**type TStringList;**

Описание

Элементы в строковом списке могут быть вставлены, перемещены и отсортированы.

Список может быть сформирован строка за строкой, или загружен из большой строки разделенной запятой или даже из текстового файла. TStringList осуществляет эти абстрактные методы (Clear, Delete и Insert). Мы рассмотрим основные свойства и методы TStringList, включая полученные из TStrings.

Свойство Count - Возвращает число строк в списке.

Свойство Capacity - Устанавливает или получает текущую вместимость строкового списка. Вы можете управлять этой вместимостью по необходимости.

Свойство Strings - Получает или корректирует строку по данному индексу в списке (первый элемент списка имеет индекс 0).

Обратите внимание, что свойство Strings является свойством, заданным по умолчанию. Это означает, что вы можете использовать его без его указания:

myName := names.Strings[4];

является эквивалентным:

myName := names[4];

Свойство Text - Устанавливает или получает список в виде большой строки. Эта строка будет содержать каждую строку заканчивающуюся комбинацией символов перевода каретки и перевода строки (CRLF). Полезно для загрузки из визуального объекта, который может содержать многочисленные строки текста.

Свойство CommaText - Получает или устанавливает список в виде большой строки. Эта строка будет иметь список строк разделенных запятыми. Это полезно для загрузки из экспорта текстовой электронной таблицы. Если при получении строка содержит вложенные пространства, то она будет заключена в двойные кавычки.

Cвойство DelimitedText - Получает или устанавливает список через большую строку. Эта строка содержит список строк разделенных значением Delimiter (по умолчанию - запятая). Строки, содержащие вложенные пробелы должны быть заключены в QuoteChar (по умолчанию - ").

Свойство QuoteChar - Используется для замыкания строк, которые имеют вложенные пробелы при использовании DelimitedText.

Свойство Delimiter - Используется для разделения строк при использовании DelimitedText

Свойство Names - Строки в строковом списке могут быть обработаны, как пары название/значение, как во втором примере кода. Каждая строка не должна иметь никаких внедренных пробелов, и содержать знак =. Это очень полезное понятие. См. свойства Value и ValueFromIndex, и метод IndexOfName.

Свойство Values - Возвращает значение для данного названия, когда используются строки пары название/значение (см. выше).

Свойство ValueFromIndex - Возвращает значение по индексу строки (начинается с 0), когда используются пары название/значение.

Свойство CaseSensitive - Когда true, Delphi обрабатывает строки чувствительно к регистру при выполнении некоторых операций, таких как Sort.

Свойство Duplicates - Это свойство может иметь одно из следующих перечислимых TDuplicates значений:

dupIgnore Игнорирует (отбрасывает) дубликаты

dupAccept Позволяют дубликаты

dupError Выбрасывает исключение, если имеются дубликаты

Свойство Sorted - Когда true, все строки будут добавляться в свою позицию отсортированной последовательности. Когда false, они будут добавляться в конец. См. также метод Sort.

Свойство Objects - Возвращает объект, связанный со строкой по данному индексу, если он существует.

Метод Add - Добавляет данную строку в список, возвращая ее позицию в списке (начинается с 0).

Метод Append - Так же как и Add, но без возвращения индексного значения.

Метод Insert - Вставляет строку в заданную индексом позицию. Позиция 0 вызовет вставку в начало.

Метод Delete - Удаляет строку по данному индексу.

Метод Clear - Удаляет все строки из списка.

Метод Move - Перемещает строку из одной позиции в другую, сдвигая другие строки соответственно.

Метод Exchange - Перестанавливает две строки в списке, идентифицированные по их индексным позициям.

Метод IndexOf - Получает индекс позиции строки в списке соответствующей данной строке. Если строка не найдена, то возвращается -1.

Метод IndexOfName - Получает индекс позиция первой пары название/значение строки, где название соответствует данной строке. Если не найдена - возвращается -1.

Метод Find - То же самое, что и IndexOf, но с использованием списков сортированных строк.

Метод Sort - Если Sorted является ложным, то это вызовет сортировку списка.

Метод AddStrings - Добавляет строки из другого списка.

Метод Assign - Заменяет текущий список содержанием другого списка.

Метод LoadFromFile - Очень полезный метод, загружает строковый список из текстового файла. Каждая текстовая строка (законченая CRLF - см. DelimitedText) становится строкой списка.

Метод SaveToFile - Сохраняет строковый список в текстовый файл.

