

G04 Version 4.4

J.Adamczewski-Musch, H.G.Essel, S.Linev

Lectures

Steps, event and processor objects

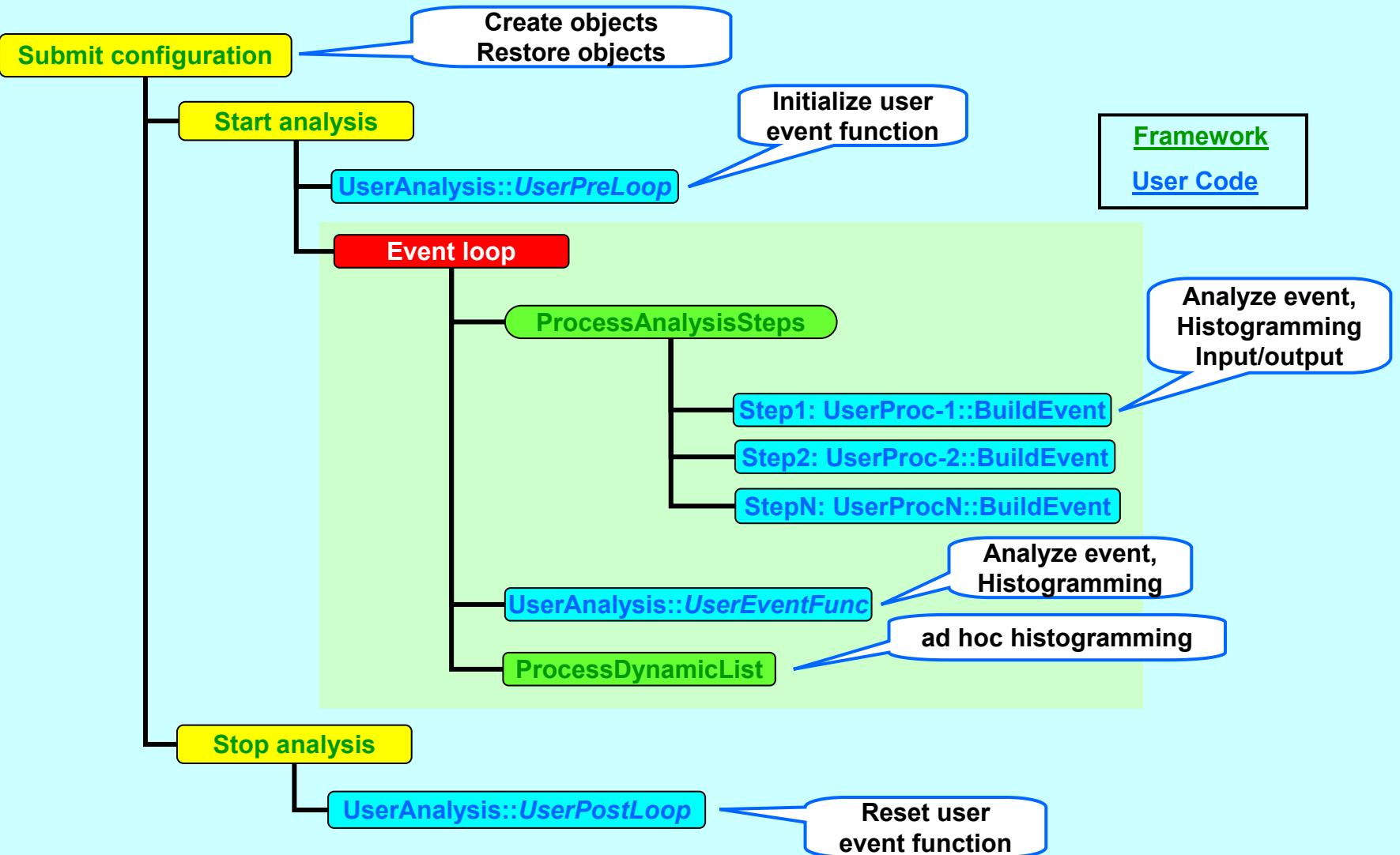
are typically created in the code (`MainUserAnalysis` or `UserAnalysis`)

