Creando colecciones

En este documento vamos a ver como crear una colección (TCollection). No vamos a ver todas las posibilidades, sólo aquellas que son más utilizadas o que yo considero son más interesantes.

Lo primero es qué entendemos por una colección. Según el DRAE (Diccionario de la Real Academia de la Lengua Española) una colección es un "*Conjunto de cosas, por lo común de una sola clase*". Ejemplos de colecciones pues muchos, desde libros, dvd's, hasta cromos. En cuanto a lo que nos afecta a nosotros serían datos con una estructura común sobre cualquier asunto : Campos de una tabla, constraints, lista de libros con información de los mismos, etc.

Nosotros para este documento utilizaremos una colección creada para la exportación de datos a HTML (en el componente HtmlExport) cuando se utiliza una plantilla.

Vamos por partes, según la definición de Colección 'es un conjunto de cosas' o lo que es lo mismo 'un conjunto de items', entonces nuestra colección (TCollection) estará compuesta por una serie de items (TCollectionItem). La colección así creada después la podremos utilizar dentro de un componente (TComponent).

La colección que nosotros crearemos se va a utilizar para sustituir determinadas cadenas de texto que se habrán introducido en una plantilla(fichero de texto con formato html) de las que utiliza el componente (THTMLExport) por otro texto o código html (o javascrit, etc). Cada item tendrá dos propiedades una ***cadena*** que representa el texto a sustituir y el ***texto/código*** que lo sustituye.

Por otra parte la colección deberá implementar un método que permita leer todos los items de un fichero.

La clase de item que nosotros crearemos tendrá esta definición:

```
TTemplateItem=class(TCollectionItem)
   private
     FVariable:string; // Cadena a sustituir
     FLines:Tstrings;  // texto/Código
   protected
      procedure SetLines(Value: TStrings);
      function GetDisplayName: string; override;
   published
      property Variable:string read Fvariable write FVariable;
      property Lines:Tstrings read Flines write SetLines;
   public
      constructor Create(Collection:TCollection);override;
      destructor Destroy;override;
end;
```

Como ya hemos dicho, tenemos dos propiedades ***Variable*** (cadena a sustituir) y ***Lines*** (texto/código que sustituye)

Además, como siempre o casi siempre, tenemos dos métodos públicos que debemos sobreescribir ***Create*** y ***Destroy*** que se encargan de crear y liberar los objetos de esta clase, más adelante veremos su implementación.

Por otra parte tenemos otros dos métodos, en este caso protegidos, que son ***SetLines*** que se encargará de asignar el código sustitutorio en la propiedad Lines y otro llamado ***GetDisplayName***, esta función va a servir para devolver al examinador de objetos un

nombre para cada item, es heredada. Por defecto nombra a los items con el nombre de la clase(TTemplateItem en nuestro caso), pero esto se puede cambiar sobreescribiendo esta función como veremos más adelante.

Una vez que tenemos la estructura (clase) de nuestros items o elementos de nuestra colección, deberemos definir la colección como tal, ésta también puede tener propiedades propias (valga la redundancia). En nuestro caso la colección va a tener una propiedad nueva ***FileName,*** que designará, en caso de rellenarse, el nombre del fichero con las distintas cadenas a sustituir y el código. Además otra propiedad llamada ***ClearBefore*** que nos indicará si debemos conservar los items que tuviéramos antes de realizar la carga de datos o no.

El fichero (FileName) tendrá el formato siguiente :

```
#var:texto1
Linea11
Linea12
Linea13
#var:texto2
Linea21
Linea22
Linea23
Linea24
....
#var:textoN
LineaN1
...
LineaNM
```

Dónde '*Texto1..TextoN'* son las distintas cadenas que tendremos que sustituir y las líneas que están entre dos '*#var'*: son el código.

Entonces la definición de nuestra clase colección será :

```
TTemplateReplace=Class(TCollection)
   private
      FOwner:TPersistent;    // Propietario de la colección
      FFilename:TFilename;   // Fichero de cadenas
      FFileContent:Tstrings;
      FClearBefore:Boolean;  //Borrar antes de cargar
      function GetItem(Index:Integer):TTemplateItem;
      Procedure SetItem(Index:Integer;Value:TTemplateItem);
      procedure SetFilename(value:tfilename);
   Protected
      function GetOwner:TPersistent;override;
   Public
      Constructor Create(Owner:TPersistent);
      Destructor Destroy;override;
      Function Add:TTemplateItem;
      //Items de la colección
      property Items[Index:Integer]:TTemplateItem read Getitem
                                    write SetItem; default;
   published
      Property FileName:TFilename read FFilename write SetFilename;
      Property ClearBefore:Boolean read FClearBefore write FClearBefore
                                 Default true;
end;
```

Pasamos a escribir y explicar el código fuente de estas nuevas clases.