Довольно часто компоненты содержат или владеют списками нескольких элементов — данных, записей, объектов и даже других компонентов. В некоторых случаях удобно инкапсулировать такой список в специальный объект и сделать этот объект свойством владельца компонента. Примером подобного подхода служит свойство Lines компонента TMemo. Данное свойство имеет тип TStrings, инкапсулирующий список строк. При этом объект TStrings отвечает за механизм работы с потоками, используемый для записи строк в файл формы при сохранении этой формы.

Как следует поступить, если требуется сохранить список элементов компонентного или объектного типов, которые еще не инкапсулированы существующим классом, таким как TStrings? В этом случае можно было бы создать класс, обеспечивающий взаимо действие элементов списка с потоками данных и сделать его свойством компонента владельца. В качестве альтернативного варианта можно переопределить стандартный механизм взаимодействия с потоками данных компонента владельца так, чтобы он знал, как работать с данным списком элементов. Но самое лучшее решение — воспользоваться преимуществами, предоставляемыми классами TCollection и TCollectionItem.

Класс TCollection — это объект, используемый для хранения списка объектов типа TCollectionItem. Сам класс TCollection — это не компонент, а просто потомок класса TPersistent. Обычно класс TCollection связан с существующим компонентом.

Чтобы воспользоваться возможностями класса TCollection для хранения списка элементов, необходимо создать экземпляр класса, производный от TCollection, назвав его, например, TNewCollection. Этот класс будет служить типом свойства компо нента. Затем следует создать новый класс из класса TCollectionItem, который можно назвать TNewCollectionItem. Класс TNewCollection будет содержать список объек тов типа TNewCollectionItem. Вся прелесть такого подхода состоит в том, что для то го, чтобы принадлежащие классу TNewCollectionItem данные были обработаны обычными средствами взаимодействия с потоками данных, достаточно их опубликовать (объявить в разделе published компонента TNewCollectionItem). Delphi знает, как записывать опубликованные свойства в поток и считывать их из него.

Примером использования класса TCollection может служить компонент TSta- tusBar, являющийся потомком класса TWinControl. Одно из его свойств, Panels, имеет тип TStatusPanels. Класс TStatusPanels является потомком класса TCol- lection и определяется следующим образом:

type

TStatusPanels = class(TCollection)

private

FStatusBar: TStatusBar;

function GetItem(Index: Integer): TStatusPanel;

procedure SetItem(Index: Integer; Value: TStatusPanel);

protected

procedure Update(Item: TCollectionItem); override;

public

constructor Create(StatusBar: TStatusBar);function Add: TStatusPanel;

property Items[Index: Integer]: TStatusPanel read GetItem

write SetItem; default;

end;

Класс TStatusPanels хранит список потомков класса TCollectionItem — объ

ектов класса TStatusPanel. Ниже приведен код определения этого класса.

type

TStatusPanel = class(TCollectionItem)

private

FText: string;

FWidth: Integer;

FAlignment: TAlignment;

FBevel: TStatusPanelBevel;

FStyle: TStatusPanelStyle;

procedure SetAlignment(Value: TAlignment);

procedure SetBevel(Value: TStatusPanelBevel);

procedure SetStyle(Value: TStatusPanelStyle);

procedure SetText(const Value: string);

procedure SetWidth(Value: Integer);

public

constructor Create(Collection: TCollection); override;

procedure Assign(Source: TPersistent); override;

published

property Alignment: TAlignment read FAlignment

write SetAlignment default taLeftJustify;

property Bevel: TStatusPanelBevel read FBevel

write SetBevel default pbLowered;

property Style: TStatusPanelStyle read FStyle

write SetStyle default psText;

property Text: string read FText write SetText;

property Width: Integer read FWidth write SetWidth;

end;

Свойства типа TStatusPanel, объявленные в разделе published класса, автомати чески получают возможность взаимодействовать с потоками в Delphi. В классе TSta- tusPanel предусмотрена передача параметра типа TCollection конструктору Cre- ate(), благодаря чему обеспечивается связь с классом TCollection. В данном случае конструктору класса TStatusPanel передается компонент TStatusBar, с которым объект TStatusPanel должен быть связан. Класс TCollection обладает механизмом взаимодействия с потоками данных, являющихся компонентами класса TCollec- tionItem, а также определяет некоторые методы и свойства для манипулирования элементами, поддерживаемыми в классе TCollection. Их описание можно найти в ин терактивной справке.

Для иллюстрации использования этих двух новых классов был создан компонент TddgLaunchPad, позволяющий пользователю хранить список компонентов TddgRunBut- ton (компонент TddgRunButton описан в главе 11, “Разработка компонентов VCL”).

