ROOT — New GUI design

Marek Biskup — *m.biskup@zodiac.mimuw.edu.pl*

August 8, 2003

Contents

1 Introduction

This document contains some consideration about new graphical user interface for ROOT.

2 Requirements

In this section I describe what is needed in the GUI.

2.1 Usecases

There are following usecases that must be present in new GUI:

1. Changing object attributes
   Attributes are for example: line color, fill color, pattern, line style, size, position on canvas, name etc.
   (a) user selects an object on a canvas
   (b) user edits its attributes
2. Changing drawing attributes
   Drawing attributes are attributes that are given to newly created objects. For example, if user changes fill color, than every new circle will have that color. Most of drawing attributes are currently kept in gStyle, but some are not, for example number of paves or border size in TPavesText. Drawing attributes are for example: line color, fill color, pattern etc, marker size etc.
   (a) user selects a tool for editing attributes
   (b) user edits tool's attributes

2.2 Nonfunctional requirements

Following nonfunctional requirements should be fulfilled:

- everything must be easy for the user

- compatibility with previous version must be retained
- previous gui elements (TAttCanvases) should be retained and user must be able to choose previous GUI style.
- user must be able to see numbers associated with parameters. The numbers are important in ROOT command line.

2.3 Attributes

There are two kinds of attributes:

- Attributes in TAttXXX — these attributes are usually present in more than one class.
- Other attributes — these are specific for a class.

Because first attributes are common to many classes, widgets for changing it may be present on the screen all the time (sometimes inactive). Their alignment on the screen should be the same for different objects so that user can get used to it.

The second attributes requires a window specific for a class. The window can be aligned somewhere on the canvas window or can be separate. If user selects another object, the window must change immediately to show new attributes with values taken from selected object. For user's comfort, the first group of attributes must be present in the same window or at least very close.

2.4 Graphic object browser

Usually in computer applications, when user deals with hierarchical structure of object, he can access the object also via object browser. The browser is made using Tree List widget. When there are many objects on the canvas it would be easier for the user to use browser than to click on the screen. There are also some problems with selecting an object with nonzero border width because it's not enough to click on the border but user must click in a strictly defined region of the border. another example is when the user has to click in really random places of the screen to drag TCirlyArc. Sometimes also the object is hidden behind another one and it's not easy for the user to find it.

2.5 Toolbar

Toolbar containing most often used commands from the menu is very useful for the user. It should have the following buttons:

- new
- open
- save
- print

- undo
- redo
- copy
- paste
- switch buttons for tool windows

and may contain some other frequently used commands.

In root we don't have copy and paste but we have Clone method for each object. Copy can be implemented by cloning existing object to a buffer (if there is an object in a buffer it should be freed) and paste by attaching an object (or another it's clone) to a pad.

3 Problem with current GUI

Current use of TAttCanvases for changing properties is quite inconvenient for the user, because

- Options appear in separate windows. These windows are placed by Window Manager in random places and user cannot get used to one position. Finding the window on the screen is quite annoying.
- TAttCanvases are large and take too much space on the screen.
- Every time user want to change object attributes he must choose option SetXXXAttributes from context menu. It takes too much time since context menus contains many options and the appropriate option is not easy to find.
- There are four different TAttCanvases and user has sometimes to use even three of them to change the options he wants to change.
- Buttons on TAttCanvases are not lowered when user clicks on them. Sometimes he doesn't know if the option was set (especially when there is nothing changed on the screen and user has to resize window to make new settings visible – it happens often).
- User cannot see current selection unless he selects something (but of course an object has color before setting new one).
- Canvases have problems with redrawing, especially when having some pattern style set (see TAttFillCanvas).
- There are (almost) no numbers of parameters visible on TAttXXXCanvases, and they are important in ROOT command line.
- Setting parameters that are not on TAttXXXCanvases is even worse because user has to select parameter from context menu and the type the value in simple text field. User cannot see more than one parameter value at the same time.

4 Problems with new solution

4.1 Single or multi document interface?

There are two different approaches to the User Interface: single document interface (SDI) or multi document interface (MDI).

In single document interface user can edit only one document at one time. If he wants to edit another document he has to start additional instance of application. In some cases the applications are completely separate, and in some they can communicate. What is important, all the windows have exactly the same set of widget. SDI is used when user usually edits only one document or the documents are not connected. Example of SDI is Microsoft Word 2000, MSIE, Mozilla.

