The Go4 Analysis Framework Reference Manual v3.3

J.Adamczewski, M.Al-Turany, D.Bertini, H.G.Essel, S. Linev
31 May 2007

1 Go4 method reference

In the following, the usable methods of the analysis framework classes are described systematically in alphabetical order. The base classes containing each method are listed in the **Classes** field. The methods of TGo4Analysis,e.g. , are available from within the user analysis class, or they may be called externally in the MainUserAnalysis executable (batch mode). The methods of TGo4EventProcessor, e.g., may be called within the user event processor directly. In macros executed in analysis pointer **go4** is a reference to TGo4Analysis::Instance().

AddAnalysisCondition (TGo4Condition con, const Text_t* subfolder):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register a condition to the Go4 framework. The condition will be visible in the Go4GUI after this call, i.e. it can be edited at the GUI view panel. After this call, Go4 adopts the condition, so the user must not delete the condition object in the destructor! Registered conditions are placed in the TFolder Go4/Conditions which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/Conditions may be specified where the histogram shall be located. If condition of the same name already existed in the specified folder, the old condition is deleted and replaced by the new condition con.

Arguments:

TGo4Condition* con	Pointer to conditions that shall be registered. Condition is adopted by Go4.
const Text_t* subfolder	Name of the subfolder of Go4/Conditions where condition shall be placed. If not specified (default), condition will be put directly under Go4/Conditions.

Return value:

Bool_t:	kTRUE if con was registered. kFALSE if argument con was null..

Visibility:

public

Example of usage:

```
TGo4WinCond* cond= new TGo4WinCond("window1","ADC1",200, 800);
AddAnalysisCondition(cond);
```

AddAnalysisStep(TGo4AnalysisStep nextstep):*

Classes: **TGo4Analysis**

Description:

Register analysis step to the Go4 framework. The analysis step object must have been created before. The order of calling this method defines the processing order of the analysis steps, i.e. the first step that was registered will be processed first. **Note that there is currently no way to remove an analysis step from the analysis once it was added!**

Arguments:

TGo4AnalysisStep* nextstep:	Pointer to instance of analysis step class. This object is adopted from the framework after calling this function.

Return value:

Bool_t:	kTRUE if adding nextstep was successful..

Visibility:

public:

Example of usage:

```
TGo4EventSourceParameter* sopar= new TGo4MbsSourceParameter("listmode.lmd");
TGo4EventStoreParameter* stpar= new TGo4FileStoreParemter("out.root");
TGo4AnalysisStep* step1 = new TGo4AnalysisStep("Unpack",sopar, stpar, 0);
AddAnalysisStep(step1);
```

AddCanvas (TCanvas can, const Text_t* subfolder):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register a ROOT TCanvas to the Go4 framework. Go4 adopts the TCanvas, so the user must not delete the canvas object in the destructor! Registered TCanvas are placed in the TFolder Go4/Canvases which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/Canvases may be specified where the TCanvas shall be located. If canvas of the same name already existed in the specified folder, the new canvas will not be registered.

Arguments:

TCanvas* can	Pointer to canvas that shall be registered. Canvas is adopted by Go4.
const Text_t* subfolder	Name of the subfolder of Go4/Canvases where canvas shall be placed. If not specified (default), canvas will be put directly under Go4/Canvases.

Return value:

Bool_t:	kTRUE if can was registered. kFALSE if canvas of same name already existed.

Visibility:

public

Example of usage:

```
TCanvas* c = new TCanvas("positions","x-y-z-n");
c->Divide(2,2);
AddCanvas(c);
```

AddDynamicHistogram(const Text_t name,*
const Text_t histo,*
const Text_t hevx,*
const Text_t hmemx,*
const Text_t hevy,*
const Text_t hmemy,*
const Text_t hevz,*
const Text_t hmemz,*
const Text_t condition,*
const Text_t cevx,*
const Text_t cmemx,*
const Text_t cevy,*
const Text_t cmemy):*

Classes: **TGo4Analysis**

Description:

Add (create) new dynamic histogram entry which connects an existing histogram with existing condition and data. Dynamic entry is specified by name. Histogram histo will be searched in registered histograms folder, condition in conditions folder. If condition is true or not existing (condition=0), histogram will be filled from the values that are found in registered eventstructure classes of names evx, evy, evz at the data members of names memx, memy, memz, for the three coordinate axes, respectively. Histogram dimension is checked against given values. Pointers to the objects and data structures that are specified by name are searched once on analysis initialization. Note that this dynamic entry is processed event-by-event, in contrast to the dynamic treehistogram entry (see *AddTreeHistogram()*), which uses a *TTree::Draw* every dynlistinterval. However, this dynamic histogram entry works without any TTree. Addresses of datamembers are searched from ROOT TClass information, values are casted and filled directly to histogram. Only data from valid events are filled into histogram. Datamembers of the following types are currently supported: Float_t, Double_t, Int_t, Short_t, Char_t, Long_t, UInt_t, UShort_t, UChar_t, ULong_t, Bool_t.

Arguments:

const Text_t* name	Name of the dynamic entry. This name will show up in the dynamic histograms display on the gui..
const Text_t* histo	Name of the registered histogram to be filled. May contain full pathname relative to the histograms folder, separated by slash "/".
const Text_t* hevx	Name of the registered eventstructure object that contains the data to be filled in x coordinate.
const Text_t* hmemx	Name of the datamember of eventstructure class that contains the data to be filled in x coordinate.
const Text_t* hevy	Name of the registered eventstructure object that contains the data to be filled in y coordinate. If zero (default), only one dimensional histogram will be used.
const Text_t* hmemy	Name of the datamember of eventstructure class that contains the data to be filled in y coordinate . If zero (default), only one dimensional histogram will be used.
const Text_t* hevz	Name of the registered eventstructure object that contains the data to be filled in z coordinate. If zero (default), only one/two dimensional histogram will be used.
const Text_t* hmemz	Name of the datamember of eventstructure class that contains the data to be filled in z coordinate . If zero (default), only one/two dimensional histogram will be used.

const Text_t* condition	Name of the registered condition to be tested before histogram can be filled.. May contain full pathname relative to the conditions folder, separated by slash "/". If zero, no condition is tested, histo is always filled.
const Text_t* cevx	Name of the registered eventstructure object that contains the data to be tested in condition as x coordinate.
const Text_t* cmemx	Name of the datamember of eventstructure class that contains the data to be tested in condition as x coordinate.
const Text_t* cevy	Name of the registered eventstructure object that contains the data to be tested in condition as y coordinate. If zero (default), only one dimensional condition will be used.
const Text_t* cmemy	Name of the datamember of eventstructure class that contains the data to be tested in condition as y coordinate . If zero (default), only one dimensional condition will be used.

Return value:

Bool_t:	Is kTRUE if adding new entry was successful. If dynamic entry of same name already existed, new entry is not added and kFALSE is returned.

Visibility:

public

Example of usage:

```
TH1D* histo1= new TH1D("spectrum1","ADC1",0,2000,2000);
AddHistogram(histo1,kTRUE,"Gamma");
TGo4WinCond* cond= new TGo4WinCond("window1","ADC1",200, 800);
AddAnalysisCondition(cond);
AddDynamicHistogram("dynamicentry1",
      "spectrum1", "EuroballRawEvent","fiA1",0,0,0,0,
      "window1","EuroballCalibratedEvent","fdT1Anode1");
// Connect registered histogram and condition
// fill histogram with member fiA1 from rawevent under
// window condition tested against member fiT2 from calievent
```

AddHistogram(TH1 his, const Text_t* subfolder, Bool_t replace):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register a histogram to the Go4 framework. The histogram will be visible in the Go4GUI after this call, i.e. it can be monitored and copied to the GUI view panel. Go4 adopts the histogram, so the user must not delete the histogram object in the destructor. Registered histograms are placed in the TFolder Go4/Histograms which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/Histograms may be specified where the histogram shall be located. Use RemoveHistogram() to unregister and delete object again. **Note1: Histogram names are unique within Go4, i.e. there must not be two histograms with the same name, even in different subfolders! This is important for the TTree::Draw() feature of the dynamic list.**

Arguments:

TH1* his	Pointer to histogram that shall be registered. Histogram is adopted by Go4.
const Text_t* subfolder	Name of the subfolder of Go4/Histograms where histogram shall be placed. If not specified (default), histogram will be put directly under Go4/Histograms.
Bool_t replace	If kTRUE, the new histogram his will replace any histogram of the same name that might be registered before in the same folder, and any historam of that name that might exist in *any* subfolder (unique histogram names) The previous histogram is deleted then. If kFALSE, the new histogram will not be registered and the old histogram remains. Default is kTRUE.

Return value:

Bool_t:	kTRUE if his was registered. Otherwise (histogram of same name already existed), kFALSE is returned.

Visibility:

public

Example of usage:

```
TH1D* histo1= new TH1D("spectrum1","ADC1",0,2000,2000);
AddHistogram(histo1,kTRUE,"Gamma");
```

AddObject(TNamed ob, const Text_t* subfolder, Bool_t replace):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register any named object to the Go4 framework. The object will be visible in the Go4GUI after this call, i.e. it can be monitored and copied to the GUI view panel. Unspecified registered objects are placed in the TFolder Go4/UserObjects which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/UserObjects may be specified where the object shall be located.

Arguments:

TNamed* ob	Pointer to object that shall be registered.
const Text_t* subfolder	Name of the subfolder of Go4/UserObjects where histogram shall be placed. If not specified (default), object will be put directly under Go4/UserObjects.
Bool_t replace	If kTRUE, the new object ob will replace any object of the same name that might be registered before in the same folder. The previous object is deleted. If kFALSE, the new object will not be registered in that case, and the old object remains. Default is kTRUE. Note that this argument is available in TGo4Analysis class only.

Return value:

Bool_t:	kTRUE if ob was registered. Otherwise (object of same name already existed in UserObjects), kFALSE is returned.

Visibility:

public

Example of usage:

```
TF1* func= new TF1("Fit1","gaus(0)*expo(3)",0,2000);
AddObject(func);
```

AddParameter (TGo4Parameter par, const Text_t* subfolder):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register a parameter object to the Go4 framework. The parameter will be visible in the Go4GUI after this call, i.e. it can be edited at the GUI view panel. Go4 adopts the parameter, so the user must not delete the parameter object in the destructor. Registered parameters are placed in the TFolder Go4/Parameters which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/Parameters may be specified where the parameter shall be located. Use *RemoveParameter()* to unregister and delete object again.

Arguments:

TGo4Parameter* par	Pointer to parameter that shall be registered. Parameter is adopted by Go4.
const Text_t* subfolder	Name of the subfolder of Go4/Parameters where parameter shall be placed. If not specified (default), parameter will be put directly under Go4/Parameters.

Return value:

Bool_t:	kTRUE if par was registered. Otherwise (parameter of same name already existed), kFALSE is returned.

Visibility:

public

Example of usage:

```
TGo4Parameter* par= new TUserParameter("userdataset");
AddParameter(par);
```

AddPicture (TGo4Picture pic, const Text_t* subfolder):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Register Go4 picture object to the Go4 framework. The picture will be visible in the Go4GUI after this call, i.e. it can be edited at the GUI view panel. Go4 adopts the picture, so the user must not delete the picture object in the destructor! Registered pictures are placed in the TFolder Go4/Pictures which is also saved in the Go4 autosave-file when the analysis is closed, or when *AutoSave()* is called. Optionally, a subfolder of Go4/Pictures may be specified where the parameter shall be located. Use *RemoveParameter()* to unregister and delete object again.

