

Init and **Fill** called by framework (step)

If no steps follow:

Event a->**Init**:

get processor 1 (from framework)

Event a->**Fill**:

call user event function of processor 1
optionally store Event a (by framework)





Macro usage demo

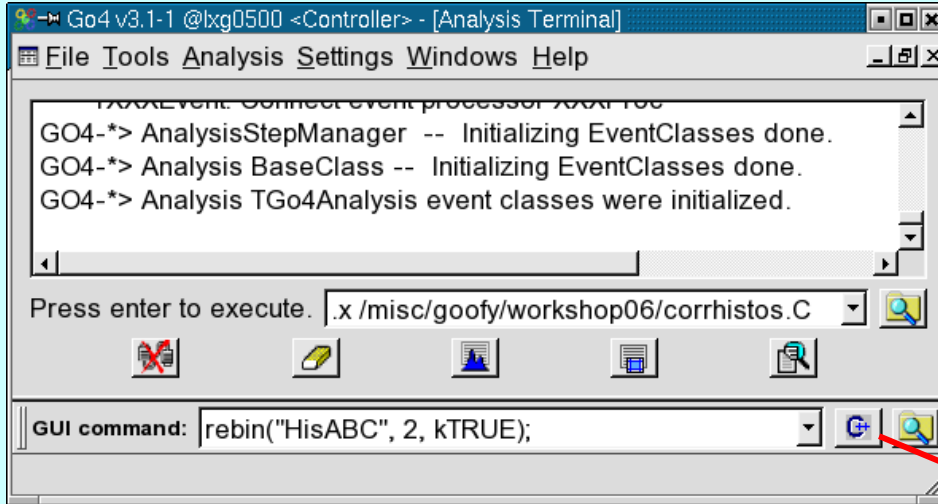




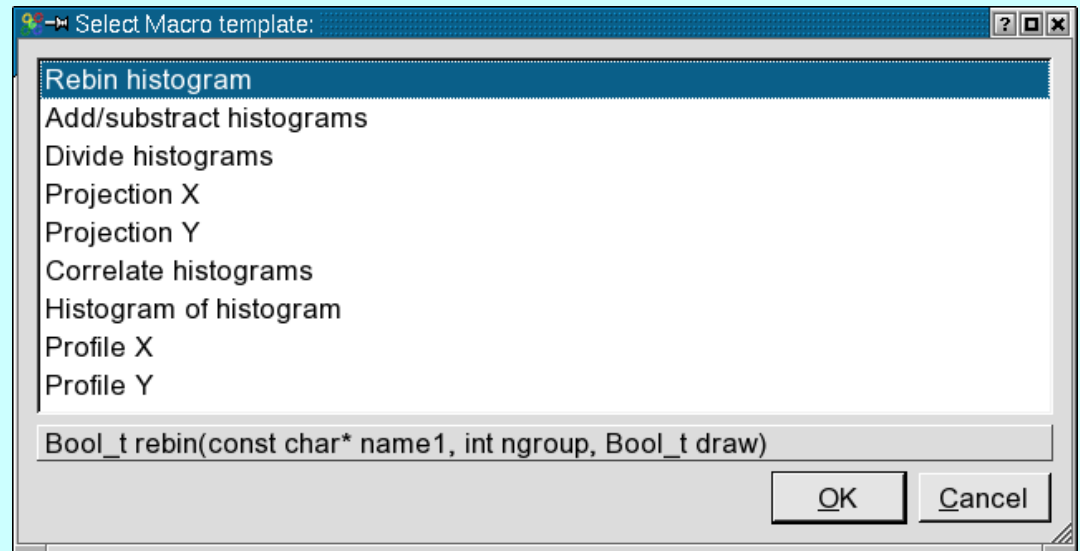
- **Macro execution in analysis code**
 - gROOT->ProcessLine(command)
 - Set up analysis
 - Set parameter values
 - Set condition values
- **Macro execution in analysis called in GUI**
 - Selective clearing of histograms
 - Saving conditions
 - Saving parameters
- **Macro execution in GUI**
 - Histogram calculations
 - Set up monitoring (hot start)
 - Elaborate graphics



Remote (analysis) and local (GUI)



some macros provided by Go4





Full description in Go4 reference manual

- **Macro execution in analysis**
 - `#ifdef __GO4ANAMACRO__ ... #endif`
 - `go4` already set to `TGo4Analysis::Instance()`
 - `TGo4Condition *cond = go4->GetAnalysisCondition(name)`
 - `TH1 *histo = go4->GetHistogram(name)`
 - `TGo4Parameter *par = go4->GetParameter(name)`
 - `TObject *object = go4->NextMatchingObject(wildcard, folder, createlist)`
- **Macro execution in GUI**
 - `#ifdef __GO4MACRO__ ... #endif`
 - `go4` is already set to `TGo4AbstractInterface::Instance()`
 - `TString *fullname = go4->FindItem(name)`
`TObject *object = go4->GetObject(fullname)`
 - `TString *fullname = go4->SaveToMemory(workspace-folder, object, ownership)`
 - `go4->MonitorItem(name, enable)`
 - `ViewPanelHandle *view = go4->StartViewPanel()`
`go4->DrawItem(fullname, view, options)`



