**Работа с MS WORD.** [**Статьи - Автор Василий Нестеров**](mailto:archer_ua@svitonline.com)

Предположим, у нас уже открыт файл. Вопросы открытия и сохранения документов уже были в других статьях,   
так что подробно на этом останавливаться не будем. Просто по ходу дела будет приведено то,   
чего раньше не встречалось - выход из документа без сохрания изменений. Как-то забыл, извините:)   
***Текст***  
Сначала о самом простом - добавлении в документ Word нужной строки текста.   
Поместим на форму компоненты **WordDocument**, **WordApplication**и **WordParagraphFormat**   
с палитры **Servers**. Нас интересуют в первую очередь свойство **Range** компонента **WordDocument**   
и свойство **Selection** компонента **WordApplication**. Классики утверждают, что они являются ссылкой на объекты  
**Range** и **Selection**. **Range** представляет из себя, проще говоря, кусок текста, это может быть   
как весь текст документа, так и любая его часть. Его пределы задаются двумя (или меньше)   
параметрами типа **OleVariant**.   
Например:   
var range1, range2, range3, a, b : OleVariant;  
...  
range1:=WordDocument1.Range;  
a:=5;  
b:=15;  
range2:=WordDocument1.Range(a,b);  
range3:=WordDocument1.Range(a);   
Первый наш объект включает в себя весь текст документа, у второго мы ограничили пределы 5-м и 15-м символами,   
третий представляет из себя весь последующий текст документа, начиная с 5-го символа. Объект имеет несколько полезных методов, например, с его помощью можем добавить текст в документ:   
range2.InsertAfter( #996600">'MS Word');   
Это мы вставили текст после выделенного **Range**. Точно также можем вставить текст и перед ним, для этого служит метод **InsertBefore()**. Текст, заключенный в объекте **Range**, можем получить так:   
WordDocument1.Range(a,b).Text;   
Кроме того, с помощью Range можем изменить шрифт в пределах объекта. Пример:   
a:=5;  
b:=15;  
WordDocument1.Range(a,b).Font.Bold:=1;  
WordDocument1.Range(a,b).Font.Size:=14;  
WordDocument1.Range(a,b).Font.Color:=clRed;   
Если хотим отменить выделение жирным шрифтом, присваиваем 0. Аналогично можно сделать шрифт курсивом, подчеркнутым - наберите **WordDocument1.Range.Font.**, и среда сама подскажет, какие могут быть варианты. Методы Select,  
Cut, Copy и Paste работают как в обычном тексте. С помощью Paste можем на место выбранного **Range** вставить не только строки, но и рисунок, находящийся в буфере обмена.   
WordDocument1.Range(a,b).Select;  
WordDocument1.Range(a,b).Cut;  
WordDocument1.Range(a,b).Copy;  
WordDocument1.Range(a,b).Paste;   
С помощью Range можем найти в документе нужную строку. Пусть в тексте содержится слово "picture". Например, нам на его место надо будет вставить рисунок.   
var a, b, vstart, vend: OleVariant;  
      j, ilengy: Integer;  
...  
ilengy:=Length(WordDocument1.Range.Text);  
for j:=0 to ilengy-8 do begin  
  a:=j;  
  b:=j+7;  
  if WordDocument1.Range(a,b).Text='picture' then begin  
   vstart:=j;  
   vend:=j+7;  
  end;  
end;  
WordDocument1.Range(vstart,vend).Select;  
Такая процедура находит и выделяет нужный кусок текста.  
Теперь про **Selection**, представляющий из себя выделенный фрагмент документа. Если выделения нет, это текущая позиция курсора в документе. С его помощью можем вставить что-либо на место выделенного фрагмента, сделать выравнивание, изменить шрифт.   
Он также имеет методы **InsertAfter() и InsertBefore()**:   
WordApplication1.Selection.InsertAfter("text1");  
WordApplication1.Selection.InsertBefore("text2");  
Форматирование выделенного текста происходит аналогично Range, например:   
WordApplication1.Selection.Font.Bold:=1;  
WordApplication1.Selection.Font.Size:=16;  
WordApplication1.Selection.Font.Color:=clGreen;  
Для выравнивания проще воспользоваться компонентом **WordParagraphFormat**. Сначала только нужно "подключить" его к выделенному фрагменту текста:   
WordParagraphFormat1.ConnectTo(WordApplication1.Selection.ParagraphFormat);  
WordParagraphFormat1.Alignment:=wdAlignParagraphCenter;   
Значения его свойства **Alignment** может принимать значения **wdAlignParagraphCenter, wdAlignParagraphLeft, wdAlignParagraphRight**, смысл которых очевиден. Имеются и методы Cut, Copy и Paste, которые в пояснениях вряд ли нуждаются:   
WordApplication1.Selection.Cut;  
WordApplication1.Selection.Copy;  
WordApplication1.Selection.Paste;   
Убираем выделение с помощью метода Collapse. При этом необходимо указать, в какую сторону сместится курсор, будет ли он до ранее выделенного фрагмента или после:   
var vcol: OleVariant;  
...  
vcol:=wdCollapseStart;  
WordApplication1.Selection.Collapse(vcol);  
При этом выделение пропадет, а курсор займет позицию перед фрагментом текста. Если присвоить переменной значение wdCollapseEnd, то курсор переместится назад. Можно просто поставить в скобках "пустышку":   
WordApplication1.Selection.Collapse(EmptyParam);   
Тогда свертывание выделения производится по умолчанию, к началу выделенного текста.   
***Рисунки***  
Логично было бы предположить, что рисунки документа будут представлять из себя коллекцию, аналогичную таблицам, и мы, обратившись к конкретной картинке, сможем менять ее свойства - обтекание, размер и т.д. Однако ничего подобного в WordDocument не обнаруживается. Потому возможности управления встраиваемыми в документ изображениями сильно ограничены.   
Простейший метод вставить в документ рисунок - по упомянутым причинам он же и единственный - скопировать его в Word из буфера обмена. Предположим, рисунок у нас находится в компоненте DBImage. Сначала нужно загнать его в буфер обмена:   
Clipboard.Assign(DBImage1.Picture);   
Теперь для его вставки следует воспользоваться методом Paste объектов Range или Selection: **WordApplication1.Selection.Paste** или **WordDocument1.Range(a,b).Paste**. Оставить для рисунка достаточное количество пустых строк и попасть в нужное место - это уже наша забота. Если он попадет посреди текста, вид будет довольно противный - при такой вставке обтекание текстом рисунка происходит как-то странно. Можно приготовить для отчета шаблон, где заменяем рисунком какое-либо ключевое слово. О том, как найти в документе нужный текст, см. выше.   
