ROOT

Classes Reference Manual

ROOT, Classes Reference Manual

Authors: René BRUN (*Rene.Brun@cern.ch*)
Nenad BUNCIC (*Nenad.Buncic@cern.ch*)
Valery FINE (*Valeri.Faine@cern.ch*)
Fons RADEMAKERS (*Fons.Rademakers@cern.ch*)

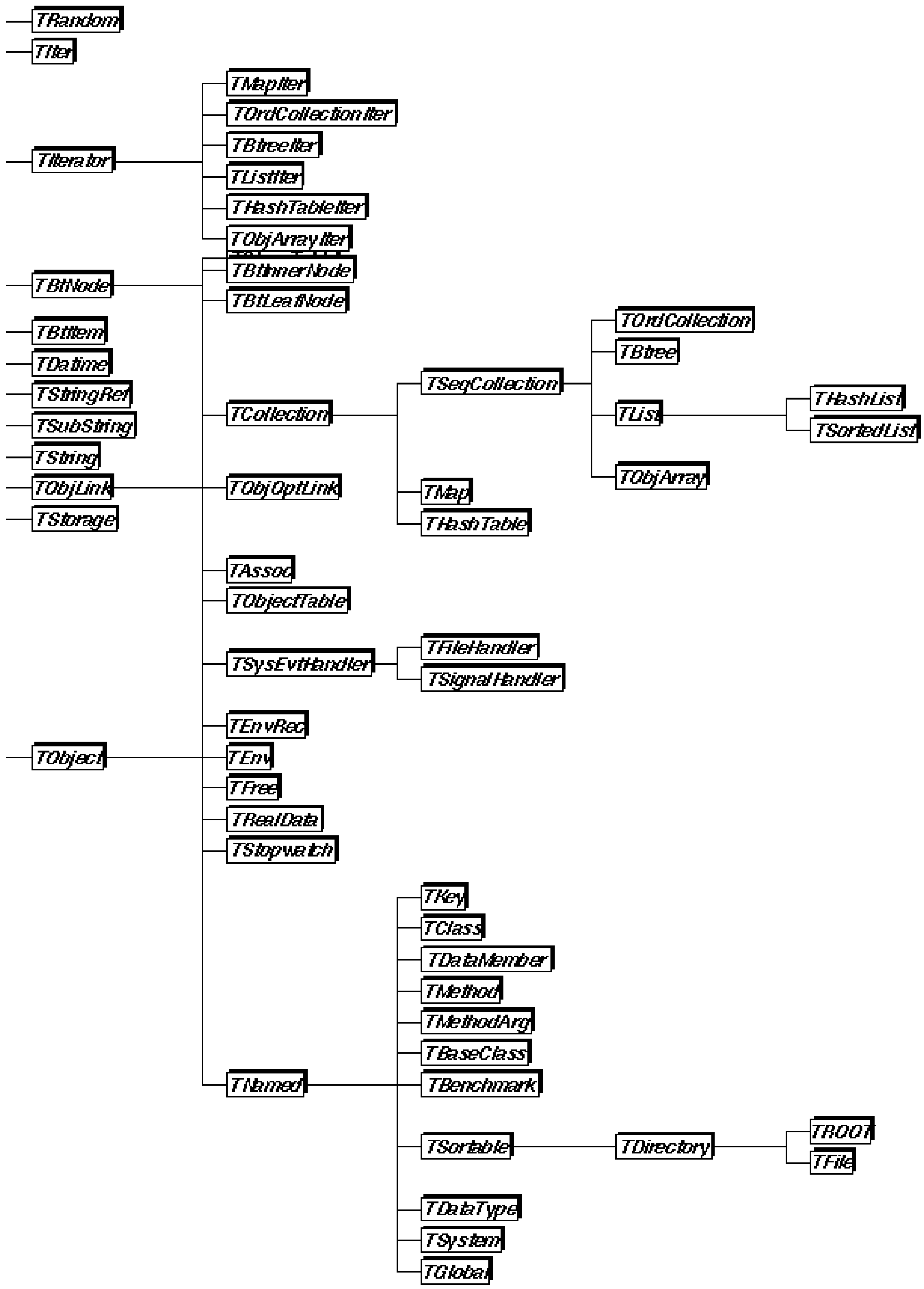
Cover Page: Marjana RAPAIC BUNCIC

Copyright (C) 1996 by CodeCERN.
All rights reserved.

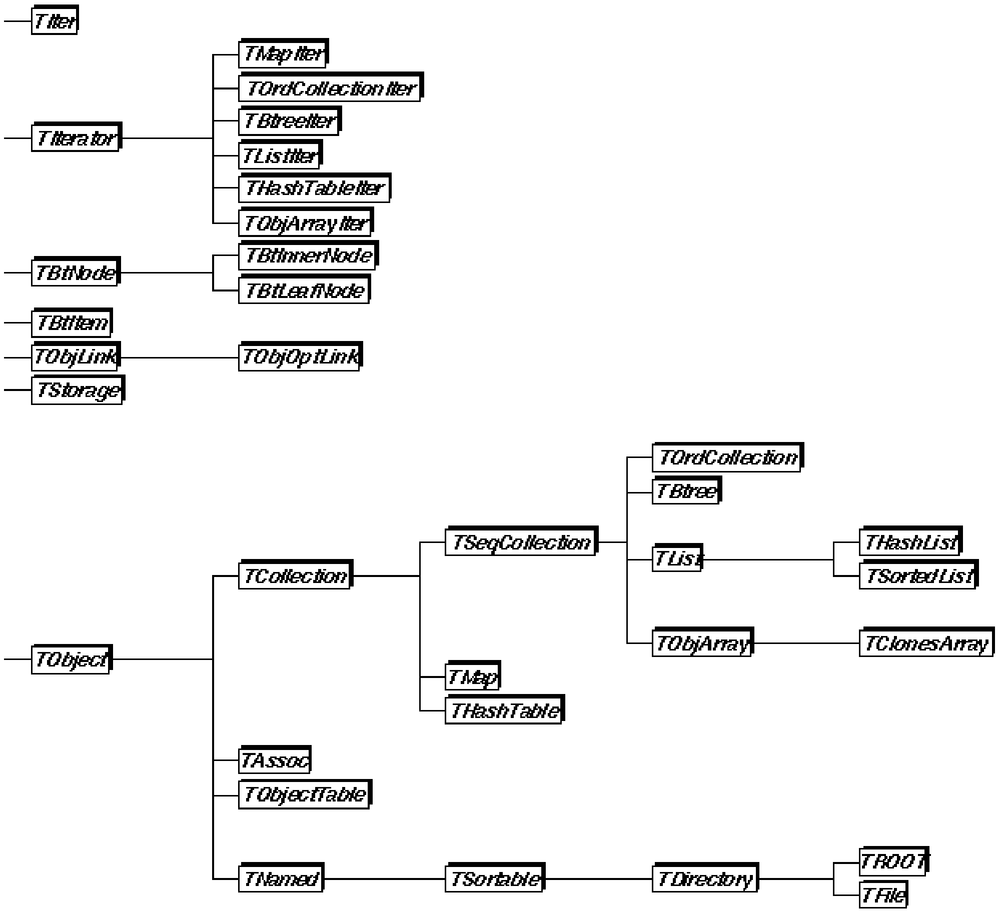
Table of Contents

Introduction

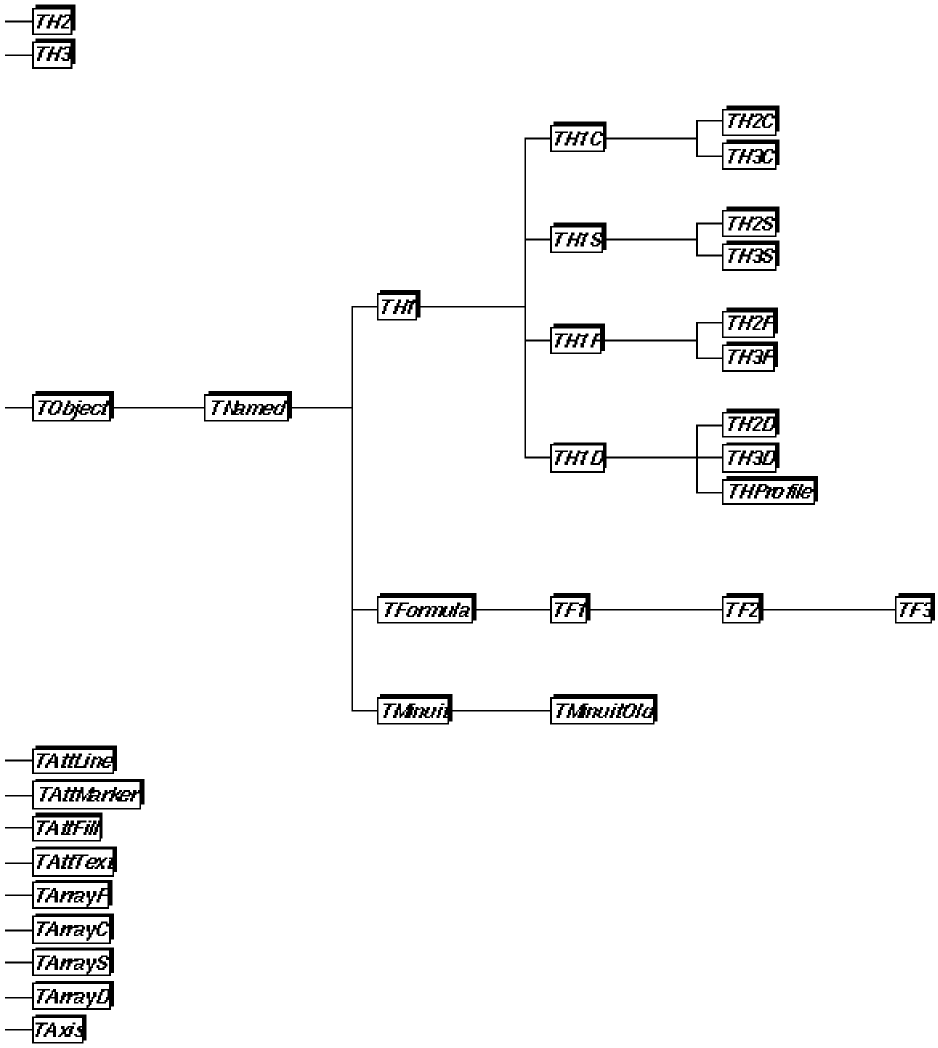
Basic ROOT classes



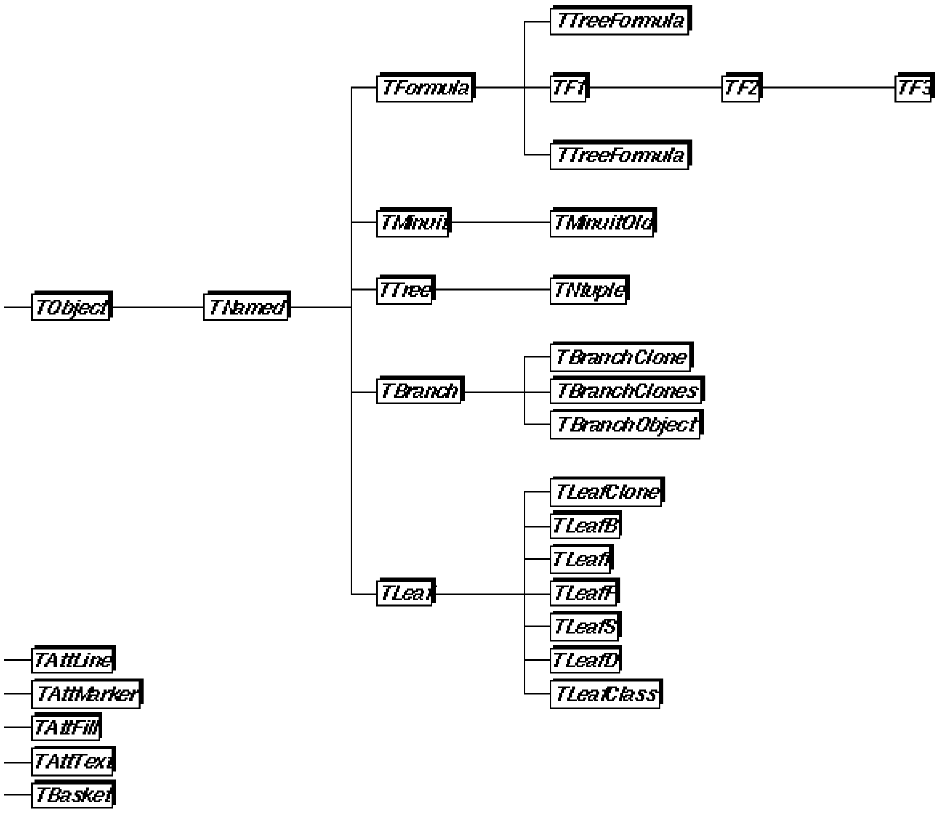
Container classes



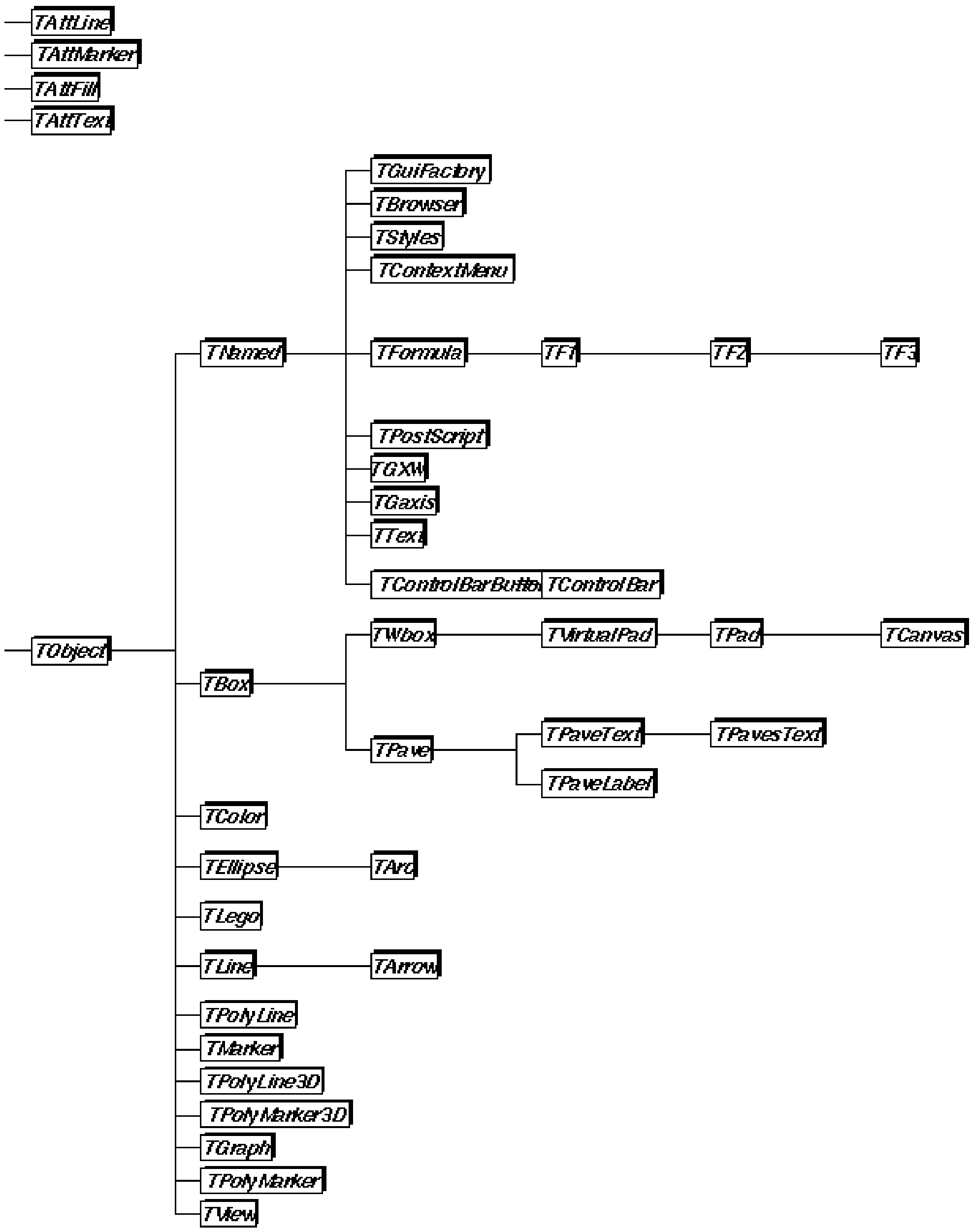
Histograms and Minimization



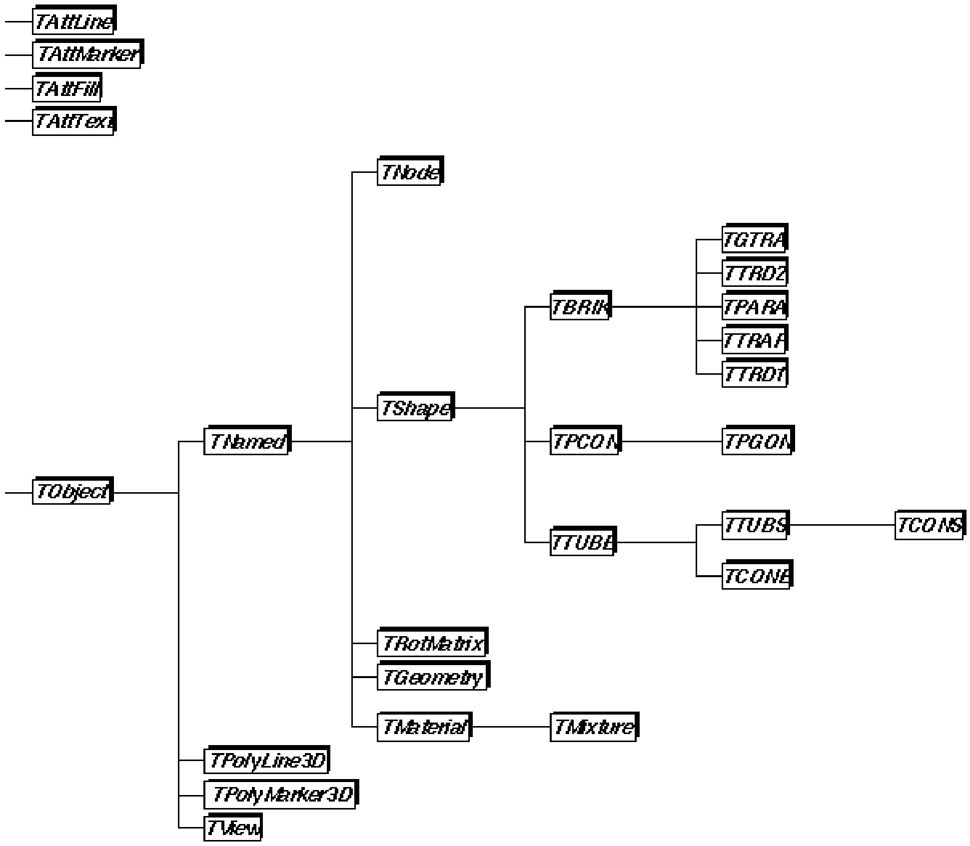
Trees and Ntuples



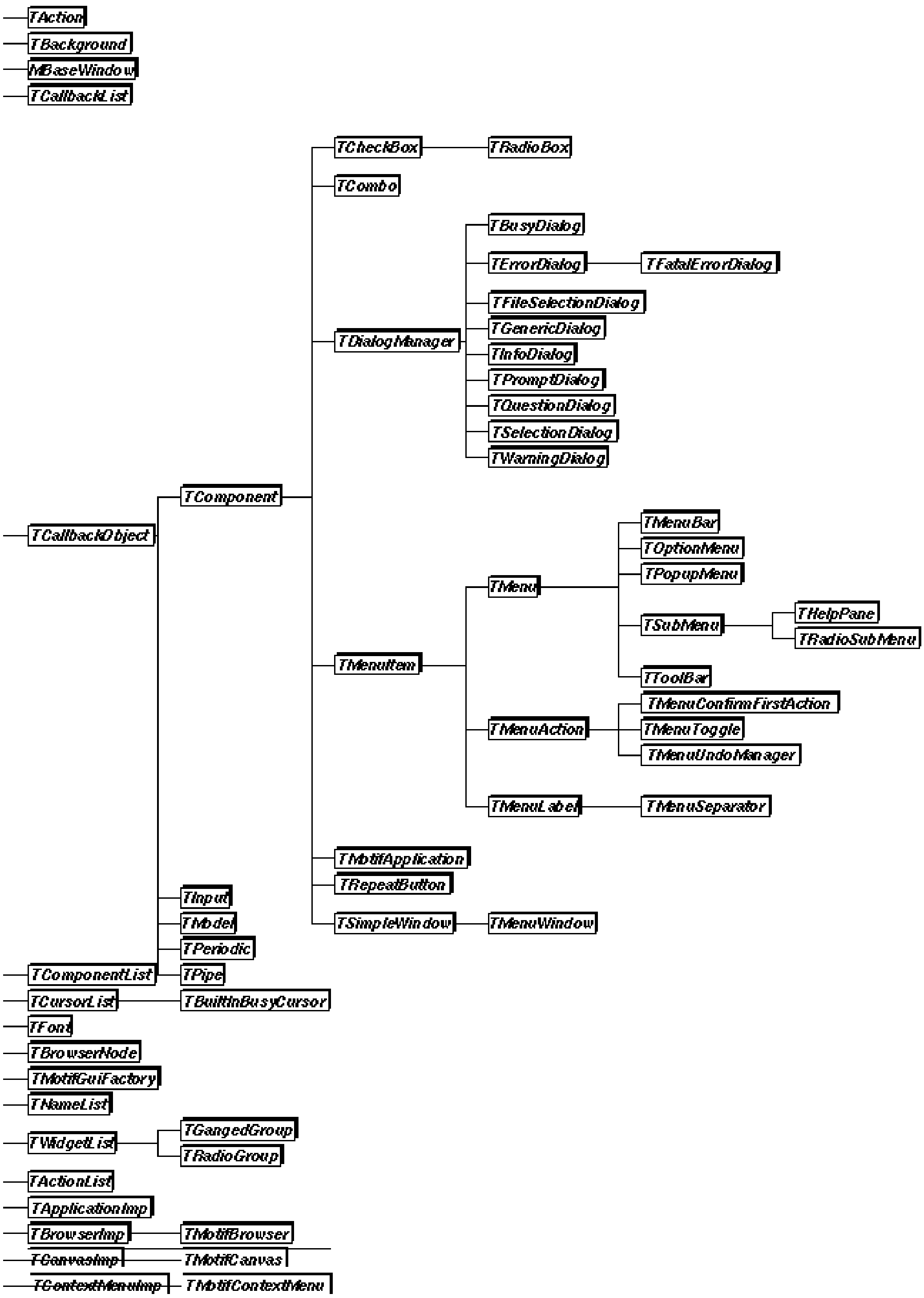
2D graphics


TAttLine
TAttMarker
TAttFill
TAttText
TObject
TNamed
TGuiFactory
TBrowser
TStyles
TContextMenu
TFormula
TF1
TF2
TF3
TPostScript
TGXW
TGaxis
TText
TControlBarButto
TControlBar
TBox
TWbox
TVirtualPad
TPad
TCanvas
TPave
TPaveText
TPavesText
TPaveLabel
TColor
TEllipse
TArc
TLego
TLine
TArrow
TPolyLine
TMarker
TPolyLine3D
TPolyMarker3D
TGraph
TPolyMarker
TView

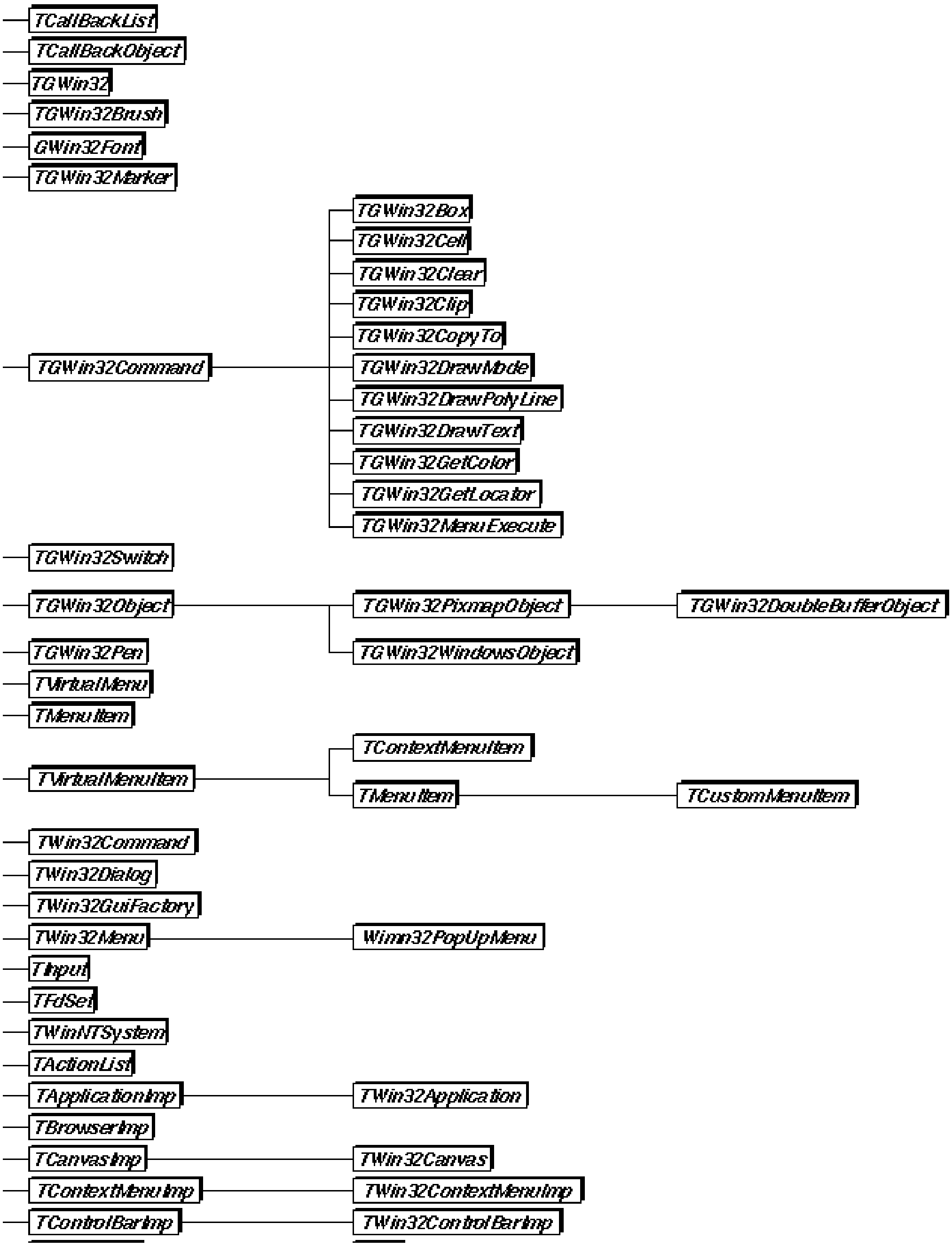
3D graphics and Detector geometry



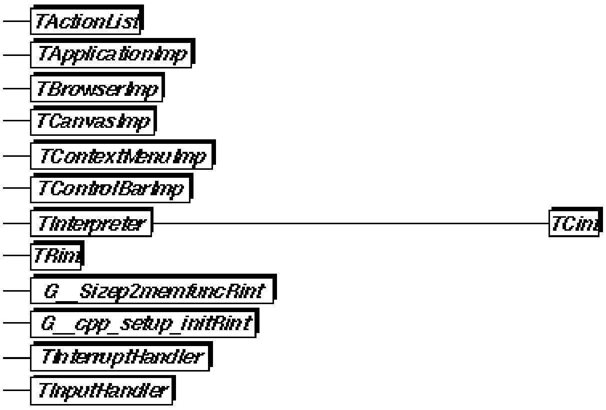
Graphical User Interface under MOTIF



Graphical User Interface under WINDOWS



Interactive Interface



Classes

TApplication

class **TApplication** : public TObject

protected:

class TApplication	**TApplication**( )
void	**Help**(Text_t* *line*)
void	**SetReturnFromRun**(Bool_t *ret*)
void	**SetSignalHandler**(class TSignalHandler* *sh*)

public:

class TApplication	**TApplication**(char* *appClassName,* int* *argc,* char** *argv,* void* *options = 0,* int *numOptions = 0*)
class TApplication	**TApplication**(class TApplication&)
void	**~TApplication**( )
virtual char*	**ApplicationName**( )
int	**Argc**( )
char**	**Argv**( )
char*	**Argv**(int *index*)
class TClass*	**Class**( )
void	**ClearInputFiles**( )
void	**CreateApplication**( )
virtual class TApplicationImp*	**GetApplicationImp**( )
const char*	**GetIdleCommand**( )
virtual void	**GetOptions**(int* *argc,* char** *argv*)
class TSignalHandler*	**GetSignalHandler**( )
virtual void	**HandleIdleTimer**( )
virtual void	**HandleTermInput**( )
virtual void	**Hide**( )
virtual void	**Iconify**( )
virtual void	**Init**( )
class TObjArray*	**InputFiles**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsCmdThread**( )
virtual void	**Lower**( )
Bool_t	**NoLogOpt**( )
virtual void	**Open**( )
virtual void	**ProcessFile**(Text_t* *line*)
virtual void	**ProcessLine**(Text_t* *line*)
Bool_t	**QuitOpt**( )
virtual void	**Raise**( )
virtual void	**RemoveIdleTimer**( )
Bool_t	**ReturnFromRun**( )
virtual void	**Run**(Bool_t *retrn = kFALSE*)
virtual void	**SetIdleTimer**(UInt_t *idleTimeInSec,* Text_t* *command*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**StartIdleing**( )
virtual void	**StopIdleing**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Terminate**(int *status = 0*)

Data Members

private:

Bool_t	**fReturnFromRun**	*When true return from Run( )*
int	**fArgc**	*Number of command line arguments*
char**	**fArgv**	*Command line arguments*

class TApplicationImp*	**fAppImp**	*Window system specific application implementation*
Bool_t	**fNoLog**	*Do not process logon and logoff macros*
Bool_t	**fQuit**	*Exit after having processed input files*
class TObjArray*	**fFiles**	*Array of input files (TObjString's)*
char*	**fIdleCommand**	*Command to execute while application is idle*
class TTimer*	**fIdleTimer**	*Idle timer*
class TSignalHandler*	**fSigHandler**	*Interrupt handler*

See Also: TProofServ, TRint

Class Description

```
TApplication

This class creates the ROOT Application Environment that interfaces
to the windowing system eventloop and eventhandlers.
This class must be instantiated exactly once in any given
application. Normally the specific application class inherits from
TApplication (see TRint).
```

TApplication(const char **appClassName,*int *argc, char **argv, void *options,int *numOptions*)

```
Create an application environment. The application environment
provides an interface to the Window Manager functionality and eventloop
(be it X, Windoze, Mac or Be).
```

~TApplication( )

```
TApplication dtor.
```

void **ClearInputFiles**( )

```
Clear list containing macro files passed as program arguments.
This method is called from TRint::Run() to ensure that the macro
files are only executed the first time Run() is called.
```

void **GetOptions**(int **argc,* char ***argv*)

```
Get and handle command line options.
```

void **HandleIdleTimer**( )

```
Handle idle timeout. When this timer expires the registered idle command
will be executed by this routine.
```

void **Help**(const Text_t **line*)

```
Print help on interpreter.
```

void **ProcessLine**(const Text_t **line*)

```
Process a single command line, either a C++ statement or an interpreter
command starting with a ".".
```

void **ProcessFile**(const Text_t **name*)

```
Process a file containing a C++ macro.
```

void **Run**(Bool_t *retrn*)

```
Main application eventloop. Calls system dependent eventloop via gSystem.
```

void **SetIdleTimer**(UInt_t *idleTimeInSec,* const Text_t **command*)

```
Set the command to be executed after the system has been idle for
idleTimeInSec seconds. Normally called via TROOT::Idle(...).
```

void **RemoveIdleTimer**( )

```
Remove idle timer. Normally called via TROOT::Idle(0).
```

void **StartIdleing**( )

```
Call when system starts idleing.
```

void **StopIdleing**( )

```
Call when system stops idleing.
```

void **Terminate**(int *status*)

```
Terminate the application by call TSystem::Exit() unless application has
been told to return from Run(), by a call to SetReturnFromRun().
```

void **CreateApplication**( )

```
Static function used to create a default application environment.
```

Author: Fons Rademakers 22/12/95
Last update: 1.03/06 20/10/97 13.45.29 by Fons Rademakers

TApplicationImp

class **TApplicationImp**

public:

class TApplicationImp	**TApplicationImp**( )
class TApplicationImp	**TApplicationImp**(char* *appClassName,* int* *argc,* char** *argv,* void* *options,* int *numOptions*)
class TApplicationImp	**TApplicationImp**(class TApplicationImp&)
void	**~TApplicationImp**( )
virtual char*	**ApplicationName**( )
class TClass*	**Class**( )
virtual void	**Hide**( )
virtual void	**Iconify**( )
virtual void	**Init**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsCmdThread**( )
virtual void	**Lower**( )
virtual void	**Open**( )
virtual void	**Raise**( )
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TMotifApplication

Class Description

```
TApplicationImp

ABC describing GUI independent application implementation protocol.
```

Author: Fons Rademakers 22/12/95
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TArc

class **TArc** : public TEllipse

public:

class TArc	**TArc**(Float_t *x1,* Float_t *y1,* Float_t *radius,* Float_t *phimin = 0,* Float_t *phimax = 360*)
class TArc	**TArc**( )
class TArc	**TArc**(class TArc& *arc*)
void	**~TArc**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *arc*)
virtual void	**DrawArc**(Float_t *x1,* Float_t *y1,* Float_t *radius,* Float_t *phimin = 0,* Float_t *phimax = 360*)
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**((*unknown*) ofstream& out, Option_t* option)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
An arc is specified with the position of its centre, its radius
a minimum and maximum angle.
The attributes of the outline line are given via TAttLine
The attributes of the fill area are given via TAttFill
```

TArc( ): **TEllipse**( )

```
Arc  default constructor
```

TArc(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *phimin,* Float_t *phimax*):**TEllipse**(*x1, y1, r1, r1, phimin, phimax, 0*)

```
Arc  normal constructor
x1,y1  : coordinates of centre of arc
r1     : arc radius
phimin : min and max angle in degrees (default is 0-->360)
phimax :
```

~TArc( )

```
Arc  default destructor
```

void **Copy**(TObject *&arc*)

```
Copy this arc to arc
```

void **DrawArc**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *phimin,* Float_t *phimax*)

```
Draw this arc with new coordinates
```

void **SavePrimitive**(ofstream *&out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 16/10/95
Last update: 1.02/00 09/07/97 22.12.17 by René Brun

TArray

class **TArray**

protected:

Bool_t	**BoundsOk**(char* *where,* Int_t *at*)
Bool_t	**OutOfBoundsError**(char* *where,* Int_t *i*)

public:

void	**~TArray**( )
class TClass*	**Class**( )
Int_t	**GetSize**( )
virtual class TClass*	**IsA**( )
virtual class TArray&	***operator=***(class TArray& *rhs*)
class TArray*	**ReadArray**(class TBuffer& *b,* class TClass* *clReq*)
virtual void	**Set**(Int_t *n*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**WriteArray**(class TBuffer& *b,* class TArray* *a*)

Data Members

public:

Int_t	**fN**	*Number of array elements*

See Also: TArrayC, TArrayD, TArrayF, TArrayI, TArrayL, TArrayS

Class Description

```
TArray

Abstract array base class. Used by TArrayC, TArrayS, TArrayI,
TArrayL, TArrayF and TArrayD.
Data member is public for historical reasons.
```

Bool_t **OutOfBoundsError**(const char *where, Int_t *i*) const

```
Generate an out-of-bounds error. Always returns false.
```

TArray* **ReadArray**(TBuffer &*b,* const TClass **clReq*)

```
Read TArray object from buffer.
```

void **WriteArray**(TBuffer &*b,* const TArray **a*)

```
Write TArray object to buffer.
```

Author: Fons Rademakers 21/10/97
Last update: 1.03/06 24/10/97 13.18.02 by Fons Rademakers

TArrayC

class **TArrayC** : public TArray

public:

class TArrayC	**TArrayC**(Int_t *n*)
class TArrayC	**TArrayC**(Int_t *n,* Char_t* *array*)
class TArrayC	**TArrayC**( )
class TArrayC	**TArrayC**(class TArrayC& *array*)
void	**~TArrayC**( )
void	**AddAt**(Char_t *c,* Int_t *i*)
void	**Adopt**(Int_t *n,* Char_t* *array*)
Char_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayC& *array*)
Char_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayC&	***operator*=**(class TArrayC& *rhs*)
Char_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Char_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Char_t*	**fArray**	*Array of fN chars*

See Also: TH1C

Class Description

```
TArrayC

Array of chars or bytes (8 bits per element).
```

TArrayC( )

```
Default TArrayC ctor.
```

TArrayC(Int_t *n*)

```
Create TArrayC object and set array size to n chars.
```

TArrayC(Int_t *n,* Char_t **array*)

```
Create TArrayC object and initialize it with values of array.
```

TArrayC(const TArrayC &*array*)

```
Copy constructor.
```

~TArrayC( )

```
Delete TArrayC object.
```

void **Adopt**(Int_t *n,* Char_t **arr*)

```
Adopt array arr into TArrayC, i.e. don't copy arr but use it directly
in TArrayC. User may not delete arr, TArrayC dtor will do it.
```

void **AddAt**(Char_t *c,* Int_t *i*)

```
Add char c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayC object to n chars. If n<0 leave array unchanged.
```

void **Set**(Int_t *n,* Char_t **array*)

```
Set array size of TArrayC object to n chars and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayC object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.22.33 by Fons Rademakers

TArrayD

class **TArrayD** : public TArray

public:

class TArrayD	**TArrayD**(Int_t *n*)
class TArrayD	**TArrayD**(Int_t *n,* Double_t* *array*)
class TArrayD	**TArrayD**( )
class TArrayD	**TArrayD**(class TArrayD& *array*)
void	**~TArrayD**( )
void	**AddAt**(Double_t *c,* Int_t *i*)
void	**Adopt**(Int_t *n,* Double_t* *array*)
Double_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayD& *array*)
Double_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayD&	***operator*=**(class TArrayD& *rhs*)
Double_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Double_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Double_t*	**fArray**	*Array of fN doubles*

See Also: TH1D

Class Description

```
TArrayD

Array of doubles (64 bits per element).
```

TArrayD( )

```
Default TArrayD ctor.
```

TArrayD(Int_t *n*)

```
Create TArrayD object and set array size to n doubles.
```

TArrayD(Int_t *n,* Double_t **array*)

```
Create TArrayD object and initialize it with values of array.
```

TArrayD(const TArrayD &*array*)

```
Copy constructor.
```

~TArrayD( )

```
Delete TArrayD object.
```

void **Adopt**(Int_t *n,* Double_t **arr*)

```
Adopt array arr into TArrayD, i.e. don't copy arr but use it directly
in TArrayD. User may not delete arr, TArrayD dtor will do it.
```

void **AddAt**(Double_t *c,* Int_t *i*)

```
Add double c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayD object to n doubles.
If n<0 leave array unchanged.
```

void **Set**(Int_t *n,* Double_t **array*)

```
Set array size of TArrayD object to n doubles and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayD object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.23.58 by Fons Rademakers

TArrayF

class **TArrayF** : public TArray

public:

class TArrayF	**TArrayF**(Int_t *n*)
class TArrayF	**TArrayF**(Int_t *n,* Float_t* *array*)
class TArrayF	**TArrayF**( )
class TArrayF	**TArrayF**(class TArrayF& *array*)
void	**~TArrayF**( )
void	**AddAt**(Float_t *c,* Int_t *i*)
void	**Adopt**(Int_t *n,* Float_t* *array*)
Float_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayF& *array*)
Float_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayF&	***operator*=**(class TArrayF& *rhs*)
Float_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Float_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Float_t*	**fArray**	*Array of fN floats*

See Also: TH1F

Class Description

```
TArrayF

Array of floats (32 bits per element).
```

TArrayF( )

```
Default TArrayF ctor.
```

TArrayF(Int_t *n*)

```
Create TArrayF object and set array size to n floats.
```

TArrayF(Int_t *n,* Float_t **array*)

```
Create TArrayF object and initialize it with values of array.
```

TArrayF(const TArrayF &*array*)

```
Copy constructor.
```

~TArrayF( )

```
Delete TArrayF object.
```

void **Adopt**(Int_t *n,* Float_t **arr*)

```
Adopt array arr into TArrayF, i.e. don’t copy arr but use it directly
in TArrayF. User may not delete arr, TArrayF dtor will do it.
```

void **AddAt**(Float_t *c,* Int_t *i*)

```
Add float c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayF object to n floats.
If n<0 leave array unchanged.
```

void **Set**(Int_t *n*, Float_t **array*)

```
Set array size of TArrayF object to n floats and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayF object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.23.44 by Fons Rademakers

TArrayI

class **TArrayI** : public TArray

public:

class TArrayI	**TArrayI**(Int_t *n*)
class TArrayI	**TArrayI**(Int_t *n,* Int_t* *array*)
class TArrayI	**TArrayI**( )
class TArrayI	**TArrayI**(class TArrayI& *array*)
void	**~TArrayI**( )
void	**AddAt**(Int_t *i,* Int_t *idx*)
void	**Adopt**(Int_t *n,* Int_t* *array*)
Int_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayI& *array*)
Int_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayI&	***operator=***(class TArrayI& *rhs*)
Int_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Int_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Int_t*	**fArray**	*Array of fN 32 bit integers*

Class Description

```
TArrayI

Array of integers (32 bits per element).
```

TArrayI( )

```
Default TArrayI ctor.
```

TArrayI(Int_t *n*)

```
Create TArrayI object and set array size to n integers.
```

TArrayI(Int_t *n,* Int_t **array*)

```
Create TArrayI object and initialize it with values of array.
```

TArrayI(const TArrayI &*array*)

```
Copy constructor.
```

~**TArrayI**( )

```
Delete TArrayI object.
```

void **Adopt**(Int_t *n,* Int_t **arr*)

```
Adopt array arr into TArrayI, i.e. don't copy arr but use it directly
in TArrayI. User may not delete arr, TArrayI dtor will do it.
```

void **AddAt**(Int_t *c,* Int_t *i*)

```
Add Int_t c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayI object to n integers.
If n<0 leave array unchanged.
```

void **Set**(Int_t *n,* Int_t **array*)

```
Set array size of TArrayI object to n integers and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayI object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.23.07 by Fons Rademakers

TArrayL

class **TArrayL** : public TArray

public:

class TArrayL	**TArrayL**(Int_t *n*)
class TArrayL	**TArrayL**(Int_t *n,* Long_t* *array*)
class TArrayL	**TArrayL**( )
class TArrayL	**TArrayL**(class TArrayL& *array*)
void	**~TArrayL**( )
void	**AddAt**(Long_t *c,* Int_t *i*)
void	**Adopt**(Int_t *n,* Long_t* *array*)
Long_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayL& *array*)
Long_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayL&	***operator=***(class TArrayL& *rhs*)
Long_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Long_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Long_t*	**fArray**	*Array of fN longs*

Class Description

```
TArrayL

Array of longs (32 or 64 bits per element).
```

TArrayL( )

```
Default TArrayL ctor.
```

TArrayL(Int_t *n*)

```
Create TArrayL object and set array size to n longs.
```

TArrayL(Int_t *n,* Long_t **array*)

```
Create TArrayL object and initialize it with values of array.
```

TArrayL(const TArrayL &*array*)

```
Copy constructor.
```

~TArrayL( )

```
Delete TArrayL object.
```

void **Adopt**(Int_t *n,* Long_t **arr*)

```
Adopt array arr into TArrayL, i.e. don't copy arr but use it directly
in TArrayL. User may not delete arr, TArrayL dtor will do it.
```

void **AddAt**(Long_t *c,* Int_t *i*)

```
Add long c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayL object to n longs. If n<0 leave array unchanged.
```

void **Set**(Int_t *n,* Long_t **array*)

```
Set array size of TArrayL object to n longs and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayL object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.23.30 by Fons Rademakers

TArrayS

class **TArrayS** : public TArray

public:

class TArrayS	**TArrayS**(Int_t *n*)
class TArrayS	**TArrayS**(Int_t *n,* Short_t* *array*)
class TArrayS	**TArrayS**( )
class TArrayS	**TArrayS**(class TArrayS& *array*)
void	**~TArrayS**( )
void	**AddAt**(Short_t *c,* Int_t *idx*)
void	**Adopt**(Int_t *n,* Short_t* *array*)
Short_t	**At**(Int_t *i*)
class TClass*	**Class**( )
void	**Copy**(class TArrayS& *array*)
Short_t*	**GetArray**( )
Stat_t	**GetSum**( )
virtual class TClass*	**IsA**( )
class TArrayS&	***operator*=**(class TArrayS& *rhs*)
Short_t&	**operator[]**(Int_t *i*)
void	**Reset**( )
virtual void	**Set**(Int_t *n*)
void	**Set**(Int_t *n,* Short_t* *array*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Short_t*	**fArray**	*Array of fN shorts*

See Also: TH1S

Class Description

```
TArrayS

Array of shorts (16 bits per element).
```

TArrayS( )

```
Default TArrayS ctor.
```

TArrayS(Int_t *n*)

```
Create TArrayS object and set array size to n shorts.
```

TArrayS(Int_t *n,* Short_t **array*)

```
Create TArrayS object and initialize it with values of array.
```

TArrayS(const TArrayS &*array*)

```
Copy constructor.
```

~TArrayS( )

```
Delete TArrayS object.
```

void **Adopt**(Int_t *n,* Short_t **arr*)

```
Adopt array arr into TArrayS, i.e. don't copy arr but use it directly
in TArrayS. User may not delete arr, TArrayS dtor will do it.
```

void **AddAt**(Short_t *c,* Int_t *i*)

```
Add short c at position i. Check for out of bounds.
```

void **Set**(Int_t *n*)

```
Set array size of TArrayS object to n shorts.
If n<0 leave array unchanged.
```

void **Set**(Int_t *n,* Short_t **array*)

```
Set array size of TArrayS object to n shorts and copy array.
If n<0 leave array unchanged.
```

void **Streamer**(TBuffer &*b*)

```
Stream a TArrayS object.
```

Author: René Brun 06/03/95
Last update: 1.03/06 27/10/97 03.22.50 by Fons Rademakers

TArrow

class **TArrow** : public TLine, public TAttFill

public:

class TArrow	**TArrow**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Float_t *arrowsize* = 0*.05,* Option_t* *option* = >)
class TArrow	**TArrow**( )
class TArrow	**TArrow**(class TArrow& *arrow*)
void	**~TArrow**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *arrow*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawArrow**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Float_t *arrowsize* = 0*.05,* Option_t* *option* = >)
Float_t	**GetAngle**( )
Float_t	**GetArrowSize**( )
const Option_t*	**GetOption**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintArrow**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Float_t *arrowsize* = 0*.05,* Option_t* *option* = >)
virtual void	**SavePrimitive**((*unknown*) ofstream& out, Option_t* option)
virtual void	**SetAngle**(Float_t *angle* = *60*)
virtual void	**SetArrowSize**(Float_t *arrowsize* = 0*.05*)
virtual void	**SetOption**(Option_t* *option* = >)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

