Copy and paste of objects in Lazarus

Michaël Van Canneyt

March 28, 2009

Abstract

Copy and paste is a natural concept when editing text. It is equally natural to be able to copy and paste a set of objects in an application which manipulates and designs a set of objects. This article shows how to do this in Lazarus, using the CD-Cover designing application as a model.

1 Introduction

Copy and paste is a built-in functionality in any text control in Windows or X-Windows. It is not built-in for graphical objects or any other custom object. There the programmer has to implement this functionality by himself.

The CD-Cover application, which was treated in previous articles, is a good case study for this: the cover objects (label, image, track-list) that have been dropped on a page of the CD-Cover, are typical things which one may want to copy and paste to the same page or to another page of the CD-Cover project.

Fortunately, the basic idea is of course not so difficult: after all, each object used in the application is a descendent of `TComponent`, so it can be streamed to a file, and can again be read from this file, so it should be possible to stream it to the clipboard, and read it again from the clipboard. In this article, this is exactly what will be done, and some pitfalls will be identified and dealt with.

2 The Lazarus built-in facilities

Copy and paste of an object is easily done; The Lazarus clipboard implementation contains this functionality by default: any component can be streamed to the clipboard, and can similarly be retrieved from the clipboard: the clipboard implementation has several methods to do this. They are declared as follows:

```
Function SetComponent(Component: TComponent): Boolean;
Function SetComponentAsText(Component: TComponent): Boolean;
```

Both functions stream a component to the clipboard, and return `True` if this was successful. The first writes a binary stream, the second writes the component as text. It uses the `CF_Component` and `CF_Text` clipboard formats for this, which can be retrieved with the following functions in the clipbrd unit:

```
function CF_Text: TClipboardFormat;
function CF_Component: TClipboardFormat;
```

To paste the components which are on the clipboard, the following functions can be used:

```
Function GetComponent(Owner, Parent: TComponent): TComponent;
Procedure GetComponent(var RootComponent: TComponent;
                  OnFindComponentClass: TFindComponentClassEvent;
                  Owner: TComponent = nil;
                  Parent: TComponent = nil);
```

A function, similar to the second one listed here, exists to read a component that was streamed as text. The first function is a simplified version of the second. The `Owner` and `Parent` arguments specify the `Owner` and `Parent` properties of the component that will be read from stream: these properties will automatically be set. The function form creates the the new component and returns the newly created instance. The procedural form accepts an already created root component. The `OnFindComponentClass` callback can be used to map type names (for instance `TTrackList`) on actual class pointers, in case the classes in the stream have not been registered in the streaming system.

This function can be used to stream a single component. So, what to do when multiple components must be streamed, such as the list of currently selected objects in the CD-Cover designer ? The streaming format does not - by default - allow to stream multiple components one after the other.

3 Streaming multiple components

There is a relatively simple way to stream multiple components. To see how this can be done, the streaming mechanisms internal workings must be examined. When a component (say `A`) is streamed, the streaming system does not stream by default the components that are owned by the component `A`. Instead, it calls the `GetChildren` function, which is defined as follows in `TComponent`:

```
TGetChildProc = procedure (Child: TComponent) of object;

procedure GetChildren(Proc: TGetChildProc; Root: TComponent); dynamic;
```

`Proc` is a callback which should be invoked for each child component that should be streamed when the component `A` is streamed.

This mechanism allows us to define a dummy component to stream the current selection:

```
TSelection = Class(TComponent)
Protected
   procedure GetChildren(Proc: TGetChildProc;
                         Root: TComponent); override;
Public
  Property Selection : TSelectionlist;
end;
```

The `TSelectionList` class was defined in the first article on the `CD-Cover` application, and is a simple list of all `TCoverObjects` that make up the selection. The `GetChildren` procedure should simply loop over all elements in the selection list:

```
procedure TSelection.GetChildren(Proc: TGetChildProc;
                                 Root: TComponent);
```

```
Var I : Integer;

begin
  If Assigned(Selection) then
    For I:=0 to Selection.Count-1 do
      Proc(FSelection.CoverObjects[i]);
end;
```

So, armed with this small component, implementing a rudimentary copy and paste can be implemented in the `TCoverEditor` class.

The procedure to copy the selection to the clipboard is quite simple:

```
procedure TCoverEditor.CopySelectionToClipboard
  (AsText : Boolean = False);

Var
  Sel : TSelection;

begin
  FLastCopyPage:=FCurrentpage;
  Sel:=TSelection.Create(Nil);
  try
    Sel.Name:='Selection';
    Sel.Selection:=FSelection;
    If AsText then
      Clipboard.SetComponentAsText(Sel)
    else
      Clipboard.SetComponent(Sel);
    Sel.Selection:=Nil;
  finally
    Sel.Free;
  end;
end;
```

The first thing that is done is saving the currently visible page in a temporary variable. The reason for this will become apparent when the paste function is discussed. After this, a `TSelection` instance is created and is saved to the clipboard using the standard functions discussed above.