Arguments:

TGo4Picture* pic	Pointer to picture that shall be registered. Picture is adopted by Go4.
const Text_t* subfolder	Name of the subfolder of Go4/Picturess where picture shall be placed. If not specified (default), picture will be put directly under Go4/Pictures.

Return value:

Bool_t:	kTRUE if pic was registered. Otherwise (picture of same name already existed), kFALSE is returned.

Visibility:

public

Example of usage:

```
TGo4Picture* pic= new TGo4Picture("picture1","tofs");  AddPicture(pic);
```

AddTree(TTree mytree, const Text_t* subfolder)*

Classes: **TGo4Analysis**

Description:

Register a root TTree to the Go4 framework. The tree will be visible in the Go4GUI after this call, i.e. the remote tree-viewing feature can be used. Tree reference is put into folder Go4/Trees. Optionally, a subfolder name may be given where the tree reference shall be located.
Note 1: A registered tree will not be saved by the Go4! This method is meant to browse trees of external analysis frameworks for online monitoring.
Note 2: The trees of the Go4 framework storage classes (TGo4FileStore, TGo4BackStore) are registered automatically, so it is not necessary to apply this method for them.

Arguments:

TTree* mytree	Pointer to external tree to be registered.
const Text_t* subfolder	Name of the subfolder of Go4/Trees where reference shall be placed. If not specified (default), tree will be put directly under Go4/Trees.

Return value:

Bool_t:	kTRUE if tree was registered. Otherwise (tree of same name already existed), kFALSE is returned.

Visibility:

public

Example of usage:

```
TTree* htree= new TTree("h","externaltree");
AddTree(htree);
```

AddTreeHistogram(const Text_t hisname,*
const Text_t treename,*
const Text_t varexpr*
TCut selection):

Classes: **TGo4Analysis**

Description:

Add Dynamic list entry which connects a histogram to a tree. If Histogram of hisname already exists, this histogram will be taken as target for a frequent *TTree::Draw* operation. If not, the histogram will be created on first *TTree::Draw*. Strings varexp and selection are used for applying cuts and variables in the *TTree::Draw* (see ROOT Users Guide). The newly created histogram is registered automatically to the Go4, i.e. it can be monitored and copied to the GUI view panel from the folder Go4/Histograms. The frequency of this Tree::Draw (number of events *n* in between two fills of the histogram) can be adjusted for all tree histograms by method *SetDynlistInterval(Int_t n)*.

Arguments:

const Text_t* hisname	Name of the target histogram where the result of the *TTree::Draw()* is filled. On first execution of tree operation, this histogram is searched under the registered Go4 histograms. If not found, ROOT will create a new histogram of that name with automatic range and binsize selection and fill into this histogram. Go4 will register the newly created histogram then.
const Text_t* treename	Name of the tree that contains the event data for the *TTree::Draw()*. On first execution of tree operation, pointer to this tree is searched under the registered Go4 TTrees. This pointer is newly initialized on every initialization of analysis.
const Text_t* varexp	Expression for the *TTree::Draw()* specifying the leafnames to be processed. See ROOT UsersGuide for all possibilities.
TCut selection	ROOT cut object that contains a condition for the histogram filling as a text string. See ROOT UsersGuide**. Note that this is not a graphical cut object TCutG**!

Return value:

Bool_t:	Is kTRUE if adding new entry was successful. If dynamic entry of same name already existed, new entry is not added and kFALSE is returned.

Visibility:

public

Example of usage:

```
// with predefined histogram spectrum1:
TH1D* histo1= new TH1D("spectrum1","ADC1",0,2000,2000);
AddHistogram(histo1,kTRUE,"Gamma");
AddTreeHistogram("spectrum1","UnpackxTree","fADC[1]",0);
// with automatically created histogram mynewhistogram and cut:
TCut window1("fQtot>560");
AddTreeHistogram("mynewhistogram","CalibratexTree","fQ1",window1);
```

AutoSave():

Classes: **TGo4Analysis**

Description:

Explicitly write all objects in the Go4 folders to the current autosave file. Additionally, the TGo4Calibration objects of all analysis steps are stored into the event stores of each step, if existing. This method can be generally disabled by calling *SetAutoSave(kFALSE).*

Arguments:

none

Return value:

void

Visibility:

public

Example of usage:

```
AutoSave();
```

ClearObjects(const Text_t name):*

Classes: **TGo4Analysis**

Description:

Clear (reset) objects matching the given name. Method looks for folder of name first and clears all objects contained. TH1 contents, e.g. are reset to zero, TGraph points are deleted. For parameters, the *Clear()* method is called and may be user-implemented. Note that objects and contents of complete folders can be protected agains clear by method *ProtectObjects()*.

Arguments:

const Text_t* name	Name of the object to be cleared, or name of object folder. If folder of that name exists in the Go4, all objects under this folder are cleared/ reset at once. If no such folder, object of name is searched in all folders and cleared.

Return value:

Bool_t	Is kTRUE if clear was successful. Otherwise (objects not found, or clear protected), kFALSE is returned.

Visibility:

public

Example of usage:

```
ClearObjects("Histograms");
```

CloseAnalysis():

Classes: **TGo4Analysis**

Description:

Closes the analysis framework. Deletes event objects (event structures, io-classes, eventprocessors) that were initialized before and closes all IO. Event objects are unregistered from Go4 folder structure.. This method is typically called from the user analysis executable in batch mode. In Gui mode, pressing the "Submit" button will execute this function before initializing the new setup. Note that this method is virtual, i.e. the user analysis class may redefine what happens on closing. So it would be possible to close an external analysis framework here. Complementary function is *InitEventClasses()*.

Arguments:

none

Return value:

virtual void

Visibility:

public

Example of usage:

```
TGo4Analysis* myanalysis =new TUserAnalysis();
myanalysis->InitEventClasses();
myanalysis->RunImplicitLoop(2000);
myanalysis->CloseAnalysis();
```

DeleteObjects(const Text_t name):*

Classes: **TGo4Analysis**

Description:

Delete object of name, or all objects in folder name, respectively. Objects are only deleted if delete protection is not set. Usually, objects registered from user code are delete protected by default. Objects created dynamically from gui are not delete protected. . Note that objects and contents of complete folders can be protected agains deletion by method *ProtectObjects()*.

Arguments:

const Text_t* name	Name of the object to be deleted, or name of object folder. If folder of that name exists in the Go4, all objects under this folder are deleted at once. If no such folder, object of name is searched in all folders and deleted.

Return value:

Bool_t	Is Is kTRUE if delete was successful. Otherwise (objects not found, or delete protected), kFALSE is returned.

Visibility:
public

Example of usage:
```
DeleteObjects("Histograms");
```

Export(TObject ob, Go4Export_t filter)*

Classes: **TGo4ExportManager**

Description:
Convert and save object ob into different file formats specified by filter keynumber. Object may be collection of subobjects. Each subobject is written into single file then, conserving the hierarchy in subdirectories. In case of ROOT format, all objects will be in one output file.

Arguments:

TObject* ob	Object to be exported. Currently supported are TH1*, TGraph*, TFolder*, TDirectory*, and TCollection*. If ob is a collection, folder, or directory, it is scanned recursively and all objects contained are exported, if possible. Sub-hierarchy of folders and directories are expressed as subdirectories in output file system.
Go4Export_t	Keynumber for the export filter: Currently available: GO4EX_ASCII : Plain text format for histogram (*.hdat) and TGraph (*.gdat). One line per bin/point, columns for x,y,z,counts; or point,x,y; respectively. GO4EX_RADWARE : Radware/gf3 format. One dimensional TH1 only! GO4EX_ROOT : Write object into one ROOT file. Filename to be specified with method TGo4ExportManager::SetOutFile(char*).

Return value:
void

Visibility:
public

Example of usage:
```
TGo4ExportManager exman("myexportfilter");
TFile* myfile=new TFile("go4autosave.root","READ");
exman.Export(myfile,GO4EX_ASCII); // convert all objects of root file
                                  // into ASCII files
```

GetAnalysisCondition(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Access a registered condition by name.

Arguments:

const Text_t* name	ROOT name of the condition to be retrieved. The subfolder name under which the condition was registered may proceed condition name, separated by a slash "/".

Return value:

TGo4Condition*:	Pointer to condition. NULL if not found.

Visibility:
public

Example of usage:
```
TGo4Conditon* con= GetAnalysisCondition("Gamma/window1");
```

GetAnalysisStep(const Text_t name)*

Classes: **TGo4Analysis**

Description:
Access an analysis step object by name.

Arguments:

const Text_t* name	ROOT name of the analysis step.

Return value:

TGo4AnalyisStep*:	Pointer to analysis step. NULL if not found. From step one can obtain the factory by TGo4StepFactory * fact = (TGo4StepFactory *)step->GetStepFactory(); This is needed in macros to define input and output event and event processor.

Visibility:
public

Example of usage:
```
TGo4AnalysisStep* first= GetAnalysisStep("Unpack");
TGo4StepFactory * fact = (TGo4StepFactory *)first->GetStepFactory();
```

GetCanvas(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Access a registered TCanvas by name.

Arguments:

const Text_t* name	ROOT name of the canvas object to be retrieved. The subfolder name under which the canvas was registered may proceed canvas name, separated by a slash "/"

Return value:

TCanvas*:	Pointer to canvas. NULL if not found.

Visibility:

public

Example of usage:

```
TCanvas* c= GetCanvas("positions");
```

GetEventStructure(const Text_t name):*

Classes: **TGo4Analysis**

Description:

Access to registered eventstructure object by TGo4EventElement name. Input and output events of the analysis steps are registered automatically. Note that name does not indicate the analysis stepname here, but the root name of the event itself.

Arguments:

Text_t* name:	Name of the event object in the folder Go4/Eventobjects/Eventstructure.

Return value:

TGo4EventElement*:	Pointer to event object. NULL if event of name was not found.

Visibility:

public:

Example of usage:

```
TGo4EventElement* eve= GetEventStructure("EuroballRawEvent");
TEbEvent* useve= dynamic_cast<TEbEvent*>(eve);
                 // cast to access special event structure
```

GetHistogram(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Access a registered histogram by name.

Arguments:

const Text_t* name	ROOT name of the histogram to be retrieved. The subfolder name under which the histogram was registered may proceed histogram name, separated by a slash "/"

Return value:

TH1*:	Pointer to histogram. NULL if not found.

Visibility:
public

Example of usage:
```
TH1D* histo1= (TH1D*) GetHistogram("Gamma/spectrum1");
TH2D* histo2= (TH2D*) GetHistogram("ElMap");
```

GetInputEvent(Text_t stepname):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Access to the input event object of a certain analysis step. The analysis step will be identified by the ROOT name of the step object, which is set in the TGo4AnalysisStep constructor.

Arguments:

Text_t* stepname:	Name of the analysis step object that holds the input event.

Return value:

TGo4EventElement*:	Pointer to input event object. NULL if step of stepname is not existing.

Visibility:
public:

Example of usage:
```
TGo4EventElement* raw= GetInputEvent("Unpack");
TGo4MbsEvent* mbseve= dynamic_cast<TGo4MbsEvent*>(raw);
                    // cast to access special event structure
```

GetObject(const Text_t name, const Text_t* topfolder)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Access a registered object by name, starting from the folder of name topfolder. If topfolder is zero (default), object is searched in all Go4 folders. Note that TTrees and event objects are not found by this method, since these objects should not be send to the gui. To retrieve these types, use the special getter methods *GetTree()*, *GetEventStructure()*, etc.

