерисовать компонент после того, как будут обработаны другие важные сообщения Windows.

Класс *TControl*

Объявление

```
virtual void __fastcall Invalidate(void);
```

Описание

Метод **Invalidate** надо вызывать, когда требуется полностью перерисовать компонент. Если более одной области компонента требует перерисовки, вызов метода **Invalidate** приводит к его полной перерисовке, что позволяет избежать мерцания изображения. Многократный вызов **Invalidate** до действительной перерисовки компонента не приводит к потере эффективности работы.

IsValidChar

Проверяет, является ли указанный символ допустимым для данного поля.

Класс *TField*

Объявление

```
virtual bool __fastcall IsValidChar(char InputChar);
```

Описание

Метод **IsValidChar** проверяет, является ли указанный символ **InputChar** допустимым для данного поля. Метод **IsValidChar** возвращает **true,** если **InputChar** — допустимый символ, и **false** — если недопустимый.

Множество допустимых для поля символов задается в свойстве **ValidChars.** По умолчанию это свойство различно для различных типов **DataType** полей:

ftBoolean	все символы
ftSmallInt	цифры от 0 до 9, знаки "+" и "–"
ftWord	цифры от 0 до 9, знаки "+" и "–"

ftAutoInc	цифры от 0 до 9, знаки "+" и "–"
ftDate	все символы
ftInteger	цифры от 0 до 9, знаки "+" и "–"
ftTime	все символы
ftCurrency	цифры от 0 до 9, знаки "+" и "–", буква "E" или "e", десятичный разделитель (точка или запятая в зависимости от установки Windows)
ftDateTime	все символы
ftFloat	цифры от 0 до 9, знаки "+" и "–", буква "E" или "e", десятичный разделитель (точка или запятая в зависимости от установки Windows)
ftBCD	цифры от 0 до 9, знаки "+" и "–", десятичный разделитель (точка или запятая в зависимости от установки Windows)
ftString	все символы
ftVarBytes	все символы
ftBytes	все символы
ftBlob	все символы
ftDBaseOle	все символы
ftFmtMemo	все символы
ftGraphic	все символы
ftMemo	все символы
ftParadoxOle	все символы
ftTypedBinary	все символы
ftUnknown	все символы
ftCursor	все символы

В некоторых типах полей (например, **ftGraphic**) **IsValidChar** возвращает **true** для любых символов, хотя не все они допустимы.

Метод **IsValidChar** неявно используется многими компонентами, связанными с данными. Явное использование **IsValidChar** см. в примере раздела «ValidChars» в гл. 3.

Метод **IsValidChar** проверяет допустимости символа независимо от его позиции во вводимом значении. Проверку с учетом позиции символа может осуществлять свойство **EditMask**.

Last

См. раздел «Навигация по наборам данных».

LineTo

Рисует на канве прямую линию, начинающуюся с текущей позиции пера и кончающуюся указанной точкой.

Класс *TCanvas*

Объявление

```
void__fastcall LineTo(int X, int Y);
```

Описание

Метод **LineTo** рисует на канве прямую линию, начинающуюся с текущей позиции пера **PenPos** и кончающуюся точкой (X, Y), исключая саму точку (X, Y). Текущая позиция пера **PenPos** перемещается в точку (X, Y). При рисовании используются текущие установки пера Реп.

Пример

Операторы

```
Image1->Canvas->MoveTo(X1,Y1);
Image1->Canvas->LineTo(X2,Y2);
Image1->Canvas->LineTo(X3,Y3);
```

рисуют кусочно-ломаную прямую, соединяющую точки (X1,Y1), (X2,Y2) и (X3,Y3).

LoadFromClipboardFormat

Загружает изображение из буфера обмена в формате Clipboard.

Класс *TGraphic*

Объявление

```
virtual void__fastcall LoadFromClipboardFormat(
                        Word AFormat, int AData, HPALETTE APalette);
```

Описание

Метод загружает изображение в графический объект в указанном формате Clipboard. Если формат **AFormat** найден среди зарегистрированных, то **AData и APalette** передаются для загрузки изображения. Стандартно зарегистрированные форматы: **CF_BITMAP** для битовых карт и **CF_METAFILEPICT** для метафайлов. Значение **AData** может быть указано методом **GetAsHandle** объекта типа **TClipBoard.** При этом надо не забыть включить в модуль директиву:

```
#include <vcl\Clipbrd.hpp>
```

Формат для нового типа графического объекта предварительно должен быть зарегистрирован методом **RegisterClipboardFormat.**

Если в буфере обмена находится не тот тип данных, который ожидается, то генерируется исключение **EInvalidGraphic.**

Пример

Операторы

```
#include <vcl\Clipbrd.hpp>
...
if (Clipboard()->HasFormat(CF_BITMAP))
{
 try
 {
  Image1->Picture->Bitmap->LoadFromClipboardFormat(
    CF_BITMAP, Clipboard()->GetAsHandle(CF_BITMAP), 0);
 }
 catch (...)
 {
  ShowMessage("Загрузка изображения невозможна");
 }
}
else
  ShowMessage("В буфере не изображение в формате битовой карты");
```

загружают изображение из буфера обмена в формате битовой карты в компонент **Image1.**

LoadFromFile — метод TGraphic

Загружает изображение, хранящееся в файле.

Класс *TGraphic*

Объявление

```
virtual void__fastcall LoadFromFile(const AnsiString FileName);
```

Описание

Метод **LoadFromFile** читает файл **FileName** и загружает его в графический объект.

Если формат графического файла не зарегистрирован, или не соответствует типу графического объекта, то генерируется исключение **EInvalidGraphic.**

Пример

```
if(OpenPictureDialog1->Execute())
  Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
```

Этот оператор открывает диалог **OpenPictureDialog1,** позволяющий пользователю выбрать файл, и загружает изображение из файла в компонент **Image1.**

LoadFromFile и другие методы загрузки и сохранения данных класса TCustomClientDataSet

Обеспечивают загрузку и сохранение данных в файле или потоке.

Класс *TCustomClientDataSet*

Объявления

```
void__fastcall LoadFromFile(const AnsiString FileName = "");
void__fastcall LoadFromStream(Classes::TStream* Stream);

enum TDataPacketFormat {dfBinary, dfXML, dfXMLUTF8}
void__fastcall SaveToFile(const AnsiString FileName = "",
                          TDataPacketFormat Format = dfBinary);
void__fastcall SaveToStream(Classes::TStream* Stream,
                          TDataPacketFormat Format = dfBinary);
```

Описание

Методы **LoadFromFile и LoadFromStream** загружают данные из файла **FileName** или потока **Stream** в клиентский набор данных. Если в вызове **LoadFromFile** имя файла не задано, например:

```
ClientDataSet1->LoadFromFile();
```

то делается попытка открыть файл, указанный свойством **FileName** набора данных. Если и в этом свойстве имя файла не указано, генерируется исключение.

Файл, из которого загружаются данные методом **LoadFromFile,** должен быть создан заранее методом **SaveToFile** данного или другого клиентского набора данных, или создан компонентом **TXMLTransform.**

Если в свойстве **FileName** компонента указано имя файла и этот файл существует, то данные из этого файла загружаются автоматически при открытии набора данных. Так что метод **LoadFromFile** надо применять только в том случае, если требуется прочитать данные из другого файла.

Метод **LoadFromFile** может вызываться неявно и в процессе проектирования. Для этого надо щелкнуть на компоненте правой кнопкой мыши и выбрать из контекстного меню раздел Load from File.

Поток, из которого читаются данные методом **LoadFromStream,** должен содержать данные, загруженные в него методом **SaveToStream.**

Методы **SaveToFile** и **SaveToStream** сохраняют данные клиентского набора данных в файле **FileName** или потоке **Stream.** Параметр **TDataPacketFormat** оп-

ределяет формат записи данных: **dfBinary** — двоичный, **dfXML** — XML с расширенным набором символов, включая escape-последовательности, **dfXMLUTF8** – XML с расширенным набором символов, представленный с помощью UTF8.

При записи в файл, если этот файл не существует, он создается, а если существует — его данные стираются и заменяются новыми.

Если при вызове **SaveToFile** имя файла не указывается, запись производится в файл, указанный свойством **FileName** набора данных. Если и в этом свойстве имя файла не указано, генерируется исключение.

Если в свойстве **FileName** компонента указано имя файла и этот файл существует, то данные в этот файл записываются автоматически при закрытии набора данных. Так что метод **SaveToFile** надо применять только в том случае, если требуется записать данные в другой файл.

При вызове **SaveToStream** поток Stream должен существовать. Если этот поток требует буфер, то для определения необходимого размера буфера можно использовать свойство **DataSize.**

LoadFromResourceID

Загружает битовую карту из файла ресурсов по указанному идентификатору.

Класс *TBitmap*

Объявление

```
void__fastcall LoadFromResourceID(int Instance, int ResID);
```

Описание

Метод **LoadFromResoureelD** загружает битовую карту из файла ресурсов выполняемого модуля. Загружаемая карта указывается идентификатором **ResID.**

LoadFromResourceName

Загружает битовую карту из файла ресурсов по указанному имени.

Класс *TBitmap*

Объявление

```
void__fastcall LoadFromResourceName(int Instance,
                                    const AnsiString ResName);
```

Описание

Метод **LoadFromResourceName** загружает битовую карту из файла ресурсов выполняемого модуля. Загружаемая карта указывается именем **ResName.**

LoadFromStream — метод TGraphic

Загружает графическое изображение из указанного потока.

Класс *TGraphic*

Объявление

```
virtual void__fastcall LoadFromStream(Classes::TStream* Stream);
```

Описание

Метод **LoadFromStream** читает из потока **Stream** графический объект. Используется, например, для загрузки изображения из объекта **TBlobStream,** чтобы прочитать графическое поле в наборе данных.

LoadFromStream — метод TCustomClientDataSet

См. разд. «LoadFromFile и другие методы загрузки и сохранения данных класса TCustomClientDataSet».

Locate

Осуществляет поиск записи в наборе данных.

Класс *TDataSet*

Объявление

```
enum TLocateOption { loCaseInsensitive, loPartialKey };
typedef Set<TLocateOption, loCaseInsensitive, loPartialKey>
                                                TLocateOptions;

virtual bool __fastcall Locate(const AnsiString KeyFields,
                               const System::Variant &KeyValues,
                               TLocateOptions Options);
```

Описание

Метод **Locate** является, наряду с **Lookup. SetKey и FindKey.** методом поиска записи в наборе данных.

В качестве первого параметра **KeyFields** передается строка, содержащая список ключевых полей, по которым осуществляется поиск. В качестве второго параметра передается **KeyValues** — массив ключевых значений. Ищется запись, в которой значения полей **KeyFields** совпадают с заданными в **KeyValues.** Третий параметр **Options** является множеством опций, элементами которого могут быть **loCaseInsensitive** — нечувствительность поиска к регистру, в котором введены символы, и **loPartialKey** — допустимость частичного совпадения. Метод возвращает **false,** если искомая запись не найдена.

Метод **Locate** отличается от методов **SetKey** и **FindKey** прежде всего отсутствием необходимости индексировать определенным образом набор данных. В силу этого **Locate** более прост и универсален. Кроме того метод **Locate,** в отличие от **SetKey** и **FindKey.** применим к компонентам Microsoft ActiveX Data Objects (ADO) — например, **ADOTable и ADOQuery,** а также к компонентам InterBase Express (IBX) — например, **IBTable и IBQuery.**

Метод **Locate** не требует индексации набора данных. Но если набор данных индексирован по полям **KeyFields,** то поиск производится быстрее.

При поиске по нескольким полям можно воспользоваться функцией **VarArrayOf,** которая формирует тип **Variant** из задаваемого ей массива параметров любого типа.

Примеры

Пусть в вашем приложении имеется набор данных **Table1,** содержащий список сотрудников, в котором поле Fam содержит фамилию сотрудника, а поле Dep — наименование отдела, в котором он работает. Приложение имеет также окно редактирования EFam, в котором пользователь вводит символы фамилии. Требуется осуществить ускоренный поиск записи по мере ввода пользователем фамилии искомого сотрудника.

При использовании метода **Locate** достаточно в обработчик события **OnChange** окна **EFam** вставить оператор

```
TLocateOptions SearchOptions;
SearchOptions << loPartialKey << loCaseInsensitive;
Table1->Locate("Fam",EFam->Text, SearchOptions);
```

По мере ввода символов в **EFam,** курсор будет перемещаться на запись, в которой первые символы значения поля Fam совпадают с введенными символами.

Приведенный код можно сократить до двух операторов:

```
TLocateOptions SearchOptions;
Table1->Locate("Fam", EFam->Text,
               SearchOptions<<loPartialKey<<loCaseInsensitive);
```

Если требуется найти не просто сотрудника с задаваемой фамилией, а сотрудника отдела, наименование которого введено в окно редактирования **EDep,** операторы надо изменить на следующие:

```
TLocateOptions SearchOptions;
Variant locvalues[] = {EDep->Text, EFam->Text};
Table1->Locate("Dep;Fam", VarArrayOf(locvalues,1),
               SearchOptions<<loPartialKey<<loCaseInsensitive);
```

В этом примере использована функция **VarArrayOf,** которая формирует тип **Variant** из задаваемого ей массива параметров любого типа.

Lock

Блокирует канву, не разрешая другим нитям многопоточного приложения рисовать на ней.

Класс *TCanvas*

Объявление

```
void __fastcall Lock(void);
```

Описание

Метод **Lock** блокирует данную канву, не разрешая другим нитям многопоточного приложения рисовать на ней. Канва остается блокированной до снятия блокады вызовом метода **Unlock.** Если имеются вложенные вызовы **Lock,** то они увеличивают свойство **LockCount,** фиксирующее количество блокировок. Канва будет оставаться блокированной, пока не будет снята последняя блокировка.

Если нежелательна вложенная многократная блокировка, лучше использовать метод **TryLock.**

Поскольку блокировка не дает другим нитям рисовать на канве, производительность работы приложения может за счет этого снизится. Так что не надо злоупотреблять блокировками. Их следует применять только тогда, когда есть вероятность нежелательных наложений операций, выполняемых в разных нитях приложения с несколькими потоками.

Lookup

Осуществляет поиск записи в наборе данных и возвращает значения указанных полей этой записи.

Класс *TDataSet*

Объявление

```
virtual System::Variant __fastcall Lookup(
                                    const AnsiStrinq KeyFields,
                                    const Variant &KeyValues,
                                    const AnsiString ResultFields);
```

Описание

Метод **Lookup** относится к методам поиска записи в наборе данных. Но в отличие от других методов поиска — **Locate. SetKey и FindKey,** метод **Lookup** не перемещает курсор на найденную запись, а возвращает значения указанных полей этой записи.

В качестве первого параметра **KeyFields** в метод передается строка, содержащая список ключевых полей, по которым осуществляется поиск. В качестве второго параметра передается **KeyValues** — массив ключевых значений. Ищется запись, в которой значения полей **KeyFields** совпадают с заданными в **KeyValues.** Третий параметр **ResultFields** — строка, перечисляющая имена полей, значения которых возвращаются. Имена полей в **ResultFields** разделяются точками с запятой.

Метод возвращает значение или значения полей, перечисленных в **ResultFields,** в виде значения **Variant** или массива **Variant.**

Возвращенные методом **Lookup** значения полей могут, в частности, использоваться вместо параметра **Key Values** в другом операторе **Lookup** или **Locate.** Это открывает широкие возможности формирования сложных запросов по нескольким таблицам.

Пример

Пусть вы хотите найти в наборе данных **Table1** запись, относящуюся к сотруднику, фамилия которого (поле **Гат)** указана в окне **ЕГат,** и вывести в окно **EDep** название отдела (поле **Dep),** в котором он работает. Эти операции можно осуществить следующим оператором:

```
EDep->Text = Table1->Lookup("Fam",EFam->Text,"Dep");
```

MessageBox

Метод, отображающий полностью русифицированное диалоговое окно сообщения.

Модуль *Forms*

Объявление

```
function MessageBox(Text, Caption: PChar; Flags: Longint): Integer;
```

Описание

Функция **MessageBox** является методом переменной **Application** типа **TApplication,** доступной в любом проекте C++Builder. Он позволяет устранить основной недостаток других функций и процедур отображения диалоговых окон, таких, как **ShowMessage. ShowMessageFmt. MessageDlg. MessageDIgPos. CreateMessageDialog.** Этим недостатком является отсутствие русификации диалоговых окон: английские надписи на кнопках и невозможность указать русский текст заголовка окна (кроме функции **CreateMessageDialog).**

Функция **MessageBox** отображает диалоговое окно с заданными кнопками, сообщением и заголовком и позволяет проанализировать ответ пользователя. Функция инкапсулирует функцию **MessageBox** API Windows. Параметр **Text** представляет собой текст сообщения, которое может превышать 255 символов. Для длинных сообщений осуществляется автоматический перенос текста. Параметр **Caption** представляет собой текст заголовка окна. Он тоже может превышать 255 символов, но не переносится. Так что длинный заголовок приводит к появлению длинного и не очень красивого диалогового окна.

Параметр **Flags** представляет собой множество флагов, определяющих вид и поведение диалогового окна. Этот параметр может комбинироваться операцией сложения по одному флагу из следующих групп.

Флаги кнопок, отображаемых в диалоговом окне

Флаг	Значение (в скобках даны надписи в русифицированных версиях Windows)
MB_ABORTRETRYIGNORE	Кнопки Abort (Стоп), Retry (Повтор) и Ignore (Пропустить).
MB_OK	Кнопка OK. Этот флаг принят по умолчанию.
MB_OKCANCEL	Кнопки OK и Cancel (Отмена).
MB_RETRYCANCEL	Кнопки Retry (Повтор) и Cancel (Отмена).
MB_YESNO	Кнопки Yes (Да) и No (Нет).
MB_YESNOCANCEL	Кнопки Yes (Да), No (Нет) и Cancel (Отмена).

Флаги пиктограмм в диалоговом окне

Флаг	Пиктограмма
MB_ICONEXCLAMATION, MB_ICONWARNING	Восклицательный знак (замечание, предупреждение).
MB_ICONINFORMATION, MB_ICONASTERISK	Буква "i" в круге (подтверждение).
MB_ICONQUESTION	Знак вопроса (ожидание ответа).
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	Знак креста на красном круге (запрет, ошибка).

Флаги, указывающие кнопку по умолчанию (которая в первый момент находится в фокусе)

Флаг	Кнопка
MB_DEFBUTTON1	Первая кнопка. Это принято по умолчанию.
MB_DEFBUTTON2	Вторая кнопка.
MB_DEFBUTTON3	Третья кнопка.
MB_DEFBUTTON4	Четвертая кнопка.

Флаги модальности

Флаг	**Пояснение**
MB_APPLMODAL	Пользователь должен ответить на запрос, прежде чем сможет продолжить работу с приложением. Но он может перейти в окна другого приложения. Он может также работать со всплывающими окнами данного приложения. Этот флаг принят по умолчанию.
MB_SYSTEMMODAL	То же самое, что **MB_APPLMODAL,** но окно диалога отображается в стиле WS_EX_TOPMOST, то есть всегда остается поверх других окон, даже если пользователь перешел к другим приложениям. Используется для предупреждения о серьезных ошибках, требующих немедленного вмешательства.

Некоторые дополнительные флаги (могут задаваться оба флага)

Флаг	Пояснение
MB_HELP	Добавляет в окно кнопку Help (Справка), щелчок на которой или нажатие клавиши F1 генерирует событие Help.
MB_TOPMOST	Помещает окно всегда сверху (в стиле WS_EX_TOPMOST).

Возможны еще некоторые флаги, определяющие характер поведения окна при работе в сети нескольких пользователей, позволяющие отображать тексты справа налево (для восточных языков) и т.п.