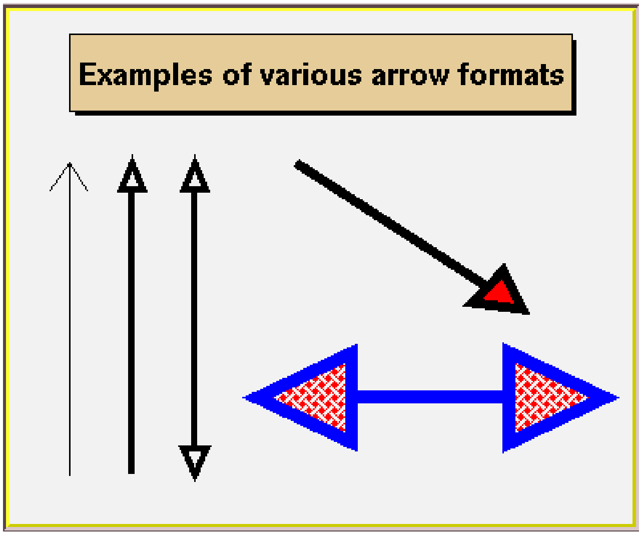
Data Members

protected:

Float_t	**fAngle**	*Arrow opening angle (degrees*)
Float_t	**fArrowSize**	*Arrow Size*
class TString	**fOption**	*Arrow shapes*

Class Description

```
Different arrow formats as explained in TArrow::TArrow or in
the picture below are provided.
Once an arrow is drawn on the screen:
 - One can click on one of the edges and move this edge.
 - One can click on any other arrow part to move the entire arrow.
```



TArrow( ): **TLine**( ), **TAttFill**( )

```
arrow default constructor
```

TArrow(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Float_t arrowsize, Option_t **option*):**TLine**(*x1, y1, x2, y2*), **TAttFill**(*0,* 1001)

```
arrow normal constructor
Define an arrow between points x1,y1 and x2,y2
the arrowsize is in percentage of the pad height
Opening angle between the two sides of the arrow is fAngle (60 degrees)
option = ">"      --------->
option = "<"      <---------
option = "<>"     <-------->
option = "<|>"    <|-----|>  arrow defined by a triangle
                 If FillColor == 0 draw open triangle
                 else  draw full triangle with fillcolor
```

~TArrow( )

```
arrow default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this arrow to arrow
```

void **Draw**(Option_t **option*)

```
Draw this arrow with its current attributes
```

void **DrawArrow**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Float_t arrowsize, Option_t **option*)

```
Draw this arrow with new coordinates
```

void **Paint**(Option_t **option*)

```
Paint this arrow with its current attributes
```

void **PaintArrow**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Float_t arrowsize, Option_t **option*)

```
Draw this arrow with new coordinates
                                        (P2)
                                         -
                                         .   -
                                         .      -
                                         .         -
       ----------------------------------.(P0)-------  (P1)
                                         .         -
                                         .      -
                                         .   -
                                         -
```

```
(P3)
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 17/10/95
Last update: 1.02/00 09/07/97 22.13.59 by René Brun

TAttAxis

class **TAttAxis**

public:

class TAttAxis	**TAttAxis**( )
class TAttAxis	**TAttAxis**(class TAttAxis&)
void	**~TAttAxis**( )
class TClass*	**Class**( )
void	**Copy**(class TAttAxis& *attaxis*)
virtual Color_t	**GetAxisColor**( )
virtual Color_t	**GetLabelColor**( )
virtual Style_t	**GetLabelFont**( )
virtual Float_t	**GetLabelOffset**( )
virtual Float_t	**GetLabelSize**( )
virtual Int_t	**GetNdivisions**( )
virtual Float_t	**GetTickLength**( )
virtual Float_t	**GetTitleOffset**( )
virtual class TClass*	**IsA**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SaveAttributes**((*unknown*) ofstream& out, char* name, char* subname)
virtual void	**SetAxisColor**(Color_t *color = 1*)
virtual void	**SetLabelColor**(Color_t *color = 1*)
virtual void	**SetLabelFont**(Style_t *font = 62*)
virtual void	**SetLabelOffset**(Float_t *offset* = 0*.005*)
virtual void	**SetLabelSize**(Float_t *size* = 0*.04*)
virtual void	**SetNdivisions**(Int_t *n = 510*)
virtual void	**SetTickLength**(Float_t *length* = 0*.03*)
virtual void	**SetTitleOffset**(Float_t *offset = 1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNdivisions**	*Number of divisions(10000*n3 + 100*n2 + n1)*
Color_t	**fAxisColor**	*color of the line axis*
Color_t	**fLabelColor**	*color of labels*
Style_t	**fLabelFont**	*font for labels*
Float_t	**fLabelOffset**	*offset of labels*
Float_t	**fLabelSize**	*size of labels*
Float_t	**fTickLength**	*length of tick marks*
Float_t	**fTitleOffset**	*offset of axis title*

See Also: TAxis

Class Description

```
Manages histogram axis attributes
```

void **SaveAttributes**(ofstream &*out,* const Text_t **name,* const Text_t **subname*)

```
Save axis attributes as C++ statement(s) on output stream out
```

void **SetAxisColor**(Color_t *color*)

```
Set color of the line axis and tick marks
```

void **SetLabelColor**(Color_t *color*)

```
Set color of labels and axis title
```

void **SetLabelOffset**(Float_t *offset*)
```
Set distance between the axis and the labels
The distance is expressed in per cent of the pad width
```

void **SetLabelSize**(Float_t *size*)
```
Set size of axis labels
The size is expressed in per cent of the pad width
```

void **SetNdivisions**(Int_t *n*)
```
Set the number of divisions for this axis
```

void **SetTickLength**(Float_t *length*)
```
Set tick mark length
The length is expressed in per cent of the pad width
```

void **SetTitleOffset**(Float_t *offset*)
```
Set distance between the axis and the axis title
Offset is a correction factor with respect to the "standard" value.
offset = 1  uses the default position that is computed in function
of the label offset and size.
offset = 1.2 will add 20 per cent more to the default offset.
```

Author: René Brun 12/12/94
Last update: 1.03/02 29/08/97 15.07.24 by René Brun

TAttCanvas

class **TAttCanvas**

public:

class TAttCanvas	**TAttCanvas**( )
class TAttCanvas	**TAttCanvas**(class TAttCanvas&)
void	**~TAttCanvas**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TAttCanvas& *attcanvas*)
Coord_t	**GetAdate**( )
Coord_t	**GetTitleFromTop**( )
Coord_t	**GetXBetween**( )
Coord_t	**GetXdate**( )
Coord_t	**GetYBetween**( )
Coord_t	**GetYdate**( )
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetAdate**(Coord_t *adate*)
virtual void	**SetTitleFromTop**(Coord_t *titlefromtop*)
virtual void	**SetXBetween**(Coord_t *xbetween*)
virtual void	**SetXdate**(Coord_t *xdate*)
virtual void	**SetYBetween**(Coord_t *ybetween*)
virtual void	**SetYdate**(Coord_t *ydate*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Coord_t	**fXBetween**	*X distance between pads*
Coord_t	**fYBetween**	*Y distance between pads*
Coord_t	**fTitleFromTop**	*Y distance of Global Title from top*
Coord_t	**fXdate**	*X position where to draw the date*
Coord_t	**fYdate**	*X position where to draw the date*
Coord_t	**fAdate**	*Alignment for the date*

Class Description

```
Manages canvas attributes. Referenced by TStyle.
```

void **Reset**(Option_t **option*)

Author: René Brun 12/12/94
Last update: 1.00/07 04/04/97 08.34.53 by René Brun

TAttFill

class **TAttFill**

public:

class TAttFill	**TAttFill**(Color_t *fcolor,* Style_t *fstyle*)
class TAttFill	**TAttFill**( )
class TAttFill	**TAttFill**(class TAttFill&)
void	**~TAttFill**( )
class TClass*	**Class**( )
void	**Copy**(class TAttFill& *attfill*)
Color_t	**GetFillColor**( )
Style_t	**GetFillStyle**( )
virtual class TClass*	**IsA**( )
virtual void	**Modify**( )
virtual void	**Reset**( )
virtual void	**SaveFillAttributes**((*unknown*) ofstream& out, Text_t* name, Int_t coldef = 1, Int_t stydef = 1001)
virtual void	**SetFillAttributes**( )
virtual void	**SetFillColor**(Color_t *fcolor*)
virtual void	**SetFillStyle**(Style_t *fstyle*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

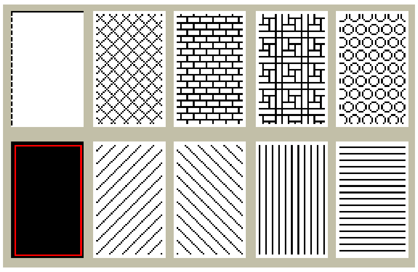
Color_t	**fFillColor**	*fill area color*
Style_t	**fFillStyle**	*fill area style*

See Also: TArrow, TBox, TEllipse, TF1, TGXW, TGraph, TH1, TLego, TMarker3DBox, TMaterial, TNode, TPolyLine, TPostScript, TShape, TStyle, TTree

Class Description

```
Fill Area Attributes class
Fill Area attributes are:
 Fill Color
 Fill Style

This class is used (in general by secondary inheritance)
by many other classes (graphics, histograms).

The following table shows the list of default colors.
```


```
Conventions for fill styles:
 0 = hollow
 1001 : Solid
 2001 : hatch style
 3000+pattern_number (see below)
The following table shows the list of pattern styles.
```



TAttFill( )

```
AttFill default constructor
Default fill attributes are taking from the current style
```

TAttFill(Color_t *color,* Style_t *style*)

```
AttFill normal constructor
 color Fill Color
 style Fill Style
```

~**TAttFill**( )

```
AttFill destructor
```

void **Copy**(TAttFill &*attfill*)

```
Copy this fill attributes to a new attfill
```

void **Modify**( )

```
Change current fill area attributes if necessary
```

void **Reset**( )

```
Reset this fill attributes to default values
```

void **SaveFillAttributes**(ofstream &*out,* const Text_t **name,* Int_t *coldef,* Int_t *stydef*)

```
Save fill attributes as C++ statement(s) on output stream out
```

void **SetFillAttributes**( )

```
Invoke the DialogCanvas Fill attributes
```

Author: René Brun 12/12/94
Last update: 1.01/05 13/06/97 10.38.00 by René Brun

TAttFillCanvas

class **TAttFillCanvas** : public TDialogCanvas

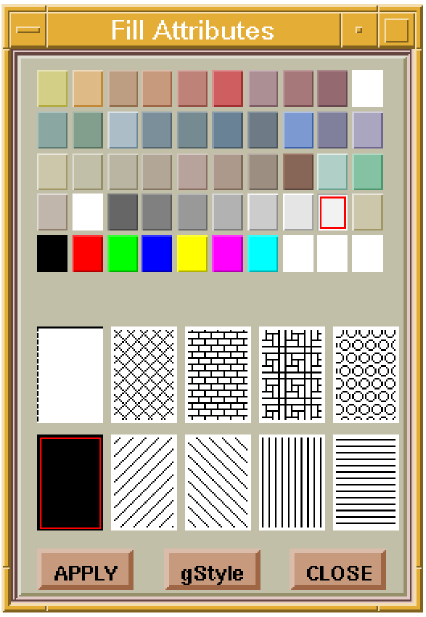
public:

class TAttFillCanvas	**TAttFillCanvas**( )
class TAttFillCanvas	**TAttFillCanvas**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 400,* UInt_t *wh = 600*)
class TAttFillCanvas	**TAttFillCanvas**(class TAttFillCanvas&)
void	**~TAttFillCanvas**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**UpdateFillAttributes**(Int_t *col,* Int_t *sty*)

Data Members

Class Description

```
An AttFillCanvas is a TDialogCanvas specialized to set fill attributes.
```



TAttFillCanvas( ) : **TDialogCanvas**( )

```
AttFillCanvas default constructor
```

TAttFillCanvas(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)

```
AttFillCanvas constructor
```

~TAttFillCanvas( )

```
AttFillCanvas default destructor
```

void **UpdateFillAttributes**(Int_t *col,* Int_t *sty*)

```
Update object attributes
```

Author: René Brun 04/07/96
Last update: 1.02/00 17/10/97 17.48.11 by René Brun

TAttFunc

class **TAttFunc**

public:

class TAttFunc	**TAttFunc**( )
class TAttFunc	**TAttFunc**(Color_t *fcolor,* Style_t *fstyle,* Width_t *fwidth*)
class TAttFunc	**TAttFunc**(class TAttFunc&)
void	**~TAttFunc**( )
class TClass*	**Class**( )
void	**Copy**(class TAttFunc& *attfunc*)
Color_t	**GetFuncColor**( )
Style_t	**GetFuncStyle**( )
Width_t	**GetFuncWidth**( )
virtual class TClass*	**IsA**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetFuncColor**(Color_t *fcolor*)
virtual void	**SetFuncStyle**(Style_t *fstyle*)
virtual void	**SetFuncWidth**(Width_t *fwidth*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Color_t	**fFuncColor**	*function color*
Style_t	**fFuncStyle**	*function style*
Width_t	**fFuncWidth**	*function width*

Class Description

```
Manages Functions attributes. Referenced by TStyle.
```

void **Reset**(Option_t **option*)

Author: René Brun 04/01/95
Last update: 1.00/07 04/04/97 08.34.53 by René Brun

TAttHist

class **TAttHist**

public:

class TAttHist	**TAttHist**(Color_t *lcolor,* Style_t *lstyle,* Width_t *lwidth*)
class TAttHist	**TAttHist**( )
class TAttHist	**TAttHist**(class TAttHist&)
void	**~TAttHist**( )
class TClass*	**Class**( )
void	**Copy**(class TAttHist& *atthist*)
Color_t	**GetHistColor**( )
Style_t	**GetHistStyle**( )
Width_t	**GetHistWidth**( )
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetHistColor**(Color_t *hcolor*)
virtual void	**SetHistStyle**(Style_t *hstyle*)
virtual void	**SetHistWidth**(Width_t *hwidth*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Color_t	**fHistColor**	*line color*
Style_t	**fHistStyle**	*line style*
Width_t	**fHistWidth**	*line width*

Class Description

```
Manages default histogram attributes. Referenced by TStyle.
```

void **Reset**(Option_t **option*)

Author: René Brun 12/12/94
Last update: 1.00/07 04/04/97 08.34.53 by René Brun

TAttLine

class **TAttLine**

public:

class TAttLine	**TAttLine**(Color_t *lcolor,* Style_t *lstyle,* Width_t *lwidth*)
class TAttLine	**TAttLine**(class TAttLine&)
class TAttLine	**TAttLine**( )
void	**~TAttLine**( )
class TClass*	**Class**( )
void	**Copy**(class TAttLine& *attline*)
Int_t	**DistancetoLine**(Int_t *px,* Int_t *py,* Float_t *xp1,* Float_t *yp1,* Float_t *xp2,* Float_t *yp2*)
Color_t	**GetLineColor**( )
Style_t	**GetLineStyle**( )
Width_t	**GetLineWidth**( )
virtual class TClass*	**IsA**( )
virtual void	**Modify**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SaveLineAttributes**((*unknown*) ofstream& out, Text_t* name, Int_t coldef = 1, Int_t stydef = 1, Int_t widdef = 1)
virtual void	**SetLineAttributes**( )
virtual void	**SetLineColor**(Color_t *lcolor*)
virtual void	**SetLineStyle**(Style_t *lstyle*)
virtual void	**SetLineWidth**(Width_t *lwidth*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Color_t	**fLineColor**	*line color*
Style_t	**fLineStyle**	*line style*
Width_t	**fLineWidth**	*line width*

See Also: TBox, TEllipse, TF1, TGXW, TGraph, TH1, TLego, TLine, TMarker3DBox, TNode, TPolyLine, TPolyLine3D, TPostScript, TShape, TStyle, TTree

Class Description

```
Line Attributes class
Line attributes are:
 Line Color (see TAttFill for the list of colors)
 Line Style 1=solid, 2=dash, 3=dash-dot, 4=dot-dot
 Line Width expressed in pixel units

This class is used (in general by secondary inheritance)
by many other classes (graphics, histograms).
```

TAttLine( )

```
AttLine default constructor
Default line attributes are taking from the current style
```

TAttLine(Color_t *color,* Style_t *style,* Width_t *width*)

```
AttLine normal constructor
Line attributes are taking from the argument list
 color : must be one of the valid color index
 style : 1=solid, 2=dash, 3=dash-dot, 4=dot-dot
 width : expressed in pixel units
```

~TAttLine( )

```
AttLine destructor
```

void **Copy**(TAttLine &*attline*)

```
Copy this line attributes to a new attline
```

Int_t **DistancetoLine**(Int_t *px,* Int_t *py,* Float_t *xp1,* Float_t *yp1,* Float_t *xp2,* Float_t *yp2*)

```
Compute distance from point px,py to a line
Compute the closest distance of approach from point px,py to this line.
The distance is computed in pixels units.

Algorithm:

 A(x1,y1)         P                            B(x2,y2)
 ------------------------------------------------
                  I
                  I
                  I
                  I
                 M(x,y)

Let us call  a = distance AM     A=a**2
            b = distance BM     B=b**2
            c = distance AB     C=c**2
            d = distance PM     D=d**2
            u = distance AP     U=u**2
            v = distance BP     V=v**2     c = u + v

D = A - U
D = B - V  = B -(c-u)**2
  ==> u = (A -B +C)/2c
```

void **Modify**( )

```
Change current line attributes if necessary
```

void **Reset**(Option_t **option*)

```
Reset this line attributes to default values
```

void **SaveLineAttributes**(ofstream &*out,* const Text_t **name,* Int_t *coldef,* Int_t *stydef,* Int_t *widdef*)

```
Save line attributes as C++ statement(s) on output stream out
```

void **SetLineAttributes**( )

```
Invoke the DialogCanvas Line attributes
```

Author: René Brun 28/11/94
Last update: 1.01/05 13/06/97 10.37.35 by René Brun

TAttLineCanvas

class **TAttLineCanvas** : public TDialogCanvas

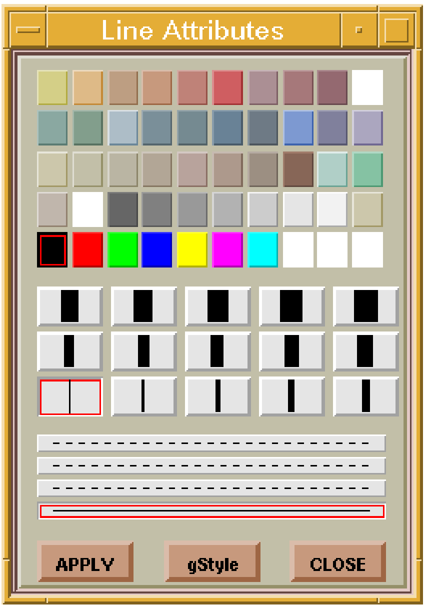
public:

class TAttLineCanvas	**TAttLineCanvas**( )
class TAttLineCanvas	**TAttLineCanvas**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 400,* UInt_t *wh = 600*)
class TAttLineCanvas	**TAttLineCanvas**(class TAttLineCanvas&)
void	**~TAttLineCanvas**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**UpdateLineAttributes**(Int_t *col,* Int_t *sty,* Int_t *width*)

Data Members

Class Description

```
An AttLineCanvas is a TDialogCanvas specialized to set line attributes.
```



TAttLineCanvas( ) : **TDialogCanvas**( )

```
AttLineCanvas default constructor
```

TAttLineCanvas(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)

```
AttLineCanvas constructor
```

~TAttLineCanvas( )

```
AttLineCanvas default destructor
```

void **UpdateLineAttributes**(Int_t *col,* Int_t *sty,* Int_t *width*)

```
Update object attributes
```

Author: René Brun 03/07/96
Last update: 1.03/06 17/10/97 19.17.41 by René Brun

TAttMarker

class **TAttMarker**

public:

class TAttMarker	**TAttMarker**(Color_t *color,* Style_t *style,* Size_t *msize*)
class TAttMarker	**TAttMarker**( )
class TAttMarker	**TAttMarker**(class TAttMarker&)
void	**~TAttMarker**( )
class TClass*	**Class**( )
void	**Copy**(class TAttMarker& *attmarker*)
virtual Color_t	**GetMarkerColor**( )
virtual Size_t	**GetMarkerSize**( )
virtual Style_t	**GetMarkerStyle**( )
virtual class TClass*	**IsA**( )
virtual void	**Modify**( )
virtual void	**Reset**(Option_t* *toption*)
virtual void	**SaveMarkerAttributes**(class ofstream& *out,* Text_t* *name,* Int_t *coldef = 1,* Int_t *stydef = 1,* Int_t *sizdef = 1*)
virtual void	**SetMarkerAttributes**( )
virtual void	**SetMarkerColor**(Color_t *tcolor = 1*)
virtual void	**SetMarkerSize**(Size_t *msize = 1*)
virtual void	**SetMarkerStyle**(Style_t *mstyle = 1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

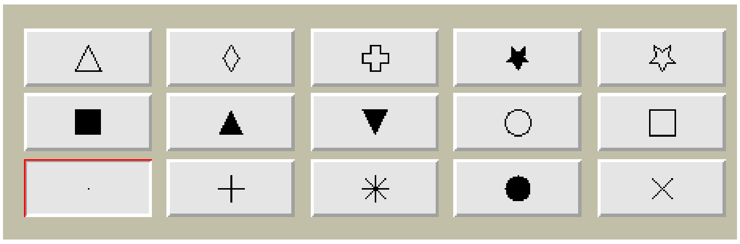
Color_t	**fMarkerColor**	*Marker color index*
Style_t	**fMarkerStyle**	*Marker style*
Size_t	**fMarkerSize**	*Marker size*

See Also: TF1, TGXW, TGraph, TH1, TMarker, TPolyMarker, TPolyMarker3D, TPostScript, TStyle, TTree

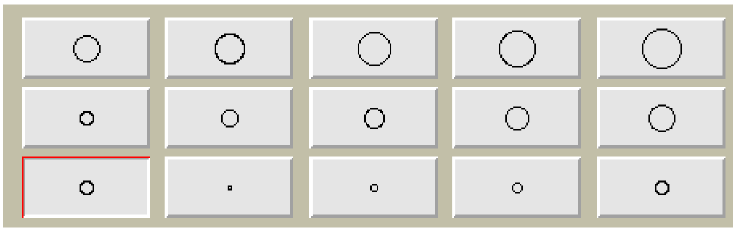
Class Description

```
Marker Attributes class
Marker attributes are:
 Marker Color
 Marker style
 Marker Size

This class is used (in general by secondary inheritance)
by many other classes (graphics, histograms).

List of the currently supported markers (screen and PostScript)
   1 : dot
   2 : +
   3 :
   4 : o (open circle)
   5 : x
   6 : full circle (small)
   7 : full circle (medium)
   8 : full circle (large)
   9 -->19 : dot
  20 : full circle (large)
  21 : full triangle up
  22 : full triangle down
  23 : two nested open hexagones
```

```
24 : open square
25 : open triangle up
26 : open diamond
27 : open cross
28 : full pentagon
29 : open hexagone
30 : open square
```



```
Various marker sizes are shown in the figure below.
The default marker size=1 is shown in the botton left corner.
Marker sizes smaller than 1 can be specified.
```



TAttMarker( )

```
AttMarker default constructor
Default text attributes are taking from the current style
```

TAttMarker(Color_t *color,* Style_t *style,* Size_t *msize*)

```
AttMarker normal constructor
Text attributes are taking from the argument list
 color : Marker Color Index
 style : Marker style (from 1 to 30)
 size  : marker size (float)
```

~**TAttMarker**( )

```
AttMarker destructor
```

void **Copy**(TAttMarker &*attmarker*)

```
Copy this marker attributes to a new atttext
```

void **Modify**( )

```
Change current marker attributes if necessary
```

void **Reset**(Option_t **option*)

```
Reset this marker attributes to default values
```

void **SaveMarkerAttributes**(ofstream &*out,* const Text_t **name,* Int_t *coldef,* Int_t *stydef,* Int_t *sizdef*)

```
Save line attributes as C++ statement(s) on output stream out
```

void **SetMarkerAttributes**( )

```
Invoke the DialogCanvas Marker attributes
```

Author: René Brun 12/05/95
Last update: 1.01/05 13/06/97 10.38.49 by René Brun

TAttMarkerCanvas

class **TAttMarkerCanvas** : public TDialogCanvas

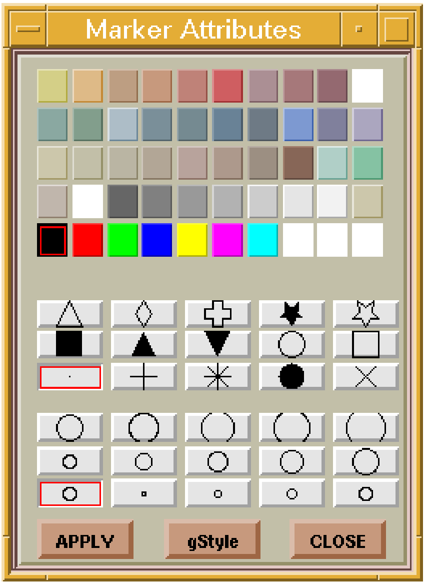
public:

class TAttMarkerCanvas	**TAttMarkerCanvas**( )
class TAttMarkerCanvas	**TAttMarkerCanvas**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 400,* UInt_t *wh = 600*)
class TAttMarkerCanvas	**TAttMarkerCanvas**(class TAttMarkerCanvas&)
void	**~TAttMarkerCanvas**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**UpdateMarkerAttributes**(Int_t *col,* Int_t *sty,* Float_t *msiz*)

Data Members

Class Description

```
An AttMarkerCanvas is a TDialogCanvas specialized to set line attributes.
```



TAttMarkerCanvas( ) : **TDialogCanvas**( )

```
AttMarkerCanvas default constructor
```

TAttMarkerCanvas(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)

```
AttMarkerCanvas constructor
```

~TAttMarkerCanvas( )

```
AttMarkerCanvas default destructor
```

void **UpdateMarkerAttributes**(Int_t *col,* Int_t *sty,* Float_t *msiz*)

```
Update marker attributes
```

Author: René Brun 04/07/96
Last update: 1.02/00 09/07/97 22.00.59 by René Brun

TAttPad

class **TAttPad**

public:

class TAttPad	**TAttPad**( )
class TAttPad	**TAttPad**(class TAttPad&)
void	**~TAttPad**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TAttPad& *attpad*)
Coord_t	**GetAfile**( )
Coord_t	**GetAstat**( )
Coord_t	**GetBottomMargin**( )
Coord_t	**GetLeftMargin**( )
Coord_t	**GetRightMargin**( )
Coord_t	**GetTopMargin**( )
Coord_t	**GetXfile**( )
Coord_t	**GetXstat**( )
Coord_t	**GetYfile**( )
Coord_t	**GetYstat**( )
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetAfile**(Coord_t *afile*)
virtual void	**SetAstat**(Coord_t *astat*)
virtual void	**SetBottomMargin**(Coord_t *bottommargin*)
virtual void	**SetLeftMargin**(Coord_t *leftmargin*)
virtual void	**SetRightMargin**(Coord_t *rightmargin*)
virtual void	**SetTopMargin**(Coord_t *topmargin*)
virtual void	**SetXfile**(Coord_t *xfile*)
virtual void	**SetXstat**(Coord_t *xstat*)
virtual void	**SetYfile**(Coord_t *yfile*)
virtual void	**SetYstat**(Coord_t *ystat*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Coord_t	**fLeftMargin**	*LeftMargin*
Coord_t	**fRightMargin**	*RightMargin*
Coord_t	**fBottomMargin**	*BottomMargin*
Coord_t	**fTopMargin**	*TopMargin*
Coord_t	**fXfile**	*X position where to draw the file name*
Coord_t	**fYfile**	*X position where to draw the file name*
Coord_t	**fAfile**	*Alignment for the file name*
Coord_t	**fXstat**	*X position where to draw the statistics*
Coord_t	**fYstat**	*X position where to draw the statistics*
Coord_t	**fAstat**	*Alignment for the statistics*

See Also: TVirtualPad

Class Description

```
Manages default Pad attributes. Referenced by TStyle.
```

void **SetBottomMargin**(Float_t *margin*)

```
Set Pad bottom margin in per cent of the pad height
```

void **SetLeftMargin**(Float_t *margin*)
```
Set Pad left margin in per cent of the pad width
```

void **SetRightMargin**(Float_t *margin*)
```
Set Pad right margin in per cent of the pad width
```

void **SetTopMargin**(Float_t *margin*)
```
Set Pad top margin in per cent of the pad height
```

Author: René Brun 04/01/95
Last update: 1.00/07 04/04/97 08.34.53 by René Brun

TAttStat

class **TAttStat**

public:

class TAttStat	**TAttStat**( )
class TAttStat	**TAttStat**(Color_t *color,* Style_t *style,* Float_t *w,* Float_t *h,* Float_t *x,* Float_t *y*)
class TAttStat	**TAttStat**(class TAttStat&)
void	**~TAttStat**( )
class TClass*	**Class**( )
void	**Copy**(class TAttStat& *attstat*)
Color_t	**GetStatColor**( )
Float_t	**GetStatH**( )
Style_t	**GetStatStyle**( )
Float_t	**GetStatW**( )
Float_t	**GetStatX**( )
Float_t	**GetStatY**( )
virtual class TClass*	**IsA**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetStatColor**(Color_t *scolor*)
virtual void	**SetStatH**(Float_t *h*)
virtual void	**SetStatStyle**(Style_t *sstyle*)
virtual void	**SetStatW**(Float_t *w*)
virtual void	**SetStatX**(Float_t *x*)
virtual void	**SetStatY**(Float_t *y*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Color_t	**fStatColor**	*stat color*
Style_t	**fStatStyle**	*stat style*
Float_t	**fStatX**	*X position of top right corner of stat box*
Float_t	**fStatY**	*Y position of top right corner of stat box*
Float_t	**fStatW**	*width of stat box*
Float_t	**fStatH**	*height of stat box*

Class Description

```
Manages default histogram statistics style and options.
Referenced by TStyle.
```

void **Reset**(Option_t **option*)

Author: René Brun 04/01/95
Last update: 1.00/07 04/04/97 08.34.53 by René Brun

TAttText

class **TAttText**

public:

class TAttText	**TAttText**( )
class TAttText	**TAttText**(Int_t *align,* Float_t *angle,* Color_t *color,* Style_t *font,* Float_t *tsize*)
class TAttText	**TAttText**(class TAttText&)
void	**~TAttText**( )
class TClass*	**Class**( )
void	**Copy**(class TAttText& *atttext*)
Short_t	**GetTextAlign**( )
Float_t	**GetTextAngle**( )
Color_t	**GetTextColor**( )
Font_t	**GetTextFont**( )
Float_t	**GetTextSize**( )
virtual class TClass*	**IsA**( )
virtual void	**Modify**( )
virtual void	**Reset**(Option_t* *toption*)
virtual void	**SaveTextAttributes**(class ofstream& *out,* Text_t* *name,* Int_t *alidef = 12,* Float_t *angdef = 0,* Int_t *coldef = 1,* Int_t *fondef = 61,* Float_t *sizdef = 1*)
virtual void	**SetTextAlign**(Int_t *align = 11*)
virtual void	**SetTextAngle**(Float_t *tangle = 0*)
virtual void	**SetTextAttributes**( )
virtual void	**SetTextColor**(Color_t *tcolor = 1*)
virtual void	**SetTextFont**(Font_t *tfont = 62*)
virtual void	**SetTextSize**(Float_t *tsize = 1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fTextAngle**	*Text angle*
Float_t	**fTextSize**	*Text size*
Short_t	**fTextAlign**	*Text alignment*
Color_t	**fTextColor**	*Text color index*
Style_t	**fTextFont**	*Text font number*

See Also: TButton, TDialogCanvas, TGXW, TGaxis, TPaveLabel, TPaveText, TPostScript, TStyle, TText

Class Description

```
Text Attributes class
Text attributes are:
 Text alignment
 Text Color
 Text Font
 Text Size
 Text Angle

This class is used (in general by secondary inheritance)
by many other classes (graphics, histograms).

List of the currently supported fonts (screen and PostScript)
Font ID       X11                       Win32 TTF       lfItalic  lfWeight  x
10
     1 : times-medium-i-normal      "Times New Roman"      1           4
     2 : times-bold-r-normal        "Times New Roman"      0           7
```

```
     3 : times-bold-i-normal        "Times New Roman"      1           7
     4 : helvetica-medium-r-normal  "Arial"                0           4
     5 : helvetica-medium-o-normal  "Arial"                1           4
     6 : helvetica-bold-r-normal    "Arial"                0           7
     7 : helvetica-bold-o-normal    "Arial"                1           7
     8 : courier-medium-r-normal    "Courier New"          0           4
     9 : courier-medium-o-normal    "Courier New"          1           4
    10 : courier-bold-r-normal      "Courier New"          0           7
    11 : courier-bold-o-normal      "Courier New"          1           7
    12 : symbol-medium-r-normal     "Symbol"               0           6
    13 : times-medium-r-normal      "Times New Roman"      0           4
    14 :                            "Wingdings"            0           4
```

TAttText( )

```
AttText default constructor
Default text attributes are taking from the current style
```

TAttText(Int_t *align,* Float_t *angle,* Color_t *color,* Style_t *font,* Float_t *tsize*)

```
AttText normal constructor
Text attributes are taking from the argument list
 align : Text alignment = 10*HorizontalAlign + VerticalAlign
         For Horizontal alignment the following convention applies:
            1=left adjusted, 2=centered, 3=rigth adjusted
         For Vertical alignment the following convention applies:
            1=bottom adjusted, 2=centered, 3=top adjusted
         For example align = 11 = Left adjusted and bottom adjusted
                     align = 32 = right adjusted and vertically centered
 angle : Text angle in degrees
 color : text Color Index
 font  : Text font code = 10*fontnumber + precision
          Font numbers must be between 1 and 14
          precision = 1 fast hardware fonts (steps in the size)
          precision = 2 scalable and rotatable hardware fonts
 size  : Character size expressed in percentage of the current pad height
         The textsize in pixels (say charheight) will be:
          charheight = textsize*canvas_height if current pad is horizontal.
          charheight = textsize*canvas_width  if current pad is vertical.
```

~TAttText( )

```
AttText destructor
```

void **Copy**(TAttText &*atttext*)

```
Copy this text attributes to a new atttext
```

void **Modify**( )

```
Change current text attributes if necessary
```

void **Reset**(Option_t **option*)

```
Reset this text attributes to default values
```

void **SaveTextAttributes**(ofstream &*out,* const Text_t **name,* Int_t *alidef,* Float_t *angdef,* Int_t *coldef,* Int_t *fondef,* Float_t *sizdef*)

```
Save text attributes as C++ statement(s) on output stream out
```

void **SetTextAttributes**( )

```
Invoke the DialogCanvas Text attributes
```

Author: René Brun 12/12/94
Last update: 1.01/05 13/06/97 10.38.16 by René Brun

TAttTextCanvas

class **TAttTextCanvas** : public TDialogCanvas

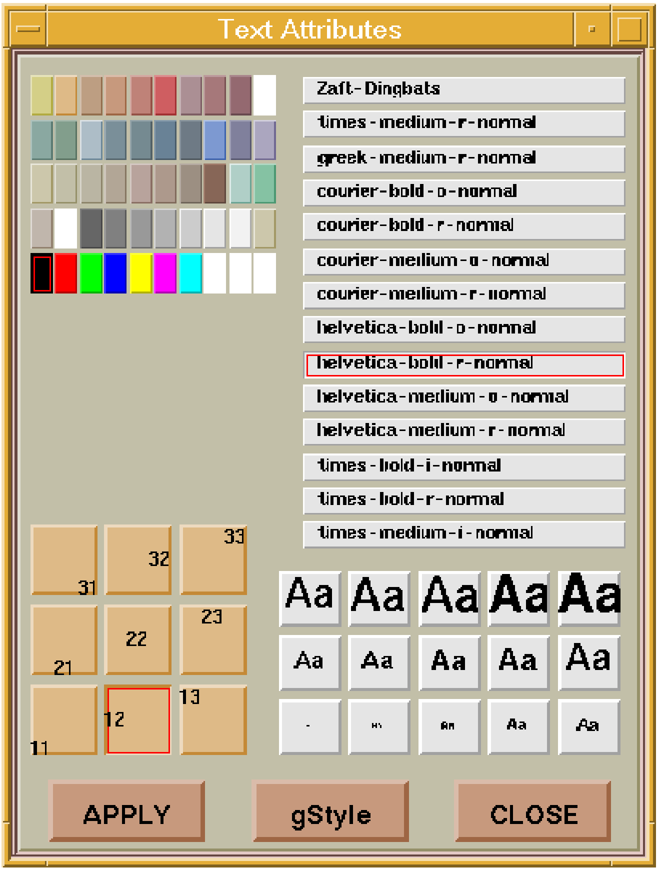
public:

class TAttTextCanvas	**TAttTextCanvas**( )
class TAttTextCanvas	**TAttTextCanvas**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 400,* UInt_t *wh = 600*)
class TAttTextCanvas	**TAttTextCanvas**(class TAttTextCanvas&)
void	**~TAttTextCanvas**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**UpdateTextAttributes**(Int_t *align,* Float_t *angle,* Int_t *col,* Int_t *font,* Float_t *tsize*)

Data Members

Class Description

An AttTextCanvas is a TDialogCanvas specialized to set text attributes.



TAttTextCanvas( ) : **TDialogCanvas**( )

AttTextCanvas default constructor

TAttTextCanvas(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)

AttTextCanvas constructor

~TAttTextCanvas( )

```
AttTextCanvas default destructor
```

void **UpdateTextAttributes**(Int_t *align,* Float_t *angle,* Int_t *col,* Int_t *font,* Float_t *tsize*)

```
Update text attributes
```

Author: René Brun 04/07/96
Last update: 1.02/00 09/07/97 22.00.32 by René Brun

TAxis

class **TAxis** : public TNamed, public TAttAxis

public:

class TAxis	**TAxis**(Int_t *nbins,* Axis_t *xmin,* Axis_t *xmax*)
class TAxis	**TAxis**(Int_t *nbins,* Axis_t* *xbins*)
class TAxis	**TAxis**( )
class TAxis	**TAxis**(class TAxis& *axis*)
void	**~TAxis**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *axis*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawAxis**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Int_t	**FindBin**(Axis_t *x*)
virtual Axis_t	**GetBinCenter**(Int_t *bin*)
virtual Text_t*	**GetBinLabel**(Int_t *bin*)
virtual Axis_t	**GetBinLowEdge**(Int_t *bin*)
virtual Axis_t	**GetBinWidth**(Int_t *bin*)
virtual void	**GetCenter**(Axis_t* *center*)
Int_t	**GetFirst**( )
virtual void	**GetLabel**(Text_t* *label*)
Int_t	**GetLast**( )
virtual void	**GetLowEdge**(Axis_t* *edge*)
Int_t	**GetNbins**( )
const Text_t*	**GetTitle**( )
class TArrayF*	**GetXbins**( )
Axis_t	**GetXmax**( )
Axis_t	**GetXmin**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintAxis**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
virtual void	**Print**(Option_t* *option*)
virtual void	**Set**(Int_t *nbins,* Axis_t* *xbins*)
virtual void	**Set**(Int_t *nbins,* Axis_t *xmin,* Axis_t *xmax*)
virtual void	**SetBinLabel**(Int_t *bin,* Text_t* *label*)
virtual void	**SetLabel**(Text_t* *label*)
virtual void	**SetLimits**(Axis_t *xmin,* Axis_t *xmax*)
virtual void	**SetRange**(Int_t *first = 0,* Int_t *last = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Int_t	**fNbins**	*Number of bins*
Axis_t	**fXmin**	*low edge of first bin*
Axis_t	**fXmax**	*upper edge of last bin*
class TArrayF	**fXbins**	*Bin edges array in X*
Char_t*	**fXlabels**	*!Labels associated to axis*
Int_t	**fFirst**	*first bin to display*
Int_t	**fLast**	*last bin to display*

Class Description

```
This class manages histogram axis. It is referenced by TH1 and TGraph.
To make a graphical representation of an histogram axis, this class
references the TGaxis class.

TAxis supports axis with fixed or variable bin sizes.
Labels may be associated to individual bins.

 see examples of various axis representations drawn by class TGaxis.
```

TAxis(Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*): **TNamed**( ), **TAttAxis**( )

```
Axis constructor for axis with fix bin size
```

TAxis(Int_t *nbins,* Axis_t **xbins*): **TNamed**( ), **TAttAxis**( )

```
Axis constructor for variable bin size
```

void **Copy**(TObject &*obj*)

```
Copy axis structure to another axis
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to an axis
```

void **Draw**(Option_t **option*)

```
Draw an axis
The axis is drawn with the current axis positions, parameters and attributes
```

void **DrawAxis**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)

```
Draw an axis
The axis is drawn with the specified parameters and the stored attributes
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when an axis is clicked with the locator

If Left button clicked on one of the line end points, the axis starting
position is changed interactively.
```

Int_t **FindBin**(Axis_t *x*)

```
Find bin number corresponding to abscissa x
```

Int_t **GetFirst**( )

```
          return first bin on the axis
    - 0 if no range defined
```

Int_t **GetLast**( )

```
          return last bin on the axis
    - fNbins if no range defined
```

Axis_t **GetBinCenter**(Int_t *bin*)

```
Return center of bin
```

Axis_t **GetBinLowEdge**(Int_t *bin*)

```
Return low edge of bin
```

Axis_t **GetBinWidth**(Int_t *bin*)

```
Return bin width
```

void **GetCenter**(Axis_t **center*)

```
Return an array with the center of all bins
```

void **GetLowEdge**(Axis_t **edge*)

```
Return an array with the lod edge of all bins
```

Text_t* **GetBinLabel**(Int_t *bin*)

`Return label associated to bin`

void **GetLabel**(Text_t **label*)

void **Set**(Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*)
`Initialize axis with fix bins`

void **Set**(Int_t *nbins,* Axis_t **xbins*)
`Initialize axis with variable bins`

void **SetLabel**(const Text_t **label*)

void **SetBinLabel**(Int_t *bin,* Text_t **label*)
`Set label associated to bin`

void **SetLimits**(Axis_t *xmin,* Axis_t *xmax*)
`Set the axis limits`

void **SetRange**(Int_t *first,* Int_t *last*)
`Set the viewing range for the axis from bin first to last`

void **Streamer**(TBuffer &*R__b*)
`Stream an object of class TAxis.`

Author: René Brun 12/12/94
Last update: 1.03/01 13/08/97 17.33.43 by René Brun

TBaseClass

class **TBaseClass** : public TDictionary

public:

class TBaseClass	**TBaseClass**((*unknown*) G__BaseClassInfo* info = 0, class TClass* cl = 0)
class TBaseClass	**TBaseClass**(class TBaseClass&)
void	**~TBaseClass**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
class TClass*	**GetClassPointer**( )
Int_t	**GetDelta**( )
virtual const char*	**GetName**( )
virtual const char*	**GetTitle**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsFolder**( )
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class G__BaseClassInfo*	**fInfo**	*pointer to CINT base class info*
class TClass*	**fClassPtr**	*pointer to the base class TClass*
class TClass*	**fClass**	*pointer to class*

Class Description

```
Each class (see TClass) has a linked list of its base class(es).
This class describes one single base class.
The base class info is obtained via the CINT api.
  see class TCint.