Arguments:

const Text_t* name	ROOT name of the object to be retrieved. The subfolder name under which the object was registered may proceed object name, separated by a slash "/"
const Text_t* topfolder	Name of the folder under which object shall be searched. Useful if path to object is not known completely.

Return value:

TNamed*:	Pointer to object. NULL if not found.

Visibility:

public

Example of usage:

```
TH1D* histo1= (TH1D*) GetObject("Histograms/Gamma/spectrum1");
TH1D* histo2= (TH1D*) GetObject("spectrum2", "Histograms");
TGo4WinCond* con= (TGo4WinCond*) GetObject("Conditions/Window3");
```

GetOutputEvent(Text_t stepname):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Access to the output event object of a certain analysis step. The analysis step will be identified by the ROOT name of the step object, which is set in the TGo4AnalysisStep constructor.

Arguments:

Text_t* stepname:	Name of the analysis step object that holds the input event..

Return value:

TGo4EventElement*:	Pointer to output event object. NULL if step of stepname is not existing.

Visibility:

public:

Example of usage:

```
TGo4EventElement* eve= GetOutputEvent("Unpack");
```

```
TUserEvent* useve= dynamic_cast<TUserEvent*>(eve);
                  // cast to access special event structure
```

GetOutputEvent():

Classes: **TGo4Analysis**

Description:

Fast access to the output event object of the ***most recent analysis step*** that was processed before this call. If called in the *UserEventFunc()*, it yields the resulting event object of the last active step of the analysis cycle.

Arguments:

None

Return value:

TGo4EventElement*:	Pointer to output event object of the last analysis step processed.

Visibility:

public:

Example of usage:

```
TGo4EventElement* eve= GetOutputEvent();
```

GetParameter(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Access a registered parameter by name.

Arguments:

const Text_t* name	ROOT name of the parameter object to be retrieved. The subfolder name under which the parameter was registered may proceed parameter name, separated by a slash "/"

Return value:

TGo4Parameter*:	Pointer to parameter. NULL if not found.

Visibility:

public

Example of usage:

```
TGo4Parameter* par= GetParameter("RunSetup1");
TUserParameter* upar=dynamic_cast<TUserParameter*>(par);
```

GetPicture(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Access a registered picture by name.

Arguments:

const Text_t* name	ROOT name of the picture object to be retrieved. The subfolder name under which the picture was registered may proceed picture name, separated by a slash "/"

Return value:

TGo4Picture*:	Pointer to picture. NULL if not found.

Visibility:
public

Example of usage:
```
TGo4Picture* pic= GetPicture("frame3");
```

GetTree(const Text_t name)*

Classes: **TGo4Analysis**

Description:
Access a registered tree by name.

Arguments:

const Text_t* name	ROOT name of the tree to be retrieved. The subfolder name under which the tree was registered may proceed tree name, separated by a slash "/"

Return value:

TTree*:	Pointer to tree. NULL if not found.

Visibility:
public

Example of usage:
```
TTree* mytree= GetTree("tree2");
```

InitEventClasses():

Classes: **TGo4Analysis**

Description:

Initialization of the analysis framework. Creates event objects (event structures, IO classes, eventprocessors) as defined in the event factories of the analysis steps. Event objects are registered automatically in Go4 folder structure. Previous event objects are deleted and unregistered first. This method is typically called from the user analysis executable in batch mode. In Gui mode, pressing the "Submit" button will execute this function, too. Note that this method is virtual, i.e. the user analysis class may redefine what happens on initialization. So it would be possible to initialize an external analysis framework here. Complementary function is *CloseAnalysis()*, which deletes all existing event objects and closes all IO.

Arguments:

none

Return value:

virtual Bool_t Is kTRUE if init was successful.

Visibility:

public

Example of usage:

```
TGo4Analysis* myanalysis =new TUserAnalysis();
myanalysis->InitEventClasses();
myanalysis->RunImplicitLoop(2000);
```

TGo4Analysis::Instance():

Classes: **available everywhere when linked against libGo4Analysis.so**

Description:

Static Method. Get the pointer to the current analysis framework singleton. This reference can be used to invoke any public method of the class TGo4Analysis from outside the class. This is e.g. meaningful if framework settings shall be changed from within the event processor. Note that a "dummy" analysis framework will be created if this method is called without having created the user analysis object before!

Arguments:

none

Return value:

TGo4Analysis* Pointer to analysis framework singleton.

Visibility:

public

Example of usage:

```
TGo4Analysis::Instance()->SetAutoSaveFile("StageTwo.root");
```

IsAutoSaveOn():

Classes: **TGo4Analysis**

Description:
Check if autosaving is enabled.

Arguments:
none

Return value:

Bool_t	Is kTRUE if autosave file is enabled for read and write.

Visibility:
public

Example of usage:
```
if(IsAutoSaveOn())
      cout <<"Autosaving is enabled" << endl;;
```

IsAutoSaveFileChange():

Classes: **TGo4Analysis**

Description:
Check if analysis runs with automatic autosave filename change. In this case, for each new *.lmd file of an input file list, a new autosave file will be created with a name derived from the current input file name. Set this property with *SetAutoSaveFileChange().*

Arguments:
none

Return value:

Bool_t	Is kTRUE if running in file change mode.

Visibility:
public

Example of usage:
```
if(IsAutoSaveFileChange())
      cout <<"analysis with autosave filechange mode" << endl;
```

IsInitDone():

Classes: **TGo4Analysis**

Description:
Check if initialization of analysis was done correctly.

Arguments:
none

Return value:
Bool_t Is kTRUE if init was successful.

Visibility:
public

Example of usage:
```
TGo4Analysis* myanalysis =new TUserAnalysis();
// .. do setup, init etc.
if(myanalysis->IsInitDone())
      myanalysis->RunImplicitLoop(2000);
else
      cerr <<" Error on init" << endl;
```

IsNewInputFile():

Classes: **TGo4Analysis**

Description:
Check if input file list has changed to another *.lmd file in the list.

Arguments:
none

Return value:
Bool_t Is kTRUE if this event belongs to a new input file.

Visibility:
public

Example of usage:
```
if(IsNewInputFile()){
      ShowEvent("Unpack",false); // print first event
      SetNewInputFile(false); // reset flag
}
```

IsRunning():

Classes: **TGo4Analysis**

Description:

Check if analysis slave is in "running"state. To be used from ROOT macro running in Go4 CintServer environment to check the state set via the start/stop buttons of controlling GUI. Running state may be changed by method *SetRunning(Bool_t)*, too.

Arguments:

none

Return value:

Bool_t	Is kTRUE if analysis slave is running, otherwise kFALSE..

Visibility:

public

Example of usage:

```
while(myanalysis->IsRunning()){
...; //user actions independent of go4 eventloop here...
}
```

*LoadObjects (const Text_t * filename):*

Classes: **TGo4Analysis**

Description:

Load objects from a Go4 autosave file back into the Go4 object manager memory. Usually, this is done automatically on submitting Go4 settings if autosave is enabled. However, one might exchange parameters or conditions on the fly by this method. Note that already exisiting histograms in memory are not overwritten by histograms from file of the same name. This method will change the general autosave filename which is used for next AutoSave()!

Arguments:

const Text_t* filename	Name of autosave file to load. If 0, objects are loaded from most recent autosave filename, or from filename as specified in the Go4 analysis setup.

Return value:

Bool_t:	kTRUE if loading was successful. kFALSE if file could not be opened.

Visibility:

public

Example of usage:

```
LoadObjects("run123.root");
```

*LoadStatus (const Text_t * filename):*

Classes: **TGo4Analysis**

Description:
Load analysis settings (parameters for event factory, filenames) from file and apply them to the analysis. On calling this method, a *CloseAnalysis()* will be executed before setting the new parameters, i.e. all event objects of the previous analysis setup are deleted and all files are closed. Status file can be saved using the *SaveStatus()* method.

Arguments:

const Text_t* filename	Name of preferences file. If no filename is specified, file Go4AnalysisPrefs.root will be used.

Return value:

Bool_t:	kTRUE if loading was successful. kFALSE if file could not be opened.

Visibility:
public

Example of usage:
```
LoadStatus("OnlinePrefs.root");
```

TGo4Log::LogfileEnable (Bool_t on*):*

Classes: **TGo4Log (available everywhere)**

Description:
Switch on or off the logfile output. The global ignore level of go4 as set with TGo4Log::SetIgnoreLevel() defines what kind of messages are logged. Logfile can be opened user-defined by calling TGo4Log::OpenLogfile. By default, go4 will open the logfile "*go4logfile.txt*" in a non-append mode.

Arguments:

Bool_ t on	If kTRUE, all go4 messages output will be written to current logfile. Otherwise, nothing is written to file.

Return value:
void

Visibility:
static public

Example of usage:
```
TGo4Log::LogfileEnable(kTRUE);
```

*TGo4Log ::Message (Int_t prio, const Text_t * text,...):*

Classes: **TGo4Log (available everywhere)**

Description:

Printout message text on the local terminal and/or write it to the logfile. Priority prio defines type of message and action. Messages with priority less than the global ignore level (*TGo4Log::SetIgnoreLevel*) are suppressed in printout and logfile. Supports variable argument list for format strings in the text, just like in C printf() function.
Note: It is recommended to use TGo4Analysis::Message instead of this method, since it provides the same printout and logging functionality, but adds the feature to display the message in the gui log window.

Arguments:

Int_t prio	Priority of message that defines output mode: 0: Debug or go4 kernel information 1: Informational message 2: Warning message 3: Error message
const Text_t* text	Text to be displayed in the message. May contain printf() format strings which define the output of the variable argument list at the end.
...	Variable argument list like in printf(). Any number of arguments of any type should be supported here. Uses internally the va_list structure of <stdarg>.

Return value:

void

Visibility:

static public

Example of usage:

```
TGo4Log::Message(2,"Found event with id %d from input %s", evid, filename);
```

*TGo4Log::OpenLogfile (const Text_t * filename, const Text_t * headercomment, Bool_t appendmode):*

Classes: **TGo4Log (available everywhere)**

Description:

Set name of logfile for Go4 messages. File will be opened with name filename. Optionally, the text of a comment may be specified that appears in the first line of the logfile. If appendmode is kTRUE, the output is added to previous file of same filename, otherwise the old filename is overwritten. Any previous logfile, if existing, will be closed by this method. Whether anything is written to file is controlled by the function TGo4Log::LogfileEnable(). The global ignore level of go4 as set with TGo4Log::SetIgnoreLevel() defines what kind of messages are displayed or logged. Without calling this function, go4 will open by default a logfile "*go4logfile.txt*" in a non-append mode.

Arguments:

const Text_t* filename	Name of logfile. If filename is NULL, go4 will open file *go4log-123456.txt*, with the current pid "123456" as part of the name.
const Text_t* headercomment	Optional text comment to be added at the beginning of the logfile. May contain formatting characters like "\n" or "\t".

Bool_ t appendmode	If kTRUE, all following output will be appended to file of filename if existing. Otherwise, content of previous file is overwritten. Default is non-append.

Return value:
void

Visibility:
static public

Example of usage:
```
TGo4Log::OpenLogfile("EuroballLog.txt","--- Logfile June-2003 JA ---", kTRUE);
```

TGo4Log::OutputEnable (Bool_t on*):*

Classes: **TGo4Log (available everywhere)**

Description:
Switch on or off the message output on terminal. The global ignore level of go4 as set with TGo4Log::SetIgnoreLevel() defines what kind of messages are displayed. By default, output is enabled. This method does not affect the writings to the logfile.

Arguments:

Bool_ t on	If kTRUE, all go4 messages will be printed. Otherwise, nothing is displayed.