А теперь о несколько ином способе вставки рисунка, который устраняет проблемы с обтеканием и дает нам возможность перемещать его по документу, масштабировать и задавать отступы между рисунком и текстом. Способ, собственно, тот же - копируем из буфера обмена, но не прямо в документ, а в "рамку" - текстовую вставку. В ней может находиться не только текст, но и картинка, чем и воспользуемся.   
"Рамки" образуют коллекцию **Frames**, нумеруются целым индексом, пробегающим значения от 1 до **WordDocument1.Frames.Count**. Добавим в документ рамку, изменим ее размер и вставим рисунок:   
Clipboard.Assign(DBImage1.Picture);  
vstart:=1;  
vend:=2;  
WordDocument1.Frames.Add(WordDocument1.Range(vstart,vend));  
i:=1;  
WordDocument1.Frames.Item(i).Height:=DBImage1.Height;  
WordDocument1.Frames.Item(i).Width:=DBImage1.Width;  
WordDocument1.Frames.Item(i).Select;  
WordApplication1.Selection.Paste;  
Здесь для простоты предполагается, что размер **DBImage** равен размеру самой картинки, а также что до этого рамок у нас в документе не было. Обратить внимание следует на несколько моментов. Размер рамки надо задавать до того, как копировать в нее рисунок. Иначе она будет иметь размер по умолчанию, под который замасштабируется и наша картинка. При попытке изменить размер рамки задним числом размер картинки уже не изменится. Кроме того, параметр Range при добавлении рамки часто никакой роли не играет. Рамка изначально все равно появится в левом верхнем углу документа, а указанный кусок текста при этом не пострадает. Но это только в том случае, если он не выделен. Если в документе есть выделение, рамка появится вместо выделенного фрагмента. Таким образом можем ее вставить в нужное место взамен какого-то ключевого слова.   
При желании можем ее подвигать в документе и "вручную". Для этого служат свойства горизонтального и вертикального позиционирования, которые задают ее отступ от левого верхнего "угла" документа:   
i:=1;  
WordDocument1.Frames.Item(i).VerticalPosition:=30;  
WordDocument1.Frames.Item(i).HorizontalPosition:=50;   
Отступ между краями рамки и текстом задается следующим образом:   
WordDocument1.Frames.Item(i).HorizontalDistanceFromText:=10;  
WordDocument1.Frames.Item(i).VerticalDistanceFromText:=10;   
А теперь о масштабировании. Для этого достаточно длину и ширину рамки умножить на одно и то же число. Например:   
WordDocument1.Frames.Item(i).Height:=DBImage1.Height\*1.5;  
WordDocument1.Frames.Item(i).Width:=DBImage1.Width\*1.5;  
При этом наша картинка в полтора раза пропорционально растянется. Точно также можно и уменьшить, но делить, как и множить, следует на одно число. Растягивать длину и ширину по-разному у меня лично не получалось. Задавать размер опять-таки надо еще до вставки рисунка. Ну и, наконец, удаление рамки:   
WordDocument1.Frames.Item(i).Delete;   
***Списки***  
 Списки в документе образуют коллекцию **Lists**, к отдельному списку обращаемся **WordDocument1.Lists.Item(i)**, где i целое число от 1 до **WordDocument1.Lists.Count** ... на этом все. Нет методов, позволяющих не то что создать новый список, а даже добавить пункт к уже существующему. Ничего страшного, настоящие герои всегда идут в обход:)) Сейчас мы все же проделаем и то, и другое. Все что нам понадобится - свойство Range отдельного списка, то есть его текст без разделения на пункты, а также возможность его выделить:   
WordDocument1.Lists.Item(i).Range.Select;  
Для этого в любом случае потребуется заготовка. Неважно, вставлена она в общий шаблонный документ или хранится в отдельном файле. Заготовку делаем так: выбираем в меню Формат/Список, и сохраняем, если это отдельный шаблон списка. У нас появляется пустой список без текста с одним маркером. Далее вспоминаем, как мы делали списки вручную - писали текст, нажимали "Enter", появлялся новый элемент списка. Теперь то же самое, только программно. Предположим, у нас уже открыт документ с заготовкой, и мы хотим внести в список пункты "Item 1" и "Item 2":   
var i: Integer;  
      vcol: OleVariant;  
...  
i:=1;  
vcol:=wdCollapseEnd;  
WordDocument1.Lists.Item(i).Range.Select;  
WordApplication1.Selection.Collapse(vcol);  
WordApplication1.Selection.InsertAfter('Item 1');  
WordDocument1.Lists.Item(i).Range.Select;  
WordApplication1.Selection.Collapse(vcol);  
WordApplication1.Selection.InsertAfter(#13);  
WordDocument1.Lists.Item(i).Range.Select;  
WordApplication1.Selection.Collapse(vcol);  
WordApplication1.Selection.InsertAfter('Item 2');  
WordDocument1.Lists.Items(i).Range.Select;  
WordApplication1.Selection.Copy;   
То есть мы вставляем в документ текст первого пункта списка, он попадает на свое место. Потом посылаем в Word символ перехода строки, он честно переходит и тем самым сам создает нам второй пункт списка, куда и вставляем нужную строку. Ну и так далее, нужное количество раз. Последние две строки нужны, если список заготовлен в отдельном файле - после их выполнения список оказывается в буфере обмена. Здесь выгода в том, что можем иметь заготовки списков разных стилей и по ходу дела выбирать, какой список создать. Затем открываем документ, где должен быть список, выделяем с помощью Range нужный кусок, копируем из буфера обмена через   
WordDocument1.Range(a,b).Paste. Чтобы не испортить файл с заготовкой, можем сразу после открытия пересохранить его под другим именем, а можем просто выйти из него без сохранения изменений   
var vsave: OleVariant;  
...  
vsave:=wdDoNotSaveChanges;  
WordDocument1.Close(vsave);   
Константа сохранения изменений может принимать значения

|  |  |
| --- | --- |
| **Символьное обозначение** | **Шестнадцатеричное** |
| wdSaveChanges | $FFFFFFFF |
| wdDoNotSaveChanges | $00000000 |
| wdPromptToSaveChanges | $FFFFFFFE |

