ROOT7 graphics hands-on

Sergey Linev

Content

- Basic concepts
- Examples from tutorials/v7/
- Small exercise

- Open questions/problems

RDrawable

- Drawing primitive in RPad/RCanvas
  - list of primitives :
    - `std::vector<std::shared_ptr<RDrawable>> fPrimitives;`

- Container of drawing attributes
  - int, double, bool, string

- Referencing data object (optional)
  - like RHistDrawable

- Has type, class (optional) and id (optional)
  - used in CSS evaluation

RDrawable attributes

- Color, line width, fill style, …
- Preserved in RAttrMap container
  - empty by default
- Accessed via RAttrBase class
  - has reference to RDrawable
  - like RAttrValue<int> or RAttrLine
  - contains default values for every field
- Lets consider RAttrLine as example

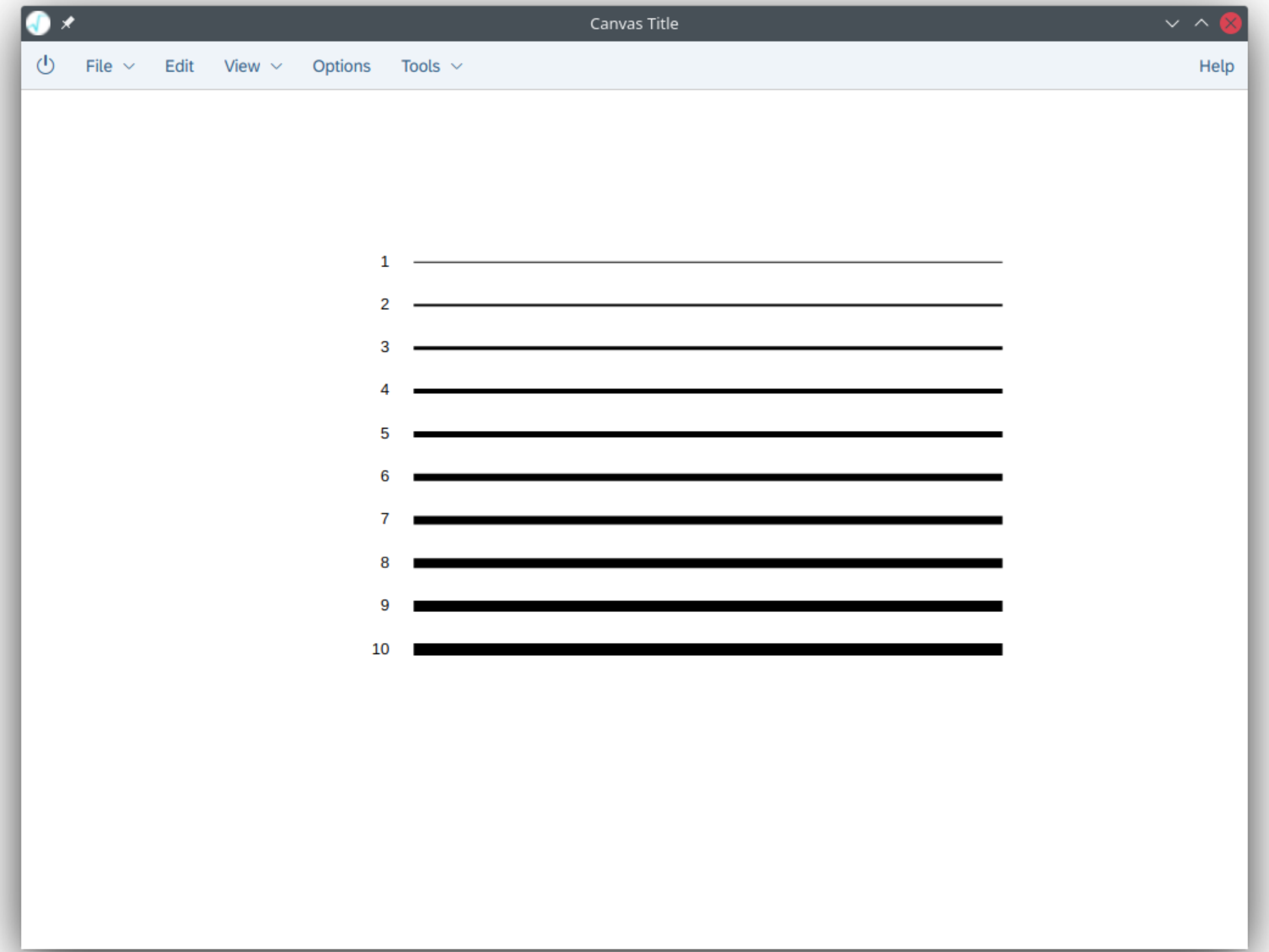
RAttrLine class

```
class RAttrLine : public RAttrBase {

   RAttrColor          fColor{this, "color"};      ///<! line color
   RAttrValue<double>  fWidth{this, "width", 1.};  ///<! line width
   RAttrValue<int>     fStyle{this, "style", 1};   ///<! line style

   /// macro defines all standard constructor/assign operators signatures
   R__ATTR_CLASS(RAttrLine, "line");

   ///The width of the line.
   RAttrLine &SetWidth(double width) { fWidth = width; return *this; }
   double GetWidth() const { return fWidth; }

   ///The style of the line.
   RAttrLine &SetStyle(int style) { fStyle = style; return *this; }
   int GetStyle() const { return fStyle; }   ///The color of the line.

   ///The color of the line
   RAttrLine &SetColor(const RColor &color) { fColor = color; return *this; }
   RColor GetColor() const { return fColor.GetColor(); }
   RAttrColor &AttrColor() { return fColor; }

};
```