Компонент TddgLaunchPad — потомок компонента TScrollBox. Одно из свойств но

вого компонента — RunButtons — является потомком класса TCollection и поддержива

ет список компонентов TRunBtnItem. Компонент TRunBtnItem — это потомок класса

TCollectionItem, свойства которого используются для создания компонента TddgRun-Button, помещенного в компонент TddgLaunchPad. В следующих разделах рассматри

Свойство Count. Количество элементов в коллекции. Аналог такого же свойства TList.

Свойство ItemClass. Дает фактический класс элементов коллекции. Этот класс задается при создании коллекции и в дальнейшем быть изменен не может. Все элементы коллекции имеют один и тот же класс (в этом смысле список на основе TList более гибок, так как не имеет подобного ограничения).

Свойство Items. Массив элементов коллекции. Аналог такого же свойства TList.

Методы Add, Clear и Insert. Аналоги соответствующих методов TList, но с одним важнейшим отличием. При добавлении (вставке) объекта в список TList или его удалении из списка сам объект не создается и, соответственно, не уничтожается. Те же операции с коллекцией приводят к автоматическому созданию и уничтожению экземпляра объекта. Конечно, это возможно именно потому, что в случае коллекции класс объекта известен заранее, а в случае TList объект может быть любым.

Метод Assign. Копирует элементы одной коллекции в другую. Конечно, если классы элементов этих коллекций не совпадают, возникнет ошибка. Кстати, такое копирование стало возможным как раз потому, что коллекции и их элементы построены на основе класса TPersistent, в то время как подобная операция со списками TList требует дополнительного кода.

Методы BeginUpdate и EndUpdate. Эти методы проще всего рассмотреть на примере перерисовки. Выше отмечалось, что коллекции предназначены, в основном, для создания «визуальных» списков. Если один из элементов коллекции обновляется, это приводит к его обновлению и на экране. Если же обновляются сразу несколько элементов, то нет смысла выполнять промежуточные перерисовки экрана, а надо выполнить только одну - после обновления всех элементов. Это и позволяют сделать два рассматриваемых метода. Важно знать, что перерисовка происходит только после того, как метод EndUpdate будет вызван ровно столько раз, сколько перед этим был вызван BeginUpdate. Чтобы гарантировать правильную работу, обычно эти вызовы используются совместно с блоком try…finally. Конечно, этот механизм может быть использован при любом обновлении элементов коллекции, а не только для их перерисовки.

Метод FindItemID. Дает элемент коллекции с заданным ID (либо Nil, если такового нет).

Метод GetNamePath. Используется для внутренних нужд IDE, как часть поддержки работы в design-time. Для прикладного программиста этот метод вряд ли представляет интерес.

Метод Changed. Этот protected-метод должен вызываться наследниками TCollection при изменении существенных свойств коллекции. Приводит к вызову метода Update, но не сразу, а после ее полного обновления (см. BeginUpdate и EndUpdate).

Метод Update. В классе TCollection этот protected-метод не делает ничего, но потомки могут заместить его для фактического обновления коллекции (например, для той же перерисовки).

В остальном класс TCollection - это, в общем-то, обычный объект (за исключением того, что имеет ряд дополнительных методов, обеспечивающих взаимодействие с IDE в design-time). Никакие события в этом классе не определены, но могут быть определены в его потомках.

Теперь нужно объявить переменную **класса TRegExpr.** После этого присваиваем свойству InputString текст, в котором будет осуществляться поиск по определенной маске. А саму маску задаем в свойстве Expression.

В классе TRegExpr есть некая функция Exec которая выполняет поиск по тексту.

Если найдены соответствия, то ф-ия выдает значение true.

Если соответствия отсутствуют, то выдает значение false.

Получить текст, который соответствует маске можно с помощью переменной Match[1], где [1] — ID.

Ну а теперь перейдем к практике. Мы будем парсить из некого текста <a href="http://domain.com">Site name</a> — саму ссылку [http://domain.com]; или название Site name.

RegExp: TRegExpr — объявляем переменную RegExp класса TRegExpr.

RegExp := TRegExpr.Create — активируем переменную RegExp.

RegExp.InputString :='<a href="http://domain.com">Site name</a>' — задаем значение свойству InputString.

RegExp.Expression := 'href="(.\*?)">(.\*?)<' — указываем маску в свойстве Expression.

if RegExp.Exec then — если ф-ия Exec находит совпадения по маске, то.

ShowMessage(RegExp.Match[1]) — показывает в сообщении первый результат.

ShowMessage(RegExp.Match[2]) — второй результат.