- **Called by gROOT->ProcessLine in analysis**
 - setup.C
 - stream.C
- **Called in GUI analysis command line**
 - savecond.C(file,1)
 - savepar.C(name,file)
- **Called in GUI command line**
 - draw1.C(#) #=16,48
 - draw2.C(#,channel) #=16,48 channel=100-500
 - draw3.C(#) #=16,48 dPicture.root(DataPict)

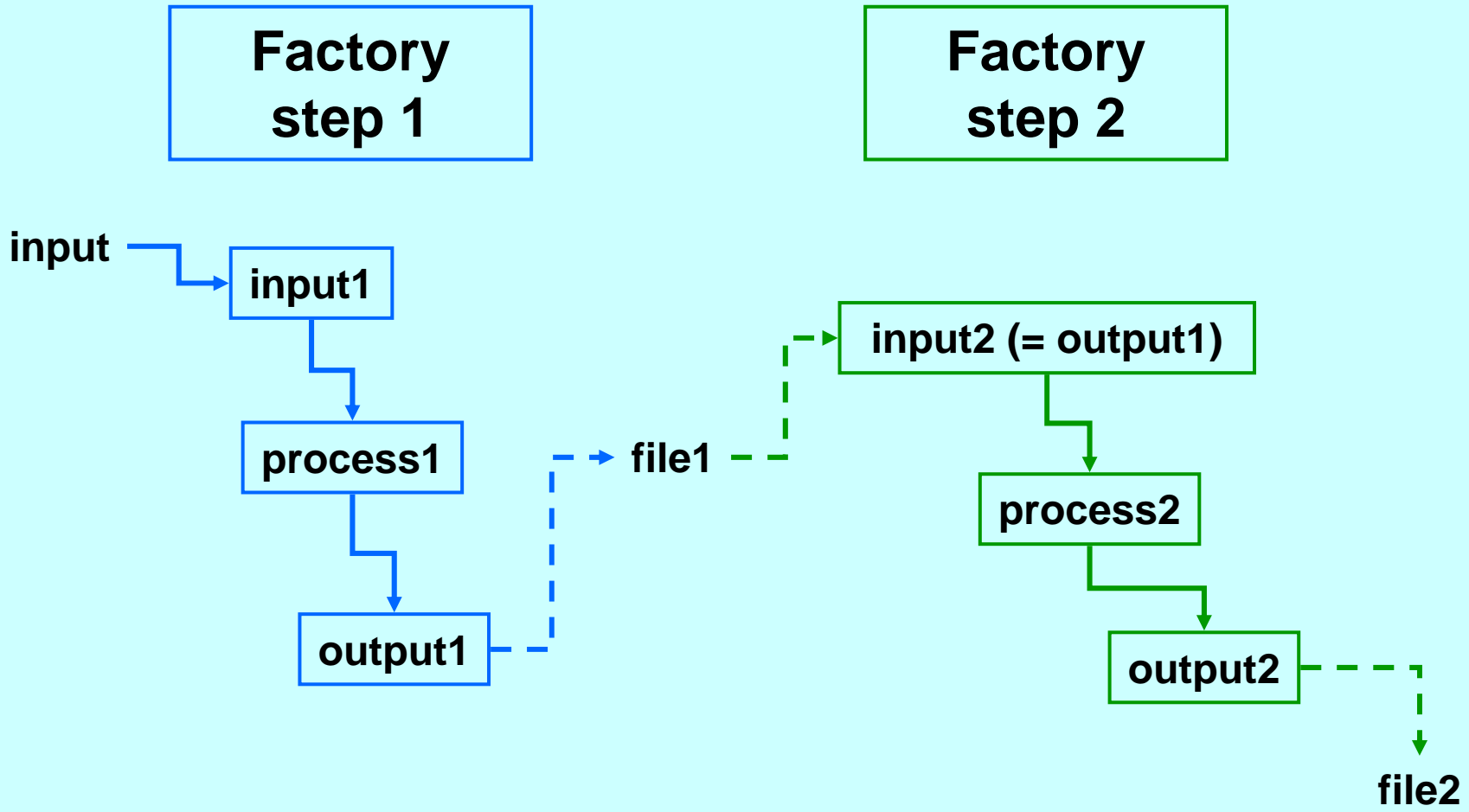
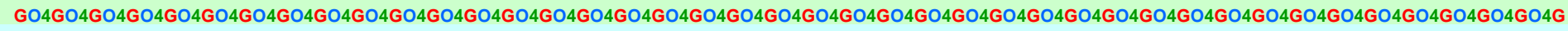