Now, the steps must be configured:

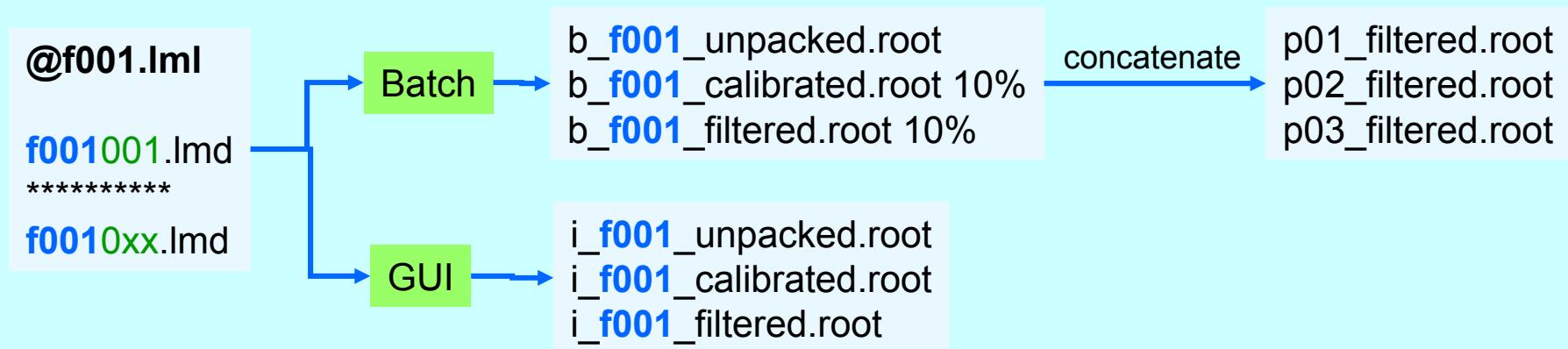
- **Enable/disable steps**
 - **Specify file names for event source and store**
 - **Enable file reading/writing**
 - **Specify overwrite mode**

Autosave filename, interval must be specified.

There are several methods, how this can be done.



Example of naming conventions (TASCA)



Data source and data store directories from shell variables



There are several methods to configure the analysis which can be combined in a defined order:

Constructor of user analysis class or MainUserAnalysis

All steps must be created.

One may use [user arguments](#) given by `go4analysis` command (behind `-x`) or in the **Launch analysis** panel (**Args**).