The base class information is used a.o. in to find all inherited
methods.
```

TBaseClass(G__BaseClassInfo **info,* TClass **cl*) : **TDictionary**( )

```
Default TBaseClass ctor. TBaseClasses are constructed in TClass
via a call to TCint::CreateListOfBaseClasses().
```

~TBaseClass( )

```
TBaseClass dtor deletes adopted G__BaseClassInfo object.
```

void **Browse**(TBrowser **b*)

```
Called by the browser, to browse a baseclass.
```

TClass* **GetClassPointer**( )

```
Get pointer to the base class TClass.
```

Int_t **GetDelta**( ) const

```
Get offset from "this" to part of base class.
```

const char* **GetName**( ) const

```
Get base class name.
```

const char* **GetTitle**( ) const

```
Get base class description (comment).
```

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )

```
Return hash value for TBaseClass based on its name.
```

Long_t **Property**( ) const

```
Get property description word. For meaning of bits see EProperty.
```

Author: Fons Rademakers 08/02/95
Last update: 1.00/07 24/03/97 11.17.30 by René Brun

TBaseWindow

class **TBaseWindow**

private:

void	**ChangeState**(XEvent* *event*)
void	**ChangeStateHandler**(Widget *w,* XtPointer *clientData,* XEvent* *event,* Boolean*)
void	**WmDeleteWindow**(*Widget, XtPointer, XtPointer*)
void	**WmQuitApp**(*Widget, XtPointer, XtPointer*)

protected:

class TBaseWindow	**TBaseWindow**( )

public:

class TBaseWindow	**TBaseWindow**(class TBaseWindow&)
class TBaseWindow	**TBaseWindow**(char* *name,* ArgList *args = NULL,* Cardinal *argCount = 0*)
void	**~TBaseWindow**( )
virtual void	**AfterRealizeHook**( )
Widget	**BaseWindowWidget**( )
class TClass*	**Class**( )
const char*	**GetTitle**( )
virtual void	**HandleRawEvent**(XEvent* *event*)
virtual void	**HandleWMDeleteMessage**( )
virtual void	**HandleWMQuitMessage**( )
virtual void	**Hide**( )
virtual void	**Iconify**( )
virtual class TClass*	**IsA**( )
Boolean	**IsIconic**( )
Boolean	**IsVisible**( )
virtual void	**Lower**( )
virtual Widget	**MainWindowWidget**( )
virtual void	**Open**( )
virtual void	**Raise**( )
void	**SetIconName**(char* *newTitle*)
void	**SetIconPixmap**(Pixmap *pix*)
void	**SetTitle**(char* *newTitle*)
virtual void	**SetUpWindowProperties**( )
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**StateChanged**(EIconState *newState*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Update**( )

Data Members

private:

char*	**fWindowTitle**
char*	**fIconName**

protected:

EIconState	**fIconState**
EVisibleState	**fVisibleState**
EStackingState	**fStackingState**
Widget	**fBaseWindowWidget**
Widget	**fMainWindowWidget**

See Also: TSimpleWindow

Class Description

```
TBaseWindow

This class defines basic window behaviour. It is a mix-in class
that give classes basic window handling capabilities.
```

TBaseWindow(const char **name,* ArgList *args,* Cardinal *argCount*)

```
Make sure that a TApplication has been created
```

void **HandleRawEvent**(XEvent *)

```
empty
```

void **AfterRealizeHook**( )

```
do nothing, user must override it
```

void **ChangeState**(XEvent **event*)

Author: Fons Rademakers 06/04/95
Last update: 1.00/09 22/08/97 11.38.49 by René Brun

TBasket

class **TBasket** : public TKey

public:

class TBasket	**TBasket**(Text_t* *name,* Text_t* *title,* class TBranch* *branch*)
class TBasket	**TBasket**( )
class TBasket	**TBasket**(class TBasket&)
void	**~TBasket**( )
virtual void	**AdjustSize**(Int_t *newsize*)
class TClass*	**Class**( )
virtual Int_t	**DropBuffers**( )
class TBranch*	**GetBranch**( )
virtual Int_t	**GetBufferSize**( )
virtual Int_t*	**GetEventOffset**( )
virtual Int_t	**GetEventPointer**(Int_t *event*)
virtual Int_t	**GetLast**( )
virtual Int_t	**GetNevBuf**( )
virtual Int_t	**GetNevBufSize**( )
char*	**GetZipBuffer**( )
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasketBuffers**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetNevBufSize**(Int_t *n*)
virtual void	**SetWriteMode**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Update**(Int_t *newlast*)
virtual Int_t	**WriteBuffer**( )

Data Members

protected:

Int_t	**fBufferSize**	*fBuffer length in bytes*
Int_t	**fNevBufSize**	*Length in Int_t of fEventOffset*
Int_t	**fNevBuf**	*Number of events in basket*
Int_t	**fLast**	*Pointer to last used byte in basket*
Bool_t	**fWritingBuffer**	*True when writing buffer*
char*	**fZipBuffer**	*Basket compressed buffer (if compression)*
Int_t*	**fEventOffset**	*Offset of events in fBuffer(TKey)*
class TBranch*	**fBranch**	*Pointer to the basket support branch*

Class Description

```
Manages buffers for branches of a Tree.
See picture in TTree.
```

TBasket( )

```
Default contructor
```

TBasket(const Text_t **name,* const Text_t **title,* TBranch **branch*)

```
Basket normal constructor
```

~TBasket( )

```
Basket destructor
```

void **AdjustSize**(Int_t *newsize*)

```
Increase the size of the current fBuffer up to newsize
```

Int_t **DropBuffers**( )

```
Drop buffers of this basket if it is not the current basket
```

Int_t **GetEventPointer**(Int_t *event*)

```
Get pointer to buffer for internal event
```

void **Print**(Option_t **option*)

```
Print Basket parameters
```

void **ReadBasketBuffers**( )

```
Read basket buffers in memory and cleanup
    Read a basket buffer. Check if buffers of previous ReadBasket
    should not be dropped. Remember, we keep buffers
    in memory up to fMaxVirtualSize.
```

void **Reset**(Option_t **option*)

void **SetWriteMode**( )

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **Update**(Int_t *offset*)

```
Update basket header and EventOffset table
```

Int_t **WriteBuffer**( )

```
Write buffer of this basket on the current file
```

Author: René Brun 19/01/96
Last update: 1.03/04 21/09/97 11.40.11 by René Brun

TBenchmark

class **TBenchmark** : public TNamed

public:

class TBenchmark	**TBenchmark**( )
class TBenchmark	**TBenchmark**(class TBenchmark&)
void	**~TBenchmark**( )
class TClass*	**Class**( )
Int_t	**GetBench**(char* *name*)
Float_t	**GetCpuTime**(char* *name*)
Float_t	**GetRealTime**(char* *name*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(char* *name*)
virtual void	**Reset**( )
virtual void	**Show**(char* *name*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Start**(char* *name*)
virtual void	**Stop**(char* *name*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Summary**( )

Data Members

protected:

Int_t	**fNbench**	*Number of active benchmarks*
Int_t	**fNmax**	*Maximum number of benchmarks initialized*
class TString*	**fNames**	*Names of benchmarks*
Float_t*	**fRealTime**	*Real Time*
Float_t*	**fCpuTime**	*Cpu Time*
class TStopwatch*	**fTimer**	*Timers*

Class Description

```
This class is a ROOT utility to help benchmarking applications

Examples of use of this class are given in the tutorials macros.
```

TBenchmark( ): **TNamed**( )

```
Benchmark default constructor
```

~TBenchmark( )

```
Benchmark default destructor
```

Int_t **GetBench**(const char **name*)

```
Returns index of Benchmark name
```

Float_t **GetCpuTime**(const char **name*)

```
Returns Cpu time used by Benchmark name
```

Float_t **GetRealTime**(const char **name*)

```
Returns Realtime used by Benchmark name
```

void **Print**(const char **name*)

```
Prints parameters of Benchmark name
```

void **Reset**( )

```
Reset all Benchmarks
```

void **Show**(const char **name*)

```
Stops Benchmark name and Prints results
```

void **Start**(const char **name*)

```
Starts Benchmark name
An independent timer (see class TStopwatch) is started.
the name of the benchmark is entered into the list of benchmarks.
Benchmark can be stopped via TBenchmark::Stop
Results can be printed via TBenchmark::Print
TBenchmark::Show can be used to stop benchmark and print results.
If name is an already existing benchmark, existing parameters are reset.
A summary of all benchmarks can be seen via TBenchmark::Summary.
```

void **Stop**(const char **name*)

```
Terminates Benchmark name
```

void **Summary**( )

```
Prints a summary of all benchmarks
```

Last update: 1.00/06 23/03/97 11.01.54 by René Brun

TBox

class **TBox** : public TObject, public TAttLine, public TAttFill

protected:

virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

public:

class TBox	**TBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TBox	**TBox**( )
class TBox	**TBox**(class TBox& *box*)
void	**~TBox**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *box*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
Coord_t	**GetX1**( )
Coord_t	**GetX2**( )
Coord_t	**GetY1**( )
Coord_t	**GetY2**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsBeingResized**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetX1**(Coord_t *x1*)
virtual void	**SetX2**(Coord_t *x2*)
virtual void	**SetY1**(Coord_t *y1*)
virtual void	**SetY2**(Coord_t *y2*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Coord_t	**fX1**	*X of 1st point*
Coord_t	**fY1**	*Y of 1st point*
Coord_t	**fX2**	*X of 2nd point*
Coord_t	**fY2**	*Y of 2nd point*
Bool_t	**fResizing**	*!True if box is being resized*

See Also: TPave, TWbox

Class Description

```
A box is defined by :
- Its bottom left coordinates x1,y1
- Its top right coordinates x2,y2

A box has line attributes (see TAttLine)
and fill area attributes (see TAttFill).
```

TBox( ): **TObject**( ), **TAttLine**( ), **TAttFill**( )

```
Box default constructor
```

TBox(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TObject**( ), **TAttLine**( ), **TAttFill**( )

```
Box standard constructor
```

~TBox( )
```
Box destructor
```

void **Copy**(TObject &*obj*)
```
Copy a Box
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a box
Compute the closest distance of approach from point px,py to the
edges of this box.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)
```
Draw this box with its current attributes
```

void **DrawBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
```
Draw this box with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function is called when a BOX/WBOX/PAD object is clicked.

If the mouse is clicked in one of the 4 corners of the box (PA,PB,PC,PD)
the box is resized with the rubber rectangle.

If the mouse is clicked inside the box, the box is moved.

If the mouse is clicked on the 4 edges (L,R,T,B), the box is rscaled
parallel to this edge (same as Motif window manager).

  PA                  T                       PB
   +--------------------------------------------+
   |                                            |
   |                                            |
   |                                            |
  L|                 INSIDE                     |R
   |                                            |
   |                                            |
   |                                            |
   |                                            |
   +--------------------------------------------+
  PD                  B                      PC
```

void **ls**(Option_t **option*)
```
List this box with its attributes
```

void **Paint**(Option_t **option*)
```
Paint this box with its current attributes
```

void **PaintBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
```
Draw this box with new coordinates
```

void **Print**(Option_t **option*)
```
Dump this box with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 12/12/94
Last update: 1.02/00 09/07/97 22.16.23 by René Brun

TBranch

class **TBranch** : public TNamed

public:

class TBranch	**TBranch**( )
class TBranch	**TBranch**(Text_t* *name,* void* *address,* Text_t* *leaflist,* Int_t *basketsize = 32000,* Int_t *compress = -1*)
class TBranch	**TBranch**(class TBranch&)
void	**~TBranch**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**DropBaskets**( )
virtual Int_t	**Fill**( )
char*	**GetAddress**( )
class TBasket*	**GetBasket**(Int_t *basket*)
Seek_t	**GetBasketSeek**(Int_t *basket*)
virtual Int_t	**GetBasketSize**( )
virtual Int_t	**GetCompressionLevel**( )
virtual Stat_t	**GetEntries**( )
virtual Int_t	**GetEvent**(Int_t *event = 0*)
Int_t	**GetEventNumber**( )
virtual Int_t	**GetEventOffsetLen**( )
class TLeaf*	**GetLeaf**(Text_t* *name*)
class TObjArray*	**GetListOfBaskets**( )
class TObjArray*	**GetListOfBranches**( )
class TObjArray*	**GetListOfLeaves**( )
virtual Int_t	**GetMaxBaskets**( )
virtual Int_t	**GetNleaves**( )
Int_t	**GetOffset**( )
Int_t	**GetReadBasket**( )
Int_t	**GetReadEvent**( )
virtual Int_t	**GetRow**(Int_t *row*)
virtual Stat_t	**GetTotBytes**( )
class TTree*	**GetTree**( )
Int_t	**GetWriteBasket**( )
virtual Stat_t	**GetZipBytes**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
Int_t	**LocateBinary**(Int_t *value,* Int_t* *array,* Int_t *nelem*)
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetAddress**(void* *add*)
virtual void	**SetEventOffsetLen**(Int_t *len*)
virtual void	**SetOffset**(Int_t *offset = 0*)
virtual void	**SetTree**(class TTree* *tree*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fCompress**	*(=1 branch is compressed, 0 otherwise*)
Int_t	**fBasketSize**	*Initial Size of Basket Buffer*
Int_t	**fEventOffsetLen**	*Initial Length of fEventOffset table in the basket buffers*

Int_t	**fMaxBaskets**	*Maximum number of Baskets so far*
Int_t	**fWriteBasket**	*Last basket number written*
Int_t	**fReadBasket**	*Current basket number when reading*
Int_t	**fReadEvent**	*Current event number when reading*
Int_t	**fEventNumber**	*Current event number (last one filled in this branch)*
Int_t	**fOffset**	*Offset of this branch*
Int_t	**fNleaves**	*Number of leaves*
Stat_t	**fEntries**	*Number of entries*
Stat_t	**fTotBytes** *compression*	*Total number of bytes in all leaves before*
Stat_t	**fZipBytes**	*Total number of bytes in all leaves after compression*
class TObjArray	**fBranches**	*List of Branches of this branch*
class TObjArray	**fLeaves**	*List of leaves of this branch*
class TObjArray	**fBaskets**	*List of baskets of this branch*
Int_t*	**fBasketEvent**	*Table of first event in eack basket*
Seek_t*	**fBasketSeek**	*Addresses of baskets on file*
class TTree*	**fTree**	*Pointer to Tree header*
char*	**fAddress**	*Address of 1st leaf (variable or object)*

See Also: TBranchClones, TBranchObject

Class Description

```
A TTree is a list of TBranches                                       //

A TBranch supports:
- The list of TLeaf describing this branch.
- The list of TBasket (branch buffers).

    See TBranch structure in TTree.

See also specialized branches:
  TBranchObject in case the branch is one object
  TBranchClones in case the branch is an array of clone objects
```

TBranch( ): **TNamed**( )

```
Default constructor for Branch
```

TBranch(const Text_t **name,* void **address,* const Text_t **leaflist,* Int_t *basketsize,* Int_t *compress*):**TNamed**(*name, leaflist*)

```
Create a Branch
    * address is the address of the first item of a structure
      or the address of a pointer to an object (see example).
    * leaflist is the concatenation of all the variable names and types
      separated by a colon character :
      The variable name and the variable type are separated by a slash (/).
      The variable type may be 0,1 or 2 characters. If no type is given,
      the type of the variable is assumed to be the same as the previous
      variable. If the first variable does not have a type, it is assumed
      of type F by default. The list of currently supported types is given
below:
         - C : a character string terminated by the 0 character
         - B : an 8 bit signed integer (Char_t)
         - b : an 8 bit unsigned integer (UChar_t)
         - S : a 16 bit signed integer (Short_t)
         - s : a 16 bit unsigned integer (UShort_t)
         - I : a 32 bit signed integer (Int_t)
         - i : a 32 bit unsigned integer (UInt_t)
         - F : a 32 bit floating point (Float_t)
         - D : a 64 bit floating point (Double_t)

      By default, a variable will be copied to the buffer with the number of
```

```
bytes specified in the type descriptor character. However, if the type
consists of 2 characters, the second character is an integer that
specifies the number of bytes to be used when copying the variable
to the output buffer. Example:
    X         ; variable X, type Float_t
    Y/I       : variable Y, type Int_t
    Y/I2      ; variable Y, type Int_t converted to a 16 bits integer
```

```
See an example of a Branch definition in the TTree constructor.

Note that in case the data type is an object, this branch can contain
only this object.
```

~**TBranch**( )

```
Default destructor for a Branch
```

void **DropBaskets**( )

```
Loop on all branch baskets. Drop all except readbasket
```

Int_t **Fill**( )

```
Loop on all leaves of this branch to fill Basket buffer
```

TBasket* **GetBasket**(Int_t *basketnumber*)

```
Return pointer to basket basketnumber in this Branch
```

Seek_t **GetBasketSeek**(Int_t *basketnumber*)

```
Return address of basket in the file
```

Int_t **GetEvent**(Int_t *event*)

```
Read all leaves of event and return total number of bytes
```

TLeaf* **GetLeaf**(Text_t **name*)

```
Return pointer to the 1st Leaf named name in thisBranch
```

Int_t **GetRow**(Int_t *row*)

```
Return all elements of one row unpacked in internal array fValues
```

Bool_t **IsFolder**( )

```
Return TRUE if more than one leaf, FALSE otherwise
```

Int_t **LocateBinary**(Int_t *value,* Int_t **array,* Int_t *nbins*)

```
Binary search in an array of nbins to locate value
array is supposed  to be sorted prior to this call.
If match is found, function returns position of element.
If no match found, function gives nearest element smaller than value.
```

void **Print**(Option_t **option*)

```
Print TBranch parameters
```

void **ReadBasket**(TBuffer &*b*)

```
Loop on all leaves of this branch to read Basket buffer
```

void **Reset**(Option_t **option*)

```
Reset a Branch
 Existing buffers are deleted
 Entries, max and min are reset
```

void **SetAddress**(void **add*)

```
Set address of this branch
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 12/01/96
Last update: 1.03/06 25/10/97 13.58.31 by René Brun

TBranchClones

class **TBranchClones** : public TBranch

public:

class TBranchClones	**TBranchClones**(Text_t* *name,* class TClonesArray* *list,* Int_t *basketsize = 32000,* Int_t *compress = -1*)
class TBranchClones	**TBranchClones**(class TBranchClones&)
class TBranchClones	**TBranchClones**( )
void	**~TBranchClones**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual Int_t	**Fill**( )
virtual Int_t	**GetEvent**(Int_t *event = 0*)
class TClonesArray*	**GetList**( )
virtual Int_t	**GetN**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetAddress**(void* *add*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TClonesArray*	**fList**	*Pointer to the clonesarray*
Int_t	**fRead**	*flag = 1 if clonesarray has been read*
Int_t	**fN**	*Number of elements in ClonesArray*
Int_t	**fNdataMax**	*Maximum value of fN*
class TString	**fClassName**	*name of the class of the objets in the ClonesArray*
class TBranch*	**fBranchCount**	*Branch with clones count*

Class Description

```
A Branch for the case of an array of clone objects                   //
See TTree.                                                           //
```

TBranchClones( ): **TBranch**( )

```
Default constructor for BranchClones
```

TBranchClones(const Text_t **name,* TClonesArray **list,* Int_t *basketsize,* Int_t *compress*):**TBranch**( )

```
Create a BranchClones
```

~TBranchClones( )

```
Default destructor for a BranchClones
```

Int_t **Fill**( )

```
Loop on all BranchClone of this BranchClones to fill Basket buffer
```

Int_t **GetEvent**(Int_t *event*)

```
Read all branches of a BranchClones and return total number of bytes
```

void **Print**(Option_t **option*)

```
Print TBranch parameters
```

void **Reset**(Option_t **option*)

```
Reset a Branch
 Existing buffers are deleted
```

```
Entries, max and min are reset
```

void **SetAddress**(void **add*)

```
Set address of this branch
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 11/02/96
Last update: 30/10/97 14.46.29 by René Brun

TBranchObject

class **TBranchObject** : public TBranch

public:

class TBranchObject	**TBranchObject**(Text_t* *name,* Text_t* *classname,* void* *addobj,* Int_t *basketsize = 32000,* Int_t *splitlevel = 0,* Int_t *compress = -1*)
class TBranchObject	**TBranchObject**( )
class TBranchObject	**TBranchObject**(class TBranchObject&)
void	**~TBranchObject**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual Int_t	**Fill**( )
virtual Int_t	**GetEvent**(Int_t *event = 0*)
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetAddress**(void* *addobj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TString	**fClassName**	*Class name of referenced object*

Class Description

```
TBranchObject

A Branch for the case of an object                                        //
```

TBranchObject( ): **TBranch**( )

```
Default constructor for BranchObject
```

TBranchObject(const Text_t **name,* const Text_t **classname,* void **addobj,* Int_t *basketsize,* Int_t *splitlevel,* Int_t *compress*):**TBranch**( )

```
Create a BranchObject
```

~TBranchObject( )

```
Default destructor for a BranchObject
```

Int_t **Fill**( )

```
Loop on all leaves of this branch to fill Basket buffer
```

Int_t **GetEvent**(Int_t *event*)

```
Read all branches of a BranchObject and return total number of bytes
If event = 0 take current event number + 1
If event < 0 reset event number to 0
```

Bool_t **IsFolder**( )

```
Return TRUE if more than one leaf, FALSE otherwise
```

void **Print**(Option_t **option*)

```
Print TBranch parameters
```

void **Reset**(Option_t **option*)

```
Reset a Branch
 Existing buffers are deleted
 Entries, max and min are reset
```

void **SetAddress**(void **add*)

```
Set address of this branch
```

Author: René Brun 11/02/96
Last update: 30/10/97 14.44.58 by René Brun

TBRIK

class **TBRIK** : public TShape

public:

class TBRIK	**TBRIK**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dx,* Float_t *dy,* Float_t *dz*)
class TBRIK	**TBRIK**( )
class TBRIK	**TBRIK**(class TBRIK&)
void	**~TBRIK**( )
class TClass*	**Class**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintGLPoints**(Float_t* *buff*)
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

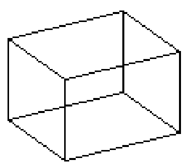
Data Members

protected:

Float_t	**fDx**	*half length in x*
Float_t	**fDy**	*half length in y*
Float_t	**fDz**	*half length in z*

See Also: TGTRA, TPARA, TSPHE, TTRAP, TTRD1, TTRD2

Class Description



```
BRIK is a box with faces perpendicular to the axes. It has 6 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dx         half-length of the box along the x-axis
  - dy         half-length of the box along the y-axis
  - dz         half-length of the box along the z-axis
```

TBRIK( )

```
BRIK shape default constructor
```

TBRIK(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dx,* Float_t *dy,* Float_t *dz*): **TShape**(*name, title, material*)

```
BRIK shape normal constructor
```

~TBRIK( )

```
BRIK shape default destructor
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a BRIK
```

```
Compute the closest distance of approach from point px,py to each corner
point of the BRIK.
```

void **Paint**(Option_t **option*)

```
Paint this 3-D shape with its current attributes
```

void **PaintGLPoints**(Float_t **vertex*)

```
Paint BRIK via OpenGL
```

void **SetPoints**(Float_t **buff*)

```
Create BRIK points
```

void **Sizeof3D**( ) const

```
Return total X3D size of this shape with its attributes
```

Author: Nenad Buncic 17/09/95
Last update: 1.01/04 30/05/97 13.49.42 by Valery Fine(fine@vxcern.cern.ch)

TBrowser

class **TBrowser** : public TNamed

public:

class TBrowser	**TBrowser**(char* *name,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TBrowser	**TBrowser**(char* *name,* class TObject* *obj,* char* *title,* UInt_t *width = 640,* UInt_t *height = 400*)
class TBrowser	**TBrowser**(char* *name = Browser,* char* *title = Browser,* UInt_t *width = 640,* UInt_t *height = 400*)
class TBrowser	**TBrowser**(char* *name,* class TObject* *obj,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TBrowser	**TBrowser**(class TBrowser&)
void	**~TBrowser**( )
void	**Add**(class TObject* *obj,* char* *name = 0*)
class TClass*	**Class**( )
void	**Create**(class TObject* *obj = 0*)
class TBrowserImp*	**GetBrowserImp**( )
class TContextMenu*	**GetContextMenu**( )
Bool_t	**GetRefreshFlag**( )
void	**Iconify**( )
virtual class TClass*	**IsA**( )
virtual void	**RecursiveRemove**(class TObject* *obj*)
void	**Refresh**( )
void	**SetRefreshFlag**(Bool_t *flag*)
void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

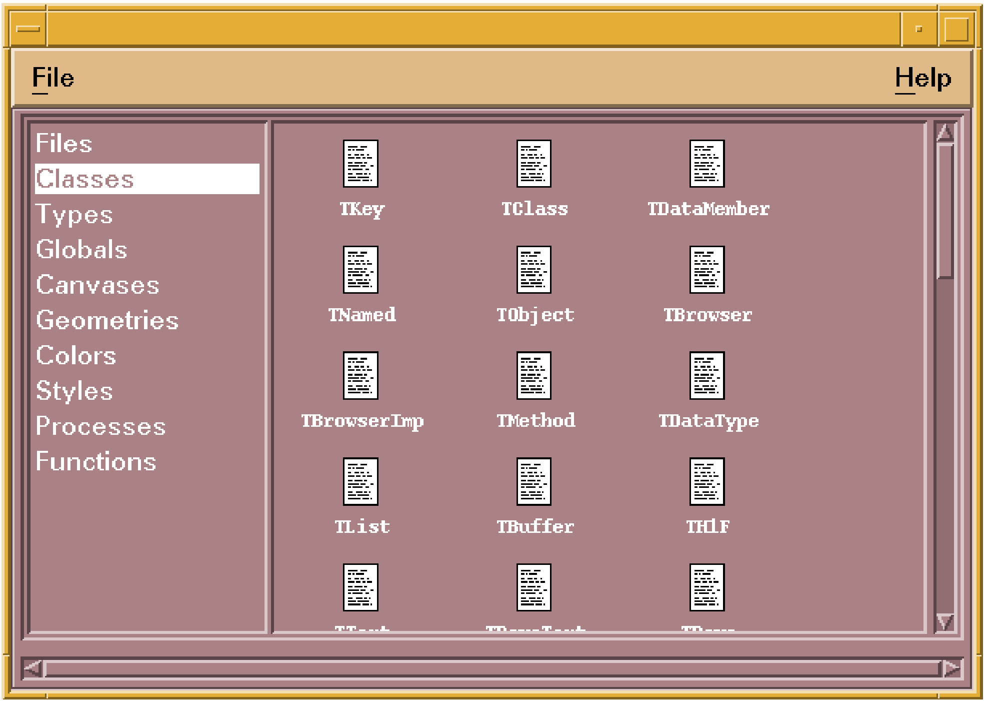
class TBrowserImp*	**fImp**	*Window system specific browser implementation*
class TBrowserTimer*	**fTimer**	*Browser's timer*

protected:

class TContextMenu*	**fContextMenu**	*Context menu pointer*
Bool_t	**fNeedRefresh**	*True if the browser needs refresh*

Class Description

```
Using a TBrowser one can browse all ROOT objects. It shows in a list
on the left side of the window all browsable ROOT classes. Selecting
one of the classes displays, in the iconbox on the right side, all
objects in the class. Selecting one of the objects in the iconbox,
will place all browsable objects in a new list and draws the
contents of the selected class in the iconbox. And so on....
```



TBrowser(const char **name,* const char **title,* UInt_t *width,* UInt_t *height*): **TNamed**(*name, title*)

```
Create a new browser with a name and title and a width and height.
```

TBrowser(const char **name,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*): **TNamed**(*name, title*)

```
Create a new browser with a name and title and a width and height.
```

TBrowser(const char **name,* TObject **obj,* const char **title,* UInt_t *width,* UInt_t *height*): **TNamed**(*name, title*)

```
Create a new browser with a name and title and a width and height for TObject
*obj
```

TBrowser(const char **name,* TObject **obj,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*): **TNamed**(*name, title*)

```
Create a new browser with a name and title and a width and height for TObject
*obj
```

~**TBrowser**( )

```
Delete the browser.
```

void **Refresh**( )

Author: Fons Rademakers 25/10/95
Last update: 1.00/06 23/03/97 11.17.19 by René Brun

TBrowserImp

class **TBrowserImp**

public:

class TBrowserImp	**TBrowserImp**(class TBrowser* *b,* char* *title,* UInt_t *width,* UInt_t *height*)
class TBrowserImp	**TBrowserImp**(class TBrowser* *b = 0*)
class TBrowserImp	**TBrowserImp**(class TBrowser* *b,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TBrowserImp	**TBrowserImp**(class TBrowserImp&)
void	**~TBrowserImp**( )
virtual void	**Add**(class TObject* *obj,* char* *name = 0*)
class TBrowser*	**Browser**( )
class TClass*	**Class**( )
virtual void	**Iconify**( )
virtual class TClass*	**IsA**( )
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**Refresh**( )
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Bool_t	**fShowCycles**	
class TBrowser*	**fBrowser**	*TBrowser associated with this implementation*

See Also: TMotifBrowser

Class Description

```
TBrowserImp

ABC describing GUI independent browser implementation protocol.
```

Author: Fons Rademakers 15/11/95
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TBtree

class **TBtree** : public TSeqCollection

private:

void	**Init**(Int_t *i*)
void	**RootIsEmpty**( )
void	**RootIsFull**( )

protected:

void	**DecrNofKeys**( )
Int_t	**IdxAdd**(class TObject& *obj*)
void	**IncrNofKeys**( )

public:

class TBtree	**TBtree**(class TBtree&)
class TBtree	**TBtree**(Int_t *ordern = 3*)
void	**~TBtree**( )
virtual void	**Add**(class TObject* *obj*)
virtual void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
virtual void	**AddAt**(class TObject* *obj,* Int_t *idx*)
virtual void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj*)
virtual void	**AddLast**(class TObject* *obj*)
virtual class TObject*	**After**(class TObject* *obj*)
virtual class TObject*	**At**(Int_t *idx*)
virtual class TObject*	**Before**(class TObject* *obj*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Delete**(Option_t* *option*)
virtual class TObject*	**FindObject**(Text_t* *name*)
virtual class TObject*	**FindObject**(class TObject* *obj*)
virtual class TObject*	**First**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Last**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir = kIterForward*)
class TObject*	**operator[]**(Int_t *i*)
Int_t	**Order**( )
Int_t	**Rank**(class TObject* *obj*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TBtNode*	**fRoot**	*root node of btree*
Int_t	**fOrder**	*the order of the tree (should be > 2)*
Int_t	**fOrder2**	*order*2+1 (assumes a memory access is*
Int_t	**fInnerLowWaterMark**	*inner node low water mark*
Int_t	**fLeafLowWaterMark**	*leaf low water mark*
Int_t	**fInnerMaxIndex**	*maximum inner node index*
Int_t	**fLeafMaxIndex**	*maximum leaf index*

Class Description

```
TBtree

B-tree class. TBtree inherits from the TSeqCollection ABC.
```

B-tree Implementation notes

This implements B-trees with several refinements. Most of them can be found in Knuth Vol 3, but some were developed to adapt to restrictions imposed by C++. First, a restatement of Knuth's properties that a B-tree must satisfy, assuming we make the enhancement he suggests in the paragraph at the bottom of page 476. Instead of storing null pointers to non-existent nodes (which Knuth calls the leaves) we utilize the space to store keys. Therefore, what Knuth calls level (l-1) is the bottom of our tree, and we call the nodes at this level LeafNodes. Other nodes are called InnerNodes. The other enhancement we have adopted is in the paragraph at the bottom of page 477: overflow control.

The following are modifications of Knuth's properties on page 478:

1. Every InnerNode has at most Order keys, and at most Order+1 sub-trees.
2. Every LeafNode has at most 2*(Order+1) keys.
3. An InnerNode with k keys has k+1 sub-trees.
4. Every InnerNode that is not the root has at least InnerLowWaterMark keys.
5. Every LeafNode that is not the root has at least LeafLowWaterMark keys.
6. If the root is a LeafNode, it has at least one key.
7. If the root is an InnerNode, it has at least one key and two sub-trees.
8. All LeafNodes are the same distance from the root as all the other LeafNodes.
9. For InnerNode n with key n[i].key, then sub-tree n[i-1].tree contains all keys < n[i].key, and sub-tree n[i].tree contains all keys >= n[i].key.
10. Order is at least 3.

The values of InnerLowWaterMark and LeafLowWaterMark may actually be set by the user when the tree is initialized, but currently they are set automatically to:

```
InnerLowWaterMark = ceiling(Order/2)
LeafLowWaterMark  = Order - 1
```

If the tree is only filled, then all the nodes will be at least 2/3 full. They will almost all be exactly 2/3 full if the elements are added to the tree in order (either increasing or decreasing). [Knuth says McCreight's experiments showed almost 100% memory utilization. I don't see how that can be given the algorithms that Knuth gives. McCreight must have used a different scheme for balancing. [No, he used a different scheme for splitting: he did a two-way split instead of the three way split as we do here. Which means that McCreight does better on insertion of ordered data, but we should do better on insertion of random data.]]

It must also be noted that B-trees were designed for DISK access algorithms, not necessarily in-memory sorting, as we intend it to be used here. However, if the order is kept small (< 6?) any inefficiency is negligible for in-memory sorting. Knuth points out that balanced trees are actually preferable for memory sorting. I'm not sure that I believe this, but it's interesting. Also, deleting elements from balanced binary trees, being beyond the scope of Knuth's book (p. 465), is beyond my scope. B-trees are good enough.

A B-tree is declared to be of a certain ORDER (3 by default). This number determines the number of keys contained in any interior node of the tree. Each interior node will contain ORDER keys, and therefore ORDER+1 pointers to sub-trees. The keys are numbered and indexed 1 to ORDER while the pointers are numbered and indexed 0 to ORDER. The 0th ptr points to the sub-tree of all elements that are less than key[1]. Ptr[1] points to the sub-tree that contains all the elements greater than key[1] and less than key[2]. etc. The array of pointers and keys is allocated as ORDER+1 pairs of keys and nodes, meaning that one key field (key[0]) is not used and therefore wasted. Given that the number of interior nodes is small, that this waste allows fewer cases of special code, and that it is useful in certain of the methods, it was felt to be a worthwhile waste.

The size of the exterior nodes (leaf nodes) does not need to be related to the size of the interior nodes at all. Since leaf nodes contain only keys, they may be as large or small as we like independent of the size of the interior nodes. For no particular reason other than it seems like a good idea, we will allocate 2*(ORDER+1) keys in each leaf node, and they will be numbered and indexed from 0 to 2*ORDER+1. It does have the advantage of keeping the size of the leaf and interior arrays the same, so that if we find allocation and de-allocation of these arrays expensive, we can modify their allocation to use a garbage ring, or something.

Both of these numbers will be run-time constants associated with each tree (each tree at run-time can be of a different order). The variable "order" is the order of the tree, and the inclusive upper limit on the indices of the keys in the interior nodes. The variable "order2" is the inclusive upper limit on the indices of the leaf nodes, and is designed

```
(1) to keep the sizes of the two kinds of nodes the same;
(2) to keep the expressions involving the arrays of keys looking
    somewhat the same:   lower limit        upper limit
      for inner nodes:        1                order
      for leaf  nodes:        0                order2
    Remember that index 0 of the inner nodes is special.
```

Currently, order2 = 2*(order+1).

```
Picture: (also see Knuth Vol 3 pg 478)

        +--+--+--+--+--+--...
        |  |  |  |  |  |
parent--->|  |     |     |
        |  |     |     |
        +*-+*-+*-+--+--+--...
         |  |  |
    +----+  |  +-----+
    |       +-----+  |
    V             |  V
    +----------+  |  +----------+
    |          |  |  |          |
this->|          |  |  |          |<--sib
    +----------+  |  +----------+
                  V
                 data
```

It is conceptually VERY convenient to think of the data as being the very first element of the sib node. Any primitive that tells sib to perform some action on n nodes should include this 'hidden' element. For InnerNodes, the hidden element has (physical) index 0 in the array, and in LeafNodes, the hidden element has (virtual) index -1 in the array. Therefore, there are two 'size' primitives for nodes:

```
Psize       - the physical size: how many elements are contained in the
            array in the node.
Vsize       - the 'virtual' size; if the node is pointed to by
            element 0 of the parent node, then Vsize == Psize;
            otherwise the element in the parent item that points to this
            node 'belongs' to this node, and Vsize == Psize+1;
```

Parent nodes are always InnerNodes.

These are the primitive operations on Nodes:

```
Append(elt)     - adds an element to the end of the array of elements in a
                node.  It must never be called where appending the element
                would fill the node.
Split()         - divide a node in two, and create two new nodes.
SplitWith(sib)  - create a third node between this node and the sib node,
                divvying up the elements of their arrays.
PushLeft(n)     - move n elements into the left sibling
PushRight(n)    - move n elements into the right sibling
BalanceWithRight() - even up the number of elements in the two nodes.
BalanceWithLeft()  - ditto
```

To allow this implementation of btrees to also be an implementation of sorted arrays/lists, the overhead is included to allow O(log n) access of elements by their rank ('give me the 5th largest element'). Therefore, each Item keeps track of the number of keys in and below it in the tree (remember, each item's tree is all keys to the RIGHT of the item's own key).

```
[ [ < 0 1 2 3 > 4 < 5 6 7 > 8 < 9 10 11 12 > ] 13 [ < 14 15 16 > 17 < 18 19 20
> ] ]
4  1 1 1 1   4   1 1 1   5   1  1  1  1      7  3   1  1  1    4    1  1  1
```

TBtree(int *order*)

    Create a B-tree of certain order (by default 3).

~TBtree( )

    Delete B-tree.

void **Add**(TObject **obj*)

    Add object to B-tree.

TObject* **After**(TObject **obj*) const

    Cannot use this method since B-tree decides order.

TObject* **Before**(TObject **obj*) const

    May not use this method since B-tree decides order.

void **Clear**(Option_t **opt*)

    Remove all objects from B-tree. Does NOT delete objects.

void **Delete**(Option_t **option*)

    Remove all objects from B-tree AND delete all heap based objects.

TObject* **FindObject**(const Text_t **name*) const

    Find object using its name (see object's GetName()). Requires sequential
    search of complete tree till object is found.

TObject* **FindObject**(TObject **obj*) const

    Find object using the objects Compare() member function.

Int_t **IdxAdd**(const TObject &*obj*)

    Add object and return its index in the tree.

void **Init**(Int_t *order*)

    Initialize a B-tree.

TIterator* **MakeIterator**(Bool_t *dir*) const

    Returns a B-tree iterator.

Int_t **Rank**(const TObject **obj*) const

    Returns the rank of the object in the tree.

TObject* **Remove**(TObject **obj*)

    Remove an object from the tree.

void **RootIsFull**( )

    The root of the tree is full. Create an InnerNode that
    points to it, and then inform the InnerNode that it is full.

void **RootIsEmpty**( )

    If root is empty clean up its space.

void **Streamer**(TBuffer &*b*)

    Stream all objects in the btree to or from the I/O buffer.

Author: Fons Rademakers 10/10/95
Last update: 1.03/04 29/09/97 19.05.57 by Fons Rademakers

TBtreeIter

class **TBtreeIter** : public TIterator

private:

class TBtreeIter	**TBtreeIter**( )

public:

class TBtreeIter	**TBtreeIter**(class TBtree* *t,* Bool_t *dir* = *kIterForward*)
class TBtreeIter	**TBtreeIter**(class TBtreeIter&)
void	**~TBtreeIter**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
virtual void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

const class TBtree*	**fTree**	*btree being iterated*
Int_t	**fCursor**	*current position in btree*
Bool_t	**fDirection**	*iteration direction*

Class Description

```
TBtree

B-tree class. TBtree inherits from the TSeqCollection ABC.
```

B-tree Implementation notes

This implements B-trees with several refinements. Most of them can be found in Knuth Vol 3, but some were developed to adapt to restrictions imposed by C++. First, a restatement of Knuth's properties that a B-tree must satisfy, assuming we make the enhancement he suggests in the paragraph at the bottom of page 476. Instead of storing null pointers to non-existent nodes (which Knuth calls the leaves) we utilize the space to store keys. Therefore, what Knuth calls level (l-1) is the bottom of our tree, and we call the nodes at this level LeafNodes. Other nodes are called InnerNodes. The other enhancement we have adopted is in the paragraph at the bottom of page 477: overflow control.

The following are modifications of Knuth's properties on page 478:

1. Every InnerNode has at most Order keys, and at most Order+1 sub-trees.
2. Every LeafNode has at most 2*(Order+1) keys.
3. An InnerNode with k keys has k+1 sub-trees.
4. Every InnerNode that is not the root has at least InnerLowWaterMark keys.
5. Every LeafNode that is not the root has at least LeafLowWaterMark keys.
6. If the root is a LeafNode, it has at least one key.
7. If the root is an InnerNode, it has at least one key and two sub-trees.
8. All LeafNodes are the same distance from the root as all the other LeafNodes.
9. For InnerNode n with key n[i].key, then sub-tree n[i-1].tree contains all keys < n[i].key, and sub-tree n[i].tree contains all keys >= n[i].key.
10. Order is at least 3.

The values of InnerLowWaterMark and LeafLowWaterMark may actually be set by the user when the tree is initialized, but currently they are set automatically to:

```
InnerLowWaterMark = ceiling(Order/2)
LeafLowWaterMark  = Order - 1
```

If the tree is only filled, then all the nodes will be at least 2/3 full. They will almost all be exactly 2/3 full if the elements are added to the tree in order (either increasing or decreasing). [Knuth says McCreight's experiments showed almost 100% memory utilization. I don't see how that can be given the algorithms that Knuth gives. McCreight must have used a different scheme for balancing. [No, he used a different scheme for splitting: he did a two-way split instead of the three way split as we do here. Which means that McCreight does better on insertion of ordered data, but we should do better on insertion of random data.]]

It must also be noted that B-trees were designed for DISK access algorithms, not necessarily in-memory sorting, as we intend it to be used here. However, if the order is kept small (< 6?) any inefficiency is negligible for in-memory sorting. Knuth points out that balanced trees are actually preferable for memory sorting. I'm not sure that I believe this, but it's interesting. Also, deleting elements from balanced binary trees, being beyond the scope of Knuth's book (p. 465), is beyond my scope. B-trees are good enough.

A B-tree is declared to be of a certain ORDER (3 by default). This number determines the number of keys contained in any interior node of the tree. Each interior node will contain ORDER keys, and therefore ORDER+1 pointers to sub-trees. The keys are numbered and indexed 1 to ORDER while the pointers are numbered and indexed 0 to ORDER. The 0th ptr points to the sub-tree of all elements that are less than key[1]. Ptr[1] points to the sub-tree that contains all the elements greater than key[1] and less than key[2]. etc. The array of pointers and keys is allocated as ORDER+1 pairs of keys and nodes, meaning that one key field (key[0]) is not used and therefore wasted. Given that the number of interior nodes is small, that this waste allows fewer cases of special code, and that it is useful in certain of the methods, it was felt to be a worthwhile waste.

The size of the exterior nodes (leaf nodes) does not need to be related to the size of the interior nodes at all. Since leaf nodes contain only keys, they may be as large or small as we like independent of the size of the interior nodes. For no particular reason other than it seems like a good idea, we will allocate 2*(ORDER+1) keys in each leaf node, and they will be numbered and indexed from 0 to 2*ORDER+1. It does have the advantage of keeping the size of the leaf and interior arrays the same, so that if we find allocation and de-allocation of these arrays expensive, we can modify their allocation to use a garbage ring, or something.

Both of these numbers will be run-time constants associated with each tree (each tree at run-time can be of a different order). The variable "order" is the order of the tree, and the inclusive upper limit on the indices of the keys in the interior nodes. The variable "order2" is the inclusive upper limit on the indices of the leaf nodes, and is designed

```
(1) to keep the sizes of the two kinds of nodes the same;
(2) to keep the expressions involving the arrays of keys looking
    somewhat the same:   lower limit        upper limit
      for inner nodes:        1                order
      for leaf  nodes:        0                order2
    Remember that index 0 of the inner nodes is special.
```

Currently, order2 = 2*(order+1).

```
Picture: (also see Knuth Vol 3 pg 478)

        +--+--+--+--+--+--...
        |  |  |  |  |  |
parent--->|  |     |     |
        |  |     |     |
        +*-+*-+*-+--+--+--...
         |  |  |
    +----+  |  +-----+
    |       +-----+  |
    V             |  V
    +----------+  |  +----------+
    |          |  |  |          |
this->|          |  |  |          |<--sib
    +----------+  |  +----------+
                  V
                 data
```

It is conceptually VERY convenient to think of the data as being the very first element of the sib node. Any primitive that tells sib to perform some action on n nodes should include this 'hidden' element. For InnerNodes, the hidden element has (physical) index 0 in the array, and in LeafNodes, the hidden element has (virtual) index -1 in the array. Therefore, there are two 'size' primitives for nodes:

```
Psize       - the physical size: how many elements are contained in the
            array in the node.
Vsize       - the 'virtual' size; if the node is pointed to by
            element 0 of the parent node, then Vsize == Psize;
            otherwise the element in the parent item that points to this
            node 'belongs' to this node, and Vsize == Psize+1;
```

Parent nodes are always InnerNodes.

These are the primitive operations on Nodes:

```
Append(elt)       - adds an element to the end of the array of elements in a
                  node.  It must never be called where appending the element
                  would fill the node.
Split()           - divide a node in two, and create two new nodes.
SplitWith(sib)    - create a third node between this node and the sib node,
                  divvying up the elements of their arrays.
PushLeft(n)       - move n elements into the left sibling
PushRight(n)      - move n elements into the right sibling
BalanceWithRight() - even up the number of elements in the two nodes.
BalanceWithLeft()  - ditto
```

To allow this implementation of btrees to also be an implementation of sorted arrays/lists, the overhead is included to allow O(log n) access of elements by their rank ('give me the 5th largest element'). Therefore, each Item keeps track of the number of keys in and below it in the tree (remember, each item's tree is all keys to the RIGHT of the item's own key).