RLine class

```
class RLine : public RDrawable {

   RPadPos fP1, fP2;                  ///< line begin/end
   RAttrLine fAttrLine{this, "line"}; ///<! line attributes

public:

   RLine() : RDrawable("line") {}
   RLine(const RPadPos &p1, const RPadPos &p2) : RLine() { fP1 = p1; fP2 = p2;   }

   const RAttrLine &GetAttrLine() const { return fAttrLine; }
   RLine &SetAttrLine(const RAttrLine &attr)   { fAttrLine = attr; return *this; }
   RAttrLine &AttrLine() { return fAttrLine; }
   ...
};


// See usage in macro tutorials/v7/lineWidth.cxx

auto line = canvas->Draw<RLine>(RPadPos(.32_normal, 1_normal*num), RPadPos(.8_normal , 1_normal*num));
line->AttrLine().SetWidth(i).SetColor(RColor::kRed);
```

tutorials/v7/lineWidth.cxx



tutorials/v7/lineWidth.cxx

```
// used in all macros
using namespace ROOT::Experimental;

// create Canvas
auto canvas = RCanvas::Create("Canvas Title");
double num = 0.3;

for (int i=10; i>0; i--){
   num = num + 0.05;
   // one can create object ourself
   auto text = std::make_shared<RText>(RPadPos(.3_normal, 1_normal*num), std::to_string(i));
   text->AttrText().SetSize(13).SetAlign(32).SetFont(52);
   canvas->Draw(text);

   // or let it create by templated Draw<T> method
   auto line = canvas->Draw<RLine>(RPadPos(.32_normal, 1_normal*num), RPadPos(.8_normal , 1_normal*num));
   line->AttrLine().SetWidth(i);
}

// show canvas
canvas->Show();
```

RPadPos

- 2D Position on the pad
  - horizontal + vertical components (RPadLength)