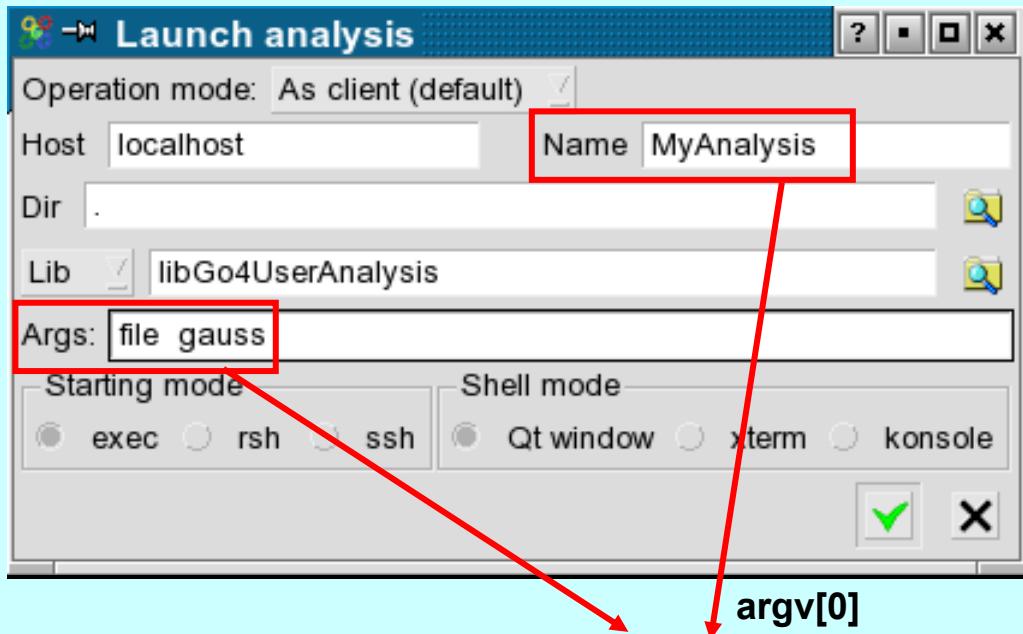
Batch: optional Go4 preferences file

With `--prefs <file>` one can enable loading a preference file. Then the arguments of `go4analysis` (before `-x`) overwrite the settings.

GUI: Go4 preferences file

When launched from GUI settings are loaded from [Go4AnalysisPrefs.root](#). With **hotstart** file the complete setting from that file is used and overwrites the settings.

To print final setup in UserAnalysis::UserPreLoop()
{ Print(); }



TUserAnalysis::TUserAnalysis(int argc, char argv) : TGo4Analysis(argc, argv)**



The screenshot shows the 'Analysis Configuration' dialog box with several tabs at the top: 'Unpacker oxo', 'Calibrator xxx', 'Checker xoo', and 'Analyzer ooo'. The 'Step Control' tab is selected. It contains three checkboxes: 'Enable Step' (checked), 'Source' (checked), and 'Store' (checked). Below these are sections for 'Event source' (set to 'Go4FileSource (1 tree/step) (*.root)') and 'Name' (set to 'i_MyAnalysis_Unpacked'). There are four spinners for event numbers: 0, all, 1, and 1 s. The 'Event store' tab is also visible, showing 'Go4FileStore (1 tree/step) (*.root)' and 'Name: i_MyAnalysis_Calibrated'. It includes spinners for 1, 100 kB, and 3, and a checked 'Overwrite' option. The 'Auto Save File' tab shows 'i_MyAnalysis_AS.root' with an 'Enabled' checkbox (unchecked), a 'once' button, a spinner for '5', and an unchecked 'Overwrite' option. The 'Analysis Configuration File' tab shows 'Go4AnalysisPrefs.root' with a file icon. At the bottom are buttons for 'Submit' (blue arrow), 'Submit+Start' (red arrow), and 'Close' (yellow arrow).



Setup and configure by shell script

With standard go4analysis

```
go4analysis -number <n> -[no]prefs [<name>] -ASF <auto-save file>  
-name <xxx>  
-step UnPack -disable-step  
-step Analysis -source <file>  
-x <user argument list>
```

TUserAnalysis::TUserAnalysis(int argc, char argv) : TGo4Analysis(argc, argv)**
gets user argument list.
argv[0] is name <xxx>

Example1Step uses this name to build output file name
Default for input is coded, but overwritten by command line

User specific MainUserAnalysis programs process their specific argument lists!

Go4 analysis, create step, configure step

MainUserAnalysis or

UserAnalysis constructor: create event, processor, and step objects

Factory is a utility class

```
TGo4StepFactory* cali = new TGo4StepFactory("CaliFact"); // any name
cali->DefEventProcessor("Calibrator","CaliProc"); // object name, class name
cali->DefInputEvent("Unpacked","UnpackEvent"); // object name, class name
cali->DefOutputEvent("Calibrated","CaliEvent"); // object name, class name
TGo4AnalysisStep* calistep = new TGo4AnalysisStep("Calibrator",cali,0,0,0);
AddAnalysisStep(calistep);
```

User setup macro: configure step, get a name as argument

```
// build filenames from argument
```

```
step = go4->GetAnalysisStep("Calibrator");
step->SetProcessEnabled(true/false);
step->SetEventStore(new TGo4FileStoreParameter(filename,...));
step->SetStoreEnabled(true/false);
// default event source is output event from previous step
if(<previous step disabled>) // event must be read from file by Go4
    step->SetEventSource(new TGo4FileSourceParameter(filename));
step->SetSourceEnabled(true);
```

Example2Step uses this mechanism



Macro execution

Macro execution interface showing the Go4 v4.4.0 environment.