In multi document interface one application can handle multiple documents. That means that we have one set of widget (menu, toolbox etc.) which can be used for all edited documents. Usually such application has a main window with child windows inside (for example Adobe Photoshop, Microsoft Word 97, Microsoft Visual Studio 6.0, SUN Forte for Java, Opera, and also Mozilla). That windows cannot be moved outside of the main window. Sometimes there is only small application window (maybe more than one) and documents windows are normal windows that can be freely moved on the screen (GIMP, Borland Delphi). MDI is used when user edits more than one documents at the same time, has to exchange date between them (like in graphics programs) or the documents are tightly connected (like in development tools),

In root we have now MDI without main window. In my opinion that is rather inconvenient for the user because sometimes it's hard to find appropriate window on the screen. Having main window is out of consideration because users make intensive use of the ROOT command line. Moving command line into main window would be too difficult and rather pointless because users would like to use root also without GUI.

What we can consider is if to put all the widgets onto each canvas window (like in SDI) or if to create separate windows for the tools (like in MDI without main widow).

Below I stated pros and cons SDI interface for ROOT:

Pros:

- Tools are easy to find for the user - they are on the same window as canvas.
- The interface is similar to those from MS Word and MS PowerPoint.
- user can set drawing options for each canvas separately

Cons:

- If user opens many documents, tools take too much space on the screen
- The canvas is smaller when tool are on the same window
- User may be confused seeing many toolboxes - separate for each canvas
- The MDI interface is similar to those from Graphics programs like GIMP and Photoshop.

- ROOT has MDI GUI now so it would require less changes and it would be easier to retain compatibility with previous versions.
- User may want to set drawing options all the canvas in the same time (for example default fill colour)

4.2 Highlighting edited object

Now there is nothing to highlight selected object and that should be done. Below I wrote several proposals:

- Writing object name and type somewhere — this is not actually highlighting but user will know what he changes. This solution is the easiest to implement.
- Drawing a frame around the object.
- Drawing a frame with animating border — a frame will be drawn with dashed line and every second the dashes will move.
- Drawing the object in different color/linestyle/fillstyle — this is not very good because user can use various attributes for his drawings.
- Drawing big control points — usually the control points are used then to resize the object. This idea is used in OpenOffice and in most applications.

In most programs selected object can be moved using keyboard and deleted by pressing Delete on the keyboard.

4.3 Editing multiple objects

In most programs user can select more that one object (usually when Control key is pressed or when he ”draws” box containing the objects) and edit their attributes at the same time. If the objects are of different type only the common attributes are shown. If current attributes value is different for some of selected objects user don’t see the value but can set the new one. If he does the value is assigned to all selected objects.

It’s not difficult to implement selection and editing multiple objects.

4.4 Setting drawing (global) attributes

Usually if no object is selected than user can edit global drawing options. User can unselect the object clicking in a place where no object is present. Unfortunately in ROOT Canvas itself is also an object with attributes so there is no easy way for unselecting. Here are some proposals:

- choosing a primitive from the toolbox
- clicking on a special button
- clicking middle mouse button somewhere (easiest for the user)

- using middle mouse button (like now) for selecting pads and canvases; then clicking with left mouse button on pad or canvas will cause unselection. This and previous way is a bit dangerous because not everyone has mouse with middle mouse button.

5 Required changes in base classes of ROOT and compatibility consideration

5.1 TGCanvas

New members:

- fSelectedForProperties — object that is selected (only for SDI)
- fShowToolBar — if to show toolbar (only for SDI)
- fShowSideFrame — if to show side frame with tools (only for SDI)
- fEditorMode — editor mode for canvas (only for SDI)
- fStyle — default drawing style (only for SDI)

For MDI all this variables must be present in a global variable.

Function HandleInput — some code must be added for selecting object on the screen. also a signal SelectedForProporties() when an object is selected.

5.2 TPad

Code for showing selected primitive must be added.

5.3 TROOT

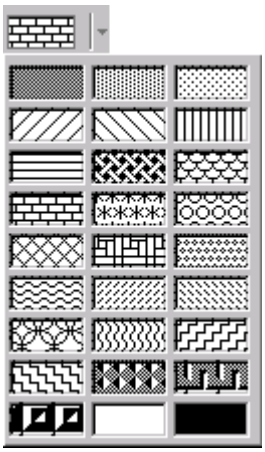
Function SetEditorMode should change editor mode also in active canvas (gPad->GetCanvas()).

6 New widgets

There are several widgets that are already written, and some require some more work.

6.1 TGPatternSelect

This is a small button with a fill style and arrow. When user clicks on the button a window with different fill styles appear. Then user can select one of the fill styles.



6.2 TGMarkerSelect

This is not written yet. It will be the same as TGPatternSelect, but with marker styles.

6.3 TGColorSelect

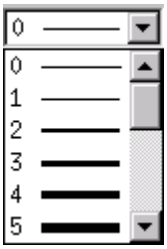
This was written some time ago.

6.4 TGLineStyleComboBox

A ComboBox with line styles.

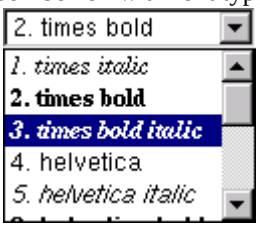
6.5 TGLineWidthComboBox

A ComboBox with line widths.