Using several analysis classes in one step



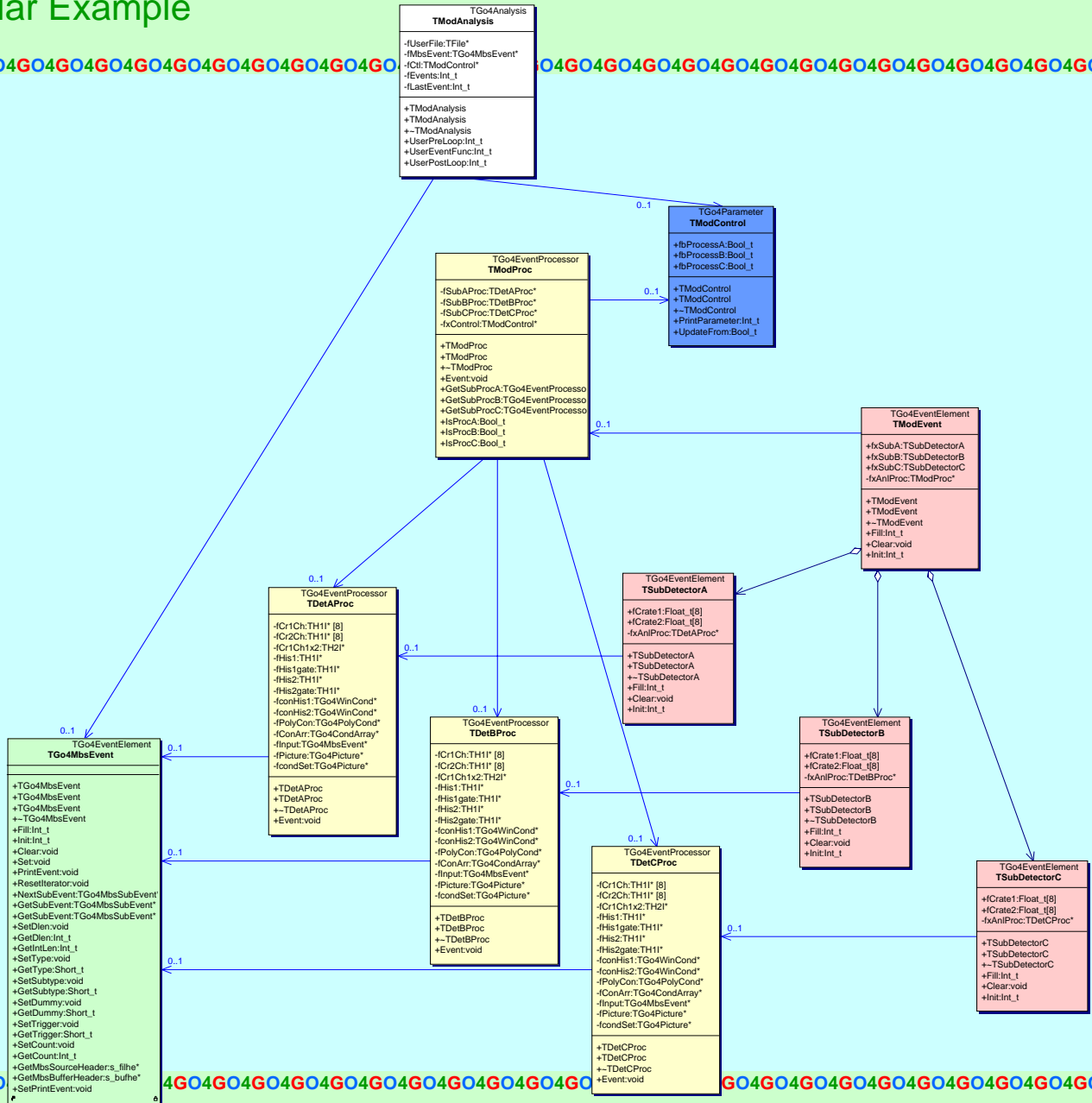


Analysis steps (event loop)





Modular Example





Using browser and graphical editors





- **Browser functionality**
- **Usage as ROOT files browser**
- **Displaying data in view panel**
- **Objects in GUI workspace**
- **Export data from browser**





- One browser for all data sources
(analysis, file, histogram server, memory,...)
- Full control by context menu
- Different objects properties displayed in columns
- Objects monitoring tool
- Objects filter tool by state (monitored, fetched, all)
- Local memory workspace with user subdirectories
- Support of drag-&-drop of items





- Go4 GUI can be used in offline as **ROOT files** browser
- To create example ROOT file:

```
shell$ cd
```

```
shell$ cp $ROOTSYS/tutorials/hsimple.C .
```

```
shell$ root -l hsimple.C
```

```
shell$ go4
```

```
go4$ Open file "hsimple.root"
```

- Subdirectories and TTree branches/leaves can be seen
- Display of any histogram by double click