The interface includes:

- File**, **Tools**, **Analysis**, **Settings**, **Windows**, **Help** menu.
- Browser** window showing the file structure:
 - Workspace
 - Analysis (selected)
 - Histograms
 - Crate1
 - Crate2
 - Cr1Ch1x2
 - His1
 - His2
 - His1g
 - His2g
 - Sum1
 - Sum2
 - Sum3
 - Eventsize
 - Conditions
 - Parameters
 - DynamicLists
 - Trees
 - Pictures
 - Canvases
 - EventObjects
 - UserObjects
- Analysis Terminal** window displaying histogram statistics:

```
TH1.Print Name = His2g, Entries= 1507970, Total sum= 1.50797e-05
TH1.Print Name = Sum1, Entries= 10700919, Total sum= 1.07009e-05
TH1.Print Name = Sum2, Entries= 15387438, Total sum= 1.53666e-05
TH1.Print Name = Sum3, Entries= 15387438, Total sum= 1.53648e-05
TH1.Print Name = Eventsize, Entries= 2367973, Total sum= 2.36797e-05
++++End Histograms+++++
Total size of all histograms is: 639428 bytes.
**** TXXXXAnalysis: Postloop ****
```
- GUI command**: `rebin("", 2, kTRUE);`
- Select Macro template** dialog box:
 - Rebin histogram
 - Add/subtract histograms
 - Divide histograms
 - Projection X
 - Projection Y
 - Correlate histograms
 - Histogram of histogram

Bool_t rebin(const char* name1, int ngroup, Bool_t draw)
- Analysis Terminal** input field: `@PrintHistograms()`
- Annotations**:
 - A red arrow labeled "drag" points from the "Select Macro template" dialog to the "Analysis Terminal" input field.
 - A red arrow labeled "Go4 function" points to the text "Go4 function" in the "Analysis Terminal" input field.



// If histograms, parameters and conditions are in default folders, use:

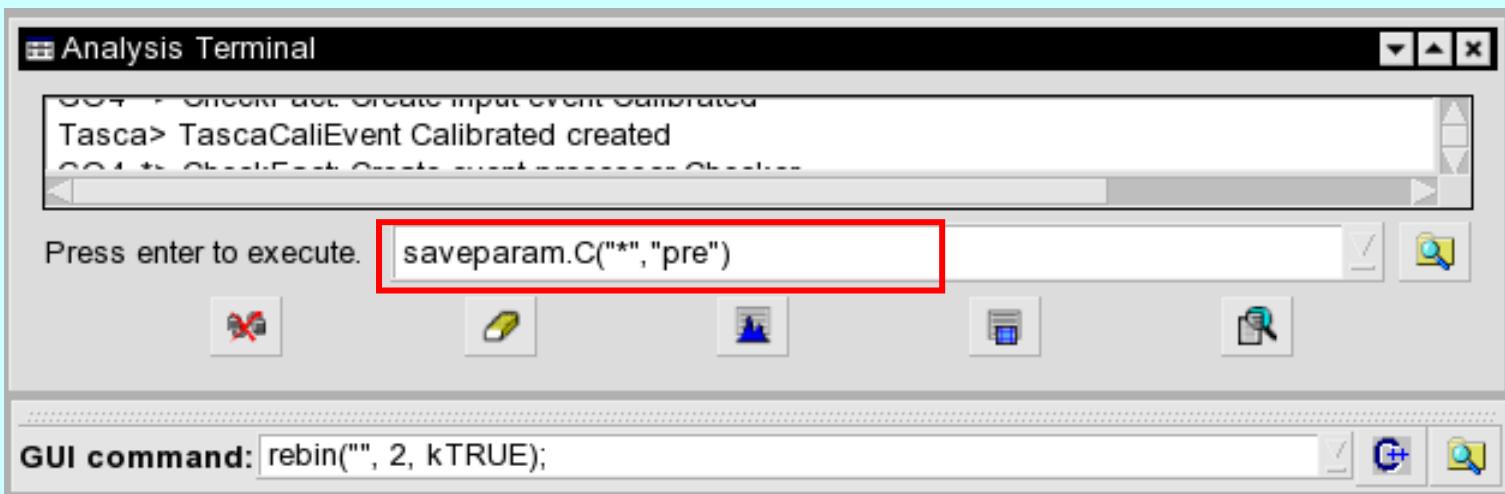
```
MyParam *x = go4->GetParameter("Param1"); // looks in folder "Parameters" and subfolders
```

```
TGo4WinCond *wc = go4->GetCondition("cond1"); // looks in folder "Conditions" and subfolders
```

```
TH1 *s = go4->GetHistogram("histo1"); // looks in folder "H  
// If histograms, parameters and conditions are in user folders, use:
```

```
MyParam *x = qo4->GetObject("Param1"); // looks in all folders and subfolders
```