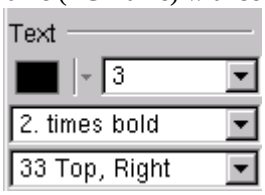
6.6 TGFontTypeComboBox

A ComboBox with font types.



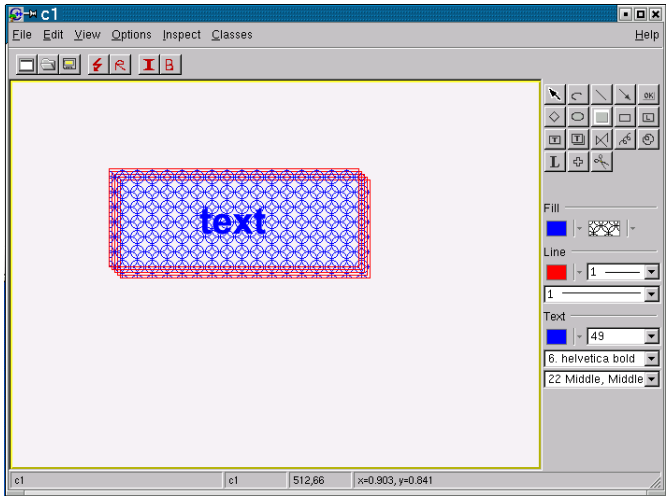
6.7 TGAttFrame

A frame (TGFrame) with some options for selected objects.