Функция возвращает нуль, если не хватает памяти для создания диалогового окна. Если же функция выполнена успешно, то возвращаемая величина свидетельствует о следующем:

Значение	Численное значение	Пояснение
IDABORT	3	Выбрана кнопка Abort (Стоп).
IDCANCEL	2	Выбрана кнопка Cancel (Отмена) или нажата клавиша Esc.
IDIGNORE	5	Выбрана кнопка Ignore (Пропустить).
IDNO	7	Выбрана кнопка No (Нет).
IDOK	1	Выбрана кнопка OK.
IDRETRY	4	Выбрана кнопка Retry (Повтор).
IDYES	6	Выбрана кнопка Yes (Да).

Пример

Ниже приведен текст, предусматривающий проверку правильности ввода данных перед пересылкой записи в базу данных.

```
if (проверка введенных данных)
{
  if (Application->MessageBox(
              "Хотите занести текущую запись в базу данных?",
              "Подтвердите занесение в базу данных",
              MB_YESNOCANCEL + MB_ICONQUESTION) != IDYES)
    {
      DataSet->Cancel();
      Abort();
    }
}
else
{
   Application->MessageBox("Ошибочные данные","Ошибка",
                           MB_ICONSTOP);
   Abort();
}
```

Отображаемые этим кодом окна приведены на рис. 4.2. Безусловно, они более удачны за счет русификации, чем аналогичные окна, приведенные в описаниях функций MessageDlg и **MessageDlgPos**.

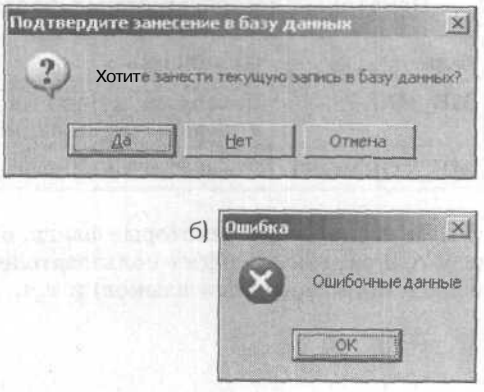


Рис. 4.2
Диалоговые окна, отображаемые функцией Application->MessageBox

Move

Меняет текущую позицию элемента в списке на заданную.

Модуль *System*

Классы *TList, TStrings*

Объявление
Для **TList:**

```
void __fastcall Move(int CurIndex, int NewIndex);
```

Для TStrings:

```
virtual void __fastcall Move(int CurIndex, int NewIndex);
```

Описание
Процедура **Move** меняет текущую позицию элемента в списке, индекс которого задан параметром **CurIndex,** на позицию с индексом, заданным параметром **NewIndex** (индексы начинаются с 0). Если со строкой в **TStrings или в TStringList** связан объект, он остается связанным с той же строкой в новой позиции.

Пример
Приведенный ниже оператор перемещает первую строку списка **MyStrings** в конец списка.

```
MyStrings->Move(0, MyStrings->Count - 1);
```

MoveBy

См. раздел «Навигация по наборам данных».

MoveTo

Изменяет текущую позицию пера на заданную, ничего при этом не рисуя.

Класс *TCanvas*

Объявление

```
void __fastcall MoveTo(int X, int Y);
```

Описание
Метод **MoveTo** изменяет текущую позицию пера **PenPos** на заданную точкой (X, Y). Это эквивалентно непосредственной установке свойства **PenPos.** При перемещении пера методом **MoveTo** ничего не рисуется.

Next

См. раздел «Навигация по наборам данных».

Open

Метод открывает набор данных.

Класс *TDataSet*

Объявление

```
void __fastcall Open(void);
```

Описание
Функция **Open** открывает набор данных, устанавливая свойство **Active в true.** Фактически, вызов метода **Open** и установка **Active в true** эквивалентны. Подробнее см. в разделе «Active» гл. 3.

OpenDatabase

Открывает соединение с базой данных.

Класс *TDBDataSet*

Объявление

```
TDatabase*__fastcall  OpenDatabase(void);
```

Описание

Метод **OpenDatabase** открывает соединение с базой данных через постоянный или временный компонент базы данных **Database.** Свойство **DatabaseName** набора данных определяет открываемую базу данных.

При вызове **OpenDatabase** прежде всего активизируется компонент текущего сеанса сетевого соединения Session. Затем определяется, соответствует ли свойство **DatabaseName** набора данных свойству **DatabaseName** одного из существующих компонентов **Database.** Если не соответствует, то создается временный компонент **Database** с именем, соответствующим свойству **DatabaseName.** В заключение выполняется метод **Open** компонента **Database** и число его ссылок увеличивается на 1. Это число ссылок учитывается впоследствии при применении метода закрытия соединения **CloseDatabase.**

Perform

Посылает оконному компоненту указанное сообщение Windows.

Класс *TControl*

Объявление

```
int__fastcall Perform(Cardinal Msg, int WParam, int LParam);
```

Параметры

Msg — идентификатор сообщения, **WParam и LParam** — параметры сообщения.

Описание

Метод **Perform** посылает сообщение тому оконному компоненту, к которому он применен. При этом **Perform** заполняет поля структуры типа TMessage значениями параметров Msg, **WParam, LParam,** и задает нулевое значение полю результата. Затем эта структура передается на обработку функции, указанной в компоненте свойством **WindowProc.** Таким образом, сообщение пересылается непосредственно окну, метод **Perform** которого используется. Например, оператор

```
Form2->Perform(WM_CLOSE,0,0);
```

передает сообщение **WM_CLOSE** форме **Form2,** закрывая окно формы.

Pie

Рисует заполненную замкнутую фигуру — сегмент окружности или эллипса.

Класс *TCanvas*

Объявление

```
void__fastcall Pie(int X1, int Y1, int X2, int Y2,
                   int X3, int Y3, int X4, int Y4);
```

Описание

Метод **Pie** рисует замкнутую фигуру — сектор окружности или эллипса с помощью текущих параметров пера Реп. Фигура заполняется текущим значением **Brush.** Точки **(X1,** Y1) и **(X2,** Y2) определяют прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (X3, Y3). Конечная точка дуги определяется пересе-

чением эллипса с прямой, проходящей через его центр и точку (X4, Y4). Дуга рисуется против часовой стрелки от начальной до конечной точки. Рисуются прямые, ограничивающие сегмент и проходящие через центр эллипса и точки (X3, Y3) и (X4, Y4).

В Windows 95 суммы X1 + X2, Y1 + Y2 и X1 + X2 + Y1 + Y2 не должны превышать 32768.

В Windows NT/2000/XP направление рисования дуги можно изменить на направление по часовой стрелке вызовом функции **SetArcDirection.**

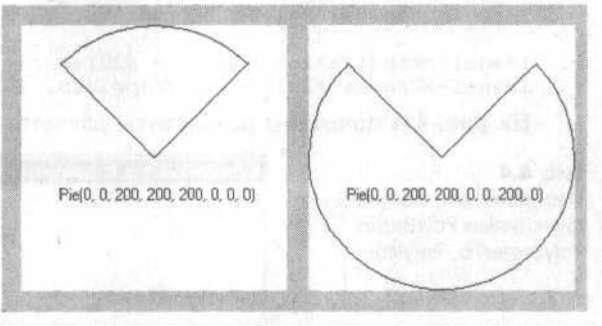
Примеры

Операторы

```
Image1->Canvas->Pie(0, 0, 200, 200, 200, O, O, 0);
Image2->Canvas->Pie(0, 0, 200, 200, O, O, 200, 0);
```

дают результат, показанный на рис. 4.3.

Рис. 4.3
Дуги, нарисованные функцией Pie



PolyBezier и PolyBezierTo

Рисуют на канве текущим пером кусочную кривую третьего порядка, сглаживающую заданное множество точек.

Класс *TCanvas*

Объявление

```
void__fastcall PolyBezier(const TPoint* Points,
                          const int Points_Size);
void__fastcall PolyBezierTo(const TPoint * Points,
                            const int Points_Size);
```

Описание

Методы **PolyBezier и PolyBezierTo** сглаживают множество точек, содержащихся в массиве **Points,** кусочной кривой третьего порядка. При этом функция **PolyBezier** точно отображает первую и последнюю точку, а **PolyBezierTo** — только последнюю. Число точек для каждого метода должно быть строго определенным: для **PolyBezierTo** оно должно быть кратно 3 (т.е. i*3), а для **PolyBezier** — на единицу больше числа, кратного 3 (т.е. i*3+l). Если число точек не равно требуемому, то функции просто ничего не рисуют.

Исходя из этого при произвольном числе точек N имеет смысл автоматически приводить число точек к требуемому, например, такими операторами:

```
PolyBezier(points, (N/3) *3) ;
PolyBezierTo(points, (N/3)*3-1);
```

В этих операторах число точек N за счет округления при целочисленном делении автоматически приводится к требуемому.

Пример

Ниже приведен код построения аппроксимаций функции -sin(x) методами **PolyBezier, PolyBezierTo и Polyline.**

```
const N = 10, Lx = 500, Ly = 100, T = 10;
TPoint points[N];

// заполнение массива
for(int i.= 0; i <= N; i++)
   points[i] = Point ( (int) (i * Lx / (N-1)),
                       (int) (sin ( (double) i * T / (N-1))*Ly) +
                      Image1->ClientHeight / 2);
// рисование
Image1->Canvas->Pen->Color = clBlack;
Image1->Canvas->Polyline(points, N-1) ;

Image1->Canvas->Pen->Color = clRed;
Image1->Canvas->PolyBezier(points, (N/3)*3);

Image1->Canvas->Pen->Color = clGreen;
Image1->Canvas->PolyBezierTo(points, (N/3)*3-1);
```

На рис. 4.4 показаны результаты расчета.

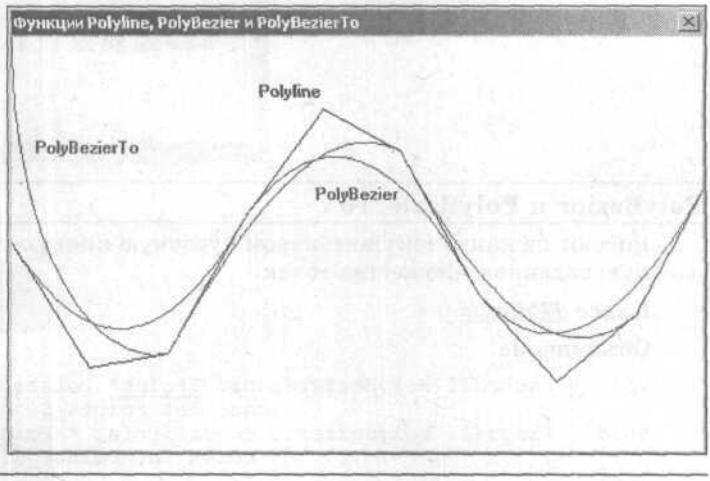


Рис. 4.4 Изображения, полученные функциями PolyBezier, PolyBezierTo, Polyline

Polygon

Рисует на канве текущим пером замкнутую фигуру (многоугольник) по заданному множеству угловых точек, замыкая первую и последнюю точки и закрашивая внутреннюю область фигуры текущей кистью.

Класс *TCanvas*

Объявление

```
void__fastcall Polygon(const Windows::TPoint * Points,
                       const int Points_Size);
```

Описание

Метод **Polygon** рисует на канве замкнутую фигуру (полигон, многоугольник) по множеству угловых точек, заданному массивом **Points.** Первая из указанных точек соединяется прямой с последней. Этим метод **Polygon** отличается от метода **Polyline**, который не замыкает конечные точки. Рисование проводится текущим пером Реп. Внутренняя область фигуры закрашивается текущей кистью **Brush.**

Метод позволяет рисовать фигуру по точкам, хранящимся в массиве элементов типа **TPoint**.

Пример
Операторы

```
TPoint points[5];
points[0] = Point(30,150);
points[1] = Point(40,130);
points[2] = Point(50,140);
points[3] = Point(60,130);
points[4] = Point(70,150);
Image1->Canvas->Polygon(points,4);
```

рисуют на канве формы многоугольник по точкам, хранящимся в массиве **points.**

Polyline

Рисует на канве текущим пером кусочно-линейную кривую по заданному множеству точек.

Класс *TCanvas*

Объявление

```
void __fastcall Polyline(const Windows::TPoint * Points,
                         const int Points_Size);
```

Описание

Метод **Polyline** рисует на канве кусочно-линейную кривую по множеству точек, заданному массивом **Points.** Отличие метод **Polyline** от метода **Polygon** заключается в том, что метод **Polygon** замыкает конечные точки, а метод **Polyline** — нет. Рисование проводится текущим пером Реп. Метод не изменяет текущей позиции PenPos пера Реп.

Метод позволяет рисовать кусочно-линейный график функции, хранящийся в массиве элементов типа **TPoint.**

То, что делает метод **Polyline,** можно сделать и с помощью методов **MoveTo и LineTo.** подведя сначала перо к первой точке, а затем последовательно выполняя **LineTo.** Различие будет заключаться в том, что метод **Polyline** не изменит текущую позицию пера, а методы **MoveTo и LineTo** изменят.

Пример
Операторы

```
TPoint points[5];
points[0] = Point(30,150);
points[1] = Point(40,130);
points[2] = Point(50,140);
points[3] = Point(60,130);
points[4] = Point(70,150);
Image1->Canvas->Polyline(points,4);
```

рисуют кусочно-линейную кривую по четырем точкам, заданным функциями **Point** в массиве **points.**

См. также пример в описании функций **PolvBezicr и PolyBezierTo.**

Post

Метод заносит измененную запись в базу данных.

Классы *TDataSet*, *TCustomClientDataSet*

Объявление

```
virtual void __fastcall Post(void);
```

Описание

В наборах данных, построенных на компонентах TTable, TQuery и других наследниках класса **TDataSet.** метод **Post** заносит запись, измененную методами **Edit. Insert. Append,** в базу данных. Метод **Post** может в ряде случаев вызываться неявно: как последствие вызова методов навигации **First. Last. Next. Prior.** Поэтому прежде, чем перемещаться на новую запись, полезно проверять свойство **Modified,** чтобы установить, была ли изменена текущая запись. Для проверки правильности измененных данных и получения от пользователя подтверждения записи в базу данных можно использовать обработчик события **BeforePost.**

В компонентах — наследниках **TCustomClientDataSet** выполнение **Post** зависит от значения свойства **LogChanges.** Если **LogChanges = true,** изменения заносятся в свойство **Delta** и затем могут объединяться с данными набора вызовом метода **MergeChangeLog.** А если **LogChanges = false,** вызов **Post** заносит изменения непосредственно в свойство **Data.**

Пример

Следующий оператор запрашивает пользователя о занесении результатов редактирования в базу данных. При положительном ответе занесение производится методом **Post,** при ином ответе результаты редактирования отменяются методом **Cancel.**

```
if (Application->MessageBox(
                        "Хотите занести текущую запись в базу данных?",
                        "Подтвердите сохранение изменений",
                        MB_YESNOCANCEL+MB_ICONQUESTION) != IDYES)
 Tablel->Cancel();
else Tablel->Post();
```

Prior

См. раздел «Навигация по наборам данных».

Realign

См. раздел «DisableAlign».

Reconcile — метод TCustomClientDataSet

См. разд. «ApplyUpdates, Reconcile — методы TCustomClientDataSet».

Rectangle

Рисует на канве текущим пером прямоугольник и закрашивает его текущей кистью.

Класс *TCanvas*

Объявление

```
void__fastcall Rectangle(int X1, int Y1, int X2, int Y2);
```

Описание

Метод **Rectangle** рисует на канве текущим пером Реп прямоугольник, верхний левый угол которого имеет координаты (X1, Y1), а нижний правый — (X2, Y2). Прямоугольник закрашивается текущей кистью **Brush.**

Рисование прямоугольника без рамки можно осуществить методом **FillRect.** Прямоугольник со скругленными углами рисуется методом **RoundRect.** Прямоугольник без внутренней закраски рисуется методом **FrameRect.**

Пример

```
Image1->Canvas->Rectangle(10,10,210,110);
```

Refresh — метод TControl

Перерисовывает изображение компонента на экране.

Класс *TControl*

Объявление

```
void __fastcall Refresh(void);
```

Описание

Метод **Refresh** приводит к немедленной перерисовке изображения на экране. Refresh вызывает метод **Repaint.** Методы **Refresh** и **Repaint** взаимозаменяемы.

Refresh, RefreshRecord — методы наборов данных

Обновляют в наборе данные, беря их из базы данных.

Классы *TDataSet*, *TCustomClientDataSet*

Объявления

```
TDataSet::void __fastcall Refresh(void);
TCustomClientDataSet::void __fastcall RefreshRecord(void);
```

Описание

Метод **Refresh** обновляет набор данных, приводя его в соответствие с текущим состоянием базы данных. Это часто необходимо делать, например, при смене фильтра или отказе от фильтрации, чтобы обеспечить отображение нового множества записей. Не все наборы данных полностью поддерживают метод **Refresh.** Например, в компонентах **TQuery** метод срабатывает только в том случае, если работа идет с «живыми» данными (см. свойство **RequestLive).** В противном случае, для обновления данных надо закрыть и открыть соответствующий набор данных.

Выполнение метода **Refresh** сопровождается событиями **BeforeRefresh и AfterRefresh**, наступающими до и после обновления.

При выполнении метода **Refresh** делается попытка не изменять положение курсора, указывающего на текущую запись. Но это не всегда возможно. Например, при обновлении может оказаться, что данная запись уже удалена из базы данных другим пользователем. Невозможно сохранить текущую позицию также в однонаправленных наборах данных. В них после обновления курсор всегда переходит на первую запись. Вообще надо учесть, что для однонаправленных наборов данных выполнение **Refresh** сводится к закрытию и открытию курсора. Это связано с генерацией событий **BeforeClose. AfterClose. BeforeOpen, AfterOpen.** Так что надо учитывать, что при этом будут срабатывать коды обработчиков этих событий.

Если метод **Refresh** применяется к клиентскому набору данных, то обновляется все содержимое набора. Если оказывается, что в наборе хранятся еще не зафиксированные в базе данных результаты редактирования, то генерируется исключение.

В отличие от метода **Refresh,** метод **RefreshRecord** клиентских наборов данных обновляет только текущую запись, запрашивая у провайдера ее текущие значения в базе данных. В результате обновления изменяются (если есть изменения) начальные значения полей, полученные от провайдера ранее, но сохраняются все изменения, сделанные в клиентском наборе.

Обновление начальных значений записи, в которой проведено редактирование, может замаскировать конфликты, которые должны были бы возникнуть при последующей записи в базу данных методом **Apply Updates.** Так что обновление отредактированных записей надо осуществлять с осторожностью (см. пример).

Пример

Ниже приведен пример обработчика события **AfterScroll,** обновляющего значения полей текущей записи при каждом перемещении пользователя на новую запись. Это гарантирует, что запись в наборе данных будет отображать истинное ее состояние в базе в данный момент времени. Но обновляются только те записи, которые не редактировались. Это позволяет избежать возможного маскирования ошибок, на которое указывалось выше в описании метода **RefreshRecord.**

```
void __fastcall TForm1::ClientDataSet1AfterScroll(
                                                TDataSet *DataSet)
{
  if (ClientDataSet1->UpdateStatus == usUnModified)
    ClientDataSet1->RefreshRecord();
}
```

RefreshLookupList

См. в гл. 3 разд. «LookupList и RefreshLookupList».

Remove

Удаляет элемент с заданным значением из списка TList.

Класс *TList*

Объявление

```
int __fastcall Remove(void * Item);
```

Описание

Функция **Remove** удаляет указатель, равный заданному параметру **Item,** из списка **TList.** Функцию можно использовать вместо метода **Delete.** когда не известен индекс, соответствующий удаляемому указателю. Функция возвращает индекс, который имел данный указатель до его удаления. Индексы всех последующих указателей в списке уменьшаются на 1. Свойство **Count** также уменьшается на 1.