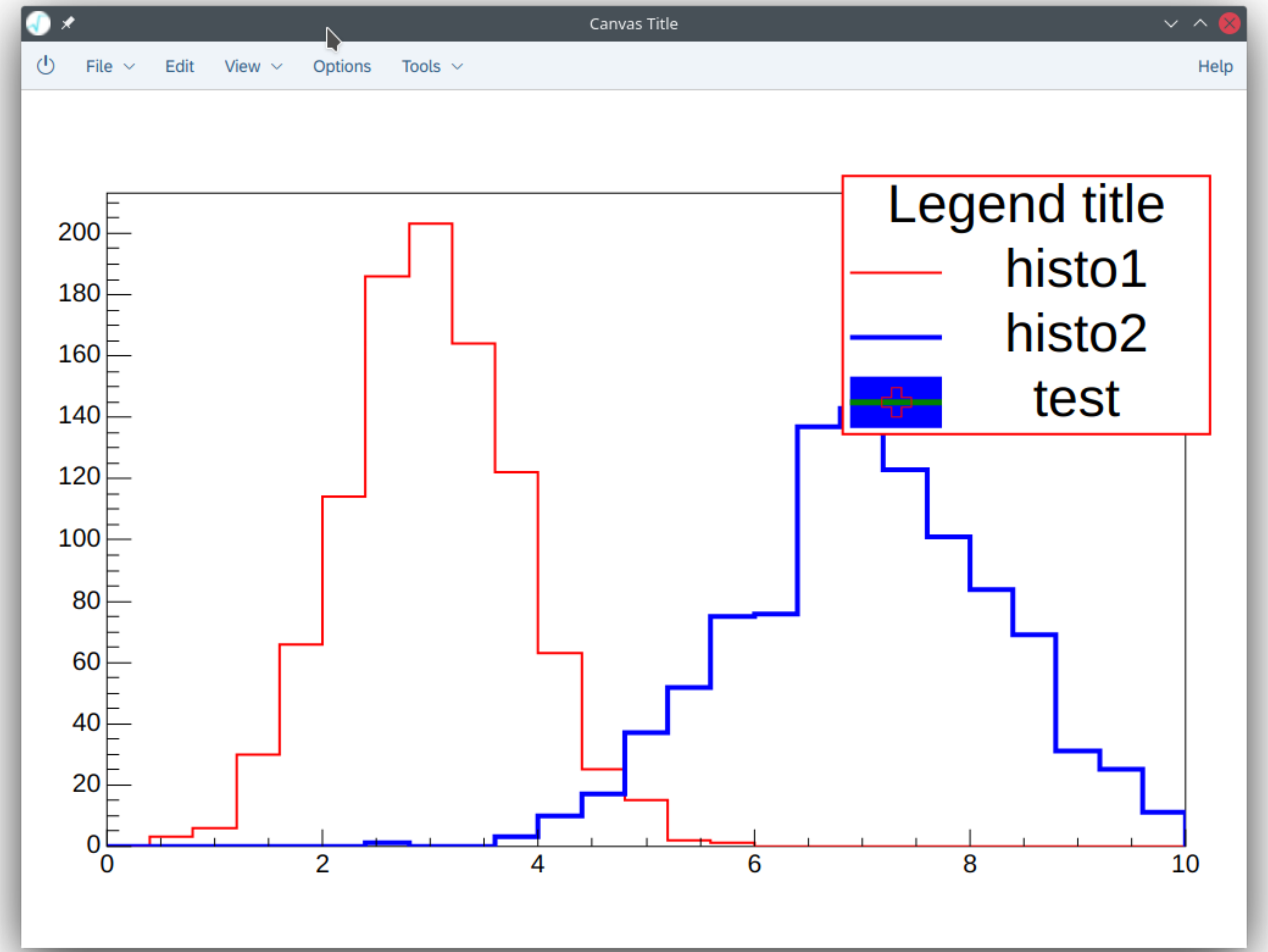
- RPadLength coordinates systems
  - normalized (0..1) like `0.5_normal`
  - pixels (0..N) like `100_px`
  - user* (axis_min…axis_max) like `5.3_user`