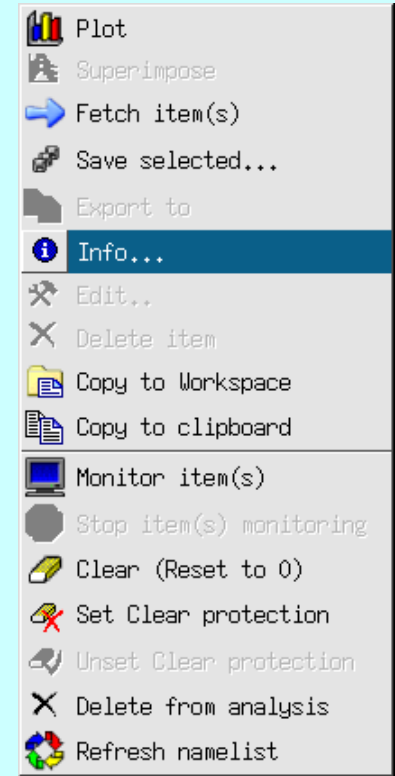
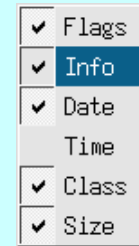
- Open file "hsimple.root"
- Tree viewer activated by double-click on any tree leaf item
- 1-D, 2-D and 3-D tree draw is possible with cut condition
- Drag & drop of leafs names to viewer fields
- Automatic histogram creation or via special dialog

- Same in ROOT session:

```
shell$ root -l  
root [0] TFile::Open("hsimple.root");  
root [1] new TBrowser;  
root [2] "double click" leaf item in TBrowser
```



- Browser displays up to 7 columns:
 - Name** – item name
 - Flags** – item status (monitored, fetched)
 - Info** – object title
 - Date, Time** – time when object is fetched
 - Class** – object class
 - Size** – object size or sum for folders
- To activate/deactivate columns, use right-mouse menu out of the **Name** column
- For each element context menu via right-mouse click on item name is available

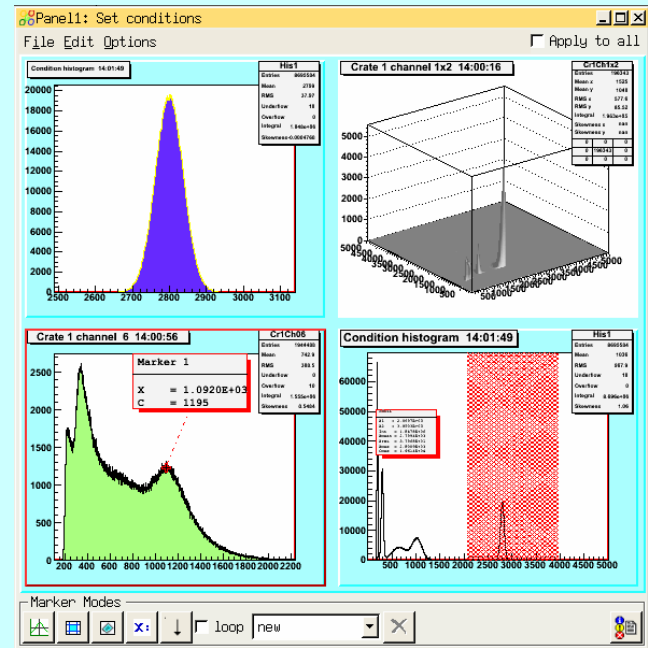




Display of data in view panel



- Display by double click or via Draw in context menu
- Start empty view panel and drag & drop from browser
- View panel features:
 - subpads division
 - superimpose mode
 - different draw / log scale options
 - zooming tools
 - ROOT editor
 - markers editor (described later)
 - save as picture / canvas / image
 - date / time / name info in pads
- Superimpose via context menu in browser
- Usage of “Apply to all” flag
- “Fetch when drawing” flag from setting menu





- Place for fixed (non-updated) objects copy
- Any ROOT-generated objects (like projections) will (should) appear in workspace
- For most objects in browser “Copy to workspace” context menu can be activated
- Copy & paste functionality for any object / folder
- Subfolders can be created / renamed / deleted
- Store of Workspace folders in ROOT file
- “Fetch when copying” flag from setting menu





- Complete content can be store in ROOT file via “Save memory” menu command
- Binary ROOT and XML format are supported
- Any selected item(s) can be stored in ROOT file
- “Fetch when saving” flag from setting menu
- Export of histograms content to ASCII (Excel, Origin) and Radware



- **Condition editor**
- **Marker editor**
- **Parameter editor**
- **Dynamic list editor**





- For demonstration example `$GO4SYS/Go4Example2Step` will be used
- To run example:

```
shell$ . go4login new
shell$ cd $GO4SYS/Go4Example2Step
shell$ go4
go4$ "Launch analysis"
go4$ Configure without Autosave
go4$ Submit & start analysis
```



- TGo4Condition – base class for any condition
- TGo4WindCond – 1-D or 2-D window condition
- TGo4PolyCond – 2-D polygon condition
- TGo4CondArray – array of conditions