```
[ [ < 0 1 2 3 > 4 < 5 6 7 > 8 < 9 10 11 12 > ] 13 [ < 14 15 16 > 17 < 18 19 20
> ] ]
4  1 1 1 1   4   1 1 1   5   1  1  1  1      7  3   1  1  1    4    1  1  1
```

TBtreeIter(const TBtree **t,* Bool_t *dir*): **fTree**(*t*), **fCursor**(0), **fDirection**(dir)

```
Create a B-tree iterator.
```

void **Reset**( )

```
Reset the B-tree iterator.
```

TObject* **Next**( )

```
Get next object from B-tree. Returns 0 when no more objects in tree.
```

Author: Fons Rademakers 10/10/95
Last update: 1.03/04 29/09/97 19.05.57 by Fons Rademakers

TBuffer

class **TBuffer** : public TObject

protected:

class TBuffer	**TBuffer**( )
class TBuffer	**TBuffer**(class TBuffer&)
void	**CheckCount**( )
void	**Expand**(Int_t *newsize*)
void	***operator=***(class TBuffer&)
void	**Read**(Text_t* *name*)
void	**Write**(Text_t* *name,* Int_t *bufs*)

public:

class TBuffer	**TBuffer**((*unknown*) TBuffer::EMode mode, Int_t bufsiz = kInitialSize, void* buf = 0)
void	**~TBuffer**( )
char*	**Buffer**( )
Int_t	**BufferSize**( )
class TClass*	**Class**( )
void	**DetachBuffer**( )
Int_t	**GetGlobalReadParam**( )
Int_t	**GetGlobalWriteParam**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsReading**( )
Bool_t	**IsWriting**( )
Int_t	**Length**( )
void	**MapObject**(class TObject* *obj*)
class TBuffer&	**operator<<**(Int_t *i*)
class TBuffer&	**operator<<**(Short_t *h*)
class TBuffer&	**operator<<**(UInt_t *i*)
class TBuffer&	**operator<<**(UShort_t *h*)
class TBuffer&	**operator<<**(Char_t *c*)
class TBuffer&	**operator<<**(Long_t *l*)
class TBuffer&	**operator<<**(Char_t* *c*)
class TBuffer&	**operator<<**(UChar_t *c*)
class TBuffer&	**operator<<**(ULong_t *l*)
class TBuffer&	**operator<<**(Float_t *f*)
class TBuffer&	**operator<<**(Double_t *d*)
class TBuffer&	**operator>>**(Char_t* *c*)
class TBuffer&	**operator>>**(ULong_t& *l*)
class TBuffer&	**operator>>**(Float_t& *f*)
class TBuffer&	**operator>>**(UShort_t& *h*)
class TBuffer&	**operator>>**(Long_t& *l*)
class TBuffer&	**operator>>**(UInt_t& *i*)
class TBuffer&	**operator>>**(Int_t& *i*)
class TBuffer&	**operator>>**(Char_t& *c*)
class TBuffer&	**operator>>**(Double_t& *d*)
class TBuffer&	**operator>>**(UChar_t& *c*)
class TBuffer&	**operator>>**(Short_t& *h*)
Int_t	**ReadArray**(Double_t*& *d*)
Int_t	**ReadArray**(Short_t*& *h*)
Int_t	**ReadArray**(UShort_t*& *h*)
Int_t	**ReadArray**(Char_t*& *c*)
Int_t	**ReadArray**(Int_t*& *i*)
Int_t	**ReadArray**(UChar_t*& *c*)
Int_t	**ReadArray**(UInt_t*& *i*)
Int_t	**ReadArray**(Long_t*& *l*)

Int_t	**ReadArray**(ULong_t*& *l*)
Int_t	**ReadArray**(Float_t*& *f*)
Int_t	**ReadBuf**(void* *buf,* Int_t *max*)
class TClass*	**ReadClass**(class TClass* *cl = 0,* UInt_t* *objTag = 0*)
void	**ReadFastArray**(Long_t* *l,* Int_t *n*)
void	**ReadFastArray**(UInt_t* *i,* Int_t *n*)
void	**ReadFastArray**(Int_t* *i,* Int_t *n*)
void	**ReadFastArray**(Float_t* *f,* Int_t *n*)
void	**ReadFastArray**(UShort_t* *h,* Int_t *n*)
void	**ReadFastArray**(Double_t* *d,* Int_t *n*)
void	**ReadFastArray**(Short_t* *h,* Int_t *n*)
void	**ReadFastArray**(UChar_t* *c,* Int_t *n*)
void	**ReadFastArray**(Char_t* *c,* Int_t *n*)
void	**ReadFastArray**(ULong_t* *l,* Int_t *n*)
class TObject*	**ReadObject**(class TClass* *cl*)
Int_t	**ReadStaticArray**(Int_t* *i*)
Int_t	**ReadStaticArray**(Char_t* *c*)
Int_t	**ReadStaticArray**(UInt_t* *i*)
Int_t	**ReadStaticArray**(UChar_t* *c*)
Int_t	**ReadStaticArray**(Long_t* *l*)
Int_t	**ReadStaticArray**(ULong_t* *l*)
Int_t	**ReadStaticArray**(Float_t* *f*)
Int_t	**ReadStaticArray**(Double_t* *d*)
Int_t	**ReadStaticArray**(Short_t* *h*)
Int_t	**ReadStaticArray**(UShort_t* *h*)
Text_t*	**ReadString**(Text_t* *s,* Int_t *max*)
Version_t	**ReadVersion**( )
virtual void	**Reset**( )
void	**ResetMap**( )
void	**SetBufferOffset**(Int_t *offset = 0*)
void	**SetGlobalReadParam**(Int_t *arraysize*)
void	**SetGlobalWriteParam**(Int_t *mapsize*)
void	**SetReadMode**( )
void	**SetReadParam**(Int_t *arraysize*)
void	**SetWriteMode**( )
void	**SetWriteParam**(Int_t *mapsize*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**WriteArray**(ULong_t* *l,* Int_t *n*)
void	**WriteArray**(Char_t* *c,* Int_t *n*)
void	**WriteArray**(Float_t* *f,* Int_t *n*)
void	**WriteArray**(UChar_t* *c,* Int_t *n*)
void	**WriteArray**(Double_t* *d,* Int_t *n*)
void	**WriteArray**(Short_t* *h,* Int_t *n*)
void	**WriteArray**(UShort_t* *h,* Int_t *n*)
void	**WriteArray**(Int_t* *i,* Int_t *n*)
void	**WriteArray**(UInt_t* *i,* Int_t *n*)
void	**WriteArray**(Long_t* *l,* Int_t *n*)
void	**WriteBuf**(void* *buf,* Int_t *max*)
void	**WriteClass**(class TClass* *cl*)
void	**WriteFastArray**(Long_t* *l,* Int_t *n*)
void	**WriteFastArray**(UInt_t* *i,* Int_t *n*)
void	**WriteFastArray**(Int_t* *i,* Int_t *n*)
void	**WriteFastArray**(UShort_t* *h,* Int_t *n*)
void	**WriteFastArray**(Double_t* *d,* Int_t *n*)
void	**WriteFastArray**(Short_t* *h,* Int_t *n*)
void	**WriteFastArray**(Float_t* *f,* Int_t *n*)
void	**WriteFastArray**(UChar_t* *c,* Int_t *n*)

void **WriteFastArray**(Char_t* *c,* Int_t *n*)
void **WriteFastArray**(ULong_t* *l,* Int_t *n*)
void **WriteObject**(class TObject* *obj*)
void **WriteString**(Text_t* *s*)
void **WriteVersion**(class TClass* *cl*)

Data Members

protected:

Bool_t	**fMode**	
Int_t	**fBufSize**	
char*	**fBuffer**	*Buffer used to store objects*
char*	**fBufCur**	*Current position in buffer*
char*	**fBufMax**	*End of buffer*
Int_t	**fMapCount**	*Number of objects or classes in map*
Int_t	**fArraySize**	*Default size of fReadArray array*
Int_t	**fMapSize**	*Default size of fWriteMap hash table*
class TObjArray*	**fReadArray**	
class TMap*	**fWriteMap**	
static Int_t	**fgArraySize**	*Default array size for all TBuffer objects*
static Int_t	**fgMapSize**	*Default map size for all TBuffer objects*

public:

static const enum TBuffer::EMode**kRead**
static const enum TBuffer::EMode**kWrite**
static const enum TObject:: **kInitialSize**
static const enum TObject:: **kMinimalSize**
static const enum TObject:: **kArraySize**
static const enum TObject:: **kMapSize**

See Also: TMessage

Class Description

```
TBuffer

Buffer base class used for serializing objects.
```

TBuffer(EMode *mode,* Int_t *bufsiz,* void **buf*)

```
Create an I/O buffer object. Mode should be either TBuffer::kRead or
TBuffer::kWrite. By default the I/O buffer has a size of
TBuffer::kInitialSize (1024) bytes.
```

~TBuffer( )

```
Delete an I/O buffer object.
```

void **CheckCount**( )

```
Check if Array or Map can contain more elements.
```

void **Expand**(Int_t *newsize*)

```
Expand the I/O buffer to newsize bytes.
```

void **MapObject**(const TObject **obj*)

```
Create the fWriteMap or fReadArray containers and initialize
them correctly. This method may only be called outside this
class by methods implementing Read and Write for non TObject
derived classes (see TArray::ReadArray() and TArray::WriteArray()).
```

void **SetReadParam**(Int_t *arraysize*)

```
Set the initial size of the array used to store object and class
references during reading. The default size is kArraySize=64.
Increasing the default has the benefit that when reading many
```

```
small objects the array does not need to be resized too often
(the system is always dynamic, even with the default everything
will work, only the resizing will cost some performance).
This method can only be called directly after the creation of
the TBuffer, before any reading is done. Globally this option
can be changed using SetGlobalReadParam().
```

void **SetWriteParam**(Int_t *mapsize*)

```
Set the initial size of the hashtable used to store object and class
references during writing. The default size is kMapSize=67.
Increasing the default has the benefit that when writing many
small objects the hashtable does not get too many collisions
(the system is always dynamic, even with the default everything
will work, only a large number of collisions will cost performance).
For optimal performance hashsize should always be a prime.
This method can only be called directly after the creation of
the TBuffer, before any writing is done. Globally this option
can be changed using SetGlobalWriteParam().
```

void **ResetMap**( )

```
Delete existing fWriteMap or fReadArray and reset map counter.
```

Int_t **ReadBuf**(void **buf,* Int_t *max*)

```
Read max bytes from the I/O buffer into buf. The function returns
the actual number of bytes read. The buffer buf is advanced by
the number of bytes read.
```

void **WriteBuf**(const void **buf,* Int_t *max*)

```
Write max bytes from buf into the I/O buffer. The buffer buf is advanced
by the number of bytes (max) written.
```

Text_t* **ReadString**(Text_t **s,* Int_t *max*)

```
Read string from I/O buffer. String is read till 0 character is
found or till max-1 characters are read (i.e. string s has max
bytes allocated).
```

void **WriteString**(const Text_t **s*)

```
Write string to I/O buffer. Writes string upto and including the
terminating 0.
```

Int_t **ReadArray**(Char_t *&*c*)

```
Read array of characters from the I/O buffer. Returns the number of
characters read. If argument is a 0 pointer then space will be
allocated for the array.
```

Int_t **ReadArray**(Short_t *&*h*)

```
Read array of shorts from the I/O buffer. Returns the number of shorts
read. If argument is a 0 pointer then space will be allocated for the
array.
```

Int_t **ReadArray**(Int_t *&*ii*)

```
Read array of ints from the I/O buffer. Returns the number of ints
read. If argument is a 0 pointer then space will be allocated for the
array.
```

Int_t **ReadArray**(Long_t *&*ll*)

```
Read array of longs from the I/O buffer. Returns the number of longs
read. If argument is a 0 pointer then space will be allocated for the
array.
```

Int_t **ReadArray**(Float_t *&*f*)

```
Read array of floats from the I/O buffer. Returns the number of floats
read. If argument is a 0 pointer then space will be allocated for the
```

array.

Int_t **ReadArray**(Double_t *&*d*)
Read array of doubles from the I/O buffer. Returns the number of doubles read. If argument is a 0 pointer then space will be allocated for the array.

Int_t **ReadStaticArray**(Char_t **c*)
Read array of characters from the I/O buffer. Returns the number of characters read.

Int_t **ReadStaticArray**(Short_t **h*)
Read array of shorts from the I/O buffer. Returns the number of shorts read.

Int_t **ReadStaticArray**(Int_t **ii*)
Read array of ints from the I/O buffer. Returns the number of ints read.

Int_t **ReadStaticArray**(Long_t **ll*)
Read array of longs from the I/O buffer. Returns the number of longs read.

Int_t **ReadStaticArray**(Float_t **f*)
Read array of floats from the I/O buffer. Returns the number of floats read.

Int_t **ReadStaticArray**(Double_t **d*)
Read array of doubles from the I/O buffer. Returns the number of doubles read.

void **ReadFastArray**(Char_t **c,* Int_t *n*)
Read array of n characters from the I/O buffer.

void **ReadFastArray**(Short_t **h,* Int_t *n*)
Read array of n shorts from the I/O buffer.

void **ReadFastArray**(Int_t **ii,* Int_t *n*)
Read array of n ints from the I/O buffer.

void **ReadFastArray**(Long_t **ll,* Int_t *n*)
Read array of n longs from the I/O buffer.

void **ReadFastArray**(Float_t **f,* Int_t *n*)
Read array of n floats from the I/O buffer.

void **ReadFastArray**(Double_t **d,* Int_t *n*)
Read array of n doubles from the I/O buffer.

void **WriteArray**(const Char_t **c,* Int_t *n*)
Write array of n characters into the I/O buffer.

void **WriteArray**(const Short_t **h,* Int_t *n*)
Write array of n shorts into the I/O buffer.

void **WriteArray**(const Int_t **ii,* Int_t *n*)
Write array of n ints into the I/O buffer.

void **WriteArray**(const Long_t **ll,* Int_t *n*)
Write array of n longs into the I/O buffer.

void **WriteArray**(const Float_t **f,* Int_t *n*)
Write array of n floats into the I/O buffer.

void **WriteArray**(const Double_t **d,* Int_t *n*)
```
Write array of n doubles into the I/O buffer.
```

void **WriteFastArray**(const Char_t **c,* Int_t *n*)
```
Write array of n characters into the I/O buffer.
```

void **WriteFastArray**(const Short_t **h,* Int_t *n*)
```
Write array of n shorts into the I/O buffer.
```

void **WriteFastArray**(const Int_t **ii,* Int_t *n*)
```
Write array of n ints into the I/O buffer.
```

void **WriteFastArray**(const Long_t **ll,* Int_t *n*)
```
Write array of n longs into the I/O buffer.
```

void **WriteFastArray**(const Float_t **f,* Int_t *n*)
```
Write array of n floats into the I/O buffer.
```

void **WriteFastArray**(const Double_t **d,* Int_t *n*)
```
Write array of n doubles into the I/O buffer.
```

TObject* **ReadObject**(const TClass **clReq*)
```
Read object from I/O buffer. clReq can be used to cross check
if the actually read object is of the requested class.
```

void **WriteObject**(const TObject **obj*)
```
Write object to I/O buffer.
```

TClass* **ReadClass**(const TClass **clReq,* UInt_t **objTag*)
```
Read class definition from I/O buffer. clReq can be used to cross check
if the actually read object is of the requested class. objTag is
set in case the object is a reference to an already read object.
```

void **WriteClass**(const TClass **cl*)
```
Write class description to I/O buffer.
```

Version_t **ReadVersion**( )
```
Read class version from I/O buffer.
```

void **WriteVersion**(const TClass **cl*)
```
Write class version to I/O buffer.
```

void **SetGlobalReadParam**(Int_t *arraysize*)
```
Set the initial size of the array used to store object and class
references during reading. The default size is kArraySize=64.
Increasing the default has the benefit that when reading many
small objects the array does not need to be resized too often
(the system is always dynamic, even with the default everything
will work, only the resizing will cost some performance).
Per TBuffer object this option can be changed using SetReadParam().
```

void **SetGlobalWriteParam**(Int_t *mapsize*)
```
Set the initial size of the hashtable used to store object and class
references during writing. The default size is kMapSize=67.
Increasing the default has the benefit that when writing many
small objects the hashtable does not get too many collisions
(the system is always dynamic, even with the default everything
will work, only a large number of collisions will cost performance).
For optimal performance hashsize should always be a prime.
Per TBuffer object this option can be changed using SetWriteParam().
```

void **SetWriteMode**( )

Author: Fons Rademakers 04/05/96
Last update: 1.03/06 24/10/97 13.25.13 by Fons Rademakers

TBusyDialog

class **TBusyDialog** : public TDialogManager

protected:

class TBusyDialog	**TBusyDialog**( )

public:

class TBusyDialog	**TBusyDialog**(char* *name*)
class TBusyDialog	**TBusyDialog**(class TBusyDialog&)
void	**~TBusyDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
The busy dialog.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TButton

class **TButton** : public TPad, public TAttText

public:

class TButton	**TButton**(Text_t* *title,* Text_t* *method,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TButton	**TButton**( )
class TButton	**TButton**(class TButton& *button*)
void	**~TButton**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *button*)
virtual void	**Divide**(Int_t *nx = 1,* Int_t *ny = 1,* Float_t *xmargin* = 0.*01,* Float_t *ymargin* = 0.*01,* Int_t *color = 0*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Bool_t	**GetFraming**( )
virtual const Text_t*	**GetMethod**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintModified**( )
virtual void	**Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetBorderMode**(Short_t *bordermode*)
virtual void	**SetFraming**(Bool_t *f = 1*)
virtual void	**SetGrid**(Int_t *valuex = 1,* Int_t *valuey = 1*)
virtual void	**SetLogx**(Int_t *value = 1*)
virtual void	**SetLogy**(Int_t *value = 1*)
virtual void	**SetMethod**(Text_t* *method*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**x3d**(Option_t* *option*)

Data Members

private:

Bool_t	**fFocused**	*If cursor is in...*
Bool_t	**fFraming**	*True if you want a frame to be painted when pressed*

protected:

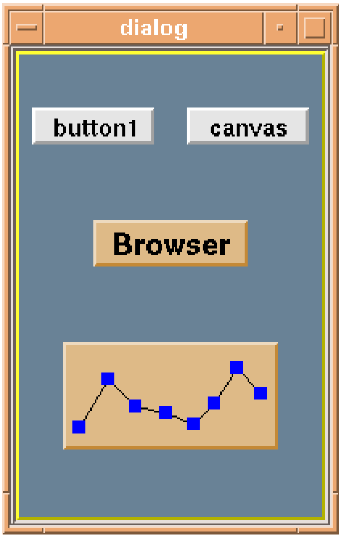
class TString	**fMethod**	*Method to be executed by this button*

See Also: TGroupButton

Class Description

```
A TButton object is a user interface object.
A TButton has a name and an associated action.
When the button is clicked with the left mouse button, the cooresponding
action is executed.
A Tbutton can be created by direct invokation of the constructors
or via the graphics editor.
The Action can be set via TButton::SetMethod.
The action can be any command. Examples of actions:
  "34+78" When the button is clicked, the result of addition is printed.
  ".x macro.C" . Clicking the button executes the macro macro.C
The action can be modified at any time via TButton::SetMethod.

To modify the layout/size/contents of one or several buttons
```

```
in a canvas, you must set the canvas editable via TCanvas::SetEditable.
By default a TCanvas is editable.
By default a TDialogCanvas is not editable.
TButtons are in general placed in a TDialogCanvas.

A TButton being a TPad, one can draw graphics primitives in it
when the TCanvas/TDialogCanvas is editable.

    Example of a macro creating a dialogcanvas with buttons
void but() {
example of a dialogcanvas with a few buttons

TDialogCanvas *dialog = new TDialogCanvas("dialog","",200,300);

Create first button. Clicking on this button will execute 34+56
TButton *but1 = new TButton("button1","34+56",.05,.8,.45,.88);
but1->Draw();

Create second button. Clicking on this button will create a new canvas
TButton *but2 = new TButton("canvas","c2 = new TCanvas("c2")",.55,.8,.95,.88);
but2->Draw();

Create third button. Clicking on this button will invoke the browser
but3 = new TButton("Browser","br = new TBrowser("br")",0.25,0.54,0.75,0.64);
but3->SetFillColor(42);
but3->Draw();

Create last button with no name. Instead a graph is draw inside the button
Clicking on this button will invoke the macro tutorials/graph.C
button = new TButton("",".x tutorials/graph.C",0.15,0.15,0.85,0.38);
button->SetFillColor(42);
button->Draw();
button->cd();

Float_t x[8] = {0.08,0.21,0.34,0.48,0.61,0.7,0.81,0.92};
Float_t y[8] = {0.2,0.65,0.4,0.34,0.24,0.43,0.75,0.52};
TGraph *graph = new TGraph(8,x,y);
graph->SetMarkerColor(4);
graph->SetMarkerStyle(21);
graph->Draw("lp");

dialog->cd();
}

Executing the macro above produces the following dialogcanvas
```



TButton( ): **TPad**( )

```
Button default constructor
```

TButton(const Text_t **title,* const Text_t **method,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TPad**("button", *title, x1, y1, x2, y2, 18, 2, 1*), **TAttText**(*22, 0, 1, 61,* 0.65)

```
Button normal constructor
Note that the button coordinates x1,y1,x2,y2 are always in the range [0,1]
```

~**TButton**( )

```
Button default destructor
```

void **Copy**(TObject &*button*)

```
Copy this button to button
```

void **Draw**(Option_t **option*)

```
Draw this button with its current attributes
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a Button object is clicked.
```

void **Paint**(Option_t **option*)

```
Paint this button with its current attributes
```

void **Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)

```
Set world coordinate system for the pad
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 01/07/96
Last update: 04/11/97 18.10.20 by René Brun

TCallbackObject

class **TCallbackObject**

private:	
void	**AddNewCallbackList**(char* *name*)
protected:	
class TCallbackObject	**TCallbackObject**( )
void	**CallCallbacks**(char* *callback,* void* *callData*)
public:	
class TCallbackObject	**TCallbackObject**(class TCallbackObject&)
void	**~TCallbackObject**( )
void	**AddCallback**(char* *name,* CallbackFunction_t *callbackFunction,* void* *clientData = NULL*)
void	**AddCallback**(char* *name,* class TCallbackObject* *component,* CallbackMethod_t *memberFunction,* void* *clientData = NULL*)
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
void	**RemoveAllCallbacks**( )
void	**RemoveAllCallbacks**(class TCallbackObject* *obj*)
void	**RemoveCallback**(char* *name,* class TCallbackObject* *otherObject,* CallbackMethod_t *memberFunction,* void* *clientData = NULL*)
void	**RemoveCallback**(char* *name,* CallbackFunction_t *callbackFunction,* void* *clientData = NULL*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:	
class TCallbackList**	**fCallbackLists**
int	**fNumberOfCallbackLists**
int	**fMaxCallbackLists**

See Also: TComponent, TInput, TPeriodic

Class Description

```
This class forms the base class for all of the user interface
components in the MOTIF graphical interface.
```

void **AddCallback**(const char **name,*TCallbackObject *component,CallbackMethod_t memberFunction,void **clientData*)

```
First we check to see that a callback of this type hasn't been
registered before.
If it has, we add it to its list.  Otherwise, we create a new
callback list for it.
```

void **RemoveCallback**(const char **name,*TCallbackObject *otherObject,CallbackMethod_t memberFunction,void **clientData*)

```
First we need to find the callback list named NAME, then we need to
locate the object and method within the callback list.  We then
pop everything down one.
```

void **CallCallbacks**(const char *const *callback,*void **callData*)

Author: Fons Rademakers 30/03/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TCanvas

class **TCanvas** : public TPad

private:

void	**Build**( )
void	**CopyPixmaps**( )
void	**DrawEventStatus**(Int_t *button,* Int_t *x,* Int_t *y,* class TObject* *selected*)

protected:

virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

public:

class TCanvas	**TCanvas**(Text_t* *name,* Text_t* *title,* Int_t *form = 1*)
class TCanvas	**TCanvas**(Text_t* *name,* Text_t* *title,* Int_t *ww,* Int_t *wh*)
class TCanvas	**TCanvas**(Text_t* *name,* Text_t* *title,* Int_t *wtopx,* Int_t *wtopy,* Int_t *ww,* Int_t *wh*)
class TCanvas	**TCanvas**( )
class TCanvas	**TCanvas**(class TCanvas& *canvas*)
void	**~TCanvas**( )
void	**cd**(Int_t *subpadnumber = 0*)
class TClass*	**Class**( )
void	**Clear**(Option_t* *option*)
void	**Close**(Option_t* *option*)
void	**Copy**(class TObject& *canvas*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**EditorBar**( )
void	**FeedbackMode**(Bool_t *set*)
void	**Flush**( )
void	**ForceUpdate**( )
Int_t	**GetCanvasID**( )
class TCanvasImp*	**GetCanvasImp**( )
virtual void	**GetCanvasPar**(Int_t& *wtopx,* Int_t& *wtopy,* UInt_t& *ww,* UInt_t& *wh*)
class TContextMenu*	**GetContextMenu**( )
const Text_t*	**GetDISPLAY**( )
Int_t	**GetDoubleBuffer**( )
class TControlBar*	**GetEditorBar**( )
Int_t	**GetEvent**( )
Color_t	**GetHighLightColor**( )
virtual void*	**GetPadDivision**(Int_t *xdivision = 1,* Int_t *ydivision = 1*)
class TVirtualPad*	**GetPadSave**( )
class TObject*	**GetSelected**( )
const Option_t*	**GetSelectedOpt**( )
class TVirtualPad*	**GetSelectedPad**( )
Int_t	**GetShowEventStatus**( )
UInt_t	**GetWh**( )
Int_t	**GetWtopX**( )
Int_t	**GetWtopY**( )
UInt_t	**GetWw**( )
Size_t	**GetXsizeReal**( )
Size_t	**GetXsizeUser**( )
Size_t	**GetYsizeReal**( )
Size_t	**GetYsizeUser**( )
virtual void	**HandleInput**(Int_t *button,* Int_t *x,* Int_t *y*)
void	**Iconify**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsBatch**( )
enum ECanvasEditMode	**IsEditable**( )
Bool_t	**IsRetained**( )

virtual void	**ls**(Option_t* *option*)
void	**MakeDefCanvas**( )
void	**MoveOpaque**(Int_t *set = 1*)
Bool_t	**OpaqueMoving**( )
Bool_t	**OpaqueResizing**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**Resize**(Option_t* *option*)
void	**ResizeOpaque**(Int_t *set = 1*)
void	**SaveSource**(Text_t* *filename,* Option_t* *option*)
void	**SetBatch**(Bool_t *batch = kTRUE*)
void	**SetCanvas**(UInt_t *ww,* UInt_t *wh*)
virtual void	**SetCursor**((*unknown*) ECursor cursor)
virtual void	**SetDoubleBuffer**(Int_t *mode = 1*)
virtual void	**SetEditable**(enum ECanvasEditMode *mode = kIsEditable*)
void	**SetHighLightColor**(Color_t *col*)
void	**SetRetained**(Bool_t *retained = kTRUE*)
void	**SetSelected**(class TObject* *obj*)
void	**SetTitle**(Text_t* *title*)
void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Size**(Float_t *xsizeuser = 0,* Float_t *ysizeuser = 0*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**ToggleEventStatus**( )
virtual void	**Update**( )
void	**UseCurrentStyle**( )

Data Members

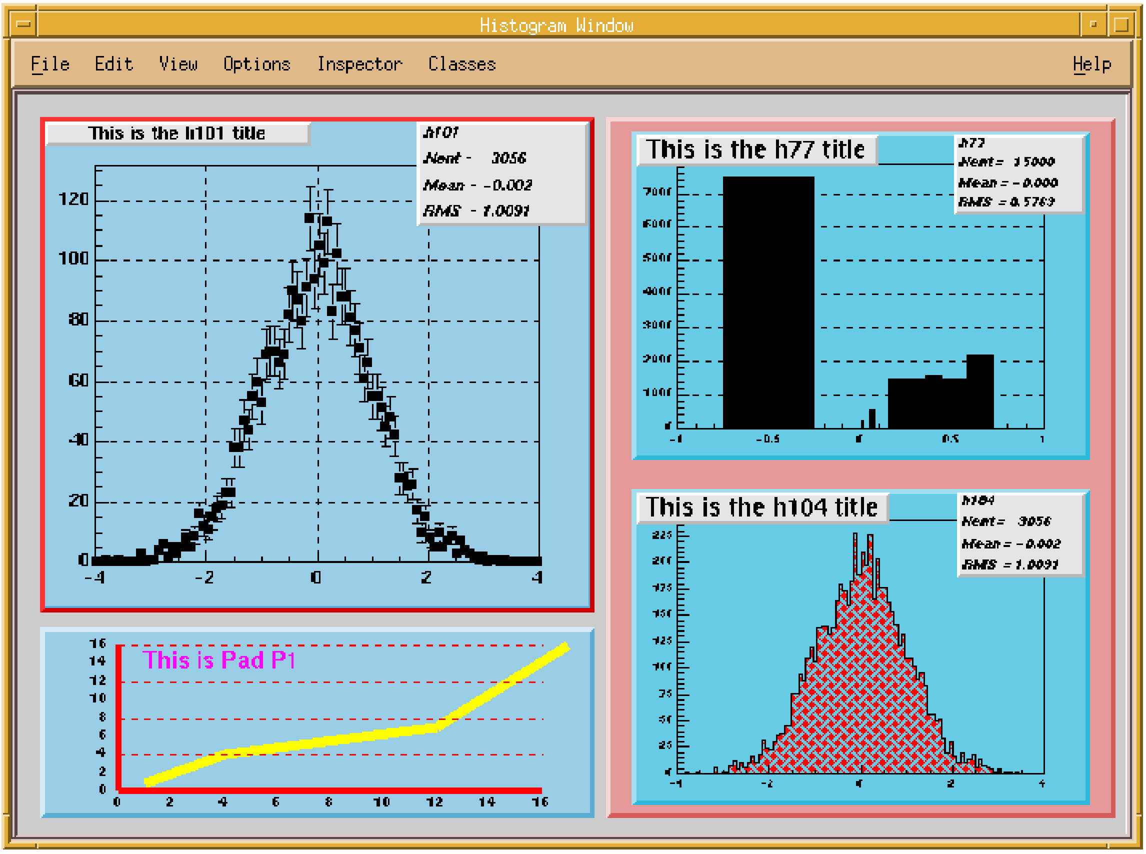
protected:

class TString	**fDISPLAY**	*Name of destination screen*
Int_t	**fDoubleBuffer**	*Double buffer flag (0=off, 1=on)*
Bool_t	**fRetained**	*Retain structure flag*
Size_t	**fXsizeUser**	*User specified size of canvas along X in CM*
Size_t	**fYsizeUser**	*User specified size of canvas along Y in CM*
Size_t	**fXsizeReal**	*Current size of canvas along X in CM*
Size_t	**fYsizeReal**	*Current size of canvas along Y in CM*
Int_t	**fCanvasID**	*Canvas identifier*
Int_t	**fCtopX**	*X top left corner of canvas in pixel coordinates*
Int_t	**fCtopY**	*Y top left corner of canvas in pixel coordinates*
UInt_t	**fCw**	*Width of the canvas along X (pixels)*
UInt_t	**fCh**	*Height of the canvas along Y (pixels)*
class TAttCanvas	**fCatt**	*Canvas attributes*
class TObject*	**fSelected**	*Currently selected object*
class TString	**fSelectedOpt**	*Drawing option of selected object*
class TVirtualPad*	**fSelectedPad**	*Pad containing currently selected object*
class TVirtualPad*	**fPadSave**	*Pointer to saved pad in HandleInput*
Int_t	**fEvent**	*Type of current or last handled event*
Int_t	**fShowEventStatus**	*Show event status panel*
class TControlBar*	**fEditorBar**	*Editor control bar*
class TCanvasImp*	**fCanvasImp**	*Window system specific canvas implementation*
Bool_t	**fMoveOpaque**	*Move objects in opaque mode*
Bool_t	**fResizeOpaque**	*Resize objects in opaque mode*
Color_t	**fHighLightColor**	*Highlight color of active pad*
Bool_t	**fBatch**	*True when in batchmode*
enum ECanvasEditMode	**fIsEditable**	*True if canvas is editable*
class TContextMenu*	**fContextMenu**	*Context menu pointer*

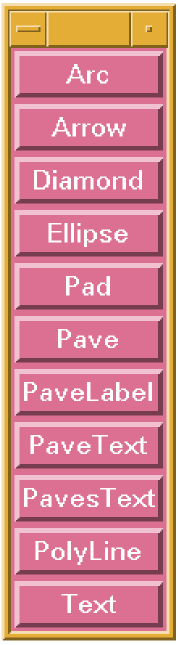
See Also: TDialogCanvas

Class Description

```
A Canvas is an area mapped to a window directly under
the control of the display manager.
A ROOT session may have several canvases open at any given time.

A Canvas may be subdivided into independent graphical areas: the PADs
A canvas has a default pad which has the name of the canvas itself.
An example of a Canvas layout is sketched in the picture below.

**********************************************************************
*          Tool Bar menus    for Canvas
**********************************************************************
*
*  ************************************    *************************
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *              P1                  *    *        P2             *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  *                                  *    *                       *
*  ************************************    *************************
*
**********************************************************************

This canvas contains two pads named P1 and P2.
Both Canvas, P1 and P2 can be moved, grown, shrinked using the
normal rules of the Display manager.
A copy of a real canvas with 4 pads is shown in the picture below.
```




```
Once objects have been drawn in a canvas, they can be edited/moved
by pointing directly to them. The cursor shape is changed
to suggest the type of action that one can do on this object.
Clicking with the right mouse button on an object pops-up
a contextmenu with a complete list of actions possible on this object.

A graphical editor may be started from the canvas "Edit" menu.
Select the "Editor" item. This will show the following editor menu.
```



```
For example, to draw a new TText primitive, select the option Text,
then click at the position where you want to draw the text. Type <CR>
when you have finished typing the string.

A canvas may be automatically divided into pads via TPad::Divide.
```

TCanvas( ) : **TPad**( )

```
Canvas default constructor
```

TCanvas(Text_t **name,* Text_t **title,* Int_t *form*) : **TPad**( )

```
Canvas constructor using a template
Create a new canvas with a predefined size form.
   if form < 0  the menubar is not shown.

form = 1    700x500 at 10,10
form = 2    500x500 at 20,20
form = 3    500x500 at 30,30
form = 4    500x500 at 40,40
form = 5    500x500 at 50,50
```

TCanvas(Text_t **name,* Text_t **title,* Int_t *ww,* Int_t *wh*) : **TPad**( )

```
Canvas constructor
Create a new canvas at a random position.

wide is the canvas size in pixels along X
   if ww < 0  the menubar is not shown.
wh is the canvas size in pixels along Y
```

TCanvas(Text_t **name,* Text_t **title,* Int_t *wtopx,* Int_t *wtopy,* Int_t *ww,* Int_t *wh*): **TPad**( )

```
Canvas constructor
Create a new canvas.

wtopx,wtopy are the pixel coordinates of the top left corner of the canvas
  if wtopx < 0) the menubar is not shown.
ww is the canvas size in pixels along X
wh is the canvas size in pixels along Y
```

void **Build**( )

```
Auxiliary for Canvas constructor
```

~TCanvas( )

```
Canvas default destructor
```

void **cd**(Int_t *subpadnumber*)

```
Set current canvas & pad
```

void **Clear**(Option_t **option*)

```
Clear canvas
Remove all primitives in default pad
Remove all pads (except default pad)
```

void **Close**(Option_t **option*)

```
Close canvas
Delete window/pads data structure
```

void **Copy**(TObject &*obj*)

```
Copy all pads in this canvas into another canvas
```

void **CopyPixmaps**( )

```
Copy the canvas pixmap of the pad to the canvas
```

void **Draw**(Option_t **option*)

```
Drawing a canvas
```

```
If a canvas with the name is already on the screen, the canvas is repainted.
This function is useful when a canvas object has been saved in a Root file.
One can then do:
Root > Tfile f("file.root");
Root > canvas.Draw();
```

void **DrawEventStatus**(Int_t *event,* Int_t *px,* Int_t *py,* TObject **selected*)

```
Report name and title of primitive below the cursor
 This function is called when the option "Event Status"
 in the canvas menu "Options" is selected.
```

void **EditorBar**( )

```
Create the Editor Controlbar
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the canvas

Only handle mouse motion events in TCanvas, all other events are
ignored for the time being
```

void **FeedbackMode**(Bool_t *set*)

```
Turn rubberband feedback mode on or off
```

void **Flush**( )

```
Flush canvas buffers
```

void **UseCurrentStyle**( )

```
Force a copy of current style for all objects in canvas
```

void* **GetPadDivision**(Int_t *xdivision,* Int_t *ydivision*)

```
Return pad corresponding to one canvas division
```

void **HandleInput**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Handle Input Events
Handle input events, like button up/down in current canvas.
```

void **ls**(Option_t **option*)

```
List all pads
```

void **MakeDefCanvas**( )

```
Static function to build a default canvas
```

void **MoveOpaque**(Int_t *set*)

```
Set option to move objects/pads in a canvas
if set = 1 (default) graphics objects are moved in opaque mode
      = 0 only the outline of objects is drawn when moving them
The option opaque produces the best effect. It requires however a
a reasonably fast workstation or response time.
```

void **Paint**(Option_t **option*)

```
Paint canvas
```

void **Resize**(Option_t **option*)

```
Recompute canvas parameters following a X11 Resize
```

void **SaveSource**(Text_t **filename,* Option_t **option*)

```
Save primitives in this canvas as a C++ macro file
```

void **SetBatch**(Bool_t *batch*)

```
Toggle batch mode. However, if the canvas is created without a window
```

```
then batch mode always stays set.
```

void **SetCursor**(ECursor *cursor*)

```
Set cursor
```

void **SetDoubleBuffer**(Int_t *mode*)

```
Set Double Buffer On/Off
```

void **SetTitle**(Text_t **title*)

```
Set Canvas title
```

void **Size**(Float_t *xsize,* Float_t *ysize*)

```
Set the canvas scale in centimeters
This information is used by PostScript to set the page size.
xsize  = size of the canvas in centimeters along X
ysize  = size of the canvas in centimeters along Y
if xsize and ysize are not equal to 0, then the scale factors will
be computed to keep the ratio ysize/xsize independently of the canvas
size (parts of the physical canvas will be unused).

if xsize = 0 and ysize is not zero, then xsize will be computed
   to fit to the current canvas scale. If the canvas is resized,
   a new value for xsize will be recomputed. In this case the aspect
   ratio is not preserved.

if both xsize = 0 and ysize = 0, then the scaling is automatic.
the largest dimension will be allocated a size of 20 centimeters.
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **Update**( )

```
Update canvas pad buffers
```

Author: René Brun 12/12/94
Last update: 1.03/06 27/10/97 18.03.52 by René Brun

TCanvasImp

class **TCanvasImp**

public:

class TCanvasImp	**TCanvasImp**(class TCanvas* *c,* char* *name,* UInt_t *width,* UInt_t *height*)
class TCanvasImp	**TCanvasImp**(class TCanvas* *c = 0*)
class TCanvasImp	**TCanvasImp**(class TCanvas* *c,* char* *name,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TCanvasImp	**TCanvasImp**(class TCanvasImp&)
void	**~TCanvasImp**( )
class TCanvas*	**Canvas**( )
class TClass*	**Class**( )
virtual void	**ForceUpdate**( )
virtual void	**Iconify**( )
virtual Int_t	**InitWindow**( )
virtual class TClass*	**IsA**( )
virtual void	**SetCanvas**(UInt_t *w,* UInt_t *h*)
virtual void	**SetCanvasTitle**(Text_t* *newTitle*)
virtual void	**SetStatusText**(Text_t* *text = 0,* Int_t *partidx = 0*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**ShowMenuBar**(Bool_t *show = kTRUE*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TCanvas*	**fCanvas**	*TCanvas associated with this implementation*

See Also: TMotifCanvas

Class Description

```
ABC describing GUI independent main window (with menubar, scrollbars
and a drawing area).
```

Author: Fons Rademakers 16/11/95
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TChain

class **TChain** : public TTree

public:

class TChain	**TChain**(Text_t* *name,* Text_t* *title*)
class TChain	**TChain**( )
class TChain	**TChain**(class TChain&)
void	**~TChain**( )
virtual void	**Add**(Text_t* *name*)
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**CreatePackets**( )
virtual void	**Draw**(Text_t* *varexp,* Text_t* *selection,* Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual class TBranch*	**GetBranch**(Text_t* *name*)
virtual Int_t	**GetEvent**(Int_t *event = 0,* Int_t *getall = 0*)
class TFile*	**GetFile**( )
class TLeaf*	**GetLeaf**(Text_t* *name*)
class TObjArray*	**GetListOfBranches**( )
class TObjArray*	**GetListOfFiles**( )
class TObjArray*	**GetListOfLeaves**( )
virtual Int_t	**GetNbranches**( )
virtual Int_t	**GetNtrees**( )
class TList*	**GetStatus**( )
class TTree*	**GetTree**( )
Int_t	**GetTreeNumber**( )
Int_t*	**GetTreeOffset**( )
virtual Int_t	**GetTreeOffsetLen**( )
virtual class TClass*	**IsA**( )
Int_t	**LoadTree**(Int_t *event*)
virtual void	**Loop**(Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual void	**ls**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SetBranchAddress**(Text_t* *bname,* void* *add*)
virtual void	**SetBranchStatus**(Text_t* *bname,* Bool_t *status = 1*)
virtual void	**SetPacketSize**(Int_t *size = 100*)
virtual void	**SetTreeOffsetLen**(Int_t *len*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fTreeOffsetLen**	*Current size of fTreeOffset array*
Int_t	**fNtrees**	*Number of Trees*
Int_t	**fTreeNumber**	*Current Tree number in fTreeOffset table*
Int_t	**fNbranches**	*Number of branches in associated Tree*
Long_t*	**fBaddresses**	*Branch addresses (if set)*
Int_t*	**fTreeOffset**	*Array of variables*
class TTree*	**fTree**	*Pointer to current tree*
class TFile*	**fFile**	*Pointer to current file*
class TObjArray*	**fFiles**	*List of file names containing the Trees*
class TList*	**fStatus**	*List of active/inactive branches*

Class Description

```
TChain
```

```
A chain is a collection of files containg TTree objects.
When the chain is created, the first parameter is the default name
for the Tree to be processed later on.

Enter a new element in the chain via the TChain::Add function.
Once a chain is defined, one can use the normal TTree functions
to Draw,Scan,etc.

Use TChain::SetBranchStatus to activate one or more branches for all
the trees in the chain.
```

TChain( ): **TTree**( )

```
Default constructor for Chain
```

TChain(const Text_t **name,* const Text_t **title*):**TTree**(*name, title*)

```
Create a Chain
A TChain is a collection of TFile objects.
 the first parameter "name" is the name of the TTree object
 in the files added with Add.
Use TChain::Add to add a new element to this chain.

 Example:
Suppose we have 3 files f1.root, f2.root and f3.root. Each file
contains a TTree object named "T".
  TChain ch("T");  creates a chain to process a Tree called "T"
  ch.Add("f1.root");
  ch.Add("f2.root");
  ch.Add("f3.root");
  ch.Draw("x");
    The Draw function above will process the variable "x" in Tree "T"
    reading sequentially the 3 files in the chain ch.
```

~**TChain**( )

```
Default destructor for a Chain
```

void **Add**(const Text_t **name*)

```
    Add a new element to this chain.
    An element can be the name of another chain or the name of a file
    containing a tree.
 name may have the following format:
    //machine/file_name.root/subdir/tree_name
   machine, subdir and tree_name are optional. If tree_name is missing,
   the chain name will be assumed.
```

void **CreatePackets**( )

```
Initialize the packet descriptor string
```

void **Draw**(const Text_t **varexp,* const Text_t **selection,* Option_t **option,* Int_t *nevents,* Int_t *firstevent*)

```
  Process all events in this chain and draw histogram
  corresponding to expression varexp
```

Int_t **GetEvent**(Int_t *event,* Int_t *getall*)

```
Return event in memory
  getall = 0 : get only active branches
  getall = 1 : get all branches
```

TObjArray* **GetListOfBranches**( )

```
Return pointer to list of branches of current tree
```

TObjArray* **GetListOfLeaves**( )

```
Return pointer to list of leaves of current tree
```

Int_t **LoadTree**(Int_t *event*)

```
Load tree corresponding to event
```

void **Loop**(Option_t **option,* Int_t *nevents,* Int_t *firstevent*)

```
Loop on nevents of this chain starting at firstevent
```

void **SetBranchAddress**(const Text_t **bname,* void **add*)

```
Set branch address
   The branch address is stored into the chain branch addresses (if set).
   When LoadTree is called, the branch address is copied
   to the current Tree branch address.
```

void **SetBranchStatus**(const Text_t **bname,* Bool_t *status*)

```
Set branch status Process or DoNotProcess
   bname is the name of a branch. if bname="*", apply to all branches.
   status = 1  branch will be processed
          = 0  branch will not be processed
```

void **SetPacketSize**(Int_t *size*)

```
Set number of events per packet for parallel root
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 03/02/97
Last update: 1.01/02 07/05/97 09.48.58 by René Brun

TChainElement

class **TChainElement** : public TNamed

public:

class TChainElement	**TChainElement**( )
class TChainElement	**TChainElement**(Text_t* *filename,* Text_t* *title*)
class TChainElement	**TChainElement**(class TChainElement&)
void	**~TChainElement**( )
class TClass*	**Class**( )
virtual void	**CreatePackets**( )
virtual Text_t*	**GetPackets**( )
virtual Int_t	**GetPacketSize**( )
virtual class TClass*	**IsA**( )
virtual void	**SetNumberEntries**(Int_t *n*)
virtual void	**SetPacketSize**(Int_t *size = 100*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fEntries**	*Number of entries in the tree of this chain element*
Int_t	**fNPackets**	*Number of packets*
Int_t	**fPacketSize**	*Number of events in one packet for parallel root*
Text_t*	**fPackets**	*Packet descriptor string*

Class Description

```
A TChainElement describes a component of a TChain.
```

TChainElement( ): **TNamed**( )

```
Default constructor for a chain element
```

TChainElement(const Text_t **name,* const Text_t **title*):**TNamed**(*name, title*)

```
Create a chain element
```

~TChainElement( )

```
Default destructor for a chain element
```

void **CreatePackets**( )

```
Initialize the packet descriptor string
```

void **SetPacketSize**(Int_t *size*)

```
Set number of events per packet for parallel root
```

Author: René Brun 11/02/97
Last update: 1.00/06 23/03/97 11.38.37 by René Brun

TCheckBox

class **TCheckBox** : public TComponent

private:

void	**ValueChangedCallback**(*Widget, XtPointer, XtPointer*)

protected:

class TCheckBox	**TCheckBox**( )

public:

class TCheckBox	**TCheckBox**(char* *name,* Widget *parent*)
class TCheckBox	**TCheckBox**(class TCheckBox&)
void	**~TCheckBox**( )
Widget	**AddItem**(char* *name,* Boolean *state = FALSE,* XtCallbackProc *proc = NULL,* XtPointer *clientDate = NULL*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
int	**GetValue**(int *index*)
virtual class TClass*	**IsA**( )
void	**SetValue**(int *index,* Boolean *newValue*)
void	**SetValues**(Boolean* *values,* int *numValues*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**ValueChanged**(int *index,* Boolean *newValue*)

Data Members

protected:

class TWidgetList*	**fWidgetList**
Widget	**fRc**
Widget	**fLabel**

public:

static const char *const	**fgItemChangedCallback**

See Also: TRadioBox

Class Description

```
This class allows the creation of check-box type toggle button with
a title and a frame surrounding them.
```

const char* **ClassName**( )

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TCint

class **TCint** : public TInterpreter

private:

class TCint	**TCint**( )

protected:

virtual void	**ExecThreadCB**(TWin32SendClass* *command*)

public:

class TCint	**TCint**(class TCint&)
class TCint	**TCint**(char* *name,* char* *title*)
void	**~TCint**( )
virtual Int_t	**Calc**(char* *line*)
class TClass*	**Class**( )
virtual void	**CreateListOfBaseClasses**(class TClass* *cl*)
virtual void	**CreateListOfDataMembers**(class TClass* *cl*)
virtual void	**CreateListOfMethodArgs**(class TFunction* *m*)
virtual void	**CreateListOfMethods**(class TClass* *cl*)
virtual void	**CreateListOfTypes**( )
virtual Int_t	**DeleteGlobal**(void* *obj*)
virtual void	**EndOfLineAction**( )
virtual void	**Execute**(Text_t* *function,* Text_t* *params*)
virtual void	**Execute**(class TObject* *obj,* class TClass* *cl,* Text_t* *method,* Text_t* *params*)
virtual void	**Execute**(class TObject* *obj,* class TClass* *cl,* class TMethod* *method,* class TObjArray* *params*)
virtual Int_t	**ExecuteMacro**(char* *filename*)
void*	**FindObject**(char* name, (*unknown*) G__ClassInfo* type, void** prevObj, void** assocPtr)
virtual void*	**GetInterfaceMethod**(class TClass* *cl,* Text_t* *method,* Text_t* *params*)
virtual void*	**GetInterfaceMethodWithPrototype**(class TClass* *cl,* Text_t* *method,* Text_t* *proto*)
virtual Int_t	**GetMore**( )
virtual char*	**GetPrompt**( )
virtual Int_t	**InitializeDictionaries**( )
virtual class TClass*	**IsA**( )
virtual void	**LoadMacro**(char* *filename*)
virtual void	**PrintIntro**( )
virtual Int_t	**ProcessLine**(char* *line*)
Int_t	**ProcessLineAsynch**(char* *line*)
virtual Int_t	**ProcessLineSynch**(char* *line*)
virtual void	**Reset**( )
virtual void	**ResetAll**( )
virtual void	**ResetGlobals**( )
virtual void	**SaveContext**( )
virtual void	**SaveGlobalsContext**( )
virtual void	**SetClassInfo**(class TClass* *cl*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual const char*	**TypeName**(char* *typeDesc*)
void	**UpdateAllCanvases**( )
virtual void	**UpdateListOfGlobalFunctions**( )
virtual void	**UpdateListOfGlobals**( )

Data Members

private:

Int_t	**fMore**	*1 if more input is required*
char	**fPrompt**	*proposed prompt string*
struct G__dictposition	**fDictPos**	*CINT dictionary context after init*
struct G__dictposition	**fDictPosGlobals**	*CINT dictionary context after ResetGlobals( )*

Class Description

```
This class defines an interface to the CINT C/C++ interpreter made
by Masaharu Goto from HP Japan.