Return value:
void

Visibility:
static public

Example of usage:
```
TGo4Log::OutputEnable(kFALSE);
```

TGo4Log::SetIgnoreLevel (Int_t level*):*

Classes: **TGo4Log (available everywhere)**

Description:
Set the Go4 global ignore level that defines what kind of messages issued by TGo4Log::Message() are displayed and logged. Default level is 1, i.e. debugging-like kernel output is supressed.

Arguments:

Int_ t level	Any messages with a priority below "level"are suppressed in printout and in logfile. Priorities are: 0 – debug output;

1 – informational;
2 – warnings;
3 and higher – errors.

Return value:
void

Visibility:
static public

Example of usage:
```
TGo4Log::SetIgnoreLevel(0); // get full debug infos
```

*Message (Int_t prio, const Text_t * text,...):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Send a text message to the gui log window, and/or printout message text on the local terminal. Priority prio defines type of message and action. Supports variable argument list for format strings in the text, just like in C printf() function. If logfile is open and enabled, message is written to logfile.

Arguments:

Int_t prio	Priority of message that defines output mode: 0: Printout on local terminal only if debug level is enabled (with *TGo4Log::SetIgnoreLevel(0)*) <0: Printout on local terminal anyway, independent of ignore level 1: Informational message on gui log window and printout 2: Warning message on gui log window and printout 3: Error message on gui log window and printout
const Text_t* text	Text to be displayed in the message. May contain printf() format strings which define the output of the variable argument list at the end.
...	Variable argument list like in printf(). Any number of arguments of any type should be supported here. Uses internally the va_list structure of <stdarg>.

Return value:
void

Visibility:
public

Example of usage:
```
Message(2,"Found event with id %d from input %s", evid, filename);
```

NewStepProcessor (const Text_t name, TGo4EventProcessorParameter* par):*

Classes: **TGo4Analysis**

Description:

Closes down eventprocessor implementation of analysis step and create new processor for this step as specified by parameter par. May be used to switch processor (algorithm) on the fly without initializing the complete analysis. User has to check eventprocessor parameter class (user subclass) in the event factory of the analysis step to enable switching.

Arguments:

const Text_t* name	Name of the analysis step.
TGo4EventProcessorParameter* par	Eventprocessor parameter that defines the new event processor.

Return value:

Bool_t:	kTRUE if analysis step of name was found, otherwise kFALSE.

Visibility:

public

Example of usage:

```
TGo4EventProcessorParameter* ppar=
      new TUserprocessorParameter("Detector2");
NewStepProcessor("Unpack",ppar);
```

NewStepSource (const Text_t name, TGo4EventSourceParameter* par):*

Classes: **TGo4Analysis**

Description:

Closes down eventsource implementation of analysis step and create new eventsource for this step as specified by parameter par. May be used to switch input on the fly without initializing the complete analysis. One output file e.g. can be filled from several input files in a loop; if one input file ends, the next will be set as source

Arguments:

const Text_t* name	Name of the analysis step.
TGo4EventSourceParameter* par	Eventsource parameter that defines the new event source.

Return value:

Bool_t:	kTRUE if analysis step of name was found, otherwise kFALSE.

Visibility:

public

Example of usage:

```
TGo4FileSourceParameter* fpar=
     new TGo4FileSourceParamter ("run1.lmd");
NewStepSource("Unpack",fpar);
```

NewStepStore (const Text_t name, TGo4EventStoreParameter* par):*

Classes: **TGo4Analysis**

Description:

Closes down eventstore implementation of analysis step and create new eventstore for this step as specified by parameter par. May be used to switch output on the fly without initializing the complete analysis. One input file e.g. can be split into several output files depending on the eventnumber.

Arguments:

const Text_t* name	Name of the analysis step.
TGo4EventStoreParameter* par	Eventstore parameter that defines the new event store.

Return value:

Bool_t:	kTRUE if analysis step of name was found, otherwise kFALSE.

Visibility:

public

Example of usage:

```
TGo4FileStoreParameter* fpar=
      new TGo4FileStoreParamter ("output2.root");
NewStepStore("Unpack",fpar);
```

NextMatchingObject (const Text_t expression, const Text_t* foldername, Bool_t reset):*

Classes: **TGo4Analysis**

Description:

Delivers pointer to next object of the Go4 folder structure with a name matching the expression. If reset is true, the list of matching objects will be created anew by comparing all names with expression. If reset is false, the next object of a previously created matching list is returned. Optionally the search can be limited to a given folder as specified by the foldername.

Arguments:

const Text_t* expression	Regular expression for object names

const Text_t* foldername	Name of the object manager subfolder to scan for the matching objects. All subfolders of this folder are scanned recursively. If foldername is 0, scan is started in Go4 top folder.
Bool_t reset	If true, search list will be created by comparing all object names with expression. Otherwise, this method just returns the next object of an already created list of matches.

Return value:

TObject* :	Pointer to next matching object. If 0, end of matching list is reached, or no matching object was found.

Visibility:

public

Example of usage:

```
TObject* result=0;
Bool_t reset=kTRUE;
while((result=NextMatchingObject("*Ch1*", "Histograms", reset))!=0){
    reset=kFALSE;
    printf("Found result object: %s\n",result->GetName());
}
```

Print(Option_t opt):*

Classes: **TGo4Analysis**

Description:

Prints out the analysis configuration, including autosave file settings and set up of the analysis steps

Arguments:

Option_t* opt	Option string. Not yet in use.

Return value:

void

Visibility:

public

Example of usage:

```
Print();
```

PrintConditions(const Text_t expression):*

Classes: **TGo4Analysis**

Description:
Prints out the counter statistics of all conditions in the Go4/Conditions folder to the local analysis terminal. Subfolder names are displayed (if any). The conditions output (one line per condition) is indented depending on the subfolder level. Condition true counter is displayed as ascii bargraph showing the percentage of total calls of TGo4Condition::Test() being true.

Arguments:

const Text_t* expression	Wildcard regular expression for the condition name. Only objects matching the expression are printed. Default is "*".

Return value:
void

Visibility:
public

Example of usage:
```
PrintConditions("Poly*");
```

PrintDynamicList():

Classes: **TGo4Analysis**

Description:
Prints out the list of dynamic list entries to the analysis output terminal.

Arguments:
none

Return value:
void

Visibility:
public

Example of usage:
```
PrintDynamicList();
```

PrintHistograms(const Text_t expression):*

Classes: **TGo4Analysis**

Description:

Prints out the total content of all histograms in the Go4/Histograms folder to the local analysis terminal. Subfolder names are displayed (if any). The histogram output (one line per histogram) is indented depending on the subfolder level.

Arguments:

const Text_t* expression	Wildcard regular expression for the histogram name. Only objects matching the expression are printed. Default is "*".

Return value:

void

Visibility:

public

Example of usage:

```
PrintHistograms();
```

Process():

Classes: **TGo4Analysis**

Description:

Process one main cycle of Go4 eventloop from a **root macro running in the Go4CintServer** (go4root executable, or started with go4_init.C script). Will first execute any command from gui in the queue, then call MainCycle() to process anaylsis steps, user event function and dynamic list (if existing). **This call is required inside any explicit loop of a macro to process go4 framework analysis action.** GUI event ratemeter is also updated by this method.

Arguments:

none

Return value:

Int_t	Indicates Go4 running state. Returns 0 if analysis is running, otherwise value <0. May be used to react on Stop button from GUI in user macro.

Visibility:

public

Example of usage:

```
TGo4Analysis* go4=TGo4Analysis::Instance();
while(go4->Process()==0){ // run go4 loop until it's stopped
; // may put additional eventwise actions here...
}
```

ProtectObjects(const Text_t name , const Option_t* flags):*

Classes: **TGo4Analysis**

Description:

Change protection properties for object of specified name. If object is a folder, change properties of all objects in this folder recursively. Flags may contain key letters like:
"+C"/"-C" to enable/disable protection against Clear() (histogram zeroing etc).
"+D"/"-D" to enable/disable protection against object deletion.
For example flags="+C-D", "+C+D", "-D-C", "-C". Properties not appearing in flags are not changed.
NOTE: Be careful when changing delete protections for objects that are created in the compiled code! By default, all objects registered from compiled code are delete protected. If protection is disabled, they might be deleted from the Go4 GUI, causing possibly invalid pointers in the analysis code!
Protection flags are conserved if objects are saved to a root file.

Arguments:

const Text_t* name	Name of the object /folder to change the protection bits.
const Option_t* flags	String with specifiers how to change protections. "+C"/"-C" to enable/disable ClearObjects(). "+D"/"-D" to enable/disable DeleteObjects().

Return value:

Bool_t:	True if at least one object of name was found and its protection was changed. False if no object of such name.

Visibility:

public

Example of usage:

```
ProtectObjects("Histograms","+C");
                    //prevent reset for all histograms
```

RemoveAnalysisCondition(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Remove a registered condition from the framework. Condition will be removed from Go4 conditions folder and deleted.

Arguments:

const Text_t* name	ROOT name of the condition to be removed. The subfolder name under which the condition was registered may proceed condition name, separated by a slash "/"

Return value:

Bool_t:	kTRUE if condition was found and removed. Is kFALSE if condition of that name was not found.

Visibility:

public

Example of usage:

```
RemoveAnalysisCondition("Gamma/window1");
```

RemoveCanvas(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Remove a registered TCanvas from the framework. Canvas will be removed from Go4/Canvases folder and deleted.

Arguments:

const Text_t* name	ROOT name of the TCanvas to be removed. The subfolder name under which the canvas was registered may proceed canvas name, separated by a slash "/"

Return value:

Bool_t:	kTRUE if canvas was found and removed, kFALSE if canvas of that name was not found.

Visibility:

public

Example of usage:

```
RemoveCanvas("positions");
```

RemoveDynamicEntry(const Text_t entryname, const Text_t* listname)*

Classes: **TGo4Analysis**

Description:

Remove a dynamic entry of entryname from the dynamic list of listname. Note that objects which are referenced from dynamic entry are not deleted, since they are owned by their special folders.

Arguments:

const Text_t* entryname	ROOT name of the Go4 dynamic entry to be removed.

const Text_t* listname	ROOT name of the dynamic list containing the entry to be removed. If zero (default), entry will be removed from the currently active dynamic list.

Return value:

Bool_t:	kTRUE if entry was found and removed, kFALSE if parameter of that name was not found.

Visibility:

public

Example of usage:

```
RemoveDynamicEntry("Dynamic1");
```

RemoveHistogram(const Text_t name, Bool_t del)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Remove a registered histogram from the framework. Histogram will be removed from Go4 histogram folder. If del is true, histogram is also deleted.

Arguments:

const Text_t* name	ROOT name of the histogram to be removed. The subfolder name under which the histogram was registered may proceed histogram name, separated by a slash "/"
Bool_t del	If kTRUE (default), removed histogram is deleted by Go4. If kFALSE, the histogram will be unregistered only, without calling its destructor. Useful if a histogram was registered that is owned by an external framework. **Note: this argument is available in class TGo4Analysis only**.

Return value:

Bool_t:	kTRUE if histogram was found and removed. Is kFALSE if histogram of that name was not found.

Visibility:

public

Example of usage:

```
RemoveHistogram("Gamma/spectrum1");
```

RemoveObject(const Text_t name, Bool_t del)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Remove a registered object from the UserObjects folder. If del is true, objects is also deleted.

