Компонент **Delphi ListBox** это компонент, служащий для хранения и обработки текстовой информации. Каждая строка **Delphi ListBox** хранит строку данных в формате **String**.

|  |  |  |
| --- | --- | --- |
| http://www.delphi-manual.ru/ListBoxInForm.JPG |  | http://www.delphi-manual.ru/ListBoxInProgram.JPG |
| Компонент Delphi ListBox на этапе конструирования | Компонент Delphi ListBox на Форме в рабочей программе |

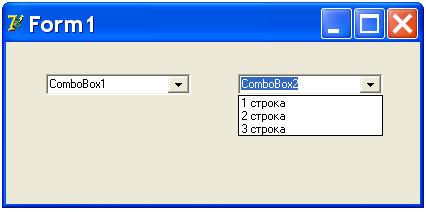
   Компонент ListBox - это массив строк. **ListBox** может загружать данные с диска, и сохранять информацию в файл. Также ListBox может сортировать строки. Доступ к строкам предоставляет свойство **Items** этого компонента. В соответствии с этим, вот как происходит обращение к строке с номером i:  
  
   ListBox1.Items[i-1]; // Минус один, так как нумерация строк в компоненте начинается с нуля.  
  
   Прежде всего рассмотрим методы добавления строк в компонент Delphi ListBox:

1. Считывание из файла;
2. Добавление строки в конец списка;
3. Добавление строки перед строкой с номером i;
4. Добавление строк на этапе конструирования.

**Методы добавления строк в компонент Delphi ListBox**

|  |  |
| --- | --- |
| **Считывание из файла** | Компонент Delphi ListBox может обращаться напрямую к текстовому файлу как для считывания информации из файла, так и для сохранения всех своих строк в текстовый файл. Каждая запись в файле будет записана в виде одной строки компонента, и при сохранении каждая строка будет сохранена в виде одной записи файла:    ListBox1.Items.LoadFromFile('Имя\_Файла');  // Процедура считывания из файла    ListBox1.Items.SaveToFile('Имя\_Файла');  // Процедура записи в файл |
| **Добавление строки в конец списка** | Программист может последовательно добавлять строки в компонент, не заботясь об их нумерации, и они будут размещаться в конец списка:    ListBox1.Items.Add('Новая\_Строка'); //Добавление строк в конец списка Так как количество строк в компоненте равно **ListBox1.Items.Count**, то новая (последняя) строка имеет номер **ListBox1.Items.Count-1**. Это объясняется тем, что нумерация строк начинается от 0. |
| **Добавление строки перед строкой с номером i** | Программист может разместить новую строку среди существующих строк там, где ему необходимо. Для этого нужно воспользоваться методом **Insert**, и указать номер строки, перед которой необходимо записать новую строку:    ListBox1.Items.Insert(i, 'Новая\_Строка');  // Добавление строки перед строкой с номером **i** При этом новая строка получает номер **i**. |
| **Добавление строк на этапе конструирования** | В Инспекторе Объектов зайдите на строчку **Items** и нажмите на появившуся кнопку. Появится окно редактора содержимого, где и можно внести нужную информацию. После нажатия кнопки **OK** содержимое редактора появится в компоненте ListBox на Форме. |

Пользователь может выделить строку, щёлкнув по ней мышкой. Номер выделенной строки возвращает свойство компонента **ListBox.ItemIndex**. То есть, получить текст выделенной строки можно так:  
  
   S := ListBox1.Items[ListBox1.ItemIndex];  
  
Не забывайте пользоваться оператором присоединения **with**. С его использованием это присваивание выглядит так:  
  
**with** ListBox1 **do**  
  **begin**  
    S := Items[ItemIndex];  
  **end**;   
  
   При многократном использовании свойств компонента данная запись очень экономит силы. Кстати, оператор присоединения допускает работу с несколькими компонентами, которые нужно разделить запятой. Например, присваивание строки компонента ListBox ячейке таблицы StringGrid выглядит так:  
  
**with** ListBox1, StringGrid1 **do**  
  **begin**  
    Cells[i, j] := Items[ItemIndex];  
  **end**;   
  
Для удаления строк из компонента применяется метод **Delete**. Удаление строки с номером **i**:  
  
   ListBox1.Items.Delete(i);  
  
   Чтобы при добавлении каждая строка была отсортирована (сортировка идёт только по возрастанию), нужно установить:  
  
   ListBox1.Sorted := True;  
  
Также возможно установить это свойство на этапе конструирования в Инспекторе Объектов.  
  