Первое значение сохраняет изменения, второе дает возможность выйти без сохранения изменений. Последняя константа вызывает при выходе стандартный диалог сохранения изменений. Можем сделать и несколько по-другому. Хотя мы не можем создать новый элемент списка, но текст в уже существующем изменить можно:   
var i,j: Integer;  
...  
i:=1;  
j:=1;  
WordDocument1.Lists.Item(i).ListParagraphs.Item(j).Range.Text:='Item 1';  
Так что можно с помощью переходов строки создать нужное количество элементов, а затем их заполнить:   
WordDocument1.Lists.Item(i).Range.Select;  
WordApplication1.Selection.Collapse(vcol);  
WordApplication1.Selection.InsertAfter(#13);  
j:=1;  
WordDocument1.Lists.Item(i).ListParagraphs.Item(j).Range.Text:='Item 1';  
j:=2;  
WordDocument1.Lists.Item(i).ListParagraphs.Item(j).Range.Text:='Item 2';  
Это было в предположении, что у нас один элемент списка в заготовке уже есть. Ну вот, в общем-то, и все про текст, списки и картинки  
***Статистика документов***  
В данном небольшом материале рассматривается вопрос подсчета статистики файлов \*.doc и \*.rtf. Такой вопрос у меня возник, когда пришлось сделать небольшую базу данных по учету документов, куда надо было заносить и статистику документа - число знаков, слов и т.п. Открывать каждый раз Word, считать статистику и забивать ее в форму ввода было лень, так что пришла в голову мысль это дело автоматизировать. Информации по данному вопросу найти так и не удалось, так что основным источником знаний служили заголовочный файл **Word2000.pas** и справка по **Visual Basic for Applications**. Ну и, конечно, множество разных экспериментов.   
Сразу оговорюсь, что я не профессиональный программист, так что в тонкости интерфейсов вникать не будем - сам в них не особо разбираюсь. Потому, не мудрствуя лукаво, просто поместим на форме компоненты **WordApplication** и **WordDocument** с палитры **Servers**. Для работы используются свойства и методы этих компонентов.   
Встроенная статистика Word подсчитывает статистику обычного текста, обычных и концевых сносок. Для подсчета статистики используется метод компонента **WordDocument ComputeStatistic()**. Он имеет один параметр, характеризующий, что именно считать, представляющий из себя шестнадцатеричную константу. Константы описаны в заголовочном файле **Word2000.pas**, он лежит обычно в **/Delphi/Ocx/Servers**.

|  |  |  |
| --- | --- | --- |
| **Шестнадцатеричная** | **Символьное обозначение** | **Смысл** |
| $00000000 | wdStatisticWords | Количество слов |
| $00000001 | wdStatisticLines | Количество строк |
| $00000002 | wdStatisticPages | Количество страниц |
| $00000003 | wdStatisticCharacters | Знаки без пробелов |
| $00000004 | wdStatisticParagraphs | Количество разделов |
| $00000005 | wdStatisticCharactersWithSpaces | Знаки с пробелами |

Это было основное, что надо знать. Ну а теперь по порядку.   
Поместив на форму упомянутые компоненты, видим, что свойств и методов у них совсем мало. В первую очередь следует определиться с методом ConnectKind компонента WordApplication. Оно может принимать различные значения, но мы оставим присваемое по умолчанию значение ckRunningOrNew. Это означает, что соединение происходит с уже работающим сервером, при его отсутствии запускается новый. Как правило, это вполне устраивает.   
Первым делом откроем документ. Предварительно надо объявить переменную FileName, она будет типа OleVariant, которой присвоим строку с именем файла.   
WordApplication1.Connect;  
WordApplication1.Documents.Open(FileName,   
EmptyParam,EmptyParam,EmptyParam,   
EmptyParam,EmptyParam,EmptyParam,   
EmptyParam,EmptyParam,EmptyParam,   
EmptyParam,EmptyParam); WordDocument1.ConnectTo(WordApplication1.ActiveDocument);  
Обратите внимание на количество параметров-"пустышек". Их число больше того, которое обычно приводится в книжках. Ну, в моих, во всяком случае. Объясняется это тем, что "книжные" функции предназначены для MS Word 97, а такая запись для работы с Word 2000 и Word XP.   
***"Plain Text"***  
Объявив нужное количество переменных типа LongInt (в очень большом файле или при суммировании по нескольким документам в принципе может оказаться больше знаков, чем пределы обычного целого типа), можем уже и приступать к подсчету. Например, посчитаем число слов, знаков с пробелами и без пробелов обычного текста, а также количество страниц в документе. Результаты сохраним соответственно в "длинных" переменных WCount, SCount, CCount, и PCount.   
WCount:=WordDocument1.ComputeStatistics($00000000);  
CCount:=WordDocument1.ComputeStatistics($00000003);  
SCount:=WordDocument1.ComputeStatistics($00000005);  
PCount:=WordDocument1.ComputeStatistics($00000002);  
Открыв нужный документ в Word'е и вызвав диалог подсчета статистики, нетрудно увидеть, что значения переменных равны параметрам вордовской статистики со сброшенным флажком "Учитывать все сноски".   
***Сноски***  
Сноски в документах могут быть обычные и концевые. То есть если первые располагаются внизу данной страницы, то концевые - строго в конце документа. Кроме того, они могут отличаться и нумерацией - автоматической или заданной пользователем. Начнем с обычных сносок как с самого простого. В терминологии объектной модели Word - Footnotes. Сначала надо вычислить количество самих сносок:   
ifcount:=WordDocument1.DefaultInterface.Footnotes.Count;   
Подсчет статистики текста в сноске производится так:   
FWCount:=WordDocument1.DefaultInterface.Footnotes.Item(ifoot).Range.ComputeStatistics($00000000);   
Здесь ifoot - целое число, "нумерующее" сноску. Для того, чтобы учесть сами номера сносок, сделаем так:   
FWCount:=FWCount+WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Reference.ComputeStatistics($00000000);   
Это мы посчитали для примера количество слов в сноске с номером ifoot и ее метке - при пользовательской нумерации в качестве "номера" может быть целое предложение. Далее начинаем перебирать их одну за другой. При этом следует учесть, что кроме статистики сносок необходимо получить и статистику их "номеров". То есть:   
for ifoot:=1 to ifcount do begin  
   FWCount:=FWCount+WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Range.ComputeStatistics($00000000);   
   FCCount:=FCCount+WordDocument1.DefaultInterface.Footnotes. Item(ifoot).Range.ComputeStatistics($00000003);   
   FSCount:=FSCount+WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Range.ComputeStatistics($00000005);   
   FCCount:=FCCount+WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Reference.ComputeStatistics($00000003);   
   FSCount:=FSCount+WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Reference.ComputeStatistics($00000005)+1;   
   if WordDocument1.DefaultInterface. Footnotes.Item(ifoot).Reference.Text<>IntToStr(ifoot)   
   then begin   
    FWCount:=FWCount+  
  WordDocument1.DefaultInterface.Footnotes.Item(ifoot). Reference.ComputeStatistics($00000000);   
   end;  
end;   
Прибавление единицы появляется оттого, что сумма статистики сносок и номеров не совпадает с тем, что выдает встроенная статистика Word. Между номером сноски и текстом сноски Word ставит пробел, который почему-то не учитывается. Условный оператор определяет, как пронумерована данная сноска - по умолчанию или нет. В последнем случае следует проверить количество слов в обозначении сноски. Такая схема дает результат, совпадающий со показаниями встроенной статистики. Кроме того, цикл у нас идет от 1 - так начинается нумерация сносок в MS Word, да и практически всех остальных объектов тоже.  
Теперь перейдем к концевым сноскам. Теоретически все то же самое, только вместо слова "Footnotes" пишем "Endnotes". И тут наталкиваемся на сюрприз - почему-то оно считает неточно. Я в данном случае поступил так: сохраняю документ под другим именем, переконвертирую концевые сноски в обычные и далее все, как сказано выше. Сохранение документа:   
WordDocument1.SaveAs(FileName, FileFormat),   
где в скобках стоят два параметра типа OleVariant - имя файла и шестнадцатеричная константа, задающая формат файла. Некоторые константы:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Шестнадцатеричная** | **Символьное обозначение** | **Смысл** | | $00000000 | wdFormatDocument | Документ Word | | $00000004 | wdFormatDOSText | Простой текст | | $00000006 | wdFormatRTF | Файл RTF | |