Arguments:

const Text_t* name	ROOT name of the object to be removed. The subfolder name under which the object was registered may proceed object name, separated by a slash "/"
Bool_t del	If kTRUE (default), removed object is deleted by Go4. If kFALSE, the object will be unregistered only, without calling its destructor. Useful if an object was registered that is owned by an external framework. Note: this argument is available in class TGo4Analysis only.

Return value:

Bool_t:	kTRUE if object was found and removed. Is kFALSE if object of that name was not found.

Visibility:

public

Example of usage:

```
RemoveObject("Graph1");
```

RemoveParameter(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Remove a registered parameter from the framework. Parameter will be removed from Go4/Parameters folder and deleted.

Arguments:

const Text_t* name	ROOT name of the parameter to be removed. The subfolder name under which the parameter was registered may proceed parameter name, separated by a slash "/"

Return value:

Bool_t:	kTRUE if parameter was found and removed, kFALSE if parameter of that name was not found.

Visibility:

public

Example of usage:

```
RemoveParameter("Calibration5");
```

RemovePicture(const Text_t name)*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:
Remove a registered picture from the framework. Picture will be removed from Go4/Pictures folder and deleted.

Arguments:

const Text_t* name	ROOT name of the picture to be removed. The subfolder name under which the picture was registered may proceed picture name, separated by a slash "/"

Return value:

Bool_t:	kTRUE if picture was found and removed, kFALSE if picture of that name was not found.

Visibility:
public

Example of usage:
```
RemovePicture("frame3");
```

RemoveTree(TTree tree)*

Classes: **TGo4Analysis**

Description:
Remove a registered tree from the Go4 folder structure. Tree will not be deleted by this operation.

Arguments:

TTree* tree	Pointer to the user tree the reference of which shall be removed.

Return value:

Bool_t:	kTRUE if TTree was found and removed in folder, otherwise kFALSE.

Visibility:
public

Example of usage:

```
TTree * t = new TTree("Mytree","test");
AddTree(t);
// do something with framework
RemoveTree(t);
```

RunImplicitLoop(Int_t times):

Classes: **TGo4Analysis**

Description:

Process the analysis loop (main cycle) for times times. This method is typically called from the user executable in batch mode. Analysis processing consists of analysis steps (event processors), dynamic list entry execution, and calling the *UserEventFunc()* implementation of the user analysis class. At the beginning of the implicit loop, the *UserPreLoop()* method is executed once. After the implicit loop, the *UserPostLoop()* method is executed.

Arguments:

Int_t times	Number of analysis cycles (events) to be processed.

Return value:

Int_t	If negative, the analysis framework was not initialized and implicit loop did not run. Null in case of correct processing. All other error handling is done by exceptions that are caught within this function..

Visibility:

public	To be called from outside analysis class only!

Example of usage:

```
TGo4Analysis* myanalysis =new TUserAnalysis();
myanalysis->InitEventClasses();
myanalysis->RunImplicitLoop(2000);
```

*SaveStatus (const Text_t * filename):*

Classes: **TGo4Analysis**

Description:

Save analysis settings (parameters of analysis steps, ..) to a root file. These settings may be restored using *LoadStatus(const Text_t*)* method.

Arguments:

const Text_t* filename	Name of preferences file. If no filename is specified, file Go4AnalysisPrefs.root will be taken.

Return value:

Bool_t:	kTRUE if saving was successful. kFALSE if status object could not be created, or if file could not be opened.

Visibility:

public

Example of usage:

```
SaveStatus("OnlinePrefs.root");
```

SendObjectToGUI (TNamed ob):*

Classes: **TGo4Analysis, TGo4EventProcessor**

Description:

Send any object from the analysis to the gui via data channel. If gui is not available (batch mode), this call has no effect except for a warning message. This method can be used to send objects only if certain conditions in the analysis are true (event driven monitoring of objects). Objects that arrive on gui side unrequested appear in the local memory folder. However, one can prepare the GUI to deliver an arrived object of a certain name to a user written gui, by method *TGo4GUIRegistry::ReDirectObject()*.

Arguments:

TNamed* ob	Pointer to object to be send..

Return value:

void

Visibility:

public

Example of usage:

```
TH1* his=GetHistogram("Gamma/spectrum1");
SendObjectToGUI(his);
```

SetAnalysisCondition (const Text_t name, TGo4Condition* con. Bool_t counter):*

Classes: **TGo4Analysis**

Description:

Sets an already registered condition of name to the values of the external condition con. If condition of given name is not existing in the folder, a copy of the external condition con is registered instead of updating the old condition.

Arguments:

const Text_t* name	Name of the registered condition under folder Go4/Conditions. Optionally, the string name may contain a subfolder name that proceeds the condition name and is separated by a slash "/". Then condition is searched in that subfolder of Go4/Conditions.
TGo4Condition* con	Pointer to external condition that contains the values for updating the registered condition. Note that this external condition is never adopted by framework! If condition of name does not exist, a new condition of the name will be created, updated from con and registered.
Bool_t counter	If kTRUE (default), the condition counter information is also updated and replaced by the counters of condition con. Otherwise, the counter settings of the condition of name are left as before, only the condition ranges are updated.

Return value:

Bool_t:	kTRUE if con was updated or registered correctly. Otherwise kFALSE is returned.

Visibility:

public

Example of usage:

```
TGo4WinCond* cond= new TGo4WinCond("window1","ADC1",200, 800);
SetAnalysisCondition("window2",cond);
```

SetAutoSave(Bool_t on):

Classes: **TGo4Analysis**

Description:

Enable or disable the functionality of *AutoSave()* completely. If disabled, the autosaving is never done, not even on closing the analysis, or when the user calls AutoSave() directly. Moreover, method *SetAutoSaveFile()* will not create a new autosave file if autosaving was disabled before**. Note that the "autosave now" button of the Go4 GUI will overrule this property, i.e. it will write the autosave file in any case.**

Arguments:

Bool_t on	If kTRUE (default), method AutoSave() will work regularily. If kFALSE, autosaving will be disabled completely, i.e. any call of AutoSave() has no effect..

Return value:

void

Visibility:

public

Example of usage:

```
SetAutoSave(kFALSE);
```

*SetAutoSaveFile (Text_t** filename, Bool_t overwrite, Int_t compression*):*

Classes: **TGo4Analysis**

Description:

Specify the ROOT file to save all registered objects in the Go4 folders on calling *AutoSave()*. File will be opened directly before auto-saving, and will be closed immediately after writing the objects.

Arguments:

const Text_t* filename	Name of autosave file for registered objects. If no filename is specified, file *Go4AnalysisFile.root* will be taken.
Bool_t overwrite	If kTRUE, old autosave file of the same name will be overwritten. If kFALSE (default if not specified), old file of the same name will be used. Note that even in non overwrite mode new objects of the same name will replace old objects within the old root file.
Int_t compression	Specify ROOT TFile compression level for the autosave file. Default (if this parameter is left out) is 5.

Return value:

void

Visibility:

public

Example of usage:

```
SetAutoSaveFile("MyASF.root");
```

*SetAutoSaveFileChange (*Bool_t on*):*

Classes: **TGo4Analysis**

Description:

Switches file for *AutoSave()* whenever the eventsource (TGo4MbsFile) changes the input file. Name of auto save file is derived from the name of the current input file. All histograms are reset to zero entries on changing the input. Useful if TGo4MbsFile is working in wildcard/filelist mode and resulting histograms should be written to different auto save files for each input.

Arguments:

Bool_t on	If kTRUE, autosave file name will be changed for each new input file (TGo4MbsFile in multi-input mode). If kFALSE (default behaviour), the resulting histograms of all input files will be summed up and will be saved in one auto save file of the original name, as defined by SetAutoSaveFile().

Return value:

void

Visibility:

public

Example of usage:

```
SetAutoSaveFileChange(kTRUE);
```

SetAutoSaveInterval (Int_t seconds):

Classes: **TGo4Analysis**

Description:

Specify the time in seconds that shall be processed in between two automatic calls of *AutoSave()*. Use method *SetAutoSaveFile()* to specify the autosave file properties. The default autosave interval is set from static constant fgiAUTOSAVECOUNTS which is currently 500 s.

Arguments:

Int_t val	Number of seconds in between two automatic saves. For seconds=0, frequent autosaving will be disabled and autosavefile will be written at the end of analysis only.

Return value:

void

Visibility:

public

Example of usage:

```
SetAutoSaveInterval(300);
```

SetDynListInterval (Int_t val):

Classes: **TGo4Analysis**

Description:

Specify the number of analysis cycles that shall be processed in between two *TTree::Draw()* executions on the tree histogram entries of the dynamic list. All events that were filled into the tree within this interval are used in the *TTree::Draw()*. The useful value strongly depends on the eventrate. It should be put such that it takes no longer than 10 seconds to update the histograms that shall be filled from the dynamic *TTree::Draw*. Default dynlistinterval is set from static constant fgiDYNLISTINTERVAL which is currently 1000.

Arguments:

Int_t val	Number of events in between two dynamic list processings on treehistogram entries..

Return value:

void

Visibility:

public

Example of usage:

```
SetDynListInterval(200);
```

SetFirstStep (Text_t name):*

Classes: **TGo4Analysis**

Description:

Sets analysis step name as first one to be processed in the chain of steps. First step will read input event from its event source. If event source of new first step is not enabled, *InitEventClasses* will fail; except for the case when the consistency check of the analysis step was disabled by SetStepChecking(kFALSE). If name is nullpointer, set first step in steplist as first active step. This method is for setting up the analysis before initialization only, it should not be invoked on the fly!

Arguments:

Text_t* name	Name of the analysis step to set first..

Return value:

Bool_t	Is kFALSE in case of error (e.g. step not found).

Visibility:

public

Example of usage:

```
//... perform analysis with old setup here
CloseAnalysis() // close all files, delete old eventclasses
TGo4FileSourceParameter* spar2 =
```

```
                          new TGo4FileSourceParameter ("raw.root");
NewStepSource("Calibrate", spar2); // setup input for second step
SetFirstStep("Calibrate");
InitEventClasses();
```

SetLastStep (Text_t name):*

Classes: **TGo4Analysis**

Description:

Sets analysis step name as last one to be processed in the chain of steps. If name is nullpointer, set last step in steplist as last active step. This method is for setting up the analysis before initialization only, it should not invoked on the fly!

Arguments:

Text_t* name	Name of the analysis step to set last.

Return value:

Bool_t	Is kFALSE in case of error (e.g. step not found).

Visibility:

public

Example of usage:

```
//... perform analysis with old setup here
CloseAnalysis() // close all files, delete old eventclasses
SetLastStep("Unpack");
InitEventClasses();
```

SetParameter (const Text_t name, TGo4Parameter* par):*

Classes: **TGo4Analysis**

Description:

Sets an already registered parameter of name to the values of the external parameter par. **Note that virtual method TGo4Parameter::UpdateFrom(TGo4Parameter*) must be implemented correctly in user parameter subclasses to let this work!** If parameter of given name is not existing in the folder, a copy of the external parameter par is registered instead of updating the old parameter.

Arguments:

const Text_t* name	Name of the registered parameter under folder Go4/Parameters. Optionally, the string name may contain a subfolder name that proceeds the parameter name and is separated by a slash "/". Then parameter is searched in that subfolder of Go4/Parameters.
TGo4Parameter* par	Pointer to external parameter that contains the values for updating the registered parameter. Note that this external parameter is never adopted by framework! If parameter of name does not exist, a new parameter of the name will be created, updated from con and registered.

Return value:

Bool_t: kTRUE if par was registered and set correctly.