   К сожалению, компонент не обладает возможностью добавлять только уникальные данные. Один и тот же текст запишется несколько раз. Для реализации этой возможности приходится дополнительно применять невизуальный аналог компонента ListBox - объект [StringList](http://www.delphi-manual.ru/stringlist.php" \o "описание объекта StringList), обладающий такой возможностью:  
  
**procedure** TForm1.Button1Click(Sender: TObject);  
**var** SL: TStringList;  
S: String;  
**begin**  
SL:=TStringList.Create; // *Создаём список строк типа TStringList*  
SL.Sorted:=True; // *Устанавливаем свойство - сортировать строки*  
SL.Duplicates:=dupError; // *Устанавливаем свойство - дубликаты вызывают ошибку добавления*  
**try**  
  SL.Add(S); // *Пробуем добавить строку в объект StringList*  
  ListBox1.Items.Add(S); // *Добавляем строку в компонент ListBox*  
**except**  
  Caption:='Попытка добавить дубликат';  
  // *Попытка добавить дубликат вызовет ошибку в первой строке секции****try/except****,*  
  // *прерывание и переход к секции****except/end****.*  
  // *Добавления в ListBox не будет.*  
**end**;  
**end**;  
  
   Компонент ListBox автоматически добавит полосу прокрутки, если количество строк не помещается по высоте компонента. Высота каждой строки равна **ItemHeight**. Если нужно чтобы при добавлении новой строки полоса прокрутки точно не возникла, этот размер нужно добавить к высоте компонента. Но делать это нужно только начиная с определённого количества строк. Поэтому нужно делать проверку:  
  
ListBox1.Items.Add('Новая\_строка');  
**with** ListBox1 **do**  
  **if** (Items.Count > (Height/ItemHeight))  
    **then** Height := Height + ItemHeight;   
  
   Но если высота компонента не нацело делится на высоту строки, то прокрутка всё равно возникнет. Поэтому нужно отнять единичку:  
  
ListBox1.Items.Add('Новая\_строка');  
**with** ListBox1 **do**  
  **if** (Items.Count > (Height/ItemHeight)**-1**)  
    **then** Height := Height + ItemHeight;   
  
   Однако на долю секунды полоса прокрутки всё-таки появляется. Поэтому сначала сделаем проверку, а затем уже будем добавлять строку. В этом случае уже нужно отнимать не 1, а 2:  
  
**with** ListBox1 **do**  
  **if** (Items.Count > (Height/ItemHeight)**-2**)  
    **then** Height := Height + ItemHeight;  
ListBox1.Items.Add('Новая\_строка');

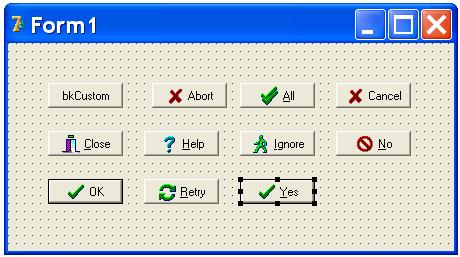
Компонент **Delphi ComboBox** представляет собой комбинацию списка строк [ListBox](http://www.delphi-manual.ru/listbox.php) со строкой ввода [Edit](http://www.delphi-manual.ru/edit.php). При этом "список строк" компонента **Delphi ComboBox** вначале скрыт, и раскрывается при щелчке мышкой по треугольничку раскрытия, который находится справа в строке ввода:  
  
  
  
   Таким образом, с помощью **Delphi ComboBox** место на Форме экономится для расмещения других элементов интерфейса программы. А при необходимости раскрытие списка строк можно вообще запретить.  
  
   Многие свойства и возможности компонента **Delphi ComboBox** по работе со строками (объектами**Items**) такие же как и у компонента [Delphi ListBox](http://www.delphi-manual.ru/listbox.php), поэтому повторно описывать их не буду. Опишу отличия и дополнительные возможности.  
  
   Итак, за возможность раскрытия компонента ComboBox отвечает стиль - свойство **Style**:

|  |  |
| --- | --- |
| http://www.delphi-manual.ru/ComboStyle.JPG | Стиль компонента ComboBox1 равен   ComboBox1.Style = csDropDown (список может быть раскрыт, можно писать в строке ввода)  Стиль компонента ComboBox2 равен   ComboBox1.Style = csSimple (список не может быть раскрыт, можно писать в строке ввода)  При стиле**csDropDownList**компонент может быть раскрыт, но свойство **Text**может принимать значения только одной из строк, сохранённых в компоненте, то есть пользователь лишён возможности писать в строке ввода. |

   Количество строк, видимых при раскрытии списка, равно **DropDownCount**. Если реальное количество сохранённых строк больше этого количества, то автоматически появляется полоса прокрутки. Программист может заставить список раскрыться в нужный момент. Для этого нужно свойству **DroppedDown** присвоить значение **True**. Для этой же цели пользователь может использовать клавиатурную комбинацию **ALT+ВНИЗ.**  
  
   Максимально допустимое количество символов в текстовой строке задаётся параметром**MaxLength**, причём значение **0** означает отсутствие ограничений. Свойство **CharCase** управляет преобразованием вводимого текста к верхнему (значение CharCase = **ecUpperCase**) или нижнему (значение CharCase = **ecLowerCase**) регистру. Значение **ecNormal** означает, что текст вводится без преобразования.  
  