Полный список констант формата можно найти все в том же файле Word2000.pas. И еще один интересный момент - если просто поставить в скобки обе константы, работать не будет. Следует предварительно объявить две переменных, присвоить им соответствующие значения и только потом сохранять.   
Ну, а теперь, собственно, можем вернуться к сноскам. Конвертирование концевых сносок в обычные происходит так:   
WordDocument1.DefaultInterface.Endnotes.Convert;  
Теперь мы имеем документ, в котором содержатся только обычные сноски. С ними никаких проблем не возникает, пример, как с ними работать, см. выше. Если интересует статистика отдельно разных типов сносок, считаем предварительно статистику обычных сносок, сохраняем ее в "буферных" переменных и считаем еще раз после конвертирования. Разница даст статистику концевых сносок по отдельности. Сложив статистику сносок и простого текста, получаем статистику документа с учетом сносок так, как ее дает сам Word.   
***Дополнительно...***  
Тут по традиции несколько покритикуем Microsoft. Как оказалось, Word показывает не все, что содержится в документе. Не принимаются в расчет колонтитулы. А ведь в них может содержаться изрядный кусок текста, особенно в справках, бланках и т.п. Оказывается, Word их на самом деле считает, но нам не показывает. Вот и посмотрим, как же его можно заставить это сделать.   
Колонтитулы в документе тесно связаны с несколько загадочной штукой под названием "разделы" - Sections. Каждый раздел может иметь верхние и нижние колонтитулы. Потому первым делом определяем количество абзацев.   
isectct:=WordDocument1.DefaultInterface.Sections.Count;   
Здесь у нас целые переменные **isectct**, **icofct**, **icohct** обозначают соответственно количество разделов как таковых, количество нижних и верхних колонтитулов данного раздела. Переменная isec служит "номером" раздела, переменные **icof**, **icoh** "нумеруют" соответственно нижние и верхние колонтитулы в пределах данного раздела. Количество колонтитулов в разделе определяем так:   
icofct:=WordDocument1.DefaultInterface.Sections.Item(isec).Footers.Count;  
icohct:=WordDocument1.DefaultInterface.Sections.Item(isec).Headers.Count;   
Теперь уже можем "достать" текст из колонтитула:   
CBWCount:=  
WordDocument1.DefaultInterface.Sections.Item(isec). Footers.Item(icof).Range.ComputeStatistics($00000000);   
В данном случае мы для примера посчитали число слов, содержащихся в нижнем колонтитуле под номером **icof**, принадлежащем разделу под номером **isec**. Теперь можем написать "двойной" цикл для подсчета статистики верхних и нижних колонтитулов. Полностью это будет выглядеть так:   
isectct:=WordDocument1.DefaultInterface.Sections.Count;  
for isec:=1 to isectct do begin  
  icofct:=WordDocument1.DefaultInterface.Sections.Item(isec).Footers.Count;  
  icohct:=WordDocument1.DefaultInterface.Sections.Item(isec).Headers.Count;  
  for icof:=1 to icofct do begin  
    CBWCount:=CBWCount+  
      WordDocument1.DefaultInterface.Sections.Item(isec). Footers.Item(icof).Range.ComputeStatistics($00000000);  
    CBCCount:=CBCCount+  
      WordDocument1.DefaultInterface.Sections.Item(isec).Footers.Item(icof). Range.ComputeStatistics($00000003);  
    CBSCount:=CBSCount+  
      WordDocument1.DefaultInterface.Sections.Item(isec).Footers.Item(icof). Range.ComputeStatistics($00000005);  
  end;  
  for icoh:=1 to icohct do begin  
    CHWCount:=CHWCount+  
WordDocument1.DefaultInterface.Sections.Item(isec).Headers.Item(icoh). Range.ComputeStatistics($00000000);  
    CHCCount:=CHCCount+  
      WordDocument1.DefaultInterface.Sections.Item(isec).Headers.Item(icoh). Range.ComputeStatistics($00000003);  
    CHSCount:=CHSCount+  
      WordDocument1.DefaultInterface.Sections.Item(isec).Headers.Item(icoh). Range.ComputeStatistics($00000005);  
  end;  
end;   
В переменных, к которым на каждом шаге добавляются результаты статистики, после перебора всех разделов накопится суммарная статистика слов, знаков с пробелами и знаков без пробелов во всех колонтитулах.   