Если массив содержит несколько одинаковых указателей, то удаляется только первое вхождение этого указателя.

Repaint

Перерисовывает изображение компонента на экране.

Класс *TControl*

Объявление

```
virtual void __fastcall Repaint(void);
```

Описание

Вызов метода **Repaint** приводит к немедленной перерисовке изображения на экране. Если свойство **ControlStyle** компонента включает **csOpaque,** компонент перерисовывает себя сам. В противном случае **Repaint** вызывает метод **Invalidate.** а затем метод **Update.**

RevertRecord

Удаляет результаты редактирования только текущей записи.

См. разд. «Cancel и другие методы отмены исправлений в наборах данных».

RoundRect

Рисует на канве прямоугольную рамку со скругленными углами.

Класс *TCanvas*

Объявление

```
void __fastcall RoundRect(int X1, int Y1, int X2,
                          int Y2, int X3, int Y3);
```

Описание

Метод **RoundRect** рисует на канве прямоугольную рамку со скругленными углами, используя текущие установки пера Реп и заполняя площадь фигуры текущей кистью **Brush.** Рамка определяется прямоугольником с координатами углов (X1,Y1) и (X2,Y2). Углы скругляются с помощью эллипсов с шириной X3 и высотой **Y3.**

Если задать ширину эллипса X3 i X2 - **X1,** то верхняя и нижняя границы рамки окажутся целиком скругленными (без прямолинейной части). Если Y3 ⩾ Y2 - **Y1,** то же самое произойдет с левой и правой границами рамки. Если же оба измерения эллипса не меньше размеров рамки, то будет рисоваться просто эллипс. Но, конечно, для рисования эллипса лучше использовать метод **Ellipse.** Если один из размеров эллипса задать нулевым, то будет рисоваться прямоугольная рамка. Но, конечно, для такой рамки лучше использовать метод **Rectangle.**

Пример

Следующие операторы вызывают изображение, показанное на рис. 4.5:

```
Image1->Canvas->RoundRect(10,10,110,210,50,100);
Image1->Canvas->RoundRect(160,10,260,210,100,100);
Image1->Canvas->RoundRect(310,10,410,210,50,200);
Image1->Canvas->RoundRect(460,10,560,210,100,200);
```

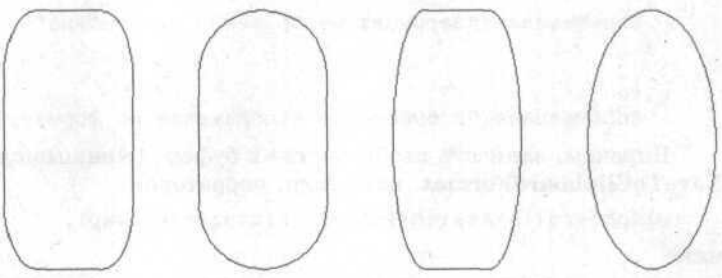


Рис. 4.5
Изображения, полученные методом RoundRect

SaveToClipboardFormat

Создает копию изображения в формате Clipboard.

Класс *TGraphic*

Объявление

```
virtual void __fastcall SaveToClipboardFormat(
                        Word &AFormat, int &AData,
                        HPALETTE &APalette) = 0;
```

Описание

Метод **SaveToClipboardFormat** создает копию изображения в формате Clipboard. Формат, указатель на данные и палитру возвращаются как параметры **AFormat, AData и APalette.** Стандартно зарегистрированные форматы: **CF_BITMAP** для битовых карт и **CF_METAFILEPICT** для метафайлов. Формат для нового типа графического объекта предварительно должен быть зарегистрирован методом **RegisterClipboardFormat.**

После применения метода **SaveToClipboardFormat** надо передать объекту **Clipboard** полученные значения **AFormat** и **AData** методом **SetAsHandle.** При этом надо не забыть включить в модуль ссылку на модуль *Clipbrd.*

Впрочем, записать изображение в Clipboard можно и проще, воспользовавшись методом Assign объекта **Clipboard** для объектов типов **TGraphic. TBitMap. TIcon, TMetafile.**

Примеры

```
tinclude <vcl\Clipbrd.hpp>
...
Word MyFormat;
THandle AData;
HPALETTE APalette;
...
Image1->Picture->Bitmap->SaveToClipboardFormat(
                                    MyFormat, AData, APalette);
Clipboard()->SetAsHandle(MyFormat,AData);
```

Приведенные операторы записывают в буфер обмена изображение, хранящееся в свойстве **Picture->Bitmap** компонента **Imagel,** вместе с палитрой и регистрируют формат **MyFormat.** В дальнейшем его можно использовать для чтения изображения из буфера:

```
if (Clipboard()->HasFormat(MyFormat))
{
 try
 {
  Image2->Picture->Bitmap->LoadFromClipboardFormat(
            CF_BITMAP, Clipboard()->GetAsHandle(MyFormat), 0);
 }
 catch (...)
 {
  ShowMessage("Загрузка изображения невозможна");
 }
}
else
  ShowMessage("В буфере не изображение по формату MyFormat");
```

Впрочем, записать изображение в буфер обмена можно и не используя метод **SaveToClipboardFormat,** например, оператором:

```
Clipboard()->Assign(Image1->Picture->Bitmap);
```

или

```
Clipboard()->Assign(Image1->Picture->Graphic);
```

SaveToFile — метод TGraphic

Сохраняет графическое изображение в файле.

Класс *TGraphic*

Объявление

```
virtual void __fastcall SaveToFile(const AnsiString Filename);
```

Описание

Метод **SaveToFile** сохраняет изображение графического объекта в файле **FileName.**

SaveToFile — метод TCustomClientDataSet

См. разд. «LoadFromFile и другие методы загрузки и сохранения данных класса TCustomClientDataSet».

SaveToStream — метод TGraphic

Сохраняет графическое изображение в потоке.

Класс *TGraphic*

Объявление

```
virtual void__fastcall SaveToStream(Classes::TStream* Stream) = 0;
```

Описание

Метод **SaveToStream** сохраняет в потоке **Stream** изображение графического объекта. Используется, например, для сохранения в объекте **TBlobStream** графического поля из набора данных,

SaveToStream — метод TCustomClientDataSet

См. разд. «LoadFromFile и другие методы загрузки и сохранения данных класса TCustomClientDataSet».

ScaleBy

Масштабирует оконный элемент и все содержащиеся в нем компоненты.

Класс *TWinControl*

Объявление

```
void__fastcall ScaleBy(int M, int D);
```

Описание

Метод **ScaleBy** масштабирует оконный элемент и все содержащиеся в нем компоненты. Масштабируются такие свойства компонента, как **Width и Height,** определяющие его размер. Свойства **Top** и **Left** остаются неизменными. Масштабируется также размер шрифта, если только в компоненте не установлено **ParentFont = true.** В последнем случае шрифт наследуется от родительского компонента и поэтому не изменяется.

Если компонент является контейнером, содержащим другие компоненты, то эти дочерние компоненты также масштабируются. Причем у них изменяются не только **Width и Height,** но также пропорционально изменяются **Top** и **Left,** определяющие их местоположение. Если во всех дочерних компонентах установлено **ParentFont = true,** а в компоненте-контейнере **ParentFont = false,** то пропорционально изменяются и шрифты всех компонентов (но, конечно, не непрерывно, а скачками, доступными тому или иному типу шрифта).

Параметры M and D определяют соответственно множитель и делитель масштаба. Например, чтобы уменьшить размеры на 10% начального значения, можно задать M равным 9, а D равным 10 (9/10). Если же вы хотите увеличить размер на 1/3, то можно задать M=133 и D=100 (133/100) или M=4 и D=3 (4/3).

Примеры

1. Оператор

   ```
   Edit1->ScaleBy(11,10);
   ```

   масштабирует окно редактирования **Edit1.** В любом случае при выполнении этого оператора увеличивается на 10% длина окна (свойство **Width),** что обеспечивает возможность наблюдать и редактировать в нем более длинный текст. Высота окна (свойство **Height)** будет изменяться пропорционально, если свойство компонента **Auto**Size равно **false.** В противном случае высота определяется только размером шрифта и при постоянном шрифте будет неизменной. А размер шрифта будет меняться, только если свойство компонента **Parent-Font** равно **false,** т.к. иначе шрифт определяется родительским компонентом.

2. Приведенный ниже обработчик события **OnKeyUp** окна редактирования **Editl** дает пользователю возможность менять длину окна. При нажатии комбинаций клавиш Alt-U и Alt-D пользователь увеличивает или уменьшает длину окна.

```
void__fastcall TForm1::Edit1KeyUp(TObject *Sender,
                                  WORD SKey, TShiftState Shift)
{
  if ((Key == 'U') && Shift.Contains(ssAlt))
    Edit1->ScaleBy(11,10);
  else if ((Key == 'D') && Shift.Contains(ssAlt))
    Edit1->ScaleBy(10,11);
}
```

Когда **Edit1** находится в фокусе, при нажатии пользователем клавиши Alt и клавиши U (в любом регистре и независимо от переключения на латинский или русский язык) длина окна редактирования увеличится на 10%, а при нажатии Alt и D соответственно уменьшится.

ScaleControls

Масштабирует дочерние компоненты оконного элемента, не изменяя масштаба самого элемента.

Класс *TWinControl*

Объявление

```
void__fastcall ScaleControls(int M, int D);
```

Описание

Метод **ScaleControls** масштабирует все компоненты, содержащиеся в оконном элементе, не изменяя масштаба самого элемента. Метод **ScaleControls** вызывает метод **ChangeScale** для каждого дочернего компонента. Отличается от метода **ScaleBy** только тем, что не изменяет масштаба самого элемента.

Параметры M and D определяют соответственно множитель и делитель масштаба. Например, чтобы уменьшить размеры на 10% начального значения, можно задать M равным 9, а D равным 10 (9/10). Если же вы хотите увеличить размер на 1/3, то можно задать M=133 и D=100 (133/100) или M=4 и D=3 (4/3).

Подробности см. в разделах **ChangeScale и ScaleBy.**

ScreenToClient

Преобразует координаты экрана в координаты клиентской области компонента.

Класс *TControl*

Объявление

```
struct TPoint // полное определение см. в гл. 6
{
 int x;
 int y;
};
Windows::TPoint__fastcall ScreenToClient(
                                const Windows::TPoint &Point);
```

Описание

Метод **ScreenToClient** преобразует координаты точки в системе координат экрана (начало координат — левый верхний угол экрана) в систему координат клиентской области компонента (начало координат — левый верхний угол клиентской области).

Совместно с обратной функцией **ClientToScreen** метод может использоваться для пересчета координат точки экрана из системы координат клиентской области одного компонента в систему координат клиентской области другого компонента.

Пример

```
P.=Comp2->ScreenToClient(Comp1->ClientToScreen(P));
```

Оператор пересчитывает координату точки P из системы координат компонента **Comp1** в систему координат компонента **Сотр2.**

ScrollBy

Сдвигает содержимое оконного элемента.

Класс *TWinControl*

Объявление

```
void__fastcall ScrollBy(int DeltaX, int DeltaY);
```

Описание

Метод **ScrollBy** сдвигает содержимое оконного элемента, включая все его дочерние компоненты. Метод применим ко всем оконным элементам, но чаще всего используется для наследников класса **TScrollingWinControl.**

Параметры **DeltaX и DeltaY** определяют величину сдвига по горизонтали и вертикали соответственно. Положительные значения параметров задают сдвиг вправо и вниз, отрицательные значения — влево и вверх.

Пример

Оператор

```
ScrollBy(1,1);
```

сдвигает все компоненты на форме на один пиксел вправо и на один вниз.

SelectFirst

Передает фокус дочернему компоненту, первому в последовательности табуляции.

Класс *TWinControl*

Объявление

```
void__fastcall SelectFirst(void);
```

Описание

Метод **SelectFirst** передает фокус дочернему компоненту, первому в последовательности табуляции. Он вызывает метод **FindNextControl,** передавая ему параметр, равный **NULL.** В результате метод **FindNextControl** возвращает первый компонент в последовательности табуляции, после чего этот компонент делается активным.

Пример

Оператор

```
SelectFirst();
```

активизирует первый в последовательности табуляции компонент на форме.

SelectNext

Передает фокус дочернему компоненту, следующему в последовательности табуляции за указанным.

Класс *TWinControl*

Объявление

```
void__fastcall SelectNext(TWinControl* CurControl,
                          bool GoForward, bool CheckTabStop);
```

Описание

Метод **SelectNext** передает фокус дочернему компоненту, следующему в последовательности табуляции за тем, который указан параметром **CurControl. Па-**

раметр GoForward определяет направление поиска: при значении **true** поиск ведется вперед, при значении **false** — назад.

Параметр **CheckTabStop** указывает, должен ли искомый компонент иметь свойство **TabStop.** равным true. Если значение **CheckTabStop** равно true, очередной компонент должен иметь значение TabStop. равное true, или поиск прекращается.

Если метод SelectNext не смог найти компонент в соответствии с заданными значениями **GoForward и CheckTabStop,** то фокус остается на компоненте **CurControl.**

SendCancelMode

Прерывает модальное состояние элемента управления.

Класс *TControl*

Объявление

```
void__fastcall SendCancelMode(TControl* Sender);
```

Описание

Вызов метода **SendCancelMode** прерывает модальное состояние элемента управления. Ряд элементов, реализованных в библиотеке визуальных компонентов C++Builder, поддерживают модальное состояние, при котором пользователь должен ответить элементу прежде, чем он сможет общаться с другими объектами формы. Метод **SendCancelMode** позволяет завершить модальное состояние без каких-либо действий со стороны пользователя.

SendToBack

Переносит компонент ниже других компонентов в Z-последовательности.

Класс *TControl*

Объявление

```
void__fastcall SendToBack(void);
```

Описание

Метод SendToBack позволяет изменять последовательность перекрытия компонентов на форме и тем самым управлять видимостью компонентов.

Перекрывающиеся компоненты на форме размещаются поверх друг друга в последовательности (называемой Z-последовательностью), соответствующей порядку размещения компонентов в процессе проектирования. Например, если вы поместили в одно и то же место формы две кнопки одинаковых размеров, то видна будет только вторая из размещенных кнопок, поскольку она расположена в Z-последовательности выше. Применение во время выполнения приложения метода **SendToBack** к верхней кнопке переместит ее вниз в Z-последовательности и пользователю станет видна нижняя кнопка.

Если переносимый вниз компонент имел фокус, то он его потеряет при переносе.

Это справедливо по отношению к неоконным объектам, таким, как кнопки, метки, изображения и т.д., а также и к оконным компонентам, таким, как **Memo, ComboBox** и др. Но все неоконные компоненты всегда расположены в Z-последовательности ниже оконных и метод **SendToBack** не может изменить это правило. Например, попытка перенести вниз методом **SendToBack** оконный компонент, под которым размещена метка, ни к чему не приведет.

Примеры

В разделе, посвященном методу BringToFront, также изменяющему последовательность компонентов, приведен ряд примеров. Во всех них вместо метода **BringToFront,** применяемому к нижнему компоненту, можно применять метод **SendToBack,** но к верхнему компоненту.

SetBounds

Устанавливает одновременно свойства **Left. Top. Width и Height.**

Класс *TControl*

Объявление

```
virtual void__fastcall SetBounds(int ALeft, int ATop,
                                 int AWidth, int AHeight);
```

Описание

Метод **SetBounds** изменяет одновременно все свойства компонента, определяющие его границу. Тот же эффект может быть достигнут совокупностью операторов изменения **Left. Top. Width и Height.** То, что метод **SetBounds** одновременно задает эти значения, не только позволяет получить более компактный код, но и дает возможность избежать перерисовки компонента после изменения каждого параметра в отдельности.

Значения **Left. Top. Width и Height** задаются при вызове **SetBounds** как соответственно параметры **ALeft, ATop, AWidth и AHeight.**

Примеры, в которых требуется изменять сразу все эти свойства, см. в разделах **Visible и BringToFront.**

SetChildOrder

Изменяет позицию компонента в списке дочерних компонентов оконного элемента.

Класс *TWinControl*

Объявление

```
DYNAMIC void__fastcall SetChildOrder(
                       Classes::TComponent* Child, int Order);
```

Описание

Метод **SetChildOrder** изменяет положение компонента, заданного параметром **Child,** в списке дочерних компонентов оконного элемента. Этот список — свойство **Controls** оконного элемента.

Параметр **Order** определяет индекс, присваиваемый компоненту **Child** (помните, что индексы начинаются с 0). Последовательность индексов остальных компонентов остается неизменной, но сами значения индексов «смыкаются», заполняя прежний индекс изменяемого компонента, и «раздвигаются» освобождая место для его нового индекса.

Пример

Если, например, компонент **Edit2** был в списке **Controls** вторым (т.е. имел индекс 1), то после выполнения оператора

```
SetChildOrder(Edit2,3);
```

элемент с индексом 0 сохранит свой индекс, элементы с индексами 2 и 3 изменят индексы на 1 и 2, компонент **Edit2** будет иметь индекс 3, а индексы остальных элементов не изменятся.

SetData

Присваивает полю неформатированные данные.

Класс *TField*

Объявление

```
void __fastcall SetData(void * Buffer, bool NativeFormat);
```

Описание

Метод **SetData** присваивает полю неформатированные данные, приводя, или не приводя их к формату типа данного поля. Это метод неявно используют многие другие методы, присваивающие значения полю.

Параметр **Buffer** указывает на буфер, который содержит вводимые данные. Параметр **NativeFormat** указывает, должны ли данные переноситься без форматирования (при значении true), или они предварительно должны приводиться к формату поля.

SetFields

Устанавливает значения всех полей записи.

Класс *TDataSet*

Объявление

```
void __fastcall SetFields(const System::TVarRec * Values,
                          const int Values_Size);
```

Описание

Метод **SetFields** устанавливает значения всех полей активной записи. Массив **Values** содержит расположенные в последовательности полей записи вносимые значения. Они могут быть строковыми константами, переменными, значениями **NULL** или **NULL.** Величина **NULL** соответствует очистке значения поля. Величина **NULL** означает, что сохраняется прежнее значение поля. Если количество передаваемых в запись значений меньше числа полей, то всем остальным полям записи присваивается значение **NULL.**

Перед вызовом **SetFields** состояние **State** набора данных надо перевести в режим **dsEdit** методом **Edit.** А после вызова **SetFields** занести отредактированную запись в базу данных можно методом **Post.**

Пример

Приведенные ниже операторы оставляют значение первого поля неизменным, значение второго поля очищают, а в значения следующих трех полей заносят данные из окон редактирования **Edit1, Edit2, Edit3.** Все последующие поля очищаются. Внесенные изменения переносятся в базу данных.

```
Table1->SetFields (ARRAYOFCONST((
   NULL, Null, Edit1->Text, Edit2->Text, Edit3->Text)));
Table1->Post();
```

SetFocus

Передает фокус элементу.

Класс *TWinControl*

Объявление

```
virtual void __fastcall SetFocus(void);
```

Описание

Метод **SetFocus** передает фокус данному компоненту, активизирует его.

Пример

Оператор

```
Edit1->SetFocus();
```

передает фокус компоненту **Editl.**

SetKey, EditKey, GotoKey, GotoNearest

Обеспечивают поиск записей по ключу в состоянии набора **dsSetKey.**

Класс *TCustomClientDataSet, TTable*

Объявления

```
void__fastcall  SetKey(void);
void__fastcall  EditKey(void);
bool__fastcall  GotoKey(void);
void__fastcall  GotoNearest(void);
```

Описание

Методы **SetKey, EditKey, GotoKey, GotoNearest** обеспечивают один из альтернативных (см. также **FindKey, Lookup и Locate**) способов поиска записи по ключу.