   Основными событиями компонента **Delphi ComboBox** являются:

|  |  |
| --- | --- |
| **Событие** | **Условие генерации** |
| **OnChange** | Изменился текст в строке ввода. |
| **OnDropDown** | Список раскрывается. Это событие нужно обработать, если содержимое списка может изменяться во время работы программы. Тогда в обработчике этого события можно заново сформировать содержимое списка. |

 Компонент **Delphi BitBtn** это пиктографическая кнопка, представляющая собой один из вариантов стандартной кнопки **Button**. В отличие от последней кнопка **Delphi BitBtn** умеет отображать на своей поверхности не только надпись, но и изображение, которое задаётся свойством **Glyph**. В состав Delphi входит большое количество готовых изображений, которые можно размещать на кнопках **Delphi BitBtn**. В стандартном случае они располагаются по адресу C:\Program Files\Common Files\Borland Shared\Images\Buttons.  
  
   Свойство ***Kind*** компонента **Delphi BitBtn** определяет один из стандартных вариантов кнопки BitBtn:



   Нажатие на любую из них (кроме кнопок со свойством **BitBtn.Kind**, равным *bkCustom* или *bkHelp*) приводит к закрытию модального окна, в котором она установлена, а результат, возвращаемый в программу равен *mrName* (*Name* - Yes, Abort, Cancel и т.д. - название кнопки). Отличие есть у кнопки Close, её нажатие (после закрытия модального окна) приводит к возвращению не *mrClose* а*mrCancel*, а для главное окно программы просто закрывается, поэтому работа программы сразу завершается.  
   В случае если пиктограмма ***Glyph*** у кнопки *Delphi BitBtn* была изменена, система Delphi автоматически изменит и её значение *BitBtn.Kind*, которое станет равным *bkCustom*  
  
Специфические свойства компонента - пиктографической кнопки Delphi BitBtn перечислены в таблице:

|  |  |
| --- | --- |
| **Glyph**: TBitmap | Определяет рисунок, который может содержать от одного до четырёх изображений, связанных с состояниями, которые может принимать кнопка. |
| **Kind** | Определяет одну из разновидностей кнопки BitBtn (см. рисунок вверху). |
| **Layout** | Определяет к какому краю кнопки будет прижиматься пиктограмма:   * blGlyphLeft - к левому; * blGlyphRight - к правому; * blGlyphBottom - к нижнему; * blGlyphTop - к верхнему. |
| **Margin** | Определяет расстояние в пикселах, на котором будет располагаться пиктограмма от края кнопки. |
| **NumGlyph** | Определяет сколько растровых изображений размещается в файле, содержащем пиктограмму (**\***см. ниже). |
| **Spacing** | Определяет расстояние в пикселах от рисунка до текста на кнопке. |
| **Style** | Определяет как выглядит кнопка в зависимости от стиля предпочтительной операционной системы. Значение *bsNew* соответствует современным вариантам*Windows*. Стиль *bsAutoDetect* приводит оформление кнопки к стилю операционной системы, управляющей работой программы в настоящий момент. |

   Рисунок с изображениями для кнопки может состоять из нескольких (до 4-х) частей. Стандартная пиктограмма имеет размер 16х16 пикселов. Соответственно, если рисунок состоит из 2-х частей (для активного или неактивного состояния кнопки - именно такие изображения содержатся в стандартной поставке Delphi), то рисунок будет иметь размер 32х16. Количество пиктограмм задаётся свойством *Glyph*. Однако, если изображение состоит из нескольких квадратных пиктограмм одного размера, система *Delphi* сама распознает их количество.  
   Если в растре задана только одна пиктограмма, то Delphi сама может менять её изображение: в состоянии "нажата" изображение на кнопке смещается на 1 пиксель вниз-вправо, а в состоянии "недоступная" (*Enabled=False*) все цвета, кроме чёрного, меняются на светло-серый, а чёрный - на белый, что создаёт эффект вдавленности.