* not yet supported on client side

RCanvas

```
// create Canvas
auto canvas = RCanvas::Create("Canvas Title");

// add new drawables
. . .

// show canvas
canvas->Show();

// modify, add, delete drawables
. . .


// modify and update canvas
canvas->Modified();
canvas->Update();


// access in global list
RCanvas::GetCanvases()[0]->Update();
Rcanvas::ReleaseHeldCanvases();
```

tutorials/v7/draw_legend.cxx



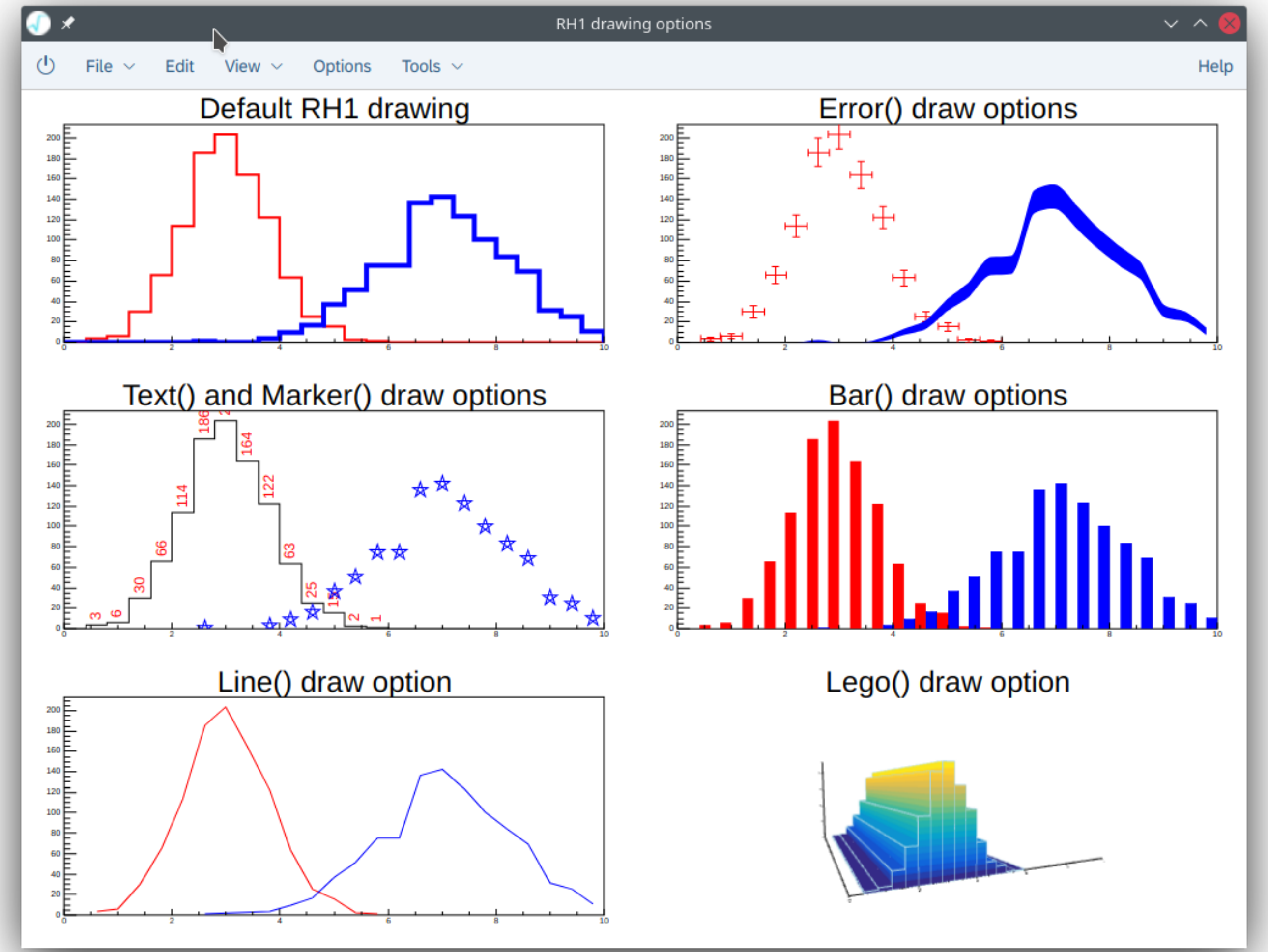
Drawing of RH1

```
// draw first histogram
auto draw1 = canvas->Draw(pHist);
draw1->AttrLine().SetWidth(2).AttrColor().SetAuto();

// draw second histogram
auto draw2 = canvas->Draw(pHist2);
draw2->AttrLine().SetWidth(4).AttrColor().SetAuto();

// assign auto colors
canvas->AssignAutoColors();


// Or directly assign colors
draw1->AttrLine().SetWidth(2).SetColor(RColor::kRed);
draw2->AttrLine().SetWidth(4).SetColor(RColor::kBlue);
```