Visibility:

public

Example of usage:

```
TUserParameter* upara= new TUserParameter("ADC1Cal",0.42,132);
SetParameter("ADC2Cal",upara);
```

SetPicture (const Text_t name, TGo4Picture* pic):*

Classes: **TGo4Analysis**

Description:

Sets an already registered picture of name to the values of the external picture pic. If picture of given name is not existing in the folder, a copy of the external picture par is registered instead of updating the old picture.

Arguments:

const Text_t* name	Name of the registered picture under folder Go4/Pictures. Optionally, the string name may contain a subfolder name that proceeds the picture name and is separated by a slash "/". Then picture is searched in that subfolder of Go4/Pictures.
TGo4Picture* pic	Pointer to external picture that contains the values for updating the registered picture. Note that this external picture is never adopted by framework! If picture of name does not exist, a new picture of the name will be created, updated from pic and registered.

Return value:

Bool_t: kTRUE if pic was registered and set correctly.

Visibility:

public

Example of usage:

```
TGo4Picture* mypic= new TGo4Picture("frame5","E1-E4");
SetPicture("frame3",mypic);
```

SetRunning(Bool_t on):

Classes: **TGo4Analysis**

Description:

Change the "running" state of the analysis slave as controlled from the GUI. To be used from ROOT macro in a Go4 CintServer environment to change the run status. Useful to react on analysis conditions checked in the running macro. **Note that from compiled code, it is rather recommended to throw a TGo4UserException with a message and error priority to halt analysis processing.**

Arguments:

Bool_t	If kTRUE, start processing; otherwise stop it.

Return value:

void

Visibility:

public

Example of usage:

```
if(!wincon->Test(mylimit)) SetRunning(kFALSE);
```

SetStepChecking (Bool_t on):

Classes: **TGo4Analysis**

Description:

Enables/Disables the consistency check of the analysis steps. By default, the input event of each step is checked if it matches the output event of the previous step with respect to class name and object name. Furthermore, the first active step must have an eventsource exisiting, and it is not possible to deactivate the processing of steps in between two active steps. This behaviour can be changed for analyses with more complex step structure (i.e. subsequent steps may not depend on the very previous step, but get input from other previous steps, or they may perform logically parallel parts of the analysis). In this case, the checks for input consistency depend on the user code.

Arguments:

Bool_t on	If kTRUE (default), steps consistency is checked. If kFALSE, steps are processed without checks from the framework.

Return value:

void

Visibility:

public

Example of usage:

```
SetStepChecking(kFALSE);
```

SetStepStorage (Text_t name, Bool_t on):*

Classes: **TGo4Analysis**

Description:

Switches the eventstore output of analysis step "name" on or off. Step must have an implemented eventstore instance

Arguments:

Text_t* name	Name of the analysis step.
Bool_t on	If kTRUE (default), storage will be switched on. If kFALSE, eventstore is disabled.

Return value:

Bool_t	Is kFALSE if step of name was not found.

Visibility:

public

Example of usage:

```
// switch storage depending on user condition. However, it is faster
// to set/unset the "IsValid" flag of the user event to suppress
// the writing of wrong events!!!
if(mycondition->Test(value))
      SetStepStorage("Calibrate",kTRUE);
else
      SetStepStorage("Calibrate",kFALSE);
```

ShowEvent(const Text_t name, Bool_t isoutput):*

Classes: **TGo4Analysis**

Description:

Prints out the contents of the current event structure to the local analysis terminal, using the TTree::Show() mechanism of ROOT. Event may be input or output event of any registered analysis step, or any registered TGo4EventElement.

Arguments:

const Text_t* name	Name of the analysis step containing the event. If no step of that name exists, an event of that name is searched directly in the Go4/EventObjects/EventStructures folder.
Bool_t isoutput	If kTRUE (default), the output event of the analysis step is displayed. Otherwise, the input event.

Return value:

void

Visibility:

public

Example of usage:

```
ShowEvent("Calibrate"); // output event of Calibrate step
ShowEvent("Unpack",kFALSE); // input event of Unpack step
```

StoreCondition(const Text_t stepname, TGo4Condition* con):*

Classes: **TGo4Analysis**

Description:

Saves condition con into the eventstore of analysis step stepname. Eventstore must have the virtual *Store(TGo4Condition*)* method implemented for this purpose. Current eventnumber will be added to the object name in the file (in case of TGo4FileStore). Useful for logging analysis parameters in the same file with the event output.

Arguments:

const Text_t* stepname	Name of the analysis step the eventstore of which is used for saving the condition.
TGo4Condition* con	Pointer to condition that shall be saved.

Return value:

Int_t result	-1= eventstore not active, +1= step not found, 0=OK.

Visibility:

public

Example of usage:

```
StoreCondition("Unpack",GetAnalysisCondition("wincon1"));
```

StoreFitter(const Text_t stepname, TGo4Fitter* fit):*

Classes: **TGo4Analysis**

Description:

Saves fitter fit into the eventstore of analysis step stepname. Eventstore must have the virtual *Store(TGo4Fitter*)* method implemented for this purpose. Current eventnumber will be added to the object name in the file (in case of TGo4FileStore). Useful for logging analysis parameters in the same file with the event output.

Arguments:

const Text_t* stepname	Name of the analysis step the eventstore of which is used for saving the fitter.
TGo4Fitter* fit	Pointer to fitter that shall be saved.

Return value:

Int_t result	-1= eventstore not active, +1= step not found, 0=OK.

Visibility:
public

Example of usage:
```
StoreFitter("Unpack",fxCalibrationfitter);
```

StoreFolder(const Text_t stepname, TFolder* fold):*

Classes: **TGo4Analysis**

Description:
Saves folder fold and all contained objects into the eventstore of analysis step stepname. Eventstore must have the virtual *Store(TFolder*)* method implemented for this purpose. Current eventnumber will be added to the object name in the file (in case of TGo4FileStore). Useful for logging analysis parameters in the same file with the event output.

Arguments:

const Text_t* stepname	Name of the analysis step the eventstore of which is used for saving the folder.
TFolder* fold	Pointer to folder that shall be saved.

Return value:

Int_t result	-1= eventstore not active, +1= step not found, 0=OK.

Visibility:
public

Example of usage:
```
StoreFolder("Analysis",GetObjectFolder());
```

StoreFolder(const Text_t stepname, const Text_t* foldername):*

Classes: **TGo4Analysis**

Description:
Saves registered go4 folder of name foldername and all contained objects into the eventstore of analysis step stepname. Eventstore must have the virtual *Store(TFolder*)* method implemented for this purpose. Current eventnumber will be added to the object name in the file (in case of TGo4FileStore). Useful for logging analysis parameters in the same file with the event output.

Arguments:

const Text_t* stepname	Name of the analysis step the eventstore of which is used for saving the folder.
const Text_t* foldername	Name of registered folder that shall be saved.

Return value:

Int_t result	-1= eventstore not active, +1= step not found, 0=OK.

Visibility:
public

Example of usage:
```
StoreFolder("Analysis","MyAnalysisParameters");
```

StoreParameter(const Text_t stepname, TGo4Parameter* par):*

Classes: **TGo4Analysis**

Description:
Saves parameter par into the eventstore of analysis step stepname. Eventstore must have the virtual *Store(TGo4Parameter*)* method implemented for this purpose. Current eventnumber will be added to the object name in the file (in case of TGo4FileStore). Useful for logging analysis parameters in the same file with the event output.

Arguments:

const Text_t* stepname	Name of the analysis step the eventstore of which is used for saving the parameter.
TGo4Parameter* par	Pointer to parameter that shall be saved.

Return value:

Int_t result	-1= eventstore not active, +1= step not found, 0=OK.

Visibility:
public

Example of usage:
```
StoreParamter("Analysis",GetParameter("Calibration"));
```

UserEventFunc():

Classes: **TGo4Analysis**

Description:

Virtual method to be overwritten in the user analysis subclass. **This function is called in the main analysis cycle from the framework, it should not be invoked from the user code!** Method is executed after each cycle of the analysis processing, i.e. after eventprocessor actions and dynamic list operations.

Arguments:

none

Return value:

virtual Int_t No effect.

Visibility:

public

Example of usage:

```
Int_t TEbAnalysis::UserEventFunc()
{
if(fxCaliEvent && fxCaliEvent->IsValid()) {
      // Ge 1, 2, 3, 4  heavy particle on source side
      Double_t dvalue= fxCaliEvent->fdNoOfCutH1;
      if(dvalue>=1.0 && dvalue<=48.0)
            fxGe1H1->Fill(fxCaliEvent->fdGammaH1);
      }
return 0;
}
```

UserPreLoop():

Classes: **TGo4Analysis**

Description:

Virtual method to be overwritten in the user analysis subclass. **This function is called in the main analysis cycle from the framework, it should not be invoked from the user code!** Method is called once before the analysis main loop, i.e. at the beginning of *RunImplicitLoop()* in batch mode, or on pressing the "Start" button in gui mode, respectively.

Arguments:

none

Return value:

virtual Int_t No effect.

Visibility:

public

Example of usage:

```
Int_t TEbAnalysis::UserPreLoop()
{
fxCaliEvent=dynamic_cast<TEbCalibratedEvent*>
                          (GetOutputEvent("Calibrate") );
return 0;
}
```

UserPostLoop():

Classes: **TGo4Analysis**

Description:

Virtual method to be overwritten in the user analysis subclass. **This function is called in the main analysis cycle from the framework, it should not be invoked from the user code!** Method is called once after the analysis main loop, i.e. at the end of *RunImplicitLoop()* in batch mode, or on pressing the "Stop" button in gui mode, respectively.

Arguments:

none

Return value:

virtual Int_t No effect.

Visibility:

public

Example of usage:

```
Int_t TEbAnalysis::UserPostLoop()
{
fxCaliEvent=0; // reset pointer to event structure
AutoSave(); // save all objects each time analysis stops
return 0;
}
```

WaitForStart():

Classes: **TGo4Analysis**

Description:

To be used from a **ROOT macro running in the Go4CintServer environment** (started by go4root executable, or loaded into regular root by go4_init.C logon script).
Polls until the Go4 is set into the "running" state (by **Start** button on GUI or *SetRunning()* method) with 1 second interval. Root event handling is done within this function.

Arguments:

none

Return value:

Int_t	Number of seconds from begin of wait until "running" is switched true. If negative value is returned, a ROOT interrupt has happened during wait (e.g. Ctrl-C on CINT Canvas)..