Часто использующиеся для рисования схемок текстовые вставки с панели рисования также представляют интерес. Сам Word формально считает их "картинками", не имеющими никакой статистики - видимо, по географическому расположению в панели инструментов. В объектной модели - Shapes. Вот тут начинается самое интересное. Во-первых, все, что находится на панели рисования, является Shapes. То есть в принципе для Word'а все едино, текстовая вставка, объект WordArt или геометрическая фигура. Вместе с тем смотрится довольно нелогично, что этот самый Shape можно переконвертировать, на выбор, во Frame или InlineShape. Они уже обладают статистикой, так что, казалось бы, все в порядке. Но коварству Microsoft, кажется, вообще нет предела. Во-первых с удивлением обнаруживаем, что Shapes нумеруются индексом типа OleVariant. Что с ним дальше делать, неясно. Если просто присваивать индексу целое число, при конвертации каждого второго Shape во Frame получаем ошибку. А если обработать исключение, то будем таки иметь статистику половины вставок. Видимо, есть какие-то тонкости с четными и нечетными индексами. Во-вторых, InlineShape штука и вовсе загадочная. Никаких ошибок при конвертации не возникало, но и количество InlineShapes неизменно оказывалось нулевым. Подсчитать статистику вставок удалось только сохранив файл как RTF и расковыряв его код, но это стоит описать отдельно. Приводился же последний абзац в надежде, что кто-то с таким сталкивался и нашел способ работы с Shapes "встроенными" способами.   
Ну, вот практически и все. Суммируя все, что мы получили, имеем статистику документа даже точнее встроенной. Еще пара замечаний. Перед подсчетом Word стоит "спрятать", чтобы он не маячил на экране:   
WordApplication1.Visible:=False;   
При подсчете статистики, особенно если в документе содержится что-то кроме простого текста, считается, что в файл внесены изменения. Потому напоследок сохраняем и закрываем документ:   
WordDocument1.Save;  
WordDocument1.Close;   
Ну и, конечно, делаем серверу **Word Disconnect**, когда он станет нам уже не нужен.   
А теперь предупреждение тем, кто заинтересовался данным вопросом и хочет поэкспериментировать сам. Офис слишком тесно связан с Windows, потому на сбои в его работе система реагирует крайне остро. При отладке программ подсчета статистики у меня после ошибки часто появлялся "голубой экран". То же самое происходило, если после разбора RTF - файла в поисках Shapes и превращения их в обычный текст в Word загружался неправильно "собранный" файл. Так что очень рекомендуется предварительно сохранить важные данные или поставить систему понадежнее. У меня стоит WindowsXP, который к таким сбоям оказался нечуствителен. Кроме того, пока приведенное здесь не было отлажено, после ошибок частенько летел сам офис. Так что имейте под рукой дистрибутив для запуска диагностики офиса и устранения повреждений.   
***Работаем с таблицами***  
 Каждый, наверное, хоть раз сталкивался с необходимостью выдачи отчета. В Delphi имеются для этого специальные компоненты, но они налагают на нас достаточно строгие ограничения на форму представления данных. Одним из выходов может служить использование программы MS Word. Здесь не будем подробно обсуждать простейший вопрос, как открыть документ и добавить в него нужную строку текста, это есть практически в каждом учебнике по Delphi, приведем только самые необходимые сведения. А из литературы на эту тему особенно рекомендуется найти книжку А.Я. Архангельского "Язык SQL в Delphi 5". Но что может придать отчету такую читабельность, как представление результатов в систематизированном табличном виде? В данной статье и обсуждается вопрос программной работы с таблицами документа Word.   
Тут могут быть два пути. Первый - если мы знаем заранее структуру данных отчета, можем приготовить шаблон, куда в ячейки таблицы затем просто занесем нужные данные. И второй - создаем отчет с нуля, рисуем в документе таблицу, заполняем ее. При этом мы можем программно добавить или удалить строки и столбцы, объединить или разбить ячейки - почти все, что мы делаем в самом Word'e.   
Все, что понадобится - компоненты WordApplication и WordDocument с палитры Servers   
Теперь все по порядку - открываем файл и приступаем. Предварительно объявляем переменную FileName, типа OleVariant, которой присваиваем строку с именем файла.