To know whether something is available on the clipboard which can be pasted, the following function is implemented:

```
Function TCoverEditor.ClipboardContentOK : Boolean;

begin
  Result:=ClipBoard.HasFormat(CF_Component) or
          Clipboard.HasFormat(CF_Text);
end;
```

In case the available format is text format, it is of course not possible to know whether it is a streamed component, this will only be clear when streaming is actually attempted.

To paste the selection from the clipboard, the `PasteSelectionFromClipboard` function is implemented:

```
procedure TCoverEditor.PasteSelectionFromClipboard;
```

```
Var
  Sel : TSelection;
  I : Integer;
  C : TCoverObject;

begin
  Sel:=TSelection.Create(Nil);
  try
    If not GetSelectionFromClipBoard(Sel) then
      Exit;
    ClearSelection;
    For I:=Sel.ComponentCount-1 downto 0 do
      begin
      C:=Sel.Components[i] as TCoverObject;
      Sel.RemoveComponent(C);
      FCDCover.InsertComponent(C);
      C.SetParentComponent(FCurrentPage);
      If (FCurrentpage=FLastCopyPage) then
        begin
        C.Left:=C.Left+ToUnits(8,CurrentDPI);
        C.Top:=C.Top+ToUnits(8,CurrentDPI);
        end;
      AddToSelection(C);
      end;
    FCurrentPanel.Repaint;
  finally
    Sel.Selection:=Nil;
    Sel.Free;
  end;
end;
```

This function starts by creating a `TSelection` instance and reading it from the clipboard with the `GetSelectionFromClipBoard` function. It will return `False` if something went wrong. The `Sel` component will now own a number of `TCoverObject` objects.

These must be inserted into the `FCDCover` component, and put on the correct page: this is done in a loop by changing the ownership (using the `RemoveComponent` and `InsertComponent` calls). The current page is set with the `SetParentComponent` call.

At this point, the reason becomes clear why the `FLastCopyPage` variable was introduced. If the selection is pasted on the same page as the one from which it was copied, the objects would appear on the exact same place. To avoid this, the objects are shifted with 8 pixels to the right and bottom (a small coordinate transformation is needed). There are other ways to deal with this eventuality, by e.g. repositioning all objects around the mouse pointer location.

Finally, the cover object is added to the current selection - which was cleared before the loop was started. This means that the new objects can be manipulated (e.g. moved) at once when the paste operation is finished.

The `GetSelectionFromClipboard` function is defined as follows:

```
Function TCoverEditor.GetSelectionFromClipboard
    (Sel : TSelection) : Boolean;
```

```
begin
  Result:=False;
  try
    With ClipBoard do
      if HasFormat(CF_Component) then
        GetComponent(Sel,@findclass,
                     FCDCover,FCurrentPage)
      else
        GetComponentAsText(Sel,@findclass,
                           FCDCover,FCurrentPage);
    Result:=True;
  except
    // Silently ignore errors.
  end;
end;
```

It simply uses the predefined clipboard functions. The `FindClass` callback function (which must be supplied) simply points to the default `FindClass` function in the classes unit:

```
procedure TCoverEditor.FindClass(AReader : TReader;
  Const AClassName : String; Var AClass :  TComponentClass);

begin
  AClass:=TComponentClass(Classes.FindClass(AClassName));
end;
```

With this, the actual work of copying and pasting to and from the clipboard is done.

Adding a couple of actions, menu items and toolbar buttons to the main form that call these functions is an easy operation, and will not be shown here, the interested reader can consult the sources that are on the disc accompagnying this issue. The result is shown in figure 1 on page 6.

4 Caveat

The copy/paste as implemented here works fine, until an object is pasted that contains references to other components, such as the `TTrackList` : A track list has a reference to a disc in the collection of discs in the CD-Cover project. When copied, the reference to the disc is gone.