CINT is an almost full ANSI compliant C/C++ interpreter.
```

TCint(const char **name,* const char **title*) : **TInterpreter**(*name,* title)

```
Initialize the CINT interpreter interface.
```

~**TCint**( )

```
Destroy the CINT interpreter interface.
```

Int_t **InitializeDictionaries**( )

```
Initialize all registered dictionaries. Normally this is already done
by G__init_cint() and G__dlmod().
```

void **EndOfLineAction**( )

```
It calls a "fantom" method to synchronize user keyboard input
and ROOT prompt line.
```

void **ExecThreadCB**(TWin32SendClass **command*)

```
This function must be called from the "Command thread only".
```

void **LoadMacro**(const char **filename*)

```
Load a macro file in CINT's memory.
```

Int_t **ProcessLine**(const char **line*)

```
Let CINT process a command line.
```

Int_t **ProcessLineAsynch**(const char **line*)

```
Let CINT process a command line asynch.
```

Int_t **ProcessLineSynch**(const char **line*)

```
Let CINT process a command line synchronously, i.e we are waiting
it will be finished.
```

Int_t **Calc**(const char **line*)

```
Directly execute an executable statement (e.g. "func()", "3+5", etc.
however not declarations, like "Int_t x;").
```

void **PrintIntro**( )

```
Print CINT introduction and help message.
```

void **Reset**( )

```
Reset the CINT state to the state saved by the last call to
TCint::SaveContext().
```

void **ResetAll**( )

```
Reset the CINT state to its initial state.
```

void **ResetGlobals**( )

```
Reset the CINT global object state to the state saved by the last
call to TCint::SaveGlobalsContext().
```

Int_t **DeleteGlobal**(void **obj*)

```
Delete obj from CINT symbol table so it cannot be accessed anymore.
Returns 1 in case of success and 0 in case object was not in table.
```

void **SaveContext**( )
    Save the current CINT state.

void **SaveGlobalsContext**( )
    Save the current CINT state of global objects.

void **UpdateListOfGlobals**( )
    Update the list of pointers to global variables. This function
    is called by TROOT::GetListOfGlobals().

void **UpdateListOfGlobalFunctions**( )
    Update the list of pointers to global functions. This function
    is called by TROOT::GetListOfGlobalFunctions().

void **SetClassInfo**(TClass **cl*)
    Set pointer to CINT's G__ClassInfo in TClass.

void **CreateListOfBaseClasses**(TClass **cl*)
    Create list of pointers to base class(es) for TClass cl.

void **CreateListOfDataMembers**(TClass **cl*)
    Create list of pointers to data members for TClass cl.

void **CreateListOfMethods**(TClass **cl*)
    Create list of pointers to methods for TClass cl.

void **CreateListOfMethodArgs**(TFunction **m*)
    Create list of pointers to method arguments for TMethod m.

void **CreateListOfTypes**( )
    Create list of pointers to datatype (typedef) definitions. This
    function is called by TROOT::GetListOfTypes().

void* **GetInterfaceMethod**(TClass **cl,* Text_t **method,* Text_t **params*)
    Return pointer to CINT interface function for a method of a class with
    parameters params (params is a string of actual arguments, not formal
    ones). If the class is 0 the global function list will be searched.

void* **GetInterfaceMethodWithPrototype**(TClass **cl,* Text_t **method,* Text_t **proto*)
    Return pointer to CINT interface function for a method of a class with
    a certain prototype, i.e. "char*,int,float". If the class is 0 the global
    function list will be searched.

void **Execute**(Text_t **function,* Text_t **params*)
    Execute a global function with arguments params.

void **Execute**(TObject **obj,* TClass **cl,* Text_t **method,* Text_t **params*)
    Execute a method from class cl with arguments params.

void **Execute**(TObject **obj,* TClass **cl,* TMethod **method,* TObjArray **params*)
    Execute a method from class cl with the arguments in array params
    (params[0] ... params[n] = array of TObjString parameters).

Int_t **ExecuteMacro**(const char **filename*)
    Execute a CINT macro.

const char* **TypeName**(const char **typeDesc*)
    Return the absolute type of typeDesc.
    E.g.: typeDesc = "class TNamed**", returns "TNamed".
    You need to use the result immediately before it is being overwritten.

void* **FindObject**(char **item,* G__ClassInfo **type,* void ***prevObj,*void ***assocPtr*)
    Static function called by CINT when it finds an un-indentified object.

```
This function tries to find the UO in the ROOT files, directories, etc.
This functions has been registered by the TCint ctor.
```

void **UpdateAllCanvases**( )

```
Update all canvases at end the terminal input command.
```

Author: Fons Rademakers 01/03/96
Last update: 1.03/05 13/10/97 08.26.09 by René Brun

TClass

class **TClass** : public TDictionary

private:

class TMethod*	**GetClassMethod**(Long_t *faddr*)

public:

class TClass	**TClass**( )
class TClass	**TClass**(Text_t* *name,* Version_t *cversion,* Text_t* *dfil = 0,* Text_t* *ifil = 0,* Int_t *dl = 0,* Int_t *il = 0,* struct IconDesc_t* *desc = 0*)
class TClass	**TClass**(class TClass&)
void	**~TClass**( )
void	**AddInstance**(Bool_t *heap = kFALSE*)
virtual void	**Browse**(class TBrowser* *b*)
void	**BuildRealData**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
void	**Destructor**(void* *obj,* Bool_t *dtorOnly = kFALSE*)
virtual void	**Draw**(Option_t* *option*)
Text_t*	**EscapeChars**(Text_t* *text*)
class TClass*	**GetBaseClass**(Text_t* *classname*)
class TClass*	**GetBaseClass**(class TClass* *cl*)
class TClass*	**GetBaseDataMember**(Text_t* *datamember*)
class G__ClassInfo*	**GetClassInfo**( )
Version_t	**GetClassVersion**( )
class TDataMember*	**GetDataMember**(Text_t* *datamember*)
Short_t	**GetDeclFileLine**( )
const char*	**GetDeclFileName**( )
UInt_t	**GetHeapInstanceCount**( )
class TIcon*	**GetIcon**( )
struct IconDesc_t*	**GetIconDesc**( )
Short_t	**GetImplFileLine**( )
const char*	**GetImplFileName**( )
UInt_t	**GetInstanceCount**( )
class TList*	**GetListOfBases**( )
class TList*	**GetListOfDataMembers**( )
class TList*	**GetListOfMethods**( )
class TList*	**GetListOfRealData**( )
void	**GetMenuItems**(class TList* *listitems*)
class TMethod*	**GetMethod**(Text_t* *method,* Text_t* *params*)
class TMethod*	**GetMethodAny**(Text_t* *method*)
class TMethod*	**GetMethodWithPrototype**(Text_t* *method,* Text_t* *proto*)
virtual const char*	**GetName**( )
Int_t	**GetNdata**( )
Int_t	**GetNmethods**( )
virtual const char*	**GetTitle**( )
virtual ULong_t	**Hash**( )
virtual Bool_t	**InheritsFrom**(class TClass* *cl*)
virtual Bool_t	**InheritsFrom**(Text_t* *cl*)
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsFolder**( )
class TClass*	**Load**(class TBuffer& *b*)
void*	**New**( )
virtual Long_t	**Property**( )
void	**ResetInstanceCount**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
Int_t	**Size**( )

void	**Store**(class TBuffer& *b*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TString	**fName**	*name of class*
class TList*	**fRealData**	*linked list for persistent members including base classes*
class TList*	**fBase**	*linked list for base classes*
class TList*	**fData**	*linked list for data members*
class TList*	**fMethod**	*linked list for methods*
const char*	**fDeclFileName**	*name of class declaration file*
const char*	**fImplFileName**	*name of class implementation file*
Short_t	**fDeclFileLine**	*line of class declaration*
Short_t	**fImplFileLine**	*line of class implementation*
UInt_t	**fInstanceCount**	*number of instances of this class*
UInt_t	**fOnHeap**	*number of instances on heap*
Version_t	**fClassVersion**	*Class version Identifier*
class TIcon*	**fIcon**	*Class icon*
struct IconDesc_t*	**fIconDesc**	*Icon description*
class G__ClassInfo*	**fClassInfo**	*pointer to CINT class info class*

Class Description

```
The ROOT global object gROOT contains a list of all defined
classes. This list is build when a reference to a class dictionary
is made. When this happens, the static "class"::Dictionary()
function is called to create a TClass object describing the
class. The Dictionary() function is defined in the ClassDef
macro and stored (at program startup or library load time) together
with the class name in the TClassTable singleton object.
For a description of all dictionary classes see TDictionary.
```

TClass( ) : **TDictionary**( )

```
Default ctor.
```

TClass(const Text_t **name,* Version_t *cversion,*const Text_t *dfil, const Text_t *ifil, Int_t dl, Int_t il,IconDesc_t **desc*): **TDictionary**( )

```
Create a TClass object. This object contains the full dictionary
of a class. It has list to baseclasses, datamembers and methods.
```

~**TClass**( )

```
TClass dtor. Deletes all list that might have been created.
```

void **Browse**(TBrowser **b*)

```
This method is called by a browser to get the class information.
```

void **BuildRealData**( )

```
Build a full list of persistent data members.
Scans the list of all data members in the class itself and also
in all base classes. For each persistent data member, inserts a
TRealData object in the list fRealData.
```

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

void **Draw**(Option_t **option*)

```
Draw detailed class inheritance structure.
If a class B inherits from a class A, the description of B is drawn
```

```
on the right side of the description of A.
Member functions overridden by B are shown in class A with a blue line
erasing the corresponding member function
```

Text_t* **EscapeChars**(Text_t **text*)

```
Introduce an escape character (@) in front of a special chars.
You need to use the result immediately before it is being overwritten.
```

TClass* **GetBaseClass**(const Text_t **classname*)

```
Return pointer to the base class "classname". Returns 0 in case
"classname" is not a base class. Takes care of multiple inheritance.
```

TClass* **GetBaseClass**(const TClass **cl*)

```
Return pointer to the base class "cl". Returns 0 in case "cl"
is not a base class. Takes care of multiple inheritance.
```

TClass* **GetBaseDataMember**(const Text_t **datamember*)

```
Return pointer to (base) class that contains datamember.
```

TDataMember* **GetDataMember**(Text_t **datamember*)

```
Return pointer to datamember object with name "datamember".
```

TIcon* **GetIcon**( )

```
Get the icon belonging to this class. Used by the TBrowser.
```

TList* **GetListOfBases**( )

```
Return list containing the TBaseClass(es) of a class.
```

TList* **GetListOfDataMembers**( )

```
Return list containing the TDataMembers of a class.
```

TList* **GetListOfMethods**( )

```
Return list containing the TMethods of a class.
```

void **GetMenuItems**(TList **list*)

```
Returns list of methods accessible by context menu.
```

TMethod* **GetMethodAny**(const Text_t **method*)

```
Return pointer to method without looking at parameters.
Does not look in (possible) base classes.
```

TMethod* **GetMethod**(const Text_t **method,* const Text_t **params*)

```
Find the best method (if there is one) matching the parameters.
The params string must contain argument values, like "3189, \"aap\", 1.3".
The function invokes GetClassMethod to search for a possible method
in the class itself or in its base classes. Returns 0 in case method
is not found.
```

TMethod* **GetMethodWithPrototype**(const Text_t **method,* const Text_t **proto*)

```
Find the method with a given prototype. The proto string must be of the
form: "char*,int,double". Returns 0 in case method is not found.
```

TMethod* **GetClassMethod**(Long_t *faddr*)

```
Look for a method in this class that has the interface function
address faddr.
```

const char* **GetTitle**( ) const

```
Return the description of the class.
```

Int_t **GetNdata**( )

```
Return the number of data members of this class
Note that in case the list of data members is not yet created, it will be done
by GetListOfDataMembers().
```

Int_t **GetNmethods**( )

```
Return the number of methods of this class
Note that in case the list of methods is not yet created, it will be done
by GetListOfMethods().
```

Bool_t **InheritsFrom**(const Text_t **classname*)

```
Return kTRUE if this class inherits from a class with name "classname".
```

Bool_t **InheritsFrom**(const TClass **cl*)

```
Return kTRUE if this class inherits from class cl.
```

void* **New**( )

```
Return a pointer to a newly allocated object of this class.
The class must have a default constructor.
```

void **Destructor**(void **obj,* Bool_t *dtorOnly*)

```
Explicitely call destructor for object.
```

Int_t **Size**( ) const

```
Return size of object of this class.
```

Long_t **Property**( ) const

```
Get property description word. For meaning of bits see EProperty.
```

TClass* **Load**(TBuffer &*b*)

```
Load class description from I/O buffer and return class object.
```

void **Store**(TBuffer &*b*) const

```
Store class description on I/O buffer.
```

Author: René Brun 07/01/95
Last update: 1.03/02 03/09/97 12.52.11 by René Brun

TClassTable

class **TClassTable** : public TObject

private:

class TClassTable	**TClassTable**( )
struct ClassRec_t*	**FindElement**(char* *cname,* Bool_t *insert* = *kFALSE*)
void	**SortTable**( )

public:

class TClassTable	**TClassTable**(class TClassTable&)
void	**~TClassTable**( )
void	**Add**(char* *cname,* Version_t *id,* VoidFuncPtr_t *dict*)
class TClass*	**Class**( )
int	**Classes**( )
VoidFuncPtr_t	**GetDict**(char* *cname*)
Version_t	**GetID**(char* *cname*)
void	**Init**( )
virtual class TClass*	**IsA**( )
char*	**Next**( )
void	**Print**(Option_t* *option*)
void	**PrintTable**( )
void	**Remove**(char* *cname*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Terminate**( )

Data Members

private:

static struct ClassRec_t**	**fgTable**
static struct ClassRec_t**	**fgSortedTable**
static int	**fgSize**
static int	**fgTally**
static Bool_t	**fgSorted**
static int	**fgCursor**

Class Description

```
This class registers for all classes their name, id and dictionary
function in a hash table. Classes are automatically added by the
ctor of a special init class when a global of this init class is
initialized when the program starts (see the ClassImp macro).
```

TClassTable( )

```
TClassTable is a singleton (i.e. only one can exist per application).
```

~TClassTable( )

```
TClassTable singleton is deleted in Terminate().
```

void **Print**(Option_t *)

```
Print the class table.
```

int **Classes**( )

---

void **Add**(const char **cname,* Version_t *id,* VoidFuncPtr_t *dict*)

```
Add a class to the class table (this is a static function).
```

void **Remove**(const char **cname*)

```
Remove a class from the class table. This happens when a shared library
```

is unloaded (i.e. the dtor's of the global init objects are called).

ClassRec_t* **FindElement**(const char **cname,* Bool_t *insert*)

Find a class by name in the class table (using hash of name). Returns 0 if the class is not in the table. Unless arguments insert is true in which case a new entry is created and returned.

Version_t **GetID**(const char **cname*)

Returns the ID of a class.

VoidFuncPtr_t **GetDict**(const char **cname*)

Given the class name returns the Dictionary() function of a class (uses hash of name).

char* **Next**( )

Returns next class from sorted class table.

void **PrintTable**( )

Print the class table. Before printing the table is sorted alphabetically.

void **SortTable**( )

Sort the class table by ascending class ID's.

void **Terminate**( )

Deletes the class table (this static class function calls the dtor).

Author: Fons Rademakers 11/08/95
Last update: 1.01/05 11/06/97 20.57.37 by Fons Rademakers

TClonesArray

class **TClonesArray** : public TObjArray

public:

class TClonesArray	**TClonesArray**( )
class TClonesArray	**TClonesArray**(Text_t* *classname,* Int_t *size = 1000,* Bool_t *call_dtor = kTRUE*)
class TClonesArray	**TClonesArray**(class TClonesArray& *a*)
void	**~TClonesArray**( )
void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
void	**AddAt**(class TObject* *obj,* Int_t *idx*)
void	**AddAtAndExpand**(class TObject* *obj,* Int_t *idx*)
Int_t	**AddAtFree**(class TObject* *obj*)
void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
void	**AddFirst**(class TObject* *obj*)
void	**AddLast**(class TObject* *obj*)
class TClass*	**Class**( )
void	**Compress**( )
void	**Delete**(Option_t* *option*)
void	**Expand**(Int_t *newSize*)
void	**ExpandCreate**(Int_t *n*)
class TClass*	**GetClass**( )
virtual class TClass*	**IsA**( )
class TObject*&	**operator[]**(Int_t *i*)
class TObject*	**Remove**(class TObject* *obj*)
void	**RemoveAt**(Int_t *idx*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Sort**(Int_t *upto = kMaxInt*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TClass*	**fClass**	*Pointer to the class*
class TMethodCall*	**fMethodCall**	*Destructor method call environment*
class TObjArray*	**fKeep**	*Saved copies of pointers to objects*

Class Description

```
An array of clone (identical) objects. Memory for the objects
stored in the array is allocated only once in the lifetime of the
clones array. All objects must be of the same class. For the rest
this class has the same properties as TObjArray.

To reduce the very large number of new and delete calls in large
loops like this (O(100000) x O(10000) times new/delete):

TObjArray a(10000);
while (TEvent *ev = (TEvent *)next()) {      // O(100000) events
   for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
      a[i] = new TTrack(x,y,z,...);
      ...
      ...
   }
   ...
   a.Delete();
}

One better uses a TClonesArray which reduces the number of
```

```
new/delete calls to only O(10000):

TCloneArray a("TTrack", 10000);
while (TEvent *ev = (TEvent *)next()) {       // O(100000) events
   for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
      new(a[i]) TTrack(x,y,z,...);
      ...
      ...
   }
   ...
   a.Delete();
}

Considering that a new/delete costs about 70 mus, O(10^9)
new/deletes will save about 19 hours.
```

TClonesArray(Text_t **classname,* Int_t *s,* Bool_t *call_dtor*) : **TObjArray**(s)

```
Create an array of clone objects of classname. If the destructor of
the clone objects is a no-op (i.e. does not free any memory), you can
set call_dtor = kFALSE to save the overhead of calling the dtor.
```

~**TClonesArray**( )

```
Delete a clones array.
```

void **Compress**( )

```
Remove empty slots from array.
```

void **Delete**(Option_t **option*)

```
Remove all objects from the array AND delete all heap based objects.
```

void **Expand**(Int_t *newSize*)

```
Expand or shrink the array to newSize elements.
```

void **ExpandCreate**(Int_t *n*)

```
Expand or shrink the array to n elements and pre-allocate space for
the clones objects. If n is less than the current size the array
is shrinked and the allocated space is freed.
```

void **RemoveAt**(Int_t *idx*)

```
Remove object at index idx.
```

TObject* **Remove**(TObject **obj*)

```
Remove object from array.
```

void **Sort**(Int_t *upto*)

```
If objects in array are sortable (i.e. IsSortable() returns true
for all objects) then sort array.
```

void **Streamer**(TBuffer &*b*)

```
Write all objects in array to the I/O buffer. ATTENTION: empty slots
are also stored (using one byte per slot). If you don't want this
use a TOrdCollection or TList.
```

Author: René Brun 11/02/96
Last update: 1.03/06 27/10/97 16.23.02 by Fons Rademakers

TCollection

class **TCollection** : public TObject

public:

void	**~TCollection**( )
virtual void	**Add**(class TObject* *obj*)
virtual void	**AddAll**(class TCollection* *col*)
void	**AddVector**(class TObject* *obj1, int*)
Bool_t	**AssertClass**(class TClass* *cl*)
virtual void	**Browse**(class TBrowser* *b*)
Int_t	**Capacity**( )
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
Bool_t	**Contains**(class TObject* *obj*)
Bool_t	**Contains**(Text_t* *name*)
virtual void	**Delete**(Option_t* *option*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**Dump**( )
void	**EmptyGarbageCollection**( )
virtual class TObject*	**FindObject**(class TObject* *obj*)
virtual class TObject*	**FindObject**(Text_t* *name*)
void	**GarbageCollect**(class TObject* *obj*)
class TCollection*	**GetCurrentCollection**( )
class TObject*	**GetParent**( )
virtual Int_t	**GetSize**( )
virtual Int_t	**GrowBy**(Int_t *delta*)
virtual class TClass*	**IsA**( )
Bool_t	**IsArgNull**(char* *where,* class TObject* *obj*)
Bool_t	**IsEmpty**( )
virtual Bool_t	**IsFolder**( )
virtual void	**ls**(Option_t* *option*)
virtual class TIterator*	**MakeIterator**(Bool_t *dir = kIterForward*)
virtual class TIterator*	**MakeReverseIterator**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**RemoveAll**(class TCollection* *col*)
void	**RemoveAll**( )
void	**SetCurrentCollection**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**StartGarbageCollection**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Write**(Text_t* *name = 0,* Int_t *bufsize = 0*)

Data Members

private:

static class TCollection*	**fgCurrentCollection**	*used by macro ForEach*
static class TObjectTable*	**fgGarbageCollection**	*used by garbage collector*
static Bool_t	**fgEmptyingGarbage**	*used by garbage collector*
static Int_t	**fgGarbageStack**	*used by garbage collector*

protected:

class TObject*	**fParent**	*parent object of collection*
Int_t	**fSize**	*number of elements in collection*

public:

static const enum TObject:: **kInitCapacity**
static const enum TObject:: **kInitHashTableCapacity**

See Also: THashTable, TMap, TSeqCollection

Class Description

```
Collection abstract base class. This class describes the base
protocol all collection classes have to implement. The ROOT
collection classes always store pointers to objects that inherit
from TObject. They never adopt the objects. Therefore, it is the
user's responsability to take care of deleting the actual objects
once they are not needed anymore. In exceptional cases, when the
user is 100% sure nothing else is referencing the objects in the
collection, one can delete all objects and the collection at the
same time using the Delete() function.

Collections can be iterated using an iterator object (see
TIterator). Depending on the concrete collection class there may be
some additional methods of iterating. See the repective classes.

TCollection inherits from TObject since we want to be able to have
collections of collections.

In a later release the collections may become templatized.
```

void **AddAll**(TCollection **col*)

```
Add all objects from collection col to this collection.
```

void **AddVector**(TObject ***va_**(*obj1*), ...)

```
Add all arguments to this collection.
```

Bool_t **AssertClass**(TClass **cl*) const

```
Make sure all objects in this collection inherit from class cl.
```

void **Browse**(TBrowser **b*)

```
Browse this collection (called by TBrowser).
```

void **Draw**(Option_t **option*)

```
Draw all objects in this collection.
```

void **Dump**( )

```
Dump all objects in this collection.
```

TObject* **FindObject**(const Text_t **name*) const

```
Find an object in this collection using its name. Requires a sequential
scan till the object has been found. Returns 0 if object with specified
name is not found.
```

TObject* **FindObject**(TObject **obj*) const

```
Find an object in this collection using the object's IsEqual()
member function. Requires a sequential scan till the object has
been found. Returns 0 if object is not found.
Typically this function is overridden by a more efficient version
in concrete collection classes (e.g. THashTable).
```

Int_t **GrowBy**(Int_t *delta*) const

```
Increase the collection's capacity by delta slots.
```

Bool_t **IsArgNull**(const char *where, TObject **obj*) const

```
Returns true if object is a null pointer.
```

void **ls**(Option_t **option*)

```
List (ls) all objects in this collection.
```

void **Paint**(Option_t **option*)

```
Paint all objects in this collection.
```

void **Print**(Option_t **option*)

```
Print all objects in this collection.
```

void **RecursiveRemove**(TObject **obj*)

```
Remove object from this collection and recursively remove the object
from all other objects (and collections).
```

void **RemoveAll**(TCollection **col*)

```
Remove all objects in collection col from this collection.
```

void **Streamer**(TBuffer &*b*)

```
Stream all objects in the collection to or from the I/O buffer.
```

void **Write**(const Text_t **name,* Int_t *bsize*)

```
Write all objects in this collection. By default all objects in
the collection are written individually (each object gets its
own key). To write all objects using one key specify a name and set
bsize == 1.
```

void **GarbageCollect**(TObject **obj*)

Author: Fons Rademakers 13/08/95
Last update: 1.03/04 30/09/97 18.04.38 by Fons Rademakers

TColor

class **TColor** : public TNamed

public:

class TColor	**TColor**(Int_t *color,* Float_t *r,* Float_t *g,* Float_t *b,* Text_t* *name*)
class TColor	**TColor**( )
class TColor	**TColor**(class TColor& *color*)
void	**~TColor**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *color*)
virtual void	**DisplayColorTable**( )
Float_t	**GetBlue**( )
Float_t	**GetGreen**( )
virtual void	**GetHLS**(Float_t& *h,* Float_t& *l,* Float_t& *s*)
Float_t	**GetHue**( )
Float_t	**GetLight**( )
Int_t	**GetNumber**( )
Float_t	**GetRed**( )
virtual void	**GetRGB**(Float_t& *r,* Float_t& *g,* Float_t& *b*)
Float_t	**GetSaturation**( )
virtual void	**HLStoRGB**(Float_t *h,* Float_t *l,* Float_t *s,* Float_t& *r,* Float_t& *g,* Float_t& *b*)
Float_t	**HLStoRGB1**(Float_t *rn1,* Float_t *rn2,* Float_t *huei*)
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**RGBtoHLS**(Float_t *r,* Float_t *g,* Float_t *b,* Float_t& *h,* Float_t& *l,* Float_t& *s*)
virtual void	**SetNumber**(Int_t *number*)
virtual void	**SetRGB**(Float_t *r,* Float_t *g,* Float_t *b*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Int_t	**fNumber**	*color number identifier*
Float_t	**fRed**	*Fraction of Red*
Float_t	**fGreen**	*Fraction of Green*
Float_t	**fBlue**	*Fraction of Blue*
Float_t	**fHue**	*Hue*
Float_t	**fLight**	*Light*
Float_t	**fSaturation**	*Saturation*

Class Description

```
C O L O R class
At initialisation time, a table of colors is generated when
the first Canvas constructor is called.
This linked list can be accessed from the ROOT object
   (see TROOT::GetListOfColors).
When a color is defined, two "companion" colors are also defined:
 - the dark version (color_index + 100)
 - the bright version (color_index + 150)
The dark and bright color are used to give 3-D effects when drawing
various boxes (see TWbox, TPave, TPaveText, TPaveLabel,etc).

This is the list of currently supported basic colors.
```

(here dark and bright colors are not shown).

TColor( ): **TNamed**( )
    Color default constructor

TColor(Int_t *color,* Float_t *r,* Float_t *g,* Float_t *b,* Text_t **name*):**TNamed**(*name,* "")
    Color normal constructor
    -
    Initialize a color structure.
    Compute the RGB and HLS parameters

~**TColor**( )
    Color default destructor

void **Copy**(TObject &*obj*)
    Copy this color to color

void **DisplayColorTable**( )
    Display Color Table in the pad

void **HLStoRGB**(Float_t *hue,* Float_t *light,* Float_t *satur,* Float_t &*r,* Float_t &*g,* Float_t &*b*)
    Compute HLS from RGB
    - See HIGZ routine IGHTOR

Float_t **HLStoRGB1**(Float_t *rn1,* Float_t *rn2,* Float_t *huei*)
    Auxiliary to HLStoRGB
    See HIGZ routine IGHR01

void **ls**(Option_t **option*)
    List this color with its attributes

void **Print**(Option_t **option*)
    Dump this color with its attributes

void **RGBtoHLS**(Float_t *r,* Float_t *g,* Float_t *b,* Float_t &*hue,* Float_t &*light,* Float_t &*satur*)
    Compute HLS from RGB

void **SetRGB**(Float_t *r,* Float_t *g,* Float_t *b*)
    Initialize this color and its assosiated colors

Author: René Brun 12/12/94
Last update: 1.03/02 22/08/97 19.34.12 by Valery Fine(fine@mail.cern.ch)

TComponent

class **TComponent** : public TCallbackObject

private:

void	**WidgetDestroyedCallback**(Widget *w,* XtPointer *callData,* XtPointer *clientData*)

protected:

virtual void	**AfterRealizeHook**( )

public:

void	**~TComponent**( )
virtual void	**Activate**( )
Widget	**BaseWidget**( )
class TClass*	**Class**( )
virtual const Text_t*	**ClassName**( )
virtual void	**Deactivate**( )
virtual Text_t*	**GetData**( )
void	**GetResources**(XtResourceList *resources,* int *numResources*)
void	**GetValues**(Arg* *arglist,* Int_t *numArgs*)
virtual void	**GetWidgetUserData**(void** *data*)
void	**GetWidgetUserData**(Widget *w,* void** *data*)
virtual void	**Hide**( )
void	**InstallDestroyHandler**( )
Bool_t	**Instantiated**( )
virtual class TClass*	**IsA**( )
const Text_t*	**Name**( )
virtual Boolean	**OkToQuit**( )
virtual Widget	**operator Widget**( )
void	**RemoveDestroyHandler**( )
virtual void	**SetData**(Text_t* *data*)
void	**SetDefaultResources**(Widget *parent,* String* *resourceSpec*)
void	**SetValues**(Arg* *arglist,* Int_t *numArgs*)
void	**SetWidgetUserData**(Widget *w,* void* *data*)
virtual void	**SetWidgetUserData**(void* *data*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**WidgetDestroyed**( )

Data Members

private:

XrmDatabase	**fXrm_db**

protected:

Bool_t	**fInstantiated**	*true if widget has been created*
Bool_t	**fActivated**	*false if widget is inactive*
Text_t*	**fName**	*name of this GUI component*
Widget	**fBaseWidget**	*MOTIF widget*
Text_t*	**fData**	*GUI element data - always in Text_t format*

public:

static const char *const	**fgDeleteCallback**

See Also: TCheckBox, TDialogItem, TDialogManager, THelpDialog, TMenuItem, TMotifApplication, TMotifContextMenu, TMotifControlBar, TMotifDialog, TRepeatButton, TSimpleWindow

Class Description

`TComponent`

Abstract class derived from TCallbackObject which allows any components to receive and register callbacks.

const Text_t* **Name**( ) const

Returns the name of the component instance.

Widget **BaseWidget**( )

Returns a widget belonging to this TComponent object.

void **SetData**(Text_t **data*)

Sets the data belonging to object. For convenience reasons data is stored as Text_t* - you can always convert it to any other format in that case. Each component has its own data field, independent of widgets' data. It transfers a data from/to widget when it's necesarry and updates its own data field.

void **Activate**( )

Sets widget and component active.

void **Deactivate**( )

Sets widget and component inactive.

void **Show**( )

By default, just manage the fBaseWidget

void **Hide**( )

By default, just unmanage the fBaseWidget

void **GetResources**(const XtResourceList *resources,*const int *numResources*)

Gets application resources.

void **SetValues**(Arg **arglist,* Int_t *numArgs*)

Convevience functions for Setting values of belonging widget. (convenient XtSetValues()).

void **SetWidgetUserData**(void **data*)

Each Widget contains a data field pointer - so called XmNuserData resource. In widgets connected with TComponent's children objects we usually use this field to store the pointer to the object. Thus, having a Widget we can easily get the object to which this widget belongs to.
This method is convenience function to set XmNuserData of the widget belonging to this component to specified pointer.

void **GetWidgetUserData**(void ***data*)

Each Widget contains a data field pointer - so called XmNuserData resource. In widgets connected with TComponent's children objects we usually use this field to store the pointer to the object. Thus, having a Widget we can easily get the object to which this widget belongs to.
This method is convenience function to get XmNuserData of the widget belonging to this component.

void **GetWidgetUserData**(Widget *w,* void ***data*)

Each Widget contains a data field pointer - so called XmNuserData resource. In widgets connected with TComponent's children objects we usually use this field to store the pointer to the object. Thus, having a Widget we can easily get the object to which this widget belongs to.
This method is convenience function to get XmNuserData of specified widget Because of the fact, that this is static method of a class it can be called from inside of other staic methods - i.e. Callback Routines.
This is the most frequent use of this method - when you get a widget pointer in a callback subroutine, you call this method to get belonging component.

void **SetWidgetUserData**(Widget *w,* void **data*)

```
Each Widget contains a data field pointer - so called XmNuserData resource.
In widgets connected with TComponent's children objects we usually use this
field to store the pointer to the object. Thus, having a Widget we can
easilly get the object to which this widget belongs to.
This method is convenience function to set XmNuserData of specified widget
Because of the fact, that this is static method of a class it can be called
from inside of other staic methods - i.e. Callback Routines.
```

Author: Fons Rademakers 30/03/95
Last update: 1.03/04 19/09/97 16.44.31 by Fons Rademakers

TCONE

class **TCONE** : public TTUBE

public:

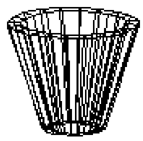
class TCONE	**TCONE**( )
class TCONE	**TCONE**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dz,* Float_t *rmin1,* Float_t *rmax1,* Float_t *rmin2,* Float_t *rmax2*)
class TCONE	**TCONE**(class TCONE&)
void	**~TCONE**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fRmin2**	*inside radius at the high z limit*
Float_t	**fRmax2**	*outside radius at the high z limit*

Class Description



```
CONE is a conical tube. It has 8 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dz         half-length in z
  - rmin1      inside radius at -DZ in z
  - rmax1      outside radius at -DZ in z
  - rmin2      inside radius at +DZ in z
  - rmax2      outside radius at +DZ in z
```

TCONE( )

```
CONE shape default constructor
```

TCONE(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dz,*Float_t rmin1, Float_t rmax1,Float_t rmin2, Float_t *rmax2*): **TTUBE**(*name, title, material, rmin1, rmax1, dz*)

```
CONE shape normal constructor
```

~TCONE( )

```
CONE shape default destructor
```

void **SetPoints**(Float_t **buff*)

```
Create CONE points
```

Author: Nenad Buncic 18/09/95
Last update: 1.01/07 26/06/97 19.02.04 by René Brun

TCONS

class **TCONS** : public TTUBS

public:

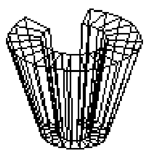
class TCONS	**TCONS**( )
class TCONS	**TCONS**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *rmin1,* Float_t *rmax1,* Float_t *rmin2,* Float_t *rmax2,* Float_t *dz,* Float_t *phi1,* Float_t *phi2*)
class TCONS	**TCONS**(class TCONS&)
void	**~TCONS**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fRmin2**	*inside radius at the high z limit*
Float_t	**fRmax2**	*outside radius at the high z limit*

Class Description



```
CONS is a segment of a conical tube. It has 10 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dz         half-length in z
  - rmin1      inside radius at -DZ in z
  - rmax1      outside radius at -DZ in z
  - rmin2      inside radius at +DZ in z
  - rmax2      outside radius at +DZ in z
  - phi1       starting angle of the segment
  - phi2       ending angle of the segment

NOTE: phi1 should be smaller than phi2. If this is not the case,
    the system adds 360 degrees to phi2.
```

TCONS( )

    CONS shape default constructor

TCONS(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rmin1,* Float_t *rmax1,* Float_t *rmin2,* Float_t *rmax2,*Float_t *dz,* Float_t *phi1,* Float_t *phi2*) : **TTUBS**(*name, title, material, rmin1, rmax1, dz, phi1,* phi2)

    CONS shape normal constructor

~TCONS( )

    CONS shape default destructor

void **SetPoints**(Float_t **buff*)

    Create CONS points

Author: Nenad Buncic 18/09/95

Last update: 1.00/07 27/03/97 17.52.07 by René Brun

TContextMenu

class **TContextMenu** : public TNamed

private:

class TContextMenu	**TContextMenu**( )

protected:

virtual void	**DisplayPopUp**(Int_t *x,* Int_t *y*)

public:

class TContextMenu	**TContextMenu**(char* *name,* char* *title* = Context sensitive popup *menu*)
class TContextMenu	**TContextMenu**(class TContextMenu&)
void	**~TContextMenu**( )
virtual void	**Action**(class TObject* *object,* class TMethod* *method*)
class TClass*	**Class**( )
const Char_t*	**ClassName**( )
virtual Char_t*	**CreateArgumentTitle**(class TMethodArg* *argument*)
virtual Char_t*	**CreateDialogTitle**(class TObject* *object,* class TMethod* *method*)
virtual Char_t*	**CreatePopupTitle**(class TObject* *object*)
virtual void	**Execute**(class TObject* *object,* class TMethod* *method,* class TObjArray* *params*)
virtual void	**Execute**(class TObject* *object,* class TMethod* *method,* Char_t* *params*)
virtual class TBrowser*	**GetBrowser**( )
virtual class TContextMenuImp*	**GetContextMenuImp**( )
virtual class TVirtualPad*	**GetSelectedCanvas**( )
virtual class TMethod*	**GetSelectedMethod**( )
virtual class TObject*	**GetSelectedObject**( )
virtual class TVirtualPad*	**GetSelectedPad**( )
virtual class TClass*	**IsA**( )
virtual void	**Popup**(Int_t *x,* Int_t *y,* class TObject* *obj,* class TVirtualPad* *c = 0,* class TVirtualPad* *p = 0*)
virtual void	**Popup**(Int_t *x,* Int_t *y,* class TObject* *obj,* class TBrowser* *b*)
virtual void	**SetBrowser**(class TBrowser* *b*)
virtual void	**SetCanvas**(class TVirtualPad* *c*)
virtual void	**SetMethod**(class TMethod* *m*)
virtual void	**SetObject**(class TObject* *o*)
virtual void	**SetPad**(class TVirtualPad* *p*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

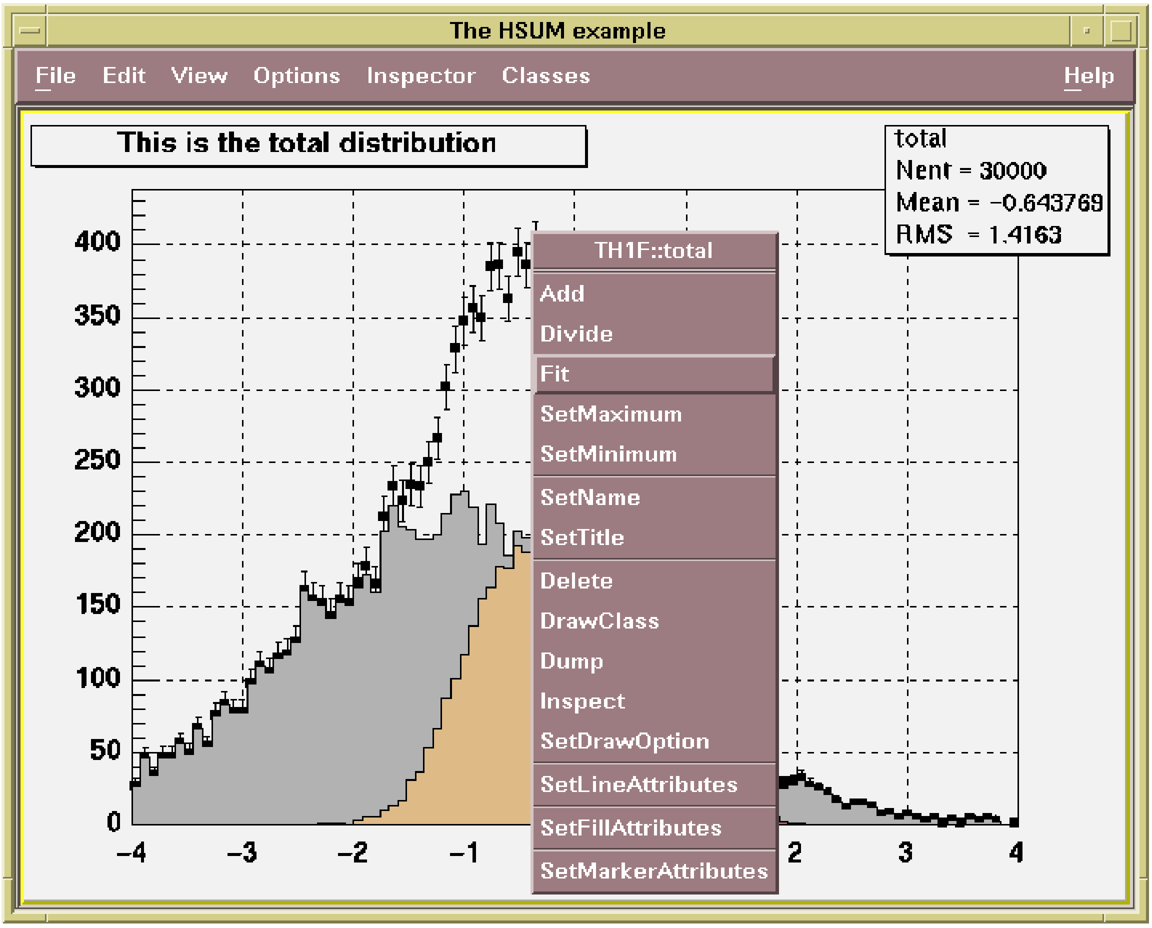
protected:

class TContextMenuImp*	**fContextMenuImp**	*Context menu system specific implementation*
class TMethod*	**fSelectedMethod**	*selected method*
class TObject*	**fSelectedObject**	*selected object*
class TVirtualPad*	**fSelectedCanvas**	*selected canvas (if exist*)
class TVirtualPad*	**fSelectedPad**	*selected pad (if exist*)
class TBrowser*	**fBrowser**	*selected browser (if exist*)

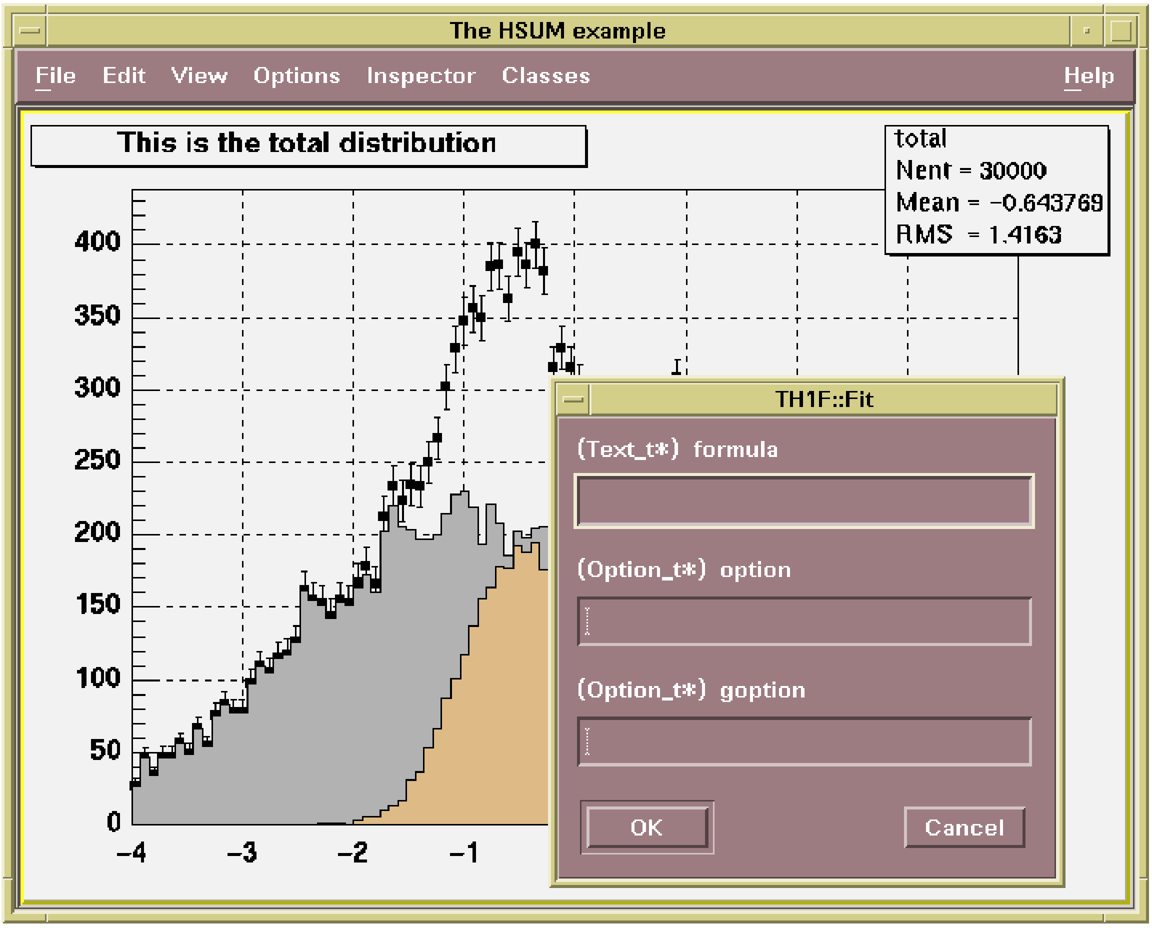
Class Description

```
TContextMenu

This class provides an interface to  context sensitive popup menus.
These menus pop up when the user hits  the right mouse button,  and
are destroyed when the menu pops downs.

Context Menus are automatically generated by ROOT using the following
convention:
```

```
If the string // *MENU* is found in the comment field of a member function
This function will be added to the list of items in the menu.

The picture below shows a canvas with a pop-up menu.
```




```
The picture below shows a canvas with a pop-up menu and a dialog box.
```



void **Popup**(Int_t *x,* Int_t *y,* TObject **obj,* TBrowser **b*)

Author: Nenad Buncic 08/02/96
Last update: 1.03/05 01/10/97 22.30.21 by Fons Rademakers

TContextMenuImp

class **TContextMenuImp**

public:

class TContextMenuImp	**TContextMenuImp**(class TContextMenu* *c = 0*)
class TContextMenuImp	**TContextMenuImp**(class TContextMenuImp&)
void	**~TContextMenuImp**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual void	**Dialog**(class TObject* *object,* class TMethod* *method*)
virtual void	**DisplayPopup**(Int_t *x,* Int_t *y*)
virtual class TContextMenu*	**GetContextMenu**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TContextMenu*	**fContextMenu**	*TContextMenu associated with this implementation*

See Also: TMotifContextMenu

Class Description

```
TContextMenuImp

This class provides an interface to GUI independent
context sensitive popup menus.
```

Author: Nenad Buncic 08/02/96
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TControlBar

class **TControlBar** : public TControlBarButton

private:

virtual void	**Create**( )
void	**Initialize**( )

public:

class TControlBar	**TControlBar**(Text_t* *orientation,* Text_t* *label,* Text_t* *hint,* Text_t* *type* = *button*)
class TControlBar	**TControlBar**( )
class TControlBar	**TControlBar**(class TControlBar&)
void	**~TControlBar**( )
void	**AddButton**(class TControlBarButton* *button*)
void	**AddButton**(Text_t* *label,* Text_t* *action,* Text_t* *hint,* Text_t* *type* = *button*)
void	**AddButton**(class TControlBarButton& *button*)
void	**AddControlBar**(class TControlBar& *controlBar*)
void	**AddControlBar**(class TControlBar* *controlBar*)
void	**AddSeparator**( )
class TClass*	**Class**( )
const Text_t*	**ClassName**( )
class TControlBarImp*	**GetControlBarImp**( )
class TList*	**GetListOfButtons**( )
Int_t	**GetNumberOfColumns**( )
Int_t	**GetNumberOfRows**( )
Int_t	**GetOrientation**( )
void	**Hide**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsContainer**( )
void	**SetNumberOfColumns**(Int_t *n*)
void	**SetNumberOfRows**(Int_t *n*)
void	**SetOrientation**(Text_t* *o*)
void	**SetOrientation**(Int_t *o*)
void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TControlBarImp*	**fControlBarImp**	*system specific implementation*
Int_t	**fOrientation**	*orientation*
class TList*	**fButtons**	*list of buttons*
Int_t	**fNoroc**	*number of rows or columns*

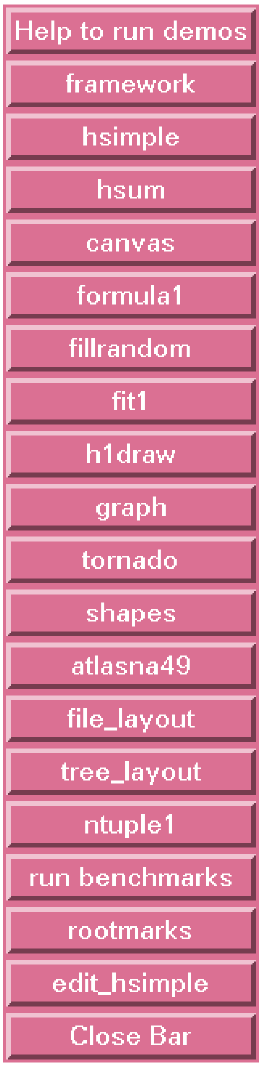
public:

static const enum TObject::	**kVertical**
static const enum TObject::	**kHorizontal**

Class Description

```
ControlBar is fully user configurable tool which provides fast
access to frequently used operations. User can choose between
buttons and drawnbuttons (let's say icons) and assign to them his
own actions (let's say ROOT or C++ commands).