Перед началом поиска таблица должна быть индексирована по тем полям, которые будут ключевыми для поиска. Начинается поиск с метода **SetKey** или **EditKey.** Оба метода переводят набор данных (его свойство **State**) в состояние **dsSetKey.** Различие между методами **SetKey и EditKey** заключается только в том, что метод **EditKey** сохраняет буфер ранее установленного ключа. Так что этот метод целесообразно применять в случаях, когда задано много ключей, а вы хотите изменить только немногие из них. Определить текущие ключи можно с помощью свойства **IndexFields.**

Следующие за операторами **SetKey** или **EditKey** операторы присваивания значений полям с помощью метода **FieldByName** задают значения ключей поиска (см. пример).

После задания ключей можно применить метод **GotoKey.** Если найдена запись, в которой ключевые поля точно совпадают с заданными значениями ключей, то курсор переместится на эту запись, а **GotoKey** вернет **true.** Если же такая запись не найдена, то курсор останется на месте, а **GotoKey** вернет **false.**

Обычно более удобен метод **GotoNearest,** который перемещает курсор на запись, значения ключевых полей которой наиболее близки к заданным значениям ключей. Это позволяет проводить, например, ускоренный поиск записей со строковыми полями при посимвольным вводе искомого значения, как показано в примере. Значение свойства **KeyExclusive** определяет, на какую именно запись переместится курсор при частичном совпадении значений полей и ключей.

Примеры

Пусть в вашем приложении имеется набор данных **Table1,** содержащий список сотрудников, в котором поле **Fam** содержит фамилию сотрудника, а поле **Dep** — наименование отдела, в котором он работает. Приложение имеет также окно редактирования **EFam,** в котором пользователь вводит символы фамилии. Требуется осуществить ускоренный поиск записи по мере ввода пользователем фамилии искомого сотрудника.

При использовании методов **SetKey** и **GotoNearest** достаточно в обработчик события **OnChange** окна **EFam** вставить операторы

```
Table1->IndexFieldNames = "Fam";
Table1->SetKey();
Table1->FieldByName("Fam")->AsString = EFam->Text;
Table1->GotoNearest();
```

Первый оператор индексирует набор данных по полю **Fam.** Без такой индексации метод **SetKey** не сработает. Метод **GotoNearest** обеспечивает переход на запись с фамилией, наиболее похожей на введенную в **EFam** (совпадающую по первым символам).

Рассмотренную задачу можно также решить методами **Locate** и совместным применением методов **FindKey и FindNearest** (см. примеры в описаниях этих методов).

SetZOrder

Перемещает компонент на вершину или в низ Z-последовательности.

Класс *TWinControl*

Объявление

```
DYNAMIC void__fastcall SetZOrder(bool TopMost);
```

Описание

Метод **SetZOrder** перемещает компонент на вершину или в низ Z-последовательности родительского элемента. Если родительского элемента нет, то элемент становится верхним или нижним окном экрана. Таким образом, этот метод перестраивает перекрывающиеся компоненты или окна.

Если параметр **TopMost** равен **true**, то элемент перемещается на вершину; в противном случае он перемещается вниз.

При этом надо иметь в виду, что оконные элементы всегда расположены в Z-последовательности выше неоконных. Так что при выполнении этого метода перемещения происходят только в этих допустимых пределах.

Show

Делает видимым невидимый компонент.

Класс *TControl*

Объявление

```
void__fastcall Show(void);
```

Описание

Метод **Show** делает видимым ранее невидимый компонент. Он задает значение **true** свойству **Visible** и проверяет, является ли видимым родительский компонент.

Примеры использования видимых и невидимых компонентов см. в разделе «Visible».

StretchDraw

Рисует графическое изображение в указанную прямоугольную область канвы, подгоняя размер изображения под заданную область.

Класс *TCanvas*

Объявление

```
void__fastcall StretchDraw(const Windows::TRect &Rect,
                           TGraphic* Graphic);
```

Описание

Метод **StretchDraw** рисует на канве изображение, содержащееся в объекте, указанном параметром **Graphic**, в прямоугольную область, указанную параметром **Rect**. При этом размер изображения подгоняется под размер заданной области. Этим метод **StretchDraw** отличается от метода **Draw**, который оставляет размер неизменным.

Объект **Graphic** может быть типа битовой матрицы, пиктограммы или метафайла. Если объект — битовая матрица типа **TBitMap**, то при переносе изображения учитывается режим копирования, установленный свойством канвы **CopyMode**.

Пример

Оператор

```
Image1->Canvas->StretchDraw(Rect(10,10,110,110),Bitmap1);
```

рисует на канве компонента **Image1** изображение из компонента **Bitmap1** в область с координатами углов (10, 10) и (110, 110). При этом размер изображения подгоняется под заданный размер области — квадрат со стороной 100.

TextExtent

Возвращает длину и высоту в пикселах текста, который предполагается написать на канве текущим шрифтом.

Класс *TCanvas*

Объявление

```
struct TSize
 (
  LONG cx;
  LONG cy;
 };

TSize__fastcall TextExtent(const AnsiStrinq Text);
```

Описание

Функция **TextExtent** возвращает структуру типа **TSize,** содержащую длину и высоту в пикселах текста **Text,** который предполагается написать на канве (см. Canvas) текущим шрифтом. Это позволяет перед выводом текста на канву определить размер надписи и расположить ее и другие элементы изображения наилучшим образом.

Только высоту или только длину текста можно определять соответственно методами **TextHeight и Text Width.**

Пример

```
v String st = Edit1->Text;
Canvas->TextOut((ClientWidth - Canvas->TextExtent(st).cx) / 2,
                Canvas->TextExtent(st).cy,st)
```

Эти операторы выводят текст, набранный пользователем в окне редактирования **Edit1,** в канву компонента **Image1,** выравнивая его при любом шрифте по середине ширины канвы и отступив одну строчку сверху.

TextHeight

Возвращает высоту в пикселах текста, который предполагается написать на канве текущим шрифтом.

Класс *TCanvas*

Объявление

```
int__fastcall TextHeight(const AnsiString Text);
```

Описание

Функция **TextHeight** возвращает высоту в пикселах текста **Text,** который предполагается написать на канве **Canvas** текущим шрифтом. Это позволяет перед выводом текста на канву определить размер надписи и расположить ее и другие элементы изображения наилучшим образом.

Имеется еще метод **TextExtent.** возвращающий одновременно и высоту, и длину текста. Метод **TextHeight** возвращает то же, что **TextExtent(Text).cy.**

Примеры

```
String st = Edit1->Text;
Image1->Canvas->TextOut((Image1->ClientWidth -
                         Image1->Canvas->TextWidth(st)) / 2,
                         Image1->Canvas->TextHeight(st), st);
```

Эти операторы выводят текст, набранный пользователем в окне редактирования Edit1, в канву компонента **Imagel,** выравнивая его при любом шрифте по середине ширины канвы и отступив одну строчку сверху.

См. также пример в описании метода TextRect.

TextOut

Пишет указанную строку текста на канве, начиная с указанной позиции.

Класс *TCanvas*

Объявление

```
void__fastcall TextOut(int X, int Y, const AnsiString Text);
```

Описание

Функция **TextOut** пишет строку текста **Text** на канве (см. Canvas), начиная с позиции с координатами (X, Y). Надпись делается в соответствии с текущими установками шрифта **Font.** Фон надписи определяется установками текущей кисти **Brush.** Текущая позиция **PenPos** пера Реп перемещается к концу выведенного текста.

Для выравнивания позиции текста на канве можно использовать методы, дающие перед выводом высоту и длину текста в пикселах: методы **TextExtent. TextHeight и** TextWidth.

Если цвет кисти *в* момент вывода текста отличается от того, которым закрашена канва, то текст получится выведенным в цветной прямоугольной рамке. Но ее размеры будут точно равны размерам надписи. Если требуется более красивая рамка с отступом от текста или если надо ограничить выводимый текст размерами определенной рамки, следует применять метод **TextRect.**

Пример

Оператор

```
Image1->Canvas->TextOut(10,10,s);
```

выводит текст, хранящийся в строковой переменной s, на канву компонента Imagel, начиная с позиции (10, 10).

См. также примеры в описаниях методов **TextExtent. TextHeight.** TextWidth **и TextRect.**

TextRect

Пишет указанную строку текста на канве, начиная с указанной позиции и усекая текст, выходящий за пределы указанной прямоугольной области.

Класс *TCanvas*

Объявление

```
void__fastcall TextRect(const Windows::TRect &Rect,
                        int X, int Y, const AnsiString Text);
```

Описание

Функция **TextRect** пишет строку текста **Text** на канве (см. Canvas), начиная с позиции с координатами (X, Y) — это левый верхний угол надписи. Часть текста, не помещающаяся в прямоугольную область **Rect,** усекается. Надпись делается в соответствии с текущими установками шрифта **Font.** Пространство внутри области **Rect** закрашивается текущей кистью **Brush.**

Для выравнивания позиции текста внутри области на канве можно использовать методы, дающие перед выводом высоту и длину текста в пикселах: методы **TextExtent. TextHeight и** TextWidth.

Примеры

```
1.  int X1,Y1,X2,Y2;
    String st = Edit1->Text;
    X1 = 100;
    Y1 = 100;
    X2 = 200;
    Y2 = 150;
    Image1->Canvas->Brush->Color = clRed;
    Image1->Canvas->TextRect  (Rect(X1,Y1,X2,Y2),
              X1+(X2-X1-Image1->Canvas->TextWidth(st)) / 2,
              Y1+(Y2-Y1-Image1->Canvas->TextHeight(st)) / 2, st) ;
```

Приведенный код рисует в заданном месте канвы компонента **Image1** красный прямоугольник и внутри него в центре пишет методом **TextRect** текст, введенный пользователем в окно редактирования **Edit1**. Если текст оказывается длиннее ширины прямоугольника, то он усекается. В данном примере будет видна только середина текста, так как текст выровнен по центру.

2. Если в приведенном примере заменить оператор на

```
Image1->Canvas->TextRect(Rect(X1-5,Y1-5,
                         X1+Image1->Canvas->TextWidth(st)+5,
                         Y1+Image1->Canvas->TextHeight(st)+5),
                         X1, Y1, st);
```

то текст будет выводиться полностью в красной прямоугольной области, на 5 пикселов отступающей во все стороны от текста. Именно этим отступом, делающим надпись более красивой, этот оператор отличается от более простого оператора

```
TextOut(X1,Y1,st);
```

использующего метод **TextOut.**

TextWidth

Возвращает длину в пикселах текста, который предполагается написать на канве текущим шрифтом.

Класс *TCanvas*

Объявление

```
int__fastcall TextWidth(const AnsiString Text);
```

Описание

Функция **TextWidth** возвращает длину в пикселах текста **Text,** который предполагается написать на канве (см. **Canvas**) текущим шрифтом. Это позволяет перед выводом текста на канву определить размер надписи и расположить его и другие элементы изображения наилучшим образом.

Имеется еще метод **TextExtent.** возвращающий одновременно и высоту, и длина текста. Метод **TextWidth** возвращает то же, что **TextExtent(Text).cx.**

Примеры

```
String st = Edit1->Text;
Image1->Canvas->TextOut((Image1->ClientWidth -
                         Image1->Canvas->TextWidth(st)) / 2,
                         Image1->Canvas->TextHeight(st), st);
```

Эти операторы выводят текст, набранный пользователем в окне редактирования **Editl,** в канву компонента **Image1,** выравнивая его при любом шрифте по середине ширины канвы и отступив одну строчку сверху.

См. также пример в описании метода **TextRect.**

Translate

Взаимно преобразует строку символов ANSI (используются в C++Builder и Windows) и строку символов OEM (используются в BDE).

Классы *TDataSet* и *TBDEDataSet*

Объявление

```
virtual void __fastcall Translate(char * Src,
                                  char * Dest, bool ToOem);
```

Описание

Метод **Translate** копирует строку **Src** в строку **Dest,** преобразуя при этом символы ANSI (используются в C++Builder и Windows) в символы OEM (используются в BDE), или наоборот. Если **ToOem = true,** то символы ANSI преобразуются в OEM. Если **ToOem = false,** то символы OEM преобразуются в ANSI.

TryLock

Блокирует канву, если она не была блокирована, не разрешая другим нитям многопоточного приложения рисовать на ней.

Класс *TCanvas*

Объявление

```
bool __fastcall TryLock(void);
```

Описание

Функция **TryLock** блокирует данную канву, не разрешая другим нитям многопоточного приложения рисовать на ней. Канва остается блокированной до снятия блокады вызовом метода **Unlock.**

Если канва не была блокирована, то функция **TryLock** устанавливает свойство **LockCount** в 1 и возвращает **true.** Если канва уже была блокирована, то функция **TryLock** возвращает **false,** не увеличивая **LockCount.** В этом ее отличие от метода **Lock,** который допускает многократное вложенное блокирование.

Поскольку блокировка не дает другим нитям рисовать на канве, производительность работы приложения может за счет этого снизится. Так что не надо злоупотреблять блокировками. Их следует применять только тогда, когда есть вероятность нежелательных наложений операций, выполняемых в разных нитях приложения с несколькими потоками.

UndoLastChange

Отменяет результаты последней операции редактирования.

См. разд. «Cancel и другие методы отмены исправлений в наборах данных».

Unlock

Уменьшает на единицу значение свойства **LockCount,** способствуя тем самым разблокированию канвы, когда **LockCount** станет равным 0.

Класс *TCanvas*

Объявление

```
void __fastcall Unlock(void);
```

Описание

Метод **Unlock** уменьшает на единицу значение свойства **LockCount,** способствуя тем самым разблокированию канвы, заблокированной ранее методами **Lock** или **TryLock.** Блокировка канвы не разрешает другим нитям многопоточного приложения рисовать на ней. Канва остается блокированной до тех пор, пока свойство

LockCount не станет равным 0. А единственный способ уменьшения **LockCount** — вызов метода **Unlock.**

Если канва была заблокирована однократно, то вызов **Unlock** немедленно разблокирует ее. Если же были вложенные вызовы **Lock,** то канва будет разблокирована после стольких вызовов **Unlock,** сколько раз ранее вызывался **Lock.**

Update — метод TFieldDefs

Обновляет описания в массиве **Items,** чтобы они отражали реальные свойства полей.

Класс *TFieldDefs*

Объявление

```
HIDESBASE void__fastcall Update(void);
```

Описание

Метод **Update** обновляет описания полей в массиве **Items,** чтобы они отражали реальные свойства полей, если они были изменены в наборе данных. Обновление происходит без открытия набора данных. Перед обновлением полезно проверить свойство **Updated.**

Update — метод TWinControl

Немедленно перерисовывает компонент.

Класс *TWinControl*

Объявление

```
virtual void__fastcall Update(void);
```

Описание

Метод **Update** вызывает немедленную перерисовку компонента, не ожидая завершения каких-то других процессов или прихода от Windows сообщений о перерисовке. Этим, в основном, этот метод и отличается от **Repaint.**

UpdateStatus

Определяет статус обновления текущей записи.

Класс *TDataSet*

Объявление

```
enum TUpdateStatus ( usUnmodified, usModified,
                     usInserted, usDeleted };

virtual TUpdateStatus__fastcall UpdateStatus(void);
```

Описание

Функция **UpdateStatus** позволяет определить в случае кэширования, была ли изменена текущая запись, и если была, то как именно. Возможные возвращаемые значения:

usUnmodified	запись не изменена
usModified	запись модифицирована
uslnserted	запись была вставлена
usDeleted	запись была удалена

В зависимости от возвращенного значения можно применять те или иные операции при переносе кэшированных изменений в базу данных.

Глава 5

События компонентов и классов C++Builder

В этой главе приведено описание более 50 основных событий, на которые реагируют компоненты и объекты классов C++Builder. Для многих событий даются примеры обработчиков. Большее число описаний событий и примеров их обработки вы можете найти в источнике [3].

В описаниях вы можете встретить идентификаторы, выделенные подчеркиванием. Например, **TDataSet.** Это означает, что в этой или других главах в соответствующем разделе вы сможете найти развернутое пояснение, комментарий или примеры, связанные с данным термином.

Объявления событий несколько упрощены. В частности, отсутствуют указания функций чтения и записи соответствующих полей классов.

AfterCancel и BeforeCancel _

События, возникающие при выполнении метода **Cancel.**

Класс *TDataSet*

Определения

```
typedef void _ fastcall ( _ closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

_ property TDataSetNotifyEvent AfterCancel
_ property TDataSetNotifyEvent BeforeCancel
```

Описание

Событие **BeforeCancel** наступает в начале выполнения метода **Cancel,** а событие **AfterCancel** наступает после выполнения этой процедуры. Параметр **DataSet** указывает объект набора данных, к которому применяется метод **Cancel.**

Обработчики событий **BeforeCancel и AfterCancel** могут использоваться для каких-то операций, которые необходимо произвести при отказе от исправлений текущей записи, осуществляемом методом **Cancel.** Например, в обработчике **BeforeCancel** можно предусмотреть сохранение изменений в каком-то буфере или вообще отказаться от выполнения **Cancel,** вызвав функцию **Abort.**

Пример

Следующий обработчик события **AfterCancel** извещает пользователя об отмене введенных изменений записи:

```
void _fastcall TForm1 : : Table1AfterCancel (TDataSet *DataSet)
{
 StatusBar1->SimpleText = "Исправление записи в таблице " +
                  ((TTable *) DataSet) ->TableName + "  отменено";
}
```

AfterClose и BeforeClose

События, возникающие при закрытии соединения с набором данных.

Класс *TDataSet*

Определения

```
typedef void _fastcall (_closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

_ property TDataSetNotifyEvent AfterClose
_ property TDataSetNotifyEvent BeforeClose
```

Описание

Событие **BeforeClose** наступает перед закрытием набора данных методом **Close** или установкой свойства **Active в false.** Событие **AfterClose** наступает после закрытия набора данных. Параметр **DataSet** указывает объект закрываемого набора данных.

Обработчики событий **BeforeClose и AfterClose** могут использоваться для каких-то операций, которые необходимо произвести при закрытии набора данных. Например, в обработчике события **BeforeClose** можно предусмотреть запрос пользователю о необходимости закрыть набор данных и при отрицательном ответе пользователя прекратить закрытие, вызвав функцию **Abort.**

Пример

Следующий обработчик события **BeforeClose** определяет перед закрытием набора данных наличие в кэше изменений, не сохраненных в базе данных, и осуществляет их сохранение.

```
void _fastcall TForm1::Table1BeforeClose(TDataSet *DataSet)
{
 if (Table1->UpdatesPending)
   Table1->ApplyUpdates();
}
```

Следующий обработчик осуществляет весь комплекс операций, связанный с пересылкой изменений в базу данных. В нем используется компонент **Database1** типа **TDatabase.** Этот обработчик события **BeforeClose,** безусловно, предпочтительнее предыдущего.

```
void _fastcall TForm1::Table1BeforeClose(TDataSet *DataSet)
(
 if (Table1->UpdatesPending)
 {
  Database1->StartTransaction();
  try
  {
   // пересылка изменений в базу данных
   Table1->ApplyUpdates();
   // попытка их зафиксировать
   Database1->Commit();
  }
  catch (...)
  {
   // откат назад при безуспешной попытке фиксация
   Database1->Rollback();
   // генерация исключения,
   // чтобы предотвратить вызов CommitUpdates
   throw;
  }
  // очистка буфера кэша
  Table1->CommitUpdates();
 }
}
```

Следующий обработчик события **AfterClose** извещает пользователя о закрытии набора данных:

```
void _fastcall TForm1 : :Table1AfterClose(TDataSet *DataSet)
 (
  StatusBar1->SimpleText = "Работа с таблицей " +
                           ((TTable*)DataSet)->TableName +" завершена";
}
```

AfterDelete и BeforeDelete

События, возникающие при удалении текущей записи методом **Delete.**

Класс *TDataSet*

Определения

```
typedef void _fastcall (_closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

_ property TDataSetNotif yEvent AfterDelete
_ property TDataSetNotif yEvent BeforeDelete
```

Описание

Событие **BeforeDelete** наступает перед удалением текущей записи методом **Delete.** Событие **AfterDelete** наступает после удаления записи. Параметр **DataSet** указывает объект набора данных, запись которого удаляется.