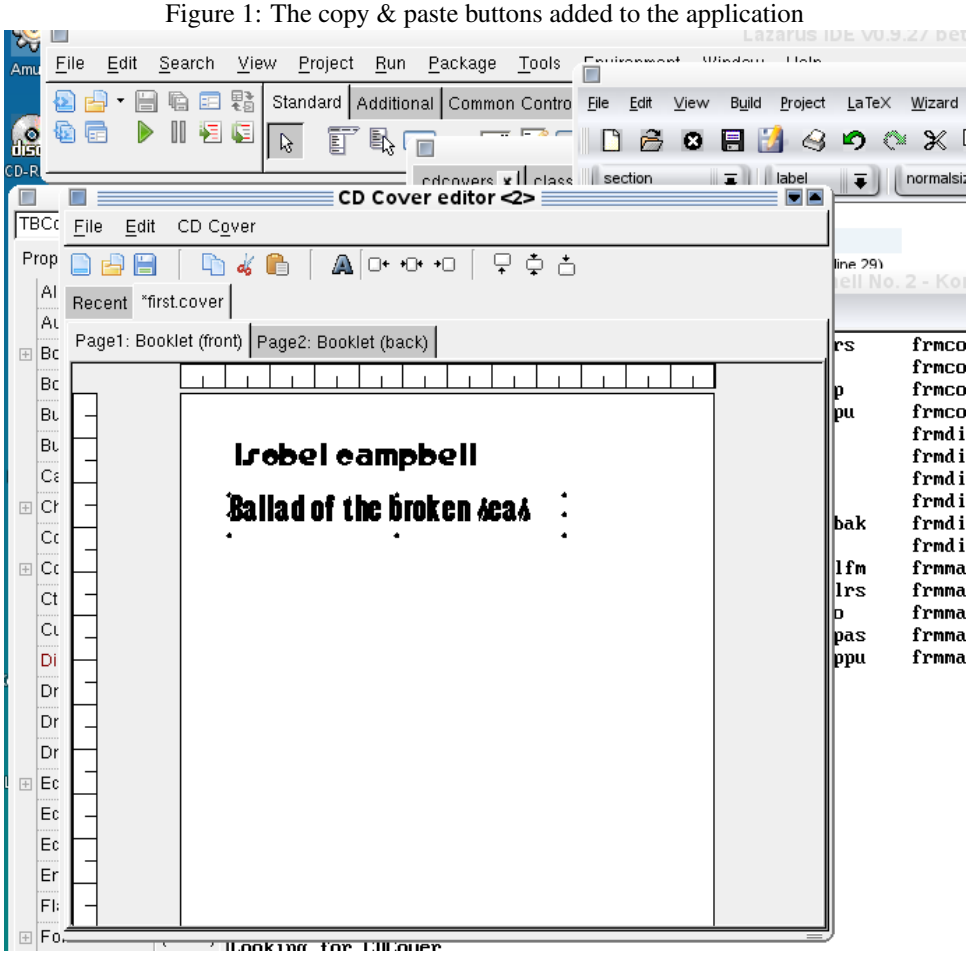
This is because the reference to the disc cannot be resolved when the track list is being read from the stream. When writing the `Disc` property of a `TTrackList` to the stream, something like this is written:

```
CDCover.Disc1
```

This tells the streaming system that, when reading from the stream, it should look for a component named `CDCover`, and then look in it's child components for the `Disc1` component, and when found, copy the instance pointer to the property `Disc` of the `TTrackList` instance.

The search procedure starts at the `Sel` instance, because that is the component that is being read from the stream. Since the `CDCover` component is not a child component of

Figure 1: The copy & paste buttons added to the application



the `Sel` component, it is not found. In such cases the streaming system starts looking for the component through a global search mechanism.

It is possible to hook into this global search mechanism, and to resolve the name 'CDCover' to the `TCDCover` instance that is being edited. To hook into the search mechanism, the `RegisterFindGlobalComponentProc` routine from the classes unit can be used. It is defined as follows:

```
TFindGlobalComponent = function(const Name: string): TComponent;

procedure RegisterFindGlobalComponentProc
   (AFindGlobalComponent: TFindGlobalComponent);
```

The `AFindGlobalComponent` callback should find the component named `name`, and return the instance that corresponds to that name. This mechanism can be used to point the streaming system to the CDCover instance that is being edited, using the following function:

```
Var
  SelCover : TCDCover;

function FindRefFromSelection(const Name: string): TComponent;

begin
  Result:=Nil;
  If (CompareText(Name,'CDCover')=0) then
    Result:=SelCover;
end;
```

The working of this function is of course straightforward. It is registered in the initialization section of the covereditor unit:

```
initialization
  RegisterFindGlobalComponentProc(@FindRefFromSelection);
end.
```

Now the `GetSelectionFromClipboard` routine can be changed so it initializes the `SelCover` prior to reading the selection from the clipboard:

```
Function TCoverEditor.GetSelectionFromClipboard
  (Sel : TSelection) : Boolean;

begin
  Result:=False;
  try
    SelCover:=FCDCOver;
    try
      With ClipBoard do
        if HasFormat(CF_Component) then
          GetComponent(Sel,@findclass,
                       FCDCover,FCurrentPage)
        else
          GetComponentAsText(Sel,@findclass,
                             FCDCover,FCurrentPage);
      Result:=True;
```

```
    Finally
      SelCover:=Nil;
    end;
  except
    // Silently ignore errors.
  end;
end;
```

After this change, all references between exiting objects in the CD-Cover component can be resolved by the streaming mechanism, and it is possible to copy a track list.

Note that if one would copy a track list from CD-Cover A to CD-Cover B, and the target CD-Cover B does not have a disc, obviously the reference to the disc still cannot be resolved.

5 Conclusion

Copying components to and from the clipboard is quite easy - lazarus has the functionality practically built-in: it can store components on the clipboard in a binary or text format. When pasting, it also takes care of many details such as renaming components when a component with the same name already exists. The main difficulty consist of resolving possible cross-component references, which require some extra coding, and even this is very easily done, as shown in this article.