Drawing of RLegend

```
// draw legend
auto legend = canvas->Draw<RLegend>("Legend title");

// set legend graphical attributes
legend->AttrFill().SetStyle(5).SetColor(RColor::kWhite);
legend->AttrBorder().SetWidth(2).SetColor(RColor::kRed);

// add entries referencing existing drawables
// line/fill/marker attributes will be taken from drawables
legend->AddEntry(draw1, "histo1");
legend->AddEntry(draw2, "histo2");

// add extra entry with fully custom attributes
legend->AddEntry("test").SetAttrLine(RAttrLine().SetColor(RColor::kGreen).SetWidth(5))
                        .SetAttrFill(RAttrFill().SetColor(RColor::kBlue).SetStyle(3004))
                        .SetAttrMarker(RAttrMarker().SetColor(RColor::kRed).SetSize(3).SetStyle(28));
```

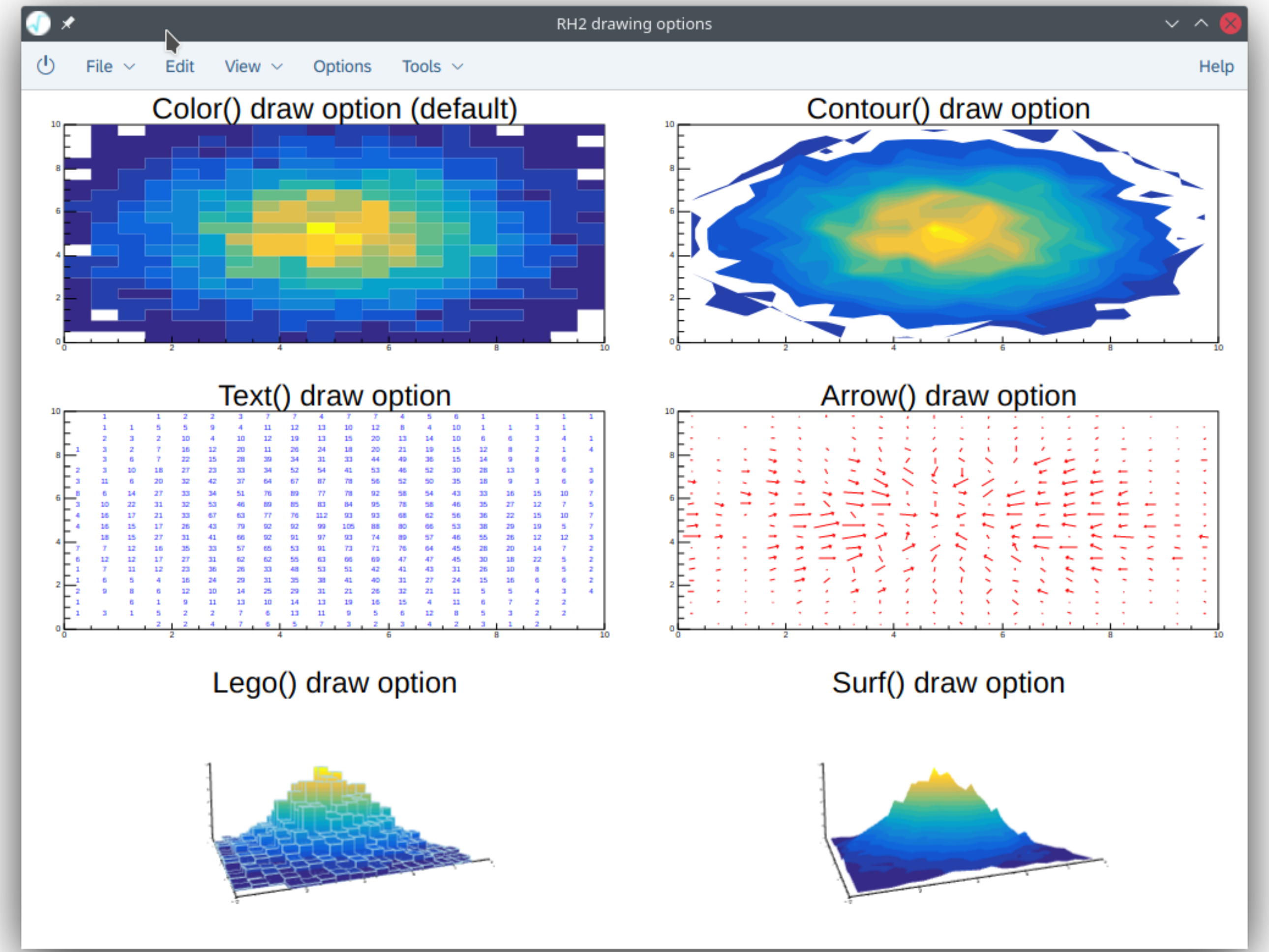
tutorials/v7/draw_rh1.cxx



Different draw options in subpads

```
// Divide canvas on 2x3 sub-pads to show different draw options
auto subpads = canvas->Divide(2,3);

// default draw option
subpads[0][0]->Draw<RFrameTitle>("Default RH1 drawing");
subpads[0][0]->Draw(pHist1)->AttrLine().SetColor(col1).SetWidth(2);
subpads[0][0]->Draw(pHist2)->AttrLine().SetColor(col2).SetWidth(4);

// errors draw options
subpads[1][0]->Draw<RFrameTitle>("Error() draw options");
subpads[1][0]->Draw(pHist1)->Error(1).AttrLine().SetColor(col1);