Visibility:

public

Example of usage:

```
TGo4Analysis* go4=TGo4Analysis::Instance();
while(1){
Int_t seconds=go4->WaitForStart();
if(seconds<0) break; // root interrupt while waiting
cout <<"Starting work after "<<seconds<<" s of waiting..." << endl;
// some user actions here!
}
```

2 Macro Abstract Interface Class Documentation

2.1 TGo4AbstractInterface Class Reference

`#include <TGo4AbstractInterface.h>`

2.1.1 Public Methods

- virtual **~TGo4AbstractInterface** ()
- TGo4ObjectManager * **OM** () const
- TGo4BrowserProxy * **Browser** () const
- TGo4AnalysisProxy * **Analysis** ()
- virtual void **HotStart** (const char *filename)=0
- virtual void **LoadLibrary** (const char *fname)
- virtual void **OpenFile** (const char *fname)
- virtual TString **FindItem** (const char *objname)
- virtual Bool_t **CopyItem** (const char *itemname)
- virtual Bool_t **DeleteItem** (const char *itemname)
- virtual void **FetchItem** (const char *itemname, Int_t wait_time=2000)
- virtual TObject * **GetObject** (const char *itemname, Int_t updatelevel=1)
- virtual TString **SaveToMemory** (const char *path, TObject *obj, Bool_t ownership=kFALSE)
- virtual Bool_t **SaveToFile** (const char *itemname, const char *filename, const char *filetitle=0)
- virtual Bool_t **ExportToFile** (const char *itemname, const char *dirpath, const char *format, const char *filetitle=0)
- virtual void **ConnectHServer** (const char *servername, Int_t portnumber, const char *basename, const char *userpass, const char *filter)
- virtual void **Wait** (double tm_sec)
- virtual void **Message** (const char *msg)
- virtual void **LaunchAnalysis** ()
- virtual void **LaunchAnalysis** (const char *ClientName, const char *ClientDir, const char *ClientExec, const char *ClientNode, Int_t ShellMode=2, Int_t TermMode=1)
- virtual void **ConnectAnalysis** (const char *ServerNode, Int_t ServerPort, Int_t UserMode, const char *password=0)
- virtual void **WaitAnalysis** (Int_t delay_sec)
- virtual Bool_t **IsAnalysisConnected** ()
- virtual void **DisconnectAnalysis** ()
- virtual void **ShutdownAnalysis** ()
- virtual void **ExecuteLine** (const char *remotecmd)
- virtual void **RequestAnalysisConfig** ()
- virtual void **SubmitAnalysisConfig** ()
- virtual void **StartAnalysis** ()
- virtual void **StopAnalysis** ()
- virtual void **SetAnalysisTerminalMode** (int mode)
- virtual void **SetAnalysisConfigMode** (int mode)
- virtual void **MonitorItem** (const char *itemname, Bool_t on=kTRUE)
- virtual void **StartMonitoring** (Int_t period=10)
- virtual void **StopMonitoring** ()
- virtual void **AnalysisAutoSave** (const char *filename, Int_t interval, Int_t compression, Bool_t enabled, Bool_t overwrite)
- virtual void **AnalysisConfigName** (const char *filename)
- virtual void **ConfigStep** (const char *stepname, Bool_t enableprocess, Bool_t enablesource, Bool_t enablestore)
- virtual void **StepFileSource** (const char *stepname, const char *sourcename, int timeout)

- virtual void **StepMbsFileSource** (const char *stepname, const char *sourcename, int timeout, const char *TagFile, int start, int stop, int interval)
- virtual void **StepMbsStreamSource** (const char *stepname, const char *sourcename, int timeout)
- virtual void **StepMbsTransportSource** (const char *stepname, const char *sourcename, int timeout)
- virtual void **StepMbsEventServerSource** (const char *stepname, const char *sourcename, int timeout)
- virtual void **StepMbsRevServSource** (const char *stepname, const char *sourcename, int timeout, int port)
- virtual void **StepRandomSource** (const char *stepname, const char *sourcename, int timeout)
- virtual void **StepUserSource** (const char *stepname, const char *sourcename, int timeout, int port, const char *expr)
- virtual void **StepFileStore** (const char *stepname, const char *storename, bool overwrite, int bufsize, int splitlevel, int compression)
- virtual void **StepBackStore** (const char *stepname, const char *storename, int bufsize, int splitlevel)
- virtual ViewPanelHandle **StartViewPanel** ()
- virtual ViewPanelHandle **StartViewPanel** (int x, int y, int width, int height, int mode=1, TGo4Picture *pic=0)
- virtual TString **GetViewPanelName** (ViewPanelHandle panel)
- virtual ViewPanelHandle **FindViewPanel** (const char *name)
- virtual Bool_t **SetViewPanelName** (ViewPanelHandle panel, const char *newname)
- virtual ViewPanelHandle **GetActiveViewPanel** ()
- virtual void **RedrawPanel** (ViewPanelHandle panel)
- virtual void **DivideViewPanel** (ViewPanelHandle panel, Int_t numX, Int_t numY)
- virtual void **SelectPad** (ViewPanelHandle panel, Int_t number=0)
- virtual void **SetSuperimpose** (ViewPanelHandle panel, Bool_t on=kTRUE)
- virtual Bool_t **DrawItem** (const char *itemname, ViewPanelHandle panel=0, const char *drawopt=0) virtual
- void **RedrawItem** (const char *itemname)

2.1.2 Static Public Methods

- TGo4AbstractInterface * **Instance** ()
- void **DeleteInstance** ()

2.1.3 Protected Methods

- **TGo4AbstractInterface** ()
- void **Initialize** (TGo4ObjectManager *om, TGo4BrowserProxy *br)
- virtual void **ProcessEvents** (Int_t timeout=-1)=0
- Bool_t **LoadHotStart** (const char *filename)
- Bool_t **IsHotStart** ()
- const char * **NextHotStartCmd** ()
- void **FreeHotStartCmds** ()

2.1.4 Static Protected Attributes

- TGo4AbstractInterface * **fgInstance** = 0

2.1.5 Detailed Description

Generic interface to GUI functionality.

Provide methods like open file, request object from analysis and so on.

Methods of that class can be used in arbitrary ROOT script inside/outside Go4 GUI. To access these methods, one should use "go4" variable, which is exported to CINT global variable space when instance of implementation of TGo4AbstractInterface class is created. One can also access such instance also via **TGo4AbstractInterface::Instance**() static method.

There are two implementations of TGo4AbstractInterface class: TGo4Script - used together with Go4 GUI, created automatically when GUI started TGo4Interface - can be used in non GUI mode, must be created once by user As long as these are two implementation of the same interface, macro, which uses internally only go4 instance, should work similarly in both GUI and non GUI mode.

Typicaly one should use interface, provided by that class, in macro, executed inside GUI. Simple example of such macro: { go4->OpenFile("example.root"); go4->DrawItem("example.root/hist1"); } Here one open file from disk and displays one histogram from that file.

Most of the action with objects are going via object manager, where each object (item) has unique name, which includes full path to that object in objects structures. In our example item name is "example.root/hist1". To locate object of known name in objects structure, one can use **FindItem**() (page 60) method. In that case example will look like: { go4->OpenFile("example.root"); go4->DrawItem(go4->FindItem("hist1")); }

There are several examples of GUI macros, which can be found in $GO4SYS/Go4GUI/scripts directory. Description of each method can be seen futher in this document. Hot start feature of go4 was implemented with usage of that class, therefore one can produce hotstart files in Go4 GUI and take some generated code from it directly. For instance, displaing of complex viewpanels or starting and configuring of analysis.

2.1.6 Constructor & Destructor Documentation

2.1.6.1 TGo4AbstractInterface::TGo4AbstractInterface () `[protected]`

Constructor

2.1.6.2 TGo4AbstractInterface::~TGo4AbstractInterface () `[virtual]`

destructor

2.1.6.3

2.1.7 Member Function Documentation

2.1.7.1 TGo4AnalysisProxy * TGo4AbstractInterface::Analysis ()

Returns pointer on analysis proxy. TGo4AnalysisProxy class provide access to analysis controlling instance in program. Noramlly should not be used in GUI script.

2.1.7.2 virtual void TGo4AbstractInterface::AnalysisAutoSave (const char * *filename*, Int_t *interval*, Int_t *compression*, Bool_t *enabled*, Bool_t *overwrite*) `[inline, virtual]`

Configure autosave properties of analysis

2.1.7.3 virtual void TGo4AbstractInterface::AnalysisConfigName (const char * *filename*) `[inline, virtual]`

Configure name of file, where analysis configuration will be saved

2.1.7.4 TGo4BrowserProxy* TGo4AbstractInterface::Browser () const [inline]

Returns pointer on browser proxy. TGo4BrowserProxy class provides complete interface to Go4 browser functionality. Should only be used when interface does not provide required functionality. Can be accessed direcly via "br" variable.

2.1.7.5 virtual void TGo4AbstractInterface::ConfigStep (const char * *stepname*, Bool_t *enableprocess*, Bool_t *enablesource*, Bool_t *enablestore*) [inline, virtual]

Set basic step property

2.1.7.6 virtual void TGo4AbstractInterface::ConnectAnalysis (const char * *ServerNode*, Int_t *ServerPort*, Int_t *UserMode*, const char * *password* = 0) [inline, virtual]

Connect to running analysis server. Parameters: ServerNode - node to connect to, ServerPort - connection port number, UserMode - mode of user operation : 0 - observer, 1 - controller, 2 - administrator password - access password, which should correspond to specified UserMode

2.1.7.7 void TGo4AbstractInterface::ConnectHServer (const char * *servername*, Int_t *portnumber*, const char * *basename*, const char * *userpass*, const char * *filter*) [virtual]

Connecta to GSI histogram server. Creates appropriate entry in browser and provides access to histogram. Parameters: servername - IP server name portnumber - socket port number basename - histogram server base name userpass - password to access histogram server filter - filter for historgams names Several connections to different histogram servers are allowed

2.1.7.8 Bool_t TGo4AbstractInterface::CopyItem (const char * *itemname*) [virtual]

Copy item to workspace. If there is subfolders with items, they also will be copied.

2.1.7.9 void TGo4AbstractInterface::DeleteInstance () [static]

Delete instance of interface class

2.1.7.10 Bool_t TGo4AbstractInterface::DeleteItem (const char * *itemname*) [virtual]

Delete item (if allowed). Can be used to close file, delete memory object, close hiostogram server connection.

2.1.7.11 virtual void TGo4AbstractInterface::DisconnectAnalysis () [inline, virtual]

Disconnects from running analysis. If analysis is running in client mode, analysis will be shutdown.

2.1.7.12 virtual void TGo4AbstractInterface::DivideViewPanel (ViewPanelHandle *panel*, Int_t *numX*, Int_t *numY*) [inline, virtual]

Divide viewpanel on subpads.

2.1.7.13 virtual Bool_t TGo4AbstractInterface::DrawItem (const char * *itemname*, ViewPanelHandle *panel* = 0, const char * *drawopt* = 0) [inline, virtual]

Draw browser item on specified viewpanel. Parameters: itemname - browser item name, panel - viewpanel, if panel==0, new viewpanel will be created. drawopt - draw options, used in obj->Draw() operation

2.1.7.14 void TGo4AbstractInterface::ExecuteLine (const char * *remotecmd*) [virtual]

Execute one macro line on analysis side. This allows to call any action on analysis, including execution scripts with ".x userscript.C". One should not mix scripts, written for analysis and scripts, written for GUI.

2.1.7.15 Bool_t TGo4AbstractInterface::ExportToFile (const char * *itemname*, const char * *dirpath*, const char * *format*, const char * *filetitle* = 0) [virtual]

Export browser item to different file formats. One, probably, should use **FetchItem**() (page 60) before calling this method. If item is folder and contains sub-items, they also will be exported. Parameters: itemname - name of browser, which should be exported, dirpath - directory on disk, where files should be created. format - export file format, can be "ASCII" - text format "Radware" - Radware format (Origin) "ROOT" - binary ROOT format "ROOT XML" - xml ROOT format filetitle - title of create file (only for ROOT formats)

2.1.7.16 void TGo4AbstractInterface::FetchItem (const char * *itemname*, Int_t *wait_time* = 2000) [virtual]

Fetch item from data source. Request item from data source, to which item corresponds to. If this is file or histogram server, object will be returned immdediately. If this is an item from analysis, first request will be send and then interface will wait "wait_time" milliseconds that object is arrived. If wait_time==0, no waiting will be done and most probably, new object will be assigned to that item several seconds after method is return

2.1.7.17 TString TGo4AbstractInterface::FindItem (const char * *objname*) [virtual]

Find item with given object name. Item name includes object name and name of all parent folders. For instance histogram of name "His1" in analysis will have item name "Analysis/Histograms/His1".