TTemplateItem.

Como ya hemos dicho esta clase implementará cada uno de los items de nuestra colección.

```
Constructor TTemplateItem.Create;
begin
   inherited;
   FVariable:='';
   FLines:=Tstringlist.create;
end;
```

El procedimiento Create crea el item y lo inicia, en nuestro caso pone la Variable a cadena vacía y crea la estructura de Lines como un StringList.

```
destructor TTemplateItem.Destroy;
begin
   flines.Free;
   inherited;
end;
```

El procedimiento Destroy libera los recursos.

```
procedure TTemplateItem.SetLines(Value: TStrings);
begin
  FLines.Assign(Value);
end;
```

El procedimiento SetLines asignará a la variable FLines de nuestro objeto el valor que se le pasa.

```
function TTemplateitem.GetDisplayName: string;
begin
  Result := FVariable;
  if Result = '' then Result := inherited GetDisplayName;
end;
```

Como dijimos anteriormente esta función es la encargada de que en el inspector de objetos aparezcan nombres distintos para cada item de nuestra colección. Si no hubiéramos re-escrito esta función en el inspector de objetos aparecerían todos los items con la leyenda TTemplateItem. Nosotros lo que vamos a hacer es que la leyenda que aparezca sea la de la cadena de texto a sustituir (por ejemplo : Texto1, Texto2,... TextoN). Para ello devolvemos el valor de FVariable, si esta fuera una cadena vacía, devolvemos el valor por defecto (TTemplateItem).

Pues la clase no tiene más que explicar, como veis es muy simple.

TTemplateReplace

Esta clase implementará la nuestra colección.

```
constructor TTemplateReplace.Create(Owner:TPersistent);
begin
   inherited Create(TTemplateItem);
   Fowner:=owner;
   FFileContent:=tstringlist.Create;
   FFilename:='';
   FClearBefore:=True;
end;
```

El procedimiento Create creará un objeto de nuestra clase colección, para ello llama al procedimiento heredado y después inicia y/o crea variables, en nuestro caso guardaremos el propietario (FOwner) que será el componente en el que aparecerá nuestra colección, una variable de trabajo FFileContent que nos servirá para trabajar de forma más amigable con el fichero desde el que leeremos las cadenas a sustituir y el texto sustitutivo, Iniciamos a vacío el nombre de este fichero y como Verdadero FClearBefore, para que limpie los posibles items anteriores a la lectura del fichero antedicho.

```
Destructor TTemplateReplace.Destroy;
begin
   ffilecontent.Free;
   inherited destroy;
end;
```

El procedimiento Destroy, como de costumbre, liberará recursos.

```
Procedure TTemplateReplace.SetFilename(value:TFilename);
   Procedure LoadVars;
   var i,index:integer;
       v:boolean;
   begin
      v:=false;
      if ClearBefore then
      begin
         self.Clear;
         index:=-1;
      end
      else
         index:=self.Count-1;
      for i:=0 to ffilecontent.Count-1 do
      begin
         if pos('#VAR:',uppercase(ffilecontent.Strings[i]))<>0 then
         begin
             inc(index);
             v:=true;
             self.Add;
             self.Items[index].Variable:=copy(ffilecontent.Strings[i],
                    pos('#VAR:',uppercase(ffilecontent.Strings[i]))+5,
                    length(ffilecontent.Strings[i]));
         end
         else
            if v then
               self.Items[index].Lines.Add(ffilecontent.Strings[i]);
```

```
      end;
   end;
begin
   if value<>'' then
   begin
      if (value<>ffilename) and FileExists(value) then
      begin
         ffilename:=value;
         ffilecontent.LoadFromFile(value);
         loadvars;
      end
   end
   else
      ffilename:='';
end;
```

Este procedimiento asigna el nombre del fichero de donde extraer los datos y los guardará como items en nuestro objeto. Para ello comprueba el valor que se le pasa y si no es nulo y es distinto del que ya tiene y el fichero existe, entonces asigna el valor a la variable FFilename, carga el fichero en la variable de trabajo FFileContent y llama al procedimiento LoadVars que se encarga de transformas esos datos leídos en items de nuestra colección.

El funcionamiento del procedimiento LoadVars es el siguiente :

1.- Asignamos el valor False a una variable booleana (v) que nos servirá para desechar posibles líneas anteriores a la primera cadena de sustitución, por ejemplo, si tuviéramos el fichero siguiente :

```
Estas son
Unas líneas a no
Tener en cuenta
#var:Texto1
Líneas
```

Las primeras líneas hasta #var:Texto1 no se tendrán en cuenta.