- To be used in analysis for testing event data against boundary condition, activating/deactivating analysis code

- Main methods:
 - Test(x), Test(x,y) – check if tested value(s) comply to condition
 - SetHistogram(name) – connect condition with histogram for displaying



Editor for TGo4Condition class

Condition editor

Analysis/Conditions/cHis1 Win 1-D

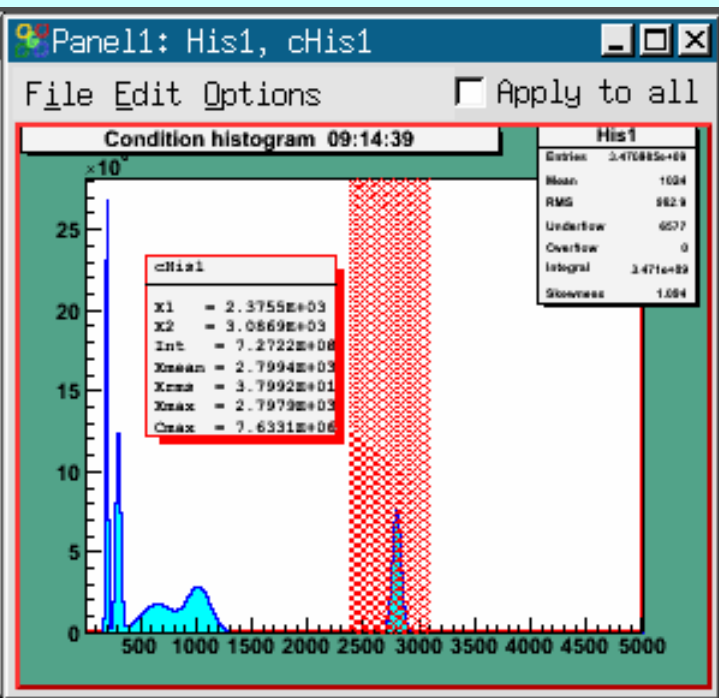
Returns Result Regular

All counts: 478899008 True: 383103967 80.00%

Limits Draw Stats Mean

Xmin: 2375.52 Xmax: 3086.92

Ymin: Ymax:





- Activation: double click on condition, condition context menu, drag & drop condition to editor
- Displaying condition on view panel with draw button or drag & drop to viewpanel
- Changing condition values in editor or in view panel
- Update condition in analysis, store / restore condition in file
- Creating new condition in analysis



- User can add following objects:
 - 1-D, 2-D or polygon marker
 - Text label
 - Arrow
- Loop mode to add many objects in once
- Select / change / delete specific markers
- Can be used as reduced condition editor





Editing of local/remote TGo4Parameter objects

The screenshot shows the Go4 v3.0-0beta parameter editor interface. The main window displays a table of object members for 'TXXXCalibPar'. A context menu is open over the 'fxLinesFinder' entry, showing 'Modify Fitter' and 'Edit...'. The right-hand pane shows a tree view of the workspace hierarchy.

Name	Type	Value	Comments
fdA[0]	Double_t	1.906823	Calibration polynom coeff
fdA[1]	Double_t	0.003414	Calibration polynom coeff
fdA[2]	Double_t	0.000000	Calibration polynom coeff
fdA[3]	Double_t	0.000000	Calibration polynom coeff
fbRecalibrate	Bool_t	1	Set to kTRUE to make calibration fit in upc
fbReadDatabase	Bool_t	0	Set to kTRUE to re-read energies from exter
fxDatabase	TString	calilines.txt	Filename for ascii file with linesname - er
fiLinesChannel[2]	Int_t	650	Centroid channel numbers for fitted lines
ffLinesEnergy[0]	Float_t	1.486708	Database energies of calibration lines
fxLinesNames[0]	TString	AlKa	Database names of calibration lines.
fxLinesFinder	TGo4Fitter*		Fitter to search lines
fxCalibrator	TGo4Fitter*		Fitter for calibration of channel/energies
fxGraphName	TString	Calibration	Name of the graph to contain the calibratic
fxSpectrumName	TString	Cr1Ch01	Name of the calibration spectrum histogram

Workspace Hierarchy:

- Workspace
 - Analysis
 - Histograms
 - Conditions
 - Parameters
 - 123 XXXPar1
 - 123 XXXPar2
 - 123 sizefitter
 - 123 specfitter
 - 123 CaliPar
 - DynamicLists
 - Trees
 - Pictures
 - Canvases
 - EventObjects
 - UserObjects

Status Bar: R3G-2, Current Ev/s: 1495, Average Ev/s: 4, 58893 Events, 2005-10-06 10:51:13