The macro belows shows an example of controlbar.
To execute an item, click with the left mouse button.
To see the HELP of a button, click on the right mouse button.
```

```
{
gROOT.Reset("a");
TControlBar bar("vertical");
bar.AddButton("Help to run demos",".x demoshelp.C",
              "Explains how to run the demos");
bar.AddButton("framework",        ".x framework.C",
              "An Example of Object Oriented User Interface");
bar.AddButton("hsimple",          ".x hsimple.C",
              "An Example Creating Histograms/Ntuples on File");
bar.AddButton("hsum",             ".x hsum.C",
              "Filling histograms and some graphics options");
bar.AddButton("canvas",           ".x canvas.C",
              "Canvas and Pad Management");
bar.AddButton("formula1",         ".x formula1.C",
              "Simple Formula and Functions");
bar.AddButton("fillrandom",       ".x fillrandom.C",
              "Histograms with Random Numbers from a Function");
bar.AddButton("fit1",             ".x fit1.C",
              "A Simple Fitting Example");
bar.AddButton("h1draw",           ".x h1draw.C",
              "Drawing Options for 1D Histograms");
bar.AddButton("graph",            ".x graph.C",
              "Examples of a simple graph");
bar.AddButton("tornado",          ".x tornado.C",
              "Examples of 3-D PolyMarkers");
bar.AddButton("shapes",           ".x shapes.C",
              "The Geometry Shapes");
bar.AddButton("atlasna49",        ".x atlasna49.C",
              "Creating and Viewing Geometries");
bar.AddButton("file_layout",      ".x file.C",
              "The ROOT file format");
bar.AddButton("tree_layout",      ".x tree.C",
              "The Tree Data Structure");
bar.AddButton("ntuple1",          ".x ntuple1.C",
              "Ntuples and Selections");
bar.AddButton("run benchmarks",   ".x benchmarks.C",
              "Runs all the ROOT benchmarks");
bar.AddButton("rootmarks",        ".x rootmarks.C",
              "Prints an estimated ROOTMARKS for your machine");
bar.AddButton("edit_hsimple",     ".!ved hsimple.C &",
              "Invokes the text editor on file hsimple.C");
bar.AddButton("Close Bar",        "gROOT.Reset("a")",
              "Close ControlBar");
bar.Show();
gROOT.SaveContext();
}
```



void **Show**( )

Author: Nenad Buncic 20/02/96
Last update: 1.00/07 24/03/97 15.41.07 by Fons Rademakers

TControlBarButton

class **TControlBarButton** : public TNamed

public:

class TControlBarButton	**TControlBarButton**(Text_t* *label,* Text_t* *action,* Text_t* *hint,* Text_t* *type = button*)
class TControlBarButton	**TControlBarButton**( )
class TControlBarButton	**TControlBarButton**(class TControlBarButton&)
void	**~TControlBarButton**( )
virtual void	**Action**( )
class TClass*	**Class**( )
virtual void	**Create**( )
virtual Text_t*	**GetAction**( )
virtual Int_t	**GetType**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsContainer**( )
virtual void	**SetAction**(Text_t* *action*)
virtual void	**SetType**(Int_t *type*)
virtual void	**SetType**(Text_t* *type*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fType**	*button type*
Text_t*	**fAction**	*action to be executed*

See Also: TControlBar

Class Description

`TControlBarButton`

void **SetType**(Int_t *type*)

Author: Nenad Buncic 20/02/96
Last update: 1.00/06 23/03/97 11.21.50 by René Brun

TControlBarImp

class **TControlBarImp**

public:

class TControlBarImp	**TControlBarImp**(class TControlBar* *c = 0*)
class TControlBarImp	**TControlBarImp**(class TControlBarImp&)
void	**~TControlBarImp**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual void	**Create**( )
virtual class TControlBar*	**GetControlBar**( )
virtual void	**Hide**( )
virtual class TClass*	**IsA**( )
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TControlBar*	**fControlBar**	*TControlBar associated with this implementation*

See Also: TMotifControlBar

Class Description

```
TControlBarImp

ABC describing GUI independent control bar
```

Author: Nenad Buncic 20/02/96
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TCTUB

class **TCTUB** : public TTUBS

public:

class TCTUB	**TCTUB**( )
class TCTUB	**TCTUB**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *rmin,* Float_t *rmax,* Float_t *dz,* Float_t *phi1,* Float_t *phi2,* Float_t *coslx,* Float_t *cosly,* Float_t *coslz,* Float_t *coshx,* Float_t *coshy,* Float_t *coshz*)
class TCTUB	**TCTUB**(class TCTUB&)
void	**~TCTUB**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fCosLow**	*dir cosinus of surface cutting tube at low z*
Float_t	**fCosHigh**	*dir cosinus of surface cutting tube at high z*

Class Description

```
'CTUB' is a cut  tube with 11 parameters.  The  first 5 parameters
     are the same  as for the TUBS.  The  remaining 6 parameters
     are the director  cosines of the surfaces  cutting the tube
     respectively at the low and high Z values.

   - name       name of the shape
   - title      shape's title
   - material  (see TMaterial)
   - rmin       inside radius
   - rmax       outside radius
   - dz         half length in z
   - phi1       starting angle of the segment
   - phi2       ending angle of the segment
   - coslx      x dir cosinus at low z face
   - cosly      y dir cosinus at low z face
   - coslz      z dir cosinus at low z face
   - coshx      x dir cosinus at high z face
   - coshy      y dir cosinus at high z face
   - coshz      z dir cosinus at high z face
```

TCTUB( )

```
CTUB shape default constructor
```

TCTUB(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rmin,*Float_t rmax, Float_t dz, Float_t phi1, Float_t phi2,Float_t coslx, Float_t cosly, Float_t coslz,Float_t coshx, Float_t coshy, Float_t *coshz*): **TTUBS**(*name, title, material, rmin, rmax, dz, phi1, phi2*)

```
CTUB shape normal constructor
```

~TCTUB( )

```
CTUB shape default destructor
```

Author: René Brun 26/06/97
Last update: 1.01/07 26/06/97 10.43.13 by René Brun

TCut

class **TCut** : public TNamed

public:

class TCut	**TCut**(Text_t* *title*)
class TCut	**TCut**( )
class TCut	**TCut**(Text_t* *name,* Text_t* *title*)
class TCut	**TCut**(class TCut& *cut*)
void	**~TCut**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
char*	**operator const char***( )
class TCut&	**operator!**( )
class TCut&	**operator+=**(class TCut& *rhs*)
class TCut&	**operator+=**(char* *rhs*)
class TCut&	***operator*=**(class TCut& *rhs*)
class TCut&	***operator*=**(char* *rhs*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TCut

A specialized string object used for TTree selections.
A TCut object has a name and a title. It does not add any data
members compared to a TNamed. It only add a set of operators to
facilitate logical string concatenation. For example, assume
  cut1 = "x<1"  and cut2 = "y>2"
then
  cut1 && cut2 will be the string "(x<1)&&(y>2)"

Operators =, +=, +, -, !, &&, || overloaded.

Examples of use:
  Root > TCut c1 = "x<1"
  Root > TCut c2 = "y<0"
  Root > TCut c3 = c1&&c2
  Root > ntuple.Draw("x", c1)
  Root > ntuple.Draw("x", c1||"x>0")
  Root > ntuple.Draw("x", c1&&c2)
```

TCut(const TCut &*cut*) : **TNamed**(cut)

```
fName  = cut.fName;
fTitle = cut.fTitle;
```

~TCut( )

Author: René Brun 14/04/97
Last update: 1.00/10 25/04/97 13.28.16 by René Brun

TCutG

class **TCutG** : public TGraph

public:

class TCutG	**TCutG**( )
class TCutG	**TCutG**(Text_t* *name*, Int_t *n*, Float_t* *x*, Float_t* *y*)
class TCutG	**TCutG**(class TCutG&)
void	**~TCutG**( )
class TClass*	**Class**( )
virtual Double_t	**EvalInstance**(Int_t *instance*)
virtual const Text_t*	**GetVarX**( )
virtual const Text_t*	**GetVarY**( )
virtual class TClass*	**IsA**( )
virtual Int_t	**IsInside**(Float_t *x*, Float_t *y*)
virtual void	**SetTree**(class TTree* *tree*)
virtual void	**SetVarX**(Text_t* *varx*)
virtual void	**SetVarY**(Text_t* *vary*)
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TString	**fVarX**	*X variable*
class TString	**fVarY**	*Y variable*
class TTreeFormula*	**fFormulaX**	*!pointer to TreeFormula for X*
class TTreeFormula*	**fFormulaY**	*!pointer to TreeFormula for Y*
class TTree*	**fTree**	*!pointer to current tree*

Class Description

```
TCutG

A Graphical cut.
A TCutG object defines a closed polygon in a x,y plot.
It can be created via the graphics editor option "CutG"
or directly by invoking its constructor.
When it is created via the graphics editor, the TCutG object
is named "CUTG". It is recommended to immediatly change the name
by using the context menu item "SetName".                           //
When the graphics editor is used, the names of the variables X,Y
are automatically taken from the current pad title.
Example:
Assume a TTree object T and:
  Root > T.Draw("abs(fMomemtum)%fEtot")
the TCutG members fVarX, fVary will be set to:
  fVarx = fEtot
  fVary = abs(fMomemtum)

A graphical cut can be used in a TTree selection expression:
 Root > T.Draw("fEtot","cutg1")
 where "cutg1" is the name of an existing graphical cut.

Note that, as shown in the example above, a graphical cut may be
used in a selection expression when drawing TTrees expressions
of 1-d, 2-d or 3-dimensions.
The TTree expressions may or may not reference the same variables
than in the fVarX, fVarY of the graphical cut.
```

```
A Graphical cut may be drawn via TGraph::Draw.
It can be edited like a normal TGraph.

A Graphical cut may be saved to a file via TCutG::Write.            //

A TCutG object is added in the gROOT->GetListOfSpecials
```

Double_t **EvalInstance**(Int_t *instance*)

```
Evaluate graphical cut value for current event instance
```

Int_t **IsInside**(Float_t *x*, Float_t *y*)

```
*.         Function which returns 1 if point x,y lies inside the
*.              polygon defined by the graph points
*.                                0 otherwise
*.
*.     The loop is executed with the end-point coordinates of a
*.     line segment (X1,Y1)-(X2,Y2) and the Y-coordinate of a
*.     horizontal line.
*.     The counter inter is incremented if the line (X1,Y1)-(X2,Y2)
*.     intersects the horizontal line.
*.     In this case XINT is set to the X-coordinate of the
*.     intersection point.
*.     If inter is an odd number, then the point x,y is not within
*.     the polygon.
*.
*.         This routine is based on an original algorithm
*.         developed by R.Nierhaus.
*.
```

void **SetVarY**(const Text_t **vary*)

Author: René Brun 16/05/97
Last update: 1.01/03 18/05/97 22.07.21 by René Brun

TDataMember

class **TDataMember** : public TDictionary

public:

class TDataMember	**TDataMember**(class TDataMember&)
class TDataMember	**TDataMember**((*unknown*) G__DataMemberInfo* info = 0, class TClass* cl = 0)
void	**~TDataMember**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
Int_t	**GetArrayDim**( )
class TClass*	**GetClass**( )
class TDataType*	**GetDataType**( )
const char*	**GetFullTypeName**( )
Int_t	**GetMaxIndex**(Int_t *dim*)
virtual const char*	**GetName**( )
Int_t	**GetOffset**( )
class TList*	**GetOptions**( )
class TMethodCall*	**GetterMethod**( )
virtual const char*	**GetTitle**( )
const char*	**GetTypeName**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsaPointer**( )
Bool_t	**IsBasic**( )
Bool_t	**IsEnum**( )
Bool_t	**IsPersistent**( )
virtual Long_t	**Property**( )
class TMethodCall*	**SetterMethod**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

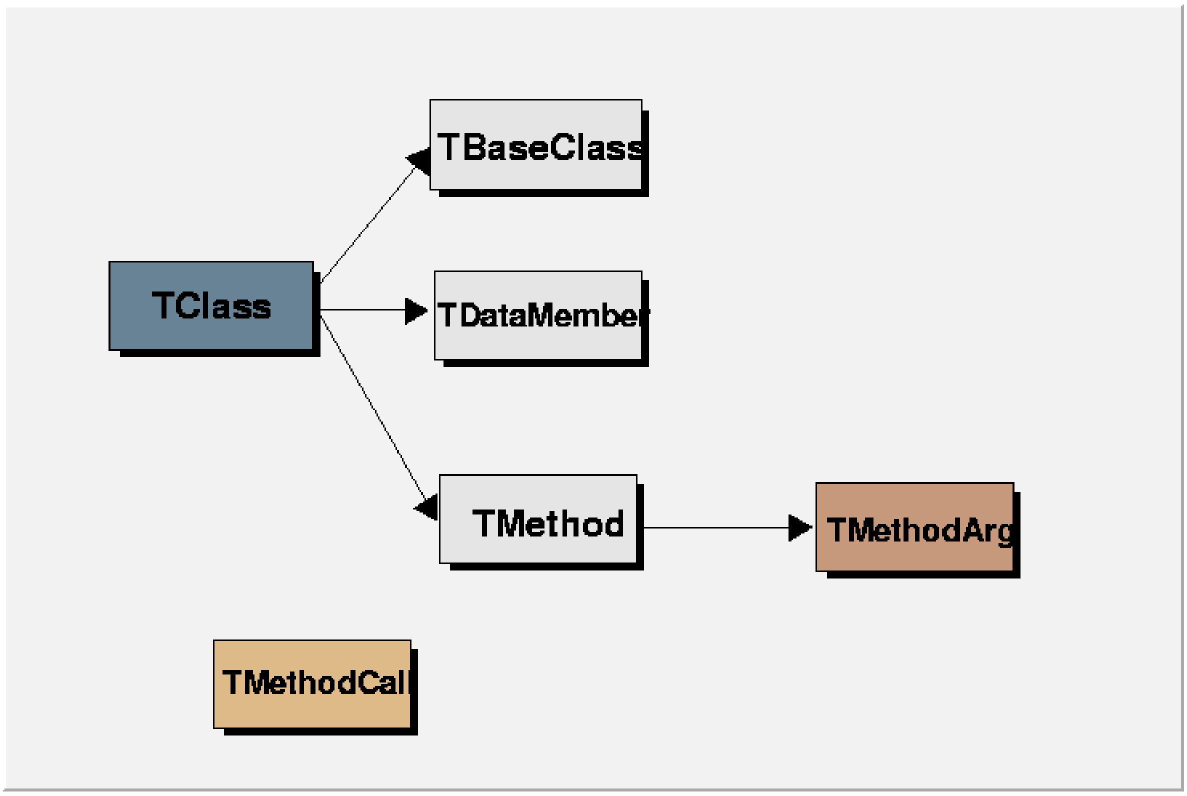
Data Members

private:

class G__DataMemberInfo*	**fInfo**	*pointer to CINT data member info*
class TClass*	**fClass**	*pointer to the class*
class TDataType*	**fDataType**	*pointer to data basic type descriptor*
class TMethodCall*	**fValueGetter**	*method that returns a value;*
class TMethodCall*	**fValueSetter**	*method which sets value;*
class TList*	**fOptions**	*list of possible values NULL=no restrictions;list of OptionLististItem values*

Class Description

```
TDataMember.

All ROOT classes may have an RTTI (run time type identification) support
added. The data is stored in so called DICTIONARY(look at TDictionary).
Information about a class is stored in  TClass.
This information may be obtained via the CINT api - see class TCint .
TClass has a list of TDataMember objects providing information about all
data members of described class.
```




```
TDataMember provides information about name of data member, its type,
a comment field string. It also tries to find a TMethodCall objects
responsible for getting/setting a value of it, and gives you pointers
to these methods. This gives you a unique possibility to access
protected and private (!) data members if only methods for doing that
are defined.
These methods could either be specified in a comment field, or found
out automatically by ROOT: here's an example:
suppose you have a class definition :

/************************************************************************

     class MyClass{
         private:
             Float_t fX1;
                 ...
         public:
             void    SetX1(Float_t x) {fX1 = x;};
             Float_t GetX1()          {return fX1;};
                 ...
     }
************************************************************************/
```

Look at the data member name and methods' names: a data member name has a prefix letter (f) and has a base name X1 . The methods for getting and setting this value has names which consist of string Get/Set and the same base name. This convention of naming data fields and methods which access them allows TDataMember find this methods by itself completely automatically. To make thid description complete, one should know, that names that are automatically recognized may be also: for data field : either fXXX or fIsXXX ; and for getter function GetXXX() or IsXXX() [where XXX is base name].

As an example of using it let's analyse a few lines which get and set a fIsEditable field in TCanvas:

```
TCanvas     *c  = new TCanvas("c");   // create a canvas
TClass      *cl = c->IsA();            // get its class description object.

TDataMember *dm = cl->GetDataMember("fIsEditable"); //This is our data member

TMethodCall *getter = dm->GetterMethod(); //find a method that gets value!
Long_t l;   // declare a storage for this value;

getter->Execute(c,"",l);  // Get this Value !!!! It will appear in l !!!

TMethodCall *setter = dm->SetterMethod();
setter->Execute(c,"0",);   // Set Value 0 !!!
```

This trick is widely used in ROOT TContextMenu and dialogs for obtaining current values and put them as initial values in dialog fields.

If you don't want to follow the convention of naming used by ROOT you still could benefit from Getter/Setter method support: the solution is to instruct ROOT what the names of this routines are. The way to do it is putting-this information in a comment string to a data field in your class declaration:

```
class MyClass{
    Int_t mydata;  //  *OPTIONS={GetMethod="Get";SetMethod="Set"}
     ...
    Int_t Get(){ return mydata;};
    void  Set(Int_t i){mydata=i;};
    }
```

However, this getting/setting functions are not the only feature of this class. The next point is providing lists of possible settings for concerned data member. The idea is to have a list of possible options for this data member, with strings identifying them. This is used in dialogs with parameters to set - for details see TMethodArg, TMotif-ContextMenu, TContextMenu. This list not only specify allowed value , nut also provide strings naming this options. Options are managed via TList of TOptionListItem objects. This list is also created automatically: if a data type is an enum type, the list will have items describing every enum value, and named according to enum name. If type is Bool_t, two options "ON" and "Off" with values 0 and 1 are created. For other types you need to instruct ROOT about possible options. The way to do it is the same as in case of specifying getter/setter method: a comment string to a data field in Your header file with class definition. The most general format of this string is:

```
*OPTIONS={GetMethod="getter";SetMethod="set-
ter";Items=(it1="title1",it2="title2", ... ) }
```

While parsing this string ROOT firstly looks for command-tokens: GetMethod, SetMethod, Items; They must be preceded by string *OPTIONS= , enclosed by {} and separated by semicolons ";". All command token should have a form TOKEN=VALUE. All tokens are optional. The names of getter and setter method must be enclosed by double-quote marks (") . Specifications of Items is slightly more complicated: you need to put token ITEMS= and then enclose all options in curly brackets "()". You separate options by comas ",". Each option item may have one of the following forms:

```
 IntegerValue  = "Text Label"

 EnumValue     = "Text Label"

"TextValue" = Text Label"
```

One can sepcify values as Integers or Enums - when data field is any Integer,Float or Enum Type; as texts - for Text_t (more precisely: Option_t).

As mentioned above - this information are mainly used by contextmenu, but also in Dump() and Inspect() methods and by the THtml class.

TDataMember(G__DataMemberInfo **info,* TClass **cl*) : **TDictionary**( )

```
Default TDataMember ctor. TDataMembers are constructed in TClass
via a call to TCint::CreateListOfDataMembers(). It parses the comment
```

```
string, initializes optionlist and getter/setter methods.
```

~TDataMember( )

```
TDataMember dtor deletes adopted G__DataMemberInfo object.
```

Int_t **GetArrayDim**( ) const

```
Return number of array dimensions.
```

Int_t **GetMaxIndex**(Int_t *dim*) const

```
Return maximum index for array dimension "dim".
```

const char* **GetTypeName**( ) const

```
Get type of data member, e,g.: "class TDirectory*" -> "TDirectory".
Result needs to be used or copied immediately.
```

const char* **GetFullTypeName**( ) const

```
Get full type description of data member, e,g.: "class TDirectory*".
```

const char* **GetName**( ) const

```
Get data member name.
```

Int_t **GetOffset**( ) const

```
Get offset from "this". This information is only available for
public datamembers.
```

const char* **GetTitle**( ) const

```
Get data member description string (comment).
```

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )

```
Return hash value for TDataMember based on its name.
```

Bool_t **IsBasic**( ) const

```
Return true if data member is a basic type, e.g. char, int, long...
```

Bool_t **IsEnum**( ) const

```
Return true if data member is an enum.
```

Bool_t **IsaPointer**( ) const

```
Return true if data member is a pointer.
```

Long_t **Property**( ) const

```
Get property description word. For meaning of bits see EProperty.
```

TList* **GetOptions**( )

```
Returns list of options - list of TOptionListItems
```

TMethodCall* **GetterMethod**( )

```
Return a TMethodCall method responsible for getting the value of data member
```

TMethodCall* **SetterMethod**( )

```
Return a TMethodCall method responsible for setting the value of data member
```

Author: Fons Rademakers 04/02/95
Last update: 13/11/97 13.17.58 by René Brun

TDataType

class **TDataType** : public TDictionary

private:

void	**SetType**(char* *name*)

public:

class TDataType	**TDataType**((*unknown*) G__TypedefInfo* info = 0)
class TDataType	**TDataType**(char* *typenam*)
class TDataType	**TDataType**(class TDataType&)
void	**~TDataType**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
const char*	**GetFullTypeName**( )
virtual const char*	**GetName**( )
virtual const char*	**GetTitle**( )
Int_t	**GetType**( )
const char*	**GetTypeName**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
const char*	**Print**(void* *buf*)
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
Int_t	**Size**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class G__TypedefInfo*	**fInfo**	*pointer to CINT typedef info*
class TString	**fName**	*name of basic type (in case fInfo==0)*
Int_t	**fSize**	*size of basic type (in case fInfo==0)*
enum EDataType	**fType**	*type id*

Class Description

```
                                                              //
Basic data type descriptor (datatype information is obtained from
CINT). This class describes the attributes of type definitions
(typedef's). The TROOT class contains a list of all currently
defined types (accessible via TROOT::GetListOfTypes()).
```

TDataType(G__TypedefInfo **info*) : **TDictionary**( )

```
Default TDataType ctor. TDataTypes are constructed in TROOT via
a call to TCint::CreateListOfTypes().
```

TDataType(const char **typenam*)

```
Constructor for basic data types, like "char", "unsigned char", etc.
```

~TDataType( )

```
TDataType dtor deletes adopted G__TypedefInfo object.
```

const char* **GetTypeName**( ) const

```
Get basic type of typedef, e,g.: "class TDirectory*" -> "TDirectory".
Result needs to be used or copied immediately.
```

const char* **GetFullTypeName**( ) const

```
Get full type description of typedef, e,g.: "class TDirectory*".
```

const char* **GetName**( ) const

```
Get name of typedef.
```

const char* **GetTitle**( ) const

```
Get typedef description string (comment).
```

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )

```
Return hash value for TDataType based on its name.
```

const char* **Print**(void **buf*) const

```
Return string containing value in buffer formatted according to
the basic data type. The result needs to be used or copied immediately.
```

Long_t **Property**( ) const

```
Get property description word. For meaning of bits see EProperty.
```

void **SetType**(const char **name*)

```
Set type id depending on name.
```

Int_t **Size**( ) const

```
Get size of basic typedef'ed type.
```

Author: René Brun 04/02/95
Last update: 1.03/02 02/09/97 17.44.44 by René Brun

TDatime

class **TDatime**

protected:

UInt_t	**Convert**( )

public:

class TDatime	**TDatime**( )
class TDatime	**TDatime**(class TDatime&)
void	**~TDatime**( )
virtual const char*	**AsString**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TDatime& *datime*)
virtual void	**FillBuffer**(char*& *buffer*)
UInt_t	**Get**( )
virtual Int_t	**GetDate**( )
void	**GetDateTime**(UInt_t *datetime,* Int_t& *date,* Int_t& *time*)
virtual Int_t	**GetTime**( )
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**Set**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

UInt_t	**fDatime**	*Date (relative to 1995*) + time

Class Description

```
TDatime

This class stores the date and time with a precision of one second
in an unsigned 32 bit word. The date is stored with the origin being
the 1st january 1995.
       See also class TStopwatch.
```

TDatime( )

```
Create a TDatime and set it to the current time.
```

const char* **AsString**( ) const

```
Return the date & time as a string (ctime() format).
```

UInt_t **Convert**( ) const

```
Convert fDatime from TDatime format to the standard time_t format.
```

void **Copy**(TDatime &*datime*)

```
Date/Time constructor
```

void **FillBuffer**(char *&*buffer*)

```
Encode Date/Time into output buffer
```

Int_t **GetDate**( )

```
Return Date from packed date/time
```

Int_t **GetTime**( )

```
Return Time from packed date/time
```

void **Print**(Option_t **option*)

```
Print Date/Time in usual format
```

void **ReadBuffer**(char *&*buffer*)

```
Decode Date/Time from output buffer
```

void **Set**( )

```
Get Date/Time from OS
Date and Time are encoded into one single unsigned 32 bit word.
Time has 1 second precision.
Date is stored with the origin being the 1st january 1995.
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **GetDateTime**(UInt_t *datetime,* Int_t &*date,* Int_t &*time*)

```
Return Date and Time unpacked
date is returned in the format 950223  February 23 1995
time is returned in the format 102459 10h 24m 59s
```

Author: René Brun 05/01/95
Last update: 1.00/06 23/03/97 11.01.55 by René Brun

TDialogCanvas

class **TDialogCanvas** : public TCanvas, public TAttText

public:

class TDialogCanvas	**TDialogCanvas**(Text_t* *name,* Text_t* *title,* UInt_t *ww,* UInt_t *wh*)
class TDialogCanvas	**TDialogCanvas**( )
class TDialogCanvas	**TDialogCanvas**(class TDialogCanvas&)
void	**~TDialogCanvas**( )
virtual void	**Apply**(Text_t* *action*)
virtual void	**BuildStandardButtons**( )
class TClass*	**Class**( )
virtual void	**Divide**(Int_t *nx = 1,* Int_t *ny = 1,* Float_t *xmargin* = 0.*01,* Float_t *ymargin* = 0.*01,* Int_t *color = 0*)
class TObject*	**GetRefObject**( )
class TPad*	**GetRefPad**( )
virtual class TClass*	**IsA**( )
virtual void	**Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**SetBorderMode**(Short_t *bordermode*)
virtual void	**SetGrid**(Int_t *valuex = 1,* Int_t *valuey = 1*)
virtual void	**SetLogx**(Int_t *value = 1*)
virtual void	**SetLogy**(Int_t *value = 1*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetObject**( )
virtual void	**SetRefObject**(class TObject* *obj*)
virtual void	**SetRefPad**(class TPad* *pad*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**x3d**(Option_t* *option*)

Data Members

protected:

class TObject*	**fRefObject**	*Pointer to object to set attributes*
class TPad*	**fRefPad**	*Pad containing object*

See Also: TAttFillCanvas, TAttLineCanvas, TAttMarkerCanvas, TAttTextCanvas, TDrawPanelHist, TFitPanel

Class Description

```
A DialogCanvas is a canvas specialized to set attributes.
It contains, in general, TGroupButton objects.
When the APPLY button is executed, the actions corresponding
to the active buttons are executed via the Interpreter.

See examples in TAttLineCanvas, TAttFillCanvas, TAttTextCanvas, TAttMarkerCan-
vas
```

TDialogCanvas( ) : **TCanvas**( )

```
DialogCanvas default constructor
```

TDialogCanvas(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TCanvas**(*name, title, -ww, wh*)

```
DialogCanvas constructor
```

~TDialogCanvas( )

```
DialogCanvas default destructor
```

void **Apply**(Text_t **action*)

`Called when the APPLY button is executed`

void **BuildStandardButtons**( )
`Create APPLY, gStyle and CLOSE buttons`

void **Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
`Set world coordinate system for the pad`

void **RecursiveRemove**(TObject **obj*)
`Recursively remove object from a pad and its subpads`

void **SetObject**( )
`Copy referenced object/pad in gROOT to this dialogcanvas`

Author: René Brun 03/07/96
Last update: 1.03/03 04/09/97 12.43.44 by René Brun

TDialogItem

class **TDialogItem** : public TComponent, public TObject

public:

void	**~TDialogItem**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual enum EDialogItemType	**DialogItemType**( )
virtual class TComponent*	**GetDialog**( )
virtual const Text_t*	**GetStorageType**( )
virtual Widget	**Instantiate**(Widget *parent,* Arg* *args = 0,* Int_t *numargs = 0*)
virtual class TClass*	**IsA**( )
virtual void	**SetStorageType**(Text_t* *storage*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

const Text_t*	**fStorageType**	*Type of stored value; i.e. "Long_t", "Option_t*", etc;default="Text_t".*
class TComponent*	**fDialog**	

See Also: TMotifButton, TMotifOptionMenu, TMotifTextField

Class Description

```
TDialogItem

This class defines the dialog items.
```

~TDialogItem( )

```
empty
```

const char* **ClassName**( )

Author: Piotr Golonka 19/08/97
Last update: 1.03/03 03/09/97 16.48.29 by Piotr Golonka

TDialogManager

class **TDialogManager** : public TComponent

private:

void	**AddDialog**(Widget *parent,* Widget *dialog*)
int	**CachedDialogFor**(Widget *parent*)
void	**DialogBoxAPPLY**(*Widget, XtPointer, XtPointer*)
void	**DialogBoxCANCEL**(*Widget, XtPointer, XtPointer*)
void	**DialogBoxOK**(*Widget, XtPointer, XtPointer*)
Widget	**GetDialog**(Widget *parent*)
void	**HandleWm**(*Widget, XtPointer, XtPointer*)
void	**SetupWindowManagerProperties**(Widget *dialog*)

protected:

virtual void	**Apply**(*Widget, XtPointer*)
virtual void	**Cancel**(*Widget, XtPointer*)
virtual Widget	**CreateDialog**(Widget *parent*)
Widget	**GetChildButton**(Widget *w,* unsigned int *child*)
virtual void	**Ok**(*Widget, XtPointer*)
Widget	**Prepost**(char* *msg,* char* *helpString,* class TSimpleWindow* *parent*)
virtual Widget	**Prepost**(char* *msg,* XtCallbackProc *okCB = NULL,* XtCallbackProc *cancelCB = NULL,* XtCallbackProc *applyCB = NULL,* XtPointer *clientData = NULL,* char* *helpString = NULL,* class TSimpleWindow* *parent = NULL*)

public:

void	**~TDialogManager**( )
void	**CenterOnScreen**(Boolean *flag*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual class TClass*	**IsA**( )
Widget	**LastPosted**( )
Widget	**Post**(char* *msg,* Widget *parent*)
Widget	**Post**(char* *msg,* char* *helpString,* Widget *parent*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* Widget *parent*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* char* *helpString,* Widget *parent*)
virtual Widget	**Post**(char* *msg = NULL,* XtCallbackProc *okCB = NULL,* XtCallbackProc *cancelCB = NULL,* XtCallbackProc *applyCB = NULL,* XtPointer *clientData = NULL,* char* *helpString = NULL,* class TSimpleWindow* *parent = NULL,* Widget* *parentWidget = NULL*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* Widget *parent*)
Widget	**Post**(char* *msg,* class TSimpleWindow* *parent*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* char* *helpString,* Widget *parent*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtCallbackProc *applyCB,* XtPointer *clientData,* Widget *parent*)
Widget	**Post**(char* *msg,* char* *helpString,* class TSimpleWindow* *parent*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* class TSimpleWindow* *parent = NULL*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget	**Post**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtCallbackProc *applyCB,* XtPointer *clientData,* class TSimpleWindow* *parent*)

TDialogManager::EDialogReason**PostAndWait**(char* *msg,* class TSimpleWindow* *parent*)
TDialogManager::EDialogReason**PostAndWait**(char* *msg,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
virtual TDialogManager::EDialogReason**PostAndWait**(char* *msg = NULL,* Boolean *ok = TRUE,* Boolean *cancel = TRUE,* Boolean *apply = FALSE,* char* *helpString = NULL,* class TSimpleWindow* *parent = NULL,* Widget* *parentWidget = NULL*)
Widget **PostBlocked**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtCallbackProc *applyCB,* XtPointer *clientData,* class TSimpleWindow* *parent*)
virtual Widget **PostBlocked**(char* *msg = NULL,* XtCallbackProc *okCB = NULL,* XtCallbackProc *cancelCB = NULL,* XtCallbackProc *applyCB = NULL,* XtPointer *clientData = NULL,* char* *helpString = NULL,* class TSimpleWindow* *parent = NULL,* Widget* *parentWidget = NULL*)
virtual Widget **PostBlocked**(char* *msg,* class TSimpleWindow* *parent*)
Widget **PostBlocked**(char* *msg,* char* *helpString,* class TSimpleWindow* *parent*)
Widget **PostBlocked**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* class TSimpleWindow* *parent = NULL*)
Widget **PostBlocked**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget **PostBlocked**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* class TSimpleWindow* *parent = NULL*)
Widget **PostBlocked**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* class TSimpleWindow* *parent = NULL*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* char* *helpString,* class TSimpleWindow* *parent = NULL*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtCallbackProc *applyCB,* XtPointer *clientData,* class TSimpleWindow* *parent*)
Widget **PostModal**(char* *msg,* Widget *parent*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* Widget *parent*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* char* *helpString,* Widget *parent*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* Widget *parent*)
virtual Widget **PostModal**(char* *msg = NULL,* XtCallbackProc *okCB = NULL,* XtCallbackProc *cancelCB = NULL,* XtCallbackProc *applyCB = NULL,* XtPointer *clientData = NULL,* char* *helpString = NULL,* class TSimpleWindow* *parent = NULL,* Widget* *parentWidget = NULL*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtPointer *clientData,* char* *helpString,* Widget *parent*)
Widget **PostModal**(char* *msg,* class TSimpleWindow* *parent*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtCallbackProc *cancelCB,* XtCallbackProc *applyCB,* XtPointer *clientData,* Widget *parent*)
Widget **PostModal**(char* *msg,* char* *helpString,* class TSimpleWindow* *parent*)
Widget **PostModal**(char* *msg,* XtCallbackProc *okCB,* XtPointer *clientData,* class TSimpleWindow* *parent = NULL*)
void **SetButtonLabels**(char* *okLabel = NULL,* char* *cancelLabel = NULL,* char* *applyLabel = NULL*)
void **SetTitle**(char* *nextTitle = NULL*)
virtual void **ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void **Streamer**(class TBuffer& *b*)

void **Unpost**( )
void **Unpost**(Widget *w*)
void **UnpostAll**( )

Data Members

private:

Widget* **fParents**
Widget* **fDialogs**
int **fNumber_of_dialogs_cached**
int **fMax_number_of_dialogs_cached**
Boolean **fCentered**
char* **fOkLabel**
char* **fCancelLabel**
char* **fApplyLabel**
Boolean **fDone_with_event_loop**

protected:

String **fDialogShellTitle**
Boolean **fOkButton**
Boolean **fCancelButton**
Boolean **fApplyButton**
Boolean **fHelpButton**
TDialogManager::EDialogReason**fAction**
Boolean **fAllowMultipleDialogs**

public:

static const TDialogManager::EDialogReason**kOk**
static const TDialogManager::EDialogReason**kApply**
static const TDialogManager::EDialogReason**kCancel**

See Also: TBusyDialog, TErrorDialog, TFileSelectionDialog, TInfoDialog, TPromptDialog, TQuestionDialog, TSelectionDialog, TWarningDialog

Class Description

```
TDialogManager

This is an abstract class for dialog managers, from which all other
dialog managers are derived.
```

void **HandleWm**(Widget *w,* XtPointer *clientData, XtPointer*)

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TDiamond

class **TDiamond** : public TPaveText

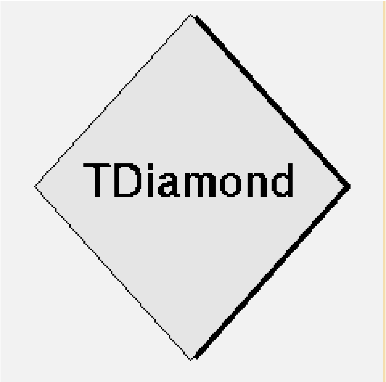
public:

class TDiamond	**TDiamond**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TDiamond	**TDiamond**( )
class TDiamond	**TDiamond**(class TDiamond& *diamond*)
void	**~TDiamond**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *diamond*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
A diamond is defined by :
- Its central left coordinates x1,y1
- Its top central coordinates x2,y2

A diamond has line attributes (see TAttLine)
and fill area attributes (see TAttFill).

Like for the class TPaveText, a TDiamond may have one or more line(s)
of text inside.
```



TDiamond( ): **TPaveText**( )

```
Diamond default constructor
```

TDiamond(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TPaveText**(*x1, y1, x2, y2*)

```
Diamond standard constructor
```

~TDiamond( )

```
Diamond destructor
```

void **Copy**(TObject &*diamond*)

```
Copy a Diamond
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a diamond
Compute the closest distance of approach from point px,py to the
edges of this diamond.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw this diamond with its current attributes
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a Diamond object is clicked.

If the mouse is clicked inside the diamond, the diamond is moved.

If the mouse is clicked on the 4 tops (L,R,T,B), the diamond is rscaled.

                            T
                       +---------+
                       |   / \   |
                       |  /   \  |
                       | /     \ |
                      L|/ inside\|R
                       |\       /|
                       | \     / |
                       |  \   /  |
                       |   \ /   |
                       +---------+
                            B
```

void **Paint**(Option_t **option*)

```
Paint this diamond with its current attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 22/06/96
Last update: 1.02/00 09/07/97 22.19.14 by René Brun

TDictionary

class **TDictionary** : public TObject

public:

void	**~TDictionary**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
virtual const char*	**GetName**( )
virtual const char*	**GetTitle**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsSortable**( )
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TBaseClass, TClass, TDataMember, TDataType, TFunction, TGlobal, TMethodArg

Class Description

```
TDictionary

This class defines an abstract interface that must be implemented
by all classes that contain dictionary information.

The dictionary is defined by the followling classes:
TDataType                             (typedef definitions)
TGlobal                               (global variables)
TGlobalFunc                           (global functions)
TClass                                (classes)
 TBaseClass                         (base classes)
 TDataMember                        (class datamembers)
 TMethod                            (class methods)
    TMethodArg                      (method arguments)

All the above classes implement the TDictionary abstract interface.
Note: the indentation shows aggregation not inheritance.

TMethodCall                           (method call environment)
```

Author: Fons Rademakers 20/06/96
Last update: 1.00/06 23/03/97 11.32.06 by René Brun

TDirectory

class **TDirectory** : public TNamed

protected:	
Bool_t	**cd1**(Text_t* *path*)
Bool_t	**Cd1**(Text_t* *path*)
public:	
class TDirectory	**TDirectory**(Text_t* *name,* Text_t* *title,* Option_t* *option*)
class TDirectory	**TDirectory**( )
class TDirectory	**TDirectory**(class TDirectory& *directory*)
void	**~TDirectory**( )
Int_t	**AppendKey**(class TKey* *key*)
virtual void	**Browse**(class TBrowser* *b*)
void	**Build**( )
virtual void	**cd**(Text_t* *path = 0*)
void	**Cd**(Text_t* *path*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Close**(Option_t* *option*)
virtual void	**Copy**(class TObject& *directory*)
void	**DecodeNameCycle**(Text_t* *namecycle,* Text_t* *name,* Short_t& *cycle*)
virtual void	**Delete**(Text_t* *namecycle*)
virtual void	**DeleteAll**(Option_t* *option*)
virtual void	**Draw**(Option_t* *option*)
void	**EncodeNameCycle**(Text_t* *buffer,* Text_t* *name,* Short_t *cycle*)
virtual void	**FillBuffer**(char*& *buffer*)
virtual class TObject*	**Get**(Text_t* *namecycle*)
virtual class TFile*	**GetFile**( )
virtual class TKey*	**GetKey**(Text_t* *name,* Short_t *cycle = 9999*)
class TList*	**GetList**( )
class TList*	**GetListOfKeys**( )
class TObject*	**GetMother**( )
virtual Int_t	**GetNkeys**( )
virtual const char*	**GetPath**( )
virtual Seek_t	**GetSeekDir**( )
virtual Seek_t	**GetSeekKeys**( )
virtual Seek_t	**GetSeekParent**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
Bool_t	**IsModified**( )
Bool_t	**IsWritable**( )
virtual void	**ls**(Option_t* *option*)
virtual class TDirectory*	**mkdir**(Text_t* *name,* Text_t* *title*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**Purge**(Short_t *nkeep = 1*)
virtual void	**pwd**( )
virtual void*	**Read**(Option_t* *option*)
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**ReadKeys**( )
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**Save**( )
virtual void	**SaveSelf**(Bool_t *force = kFALSE*)
void	**SetModified**( )
void	**SetMother**(class TObject* *mother*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)

virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Write**(Option_t* *option*)
virtual void	**WriteDirHeader**( )
virtual void	**WriteKeys**( )

Data Members

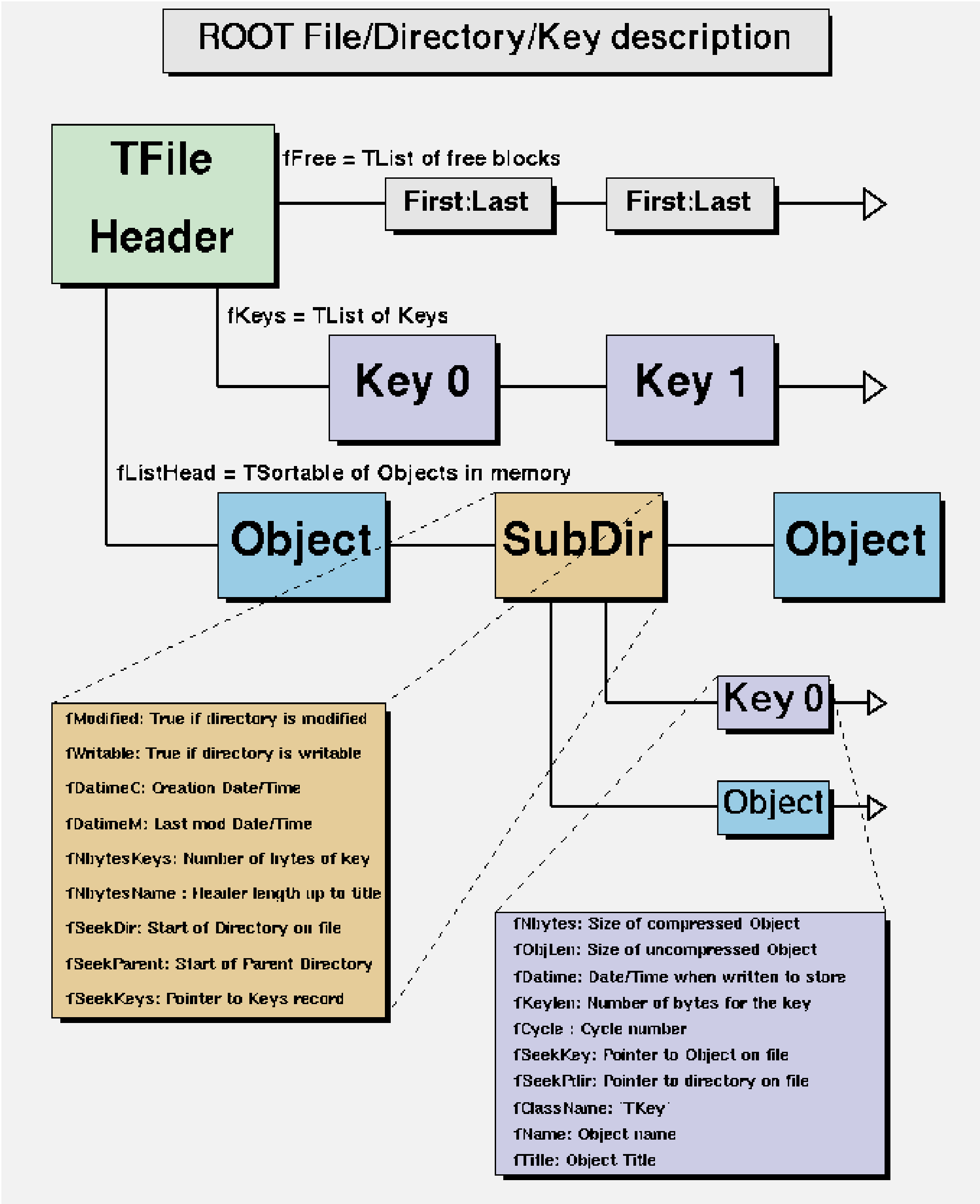
protected:

Bool_t	**fModified**	*true if directory has been modified*
Bool_t	**fWritable**	*true if directory is writable*
class TDatime	**fDatimeC**	*Date and time when directory is created*
class TDatime	**fDatimeM**	*Date and time of last modification*
Int_t	**fNbytesKeys**	*Number of bytes for the keys*
Int_t	**fNbytesName**	*Number of bytes in TNamed at creation time*
Seek_t	**fSeekDir**	*Location of directory on file*
Seek_t	**fSeekParent**	*Location of parent directory on file*
Seek_t	**fSeekKeys**	*Location of Keys record on file*
class TFile*	**fFile**	*pointer to current file in memory*
class TObject*	**fMother**	*pointer to mother of the directory*
class TList*	**fList**	*Pointer to objects list in memory*
class TList*	**fKeys**	*Pointer to keys list in memory*

See Also: TFile, TROOT

Class Description

```
A ROOT file is structured in Directories (like a file system).
Each Directory has a list of Keys (see TKeys) and a list of objects
in memory. A Key is a small object that describes the type and location
of a persistent object in a file. The persistent object may be a directory.
```




```
The structure of a file is shown in TFile::TFile
```

TDirectory( ) : **TNamed**( )

```
Directory default constructor
```

TDirectory(const Text_t **name*, const Text_t **title*, Option_t **option*): **TNamed**(*name, title*)

```
Create a new Directory
A new directory with name,title is created in the current directory
The directory header information is immediatly saved on the file
A new key is added in the parent directory
```

~**TDirectory**( )

```
Directory destructor
```

Int_t **AppendKey**(TKey **key*)

```
Insert key in the linked list of keys of this directory
```

void **Build**( )

```
Initialise directory to defaults
```

void **cd**(const Text_t **path*)

```
Change current directory to "this" directory . Using path one can
change the current directory to "path". The absolute path syntax is:
file.root:/dir1/dir2
where file.root is the file and /dir1/dir2 the desired subdirectory
in the file. Relative syntax is relative to "this" directory. E.g:
../aa.
```

Bool_t **cd1**(const Text_t **path*)

```
Change current directory to "this" directory . Using path one can
change the current directory to "path". The absolute path syntax is:
file.root:/dir1/dir2
where file.root is the file and /dir1/dir2 the desired subdirectory
in the file. Relative syntax is relative to "this" directory. E.g:
../aa. Returns kFALSE in case path does not exist.
```

void **Cd**(const Text_t **path*)

