Cargar/grabar definición componentes

Con las siguientes funciones y procedimientos vamos a poder guardar/recrear la definición (valor de propiedades, objetos) de un componente concreto y asignárselo al mismo objeto o a otro de la misma clase.

Las funciones y procedimientos que necesitaremos para hacer esto tendrán que permitirnos :

- Pasar de la definición binaria del componente a la texto[1].
- El paso contrario, de la definición textual a la binaria[2].
- Grabar la definición textual en un fichero (la textual porque así podremos editarla, si queremos, con un editor de texto)[3].
- Leer la definición textual desde un fichero[4].

Las funciones y procedimientos entonces serán los siguientes :

function ComponentToString(Component: TComponent): string

Esta función transformará un componente en un string, o sea la definición de las propiedades y objetos de un componente a una cadena de caracteres.

```
function ComponentToString(Component: TComponent): string;
var
  BinStream:TMemoryStream; // Stream de memoria
  StrStream: TStringStream; // Stream de cadena
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    Try
      // Inicia la escritura del componente en el stream de memoria
      // con representación binaria
      BinStream.WriteComponent(Component);
      // Coloca el puntero al principio del stream
      BinStream.Seek(0, soFromBeginning);
      // Transforma el objeto de binario a texto
      ObjectBinaryToText(BinStream, StrStream);
      // Coloca el puntero al principio del stream
      StrStream.Seek(0, soFromBeginning);
      // Devuelve el componente en una cadena
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;
```

[1] function ComponentToString
[2] procedure StringToComponent
[3] Procedure SaveComponent
[4] Procedure LoadComponent

El siguiente procedimiento hará lo contrario, es decir, transformar una cadena en un componente. Logicamente la cadena debe contener la definición correcta de un componente.

procedure StringToComponent(Component: TComponent;Value: string);

```
procedure StringToComponent(Component: TComponent;Value: string);
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
Begin
  // Crea el stream de cadena partiendo del valor pasado al procedimiento
  // que será la cadena que contiene la definición del componente
  StrStream := TStringStream.Create(Value);
  Try
    BinStream := TMemoryStream.Create;
    Try
      // Transforma el objeto de texto a binario
      ObjectTextToBinary(StrStream, BinStream);
      // Coloca el puntero al principio del stream
      BinStream.Seek(0, soFromBeginning);
      // Crea el componente partiendo de objeto binario de memoria
      BinStream.ReadComponent(component);
    Finally
      BinStream.Free;
    end;
  finally
    StrStream.Free;
  End;
end;
```

El siguiente procedimiento graba la definición de un componente (propiedades y/u objetos que le pertenezcan) en un fichero que se le pasa como parámetro.

Procedure SaveComponent(Component: TComponent;filename:tfilename;obsave:boolean=true);

Dónde :

Component : Componente cuya definición queremos grabar

Filename : Ruta completa del fichero en el que grabaremos la definición del componente

Obsave : Se graban también los objetos que le pertenezcan(true) o no(false)

```
Procedure SaveComponent(Component: TComponent;filename:tfilename;
obsave:boolean=true);
var
   S:tstringlist;
   i,i1,i2:integer;
begin
   s:=tstringlist.Create;
   // Transformamos el componente en su definición textual
   s.Add(ComponentToString(Component));
   // Si no queremos grabar las definiciones de los objetos de los cuales es padre
   if not obsave then
   begin
      s.SaveToFile(filename);
```

```
      s.LoadFromFile(filename);
      i:=1;
      // En las definiciones de los objetos, primero están las propiedades del
      // objeto y después el resto de objetos que le pertenecen, por lo que
      // borramos todas las líneas a partir de que encontremos la palabra 'OBJECT'
      // menos la última línea que contendrá el finalizador (end).
      while (pos('OBJECT',uppercase(s.Strings[i]))=0) and (i<s.Count-1) do
         i:=i+1;
      if pos('OBJECT',uppercase(s.Strings[i]))>0 then
      begin
         i2:=(s.Count-3)-i;
         for i1:=0 to i2 do s.Delete(i);
      end;
   end;
   s.SaveToFile(filename);
   s.Free
end;
```

El siguiente procedimiento leerá desde un fichero la definición y se la asignará a un componente.

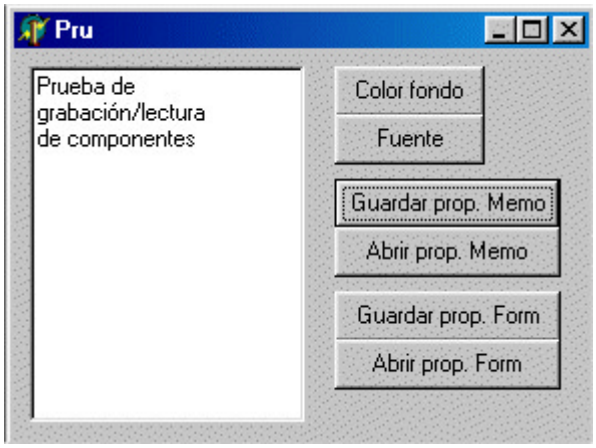
Procedure LoadComponent(Component:TComponent;filename:tfilename);

```
Procedure LoadComponent(Component:TComponent;filename:tfilename);
var
   S:tstringlist;
   i:integer;
   cad:string;
begin
   cad:='';
   s:=tstringlist.Create;
   // Carga la definición
   s.LoadFromFile(filename);
   // Sustituye el nombre del objeto (se le puede asignar a cualquier objeto
   // de la misma clase o que siendo de distinta clase tenga las mismas propiedades
   s.Insert(1,'object '+component.Name+': '+component.ClassName);
   for i:=1 to s.Count-1 do
      cad:=cad+s.Strings[i]+#10;
   try
      // recrea el objeto
      StringToComponent(component,cad);
   finally
      s.Free
   end;
end;
```

Bueno, y después de todo lo explicado, ¿para qué me sirve?. Pues las posibilidades son varias :

- Deshacer cambios (guardamos el componente antes de realizar modificaciones en el como podría ser antes de escribir en un campo de texto, y podemos dar la posibilidad de recuperar el texto anterior).
- Guardar/recuperar las propiedades de los componentes (posicionamiento/tamaño de un form,...).
- Clonar objetos.
- .....

El siguiente ejemplo muestra el funcionamiento de estas funciones :



En este ejemplo podemos guardar y recuperar las propiedades (no guardaremos los objectos contenidos en él) del form o las del campo memo.

Puedes descargarte el código de http://www.i-griegavcl.com/downloads/ejcomp.zip