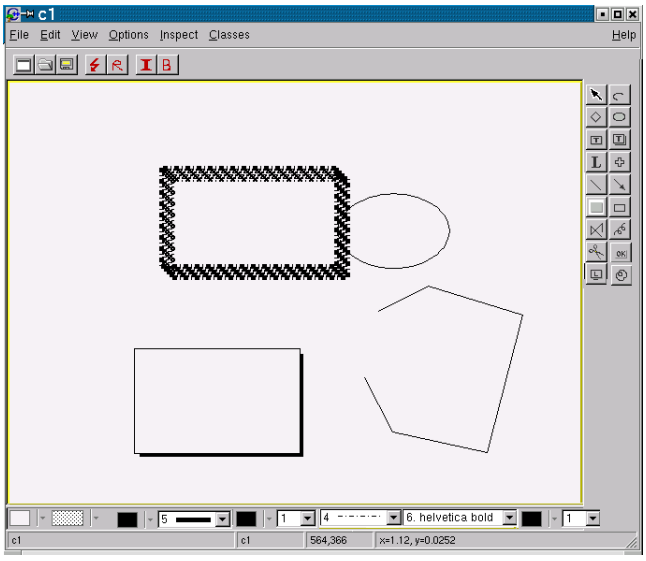


7 Proposed layout for SDI

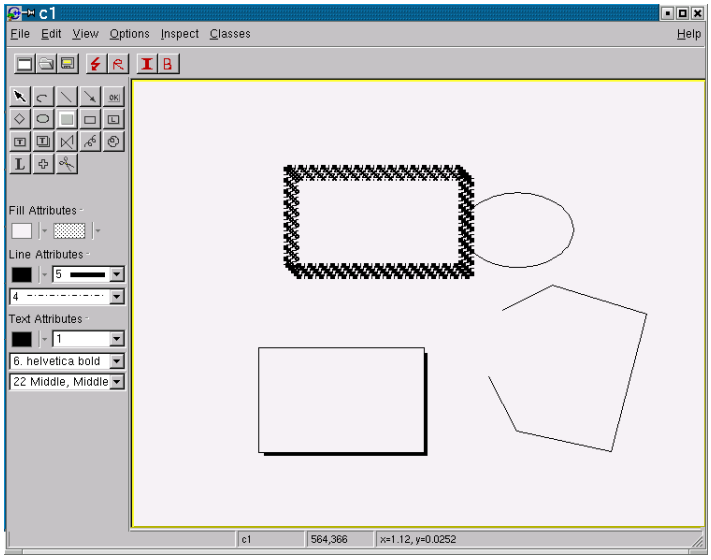
7.1 Layout 1



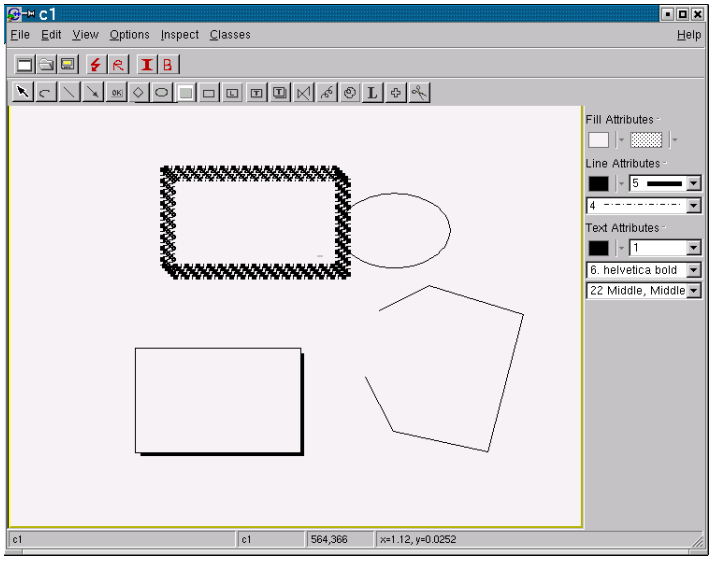
7.2 Layout 2



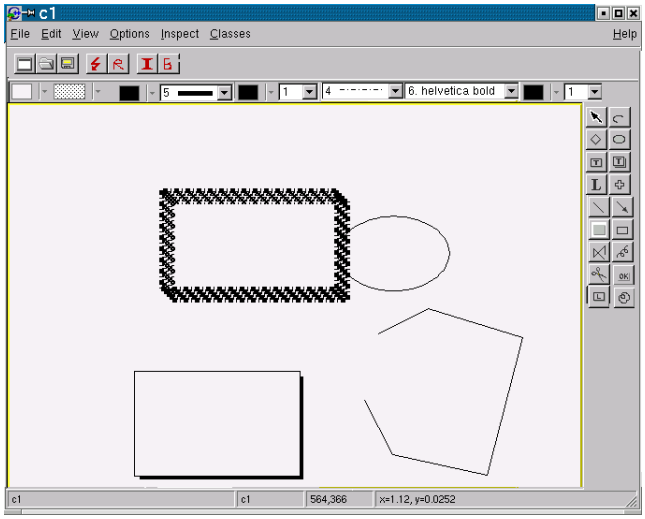
7.3 Layout 3



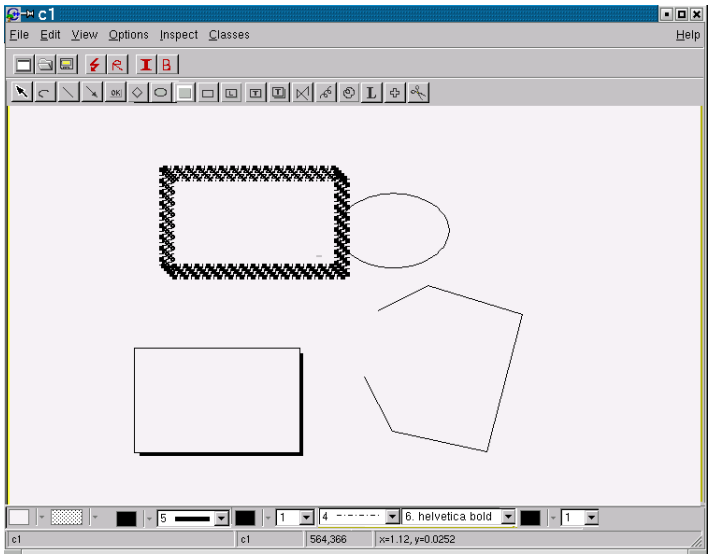
7.4 Layout 4



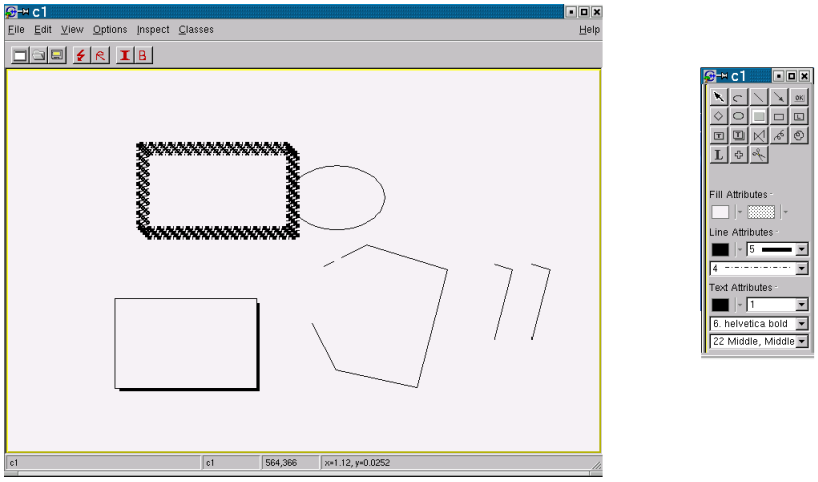
7.5 Layout 5



7.6 Layout 6



7.7 Layout 7



7.8 Layout 8