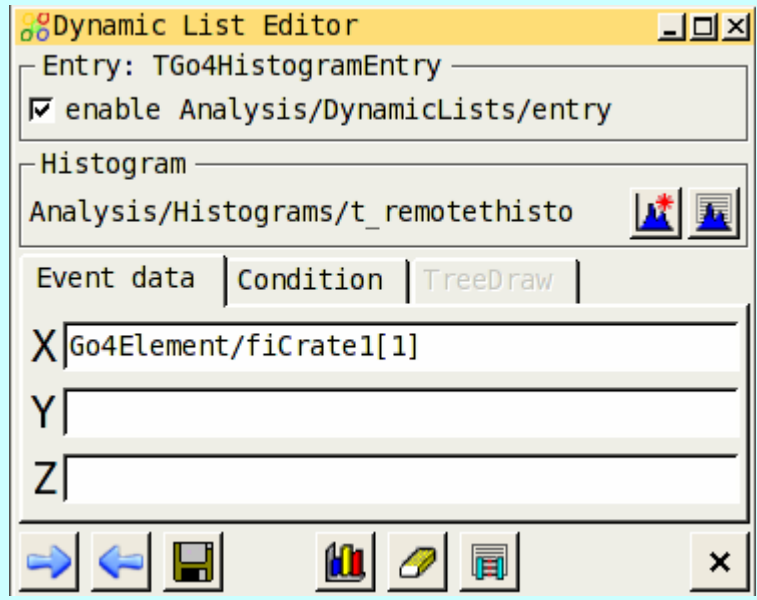
- User classes, derived from TGo4Parameter, should be used as coefficients table
- TGo4Parameter::UpdateFrom() method can be implemented for user control how values will be assigned in the analysis
- For simple cases UpdateFrom() implementation is no longer required



- Basic data types, array of basic data types, TString and TGo4Fitter are supported for editing
- Array contents can be expanded / shrunk
- Comments in class declaration are visible in parameter editor
- Class library **is not required** for parameter editing (only for file I/O)
- Parameter can be updated in analysis or stored / restored in file



For fast histogramming without code changing





- Browser displays analysis event structure and opened tree in analysis
- These data updated once per event and therefore can be used for histogramming
- Two alternatives for dynamic histogramming:
 - over any element(s) in events structures, including condition
 - TTree::Draw() operation