Обработчики событий **BeforeDelete и AfterDelete** могут использоваться для каких-то операций, которые необходимо произвести при удалении записи. Например, в обработчике события **BeforeDelete** можно предусмотреть запрос пользователю о необходимости удаления и при отрицательном ответе пользователя прекратить удаление, вызвав функцию **Abort.**

Пример

Следующий обработчик события **BeforeDelete** запрашивает у пользователя разрешение на удаление записи. При отсутствии утвердительного ответа удаление не производится.

```
void _fastcall TForm1 : :Table1BeforeDelete(TDataSet *DataSet)
{
 if (Application->MessageBox(
                             "Действительно хотите удалить запись?",
                             "Подтвердите удаление записи",
                             MB_YESNOCANCEL + MB_ICONQUESTION) != IDYES)
   Abort();
}
```

AfterEdit и BeforeEdit

События, возникающие перед началом и после начала редактирования записи.

Класс *TDataSet*

Определения

```
typedef void__fastcall (__closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterEdit
__property TDataSetNotifyEvent BeforeEdit
```

Описание

Событие **BeforeEdit** наступает перед началом перехода набора данных в режим редактирования. Событие **AfterEdit** наступает сразу после перехода набора данных в режим редактирования. Параметр **DataSet** указывает объект редактируемого набора данных.

Обработчик события **BeforeDelete** можно использовать, например, чтобы запретить редактирование. Для этого в нем можно вызвать функцию **Abort.** Обработчик события **AfterDelete** можно использовать для какого-то извещения пользователя о начале редактирования, например, для отображения соответствующего сообщения в строке состояния приложения.

AfterInsert и BeforeInsert

События, возникающие непосредственно перед и после вставки новой записи.

Класс *TDataSet*

Определения

```
typedef void__fastcall (__closure *TDataSetNotifyEvent)
                                          (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterInsert
__property TDataSetNotifyEvent BeforeInsert
```

Описание

Событие **BeforeInsert** наступает перед переходом набора данных в режим вставки записи в результате вызова методов **Insert** или **Append.** Событие **AfterInsert** наступает сразу после вставки записи в результате вызова этих методов. Параметр **DataSet** указывает объект редактируемого набора данных.

Обработчик события **BeforeInsert** можно использовать, например, чтобы запретить вставку. Для этого в нем можно вызвать функцию **Abort.** Обработчик события **AfterInsert** можно использовать для какого-то извещения пользователя о начале вставки, например, для отображения соответствующего сообщения в строке состояния приложения.

Пример

Следующий обработчик события **BeforeInsert** запрашивает у пользователя разрешение на вставку записи. При отсутствии утвердительного ответа вставка не производится.

```
void__fastcall TForm1::Table1BeforeInsert(TDataSet *DataSet)
{
  if (Application->MessageBox(
                          "Действительно хотите вставить запись "
                          " в базу данных?",
                          "Подтвердите вставку записи",
                          MB_YESNOCANCEL + MB_ICONQUESTION)  != IDYES)
    Abort();
}
```

AfterOpen и BeforeOpen

События, возникающие непосредственно перед и после открытия набора данных.

Класс *TDataSet*

Определения

```
typedef void__fastcall (__closure *TDataSetNotifyEvent)
                                          (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterOpen
__property TDataSetNotifyEvent BeforeOpen
```

Описание

Событие **BeforeOpen** наступает перед открытием набора данных, т.е. при установке свойства **Active в true** или при вызове метода **Open.** Событие **AfterOpen** наступает сразу после открытием набора данных, когда он устанавливается в состоя-

ние **dsBrowse**. Параметр DataSet указывает объект редактируемого набора данных.

Обработчики событий **BeforeOpen** и **AfterOpen** могут выполнять какие-то действия, необходимые при открытии набора данных. Например, обработчик события AfterOpen можно использовать для того, чтобы прочитать в реестре или файле конфигурации номер записи, с которой приложение работало в последний раз, и установить курсор на эту запись.

Пример

Следующий обработчик события AfterOpen извещает пользователя об открытии набора данных:

```
void__fastcall TForm1::Table1AfterOpen(TDataSet *DataSet)
{
 StatusBar1->SimpleText = "Открыта таблица " +
                          ((TTable *)DataSet)->TableName;
}
```

AfterPost и BeforePost

События, возникающие непосредственно перед и после пересылки изменений в текущей записи в базу данных или буфер.

Класс *TDataSet*

Определения

```
typedef void__fastcall(__closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterPost
__property TDataSetNotifyEvent BeforePost
```

Описание

Событие **BeforePost** наступает перед пересылкой методом **Post** изменений в текущей записи в базу данных или буфер кеширования. Событие **AfterPost** наступает сразу после пересылки измененной записи. Параметр **DataSet** указывает объект редактируемого набора данных.

Обработчик события **BeforePost** можно использовать для проверки правильности исправлений в записи и запроса подтверждения исправлений со стороны пользователя. В этом обработчике можно отказаться от пересылки исправлений, вызвав функцию **Abort**. Обработчик события **AfterPost** можно использовать для сообщения пользователю о результатах пересылки исправлений в базу данных.

Пример

Один из множества возможных вариантов проверки достоверности данных перед их пересылкой в базу данных — использование события **BeforePost**. Обработчик этого события может иметь вид:

```
void__fastcall TForm1::Table1BeforePost(TDataSet *DataSet)
{
 if (проверка введенных данных)
 {
  if (Application->MessageBox(
                        "Хотите занести текущую запись в базу данных?",
                        "Подтвердите занесение в базу данных",
                       MB_YESNOCANCEL + MB_ICONQUESTION) != IDYES)
    {
      DataSet->Cancel();
      Abort();
    )
 }
 else
```

```
    {
      Application->MessageBox("Ошибочные данные", "Ошибка", MB_ICONSTOP);
      Abort();
    }
}
```

К этому обработчику будет происходить обращение перед выполнением метода **Post,** как бы он не был вызван: явно или вследствие перемещения по базе данных, если текущая запись была изменена. В обработчике сначала производится проверка данных в записи. Если она дает неудовлетворительный результат, то пользователю дается сообщение об ошибочности данных и выполняется функция **Abort,** прерывающая выполнение **Post.** Текущая запись остается в состоянии **dsEdit,** но ошибочные данные в ней не сбрасываются, что позволяет пользователю исправить их.

Если проверка данных в записи показала их правильность, то у пользователя запрашивается подтверждение изменений в базе данных. Если он ответит отрицательно, т.е. не нажмет кнопку Да, то для набора данных выполняется метод **Cancel,** а затем выполняется функция **Abort. Cancel** возвращает данные в текущей записи к состоянию, которое было до их редактирования, т.е. уничтожает результаты редактирования.

AfterRefresh и BeforeRefresh

События, возникающие непосредственно перед и после обновления отображаемых данных методом **Refresh.**

Класс *TDataSet*

Определения

```
typedef void __fastcall (__closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterRefresh
__property TDataSetNotifyEvent BeforeRefresh
```

Описание

Событие **BeforeRefresh** наступает перед обновлением отображаемых данных методом **Refresh.** Событие **AfterRefresh** наступает сразу после такого обновления. Параметр **DataSet** указывает объект отображаемого набора данных.

Обработчики событий **AfterRefresh и BeforeRefresh** можно использовать для каких-то операций, которые должны осуществляться при обновлении отображаемых данных, например, для соответствующей настройки компонентов, отображающих данные, для выдачи каких-то сообщений пользователю и т.п.

AfterScroll и BeforeScroll

События, возникающие непосредственно перед и после перемещения указателя таблицы на новую запись.

Класс *TDataSet*

Определения

```
typedef void __fastcall (__closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

__property TDataSetNotifyEvent AfterScroll
__property TDataSetNotifyEvent BeforeScroll
```

Описание

Событие **BeforeScroll** наступает перед перемещением на новую запись методами **First. Last. MoveBy, Next. Prior. FindKey, FindFirst. FindNext. FindLast. FindPrior. Locate.** Это событие наступает раньше всех иных событий, связанных

с перемещением по таблице. Событие **AfterScroll** наступает сразу после такого перемещения. Параметр **DataSet** указывает объект набора данных.

Обработчики событий **AfterScroll** и **BeforeScroll** можно использовать для каких-то операций, которые должны осуществляться при навигации по таблице. Например, в обработчике события **BeforeScroll** можно запретить перемещение, вызвав функцию **Abort.** Можно проверить по свойству **Modified,** были ли изменения в текущей записи, и если были, то решить, допустим ли последующий автоматический вызов метода **Post.**

Пример

См. пример в гл. 4, в разд. «Refresh, RefreshRecord — методы наборов данных».

BeforeCancel

Событие, возникающее в начале выполнения метода **Cancel.**

Подробнее см. в разделе «AfterCancel и BeforeCancel».

BeforeClose

Событие, возникающее перед закрытием соединения с набором данных.

Подробнее см. в разделе «AfterClose и BeforeClose».

BeforeDelete

Событие возникающее перед удалением текущей записи.

Подробнее см. в разделе «AfterDelete и BeforeDelete».

BeforeEdit

Событие, возникающее после начала редактирования записи.

Подробнее см. в разделе «AfterEdit и BeforeEdit».

BeforeInsert

Событие, возникающее непосредственно перед вставкой новой записи.

Подробнее см. в разделе «AfterInsert и BeforeInsert».

BeforeOpen

Событие, возникающее непосредственно перед открытием набора данных.

Подробнее см. в разделе «AfterOpen и BeforeOpen».

BeforePost

Событие, возникающее непосредственно перед пересылкой изменений в текущей записи в базу данных или буфер.

Подробнее см. в разделе «AfterPost и BeforePost».

BeforeRefresh

Событие, возникающее непосредственно перед обновлением отображаемых данных методом **Refresh.**

Подробнее см. в разделе «AfterRefresh и BeforeRefresh».

BeforeScroll

Событие, возникающее непосредственно перед перемещением указателя таблицы на новую запись.

Подробнее см. в разделе «AfterScroll и BeforeScroll».

OnCalcFields

Событие, наступающее при пересчете вычисляемых полей.

Класс *TDataSet*

Определение

```
typedef void __fastcall (__closure *TDataSetNotifyEvent)
                                        (TDataSet* DataSet);

__property TDataSetNotifyEvent OnCalcFields
```

Описание

Событие **OnCalcFields** наступает, когда осуществляется пересчет вычисляемых полей. Вычисляемые поля — это поля, отсутствующие в исходной базе данных, значения которых определяет приложение по значениям каких-то других полей записи. Этот расчет значений вычисляемых полей оформляется как обработчик события **OnCalcFields,**

Событие **OnCalcFields** наступает только в случае, если свойство **AutoCalcFields** набора данных установлено в **true.**

Событие **OnCalcFields** наступает при:

- открытии набора данных
- переходе набора данных в состояние **dsEdit**
- перемещении фокуса с одного компонента, отображающего данные, на другой
- перемещении в компоненте-таблице, отображающей данные, с одного столбца на другой
- модификации текущей записи
- получении записи из базы данных

Обработчик события **OnCalcFields** не должен изменять набор данных. В противном случае подобное изменение вызовет генерацию нового события **OnCalcFields,** что приведет к зацикливанию вычислений.

Пример

Следующий обработчик рассчитывает вычисляемое поле Age (возраст) таблицы **Table1** по значению поля Year_b (год рождения) и текущей дате.

```
void __fastcall TForm1::Table1CalcFields(TDataSet *DataSet)
{
 unsigned short Year, Month, Day;
 Date().DecodeDate(&Year,&Month,&Day);
 Table1Age->Value = Year - Table1Year_b->Value;
}
```

Вместо непосредственного обращения к объектам полей **Table1Age и Table1Year_b** можно было бы использовать более общий вид — обращение через параметр **DataSet: DataSet->FieldByName("Age") и DataSet->FieldByName("Year_b").**

В коде обработчика введены переменные **Year, Month** и **Day** для хранения текущего года, месяца и дня. Использована процедура **DecodeDate** для преобразования своего первого аргумента, имеющего тип **TDateTime** (этот тип используется в C++Builder для хранения дат и времени), в целые значения года, Месяца и дня. А в качестве первого аргумента этой процедуры указана функция **Date,** возвращающая текущую дату. В результате переменная **Year** становится равной текущему году (переменные **Month** и **Day** не нужны и объявлены только для того, чтобы можно было обратиться к процедуре **DecodeDate).**

Вместо принятого в примере непосредственного обращения к объектам полей **Table1Age и Table1Year_b** можно было бы использовать более общий вид — обращение через параметр **DataSet:**

```
DataSet->FieldByName("Age")
```

и

```
DataSet->FieldByName("Year_b")
```

OnChange — событие выпадающих списков

Событие наступает, когда изменяется текст, отображаемый в области редактирования списка.

Класс *TCustomComboBox*

Определение

```
typedef void __closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnChange
```

Описание

Событие **OnChange** наступает сразу после того, как пользователь изменил текст, отображаемый в области редактирования списка. Это изменение может быть связано как с непосредственным редактированием текста, так и с выбором пользователем нового значения в списке. В момент обработки события свойство Text равно новому значению текста.

Событие не наступает, если свойство Text изменяется программно, а не в результате действий пользователя.

OnChange — событие класса TField

Событие наступает после занесения данных поля в буфер записи.

Класс *TField*

Определение

```
typedef void __fastcall (__closure *TFieldNotifyEvent) (TField* Sender);

__property TFieldNotifyEvent OnChange
```

Описание

При занесении значения поля **Value** в буфер текущей записи последовательно выполняются следующие операции:

1. Вызывается обработчик события **On Validate**, позволяющий проверить допустимость записываемых данных
2. Если обработчик **OnValidate** не отказался от записи данных, то данные записываются в буфер записи
3. Если при записи данных не сгенерировано исключение, вызывается обработчик события **OnChange**

Обработчик события **OnChange** позволяет определить реакцию на данные, успешно записанные в буфер записи, например, известить об этом пользователя.

OnChange — событие класса TGraphicsObject

Событие наступает после изменения графического объекта.

Класс *TGraphicsObject*

Определение

```
typedef void __closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnChange
```

Описание

Обработчик события **OnChange** должен осуществить необходимые операции при изменении графического объекта и отразить его новые установки.

OnChange — событие меню

Событие наступает при изменении меню.

Классы *TMenu, TMainMenu, TPopupMenu*

Определение

```
typedef void__fastcall (__closure *TMenuChangeEvent)
          (System::TObject* Sender, TMenuItem* Source, bool Rebuild);

__property TMenuChangeEvent OnChange
```

Описание

Событие **OnChange** наступает при загрузке меню *в* память и при изменении структуры меню: добавлении или удалении разделов, изменении их свойств.

В обработчике события параметр **Sender** - - изменяемое меню, параметр Source — изменяемый раздел меню (если таковой имеется). Параметр **Rebuild** указывает, будет ли меню перестраиваться (например, при удалении или добавлении разделов).

OnChange — событие окон редактирования

Событие наступает, когда может быть изменился текст в окне редактирования.

Класс *TCustomEdit*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEven OnChange
```

Описание

Событие **OnChange** наступает, когда может быть изменился текст в окне редактирования. В момент обработки события свойство **Text** равно новому значению текста. Определить, изменен ли в действительности текст, можно по значению свойства **Modified**, получающему значение **true**, если текст изменен.

OnChange и OnChanging — события класса TCanvas

Событие происходит сразу после изменения изображения на канве.

Класс *TCanvas*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnChange
__property Classes::TNotifyEvent OnChanging
```

Описание

Событие **OnChange** наступает после изменения изображения на канве. При вызове любого метода рисования осуществляется следующая последовательность операций:

1. Наступает событие **OnChanging.**
2. Вызванный метод канвы **TCanvas** делает изменения в изображении.
3. Наступает событие **OnChange.**

Событие канвы OnChange наступает при изменении именно самого изображения, а не свойств канвы. Такие свойства канвы, как объекты **Font** — шрифт, **Brush** — кисть и Реп — перо имеют свои собственные события **OnChange.**

OnChanging — событие TCanvas

См. разд. «OnChange и OnChanging — события класса TCanvas».

OnClick

Событие соответствует щелчку мыши на компоненте и некоторым другим действиям пользователя.

Класс *TControl*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnClick
```

Описание

Обычно событие **OnClick** наступает, если пользователь нажал и отпустил основную кнопку мыши, когда указатель мыши находился на компоненте. Это событие происходит также, если:

- Пользователь выбрал элемент в таблице, дереве, списке, выпадающем списке, нажав клавишу со стрелкой.
- Пользователь нажал клавишу пробела, когда кнопка или индикатор были в фокусе.
- Пользователь нажал клавишу Enter, а активная форма имеет кнопку по умолчанию, указанную свойством **Default.**
- Пользователь нажал клавишу Esc, а активная форма имеет кнопку прерывания, указанную свойством **Cancel.**
- Пользователь нажал клавиши быстрого доступа к кнопке или индикатору. Например, если свойство **Caption** индикатора записано как "&Полужирный" и символ П подчеркнут, то нажатие пользователем комбинации клавиш Alt-П вызовет событие **OnClick** в этом индикаторе.
- Приложение установило в **true** свойство **Checked** радиокнопки **RadioButton.**
- Приложение изменило свойство **Checked** индикатора **CheckBox.**
- Вызван метод **Click** элемента меню.

Для формы событие **OnClick** наступает, если пользователь щелкнул на пустом месте формы или на недоступном компоненте.

Параметр обработчика **Sender** содержит объект, в котором произошло событие, и может использоваться для дифференцированной реакции на события в разных компонентах.

Пример

Один обработчик события **OnClick** может использовать для обработки событий в различных компонентах. Если при этом требуется различать, в каком компоненте произошло событие, можно использовать параметр **Sender,** как **в** приведенном чисто демонстрационном примере, отображающем сообщение о том, в каком компоненте произошло событие:

```
ShowMessage("OnClick в " + ((TComponent *)Sender)->Name);
```

OnCreate

Событие происходит при создании формы.

Класс *TCustomForm*

Определение

```
typedef void __closure *TNotifyEvent (System::TObject* Sender);

__property Classes::TNotifyEvent OnCreate;
```

Описание

Событие **OnCreate** возникает в момент создания формы и может использоваться для выполнения каких-то процедур настройки ее самой или содержащихся на ней компонентов. Если в обработчике этого события создаются какие-то объекты, они должны разрушаться, освобождая память, в обработчике события **OnDestroy.**

Последовательность событий при создании формы, имеющей значение свойства **Visible.** равное **true: OnCreate, OnShow, OnActivate, OnPaint.**

OnDataRequest — событие TCustomProvider

См. в гл. 4 разд. «DataRequest, OnDataRequest — методы и событие».

OnDblClick

Событие соответствует двойному щелчку мыши на компоненте.

Класс *TControl*

Определение

```
typedef void __closure *TNotifyEvent (System::TObject* Sender);

__property Classes::TNotifyEvent OnDblClick
```

Описание

Событие **OnDblClick** наступает, если пользователь осуществил двойной щелчок: дважды с коротким интервалом нажал и отпустил основную кнопку мыши, когда указатель мыши находился на компоненте. К одному и тому же компоненту нельзя написать обработчики событий **OnClick и OnDblClick,** поскольку первый из них всегда перехватит первый из щелчков.

Параметр **Sender** содержит объект, в котором произошло событие, и может использоваться для дифференцированной реакции на события в разных компонентах.

OnDeleteError

Событие, наступающее при генерации исключения в процессе удаления записи.

Класс *TDataSet*

Определение