2.1.7.18 virtual ViewPanelHandle TGo4AbstractInterface::FindViewPanel (const char * *name*) [inline, virtual]

Return handle on viewpanel with specified name/

2.1.7.19 TObject * TGo4AbstractInterface::GetObject (const char * *itemname*, Int_t *updatelevel* = 1) [virtual]

Returns object, assigned to specified browser item. Parameter updatelevel specifies, how object will be requested from data source, to which item correspond to. Possible values for updatelevel parameter: 0 - no request will be done, return last requested object 1 - request to data source only first time, no waiting 2 - request to data source in any case, no waiting >9 - request to data source and wait as many milliseconds as specified by updatelevel Wating required, when object requested from the analysis

2.1.7.20 virtual TString TGo4AbstractInterface::GetViewPanelName (ViewPanelHandle *panel*) [inline, virtual]

Return name of viewpanel.

2.1.7.21 virtual void TGo4AbstractInterface::HotStart (const char * *filename*) [pure virtual]

Executes hotstart file, generated in go4 GUI. Hot start files can not be executed as normal CINT scripts, therefore one should use this method to activate them

2.1.7.22 TGo4AbstractInterface * TGo4AbstractInterface::Instance () [static]

Return pointer on instance of implemntation of TGo4AbstractInterface class Normally, inside CINT script one should use "go4" variable, which containes **TGo4AbstractInterface::Instance**() (page 60) value.

2.1.7.23 Bool_t TGo4AbstractInterface::IsAnalysisConnected () [virtual]

Indicate, if analysis was succesfully connected

2.1.7.24 virtual void TGo4AbstractInterface::LaunchAnalysis (const char * *ClientName*, const char * *ClientDir*, const char * *ClientExec*, const char * *ClientNode*, Int_t *ShellMode* = 2, Int_t *TermMode* = 1) [inline, virtual]

Launch analysis in client mode. Parameters: ClientName - arbitrary name of analysis, used for diaplay ClientDir - directory, where analysis should be started ClientExec - main analysis executable ClientNode - node name, where analysis should be started ShellMode - shell, used to launch analysis: 1 - rsh, 2 - ssh [default] TermMode - terminal program: 1 - internal Qt window, 2 - xterm, 3 - KDE konsole

2.1.7.25 virtual void TGo4AbstractInterface::LaunchAnalysis () [inline, virtual]

Launch analysis in client mode, using default configuration. Can only work in standard GUI mode

2.1.7.26 void TGo4AbstractInterface::LoadLibrary (const char * *fname*) [virtual]

Load specified ROOT library

2.1.7.27 virtual void TGo4AbstractInterface::Message (const char * *msg*) [inline, virtual]

Dsiplay message in GUI status line

2.1.7.28 void TGo4AbstractInterface::MonitorItem (const char * *itemname*, Bool_t *on* = kTRUE) [virtual]

Enable/disable monitoring of browser item. In monitoring mode item regularly will be requested from analysis and viewpanels, where item is drawn, will be updated

2.1.7.29 TGo4ObjectManager* TGo4AbstractInterface::OM () const [inline]

Returns pointer on object manager. TGo4ObjectManager class should only be used when standard interface does not provide required functionality. Can be accessed direcly via "om" variable.

2.1.7.30 void TGo4AbstractInterface::OpenFile (const char * *fname*) [virtual]

Open specified file in read-only mode* File and its structure should appear in browser.

2.1.7.31 virtual void TGo4AbstractInterface::RedrawItem (const char * *itemname*) [virtual]

Redraw item of given name on all viewpanels/editors. Useful for the case, when content of object (histogram, for example) changed directly in script and after that should be updated in viewpanel.

2.1.7.32 virtual void TGo4AbstractInterface::RedrawPanel (ViewPanelHandle *panel*) [inline, virtual]

Forces of panel redraw.

2.1.7.33 void TGo4AbstractInterface::RequestAnalysisConfig () [virtual]

Requests current analysis configuration

2.1.7.34 Bool_t TGo4AbstractInterface::SaveToFile (const char * *itemname*, const char * *filename*, const char * *filetitle* = 0) [virtual]

Save specified browser item to file. Only object, which are already fetched from the data source, will be saved. Therefore, before one call this method, one probably should use **FetchItem**() (page 60) method to get objects from file/analysis/histogram server before save them. If item is folder and contains sub-items, they also will be saved. Only binary ROOT files (extension .root) and XML file (extension .xml) are supported.

2.1.7.35 TString TGo4AbstractInterface::SaveToMemory (const char * *path*, TObject * *obj*, Bool_t *ownership* = kFALSE) [virtual]

Save object in browser workspace. Object of any type can be saved in browser. It will appiar in browser folder "Workspace", where local memory objects are placed. Parameters: path - subpath in "Workspace" folder, where object should be placed obj - pointer ob object ownership - is browser becomes owner of that object If item of that name exists in browser, it will be overwritten by new object

2.1.7.36 virtual void TGo4AbstractInterface::SelectPad (ViewPanelHandle *panel*, Int_t *number* = 0) [inline, virtual]

Set active pad on viewpanel. If number==0, main pad (canvas) will be set as active, otherwise one of subpads will be activated. Most of methods, working with viewpanel, acting with active pad of this panel.

2.1.7.37 virtual void TGo4AbstractInterface::SetAnalysisConfigMode (int *mode*) [inline, virtual]

Set analysis configuration window mode. Parameters mode indicate that configuration window: -1 - closed, 0 - minimized, 1 - normal state

2.1.7.38 virtual void TGo4AbstractInterface::SetAnalysisTerminalMode (int *mode*) [inline, virtual]

Set analysis terminal window mode. Parameters mode indicate that terminal window: -1 - closed, 0 - minimized, 1 - normal state

2.1.7.39 virtual void TGo4AbstractInterface::SetSuperimpose (ViewPanelHandle *panel*, Bool_t *on* = kTRUE) [inline, virtual]

Set superimpose flag for active pad of viewpanel.

2.1.7.40 virtual Bool_t TGo4AbstractInterface::SetViewPanelName (ViewPanelHandle *panel*, const char * *newname*) [inline, virtual]

Change name of viewpanel.

2.1.7.41 virtual void TGo4AbstractInterface::ShutdownAnalysis () [inline, virtual]

Shutdown running analysis. If analysis is running in server mode, only user with administrator privilegies can do this

2.1.7.42 virtual void TGo4AbstractInterface::StartAnalysis () [inline, virtual]

Starts (resume) analysis execution

2.1.7.43 void TGo4AbstractInterface::StartMonitoring (Int_t *period* = 10) [virtual]

Start monitoring mode. Parameter "period" specifies how often (in seconds) each monitored item will be updated.

2.1.7.44 virtual ViewPanelHandle TGo4AbstractInterface::StartViewPanel (int *x*, int *y*, int *width*, int *height*, int *mode* = 1, TGo4Picture * *pic* = 0) [inline, virtual]

Create new viewpanel with specified parameters. Returns handle of newly created viewpanel. Parameters: x,y - left top corner coordinate of view panel; width, height - panel size; mode - diplay view panel 0 - minimized, 1 - normal, 2 - maximized; pic - TGo4Picture object, which includes configuration of viewpanel. To understand usage of this method together with TGo4Picture class, one can create and configure viewpanel and then create hotstart file. This file will include complete TGo4Picture configuration, which contains attributes like colors, ranges, pad divisions and so on.

2.1.7.45 virtual ViewPanelHandle TGo4AbstractInterface::StartViewPanel () [inline, virtual]

Create new view panel. Handle, returned by this method, must be used for other operation, like **DivideViewPanel**() (page 59) or **SelectPad**() (page 62)

2.1.7.46 virtual void TGo4AbstractInterface::StepBackStore (const char * *stepname*, const char * *storename*, int *bufsize*, int *splitlevel*) [inline, virtual]

Set back store ss step data storage

2.1.7.47 virtual void TGo4AbstractInterface::StepFileSource (const char * *stepname*, const char * *sourcename*, int *timeout*) [inline, virtual]

Set file source as step data source

2.1.7.48 virtual void TGo4AbstractInterface::StepFileStore (const char * *stepname*, const char * *storename*, bool *overwrite*, int *bufsize*, int *splitlevel*, int *compression*) [inline, virtual]

Set file as step data storage

2.1.7.49 virtual void TGo4AbstractInterface::StepMbsEventServerSource (const char * *stepname*, const char * *sourcename*, int *timeout*) [inline, virtual]

Set MBS event server as step data source

2.1.7.50 virtual void TGo4AbstractInterface::StepMbsFileSource (const char * *stepname*, const char * *sourcename*, int *timeout*, const char * *TagFile*, int *start*, int *stop*, int *interval*) [inline, virtual]

Set MBS file source as step data source

2.1.7.51 virtual void TGo4AbstractInterface::StepMbsRevServSource (const char * *stepname*, const char * *sourcename*, int *timeout*, int *port*) [inline, virtual]

Set MBS remote event server as step data source

2.1.7.52 virtual void TGo4AbstractInterface::StepMbsStreamSource (const char * *stepname*, const char * *sourcename*, int *timeout*) [inline, virtual]

Set MBS stream server as step data source

2.1.7.53 virtual void TGo4AbstractInterface::StepMbsTransportSource (const char * *stepname*, const char * *sourcename*, int *timeout*) [inline, virtual]

Set MBS transport server as step data source

2.1.7.54 virtual void TGo4AbstractInterface::StepRandomSource (const char * *stepname*, const char * *sourcename*, int *timeout*) [inline, virtual]

Set random generator as step data source

2.1.7.55 virtual void TGo4AbstractInterface::StepUserSource (const char * *stepname*, const char * *sourcename*, int *timeout*, int *port*, const char * *expr*) [inline, virtual]

Set user data source as step data source

2.1.7.56 virtual void TGo4AbstractInterface::StopAnalysis () [inline, virtual]

Stop (suspend) analysis execution

2.1.7.57 void TGo4AbstractInterface::StopMonitoring () [virtual]

Stop monitoring mode.

2.1.7.58 virtual void TGo4AbstractInterface::SubmitAnalysisConfig () [inline, virtual]

Submit configuration to analysis.

Configurations can be changed with following methods:

- **AnalysisAutoSave**() (page 58);
- **AnalysisConfigName**() (page58);
- **ConfigStep**() (page 59);
- **StepFileSource**() (page 63);
- **StepMbsFileSource**() (page 63);
- **StepMbsStreamSource**() (page63);
- **StepMbsTransportSource**() (page 63);
- **StepMbsEventServerSource**() (page 63);
- **StepMbsRevServSource**() (page 63);
- **StepRandomSource**() (page 63);
- **StepUserSource**() (page 63);
- **StepFileStore**() (page 63);
- **StepBackStore**() (page 63).

To understand more how these methods can be used together, one should launch analysis, configure it and then generate hotstart file. In this file one can find correct sequence and parameters for all these methods.

2.1.7.59 virtual void TGo4AbstractInterface::Wait (double *tm_sec*) [inline, virtual]

Wait specified number of seconds. Suppress macro execution, but keeps GUI functional, therefore it is different from gSystem->Sleep() call.

2.1.7.60 virtual void TGo4AbstractInterface::WaitAnalysis (Int_t *delay_sec*) [inline, virtual]

Waits, until connection to analysis is established. Method must be called before any other action like configuration, start/stop can be done. If analysis is connected, **IsAnalysisConnected**() (page 60) return kTRUE. delay_sec specifies, how long one should wait until analysis is connected

---

The documentation for this class was generated from the following files:

- TGo4AbstractInterface.h
- TGo4AbstractInterface.cxx

3 Index