2.- Comprobamos el valor de la propiedad ClearBefore para saber si debemos o no eliminar los items que ya pudiera haber en nuestro objeto. En caso de ser True, limpiamos el objeto (clear) y ponemos la variable que nos servirá de índice a –1 (Index), en caso contrario la iniciamos al nº de items que ya tiene nuestro objeto.

3.- Comenzamos a agregar items, para ello recorremos FFileContent, que no olvidemos que contiene el fichero leído, si la línea contiene el valor '#var:' crearemos un nuevo item : la variable 'v' la pondremos como true (para que a partir de este momento dejemos de desechar las posibles líneas anteriores a la primera cadena de sustitución, incrementamos nuestro Index que nos indicará el índice de nuestro item, agregamos el item (Add) y asignamos el texto que sigue a la subcadena #var: a la propiedad Variable del item añadido.

Si la línea no contiene la subcadena #var: entonces comprobamos el valor de 'v' si esta es true, nos indicará que la línea es del código sustitutorio, con lo añadiremos la línea a la propiedad Lines del item creado con anterioridad.

```
Procedure TTemplateReplace.SaveToFile(filename:TFilename);
var
   temp:tstringlist;
   i:integer;
begin
   try
      temp:=tstringlist.create;
      for i:=0 to self.count-1 do
      begin
         temp.add('#var:'+self.items[i].variable);
         temp.text:=temp.text+self.items[i].lines.text;
      end;
      try
         temp.SaveToFile(filename);
      except
         showmessage('Error. No se puede grabar');
      end;
      temp.free;
   except
      showmessage('Error. No se puede grabar');
   end;
end;
```

El procedimiento SaveToFile, creará un fichero de texto con los items de nuestra colección. Su funcionamiento es como sigue :

1.- Crea un objeto temporal de tipo TStringList que nos facilitará la tarea de convertir nuestros items en un fichero de texto.

2.- Recorremos el objeto colección y vamos añadiendo la subcadena '#var:' delante de cada una de las cadenas a sustituir y despues añadimos el texto/código sustitutorio[1].

3.- Creamos el fichero.

Las funciones siguientes GetItem y SetItem, lo único que hacen es devolver un objeto item y asignar un objeto item a nuestra colección, para lo que llaman a la función y procedimiento del mismo hombre heredado.

```
function TTemplateReplace.GetItem(Index:Integer):TTemplateItem;
begin
   result:=TTemplateItem(inherited GetItem(Index));
end;
```

```
Procedure TTemplateReplace.SetItem(Index:Integer;Value:TTemplateItem);
begin
   inherited SetItem(Index,Value);
end;
```

[1] Para agregar todas las líneas de la propiedad Lines utilizamos un pequeño truco que nos evita mucho código y es utilizar la propiedad Text que toda clase descendiente de TStrings tiene, esta propiedad devuelve todas las línea de un TStrings como una cadena el la que están todas las líneas del TStrings separadas por los caracteres de salto de línea y retorno de carro (#10#13).

La siguiente función devuelve el propietario de la colección (como ya hemos visto, cuando se crea la colección se le asgina un propietario).

```
Function TTemplateReplace.GetOwner:Tpersistent;
begin
   result:=Fowner;
end;
```

La funcion Add añade un nuevo elemento (item) a nuestra colección, para ello llama a la funcion Add heredada y devuelve el objeto creado.

```
Function TTemplateReplace.Add:TTemplateItem;
begin
   result:=TTemplateItem(Inherited Add);
end;
```

En las funciones anteriores Add y GetItem hemos empleado el polimorfismo que es una característica de los objetos (TTemplateItem(Inherited Add)).

Pues con esto ya hemos creado nuestra clase Collection. Otro procedimiento importante y que nosotros no hemos implementado es 'Assign' que permite asignar el valor de la propiedades de un objeto a otro, en nuestro caso podría ser de la forma siguiente :

```
Procedure TTemplateItem.Assign(source:Tpersistent);
Begin
   If source is TTemplateItem then
   Begin
      Variable:= TTemplateItem(source).Variable;
      Lines.Assign(TTemplateItem(source).Lines);
      Exit;
   End;
   Inherited Assign(Source);
End;
```

Resumen.

Toda colección (TCollection) se compone de una serie de elementos (TCollectionItem) por lo que a la hora de crear una colección deberemos implementar dos clases descendientes de estas.

Cada una de las clases podrán tener sus propiedades.

La clase TCollection, deberá implementar como mínimo una propiedad Items (que se puede llamar de otra manera), la función Add para añadir items a la colección y reescribir la función GetOwner para conocer el propietario de dicha colección.