```
enum TDataAction { daFail, daAbort, daRetry };
typedef void __fastcall (__closure *TDataSetErrorEvent)
                         (TDataSet* DataSet, EDatabaseError* E,
                         TDataAction &Action);

__property TDataSetErrorEvent OnDeleteError;
```

Описание

Событие **OnDeleteError** наступает, если была сделана попытка удалить запись и она закончилась неудачей — генерацией исключения. В обработчике события **OnDeleteError** можно предусмотреть действия, необходимые в этом случае.

Параметр **DataSet** указывает набор данных, из которого неудачно удалялась запись. Параметр E является указателем на объект исключения **EDatabaseError,** свойство **Message** которого (E.Message) содержит строку сообщения об ошибке. Впрочем, если уж отображать сообщение об ошибке, то лучше это делать на рус-

ском языке, заменив стандартное значение **Message.** Параметр **Action** типа **TDataAction** (объявлен в модуле *Comctrls),* передаваемый по ссылке, указывает характер действий после выхода из обработчика:

daFail	Прерывание операции, приведшей к ошибке, и отображение пользователю сообщения об ошибке
daAbort	Прерывание операции, приведшей к ошибке, без отображения пользователю сообщения об ошибке
daRetry	Повторение операции, приведшей к ошибке. Если обработчик возвращает это значение, то он должен принять меры к устранению причин, вызвавших ошибку

При обращении к обработчику значение **Action** равно **daFail.**

QnDragDrop

Событие наступает в момент отпускания перетаскиваемого компонента над данным компонентом

Класс *TControl*

Определение

```
typedef void __closure *TDragDropEvent) (System::TObject* Sender,
                                         System::TObject* Source,
                                         int X, int Y);
__property TDragDropEvent OnDragDrop
```

Описание

Событие **OnDragDrop** наступает в момент отпускания перетаскиваемого компонента над данным компонентом. В обработчике события надо описать, что в этот момент должно произойти. Параметр **Source** соответствует перетаскиваемому объекту, а параметр **Sender** — объекту, над которым объект был отпущен. Параметры X и Y содержат координаты позиции курсора мыши над компонентом в системе координат клиентской области этого компонента.

Примеры

1. Пусть на форме имеется несколько списков типа **TListBox** и вы хотите позволить пользователю перемещать строки из одного списка в другой. Это можно сделать следующим образом.

   Во всех списках задаются значения свойств **DragMode.** равные **dm Automatic.** Это обеспечивает автоматическое начало перетаскивания.

   Далее для всех списков пишется единый обработчик события **OnDragOver** вида:

```
void __fastcall TForm1::ListBox1DragOver(TObject *Sender,
                                         TObject *Source, int X, int Y,
                                         TDragState State, bool &Accept)
{
 if(Sender != Source)
  Accept = Source->ClassNameIs("TListBox");
 else Accept = false;
}
```

   В нем сначала проверяется, не являются ли данный компонент **(Sender)** и перетаскиваемый объект **(Source)** одним и тем же объектом. Это сделано, чтобы избежать перетаскивания информации внутри одного и того же списка. Если источник и приемник являются одним и тем же объектом, то срабатывает **else**

и параметр **Accept** становится равным **false,** запрещая прием информации. Если же это разные объекты, то **Accept** делается равным **true,** если источником является какой-то другой список (компонент класса **TListBox),** и равным **false,** если источник является объектом любого другого типа. Таким образом компонент сообщает, что готов принять информацию из любого другого списка.

Далее для всех списков пишется единый обработчик события **OnDragDrop** вида:

```
void__fastcall TForm1::ListBox1DragDrop(TObject *Sender,
                                        TObject *Source, int X, int Y)
{
 TListBox *S = (TListBox *)Source;
 ((TListBox*)Sender)->Items->Add(S->Items->Strings[S->ItemIndex]);
}
```

В этом обработчике первый оператор создает указатель S на объект класса **TListBox,** и передает в него ссылку на объект **Source,** воспринимаемый как объект **TListBox.** Это сделано просто для того, чтобы не повторять несколько раз в следующем операторе приведение типа **Source** к указателю на объект класса **TListBox.** А такое приведение типа необходимо по следующей причине. Параметр **Source** объявлен как указатель на объект класса **TObject.** Но в этом классе отсутствуют свойства **Items и ItemIndex.** Они имеются только в классе **TListBox.** Поэтому прежде, чем обратиться к этим свойствам, надо произвести соответствующее приведение типа **Source.**

Второй оператор обработчика заносит методом Add выделенную строку списка-источника S в список-приемник **Sender.**

Подобные обработчики событий позволяют пользователю перетаскивать строки между любыми имеющимися списками.

2. Если приведенный выше пример по каким-то соображениям желательно осуществлять не в автоматическом режиме, а в режиме ручного управления (например, перетаскивание возможно только при нажатой клавише Alt или в каком-то конкретном режиме работы приложения), то отличия в реализации примера заключаются в следующем.

   Во всех списках задаются значения свойств **DragMode,** равные **dmManual.** Это обеспечивает управление началом перетаскивания.

   Затем в списках задается обработчик события **OnMouseDown** вида:

```
void__fastcall TForm1::ListBox1MouseDown(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int У)
{
 if ((Button == mbLeft) && Shift.Contains(ssAlt))
   ListBox1->BeginDrag(false,5);
}
```

   В этом обработчике первое условие проверяет, нажата ли именно левая кнопка мыши, второе — нажатие клавиши Alt (можно задать какое-то другое условие, по которому нажатие кнопки мыши можно ассоциировать с началом перетаскивания). Затем методом **BeginDrag** начинается перетаскивание. Поскольку в параметре метода задано значение false, то перетаскивание в действительности начнется только после сдвига мыши на *5* пикселов.

   Все остальные обработчики событий не отличаются от указанных в предыдущем примере.

OnDragQver

Событие относится ко времени, в течение которого пользователь перемещает перетаскиваемый объект над компонентом.

Класс *TControl*

Определение

```
typedef void __closure *TDragOverEvent)
                                    (System::TObject* Sender,
                                     System::TObject* Source,
                                     int X, int Y, TDragState State,
                                     bool &Accept);
__property TDragOverEvent OnDragOver
```

Описание

Событие **OnDragOver** начинается в момент, когда перетаскиваемый объект пересек границу данного компонента и оказался внутри его контура. Заканчивается событие, когда объект, покидая компонент, пересек его границу. Обработчик события **OnDragOver** используется для того, чтобы дать сигнал о готовности компонента принять перетаскиваемый объект в случае, если пользователь отпустит его над данным компонентом. Если компонент готов принять объект, в обработчике надо задать значение параметра **Accept,** равное **true.** Впрочем, это значение по умолчанию равно **true,** так что его можно не задавать. Вообще в предельном случае обработчик может быть пустым, что будет означать готовность компонента принять любой объект. Но даже пустой обработчик нужен, так как иначе сообщения о приеме компонента приложение не получит.

Во время перетаскивания над компонентом объекта, который может быть принят, форма курсора мыши может изменяться, сигнализируя пользователю о готовности компонента принять объект. Чтобы это было так, надо до момента события **OnDragOver** (а обычно — во время проектирования) задать соответствующее значение свойства компонента **DragCursor.**

Параметр **Source** определяет перетаскиваемый объект, параметр **Sender** — сам компонент, параметры X и Y — координаты точки экрана в пикселах. Параметр State типа **TDragState** определяет состояние перетаскиваемого объекта по отношению к другим объектам. Возможны следующие состояния:

Значение	Описание
dsDragEnter	Курсор мыши входит в пределы компонента.
dsDragMove	Курсор мыши перемещается в пределах компонента.
dsDragLeave	Курсор мыши выходит за пределы компонента.

Пример

См. пример в описании события **OnDragDrop.**

OnEditError

Событие, наступающее при генерации исключения в процессе редактирования или вставки записи.

Класс *TDataSet*

Определение

```
enum TDataAction ( daFail, daAbort, daRetry };
typedef void___fastcall  (__closure *TDataSetErrorEvent)
                         (TDataSet* DataSet, EDatabaseError* E,
                          TDataAction &Action);
__property TDataSetErrorEvent OnEditError
```

Описание

Событие **OnEditError** наступает, если была сделана попытка редактирования или вставки записи и она закончилась неудачей — генерацией исключения. В обработчике события **OnEditError** можно предусмотреть действия, необходимые в этом случае.

Параметр **DataSet** — редактируемый набор данных. Параметр E — указатель на объект ошибки, содержащий сообщение сгенерированного исключения. Его свойство **E.Message** содержит сообщение об ошибке, а свойство **E.HelpContext** — номер темы контекстной справки. Пользуясь этими свойствами, можно отобразить пользователю сообщение об ошибке и какие-то рекомендации по его дальнейшим действиям.

Передаваемый по ссылке параметр **Action** указывает, как должен реагировать набор данных в ответ на произошедшую ошибку после завершения выполнения данного обработчика события. Параметр **Action** может иметь значения:

daFail	Прерывание операции, приведшей к ошибке, и отображение пользователю сообщения об ошибке
daAbort	Прерывание операции, приведшей к ошибке, без отображения пользователю сообщения об ошибке
daRetry	Повторение операции, приведшей к ошибке. Если обработчик возвращает это значение, то он должен принять меры к устранению причин, вызвавших ошибку

При обращении к обработчику значение **Action** равно **daFail.**

OnEndDrag

Событие наступает в момент прерывания или окончания перетаскивания компонента.

Класс *TControl*

Определение

```
typedef void (__closure *TEndDragEvent)
                        (System::TObject* Sender,
                         System::TObject* Target, int X, int Y);

__property TEndDragEvent OnEndDrag t;
```

Описание

Событие **OnEndDrag** наступает при любом окончании процесса перетаскивания компонента — успешном (компонент перетащен в приемник) или безуспешном (компонент отпущен над формой или компонентом, не способным его принять). Событие наступает в перетаскиваемом компоненте.

Обработка этого события не требуется для осуществления процесса перетаскивания. Соответствующий обработчик может быть написан, если требуется какое-то действие или сообщение, подтверждающее результат перетаскивания, или какая-то реакция в перетаскивавшемся компоненте.

Параметр **Sender** — это сам объект перетаскивания. Параметр **Target** — это компонент-приемник, если объект был им принят, или nil, если перетаскивание закончилось неудачей. Параметры X и Y — координаты экрана в пикселах.

Пример

```
void __fastcall TForm1::ListBox1EndDrag(TObject *Sender,
                                        TObject *Target, int X, int Y)
{
```

```
  if (Target == NULL)
   ShowMessage("Перенесение объекта " +
               ((TControl *)Sender)->Name + " прервано");
  else
   ShowMessage(((TControl *)Sender)->Name + " перенесен в " +
               ((TControl *)Target)->Name);
}
```

В этом примере просто отображается сообщение о результатах перетаскивания типа «Перенесение объекта ListBoxl прервано» или «ListBoxl перенесен в ListBox2». Но, конечно, аналогичным образом можно предусмотреть любые действия.

OnEnter

Событие наступает в момент получения элементом фокуса.

Класс *TWinControl*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnEnter
```

Описание

Событие **OnEnter** наступает в момент получения элементом фокуса. Это событие не наступает при переключениях между формами или между приложениями.

При переключениях между элементами, расположенными в разных контейнерах, например, на разных панелях, событие **OnEnter** сначала наступает для контейнера, а потом для содержащегося в нем элемента. Это противоположно последовательности событий **OnExit.** которые при переключении на компонент другого контейнера наступают сначала для компонента, а потом для контейнера.

Пример

Пусть форма имеет кнопку OK и групповую панель, включающую три радиокнопки. Пусть в начальный момент активна кнопка OK. Когда пользователь щелкнет на одной из радиокнопок, в кнопке OK наступит событие **OnExit.** затем наступит событие **OnEnter** групповой панели, и только затем наступит событие **OnEnter** той кнопки, на которой щелкнули. Если после этого пользователь щелкнет на кнопке OK, то сначала наступит событие **OnExit** радиокнопки, затем событие **OnExit** групповой панели, а затем событие **OnEnter** кнопки OK.

OnExit

Событие наступает в момент потери элементом фокуса.

Класс *TWinControl*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnExit
```

Описание

Событие **OnExit** наступает в момент потери элементом фокуса, в момент его переключения на другой элемент. Это событие не наступает при переключениях между формами или между приложениями.

Значение свойства **ActiveControl** изменяется прежде, чем происходит событие **OnExit.**

При переключениях между элементами, расположенными в разных контейнерах, например, на разных панелях, событие **OnExit** сначала наступает для элемента, а потом для содержащего его контейнера. Это противоположно последовательности

событий **OnEnter**, которые при переключении из другого контейнера на компонент данного контейнера наступают сначала для контейнера, а потом для компонента.

Пример, поясняющий последовательность событий при переключениях фокуса, приведен в разделе «OnEnter».

OnFilterRecord

Событие, наступающее каждый раз при перемещении на другую запись, если разрешена фильтрация.

Класс *TDataSet*

Определение

```
typedef void__fastcall (__closure *TFilterRecordEvent)
                                    (TDataSet* DataSet, bool &Accept);

__property TFilterRecordEvent OnFilterRecord
```

Описание

Событие **OnFilterRecord** наступает каждый раз при перемещении на другую запись, если разрешена фильтрация, т.е. если свойство **Filtered** установлено в **true.** Обработчик события **OnFilterRecord** может определить, должна ли данная запись быть видимой в приложении. Если должна, то параметр **Accept** надо установить в **true,** если не должна — в **false.** Параметр **DataSet** — набор данных, с которым ведется работа.

Обработчик события **OnFilterRecord** позволяет сформулировать сложные условия фильтрации, основанные на сравнении значений различных полей, которые невозможно занести в компактной форме в свойство **Filter.** Впрочем, если используется и **Filter,** и обработчик **OnFilterRecord,** то они действуют совместно и надо следить за тем, чтобы соответствующие условия фильтрации не противоречили друг другу.

Пример

В приведенном ниже обработчике события **OnFilterRecord** предполагается, что набор данных **Table1** имеет поле **Dep** — отдел, в котором работает сотрудник, и поле **Year_b** — год рождения. Приложение имеет выпадающий список **CBDep,** в котором пользователь задает отдел, и компоненты **TCSpinEdit** с именами **SEmin и SEmax,** в которых пользователь задает минимальный и максимальный год рождения. Подразумевается, что должны отфильтровываться записи сотрудников, работающих в указанном отделе, год рождения которых лежит в указанных пределах.

```
void__fastcall TForm1::Table1FilterRecord(
                                TDataSet *DataSet, bool &Accept)
{
 Accept = (Table1Dep->Value == CBDep->Text) &&
          (Table1Year_b->Value <= SEmax->Value) and
          (Table1Year_b->Value >= SEmin->Value);
}
```

Значение **Accept** получается равным **true,** если запись удовлетворяет поставленным условиям фильтрации.

Все это будет работать, если в компоненте **Table1** значение **Filtered = true.** Кроме того, надо обеспечить, чтобы при смене условий фильтрации (введенных пользователем значений) происходило бы новое событие **OnFilterRecord.** Этого можно добиться, если в соответствующих местах программы (в частности, в обработчике события **OnClick** компонента **CBDep)** вставить операторы:

```
Table1->Filtered = false;Table1->Filtered = true;
```

Они обеспечивают отключение и включение фильтрации. При включении и происходят события **OnFilterRecord.**

OnGetText

Событие наступает при обращении к свойствам DisplayText или Text.

Класс *TField*

Определение

```
typedef void__fastcall (_closure*TFieldGetTextEvent)
                              (TField* Sender, AnsiString &Text,
                              bool DisplayText);

__property TFieldGetTextEvent OnGetText
```

Описание

Обработчик события **OnGetText** позволяет изменить форму отображения значения поля **Value**, т.е. изменить свойства **DisplayText** и **Text**, которые обычно просто соответствуют свойству AsString. При этом можно применить какое-то форматирование отображения или вообще изменить отображаемые тексты.

Параметр **Sender** представляет собой объект поля **TField.** В параметр **Text** надо занести отображаемую строку текста. Параметр **DisplayText** показывает, будет ли строка **Text** использоваться только для отображения, или она будет использоваться для редактирования.

Пример

Следующий пример анализирует значение поля Sex (предполагается, что это булево поле, в котором **true** соответствует мужскому полу сотрудника, а **false** – женскому). Обработчик события **OnGetText** изменяет отображаемое значение соответственно на 'м' и 'ж'.

```
void__fastcall TForm1::Table1SexGetText(TField *Sender,
                                        AnsiString &Text, bool DisplayText)
{
 if (Sender->Value) Text = "м";
 else  Text = "ж";
}
```

OnKeyDown

Событие наступает при нажатии пользователем любой клавиши.

Класс *TWinControl*

Определение

```
enum Classes__1 {ssShift, ssAlt, ssCtrl, ssLeft, ssRight,
                 ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble> TShiftState;

typedef void (__closure *TKeyEvent) (System::TObject* Sender,
                                     Word &Key, Classes::TShiftState Shift);

__property TKeyEvent OnKeyDown
```

Описание

Событие **OnKeyDown** наступает, если компонент находится в фокусе, при нажатии пользователем любой клавиши, включая функциональные и вспомогательные, такие, как Shift, Alt и Ctrl.

В обработчик события передаются, кроме обычного параметра **Sender**, указывающего на компонент, в котором произошло событие, также параметры **Key** и **Shift.** Параметр Key определяет нажатую клавишу клавиатуры. Для не алфавитно-цифровых клавиш используется виртуальный код API Windows. Эти коды и способы проверки параметра Key см. во встроенной справке C++Builder или в ис-

точниках [2] и [3]. Коды не различают символы в верхнем и нижнем регистрах и не различают символы кириллицы и латинские.

Параметр Shift является множеством, которое может быть пустым или включать следующие элементы:

Элемент	Значение
ssShift	Нажата клавиша Shift
ssAlt	Нажата клавиша Alt
ssCtrl	Нажата клавиша Ctrl

Значения элементов Shift, соответствующие нажатиям кнопок мыши, в данном событии не используются.

Примеры

Пусть вы хотите распознать комбинацию клавиш Alt-X. Для этого вы можете написать оператор:

```
if((Key == 'X') ss (Shift.Contains(ssAlt)))
 ... ;
```

Реакцию на нажатие пользователем клавиши Enter можно оформить одним из следующих операторов:

```
if(Key == 13) ... ;
```

или

```
if(Key == 0x0D) ... ;
```

ИЛИ

```
if (Key == VK_RETURN) ... ;
```

OnKeyPress

Событие наступает при нажатии пользователем клавиши символа.

Класс *TWinControl*

Определение

```
typedef void (_closure *TKeyPressEvent)
                          (System::TObject* Sender, char &Key);