subpads[1][0]->Draw(pHist2)->Error(4).AttrFill().SetColor(col2).SetStyle(3003);

. . .
```

tutorials/v7/draw_rh2.cxx



RFrame

- Special drawable
  - created automatically when histogram is drawn
- Contains basic frame attributes
  - margin, border, fill, grids
- Plus attributes for X/Y/Z axis
  - fonts, ticks, zoom, log-scale, …

```
// configure RFrame with direct API calls
auto frame = canvas->GetOrCreateFrame();
frame->AttrFill().SetColor(RColor::kBlue);
frame->AttrBorder().SetColor(RColor::kRed);
frame->AttrBorder().SetWidth(3);
frame->Margins().SetTop(0.3_normal);
```

RStyle – CSS for RDrawable

```
// See graf2d/gpagv7/test/rstyle.cxx macro

class CustomDrawable : public RDrawable {
   RAttrLine  fAttrLine{this, "line"};    ///<! line attributes
   RAttrBox   fAttrBox{this, "box"};      ///<! box attributes
   RAttrText  fAttrText{this, "text"};    ///<! text attributes
public:
   CustomDrawable() : RDrawable("custom") {}
   . . .
}


// Configure id and class
CustomDrawable drawable;
drawable.SetId("customid");
drawable.SetCssClass("custom_class");

auto style = RStyle::Parse(file_content);
drawable.UseStyle(style);
```

```
// CSS file
custom {
   line_width: 2;
}

#customid {
   box_fill_style: 5;
}

.custom_class {
   text_size: 3;
}
```