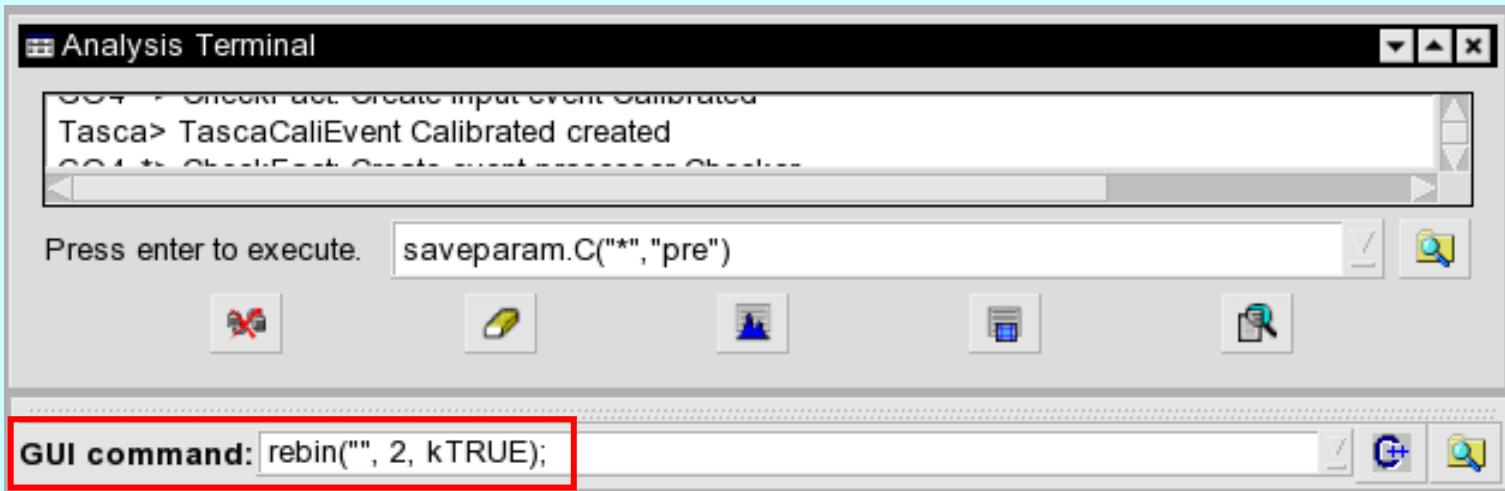
Called in user code (analysis or processors) or from GUI
gROOT->ProcessLine(".x macro.C()");





```
TString fullname1 = go4->FindItem("histo1");
TH1* his=(TH1 *)go4->GetObject(fullname1,1000); // 1000ms timeout to get object from analysis
```

Called in GUI



Jan 2010



```
#ifdef __GO4MACRO__
// macro runs in GUI
TString fullname1 = go4->FindItem("histo1");
TH1* his=(TH1 *)go4->GetObject(fullname1,1000); // 1000ms timeout to get object from analysis
#endif

#ifndef __GO4ANAMACRO__
// macro runs in analysis
MyParam *x = go4->GetObject("Param1"); // looks in all folders and subfolders
#endif

#ifndef __NOGO4MACRO__
// macro runs from ROOT shell
TFile *f = TFile::Open("file.root","r"); // use standard ROOT to locate object in file
#endif
```

Jan 2010