```
Change current directory to "path". The absolute path syntax is:
file.root:/dir1/dir2
where file.root is the file and /dir1/dir2 the desired subdirectory
in the file. Relative syntax is relative to the current directory
gDirectory, e.g.: ../aa.
```

Bool_t **Cd1**(const Text_t **path*)

```
Change current directory to "path". The path syntax is:
file.root:/dir1/dir2
where file.root is the file and /dir1/dir2 the desired subdirectory
in the file. Returns kFALSE in case path does not exist.
```

void **Clear**(Option_t **option*)

```
Delete all objects from a Directory list
```

void **Close**(Option_t **option*)

```
Delete all objects from memory and directory structure itself
```

void **Copy**(TObject &*directory*)

```
Copy "this" directory into directory
The target directory must have been created first
```

void **DeleteAll**(Option_t **option*)

```
Delete all objects from memory
```

void **Delete**(Text_t **namecycle*)

```
Delete Objects or/and keys in a directory
namecycle has the format name;cycle
namecycle = "" same as namecycle ="T*"
name  = * means all
cycle = * means all cycles (memory and keys)
cycle = "" or cycle = 9999 ==> apply to a memory object
When name=* use T* to delete subdirectories also

examples:
  foo   : delete object named foo in memory
  foo;1 : delete cycle 1 of foo on file
  foo;* : delete all cycles of foo on file and also from memory
  *;2   : delete all objects on file having the cycle 2
  *;*   : delete all objects from memory and file
 T*;*   : delete all objects from memory and file and all subdirectories
```

void **Draw**(Option_t **option*)

```
Fill Graphics Structure and Paint
Loop on all objects (memory or file) and all subdirectories
```

void **FillBuffer**(char *&*buffer*)

```
Encode directory header into output buffer
```

TObject* **Get**(Text_t **namecycle*)

```
return pointer to object identified by namecycle
namecycle has the format name;cycle
name  = * is illegal, cycle = * is illegal
cycle = "" or cycle = 9999 ==> apply to a memory object

examples:
  foo   : get object named foo in memory
          if object is not in memory, try with highest cycle from file
  foo;1 : get cycle 1 of foo on file
```

TKey* **GetKey**(const Text_t **name,* const Short_t *cycle*)

```
Return pointer to key with name,cycle
if cycle = 9999 returns highest cycle
```

const char* **GetPath**( ) const

```
Returns the full path of the directory. E.g. file:/dir1/dir2.
The returned path will be re-used by the next call to GetPath().
```

TDirectory* **mkdir**(const Text_t **name,* const Text_t **title*)

```
Create a sub-directory and return a pointer to the created directory.
Returns 0 in case of error.
```

void **ls**(Option_t **option*)

```
List Directory contents
Indentation is used to identify the directory tree
Subdirectories are listed first, then objects in memory, then objects on the
file

If option contains "m" only memory objects are listed.
If option contains "d" only disk objects are listed.
By default memory and disk objects are listed.
```

void **Paint**(Option_t **option*)

```
Paint all objects in the directory
```

void **Print**(Option_t **option*)

```
Print all objects in the directory
```

void **Purge**(Short_t *nkeep*)

```
Purge lowest cycles in a directory
By default, only the highest cycle of a key is kept. Keys for which
the "KEEP" flag has been set are not removed. See TKey::Keep()
```

void **pwd**( ) const

```
Print the path of the directory.
```

void* **Read**(Option_t **option*)

```
Read objects from file into memory
If an object is already in memory, the memory copy is deleted
and the object is again read from the file. The read callback routine
for each object is called by TKey::Read(). The address of the callback
routine is set in the class Dictionary() function.
```

void **ReadBuffer**(char *&*buffer*)

```
Decode input buffer
```

void **ReadKeys**( )

```
Read the KEYS linked list
Every directory has a linked list (fKeys). This linked list has been
written on the file via WriteKeys as a single data record.
```

void **RecursiveRemove**(TObject **obj*)

```
Recursively remove object from a Directory
```

void **Save**( )

```
Save recursively all directory keys and headers
```

void **SaveSelf**(Bool_t *force*)

```
Save Directory keys and header
If the directory has been modified (fModified set), write the keys
and the directory header. This function assumes the cd is correctly set.
```

Int_t **Sizeof**( ) const

```
Return the size in bytes of the directory header
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **Write**(Option_t **option*)

```
Objects in memory become persistent
Loop on all objects in memory (including subdirectories)
A new key is created in the KEYS linked list for each object
The list of keys is then saved on the file (via WriteKeys)
as a single data record
The directory header info is rewritten on the directory header record
```

void **WriteDirHeader**( )

```
Overwrite the Directory header record
```

void **WriteKeys**( )

```
Write KEYS linked list on the file
The linked list of keys (fKeys) is written as a single data record
```

void **EncodeNameCycle**(Text_t **buffer,* const Text_t **name,* Short_t *cycle*)

```
Encode the name and cycle into buffer like: "aap;2".
```

void **DecodeNameCycle**(const Text_t **buffer,* Text_t **name,* Short_t &*cycle*)

```
Decode a namecycle "aap;2" into name "aap" and cycle "2".
```

Author: René Brun 28/11/94
Last update: 1.02/00 15/07/97 12.43.03 by René Brun

TDrawPanelHist

class **TDrawPanelHist** : public TDialogCanvas

public:

class TDrawPanelHist	**TDrawPanelHist**( )
class TDrawPanelHist	**TDrawPanelHist**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 500,* UInt_t *wh = 600*)
class TDrawPanelHist	**TDrawPanelHist**(class TDrawPanelHist&)
void	**~TDrawPanelHist**( )
virtual void	**AddOption**(Option_t* *option*)
virtual void	**Apply**(Text_t* *action*)
virtual void	**BuildStandardButtons**( )
class TClass*	**Class**( )
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
class TObject*	**GetHistogram**( )
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetDefaults**( )
virtual void	**SetSame**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

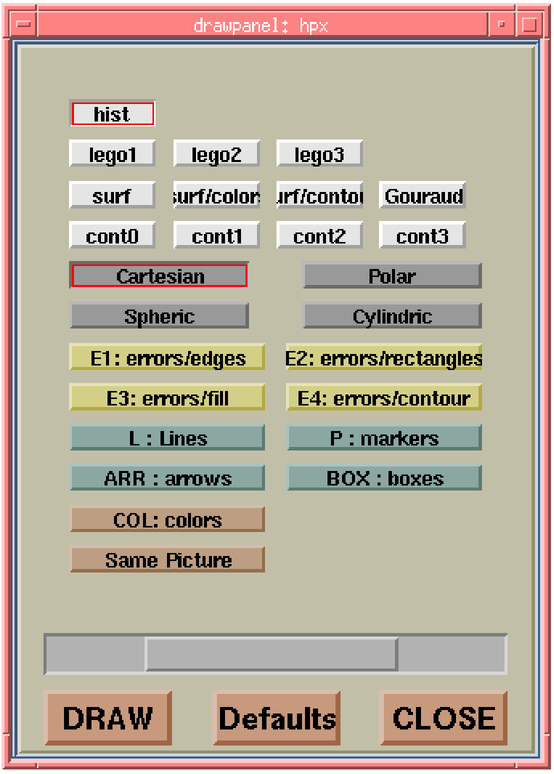
Data Members

protected:

class TString	**fOption**	*Fitting options*
class TSlider*	**fSlider**	*Pointer to fitpanel slider*
class TObject*	**fHistogram**	*Pointer to histogram*

Class Description

```
A TDrawPanelHist is a TDialogCanvas specialized to control
histogram drawing options.
With the mouse, the user can control:
  - the drawing range in X and Y
  - the drawing options
When the DRAW button is executed, the selected histogram is drawn
with the current parameters.

Use the slider to control the range of the histogram to be drawn.

The options are documented in TH1::Draw.
```



TDrawPanelHist( ) : **TDialogCanvas**( )

```
DrawPanelHist default constructor
```

TDrawPanelHist(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)

```
DrawPanelHist constructor
```

~**TDrawPanelHist**( )

```
DrawPanelHist default destructor
```

void **AddOption**(Option_t **option*)

```
Add option to the current list of options
```

void **Apply**(Text_t **action*)

```
Collect all options and fit histogram
```

void **BuildStandardButtons**( )

```
Create FIT, Defaults and CLOSE buttons
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Control mousse events when slider is used in a fitpanel
 This function is called by TPad::ExecuteEvent or TSliderBox::ExecuteEvent
 We return in the first case.
 When called by the slider,  px = 0 and py = 0
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save this fitpanel in a macro
```

void **SetDefaults**( )

```
Set default fit panel options
```

void **SetSame**( )

```
Set graphics option "same"
```

Author: René Brun 26/11/96
Last update: 1.03/01 22/08/97 09.10.31 by René Brun

TEllipse

class **TEllipse** : public TObject, public TAttLine, public TAttFill

public:

class TEllipse	**TEllipse**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2 = 0,* Float_t *phimin = 0,* Float_t *phimax = 360,* Float_t *theta = 0*)
class TEllipse	**TEllipse**( )
class TEllipse	**TEllipse**(class TEllipse& *ellipse*)
void	**~TEllipse**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *ellipse*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawEllipse**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2,* Float_t *phimin,* Float_t *phimax,* Float_t *theta*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Float_t	**GetPhimax**( )
Float_t	**GetPhimin**( )
Float_t	**GetR1**( )
Float_t	**GetR2**( )
Float_t	**GetTheta**( )
Float_t	**GetX1**( )
Float_t	**GetY1**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintEllipse**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2,* Float_t *phimin,* Float_t *phimax,* Float_t *theta*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**((*unknown*) ofstream& out, Option_t* option)
virtual void	**SetPhimax**(Float_t *phi = 360*)
virtual void	**SetPhimin**(Float_t *phi = 0*)
virtual void	**SetR1**(Float_t *r1*)
virtual void	**SetR2**(Float_t *r2*)
virtual void	**SetTheta**(Float_t *theta = 0*)
virtual void	**SetX1**(Float_t *x1*)
virtual void	**SetY1**(Float_t *y1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

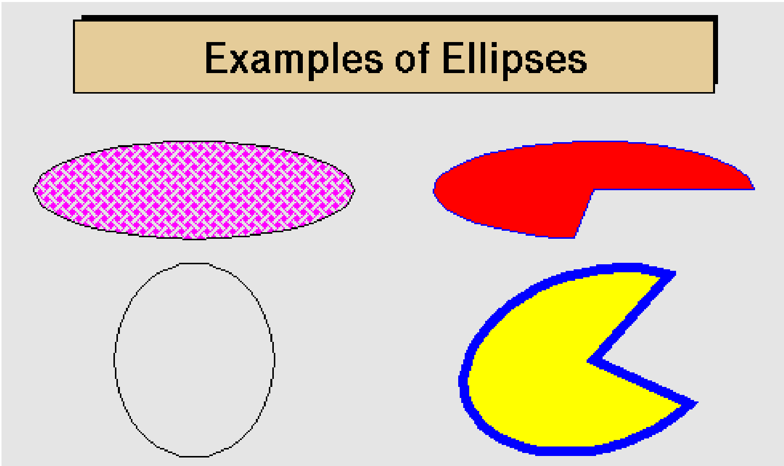
Float_t	**fX1**	*X coordinate of centre*
Float_t	**fY1**	*Y coordinate of centre*
Float_t	**fR1**	*first radius*
Float_t	**fR2**	*second radius*
Float_t	**fPhimin**	*Minimum angle (degrees*)
Float_t	**fPhimax**	*Maximum angle (degrees*)
Float_t	**fTheta**	*Rotation angle (degrees*)

See Also: TArc

Class Description

```
Ellipse class
Ellipse is a general ellipse that can be truncated and rotated.
```

```
An Ellipse is defined by its center (x1,y1) and two radius r1 and r2.
A minimum and maximum angle may be specified (phimin, phimax).
The Ellipse may be rotated with an angle theta.
The attributes of the outline line are given via TAttLine.
The attributes of the fill area are given via TAttFill.
The picture below illustrates different types of ellipses.
```



TEllipse( ): **TObject**( ), **TAttLine**( ), **TAttFill**( )

```
Ellipse default constructor
```

TEllipse(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2,* Float_t *phimin,* Float_t *phimax,* Float_t *theta*):**TObject**( ), **TAttLine**( ), **TAttFill**(*0, 1001*)

```
Ellipse normal constructor
```

~TEllipse( )

```
Ellipse default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this ellipse to ellipse
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to an ellipse
Compute the closest distance of approach from point px,py to this ellipse.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw this ellipse with its current attributes
```

void **DrawEllipse**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2,* Float_t *phimin,* Float_t *phimax,* Float_t *theta*)

```
Draw this ellipse with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a line is clicked with the locator

If Left button clicked on one of the line end points, this point
  follows the cursor until button is released.

if Middle button clicked, the line is moved parallel to itself
  until the button is released.
```

void **ls**(Option_t **option*)

```
List this ellipse with its attributes
```

void **Paint**(Option_t **option*)

```
Paint this ellipse with its current attributes
```

void **PaintEllipse**(Float_t *x1,* Float_t *y1,* Float_t *r1,* Float_t *r2,* Float_t *phimin,* Float_t *phimax,* Float_t *theta*)

```
Draw this ellipse with new coordinates
```

void **Print**(Option_t **option*)

```
Dump this ellipse with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 16/10/95
Last update: 1.02/00 09/07/97 22.20.46 by René Brun

TELTU

class **TELTU** : public TTUBE

public:

class TELTU	**TELTU**( )
class TELTU	**TELTU**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *rx,* Float_t *ry,* Float_t *dz*)
class TELTU	**TELTU**(class TELTU&)
void	**~TELTU**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fRx**	*inside radius at the high z limit*
Float_t	**fRy**	*outside radius at the high z limit*

Class Description

```
'ELTU' is a  cylinder with  an elliptical  section.  It  has three
     parameters:  the  ellipse  semi-axis   in  X,  the  ellipse
     semi-axis in Y  and the half length in Z.   The equation of
     the conical curve is:
          X**2/fRx**2  +  Y**2/fRy**2  =  1
     ELTU is not divisible.

   - name       name of the shape
   - title      shape's title
   - material  (see TMaterial)
   - rx         the  ellipse  semi-axis   in  X
   - ry         the  ellipse  semi-axis   in  Y
   - dz         half-length in z
```

TELTU( )

```
ELTU shape default constructor
```

TELTU(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rx,* Float_t *ry,* Float_t *dz*): **TTUBE**(*name, title, material, 0, rx, dz*)

```
ELTU shape normal constructor
```

~TELTU( )

```
ELTU shape default destructor
```

Author: René Brun 26/06/97
Last update: 1.01/07 26/06/97 08.49.55 by René Brun

TEnv

class **TEnv** : public TObject

private:

const char*	**Getvalue**(char* *name*)

public:

class TEnv	**TEnv**(char* *name*)
class TEnv	**TEnv**(class TEnv&)
void	**~TEnv**( )
class TClass*	**Class**( )
Bool_t	**Defined**(char* *name*)
class TOrdCollection*	**GetTable**( )
const char*	**GetValue**(char* *name,* char* *dflt*)
int	**GetValue**(char* *name,* int *dflt*)
class TObject*	**GetValue**(char* *name,* class TObject* *dflt*)
virtual class TClass*	**IsA**( )
class TEnvRec*	**Lookup**(char* *n*)
void	**Print**(Option_t* *option*)
void	**PrintEnv**((*unknown*) EEnvLevel level = kEnvAll)
void	**ReadFile**(char* fname, (*unknown*) EEnvLevel level)
void	**Save**( )
void	**SaveLevel**((*unknown*) EEnvLevel level)
void	**SetValue**(char* *name,* double *value*)
void	**SetValue**(char* *name,* int *value*)
void	**SetValue**(char* name, char* value, (*unknown*) EEnvLevel level = kEnvChange, char* type = 0)
void	**SetValue**(char* name, (*unknown*) EEnvLevel level = kEnvChange)
void	**SetValue**(char* name, class TString& value, (*unknown*) EEnvLevel level, char* type)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TOrdCollection*	**fTable**
class TString	**fRcName**

Class Description

```
TEnv

The TEnv class reads a config file, by default .rootrc. Three types
of .rootrc files are read: global, user and local files. The global
file resides in $ROOTSYS, the user file in ~/ and the local file in
the current working directory.
The format of the .rootrc file is similar to the .Xdefaults format:

<SystemName>.<RootName|ProgName>.<name>[(type)]:  <value>

Where <SystemName> is either Unix, WinNT or MacOS,
<RootName> the root name as given in the TROOT ctor,
<ProgName> the current program name and
<name> is the resource name, with optionally a type specification.

E.g.:

Unix.rint.Root.DynamicPath: .:$ROOTSYS/lib:~/lib
Rint.Root.Debug:  FALSE
```

```
TH.Root.Debug: YES
*.Root.MemStat: 1

<SystemName> and <ProgName> or <RootName> may be the wildcard "*".
A # in the first column starts comment line.

Currently the following resources are defined:
 Root.Debug                (bool)         (*)
 Root.MemStat              (bool)         (*)
 Root.MemStat.size         (int)          (*)
 Root.MemStat.cnt          (int)          (*)
 Root.ObjectStat           (bool)         (*)
 Root.MacroPath            (string)
 Root.DynamicPath          (string)
 Rint.Logon                (string)
 Rint.Logoff               (string)

 (*) work only with the <RootName> since no <ProgName> is available
   at time of initialization.

Note that the .rootrc config files contain the config for all ROOT
based applications.
```

TEnv(const char **name*)

    Create a resource table and read the (possibly) three resource files.

~**TEnv**( )

    Delete the resource table.

const char* **Getvalue**(const char **name*)

    Returns the character value for a named resouce.

int **GetValue**(const char **name,* int *dflt*)

    Returns the integer value for a resource. If the resource is not found
    return the dflt value.

const char* **GetValue**(const char **name,* const char **dflt*)

    Returns the character value for a named resouce. If the resource is
    not found the dflt value is returned.

TObject* **GetValue**(const char **name,* TObject **op*)

    Read in object op the values of the resource and return the object.

TEnvRec* **Lookup**(const char **name*)

    Loop over all resource records and return the one with name.
    Return 0 in case name is not in the resoucre table.

void **Print**(Option_t **opt*)

    Print all resources or the global, user or local resources separately.

void **PrintEnv**(EEnvLevel *level*)

    Print all resources for a certain level (global, user, local).

void **ReadFile**(const char **fname,* EEnvLevel *level*)

    Read and parse the resource file for a certain level.

void **Save**( )

    Write the resource files for each level. The new files have the same
    name as the original files. The old files are renamed to *.bak.

void **SaveLevel**(EEnvLevel *level*)

    Write the resource file for a certain level.

void **SetValue**(const char **name,* const TString &*value,* EEnvLevel *level,* const char **type*)
```
Set the value of a resource or create a new resource.
```

void **SetValue**(const char **name,* const char **value,* EEnvLevel *level,* const char **type*)
```
Set the value of a resource or create a new resource.
```

void **SetValue**(const char **name,* EEnvLevel *level*)
```
Set the value of a resource or create a new resource.
```

void **SetValue**(const char **name,* int *value*)
```
Set or create an integer resource value.
```

void **SetValue**(const char **name,* double *value*)
```
Set or create a double resource value.
```

Author: Fons Rademakers 22/09/95
Last update: 1.03/04 30/09/97 15.35.01 by Fons Rademakers

TErrorDialog

class **TErrorDialog** : public TDialogManager

protected:

class TErrorDialog	**TErrorDialog**( )

public:

class TErrorDialog	**TErrorDialog**(char* *name*)
class TErrorDialog	**TErrorDialog**(class TErrorDialog&)
void	**~TErrorDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TFatalErrorDialog

Class Description

```
TErrorDialog

This class handles error dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TEventList

class **TEventList** : public TNamed

public:

class TEventList	**TEventList**(Text_t* *name,* Text_t* *title,* Int_t *initsize = 0,* Int_t *delta = 0*)
class TEventList	**TEventList**( )
class TEventList	**TEventList**(class TEventList& *list*)
void	**~TEventList**( )
virtual void	**Add**(class TEventList* *list*)
class TClass*	**Class**( )
virtual Bool_t	**Contains**(Int_t *entry*)
virtual void	**Enter**(Int_t *entry*)
virtual Int_t	**GetEntry**(Int_t *index*)
virtual Int_t	**GetIndex**(Int_t *entry*)
virtual Int_t*	**GetList**( )
virtual Int_t	**GetN**( )
virtual Int_t	**GetSize**( )
virtual class TClass*	**IsA**( )
class TEventList&	***operator*=**(class TEventList& *list*)
virtual void	**Print**(Option_t* *option*)
virtual void	**Resize**(Int_t *delta = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sort**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Subtract**(class TEventList* *list*)

Data Members

protected:

Int_t	**fN**	*Number of elements in the list*
Int_t	**fSize**	*Size of array*
Int_t	**fDelta**	*Increment size*
Int_t*	**fList**	*Array of elements*

Class Description

```
A TEventList object is a list of selected events (entries) in a TTree

Use function Enter to enter an element in the list
The function Add may be used to merge two lists.
The function Subtract may be used to subtract two lists.
Operators + and - correspond to functions Add and Subtract.
A TEventList object can be saved on a file via the Write function.
```

TEventList( ): **TNamed**( )

```
Default constructor for a EventList
```

TEventList(const Text_t **name,* const Text_t **title,* Int_t *initsize,* Int_t *delta*):**TNamed**(*name, title*)

```
Create a EventList
```

TEventList(const TEventList &*list*)

```
Copy constructor
```

~TEventList( )

```
Default destructor for a EventList
```

void **Add**(const TEventList **alist*)

```
        Merge contents of alist with this list.
Both alist and this list are assumed to be sorted prior to this call
```

Bool_t **Contains**(Int_t *entry*)

```
Return TRUE if list contains entry.
```

void **Enter**(Int_t *entry*)

```
Enter element entry into the list
```

Int_t **GetEntry**(Int_t *index*) const

```
Return value of entry at index in the list.
Return -1 if index is not in the list range
```

Int_t **GetIndex**(Int_t *entry*) const

```
Return index in the list of element with value entry
array is supposed  to be sorted prior to this call.
If match is found, function returns position of element.
If no match found, function gives nearest element smaller than entry.
```

void **Print**(Option_t **option*)

```
Print contents of this list
```

void **Resize**(Int_t *delta*)

```
Resize list by delta entries
if delta = 0 , remove unused element from the list;
```

void **Sort**( )

```
Sort list entries in increasing order
```

void **Streamer**(TBuffer &*b*)

```
Stream an object of class TEventList.
```

void **Subtract**(const TEventList **alist*)

```
Remove elements from this list that are present in alist.
```

Author: René Brun 11/02/97
Last update: 1.00/09 21/04/97 16.24.21 by Fons Rademakers

TF1

class **TF1** : public TFormula, public TAttLine, public TAttFill, public TAttMarker

public:

class TF1	**TF1**(char* *name,* char* *formula,* Float_t *xmin = 0,* Float_t *xmax = 1*)
class TF1	**TF1**(char* *name,* void* *fcn,* Float_t *xmin = 0,* Float_t *xmax = 1,* Int_t *npar = 0*)
class TF1	**TF1**(char* name, Double_t (*)(Double_t *, Double_t *) fcn, Float_t xmin = 0, Float_t xmax = 1, Int_t npar = *0*)
class TF1	**TF1**( )
class TF1	**TF1**(class TF1& *f1*)
void	**~TF1**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *f1*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**DrawF1**(char* *formula,* Float_t *xmin,* Float_t *xmax,* Option_t* *option*)
virtual void	**DrawPanel**( )
virtual Double_t	**Eval**(Double_t *x,* Double_t *y = 0,* Double_t *z = 0*)
virtual Double_t	**EvalPar**(Double_t* *x,* Double_t* *params = 0*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Double_t	**GetChisquare**( )
class TH1F*	**GetHistogram**( )
class TMethodCall*	**GetMethodCall**( )
Int_t	**GetNpx**( )
Int_t	**GetNumberFitPoints**( )
class TObject*	**GetParent**( )
Double_t	**GetParError**(Int_t *ipar*)
virtual void	**GetParLimits**(Int_t *ipar,* Double_t& *parmin,* Double_t& *parmax*)
virtual Double_t	**GetRandom**( )
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *xmax*)
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *zmin,* Float_t& *xmax,* Float_t& *ymax,* Float_t& *zmax*)
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *xmax,* Float_t& *ymax*)
Float_t	**GetXmax**( )
Float_t	**GetXmin**( )
virtual void	**InitArgs**(Double_t* *x,* Double_t* *params*)
virtual Double_t	**Integral**(Float_t *a,* Float_t *b,* Double_t* *params = 0,* Float_t *epsilon =* 0.*000001*)
virtual class TClass*	**IsA**( )
virtual Int_t	**LocateBinary**(Double_t *value,* Double_t* *array,* Int_t *nbins*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetChisquare**(Double_t *chi2*)
virtual void	**SetMaximum**(Float_t *maximum = -1111*)
virtual void	**SetMinimum**(Float_t *minimum = -1111*)
virtual void	**SetNpx**(Int_t *npx = 100*)
virtual void	**SetNumberFitPoints**(Int_t *npfits*)
virtual void	**SetParent**(class TObject* *p = 0*)
virtual void	**SetParError**(Int_t *ipar,* Double_t *error*)
virtual void	**SetParLimits**(Int_t *ipar,* Double_t *parmin,* Double_t *parmax*)
virtual void	**SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *zmin,* Float_t *xmax,* Float_t

	ymax, Float_t *zmax*)
virtual void	**SetRange**(Float_t *xmin,* Float_t *xmax*)
virtual void	**SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

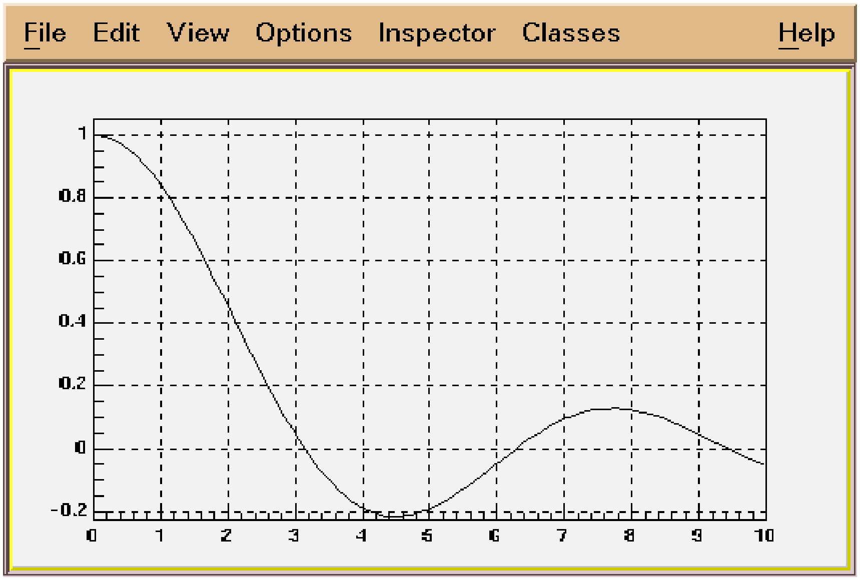
protected:

Float_t	**fXmin**	*Lower bounds for the range*
Float_t	**fXmax**	*Upper bounds for the range*
Int_t	**fNpx**	*Number of points used for the graphical representation*
Int_t	**fType**	*(=0 for standard functions, 1 if pointer to function)*
Int_t	**fNpfits**	*Number of points used in the fit*
Double_t	**fChisquare**	*Function fit chisquare*
Double_t*	**fIntegral**	*Integral of function binned on fNpx bins*
Double_t*	**fParErrors**	*Array of errors of the fNpar parameters*
Double_t*	**fParMin**	*Array of lower limits of the fNpar parameters*
Double_t*	**fParMax**	*Array of upper limits of the fNpar parameters*
class TObject*	**fParent**	*Parent object hooking this function (if one)*
class TH1F*	**fHistogram**	*Pointer to histogram used for visualisation*
Float_t	**fMaximum**	*Maximum value for plotting*
Float_t	**fMinimum**	*Minimum value for plotting*
class TMethodCall*	**fMethodCall**	*Pointer to MethodCall in case of interpreted function*
Double_t (*)(Double_t *, Double_t *)	**fFunction**	*Pointer to function*

See Also: TF2

Class Description

```
a TF1 object is a 1-Dim function defined between a lower and upper limit.
The function may be a simple function (see TFormula) or a precompiled
user function.
The function may have associated parameters.
TF1 graphics function is via the TH1/TGraph drawing functions.

The following types of functions can be created:
 A- Expression using variable x and no parameters
 B- Expression using variable x with parameters
 C- A general C function with parameters

   Example of a function of type A

TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
f1->Draw();
```




```
   Example of a function of type B
TF1 *f1 = new TF1("f1","[0]x*sin([1]*x",-3,3);
 This creates a function of variable x with 2 parameters.
 The parameters must be initialized via:
   f1->SetParameter(0,value_first_parameter);
   f1->SetParameter(1,value_second_parameter);
 Parameters may be given a name:
   f1->SetParName(0,"Constant");

  Example of function of type C
Consider the macro myfunc.C below
-------------macro myfunc.C----------------------------
Double_t myfunction(Double_t *x, Double_t *par)
{
Float_t xx =x[0];
Double_t f = TMath::Abs(par[0]*sin(par[1]*xx)/xx);
return f;
}
void myfunc()
{
TF1 *f1 = new TF1("myfunc",myfunction,0,10,2);
f1->SetParameters(2,1);
f1->SetParNames("constant","coefficient");
f1->Draw();
}
void myfit()
{
TH1F *h1=new TH1F("h1","test",100,0,10);
h1->FillRandom("myfunc",20000);
TF1 *f1=gROOT->GetFunction("myfunc");
f1->SetParameters(800,1);
h1.Fit("myfunc");
}
--------end of macro myfunc.C---------------------------------

In an interactive session you can do:
Root > .L myfunc.C
Root > myfunc();
Root > myfit();
```

TF1( ): **TFormula**( ), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
F1 default constructor
```

TF1(const char **name,* const char **formula,* Float_t *xmin,* Float_t *xmax*):**TFormula**(*name, formula*), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
F1 constructor using a formula definition
See TFormula constructor for explanation of the formula syntax.

Creates a function of type A or B between xmin and xmax
```

TF1(const char **name,* void **fcn,* Float_t *xmin,* Float_t *xmax,* Int_t *npar*):**TFormula**( ), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
F1 constructor using pointer to an interpreted function
See TFormula constructor for explanation of the formula syntax.

Creates a function of type C between xmin and xmax.
The function is defined with npar parameters
fcn must be a function of type:
  Double_t fcn(Double_t *x, Double_t *params)

This constructor is called for functions of type C by CINT.
```

TF1(const char *name, **Double_t**(**fcn*)(Double_t *, Double_t *), Float_t xmin, Float_t xmax, Int_t npar):**TFormula**( ), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
F1 constructor using a pointer to real function
npar is the number of free parameters used by the function

This constructor creates a function of type C when invoked
with the normal C++ compiler.
```

~TF1( )

```
F1 default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this F1 to a new F1
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a function
Compute the closest distance of approach from point px,py to this function.
The distance is computed in pixels units.

Algorithm:
```

void **Draw**(Option_t **option*)

```
Draw this function with its current attributes
Possible option values are:
"SAME"  superimpose on top of existing picture
"L"     connect all computed points with a straight line
"C"     connect all computed points with a smooth curve.

Note that the default value is "L". Therefore to draw on top
of an existing picture, specify option "LSAME"

NB. You must use DrawCopy if you want to draw several times the same
  function in the current canvas.
```

void **DrawCopy**(Option_t **option*)

```
Draw a copy of this function with its current attributes
This function MUST be used instead of Draw when you want to draw
the same function with different parameters settings in the same canvas.

Possible option values are:
"SAME"  superimpose on top of existing picture
"L"     connect all computed points with a straight line
"C"     connect all computed points with a smooth curve.
```

```
Note that the default value is "L". Therefore to draw on top
of an existing picture, specify option "LSAME"
```

void **DrawF1**(const char **formula,* Float_t *xmin,* Float_t *xmax,* Option_t **option*)

```
Draw formula between xmin and xmax
```

void **DrawPanel**( )

```
Display a panel with all function drawing options
See class TDrawPanelHist for example
```

Double_t **Eval**(Double_t *x,* Double_t *y,* Double_t *z*)

```
Evaluate this formula
Computes the value of this function (general case for a 3-d function)
at point x,y,z.
For a 1-d function give y=0 and z=0
The current value of variables x,y,z is passed through x, y and z.
The parameters used will be the ones in the array params if params is given
 otherwise parameters will be taken from the stored data members fParams
```

Double_t **EvalPar**(Double_t **x,* Double_t **params*)

```
Evaluate function with given coordinates and parameters
   Compute the value of this function at point defined by array x
   and current values of parameters in array params.
   If argument params is omitted or equal 0, the internal values
   of parameters (array fParams) will be used instead.
   For a 1-D function only x[0] must be given.
   In case of a multi-dimemsional function, the arrays x must be
   filled with the corresponding number of dimensions.
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a F1 is clicked with the locator
```

void **GetParLimits**(Int_t *ipar,* Double_t *&parmin,* Double_t *&parmax*)

```
Return limits for parameter ipar
```

Double_t **GetRandom**( )

```
Return a random number following this function shape
The distribution contained in the function fname (TF1) is integrated
over the channel contents.
It is normalized to 1.
Getting one random number implies:
  - Generating a random number between 0 and 1 (say r1)
  - Look in which bin in the normalized integral r1 corresponds to
  - make a linear interpolation in the returned bin
```

void **GetRange**(Float_t *&xmin,* Float_t *&xmax*)

```
Return range of a 1-D function
```

void **GetRange**(Float_t *&xmin,* Float_t *&ymin,* Float_t *&xmax,* Float_t *&ymax*)

```
Return range of a 2-D function
```

void **GetRange**(Float_t *&xmin,* Float_t *&ymin,* Float_t *&zmin,* Float_t *&xmax,* Float_t *&ymax,* Float_t *&zmax*)

```
Return range of function
```

void **InitArgs**(Double_t **x,* Double_t **params*)

```
Initialize parameters addresses
```

Double_t **Integral**(Float_t *a,* Float_t *b,* Double_t **params,* Float_t *epsilon*)

```
Return Integral of function between a and b

based on original CERNLIB routine DGAUSS by Sigfried Kolbig
converted to C++ by René Brun
```

This function computes,
to an attempted specified accuracy, the value of the integral

$$I = \int_A^B f(x)dx.$$

Usage:

In any arithmetic expression, this function
has the approximate value of the integral I.

a,b
End-points of integration interval. Note that B
may be less than A.
params
Array of function parameters. If 0, use current parameters.
epsilon
Accuracy parameter (see **Accuracy**).

Method:

For any interval [*a*,*b*] we define to be the
$g_{16}(a,b)$
8-point and 16-point Gaussian quadrature approximations to

$$\int_a^b f(x)dx$$

and define

$$r(a,b) = \frac{|g_{16}(a,b) - g_8(a,b)|}{1 + |g_{16}(a,b)|}.$$

Then,

$$G = \sum_{i=1}^{k} g_{16}(x_{i-1}, x_i),$$

where, starting with ,
$x_k = B$
the subdivision points are given by
$x_i \ (i = 1, 2, \ldots)$

$$x_i = x_{i-1} + \lambda(B - x_{i-1}),$$

with equal to the first member of the sequence
λ
for which
$1, \frac{1}{2}, \frac{1}{4}, \ldots$
.
$r(x_{i-1}, x_i) < \mathrm{EPS}$
If, at any stage in the process of subdivision, the ratio

$$q = \left| \frac{x_i - x_{i-1}}{B - A} \right|$$

```
is so small that 1+0.005q is indistinguishable from 1 to
machine accuracy, an error exit occurs with the function value
set equal to zero.
```

Accuracy:

```
Unless there is severe cancellation of positive and negative
values of f(x) over the interval [A,B], the argument EPS
may be considered as specifying a bound on the relative error of
I in the case  |I|>1, and a bound on the absolute error in
the case |I|<1. More precisely, if k is the number of sub-intervals
contributing to the approximation (see Method), and if
```

$$I_{abs} = \int_A^B |f(x)|dx,$$

```
then the relation
```

$$\frac{|G - I|}{I_{abs} + k} < \mathrm{EPS}$$

```
will nearly always be true, provided the routine terminates
without printing an error message. For functions
f having no singularities in the closed interval [A,B]
the accuracy will usually be much higher than this.
```

Error handling:

```
The requested accuracy cannot be
obtained (see Method).
The function value is set equal to zero.
```

Notes:

```
Values of the function f(x) at the interval end-points
A and B are not required. The subprogram may therefore
be used when these values are undefined.

---------------------------------------------------------------
```

Int_t **LocateBinary**(Double_t *value,* Double_t **array,* Int_t *nbins*)

```
Binary search in an array of nbins to locate value
array is supposed  to be sorted prior to this call.
If match is found, function returns position of element.
If no match found, function gives nearest element smaller than value.
```

void **Paint**(Option_t **option*)

```
Paint this function with its current attributes
```

void **Print**(Option_t **option*)

```
Dump this function with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetNpx**(Int_t *npx*)

```
Set the number of points used to draw the function
```

void **SetParLimits**(Int_t *ipar,* Double_t *parmin,* Double_t *parmax*)

```
Set limits for parameter ipar
```

```
  The specified limits will be used in a fit operation
  when the option "B" is specified (Bounds).
```

void **SetRange**(Float_t *xmin,* Float_t *xmax*)

```
Initialize the upper and lower bounds to draw the function
  The function range is also used in an histogram fit operation
  when the option "R" is specified.
```

void **SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)

```
Initialize the upper and lower bounds to draw the function
```

void **SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *zmin,* Float_t *xmax,* Float_t *ymax,* Float_t *zmax*)

```
Initialize the upper and lower bounds to draw the function
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 18/08/95
Last update: 08/11/97 15.49.25 by René Brun

TF2

class **TF2** : public TF1

public:

class TF2	**TF2**(char* *name,* char* *formula,* Float_t *xmin = 0,* Float_t *xmax = 1,* Float_t *ymin = 0,* Float_t *ymax = 1*)
class TF2	**TF2**(char* *name,* void* *fcn,* Float_t *xmin = 0,* Float_t *xmax = 1,* Float_t *ymin = 0,* Float_t *ymax = 1,* Int_t *npar = 0*)
class TF2	**TF2**(char* name, Double_t (*)(Double_t *, Double_t *) fcn, Float_t xmin = 0, Float_t xmax = 1, Float_t ymin = 0, Float_t ymax = 1, Int_t npar = *0*)
class TF2	**TF2**( )
class TF2	**TF2**(class TF2& *f2*)
void	**~TF2**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *f2*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**DrawF2**(char* *formula,* Float_t *xmin,* Float_t *xmax,* Float_t *ymin,* Float_t *ymax,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Int_t	**GetContour**( )
Int_t	**GetNpy**( )
Double_t	**GetRandom**( )
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *xmax,* Float_t& *ymax*)
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *zmin,* Float_t& *xmax,* Float_t& *ymax,* Float_t& *zmax*)
virtual Float_t	**GetYmax**( )
virtual Float_t	**GetYmin**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SetContour**(Int_t *nl = 20*)
virtual void	**SetNpy**(Int_t *npy = 100*)
virtual void	**SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

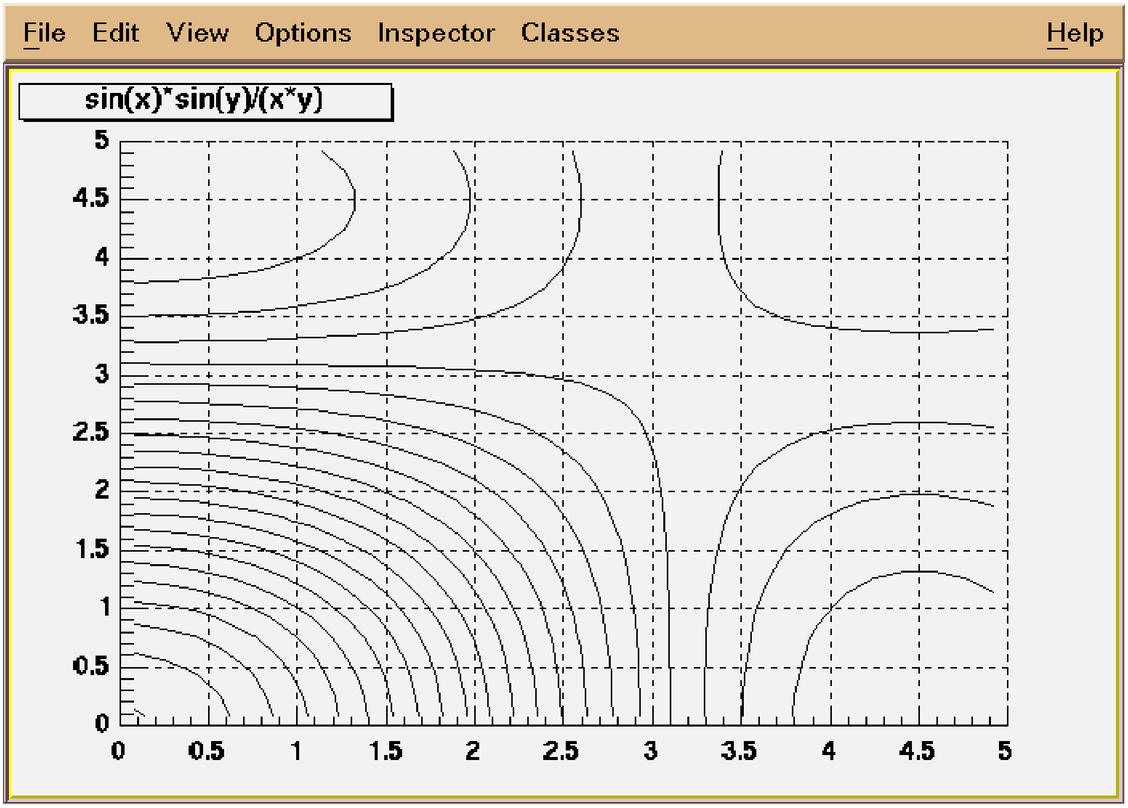
protected:

Float_t	**fYmin**	*Lower bound for the range in y*
Float_t	**fYmax**	*Upper bound for the range in y*
Int_t	**fNpy**	*Number of points along y used for the graphical representation*
Int_t	**fNlevels**	*Number of contour levels*

See Also: TF3

Class Description

```
a 2-Dim function with parameters
TF2 graphics function is via the TH1 drawing functions.

   Example of a function

TF2 *f2 = new TF2("f2","sin(x)*sin(y)/(x*y)",0,5,0,5);
f2->Draw();
```




```
See TF1 class for the list of functions formats
```

TF2( ): **TF1**( )

```
F2 default constructor
```

TF2(const char **name,* const char **formula,* Float_t *xmin,* Float_t *xmax,* Float_t *ymin,* Float_t *ymax*):**TF1**(*name, formula, xmin, xmax*)

```
F2 constructor using a formula definition
See TFormula constructor for explanation of the formula syntax.
```

TF2(const char **name,* void **fcn,* Float_t *xmin,* Float_t *xmax,* Float_t *ymin,* Float_t *ymax,* Int_t *npar*): **TF1**(*name, fcn, xmin, xmax, npar*)

```
F2 constructor using a pointer to an interpreted function
npar is the number of free parameters used by the function

Creates a function of type C between xmin and xmax and ymin,ymax.
The function is defined with npar parameters
fcn must be a function of type:
  Double_t fcn(Double_t *x, Double_t *params)

This constructor is called for functions of type C by CINT.
```

TF2(const char *name, **Double_t**(**fcn*)(Double_t *, Double_t *), Float_t xmin, Float_t xmax, Float_t ymin, Float_t ymax, Int_t npar): **TF1**(*name, fcn, xmin, xmax, npar*)

```
F2 constructor using a pointer to a compiled function
npar is the number of free parameters used by the function

This constructor creates a function of type C when invoked
with the normal C++ compiler.
```

~**TF2**( )

```
F2 default destructor
```

void **Copy**(TObject &*obj*)
```
Copy this F2 to a new F2
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a function
Compute the closest distance of approach from point px,py to this function.
The distance is computed in pixels units.

Algorithm:
```

void **Draw**(Option_t **option*)
```
Draw this function with its current attributes
NB. You must use DrawCopy if you want to draw several times the same
  function in the current canvas.
```

void **DrawCopy**(Option_t **option*)
```
Draw a copy of this function with its current attributes
This function MUST be used instead of Draw when you want to draw
the same function with different parameters settings in the same canvas.

Possible option values are:
"SAME"  superimpose on top of existing picture
"L"     connect all computed points with a straight line
"C"     connect all computed points with a smooth curve.

Note that the default value is "F". Therefore to draw on top
of an existing picture, specify option "SL"
```

void **DrawF2**(const char **formula,* Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Option_t **option*)
```
Draw formula between xmin,ymin and xmax,ymax
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function is called when a F2 is clicked with the locator
```

Double_t **GetRandom**( )
```
Return a random number following this function shape
```

void **GetRange**(Float_t &*xmin,* Float_t &*ymin,* Float_t &*xmax,* Float_t &*ymax*)
```
Return range of a 2-D function
```

void **GetRange**(Float_t &*xmin,* Float_t &*ymin,* Float_t &*zmin,* Float_t &*xmax,* Float_t &*ymax,* Float_t &*zmax*)
```
Return range of function
```

void **Paint**(Option_t **option*)
```
Paint this 2-D function with its current attributes
```

void **SetNpy**(Int_t *npy*)
```
Set the number of points used to draw the function
```

void **SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
```
Initialize the upper and lower bounds to draw the function
```

Author: René Brun 23/08/95
Last update: 1.03/06 24/10/97 09.30.54 by René Brun

TF3

class **TF3** : public TF2

public:

class TF3	**TF3**(char* *name,* char* *formula,* Float_t *xmin = 0,* Float_t *ymin = 0,* Float_t *zmin = 0,* Float_t *xmax = 1,* Float_t *ymax = 1,* Float_t *zmax = 1*)
class TF3	**TF3**(char* name, Double_t (*)(Double_t *, Double_t *) fcn, Int_t npar = *0*)
class TF3	**TF3**( )
class TF3	**TF3**(class TF3& *f3*)
void	**~TF3**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *f3*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Int_t	**GetNpz**( )
virtual void	**GetRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *zmin,* Float_t& *xmax,* Float_t& *ymax,* Float_t& *zmax*)
virtual Float_t	**GetZmax**( )
virtual Float_t	**GetZmin**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SetNpz**(Int_t *npz = 30*)
virtual void	**SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *zmin,* Float_t *xmax,* Float_t *ymax,* Float_t *zmax*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fZmin**	*Lower bound for the range in z*
Float_t	**fZmax**	*Upper bound for the range in z*
Int_t	**fNpz**	*Number of points along z used for the graphical representation*

Class Description

```
a 3-Dim function with parameters
```

TF3( ): **TF2**( )

```
F3 default constructor
```

TF3(const char **name,* const char **formula,* Float_t *xmin,* Float_t *ymin,* Float_t *zmin,* Float_t *xmax,* Float_t *ymax,* Float_t *zmax*):**TF2**(*name, formula, xmin, ymin, xmax, ymax*)

```
F3 constructor using a formula definition
See TFormula constructor for explanation of the formula syntax.
```