WordApplication1.Connect;

WordApplication1.Documents.Open(FileName,

               EmptyParam,EmptyParam,EmptyParam,

               EmptyParam,EmptyParam,EmptyParam,

              EmptyParam,EmptyParam,EmptyParam,

               EmptyParam,EmptyParam);

WordDocument1.ConnectTo(WordApplication1.ActiveDocument);

Обратите внимание на количество параметров - "пустышек". Их число не совпадает с тем, что обычно приводится в книжках. Объясняется это тем, что "книжная" функция предназначена для MS Word 97, а такая запись для Word 2000 и Word XP.   
Создание нового документа выглядит проще:

WordApplication1.Connect;

WordApplication1.Documents.Add(EmptyParam, EmptyParam, EmptyParam, EmptyParam);

WordDocument1.ConnectTo(WordApplication1.ActiveDocument);  
Здесь также ставим на пару "пустышек" больше - по тем же самым причинам. Кроме того, полезно будет сразу же отключить проверку орфографии, чтобы Word не тратил время понапрасну:

WordApplication red; FONT-FAMILY: Arial">1.Options red;   
FONT-FAMILY: Arial">.CheckSpellingAsYouType red; FONT-FAMILY: Arial">:=False red;   
FONT-FAMILY: Arial">;

WordApplication1.Options.CheckGrammarAsYouType:=False;  
По окончании работы нам надо сохранить или распечатать наш отчет:

WordDocument red; FONT-FAMILY: Arial">1.PrintOut red; FONT-FAMILY: Arial">;

WordDocument1.SaveAs(FileName);  
где переменная в скобках типа OleVariant, ей присваиваем строку с именем файла.  
***Объект Range***  
В документе нас пока больше всего интересует объект Range, который нам понадобится при создании таблицы. Он представляет из себя кусок текста, который может включать в себя как весь текст документа, так и любую его часть. То есть:

var range1, range2, range3, a, b : OleVariant;

...

range1:=WordDocument1.Range;

a:=5;

b:=15;

range2:=WordDocument1.Range(a,b);

range3:=WordDocument1.Range(a);

Первый наш объект включает в себя весь текст документа, у второго мы ограничили пределы 5-м и 15-м символами, третий представляет из себя весь последующий текст документа, начиная с 5-го символа.   
Объект имеет несколько полезных методов, например, с его помощью можем добавить текст в документ:

red; FONT-FAMILY: Arial">range2.InsertAfter('MS Word');

Это мы вставили текст после выделенного Range. Точно также можем вставить текст и перед ним, для этого служит метод InsertBefore(). Текст, заключенный в объекте Range, можем получить так:

red; FONT-FAMILY: Arial">WordDocument1.Range(a,b).Text;

Кроме того, с помощью Range можем изменить шрифт в пределах объекта.   
Пример:

red; FONT-FAMILY: Arial">a:=5;

red; FONT-FAMILY: Arial">b:=15;

red; FONT-FAMILY: Arial">WordDocument1.Range(a,b).Font.Bold:=1;

Если хотим отменить выделение жирным шрифтом, присваиваем 0. Аналогично можно сделать шрифт курсивом, подчеркнутым - наберите WordDocument1.Range.Font., и среда сама подскажет, какие могут быть варианты. Методы Select, Cut, Copy и Paste работают как в обычном тексте. С помощью Paste можем на место выбранного Range вставить не только строки, но и рисунок, находящийся в буфере обмена.   
**Т а б л и ц ы**  
***Работа со столбцами, строками и ячейками***  
Таблицы в документе Word образуют коллекцию Tables. Их количество можем узнать так:

red; FONT-FAMILY: Arial">tcount:=WordDocument1.Tables.Count;

red; FONT-FAMILY: Arial">

к отдельной таблице обращаемся по ее номеру:

i:=1;

WordDocument1.Tables.Item(i) ...,

red; FONT-FAMILY: Arial">

где i - целое число. В данном случае мы обращаемся к первой таблице, а вообще i может принимать значения от 1 до WordDocument1.Tables.Count. Если нам необходимо создать таблицу самим, следует поступить так:

WordDocument1.Tables.Add(WordDocument1.Range, i, j, EmptyParam, EmptyParam);

red; FONT-FAMILY: Arial">

Эта таблица - единственное, что будет в документе, так как она заменяет собой указанный в числе параметров объект Range. В данном случае получаем таблицу на i строк и j столбцов. Если уже еcть какой-то текст, который надо сохранить, совершенно аналогичным образом можем указать пределы объекта Range:

a:=5;

b:=15;

WordDocument1.Tables.Add(WordDocument1.Range(a,b), i, j, EmptyParam, EmptyParam);  
Переменные a и b должны быть объявлены как OleVariant.   
Ну вот, теперь у нас есть таблица. Неважно, содержалась она уже в документе или мы создали ее сами. Посмотрим, что же мы с ней можем сделать. Число столбцов и строк узнаем так:

i:=1;

k:=WordDocument1.Tables.Item(i).Columns.Count;

j:=WordDocument1.Tables.Item(i).Rows.Count;  
Здесь мы опять обратились к первой таблице, но можем работать с любой - надо только правильно указать ее номер. Теперь изменим ширину столбцов или высоту строк:

WordDocument1.Tables.Item(i).Columns.Width:=90;

WordDocument1.Tables.Item(i).Rows.Height:=45;  
Аналогично можем задавать размеры отдельных строк и столбцов:

WordDocument1.Tables.Item(i).Columns.Item(j).Width:=90;

WordDocument1.Tables.Item(i).Rows.Item(j).Height:=45;  
Здесь j - опять таки целое число, начинается от 1. Можем обратится к отдельной ячейке, прочитать или изменить содержащийся в ней текст:

WordDocument1.Tables.Item(i).Cell(j,k).Range.Text;

red; FONT-FAMILY: Arial">

Здесь j и k целые переменные, изменяются от 1 до числа строк или столбцов соответственно. Присвоив данной величине строковое выражение, увидим, что оно появилось в ячейке (j,k). Несколько непривычно, но в таблицах Word на первом месте стоит именно номер строки. Можем также задать программно отступы от края ячеек, как для всей таблицы сразу, так и для отдельной ячейки:

WordDocument1.Tables.Item(i).TopPadding:=10;

WordDocument1.Tables.Item(i).BottomPadding:=10;

WordDocument1.Tables.Item(i).RightPadding:=10;

WordDocument1.Tables.Item(i).LeftPadding:=10;  
В данном случае мы задали одинаковые отступы для всей таблицы, но аналогичные четыре свойства есть и у отдельной ячейки. Выделить нужную ячейку, столбец или строку можем следующим образом:

WordDocument1.Tables.Item(i).Cell(j,k).Select;

WordDocument1.Tables.Item(i).Columns.Item(j).Select;

WordDocument1.Tables.Item(i).Rows.Item(j).Select;  
Кроме того, можем подогнать размеры ячеек по содержимому. Для этого вызываем метод AutoFit:

WordDocument1.Tables.Item(i).Columns.AutoFit;

red; FONT-FAMILY: Arial">

Добавить строку или столбец также не представляет сложностей:

WordDocument1.Tables.Item(i).Columns.Add(EmptyParam);

WordDocument1.Tables.Item(i).Rows.Add(EmptyParam);

Мы добавили строку внизу и столбец справа. Теперь вставим столбец в определенном месте таблицы:

var i, j: Integer;

      varcol: OleVariant;