__property TKeyPressEvent OnKeyPress
```

Описание

Событие **OnKeyPress** наступает, если компонент находится в фокусе, при нажатии пользователем клавиши символа. Параметр Key в обработчике этого события имеет тип **Char** и соответствует символу нажатой клавиши. При этом различаются символы в верхнем и нижнем регистрах и символы кириллицы и латинские. Клавиши, не отражаемые в кодах ASCII (функциональные клавиши и такие, как Shift, Alt, Ctrl), не вызывают этого события. Поэтому нажатие таких комбинаций клавиш, как, например, Shift-A, генерирует только одно событие **OnKeyPress,** при котором параметр Key равен "A". Для того чтобы распознавать клавиши, не соответствующие символам, или комбинации клавиш, надо использовать обработчики событий **OnKeyDown** и **OnKeyUp**.

Следует отметить, что событие **OnKeyPress** заведомо наступает, если нажимается только клавиша символа или клавиша символа при нажатой клавише Shift. Если же клавиша символа нажимается одновременно с какой-то из вспомогательных клавиш, то событие **OnKeyPress** может не наступить (произойдут только со-

бытия **OnKeyDown** при нажатии и **OnKeyUp** при отпускании) или, если и наступит, то укажет на неверный символ. Например, при нажатой клавише Alt событие **OnKeyPress** при нажатии символьной клавиши не наступает. А при нажатой клавише Ctrl событие **OnKeyPress** при нажатии символьной клавиши наступает, но символ не распознается.

Поскольку параметр Key передается в обработчик как var, его можно изменять, передавая для дальнейшей стандартной обработки другой символ, как в приведенных ниже примерах.

Примеры

1. Пусть вы хотите, чтобы пользователь не мог вводить в окно редактирования **Edit1** какие-либо символы, кроме цифр. Это можно сделать, написав для **Edit1** следующий обработчик события **OnKeyPress:**

```
Set <char, '0', '9'> Dig;
Dig << '0' << '1' << '2' << '3' << '4' << '5'
    << '6' << '7' << '8' << '9';
if ( ! Dig.Contains(Key))
   Key = 0;
```

   Этот оператор трансформирует все нецифровые символы в нулевые, и они не будут отражаться в окне редактирования.

2. Приведенный ниже оператор обработчика события OnKeyPress переводит символы в верхний регистр, независимо от того, в каком регистре набрал их пользователь:

```
Key = UpCase(Key);
```

   Этот оператор действует только на латинские символы. Приведенный ниже аналогичный оператор действует и на латинский символы, и на символы кириллицы:

```
Key = AnsiUpperCase(Key)[1];
```

OnKeyUp

Событие наступает при отпускании пользователем любой клавиши.

Класс *TWinControl*

Определение

```
enum Classes__1 {ssShift, ssAlt, ssCtrl, ssLeft, ssRight,
                 ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble> TShiftState;

typedef void __fastcall (__closure *TKeyEvent)
                          (System::TObject* Sender, Word &Key,
                          Classes::TShiftState Shift);
__property TKeyEvent OnKeyUp
```

Описание

Событие **OnKeyUp** наступает, если компонент находится в фокусе, при отпускании пользователем любой ранее нажатой клавиши, включая функциональные и вспомогательные, такие, как Shift, Alt и Ctrl.

В обработчик события передаются, кроме обычного параметра **Sender,** указывающего на компонент, в котором произошло событие, также параметры Key и Shift. Параметр Key определяет клавишу клавиатуры, которая отпускается. Для не алфавитно-цифровых клавиш используются виртуальные коды API Windows. Эти коды и способы проверки параметра Key см. во встроенной справке C++Builder или в источниках [2] и [3]. Коды не различают символы в верхнем и нижнем регистрах и не различают символы кириллицы и латинские.

Параметр **Shift** является множеством, которое может быть пустым или включать следующие элементы:

Элемент	Значение
ssShift	Отпускается клавиша Shift
ssAlt	Отпускается клавиша Alt
ssCtrl	Отпускается клавиша Ctrl

Значения элементов Shift, соответствующих нажатиям кнопок мыши, в данном событии не используются.

Событие **OnKeyUp** наиболее удобно, чтобы распознавать нажатые клавиши и кнопки, особенно, комбинации клавиш. Надо только не забывать, что параметр Key имеет тип **word.** Поэтому для его распознавания надо распознавания надо использовать виртуальные коды клавиш. К тому же, надо учитывать, что виртуальный код не различает символы в верхнем и нижнем регистре и не реагирует на то, русский или английский язык включен в данный момент.

Примеры

Пусть вы хотите написать обработчик, который бы реагировал на нажатие клавиш Shift-Y. Проверить нажатые клавиши можно оператором:

```
if((Key == 'Y') && (Shift.Contains(ssShift))) ... ;
```

Если же вы напишете оператор

```
if (Key == 'Y') ...
```

то он будет реагировать и на "Y", и на "y", и даже на русские буквы "Н" и "н", которые обычно расположены на той же клавише, что и латинская "Y".

OnMouseDown и OnMouseUp

События наступают в моменты нажатия и отпускания пользователем клавиши мыши над компонентом.

Класс *TControl*

Определение

```
enum TMouseButton { mbLeft, mbRight, mbMiddle };

enum Classes__1 {ssShift, ssAlt, ssCtrl, ssLeft,
                 ssRight, ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble> TShiftState;

typedef void (__closure *TMouseEvent) (System::TObject* Sender,
                                       TMouseButton Button,
                                       Classes::TShiftState Shift,
                                       int X, int Y) ;

__property TMouseEvent OnMouseDown
__property TMouseEvent OnMouseUp;
```

Описание

Обработка событий **OnMouseDown и OnMouseUp** используется для операций, требуемых при нажатии и отпускании пользователем какой-нибудь кнопки мыши.

Если требуется различная *обработка* событий в зависимости от того, какая кнопка мыши нажата или какая нажата вспомогательная клавиша, можно анализировать параметры **Button и Shift.** Значения параметра **Button** определяют, какая кнопка мыши нажата: **mbLeft** — левая, **mbRight** — правая, **mbMiddle** — сред-

няя. Параметр **Shift** представляет собой множество, содержащее помимо обозначения нажатой кнопки еще и обозначения нажатых одновременно с этим вспомогательных клавиш Shift, Alt, Ctrl (соответствуют элементам множества **ssShift, ssAlt, ssCtrl).** Параметры X и Y определяют координаты указателя мыши в клиентской области компонента. Параметр **Sender** — указатель на компонент, в котором произошло событие.

Примеры

1. Обработчик события **OnMouseDown** может использоваться для начала процесса перетаскивания компонента, если вы решили задать какое-то дополнительное условие (например, проверка каких-то опций), по которому можно начинать перетаскивание. В этом случае в компоненте вы задаете свойство **DragMode.** равным **dmManual,** что обеспечивает управление началом перетаскивания. Обработчик события **OnMouseDown** может иметь вид:

```
void__fastcall TForm1::ListBox1MouseDown(TObject *Sender,
                  TMouseButton Button, TShiftState Shift, int X, int Y)
{
 if ((Button == mbLeft) &&
      <проверка какого-нибудь условия>)
 ListBox1->BeginDrag(false,5);
}
```

   В приведенной структуре **if** первое условие **(Button == mbLeft)** можно заменить эквивалентным ему условием, проверяющим параметр **Shift:**

```
if (Shift.Contains(ssLeft)) ...
```

2. Если вы пишете обработчик, описанный выше, но хотите, чтобы перетаскивание начиналось только в случае, когда пользователь нажал левую кнопку мыши при нажатой клавише Alt, оператор обработки может иметь вид:

```
if ((Button == mbLeft) ss Shift.Contains(ssAlt))
   ListBox1->BeginDrag(false,5);
```

OnMouseEnter, OnMouseLeave

События наступают в начале и конце прохождения курсора мыши над меткой.

Класс *TCustomLabel*

Определения

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnMouseEnter
__property Classes::TNotifyEvent OnMouseLeave
```

Описание

Событие **OnMouseEnter** наступает в начале, а событие **OnMouseLeave** — в конце прохождения курсора мыши над меткой. Обработчики этих событий могут использоваться, например, для изменения шрифта или цвета фона метки при прохождении над ней курсора мыши. Например, следующие обработчики меняют цвет надписи метки на красный при прохождении над ней курсора:

```
void__fastcall TForm1::Label1MouseEnter(TObject *Sender)
{
 Label1->Font->Color = clRed;
}

void __fastcall TForm1::Label1MouseLeave(TObject *Sender)
 (
 Label1->Font->Color = clBlack;
}
```

OnMouseMove

Событие наступает при перемещении курсора мыши над компонентом.

Класс *TControl*

Определение

```
enum Classes__1 {ssShift, ssAlt, ssCtrl, ssLeft,
                 ssRight, ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble> ShiftState;

typedef void (__closure *TMouseMoveEvent)
                                (System::TObject* Sender,
                                 Classes::TShiftState Shift,
                                 int X, int Y);

__property TMouseMoveEvent OnMouseMove
```

Описание

Обработчик события **OnMouseMove** пишется, если надо произвести какие-то операции при перемещении курсора мыши над компонентом.

Параметр Shift, являющийся множеством, содержит элементы, позволяющие определить, какие кнопки мыши и какие вспомогательные клавиши (Shift, Ctrl и Alt) нажаты в этот момент. Параметры X и Y определяют координаты указателя мыши в клиентской области компонента. Параметр **Sender** (источник события) — сам компонент.

Событие **OnMouseMove** возникает независимо от того, нажаты ли какие-то кнопки или клавиши. Правда, хотя это и не документировано в C++Builder, при нажатой левой кнопке мыши это событие, почему-то, не наблюдается.

Пример

```
if (Shift.Contains(ssAlt)) ...
```

Этот оператор проверяет, не нажата ли клавиша Alt во время перемещения курсора мыши над компонентом, и, если нажата, то предпринимаются какие-то действия.

OnMouseUp

См. раздел «OnMouseDown и OnMouseUp».

OnMouseWheel, OnMouseWheelUp, OnMouseWheelDown

События наступают при вращении колесика мыши.

Класс *TControl*. до C++Builder 6 — *TWinControl*

Определения

```
enum Classes__1 (ssShift, ssAlt, ssCtrl, ssLeft, ssRight,
                 ssMiddle, ssDouble};
typedef Set<Classes__1, ssShift, ssDouble>  TShiftState;
typedef void__fastcall (__closure *TMouseWheelEvent)
          (System::TObject* Sender, Classes:: TShiftState Shift,
           int WheelDelta, const Types::TPoint &MousePos,
           bool &Handled);

__property TMouseWheelEvent

typedef void__fastcall (__closure *TMouseWheelUpDownEvent)
          (System::TObject* Sender, Classes::TShiftState Shift,
           const Types::TPoint &MousePos, bool &Handled);
```

```
__property TMouseWheelUpDownEvent OnMouseWheelDown

typedef void __fastcall (__closure *TMouseWheelUpDownEvent)
          (System::TObject* Sender, Classes::TShiftState Shift,
           const Types::TPoint &MousePos, bool &Handled);

__property TMouseWheelUpDownEvent OnMouseWheelUp
```

Описание

События **OnMouseWheel, OnMouseWheelUp и OnMouseWheelDown** наступают при вращении колесика мыши, если, конечно, у вашей мыши есть колесико. Первым наступает событие **OnMouseWheel** при вращении колесика в любую сторону. Параметр **Sender** указывает на компонент, в котором произошло событие. Параметр Shift, являющийся множеством, содержит элементы, позволяющие определить, какие кнопки мыши и какие вспомогательные клавиши (Shift, Ctrl и Alt) нажаты в этот момент. Параметр **WheelDelta** показывает, сколько раз повернулось колесико. Это положительное число при вращении вверх и отрицательное — при вращении вниз. Параметр **MousePos** типа **TPoint** определяет позицию курсора мыши. А передаваемый по ссылке параметр **Handled** указывает, завершена ли обработка события. Если по окончании обработки задать **Handled = false,** то событие далее будет обрабатываться обработчиком родительского компонента. Во многих оконных компонентах: **Memo. RichEdit** и других заложена обработка события **OnMouseWheel** по умолчанию. Учтите, что эти обработчики по умолчанию будут срабатывать только в том случае, если в компоненте заданы полосы прокрутки (собственно, это обработчики не самих элементов, а полос прокрутки).

Поскольку в компонентах имеются собственные обработчики, писать свой обработчик надо только в тех случаях, когда требуется какая-то нестандартная реакция на вращение колесика.

Если обработчик события **OnMouseWheel** отсутствует или если в нем задано **Handled = false,** то в зависимости от направления вращения колесика наступает событие **OnMouseWheelUp** при вращении вверх, или **OnMouseWheelDown** при вращении вниз. Параметры обработчиков этих событий те же, что в событии **OnMouseWheel.** Только отсутствует параметр **WheelDelta.**

Примеры

Если вы зададите для формы обработчики

```
void __fastcall TForm1::FormMouseWheelDown(TObject *Sender,
          TShiftState Shift, TPoint &MousePos, bool &Handled)
{
 ScaleBy(100,101);
}

void __fastcall TForm1::FormMouseWheelUp(TObject *Sender,
          TShiftState Shift, TPoint &MousePos, bool SHandled)
1
 ScaleBy(101,100);
}
```

то при вращении колесика размеры всех компонентов формы будут плавно уменьшаться или увеличиваться. Того же результата можно добиться одним обработчиком события **OnMouseWheel:**

```
void __fastcall TForm1::FormMouseWheel(TObject *Sender,
                              TShiftState Shift, int WheelDelta,
                              TPoint &MousePos, bool &Handled)
{
 ScaleBy(100 + WheelDelta / 100,100);
 Handled = true;
}
```

OnNewRecord

Событие, наступающее при вставке в набор данных новой записи.

Класс *TDataSet*

Определение

```
typedef void__fastcall (__closure *TDataSetNotifyEvent)
(TDataSet* DataSet);

__property TDataSetNotifyEvent OnNewRecord
```

Описание

Событие **OnNewRecord** наступает при выполнении методов **Insert и Append.** осуществляющих вставку новой записи в набор данных **DataSet.** Обработчик этого события может предусматривать задание начальных значений полей записи или осуществлять каскадную вставку новых записей в связанные Наборы данных.

OnPaint

Событие наступает при получении сообщения Windows о необходимости перерисовать испорченное изображение.

Классы *TCustomForm, TPaintBox*

Определение

```
typedef void (__closure *TNotifyEvent) (System::TObject* Sender);

__property Classes::TNotifyEvent OnPaint
```

Описание

Событие **OnPaint** наступает, когда приходит сообщение Windows о необходимости перерисовать испорченное изображение. Изображение может испортиться из-за временного перекрытия данного окна другим окном того же или стороннего приложения. Обработчик данного события должен перерисовать изображение. При перерисовке изображения канвы **Canvas** можно использовать свойство **ClipRect.** которое указывает область канвы, внутри которой изображение испорчено.

Примеры

Если копия изображения, отображаемого на канве, хранится в компоненте **BitMap,** то обработчик события **OnPaint** для формы может иметь вид:

```
Canvas->Draw(0,0,Bitmap);
```

а для компонента **PaintBox**

```
PaintBox1->Canvas->Draw(0,0,Bitmap);
```

Более быстрая перерисовка получается при использовании свойства **ClipRect** канвы, например:

```
Canvas->CopyRect(Canvas->ClipRect,Bitmap->Canvas,
                 Canvas->ClipRect);
```

OnPostError

Событие, наступающее при генерации исключения в процессе пересылки в базу данных измененной записи.

Класс *TDataSet*

Определение

```
enum TDataAction { daFail, daAbort, daRetry };
typedef void__fastcall (__closure *TDataSetErrorEvent)
                        (TDataSet* DataSet, EDatabaseError* E,
```

```
                                TDataAction &Action);

__property TDataSetErrorEvent OnPostError
```

Описание

Событие **OnPostError** наступает, если была сделана попытка переслать методом **Post** в базу данных или в буфер кэша измененную запись и она закончилась неудачей — генерацией исключения. В обработчике события **OnPostError** можно предусмотреть действия, необходимые в этом случае.

Параметр **DataSet** — набор данных, с которым ведется работа. Параметр E — указатель на объект ошибки, содержащий сообщение сгенерированного исключения. Его свойство **E.Message** содержит сообщение об ошибке, а свойство **E.HelpContext** — номер темы контекстной справки. Пользуясь этими свойствами, можно отобразить пользователю сообщение об ошибке и какие-то рекомендации по его дальнейшим действиям.

Передаваемый по ссылке параметр **Action** указывает, как должен реагировать набор данных в ответ на произошедшую ошибку после завершения выполнения данного обработчика события. Параметр **Action** может иметь значения:

daFail	Прерывание операции, приведшей к ошибке, и отображение пользователю сообщения об ошибке
daAbort	Прерывание операции, приведшей к ошибке, без отображения пользователю сообщения об ошибке
daRetry	Повторение операции, приведшей к ошибке. Если обработчик возвращает это значение, то он должен принять меры к устранению причин, вызвавших ошибку

При обращении к обработчику значение Action равно **daFail.**

OnProgress

События происходят при медленных процессах изменения графического изображения и позволяют построить индикатор хода процесса.

Класс *TGraphic*

Определение

```
enum TProgressStage {psStarting, psRunning, psEnding};
typedef void (__closure *TProgressEvent) (System::TObject* Sender,
                                          TProgressStage Stage,
                                          Byte PercentDone,
                                          boolRedrawNow,
                                          const Windows::TRect &R,
                                          const AnsiString Msg);

__property TProgressEvent OnProgress
```

Описание

События **OnProgress** наступают во время медленных процессов изменения графического изображения таких, как загрузка, сохранение, трансформация. Эти события позволяют построить в приложении индикатор хода процесса, обеспечивающий обратную связь с пользователем. Разработчики новых компонентов могут генерировать события **OnProgress,** вызывая защищенный метод **Progress.**

Параметр Stage указывает стадию процесса: начало, продолжение, окончание. Он может соответственно принимать значения **psStarting, psRunning и psEnding.** Если приложение предусматривает индикацию процесса, то можно создавать индикатор при **Stage** = **psStarting,** изменять его показания, пока Stage = **psRunning**

и закрывать при Stage = psEnding. Параметр **PercentDone** показывает, какая примерно часть процесса выполнена. Этот параметр может использоваться в индикаторе процесса.

Параметр **RedrawNow** указывает, может ли изображение в данный момент быть успешно отображено на экране. Параметр R указывает область изображения, которая изменена и нуждается в перерисовке.

Параметр Msg содержит краткую справку о протекающем процессе. Например, "Loading", "Storing" или "Reducing colors". Строка Msg может быть и пустой.

OnReconcileError, OnUpdateError — события TCustomClientDataSet

Наступают при необходимости принятия решение относительно записи, вызвавшей ошибку при обновлении базы данных.

Класс *TCustomClientDataSet*

Объявления

```
enum TUpdateKind {ukModify, ukInsert, ukDelete};
enum TReconcileAction {raSkip, raAbort, raMerge,
                       raCorrect, raCancel, raRefresh};
typedef void __fastcall (__closure *TReconcileErrorEvent)
                                    (TCustomClientDataSet* DataSet,
                                     EReconcileError* E,
                                     Db::TUpdateKind UpdateKind,
                                     TReconcileAction &Action);

__property TReconcileErrorEvent OnReconcileError
__property TResolverErrorEvent OnUpdateError
```

Описание

События **OnUpdateError** возникают во внутреннем провайдере, а события **OnReconcileError** генерируются методом **Reconcile**, который, в свою очередь, вызывается методом **ApplyUpdates**, заносящим изменения в базу данных. Суть этих событий одна и та же. Если обнаруживаются записи, вызывающие ошибки при занесении их в базу данных, то для каждой такой записи генерируется соответствующее событие. Различие между **OnUpdateError и OnReconcileError** заключается в том, что события **OnUpdateError** возникают в процессе обновления базы данных, а события **OnReconcileError** — после. Так что события **OnReconcileError** возникают только для тех записей, для которых в обработчике **OnUpdateError** (если он вообще задан) конфликт не разрешился.

Следует отметить, что провайдер не может обнаружить конфликты в полях типа **Memo**. Так что при конфликтах в таких полях события не генерируются.

Параметр **DataSet** — это тот клиентский набор данных, записи которого вызвали ошибку. Если ошибка произошла во вспомогательном наборе данных, то **DataSet** указывает этот набор, хотя событие генерируется для головного набора данных. При обработке события можно использовать для разрешения конфликта такие свойства набора данных DataSet, как свойства полей **NewValue, OldValue, CurValue**. Однако в обработчике событий нельзя изменять текущую запись DataSet, применяя методы навигации (Next. **Prior и т.п.**).