Fitting interactively and in code



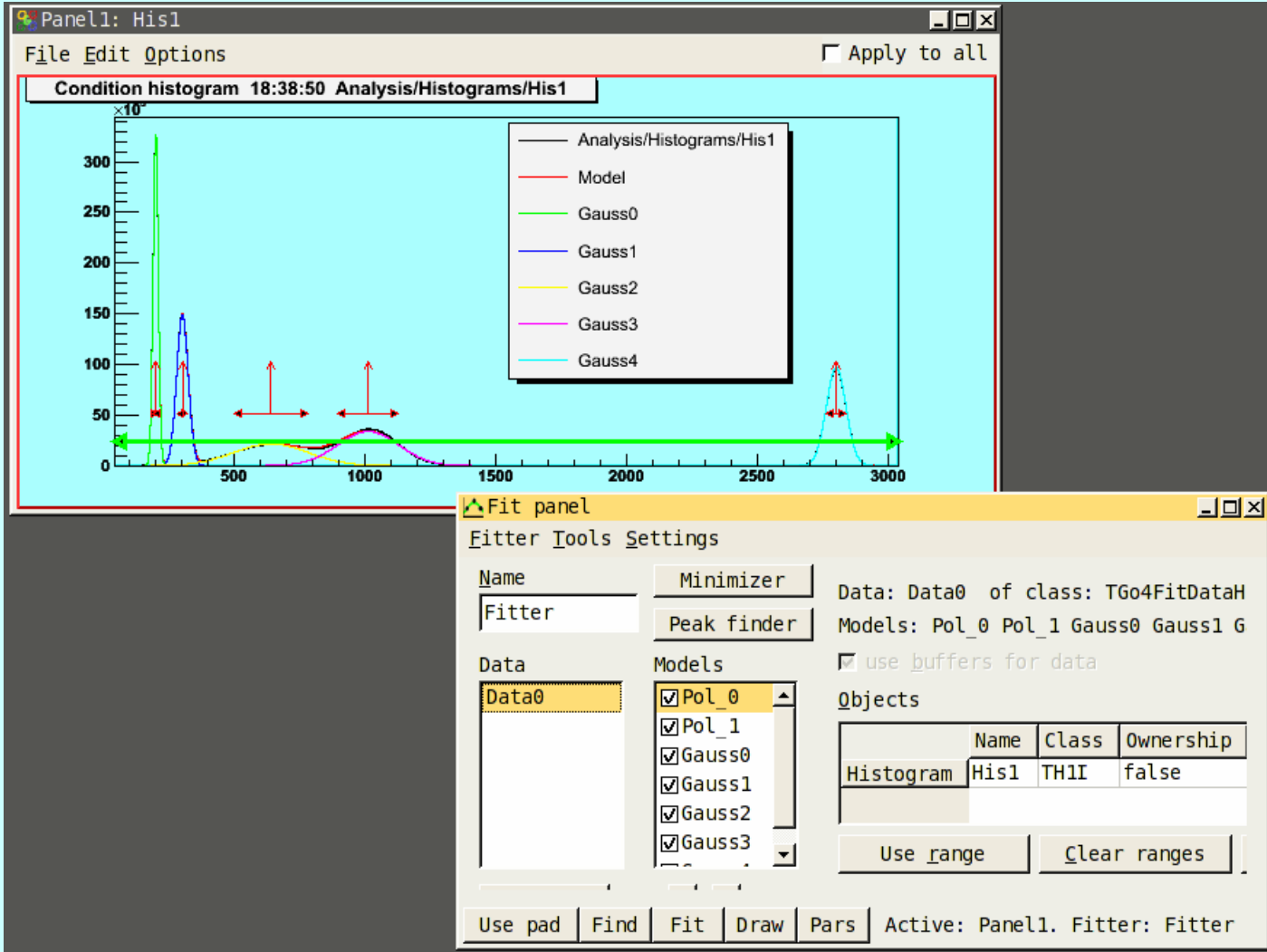


- Fit panel in Go4
- Use of TGo4Fitter in macro
- Reuse of fitter in GUI / Analysis



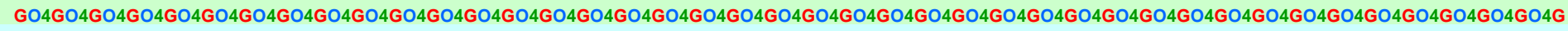


Fit panel





- Fitting of histograms / graphs for any kind of model
- Peak finder
- Manual change of model components / fit parameters
- Fitting and parameters view, parameters output
- Fit panel menu commands
- Different display modes:
 - show only model
 - show model components
 - use different panel for drawing
- Store fitter in file



- Can be used without rest Go4 framework
- Just gSystem->Load(“libGo4Fit.so”) before usage
- \$GO4SYS/Go4FitExample package with many different examples of TGo4Fitter usage
- Can be used as macros or compiled into executable





```
// create fitter, select fit function and add standard actions list
TGo4Fitter fitter("Fitter", TGo4Fitter::ff_ML_Poisson, kTRUE);

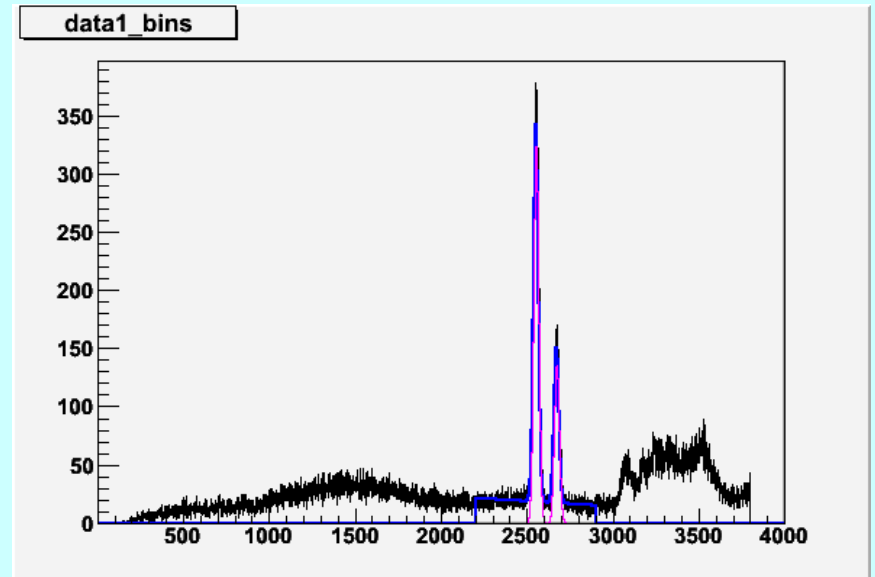
// add histogram to fitter, which should be fitted
fitter.AddH1("data1", GetHistogram("hDeg120_P_c"), kTRUE, 2200., 2900.);

// create polynom of first order
fitter.AddPolynomX("data1", "Pol", 1);

// create two gaussians
fitter.AddGauss1("data1", "Gauss1", 2553., 15.);
fitter.AddGauss1("data1", "Gauss2", 2672., 15.);

// execute all actions
fitter.DoActions();

// draw data, full model and two gaussians
fitter.Draw("#data1,Gauss1,Gauss2");
```