...

j:=2;

varcol:=WordDocument1.Tables.Item(i).Columns.Item(j);

WordDocument1.Tables.Item(i).Columns.Add(varcol);  
Совершенно аналогично поступаем и со строками. Вообще в скобках указаны строка или столбец, перед которыми происходит вставка. Однако явно указать в скобках почему-то нельзя, надо через переменную.   
Теперь про объединение ячеек. Довольно просто:

WordDocument1.Tables.Item(i).Cell(j,k).Merge(WordDocument1.Tables.Item(i).Cell(j,k+1));

Мы объединили две соседние по горизонтали ячейки (j,k) и (j,k+1). При этом получается, что большая ячейка как бы имеет два "адреса". Аналогично надо действовать и при объединении по вертикали. Все точно так же, но с нумерацией ячеек после объединения двух соседних по вертикали начинается путаница и при попытке заполнить таблицу возникают ошибки. Теперь разобьем ячейки.

varrow:=1;

varcol:=2;

WordDocument1.Tables.Item(i).Cell(j,k).Split(varrow, varcol);

Здесь мы разбили ячейку (j,k) на две по горизонтали. Переменные varcol и varrow типа OleVariant, это количество столбцов и строк, на которые разбивается данная ячейка. Здесь снова с нумерацией начинается чехарда, так что этот вопрос разбиения и объединения ячеек представляет скорее чисто теоретический интерес. В таких случаях лучше заранее приготовить шаблоны.   
Теперь для примера удалим из таблицы второй столбец или третью строку:

WordDocument1.Tables.Item(i).Columns.Item(2).Delete;

WordDocument1.Tables.Item(i).Rows.Item(3).Delete;

***red; FONT-FAMILY: Arial">Внешний вид таблицы****red; FONT-FAMILY: Arial">*

Простейшая таблица, конечно, смотрится не очень. Теперь посмотрим, как мы можем ее приукрасить. При желании все сделать посимпатичнее можем использовать текстуру. Выглядеть это будет так:

WordDocument1.Tables.Item(i).Cell(j,k).Shading.Texture:=wdTexture20Percent;

Совершенно аналогично можем сделать текстуру в целом столбце или строке:

WordDocument1.Tables.Item(i).Columns.Item(j).Shading.Texture:=wdTexture20Percent;

WordDocument1.Tables.Item(i).Rows.Item(j).Shading.Texture:=wdTexture20Percent;

Текстура задается шестнадцатеричной константой, список констант можно найти заголовочном файле Word2000.pas. Можно их использовать как в шестнадцатеричном, так и в символьном виде. Чтобы не загромождать материал, значения констант будут выноситься в "Приложение" в конце статьи. Сразу оговорюсь, что заливка будет черно-белая или в шкале серого. Заливку определенным цветом пока так и не удалось обнаружить. Самая первая константа означает отсутствие заливки. Ее можно использовать, чтобы отменить текстуру.   
Чтобы выделить что-нибудь важное, можем изменить шрифт текста в определенной ячейке. Для этого воспользуемся свойствами объекта Selection:

WordDocument1.Tables.Item(i).Cell(1,2).Select;

WordApplication1.Selection.Font.Color:=clRed;

WordApplication red; FONT-FAMILY: Arial">1.Selection red;   
FONT-FAMILY: Arial">.Font red;   
FONT-FAMILY: Arial">.Italic red; FONT-FAMILY: Arial">:=1;

WordApplication1.Selection.Font.Size:=16;  
В данном примере мы сделали цвет текста в ячейке (1,2) красным, выделили его курсивом и изменили размер на 16. Кроме того, можем сделать шрифт подчеркнутым, перечеркнутым и т.п.   
Еще один способ изменить внешний вид таблицы - использовать стилевые шаблоны Word'a. У таблицы имеется метод AutoFormat, который меняет внешний вид таблицы в соответствии с некими предопределенными стилями. В заголовочном файле он описан следующим образом:

procedure AutoFormat(var Format: OleVariant; var ApplyBorders: OleVariant;

var ApplyShading: OleVariant; var ApplyFont: OleVariant; var ApplyColor: OleVariant;

var ApplyHeadingRows: OleVariant; var ApplyLastRow: OleVariant; var ApplyFirstColumn:

OleVariant; var ApplyLastColumn: OleVariant; var AutoFit: OleVariant);

Первый параметр представляет из себя собственно константу, задающую стиль, а остальные показывают, будут ли требования нового стиля применяться конкретно к границам, тени, шрифту, цвету, первой строке, последней строке, первому столбцу и последнему столбцу. Последний параметр в списке указывает, надо ли подгонять размер ячеек по их содержимому - лучше самому потом вызвать AutoFit.   
Как показала практика, работают только два первых параметра. Все остальные заменяем "пустышками". То есть это скорее будет просто способ изменения стиля границ, но и на том спасибо. Некоторые стили таблиц даны в приложении, полный же список ищите в заголовочном файле. Для примера применим к нашей таблице стиль "Вэб3". Вместо второго параметра ставим вариантную переменную, которой присваиваем wdTableFormatApplyBorders. То есть на практике это выглядит так:

var tformat, tappbrd: OleVariant;

...

tformat:=wdTableFormatWeb3;

tappbrd:=wdTableFormatApplyBorders;

i:=1;

WordDocument1.Tables.Item(i).AutoFormat(tformat, tappbrd,   
EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam,

EmptyParam, EmptyParam, EmptyParam);

***И еще об одном способе создания таблиц***  
Людям, интересующимся работой с MS Word, возможно, тоже попадались в интернете компоненты, превращающие в таблицу соответствующим образом отформатированный текст. Вот мы как раз и разберемся, как же они устроены. Здесь нам опять надо вспомнить про объект Range, а именно про имеющийся у него метод ConvertToTable. В заголовочном файле это выглядит так:

function ConvertToTable(var Separator: OleVariant; var NumRows: OleVariant;

var NumColumns: OleVariant; var InitialColumnWidth: OleVariant;

var Format: OleVariant; var ApplyBorders: OleVariant;   
var ApplyShading: OleVariant; var ApplyFont: OleVariant;

var ApplyColor: OleVariant; var ApplyHeadingRows: OleVariant; var ApplyLastRow:

OleVariant; var ApplyFirstColumn: OleVariant; var ApplyLastColumn: OleVariant;

var AutoFitBehavior: OleVariant; var DefaultTableBehavior: OleVariant)

Здесь опять видим все те же константы применения стиля, что и в автоформате. Только в отличие от него тут они ошибок не вызывают. Правда, есть у меня жуткое подозрение, что они все равно не работают и можем со спокойной совестью поставить вместо них "пустышки" - всех, кроме опять-таки, первых двух параметров "применения изменений". Теперь по порядку. Первый параметр у нас задает символ, которым будут отделяться ячейки одной строки нашей будущей таблицы, второй - число строк таблицы, третий - число столбцов, затем идет ширина столбцов. Следующая группа параметров задает стиль таблицы и особенности его применения, необходимые значения констант есть в таблице "Приложения". Последние три параметра задают подгонку размера ячеек по содержимому, но на самом деле не работают. Так что ставим вместо них EmptyParam. И, наконец, практический пример. Предположим, мы открыли новый документ и занесли в него нужный текст:

WordDocument1.Range.InsertAfter('column1');

WordDocument1.Range.InsertAfter(#9);

WordDocument1.Range.InsertAfter('column2');

WordDocument1.Range.InsertAfter(#9);

WordDocument1.Range.InsertAfter('column3');

WordDocument1.Range.InsertAfter(#9);

WordDocument1.Range.InsertAfter('column4');

WordDocument1.Range.InsertAfter(#13);  
Такую операцию повторим трижды, и у нас будет заготовка для таблицы на 4 столбца и 3 строки. Будущие столбцы отделяются символами табуляции, а строки - переходами на новую строку. Теперь выделяем объект Range - в данном случае это весь текст документа, и превращаем его в таблицу:

var tsepar, tnumrows, tnumcols, tincolw, tformat,

      tappbrd, tappshd, tappfnt, tappclr, tapphr, tapplr,

      tappfc, tapplc: OleVariant;

...

tsepar:=wdSeparateByTabs;

tnumrows:=3;

tnumcols:=4;

tincolw:=150;

tformat:=wdTableFormatNone;

tappbrd:=wdTableFormatApplyBorders;

tappshd:=wdTableFormatApplyShading;

tappfnt:=wdTableFormatApplyFont;

tappclr:=wdTableFormatApplyColor;

tapphr:=wdTableFormatApplyHeadingRows;

tapplr:=wdTableFormatApplyLastRow;

tappfc:=wdTableFormatApplyFirstColumn;

tapplc:=wdTableFormatApplyLastColumn;

WordDocument1.Range.ConvertToTable(tsepar, tnumrows, tnumcols, tincolw, tformat,

      tappbrd, tappshd, tappfnt, tappclr, tapphr, tapplr,

      tappfc, tapplc, EmptyParam, EmptyParam, EmptyParam);

В общем-то, мы можем задавать только первые шесть параметров, а все остальные заменить на "пустышки". Единственное, что здесь новенькое, это параметр "разделителя". В данном случае в качестве разделителя будет использоваться символ табуляции. Другие варианты смотрите в последней таблице "Приложения", но мне кажется, особой практической пользы от них нет.   
Проделав все это, обнаруживаем, что наш текст превратился в таблицу. Все это несколько напоминает создание таблиц в HTML.

**Приложения**

Список некоторых констант. Смотрите файл Word2000.pas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Описание | Символьное обозначение | Шестнадцатеричное | | Текстуры | | | | Некоторые константы текстур - откройте Word2000.pas в "блокноте", введите в диалог поиска "texture", и получите полный список. |  |  | | wdTextureNone | $00000000 | | wdTexture2Pt5Percent | $00000019 | | wdTexture7Pt5Percent | $0000004B | | wdTexture35Percent | $0000015E | | wdTexture62Pt5Percent | $00000271 | | wdTextureSolid | $000003E8 | | wdTextureDarkHorizontal | $FFFFFFFF | | wdTextureCross | $FFFFFFF5 | | Формат таблицы | | | | Константы формата таблицы. Необходимы при использовании автоформата и преобразовании текста в таблицу. |  |  | | wdTableFormatNone | $00000000 | | wdTableFormatSimple1 | $00000001 | | wdTableFormatSimple3 | $00000003 | | wdTableFormatClassic1 | $00000004 | | wdTableFormatClassic3 | $00000006 | | wdTableFormatColorful1 | $00000008 | | wdTableFormatColumns1 | $0000000B | | wdTableFormatGrid1 | $00000010 | | wdTableFormatGrid3 | $00000012 | | wdTableFormatList1 | $00000018 | | wdTableFormat3DEffects1 | $00000020 | | wdTableFormat3DEffects2 | $00000021 | | wdTableFormat3DEffects3 | $00000022 | | wdTableFormatContemporary | $00000023 | | wdTableFormatElegant | $00000024 | | wdTableFormatProfessional | $00000025 | | wdTableFormatSubtle1 | $00000026 | | wdTableFormatWeb3 | $0000002A | | Стили таблицы | | | | Константы "применения изменений" стиля таблицы. В идеале предполагается, что можно задавать отдельно, изменять ли стиль границ, текста, цвета и т.д. В автоформате работают, мягко выражаясь, невразумительно, все время возникают какие-то непонятные ошибки. |  |  | | wdTableFormatApplyBorders | $00000001 | | wdTableFormatApplyShading | $00000002 | | wdTableFormatApplyFont | $00000004 | | wdTableFormatApplyColor | $00000008 | | wdTableFormatApplyAutoFit | $00000010 | | wdTableFormatApplyHeadingRows | $00000020 | | wdTableFormatApplyLastRow | $00000040 | | wdTableFormatApplyFirstColumn | $00000080 | | wdTableFormatApplyLastColumn | $00000100 | | Разделители | | | | Константы разделителей для конвертации текста в таблицу. |  |  | | wdSeparateByParagraphs | $00000000 | | wdSeparateByTabs | $00000001 | | wdSeparateByCommas | $00000002 | | wdSeparateByDefaultListSeparator | $00000003  Copied from: http://articles.org.ru/docum/msword.php | |