Параметр E является указателем на объект типа **EReconcileError**. Он содержит информацию, которую можно использовать при обработке ошибки. Основные свойства объекта типа **EReconcileError:**

Код ошибки, возвращаемый BDE, ADO, dbExpress или иными технологиями:

```
__property Word ErrorCode
```

Код предыдущей ошибки в процессе их обработки (если ошибки не было — 0):

```
__property Word PreviousError
```

Дополнительная информация (может быть NULL):

```
__property AnsiString Context
```

Параметр UpdateKind указывает тип операции, вызвавшей ошибку: **ukModify** — изменение записи, **ukInsert** — вставка, **ukDelete** — удаление.

В параметр **Action** надо занести результат обработки ошибки:

raSkip	Пропустить обработку данной записи, т.е. оставить ее в списке незавершенных исправлений
raAbort	Прервать все операции обработки ошибок
raMerge	Объединить изменяемую запись с записями сервера (фактически, добавить данную запись как новую)
raCorrect	Заменить текущую обновляемую запись записью, сформированной в обработчике события (это случай, когда вы можете исправить ошибку)
raCancel	Отменить все исправления данной записи, вернувшись к первоначальным значениям всех ее полей
raRefresh	Отменить все исправления данной записи и заменить запись той, которая хранится на сервере

В обработчике события **OnReconcileError** надо предоставить пользователю всю доступную информацию о возникшей проблеме и дать возможность принять решение. Например, пользователь может изменить значение **NewValue** соответствующего поля набора **DataSet** (то значение, которое вызвало ошибку) и задать **Action = raCorrect.** В этом случае будет сделана попытка занести в базу данных исправленный вариант записи. Пользователь может выбрать **Action = raRefresh,** т.е. отказаться от сделанных исправлений и согласиться на ту запись, которая находится в данный момент в базе данных. Он может также выбрать **Action = raCancel,** т.е. отказаться от сделанных исправлений и принять те значения полей, которые были перед началом редактирования. Наконец, пользователь может отказаться временно от принятия решения **(Action = raSkip** или **raAbort)** и продолжить работу по редактированию записей.

Примеры

Ниже приведен пример фрагмента обработчика события **OnReconcileError,** в котором пользователю сообщается информация о возникшей проблеме. Информация заносится в окно **Memo1.**

```
void__fastcall TForm1::ClientDataSet1ReconcileError(
          TCustomClientDataSet *DataSet, EReconcileError *E,
          TUpdateKind UpdateKind, TReconcileAction &Action)
{
 Memo1->Clear();
 Memo1->Lines->Add("Код ошибки: " + IntToStr(E->ErrorCode));
 Memo1->Lines->Add("Прежний код: " +
                                IntToStr(E->PreviousError));
 Memo1->Lines->Add("Сообщение: " + E->Message);
 if (E->Context != "")
   Memo1->Lines->Add(E->Context);
 AnsiString S = "Операция: ";
 switch (UpdateKind)
 {
   case ukModify: S += "изменение записи";
                  break;
   case ukInsert: S += "вставка";
```

```
                    break;
      case ukDelete: S += "удаление";
    }
    Memo1->Lines->Add(S);
    if (UpdateKind == ukModify)
     for(int i = 0; i < DataSet->FieldCount; i++)
     {
      TField *F= DataSet->Fields->Fields[i];
      if ((F->FieldKind != fkData) ||
          VarIsEmpty(F->NewValue))  continue;
      Memo1->Lines->Add("");
      Memo1->Lines->Add("Поле " + F->FieldName);
      Memo1->Lines->Add("Прежнее значение: " +
                        VarToStr(F->OldValue));
      Memo1->Lines->Add("Записываемое значение " +
                        VarToStr(F->NewValue));
      if (! VarIsEmpty(F->CurValue))
        Memo1->Lines->Add("Значение в БД: " +
                          VarToStr(F->CurValue));
     }
    ...
}
```

Первые операторы приведенного кода заносят в **Memo1** коды ошибок, текст сообщения об ошибке и характер проводимой операции. Отметим, что типичные сообщения об ошибках желательно перевести на русский язык (в коде это не делается). Затем, в случае, если выполнялась операция модификации записи, организуется цикл по всем полям записи. Для тех полей, которые содержат измененные данные (тип поля **fkData** — данные, имеется значение **NewValue)** дается информация обо всех значениях этого поля.

Далее в обработчике должна быть предусмотрена возможность выбора пользователем дальнейших действий, как указано в описании события **OnReconcileError.**

В обработчике события **OnReconcileError** может быть успешно использован диалог Reconcile Error Dialog, содержащийся в Депозитарии на странице Dialogs. Он предоставляет пользователю подробную информацию о возникшей проблеме и дает полную возможность принять решение. Желательно только русифицировать его. Для этого достаточно перевести надписи меток на форме и определить следующим образом глобальные переменные:

```
char *ActionStr[] = {"Пропустить", "Прервать", "Объединить",
                     "Исправить", "Отменить", "Обновить"};
char *UpdateKindStr[] = {"Изменение", "Вставка", "Удаление"};
char *SCaption = "Ошибка ";
char *SUnchanged = "<Не изменено";
char *SBinary = "(двоичное)";
char *SFieldName = "Имя поля";
char *SOriginal = "Начальное значение";
char *SConflict = "Конфликтное значение";
char *SValue = "Значение";
char *SNoData = "<Нет данных>";
char *SNew = "Новое";
```

Если используется этот диалог, обработчик события **OnReconcileError** состоит всего из одного оператора:

```
Action = HandleReconcileError(this, DataSet, UpdateKind, E);
```

OnSetText

Событие наступает при задании свойству **Text** нового значения.

Класс *TField*

Определение

```
typedef void__fastcall (__closure *TFieldSetTextEvent)
                       (TField* Sender, const AnsiString Text);

__property TFieldSetTextEvent OnSetText
```

Описание

Обработчик события **OnSetText** позволяет отреагировать на изменение свойства **Text** — строки, используемой для ввода пользователем данных в режиме редактирования. Событие **OnSetText** генерируется после редактирования пользователем значения поля перед присвоением нового значения свойству **Value.** Параметр обработчика **Text** отображает в текстовом виде новое введенное пользователем значение поля. Это значение можно проверить и, если оно допустимо, то передать его в соответственно отформатированном виде в значение **Value.** Если такого присваивания **Value** не сделано, то введенное значение поля не перепишется в **Value.**

OnStartDrag

Событие наступает, когда пользователь начинает процесс перетаскивания компонента.

Класс *TControl*

Определение

```
typedef void (__closure *TStartDragEvent)
        ( System::TObject* Sender, TDragObject* &DragObject);

__property TStartDragEvent OnStartDrag
```

Описание

Событие **OnStartDrag** наступает, когда пользователь нажал левую кнопку мыши над объектом и, не отпуская ее, начал смещать курсор мыши, т.е. начал перетаскивание.

Обработчик события **OnStartDrag** позволяет описать какие-то специальные действия, необходимые при начале перетаскивания. Параметр **Sender** является тем компонентом, который должен перетаскиваться или который содержит объект будущего перетаскивания.

Переменная **DragObject** по умолчанию равняется NULL. Это означает, что переноситься будет сам компонент или его объект. При этом автоматически создается объект типа **TDragControlObject,** с которым C++Builder осуществляет весь процесс перетаскивания. Но обработчик может создавать и новый объект перетаскивания, в котором определен какой-нибудь особый вид курсора и т.п.

OnUpdateError — событие TBDEDataSet

Событие, наступающее при генерации исключения в процессе пересылки в базу кэшированных изменений.

Класс *TBDEDataSet*

Определение

```
enum TUpdateKind {ukModify, ukInsert, ukDelete};
enum TUpdateAction {uaFail, uaAbort, uaSkip, uaRetry, uaApplied};
typedef void__fastcall (__closure *TUpdateRecordEvent)
          (Db::TDataSet * DataSet, Db::TUpdateKind UpdateKind,
           TUpdateAction &UpdateAction);

__property TUpdateRecordEvent OnUpdateError
```

Описание

Событие **OnUpdateError** наступает, если была сделана попытка переслать в базу кэшированные изменения и она закончилась неудачей — генерацией исключения. Поскольку начало кэширования и пересылка изменений в базе данных разнесены во времени, то может оказаться, что другое приложение уже успело изменить записи, которые были кэшированы. В этом случае Borland Database Engine (BDE) осуществляет соответствующие проверки, и если они не привели к удовлетворительному результату, то генерируется исключение и наступает событие **OnUpdateError.** В обработчике этого события можно предусмотреть необходимые действия.

Параметр **DataSet** — набор данных, с которым ведется работа. Параметр E — указатель на объект ошибки, содержащий сообщение сгенерированного исключения. Его свойство **E.Message** содержит сообщение об ошибке. Пользуясь им, можно отобразить пользователю сообщение об ошибке и какие-то рекомендации по его дальнейшим действиям.

Значение параметра **UpdateKind** указывает, случилась ли ошибка во время вставки (значение **ukInsert**), удаления (значение **ukDelete**) или модификации (значение **ukModify**) записи.

Передаваемый по ссылке параметр **UpdateAction** указывает, как должен реагировать набор данных в ответ на произошедшую ошибку после завершения выполнения данного обработчика события. Параметр **UpdateAction** может иметь значения:

uaAbort	Прерывание операции, приведшей к ошибке, без отображения пользователю сообщения об ошибке
uaFail	Прерывание операции, приведшей к ошибке, и отображение пользователю сообщения об ошибке
daRetry	Повторение операции, приведшей к ошибке. Если обработчик возвращает это значение, то он должен принять меры к устранению причин, вызвавших ошибку
uaSkip	Пропуск пересылки текущей записи и оставление ее в кэше

При обращении к обработчику значение **UpdateAction** равно **daFail.**

Обработчик события может использовать свойства **TField::OldValue и TField::NewValue,** чтобы изменить условия сравнения и задать новое значение **TField::NewValue.** В этом случае надо повторить попытку пересылки, задав UpdateAction равным **uaRetry.**

Следует отметить, что если исключение сгенерировано методом **ApplyUpdates** и не перехвачено в блоке **try...catch,** то пользователю будет предъявлено окно с сообщением об ошибке. Если в этом случае обработчик события **OnUpdateError** оставил неизменным и равным **uaFail** значение **UpdateAction,** то окно с сообщением об ошибке будет отображаться дважды. Чтобы предотвратить это, надо установить **UpdateAction** равным **uaAbort.**

Обработчик события **OnUpdateError** не должен вызывать методов навигации, перемещающих курсор на другую запись.

OnUpdateError — событие TCustomClientDataSet

См. разд. «OnReconcileError, OnUpdateError — события TCustomClientDataSet».

OnUpdateRecord

Событие, наступающее при пересылке кэшированной записи в базу данных.

Класс *TBDEDataSet*

Определение

```
enum TUpdateKind {ukModify, ukInsert, ukDelete};
enum TUpdateAction {uaFail, uaAbort, uaSkip, uaRetry, uaApplied};
typedef void __fastcall (__closure *TUpdateRecordEvent)
                    (Db::TDataSet * DataSet, Db::TUpdateKind UpdateKind,
                     TUpdateAction &UpdateAction);

__property TUpdateRecordEvent OnUpdateRecord
```

Описание

Событие **OnUpdateRecord** наступает при пересылке в базу кэшированной записи методом **ApplyUpdates.** Обработчик этого события можно использовать, например, когда пересылка данной записи сопряжена с каскадом операций по вставке, удалению или модификации других записей. В обработчике события **OnUpdateRecord** можно также предусмотреть какой-то дополнительный контроль данных.

Параметр **DataSet** — набор данных, с которым ведется работа. Значение параметра **UpdateKind** указывает, случилась ли ошибка вр время вставки (значение uklnsert), удаления (значение **ukDelete)** или модификации (значение **ukModify)** записи.

Передаваемый по ссылке параметр **UpdateAction** указывает, какие действия должны выполняться после окончания выполнения обработчика события. Параметр **UpdateAction** может иметь значения:

uaAbort	Прерывание операции без отображения пользователю сообщения об ошибке
uaApplied	Завершение пересылки и удаление текущей записи из кэша
uaFail	Прерывание операции и отображение пользователю сообщения об ошибке
uaSkip	Пропуск пересылки текущей записи и оставление ее в кэше

При обращении к обработчику значение **UpdateAction** равно **daFail.** Если в обработчике не обнаружено препятствий к завершению пересылки, значение **UpdateAction** следует задать равным **uaApplied.**

Обработчик события **OnUpdateRecord** не должен вызывать методов навигации, перемещающих курсор на другую запись.

Альтернативой написания обработчика события **OnUpdateRecord** является использование для обработки пересылки клиентского набора данных и объекта типа **TUpdateSQL.** Это может обеспечить лучшее управление пересылкой.

OnValidate

Событие наступает перед занесением данных в буфер записи.

Класс *TField*

Определение

```
typedef void __fastcall (__closure *TFieldNotifyEvent)
                                                (TField* Sender);

__property TFieldNotifyEvent OnValidate
```

Описание

При занесении значения поля **Value** в буфер текущей записи последовательно выполняются следующие операции:

1. Вызывается обработчик события **OnValidate,** позволяющий проверить допустимость записываемых данных
2. Если обработчик **On Validate** не отказался от записи данных, то данные записываются в буфер записи
3. Если при записи данных не сгенерировано исключение, вызывается обработчик события **OnChange**

Обработчик события **OnValidate** позволяет проверить данные перед их записью в буфер. Если в результате проверки данные оказываются недопустимыми, в обработчике следует сгенерировать исключение, например, вызвать функцию **Abort.** Проверка в обработчике **OnValidate** может быть более полной, чем проводимая при вводе нового значения проверка отдельных символов с помощью **EditMask.** А если значение поля меняется программно, то проверка с помощью **Edit**Mask вообще отсутствует.

Дополнительные источники информации о C++Builder 6

Ниже приведены сведения о некоторых книгах и иных информационных материалах автора по C++Builder. Оперативную информацию о готовящихся к выпуску и вышедших книгах вы можете найти на сайте автора **http://delci.hl.ru** и сайте издательства **www.binom-press.ru**.

1. Архангельский А. Я. Программирование в C++Builder 6 — М: ЗАО «Издательство БИНОМ», 2002

Книга содержит методические и, частично, справочные материалы по C++Builder 6 и предшествующим версиям C++Builder 5 и 4. Рассмотрены такие возможности C++Builder, как построение кросс-платформенных приложений, технологии доступа к данным ADO, InterBase Express, dbExpress, компоненты -- серверы COM, технологии распределенных приложений COM, CORBA, MIDAS, новая методика диспетчеризации действий. Дается методика построения прикладных программ, реализующих текстовые и графические редакторы, мультипликацию и мультимедиа, работу с базами данных, создание отчетов, приложений для Интернет, распределенных приложений, клиентов и серверов.

Справочная часть книги содержит некоторые материалы по языку, функциям, типам и классам C++Builder, но, конечно, в значительно меньшем объеме, чем книга [2]. Приводятся там и некоторые справочные данные по классам, компонентам, их свойствам, методам, событиям. Но это материал несоизмерим по объему сведений с данной книгой. Зато, там дается методика построения приложений самого разного назначения, рассказывается о техники связи с базами данных, о построении распределенных приложений и т.п. Конечно, та и данная книги частично перекликаются. Но, мне кажется, что они в значительной степени дополняют друг друга. А объединить их в одну и совсем избежать повторов невозможно: подобная книга содержала бы около двух тысяч страниц. Ее технически невозможно издать, и пользоваться ею тоже было бы невозможно — уж очень она была бы увесистой.

2. Архангельский А. Я. Справочное пособие по C++Builder 6. Книга 1. Язык C++ — М: ЗАО «Издательство БИНОМ», 2002

В книге даются исчерпывающие справочные сведения по языку C++ в C++Builder 6: синтаксис языка, все операции и операторы, все типы данных. Подробно рассматривается работа с исключениями, с текстовыми и двоичными файлами, со строками разных типов, массивами, множествами, структурами, классами. Обсуждается обработка и генерация сообщений Windows. Рассматривается около 650 функций C, C++, API Windows, из них более 300 с подробными описаниями и примерами.

Рассматривается стандартная библиотека шаблонов STL: все типы контейнеров, итераторов, все алгоритмы и функции-объекты.

Представленный в книге справочный материал снабжен подробными комментариями и примерами, что позволяет читателю изучать его практически с нуля.

3. Серия справочных файлов «Русские справки по C++Builder 6»

Серия справок — это программный продукт, призванный оказать вам поддержку в процессе проектирования. Справки встраиваются в среду C++Builder командой Help | Customize в дополнение к англоязычной справке и в процессе проектирования при нажатии клавиши F1 вам предлагаются на выбор темы английских или русских справок. Русские справки — это не перевод с английского, а, скорее,

расширенный электронный вариант материалов данной книги и книг [1] и [2]. Так что они могут быть полезны не только тем, кто испытывает определенные сложности с английским, но и всем пользователям C++Builder, поскольку содержат иначе построенное и скомпонованное изложение справочных данных, иные примеры, в них устранен ряд ошибок англоязычных справок (надеюсь, не добавлено собственных ошибок).

В настоящее время серия включает в себя четыре справки: по C++ в C++Builder, по компонентам и классам C++Builder, по графикам и диаграммам TeeChart, по стандартной библиотеке STL C++. Число входов предметного указателя справок около 3000, а число страниц текста в несколько раз превышает объем данной книги. Намечен также выпуск дополнительных справок по Интернет, по методике проектирования, по развернутым и прокомментированным примерам и ряд других.

Достоинство справок по сравнению с книгами в том, что они обеспечивают оперативную помощь в среде разработки C++Builder, облегчают поиск нужной информации (в книгах это делать значительно сложнее), позволяют легко переносить примеры в свой проект. Да и стоят справки намного дешевле, чем книги. Но, конечно, справки не заменяют книг, хотя и содержат много материала, не поместившегося в книги.

Справки распространяются через Интернет по адресу: **http://lab18.ipu.rssi.ru/help2/.** Там вы найдете условия распространения, включающие бесплатную поддержку — каждые 3-4 месяца выходят дополнения к справкам, которые распространяются бесплатно тем, кто приобрел начальную версию.

Распространение справок через Интернет не означает, что вы обязательно должны иметь доступ в Интернет с домашнего компьютера и иметь собственный адрес e-mail. Достаточно, если доступ в Интернет и e-mail есть у вас на работе или у кого-то из ваших друзей и знакомых. Вы можете воспользоваться этими возможностями, а затем на дискетах перенести файлы на свой компьютер.

4. Архангельский А. Я. Решение типовых задач в C++Builder 6. — М: ЗАО «Издательство БИНОМ», 2003

Эта книга будет готова не раньше следующего года (в этом году, надеюсь, выйдет аналогичная книга по Delphi 6). Она рассчитана на подготовленных читателей, ознакомившихся с C++Builder, например, по перечисленным выше книгам. В ней рассматриваются вопросы, совершенно не затронутые в предыдущих книгах, или только упомянутые в них: решение типовых вычислительных задач (решение систем линейных и нелинейных уравнений, операции с матрицами, векторами и т.п.), ориентированное на особенности C++Builder 6; графики и диаграммы в C++Builder 6; разработка распределенных приложений; множество задач, связанных с работой в Интернет и интранет и многое другое. Значительная часть книги посвящена рассмотрению множества частных задач, с которыми приходится сталкиваться при разработке приложений и по которым задается множество вопросов на различных форумах в Интернет. Книга содержит немало примеров прикладных программ, многие из которых могут рассматриваться как законченные программные продукты. Так что вы просто можете использовать их в своей текущей работе. В книге содержатся указания по доработке этих приложений для ваших целей, так что вы можете на их основе создавать свои собственные программные средства.