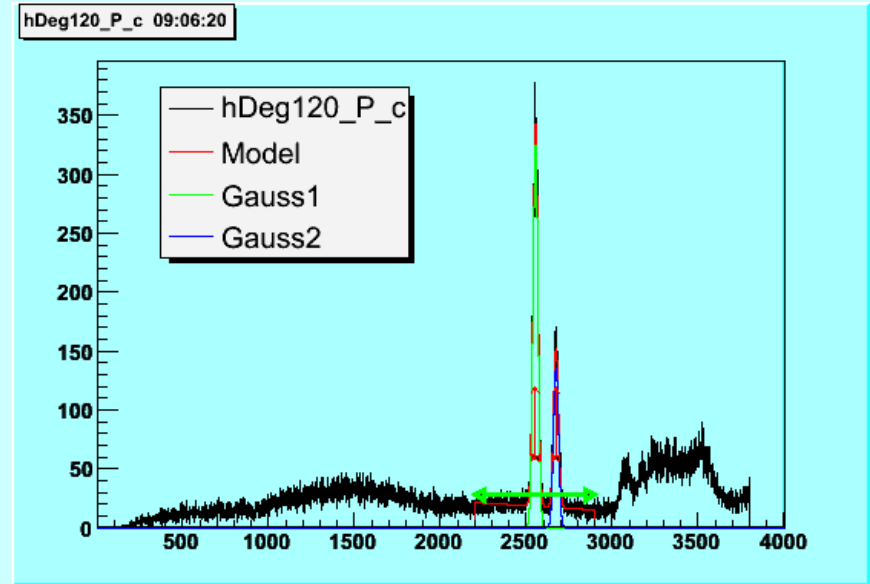




- Store fitter in macro:

```
TFile* f = TFile::Open("fitter.root","recreate");  
fitter.ClearObjects(0, kFALSE); // do not store histos with fitter  
fitter.Write();  
delete f;
```

- Load file with fitter in Go4 browser
- Drop fitter on histogram to be fitted
- Press **Fit** button





- Do fitting in fit panel
- Store fitter in file (via browser)
- Load fitter in macro and assign histograms:

```
// read fitter from file
TFile* f = TFile::Open("fitter2.root");
TGo4Fitter* fitter = (TGo4Fitter*) f->Get("Fitter");
delete f;
if (fitter==0) return;

// set histogram in fitter
fitter->SetObject("data1", GetHistogram("hDeg120_P_c"), kTRUE);

// make fitting
fitter->DoActions();

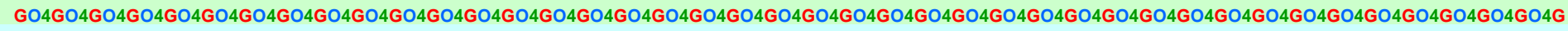
// just draw fitter
fitter->Draw("#data1,Gauss1,Gauss2");
```

- Repeat fitting as many time as necessary



- Simultaneous fit of several histograms
- Fitting model definition via TFormula, sample histogram or shared library
- Different range settings
- Two-dimensional histograms fit
- See Fit tutorial on Go4 webpage

- Most are supported by “Expert” mode in the Fit panel



How to write user GUI





- **User GUI basics**
- **Standard User GUI example**
- **Functionality of QGo4Widget**
- **Getting started with own GUI**



- User GUI is must be placed in shared library
- GO4USERGUI shell variable should specify path and name of shared library
- StartUserPanel() function must be defined
- Top-level widget must be inherited from QGo4Widget class, no extra classes
- Plugin for Qt designer can be generated by:
 make plugin



- In GO4SYS/Go4UserGUI
- Contains one widget, inherited from QGo4Widget
- Shows:
 - drag & drop items from browser
 - draw / edit / info of any item
 - object update/delete notification
 - usage of QtRoot canvas



Via two virtual methods of QGo4Widget:

- `bool IsAcceptDrag(itemname, class, kind)`
 - Here user should decide if item of that class and kind (object, parameter, tree leaf, event and so on) can be accepted or not
- `void DropItem(itemname, cl, kind)`
 - when drag is accepted and user drop item on panel, any action can be done. In example label is updated



- Many useful methods of QGo4Widget:
 - EditItem(const char* itemname)
 - DrawItem(const char* itemname)
 - ShowItemInfo(const char* itemname)
 - SaveItemToFile()
 - UpdateItemInAnalysis()
- Methods of TGo4Interface, TGo4BrowserProxy and TGo4AnalysisProxy also accessible





- Object manager can inform user GUI when object is updated / deleted in browser
- To enable, link between browser item and user widget must be established (typically, when item is dropped):
 AddLink(itemname,"LinkName")
- Virtual methods must be reimplemented:
 QUserPanel::linkedObjectUpdated(linkname, obj)
 QUserPanel::linkedObjectRemoved(linkname)
- In these methods correct update of user panel must be implemented.
- This directly enable monitoring of objects in user panel



- Copy content of \$GO4SYS/Go4UserGUI in new location
- Compile:
 - shell\$ make clean
 - shell\$ make all
- Set GO4USERGUI variable:
 - shell\$ export GO4USERGUI=`pwd`
- Run go4 and activate user GUI panel
- If necessary, install designer plugin (from \$GO4SYS):
 - shell\$ make plugin
- Copy plugin to designed plugins location (see comments after make plugin operation)
- Start Qt designer, open project “Go4UserGui.pro”, modify existing panel, create new tabs, new widgets and so on