Related tutorials from /tutorials/v7/ dir

draw_axes.cxx	RAxis
draw_frame.cxx	RFrame attributes, using RStyle
draw_legend.cxx	RLegend
draw_pave.cxx	RPave
draw_rh1_large.cxx	Large RH1
draw_rh1.cxx	RH1 draw options
draw_rh2_large.cxx	Large RH2
draw_rh2.cxx	RH2 draw options
draw_rh3_large.cxx	Large RH3
draw_rh3.cxx	RH3 draw options
draw_text.cxx	RText
draw_subpads.cxx	Using sub-sub pads
draw_v6.cxx	Draw TH1/TH2/TGraph inside RCanvas
line.cxx	Batch RLine drawing into PNG file
lineWidth.cxx	RLine, different line widths
lineStyle.cxx	RLine, with different line style
lineRStyle.cxx	Rline, using Rstyle
markerStyle.cxx	RMarker
pad.cxx	simple sup-pads example

Not covered

- Important graphics classes
  - RColor, RAttrColor, RPalette
- RDsiplayItem
  - data displayed on the clients side, generated by RDrawable
- RWebWindow
  - communication between C++ and web-browser
- JavaScript ROOT
  - large code sharing between ROOT6 and ROOT7

Several exercises

1. Create RH1, draw it, add more entries, update canvas

2. Draw RH1 10 times with same draw attributes:

   red line color, line width 5, green fill

3. Draw a histogram with an x-axis with base-2 log scale

Discussion

RAttrLine – aggregation or inheritance?

```
// Example how inheritance approach can look like

class RLine : public Rdrawable, public RAttrLine  {
   RPadPos fP1, fP2;                    ///< line begin/end
~~   RAttrLine fAttrLine{this, "line"}; ///<! line attributes~~
public:
   RLine() : RDrawable("line"), RAttrLine(this, "line")  {}
   RLine(const RPadPos &p1, const RPadPos &p2) : RLine() { fP1 = p1; fP2 = p2; }
   ~~const RAttrLine &GetAttrLine() const { return fAttrLine; }~~
   ~~RLine &SetAttrLine(const RAttrLine &attr)   { fAttrLine = attr; return *this; }~~
   ~~RAttrLine &AttrLine() { return fAttrLine; }~~
   ...
};


// Fictional example, much more like ROOT6

auto line = canvas->Draw<RLine>(RPadPos(.32_normal, 1_normal*num), RPadPos(.8_normal , 1_normal*num));
~~line->AttrLine().SetWidth(i);~~
line->SetLineWidth(i);
```

Inheritance vs aggregation

- Aggregation more flexible
  - allows multiple line attributes in same drawable
    - but no real usecases till now
  - more coding overhead
    - also for users

- Inheritance is much more convenient
  - ROOT6-style method names
    - like SetLineWidth
  - less coding efforts
  - better acceptance

Programming model

- How to assign attributes to drawable
  - line.LineAttr().SetWidth(12)
  - line.SetLineWidth(12)
  - line("line-width", 12)
  - line(.line_width: 12)
  - line("line-width=12“)
  - line << "line-width" << 12

Histogram drawings

- RHist API not finalized
  - should we provide RDrawable for TH1/TH2/TH3 classes
  - special drawable required for optimized drawing of huge histos
  - see RHistDrawable and tutorials/v7/draw_rh2_large.cxx

Backup

I/O Problem

- Missing I/O std::shared_ptr support
  - special handle and workarounds in RCanvas/RDrawable classes
  - difficult to use in user code
  - maintanance

- Inheritance vs aggregation
  - aggregation more flexible
    - allows multiple line attributes in same drawable (no example till now)
    - more coding overhead (also for users)
  - inheritance is much more convenient
    - ROOT6-style method names (aka SetLineWidth)
    - less coding efforts
    - better acceptance