TF3(const char *name, **Double_t**(**fcn*)(Double_t *, Double_t *), Int_t npar):**TF2**(*name, fcn, npar*)

```
F3 constructor using a pointer to real function
npar is the number of free parameters used by the function

For example, for a 2-dim function with 3 parameters, the user function
   looks like:
 Double_t fun1(Double_t *x, Double_t *par)
     return par[0] + par[1]*exp(par[2]*x[0]*x[1]);
```

~TF3( )

```
F3 default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this F3 to a new F3
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a function
Compute the closest distance of approach from point px,py to this function.
The distance is computed in pixels units.

Algorithm:
```

void **Draw**(Option_t **option*)

```
Draw this function with its current attributes
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a F3 is clicked with the locator
```

void **GetRange**(Float_t &*xmin,* Float_t &*ymin,* Float_t &*zmin,* Float_t &*xmax,* Float_t &*ymax,* Float_t &*zmax*)

```
Return range of function
```

void **Paint**(Option_t **option*)

```
Paint this 2-D function with its current attributes
```

void **SetNpz**(Int_t *npz*)

```
Set the number of points used to draw the function
```

void **SetRange**(Float_t *xmin,* Float_t *ymin,* Float_t *zmin,* Float_t *xmax,* Float_t *ymax,* Float_t *zmax*)

```
Initialize the upper and lower bounds to draw the function
```

Author: René Brun 27/10/95
Last update: 1.00/07 04/04/97 08.50.37 by René Brun

TFatalErrorDialog

class **TFatalErrorDialog** : public TErrorDialog

protected:

class TFatalErrorDialog	**TFatalErrorDialog**( )

public:

class TFatalErrorDialog	**TFatalErrorDialog**(char* *name*)
class TFatalErrorDialog	**TFatalErrorDialog**(class TFatalErrorDialog&)
void	**~TFatalErrorDialog**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
void	**Ok**(Widget *w,* XtPointer *cd*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TFatalErrorDialog

This class handles fatal error dialogs.
```

void **Ok**(*Widget, XtPointer*)

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TFile

class **TFile** : public TDirectory

protected:

void	**Init**(Bool_t *create*)

public:

class TFile	**TFile**(Text_t* *fname,* Option_t* *option,* Text_t* *ftitle,* Int_t *compress = 1*)
class TFile	**TFile**( )
class TFile	**TFile**(class TFile& *file*)
void	**~TFile**( )
class TClass*	**Class**( )
virtual void	**Close**( )
virtual void	**Copy**(class TObject& *file*)
virtual void	**Delete**(Text_t* *namecycle*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**FillBuffer**(char*& *buffer*)
virtual void	**Flush**( )
Int_t	**GetBestBuffer**( )
UInt_t	**GetBytesRead**( )
UInt_t	**GetBytesWritten**( )
Float_t	**GetCompressionFactor**( )
Int_t	**GetCompressionLevel**( )
virtual Seek_t	**GetEND**( )
Int_t	**GetFd**( )
UInt_t	**GetFileBytesRead**( )
UInt_t	**GetFileBytesWritten**( )
class TList*	**GetListOfFree**( )
virtual Int_t	**GetNfree**( )
const Option_t*	**GetOption**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsOpen**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**MakeFree**(Seek_t *first,* Seek_t *last*)
virtual void	**Map**( )
class TFile*	**Open**(Text_t* *name,* Option_t* *option,* Text_t* *ftitle,* Int_t *compress = 1*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void*	**Read**(Option_t* *option*)
virtual Bool_t	**ReadBuffer**(char* *buf,* int *len*)
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**ReadFree**( )
virtual void	**Recover**( )
virtual void	**Seek**(Seek_t *offset,* enum TFile::ERelativeTo *pos = kBeg*)
virtual void	**SetCompressionLevel**(Int_t *level = 1*)
virtual void	**SetEND**(Seek_t *last*)
void	**SetFileBytesRead**(UInt_t *bytes = 0*)
void	**SetFileBytesWritten**(UInt_t *bytes = 0*)
virtual void	**SetOption**(Option_t* *option* = >)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)
void	**SumBuffer**(Int_t *bufsize*)
virtual void	**Write**(Option_t* *option*)
virtual Bool_t	**WriteBuffer**(char* *buf,* int *len*)
virtual void	**WriteFree**( )
virtual void	**WriteHeader**( )

Data Members

protected:

Int_t	**fD**	*File descriptor*
Seek_t	**fBEGIN**	*First used byte in file*
Seek_t	**fEND**	*Last used byte in file*
Int_t	**fVersion**	*File format version*
Int_t	**fCompress**	*(=1 file is compressed, 0 otherwise)*
class TString	**fOption**	*File options*
Char_t	**fUnits**	*Number of bytes for file pointers*
Seek_t	**fSeekFree**	*Location on disk of free segments structure*
Int_t	**fNbytesFree**	*Number of bytes for free segments structure*
Int_t	**fWritten**	*Number of objects written so far*
Double_t	**fSumBuffer**	*Sum of buffer sizes of objects written so far*
Double_t	**fSum2Buffer**	*Sum of squares of buffer sizes of objects written so far*
class TList*	**fFree**	*Free segments linked list table*
UInt_t	**fBytesWrite**	*Number of bytes written to this file*
UInt_t	**fBytesRead**	*Number of bytes read from this file*
static UInt_t	**fgBytesWrite**	*Number of bytes written by all TFile objects*
static UInt_t	**fgBytesRead**	*Number of bytes read by all TFile objects*
static const Int_t	**kBegin**	
static const Char_t	**kUnits**	

public:

static const enum TFile::ERelativeTo**kBeg**
static const enum TFile::ERelativeTo**kCur**
static const enum TFile::ERelativeTo**kEnd**

See Also: TNetFile, TWebFile

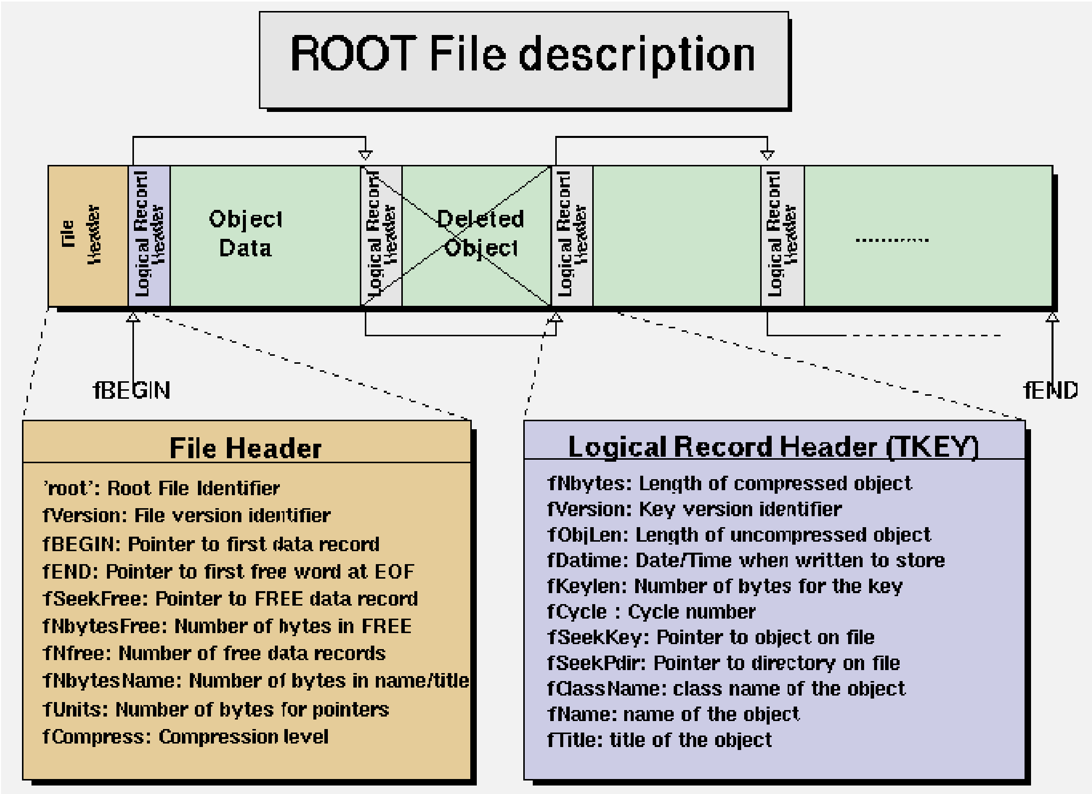
TFile( ) : **TDirectory**( )

```
File default Constructor
```

TFile(const Text_t **fname1,* Option_t **option,* const Text_t **ftitle,* Int_t *compress*):**TDirectory**(*fname1, ftitle,* "FILE")

```
File Constructor
If Option = NEW or CREATE   create a new file and open it for writing,
                           if the file already exists the file is
                           not opened.
         = RECREATE        create a new file, if the file already
                           exists it will be overwritten.
         = UPDATE          open an existing file for writing.
         = READ            open an existing file for reading.
Use IsOpen() to check if the file is opened successfuly.

The parameter name is used to identify the file in the current job
 name may be different in a job writing the file and another job
 reading/writing the file.
When the file is created, the name of the file seen from the file system
 is fname. It is recommended to specify fname as "file.root"
 The suffix root will be used by object browsers to automatically
 identify ROOT files.
The title of the file (ftitle) will be shown by the ROOT browsers.

A ROOT file (like a Unix file system) may contain objects and
 directories. There are no restrictions for the number of levels
 of directories.

A ROOT file is designed such that one can write in the file in pure
 sequential mode (case of BATCH jobs). In this case, the file may be
 read sequentially again without using the file index written
```

```
 at the end of the file. In case of a job crash, all the information
 on the file is therefore protected.
A ROOT file can be used interactively. In this case, one has the
 possibility to delete existing objects and add new ones.
When an object is deleted from the file, the freed space is added
into the FREE linked list (lFree). The FREE list consists of a chain
of consecutive free segments on the file. At the same time, the first
4 bytes of the freed record on the file are overwritten by GAPSIZE
where GAPSIZE = -(Number of bytes occupied by the record).

compress = 0 objects written to this file will not be compressed.
compress = 1 minimal compression level but fast.
....
compress = 9 maximal compression level but slow.

Note that the compression level may be changed at any time.
The new compression level will only apply to newly written objects.
The function TFile::Map shows the compression factor
for each object written to this file.
The function TFile::GetCompressionFactor returns the global
compression factor for this file.

A ROOT file is a suite of consecutive data records with the following
 format (see also the TKey class);
TKey ---------------------
   byte 1->4  Nbytes    = Length of compressed object (in bytes)
        5->6  Version   = TKey version identifier
        7->10 ObjLen    = Length of uncompressed object
       11->14 Datime    = Date and time when object was written to file
       15->16 KeyLen    = Length of the key structure (in bytes)
       17->18 Cycle     = Cycle of key
       19->22 SeekKey   = Pointer to record itself (consistency check)
       23->26 SeekPdir  = Pointer to directory header
       27->27 lname     = Number of bytes in the class name
       28->.. ClassName = Object Class Name
       ..->.. lname     = Number of bytes in the object name
       ..->.. Name      = lName bytes with the name of the object
       ..->.. lTitle    = Number of bytes in the object title
       ..->.. Title     = Title of the object
       -----> DATA      = Data bytes associated to the object

The first data record starts at byte fBEGIN (currently set to kBegin)
Bytes 1->kBegin contain the file description:
    byte  1->4  "root"      = Root file identifier
          5->8  fVersion    = File format version
          9->12 fBEGIN      = Pointer to first data record
         13->16 fEND        = Pointer to first free word at the EOF
         17->20 fSeekFree   = Pointer to FREE data record
         21->24 fNbytesFree = Number of bytes in FREE data record
         25->28 nfree       = Number of free data records
         29->32 fNbytesName = Number of bytes in TNamed at creation time
         33->33 fUnits      = Number of bytes for file pointers
         34->37 fCompress   = Zip compression level
```




```
The structure of a directory is shown in TDirectory::TDirectory
```

~**TFile**( )
```
File destructor
```

void **Init**(Bool_t *create*)
```
Initialize a TFile object.
```

void **Close**( )
```
Close a file
```

void **Copy**(TObject &*obj*)
```
Copy content of "this" file onto file
```

void **Delete**(Text_t **namecycle*)
```
Delete object namecycle
namecycle identifies an object in the top directory of the file
namecycle has the format name;cycle
name  = * means all
cycle = * means all cycles (memory and keys)
cycle = "" or cycle = 9999 ==> apply to a memory object
When name=* use T* to delete subdirectories also

examples:
  foo   : delete object named foo in memory
  foo;1 : delete cycle 1 of foo on file
  foo;* : delete all cycles of foo on disk and also from memory
  *;2   : delete all objects on file having the cycle 2
  *;*   : delete all objects from memory and file
 T*;*   : delete all objects from memory and file and all subdirectories
```

void **Draw**(Option_t **option*)
```
Fill Graphics Structure and Paint
Loop on all objects (memory or file) and all subdirectories
```

void **Flush**( ) const

```
Synchornize a file's in-core and on-disk states.
```

void **FillBuffer**(char *&*buffer*)

```
Encode file output buffer
The file output buffer contains only the FREE data record
```

Int_t **GetBestBuffer**( )

```
Return the best buffer size of objects on this file
The best buffer size is estimated based on the current mean value
and standard deviation of all objects written so far to this file.
Returns mean value + one standard deviation.
```

Float_t **GetCompressionFactor**( )

```
Return the file compression factor
Add total number of compressed/uncompressed bytes for each key.
return ratio of the two.
```

void **ls**(Option_t **option*)

```
List File contents
Indentation is used to identify the file tree
Subdirectories are listed first
then objects in memory
then objects on the file
```

Bool_t **IsOpen**( ) const

```
Returns kTRUE in case file is open and kFALSE if file is not open.
```

void **MakeFree**(Seek_t *first,* Seek_t *last*)

```
Mark unused bytes on the file
The list of free segments is in the lFree linked list
When an object is deleted from the file, the freed space is added
into the FREE linked list (lFree). The FREE list consists of a chain
of consecutive free segments on the file. At the same time, the first
4 bytes of the freed record on the file are overwritten by GAPSIZE
where GAPSIZE = -(Number of bytes occupied by the record).
```

void **Map**( )

```
List the contents of a file sequentially
For each logical record found, it prints
  Date/Time  Record_Adress Logical_Record_Length  ClassName  CompressionFactor

Example of output
960122/105933  At:64        N=160       TFile
960122/105933  At:224       N=402       TH1F           CX = 2.09
960122/105934  At:626       N=1369      TH2F           CX = 5.57
960122/105934  At:1995      N=1761      TProfile       CX = 1.63
960122/105936  At:3756      N=181640    THN            CX = 1.10
960122/105936  At:185396    N=326       TFile
960122/105936  At:185722    N=98        TFile
```

void **Paint**(Option_t **option*)

```
Paint all objects in the file
```

void **Print**(Option_t **option*)

```
Print all objects in the file
```

Bool_t **ReadBuffer**(char **buf,* int *len*)

```
Read a buffer from the file. This is the basic low level read operation.
Returns kTRUE in case of failure.
```

void* **Read**(Option_t **option*)

```
Read objects from file into memory
```

```
If an object is already in memory, the memory copy is deleted
and the object is again read from the file.
The read callback routine for each object is called by TKey::Read().
The address of the callback routine is set in the class Dictionary()
function.
```

void **ReadBuffer**(char *&*buffer*)

```
Decode file input buffer
The file input buffer contains only the FREE data record
```

void **ReadFree**( )

```
Read the FREE linked list
Every file has a linked list (fFree) of free segments
This linked list has been written on the file via WriteFree
as a single data record
```

void **Recover**( )

```
Attempt to recover file if not correctly closed
```

void **Seek**(Seek_t *offset,* ERelativeTo *pos*)

```
Seek to a specific position in the file. Pos it either kBeg, kCur or kEnd.
```

void **SetCompressionLevel**(Int_t *level*)

```
Set level of compression for this file
level = 0 objects written to this file will not be compressed.
level = 1 minimal compression level but fast.
....
level = 9 maximal compression level but slow.

Note that the compression level may be changed at any time.
The new compression level will only apply to newly written objects.
The function TFile::Map shows the compression factor
for each object written to this file.
The function TFile::GetCompressionFactor returns the global
compression factor for this file.
```

Int_t **Sizeof**( ) const

```
Return the size in bytes of the file header
```

void **Streamer**(TBuffer &*b*)

```
Stream a TFile object.
```

void **SumBuffer**(Int_t *bufsize*)

```
Increment statistics for buffer sizes of objects in this file
```

void **Write**(Option_t **option*)

```
Write memory objects to this file
Loop on all objects in memory (including subdirectories)
A new key is created in the KEYS linked list for each object
The list of keys is then saved on the file (via WriteKeys)
as a single data record
The directory header info is rewritten on the directory header record
The linked list of FREE segments is written
The file header is written (bytes 1->fBEGIN)
```

Bool_t **WriteBuffer**(const char **buf,* int *len*)

```
Write a buffer to the file. This is the basic low level write operation.
Returns kTRUE in case of failure.
```

void **WriteFree**( )

```
Write FREE linked list on the file
The linked list of FREE segments (fFree) is written as a single data
record
```

void **WriteHeader**( )

```
Write File Header
```

TFile* **Open**(const Text_t **name,* Option_t **option,* const Text_t **ftitle,* Int_t *compress*)

```
Static member function allowing the creation/opening of either a
TFile, TNetFile or a TWebFile. The type of TFile depends on the
file name. If the file starts with "root:" a TNetFile object will
be returned, with "http:" a TWebFile, with "file:" or the default
a local TFile. However, before opening a file via TNetFile a check
is made to see if the URL specifies a local file. If that is the
case the file will be opened via a normal TFile. To force the opening
of a local file via a TNetFile use either TNetFile directly or specify
as host "localhost".
```

void **SetFileBytesWritten**(UInt_t *bytes*)

Author: René Brun 28/11/94
Last update: 1.03/02 03/09/97 17.10.42 by René Brun

TFileHandler

class **TFileHandler** : public TSysEvtHandler

protected:

class TFileHandler	**TFileHandler**( )

public:

class TFileHandler	**TFileHandler**(int *fd,* int *mask*)
class TFileHandler	**TFileHandler**(class TFileHandler&)
void	**~TFileHandler**( )
class TClass*	**Class**( )
int	**GetFd**( )
virtual Bool_t	**HasReadInterest**( )
virtual Bool_t	**HasWriteInterest**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**Notify**( )
virtual Bool_t	**ReadNotify**( )
virtual void	**Remove**( )
void	**SetFd**(int *fd*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual Bool_t	**WriteNotify**( )

Data Members

protected:

int	**fFileNum**	*File descriptor*
int	**fMask**	*Event mask*

Class Description

```
TSysEvtHandler

Abstract base class for handling system events.
```

TFileHandler(int *fd,* int *mask*)

```
Create a file descriptor event handler.
```

Bool_t **Notify**( )

```
Notify when event occured on descriptor associated with this handler.
```

Bool_t **ReadNotify**( )

```
Notify when something can be read from the descriptor associated with
this handler.
```

Bool_t **WriteNotify**( )

```
Notify when something can be written to the descriptor associated with
this handler.
```

Bool_t **HasReadInterest**( )

```
True if handler is interested in read events.
```

Bool_t **HasWriteInterest**( )

```
True if handler is interested in write events.
```

void **Remove**( )

```
Remove file event handler from system file handler list.
```

Author: Fons Rademakers 16/09/95
Last update: 1.03/02 25/08/97 16.23.18 by Fons Rademakers

TFileSelectionDialog

class **TFileSelectionDialog** : public TDialogManager

private:

void	**SetFilename**(char* *new_filename*)
void	**SetFilenameCallback**(*Widget, XtPointer, XtPointer*)

protected:

class TFileSelectionDialog	**TFileSelectionDialog**( )

public:

class TFileSelectionDialog	**TFileSelectionDialog**(char* *name*)
class TFileSelectionDialog	**TFileSelectionDialog**(class TFileSelectionDialog&)
void	**~TFileSelectionDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
const char*	**FileName**( )
virtual class TClass*	**IsA**( )
void	**SetDirectory**(char* *directory*)
void	**SetFilterPattern**(char* *pattern*)
void	**SetSelection**(char* *selection*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

char*	**fDirectory**
char*	**fPattern**
char*	**fSelection**
char*	**fFilename**

Class Description

```
TFileSelectionDialog

This class handles file selection dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TFileType

class **TFileType** : public TNamed

public:

class TFileType	**TFileType**(char* *name,* char* *type,* char* *title*)
class TFileType	**TFileType**(class TFileType&)
void	**~TFileType**( )
class TClass*	**Class**( )
void	**ExecuteDefaultAction**(class TBrowser* *b,* char* *filename*)
const char*	**GetType**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TString	**fType**	*File type (class or icon name*)

Class Description

```
TFileType describes a system file type.
File type name is of the form *.root, *.C, *.cxx
File type is either a Classname or an Icon name . In case of a class
  the icon used by the browser will be the class icon.
File title contains the action to be performed by a double-click
  in the browser.
```

TFileType(const Text_t **name,* const Text_t **type,* const Text_t **title*): **TNamed**(*name, title*)

```
FileType normal constructor
```

~TFileType( )

```
FileType default destructor
```

void **ExecuteDefaultAction**(TBrowser **b,* const char **filename*)

```
Executes default action/command when double-click in browser
```

Author: René Brun 28/06/96
Last update: 1.00/06 23/03/97 11.01.56 by René Brun

TFitPanel

class **TFitPanel** : public TDialogCanvas

public:

class TFitPanel	**TFitPanel**( )
class TFitPanel	**TFitPanel**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 300,* UInt_t *wh = 400*)
class TFitPanel	**TFitPanel**(class TFitPanel&)
void	**~TFitPanel**( )
virtual void	**AddOption**(Option_t* *option*)
virtual void	**Apply**(Text_t* *action*)
virtual void	**BuildStandardButtons**( )
class TClass*	**Class**( )
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
class TObject*	**GetObjectFit**( )
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetDefaults**( )
virtual void	**SetFunction**(Text_t* *function*)
virtual void	**SetSame**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

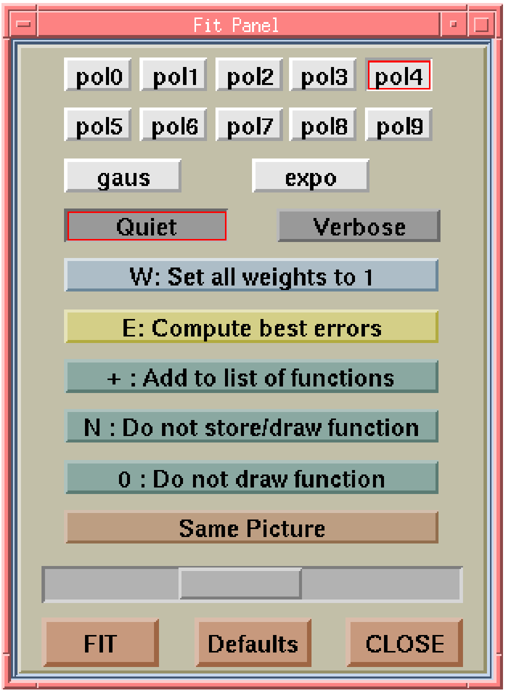
protected:

class TString	**fOption**	*Fitting options*
class TString	**fFunction**	*Function to fit*
class TString	**fSame**	*graphics option to superimpose new fit on existing picture*
class TSlider*	**fSlider**	*Pointer to fitpanel slider*
class TObject*	**fObjectFit**	*Pointer to object to fit*

See Also: TFitPanelGraph

Class Description

```
A FitPanel is a TDialogCanvas specialized to control histogram fits.
With the mouse, the user can control:
  - the type of function to be fitted
  - the various fit options
  - the drawing options
When the FIT button is executed, the selected histogram is fitted
with the current parameters.

One can select a range of the histogram to be fitted via the slider.

The options are documented in TH1::Fit.
```



TFitPanel( ) : **TDialogCanvas**( )
    `FitPanel default constructor`

TFitPanel(Text_t **name*, Text_t **title*, UInt_t *ww*, UInt_t *wh*): **TDialogCanvas**(*name, title, ww, wh*)
    `FitPanel constructor`

~**TFitPanel**( )
    `FitPanel default destructor`

void **AddOption**(Option_t **option*)
    `Add option to the current list of options`

void **Apply**(Text_t **action*)
    `Collect all options and fit histogram`

void **BuildStandardButtons**( )
    `Create FIT, Defaults and CLOSE buttons`

void **ExecuteEvent**(Int_t *event*, Int_t *px*, Int_t *py*)
```
Control mouse events when slider is used in a fitpanel
 This function is called by TPad::ExecuteEvent or TSliderBox::ExecuteEvent
 We return in the first case.
 When called by the slider,  px = 0 and py = 0
```

void **SavePrimitive**(ofstream &*out*, Option_t **option*)
    `Save this fitpanel in a macro`

void **SetDefaults**( )
    `Set default fit panel options`

void **SetFunction**(Text_t **function*)
    `Set the function to be used in the fit`

void **SetSame**( )
    `Set graphics option "same"`

Author: René Brun 24/11/96
Last update: 1.02/00 09/07/97 21.56.55 by René Brun

TFitPanelGraph

class **TFitPanelGraph** : public TFitPanel

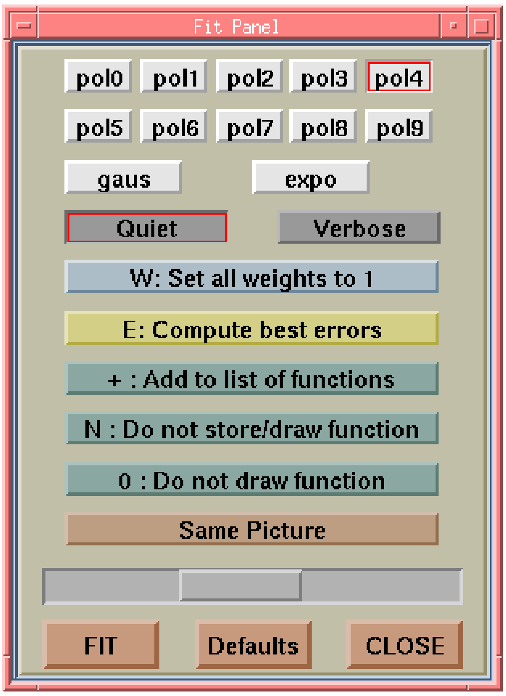
public:

class TFitPanelGraph	**TFitPanelGraph**( )
class TFitPanelGraph	**TFitPanelGraph**(Text_t* *name,* Text_t* *title,* UInt_t *ww = 300,* UInt_t *wh = 400*)
class TFitPanelGraph	**TFitPanelGraph**(class TFitPanelGraph&)
void	**~TFitPanelGraph**( )
virtual void	**Apply**(Text_t* *action*)
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
A TFitPanelGraph is a TFitPanel specialized to control graph fits.
With the mouse, the user can control:
  - the type of function to be fitted
  - the various fit options
  - the drawing options
When the FIT button is executed, the selected histogram is fitted
with the current parameters.
The options are documented in TGraph::Fit.
```



TFitPanelGraph( ) : **TFitPanel**( )

```
FitPanelGraph default constructor
```

TFitPanelGraph(Text_t **name,* Text_t **title,* UInt_t *ww,* UInt_t *wh*): **TFitPanel**(*name, title, ww, wh*)

```
FitPanelGraph constructor
```

~TFitPanelGraph( )

    FitPanelGraph default destructor

void **Apply**(Text_t **action*)

    Collect all options and fit histogram

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

    Save this fitpanelgraph in a macro

Author: René Brun 28/11/96
Last update: 1.03/02 29/08/97 10.32.52 by René Brun

TFont

class **TFont**

private:

Boolean	**ParseAFM**(FILE* *fp*)

protected:

class TFont	**TFont**( )

public:

class TFont	**TFont**(class TFont&)
class TFont	**TFont**(char* *xlfd,* char* *fontName = Courier,* float *pointSize* = 11*.0,* FontTraitMask_t *traitMask = 0*)
void	**~TFont**( )
float	**Ascent**( )
class TClass*	**Class**( )
float	**Descent**( )
const char*	**DisplayName**( )
const char*	**FamilyName**( )
Boolean	**HasAFMDictionary**( )
float	**HeightOfBoundingRect**(char aGlyph=(*char*)0)
virtual class TClass*	**IsA**( )
float	**PointSize**( )
class TFont*	**PrinterFont**( )
XFontStruct*	**ScreenFont**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
float	**StringWidth**(char* *string*)
float	**StringWidth**(char* *string,* int *num_chars*)
float	**WidthOfBoundingRect**(char aGlyph=(*char*)0)
const float*	**Widths**( )
float	**XAdvancementForGlyph**(char *aGlyph,* char *fromAGlyph* = *\0*)
float	**YAdvancementForGlyph**(char *aGlyph*)

Data Members

private:

XFontStruct*	**fFs**
char*	**fFontName**
float	**fPointSize**
char*	**fFamilyName**
char*	**fScreenName**
Boolean	**fHasAFM**

Class Description

```
TFont

MOTIF font manager. The majority of operations, however,
should be performed via the TFontManager instead of individual
fonts, so the application maintains information of these changes.
```

float **Descent**( )

Author: Fons Rademakers 30/03/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TFormula

class **TFormula** : public TNamed

public:

class TFormula	**TFormula**(char* *name,* char* *formula*)
class TFormula	**TFormula**( )
class TFormula	**TFormula**(class TFormula& *formula*)
void	**~TFormula**( )
virtual void	**Analyze**(char* *schain,* Int_t& *err*)
class TClass*	**Class**( )
virtual Int_t	**Compile**(Text_t* *expression*)
virtual void	**Copy**(class TObject& *formula*)
virtual char*	**DefinedString**(Int_t *code*)
virtual Double_t	**DefinedValue**(Int_t *code*)
virtual Int_t	**DefinedVariable**(class TString& *variable*)
virtual Double_t	**Eval**(Double_t *x,* Double_t *y = 0,* Double_t *z = 0*)
virtual Double_t	**EvalPar**(Double_t* *x,* Double_t* *params = 0*)
Int_t	**GetNdim**( )
Int_t	**GetNpar**( )
Int_t	**GetNumber**( )
Double_t	**GetParameter**(Int_t *ipar*)
virtual void	**GetParameters**(Double_t* *params*)
virtual const char*	**GetParName**(Int_t *ipar*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**SetNumber**(Int_t *number*)
virtual void	**SetParameter**(Int_t *ipar,* Double_t *parvalue*)
virtual void	**SetParameters**(Double_t *p0,* Double_t *p1,* Double_t *p2 = 0,* Double_t *p3 = 0,* Double_t *p4 = 0,* Double_t *p5 = 0,* Double_t *p6 = 0,* Double_t *p7 = 0,* Double_t *p8 = 0,* Double_t *p9 = 0*)
virtual void	**SetParameters**(Double_t* *params*)
virtual void	**SetParName**(Int_t *ipar,* char* *name*)
virtual void	**SetParNames**(char* *name0 = p0,* char* *name1 = p1,* char* *name2 = p2,* char* *name3 = p3,* char* *name4 = p4,* char* *name5 = p5,* char* *name6 = p6,* char* *name7 = p7,* char* *name8 = p8,* char* *name9 = p9*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNdim**	*Dimension of function (1=1-Dim, 2=2-Dim, etc*)
Int_t	**fNpar**	*Number of parameters*
Int_t	**fNoper**	*Number of operators*
Int_t	**fNconst**	*Number of constants*
Int_t	**fNumber**	*formula number identifier*
Int_t	**fNval**	*Number of different variables in expression*
Int_t	**fNstring**	*Number of different constants character strings*
class TString*	**fExpr**	*List of expressions*
Int_t*	**fOper**	*List of operators*
Double_t*	**fConst**	*Array of fNconst formula constants*
Double_t*	**fParams**	*Array of fNpar parameters*
class TString*	**fNames**	*Array of parameter names*

See Also: TF1, TTreeFormula

Class Description

```
The  F O R M U L A  class
This class has been implemented by  Nicolas Brun (age 18).
Example of valid expressions:
  -  sin(x)/x
  -  [0]*sin(x) + [1]*exp(-[2]*x)
  -  x + y**2
  -  x^2 + y^2
  -  [0]*pow([1],4)
  -  2*pi*sqrt(x/y)
  -  Gaus(0)*Expo(3) + yPol3(5)*x

In the last example above:
  Gaus(0) is a substitute for [0]*exp(-0.5*((x-[1])/[2])**2)
     and (0) means start numbering parameters at 0
  Expo(3) is a substitute for exp([3]+[4])*x)
  Pol3(5) is a substitute for par[5]+par[6]*x+par[7]*x**2+par[8]*x**3
      (here Pol3 stands for Polynomial of degree 3)

Comparisons operators are also supported (&&, ||, ==, <=, >=, !)
Examples:
   sin(x*(x<0.5 || x>1))
If the result of a comparison is TRUE, the result is 1, otherwise 0.

Already predefined names can be given. For example, if the formula
  TFormula old(sin(x*(x<0.5 || x>1))) one can assign a name to the formula. By
default
  the name of the object = title = formula itself.
  old.SetName("old").
  then, old can be reused in a new expression.
  TFormula new("x*old") is equivalent to:
  TFormula new("x*sin(x*(x<0.5 || x>1))")

Up to 3 dimensions are supported (indicated by x, y, z)
An expression may have 0 parameters or a list of parameters
indicated by the sequence [par_number]

A graph showing the logic to compile and analyze a formula
is shown in TFormula::Compile and TFormula::Analyze.
Once a formula has been compiled, it can be evaluated for a given
set of parameters. see graph in TFormula::EvalPar.

This class is the base class for the function classes TF1,TF2 and TF3.
It is also used by the ntuple selection mechanism TNtupleFormula.
```

TFormula( ): **TNamed**( )

```
Formula default constructor
```

TFormula(const char **name,* const char **expression*) :**TNamed**(*name,* expression)

```
Normal Formula constructor
```

~TFormula( )

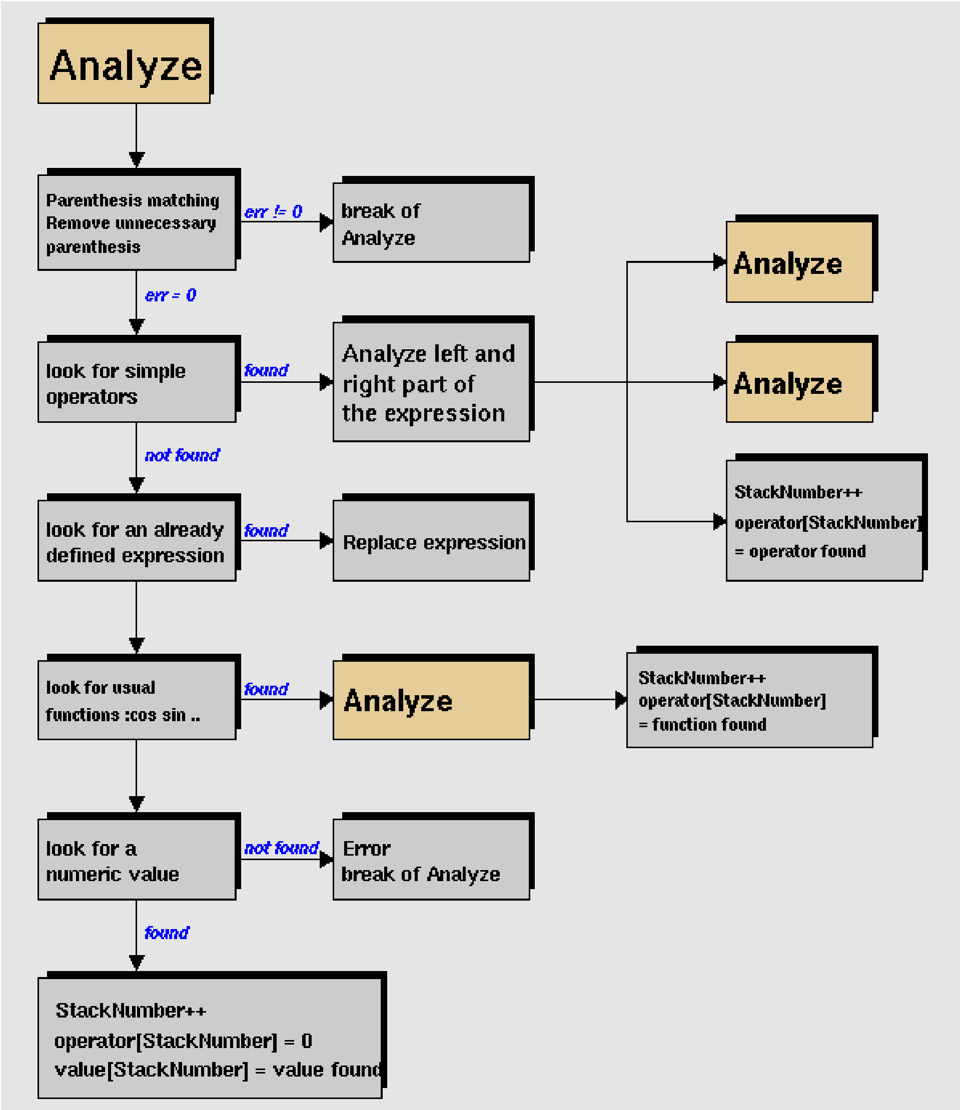
```
Formula default destructor
```

void **Analyze**(const char **schain,* Int_t &*err*)

```
Analyze a sub-expression in one formula
Expressions in one formula are recursively analyzed.
Result of analysis is stored in the object tables.

                Table of function codes and errors
* functions :

  +          1                   pow          20
```

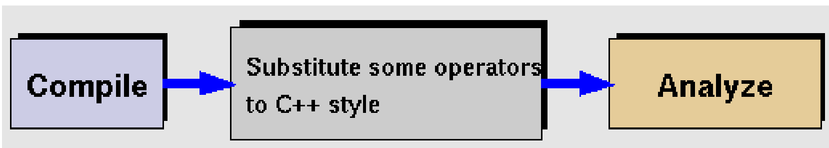
```
  -           2                      sq          21
  *           3                      sqrt        22
  /           4                      strstr      23

                                     log         30
  cos         10                     exp         31
  sin         11                     log10       32
  tan         12
  acos        13                     abs         41
  asin        14                     sign        42
  atan        15
  atan2       16                     rndm        50

  cosh        70                     acosh       73
  sinh        71                     asinh       74
  tanh        72                     atanh       75

  expo       10xx                    gaus       20xx
  expo(0)    1000                    gaus(0)    2000
  expo(1)    1001                    gaus(1)    2001
  xexpo      10xx                    xgaus      20xx
  yexpo      11xx                    ygaus      21xx
  zexpo      12xx                    zgaus      22xx
  xyexpo     15xx                    xygaus     25xx
  yexpo(5)   1105                    ygaus(5)   2105
  xyexpo(2)  1502                    xygaus(2)  2502

  pol0       100xx                   pol1      101xx
  pol0(0)    10000                   pol1(0)   10100
  pol0(1)    10001                   pol1(1)   10101
  xpol0      100xx                   xpol1     101xx
  ypol0      200xx                   ypol1     201xx
  zpol0      300xx                   zpol1     301xx
  ypol0(5)   20005                   ypol1(5)  20105

  pi          40

  &&          60                     <           64
  ||          61                     >           65
  ==          62                     <=          66
  !=          63                     =>          67
  !           68
  ==(string)  76                     &           78
  !=(string)  77                     |           79

* constants :

 c0  50000      c1  50001  etc..

* strings :

 s0  80000      s1  80001  etc..

* variables :

  x    100000     y    100001     z    100002     t    100003

* parameters :

  [1]        101
  [2]        102
  etc.
```

```
* errors :

  1  : Division By Zero
  2  : Invalid Floating Point Operation
  4  : Empty String
  5  : invalid syntax
  6  : Too many operators
  7  : Too many parameters
 10  : z specified but not x and y
 11  : z and y specified but not x
 12  : y specified but not x
 13  : z and x specified but not y
 20  : non integer value for parameter number
 21  : atan2 requires two arguments
 22  : pow requires two arguments
 23  : degree of polynomial not specified
 24  : Degree of polynomial must be positive
 25  : Degree of polynomial must be less than 20
 26  : Unknown name
 27  : Too many constants in expression
 28  : strstr requires two arguments
 40  : '(' is expected
 41  : ')' is expected
 42  : '[' is expected
 43  : ']' is expected
```



Int_t **Compile**(Text_t **expression*)

```
Compile expression already stored in fTitle
Loop on all subexpressions of formula stored in fTitle
```



void **Copy**(TObject &*obj*)

```
Copy this formula
```

char* **DefinedString**(Int_t *code*)

```
Return address of string corresponding to special code
This member function is inactive in the TFormula class.
It is redefined in derived classes such as TTreeFormula.
```

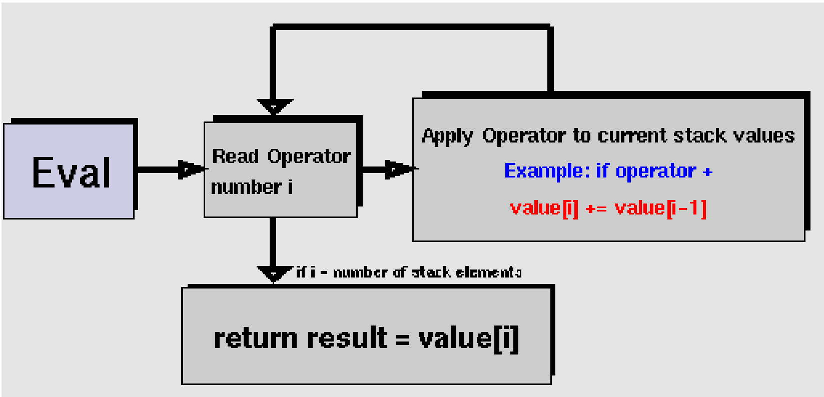
Double_t **DefinedValue**(Int_t *code*)

```
Return value corresponding to special code
This member function is inactive in the TFormula class.
It is redefined in derived classes such as TTreeFormula.
```

Int_t **DefinedVariable**(TString &*chaine*)
```
Check if expression is in the list of defined variables
This member function can be overloaded in derived classes
```

Double_t **Eval**(Double_t *x,* Double_t *y,* Double_t *z*)
```
Evaluate this formula
The current value of variables x,y,z is passed through x, y and z.
The parameters used will be the ones in the array params if params is given
 otherwise parameters will be taken from the stored data members fParams
```

Double_t **EvalPar**(Double_t **x,* Double_t **params*)
```
Evaluate this formula
The current value of variables x,y,z is passed through the pointer x.
The parameters used will be the ones in the array params if params is given
 otherwise parameters will be taken from the stored data members fParams
```



const char* **GetParName**(Int_t *ipar*) const
```
Return name of one parameter
```

void **Print**(Option_t **option*)
```
Dump this formula with its attributes
```

void **SetParameter**(Int_t *ipar,* Double_t *value*)
```
Initialize parameter number ipar
```

void **SetParameters**(Double_t **params*)
```
Initialize array of all parameters
```

void **SetParameters**(Double_t *p0,* Double_t *p1,* Double_t *p2,* Double_t *p3,* Double_t p4, Double_t p5, Double_t p6, Double_t p7, Double_t p8, Double_t *p9*)
```
Initialize up to 10 parameters
```

void **SetParNames**(const char**name0,* const char**name1,* const char**name2,* const char**name3,* const char**name4,* const char*name5, const char*name6, const char*name7, const char*name8, const char**name9*)
```
Set up to 10 parameter names
```

void **Streamer**(TBuffer &*b*)
```
Stream a class object
```

Author: Nicolas Brun 19/08/95
Last update: 03/11/97 15.53.47 by René Brun

TFrame

class **TFrame** : public TWbox

public:

class TFrame	**TFrame**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TFrame	**TFrame**( )
class TFrame	**TFrame**(class TFrame& *frame*)
void	**~TFrame**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *frame*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**Pop**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
a TFrame is a TWbox for drawing histogram frames.
```

TFrame( ): **TWbox**( )

```
frame default constructor
```

TFrame(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TWbox**(*x1, y1, x2, y2*)

```
frame normal constructor
```

~TFrame( )

```
wbox default destructor
```

void **Copy**(TObject &*frame*)

```
Copy this frame to frame
```

void **Draw**(Option_t **option*)

```
Draw this wbox with its current attributes
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a TFrame object is clicked.
```

void **Paint**(Option_t **option*)

```
Paint this wbox with its current attributes
```

void **Pop**( )

```
Do not pop frame's, if allowed they would cover the picture they frame.
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 31/10/96
Last update: 1.02/00 09/07/97 22.21.10 by René Brun

TFree

class **TFree** : public TObject

public:

class TFree	**TFree**( )
class TFree	**TFree**(Seek_t *first,* Seek_t *last*)
class TFree	**TFree**(class TFree&)
void	**~TFree**( )
class TFree*	**AddFree**(Seek_t *first,* Seek_t *last*)
class TClass*	**Class**( )
virtual void	**FillBuffer**(char*& *buffer*)
class TFree*	**GetBestFree**(Int_t *nbytes*)
Seek_t	**GetFirst**( )
Seek_t	**GetLast**( )
virtual class TClass*	**IsA**( )
virtual void	**ReadBuffer**(char*& *buffer*)
void	**SetFirst**(Seek_t *first*)
void	**SetLast**(Seek_t *last*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Seek_t	**fFirst**	*First free word of segment*
Seek_t	**fLast**	*Last free word of segment*

Class Description

```
Service class for TFile.
Each file has a linked list of free segments. Each free segment
is described by its firts and last address.
When an object is written to a file via TObject::Write, a new Key (see TKey)
is created. The first free segment big enough to accomodate the object
is used.
If the object size has a length corresponding to the size of the free segment,
the free segment is deleted from the list of free segments.
When an object is deleted from a file, a new TFree object is generated.
If the deleted object is contiguous to an already deleted object, the free
segments are merged in one single segment.
```

TFree( )

```
TFree default constructor
```

TFree(Seek_t *first,* Seek_t *last*)

```
Constructor for a FREE segment
```

TFree* **AddFree**(Seek_t *first,* Seek_t *last*)

```
Add a new free segment to the list of free segments
If last just preceedes an existing free segment, then first becomes
  the new starting location of the free segment.
if first just follows an existing free segment, then last becomes
  the new ending location of the free segment.
if first just follows an existing free segment AND last just preceedes
  an existing free segment, these two segments are merged into
  one single segment.
```

~TFree( )

```
TFree Destructor
```

void **FillBuffer**(char *&*buffer*)

```
Encode FREE structure into output buffer
```

TFree* **GetBestFree**(Int_t *nbytes*)

```
Return the best free segment where to store nbytes
```

void **ReadBuffer**(char *&*buffer*)

```
Decode one FREE structure from input buffer
```

Author: René Brun 28/12/94
Last update: 1.02/00 09/07/97 22.52.32 by René Brun

TFunction

class **TFunction** : public TDictionary

protected:

virtual void	**CreateSignature**( )

public:

class TFunction	**TFunction**((*unknown*) G__MethodInfo* info = 0)
class TFunction	**TFunction**(class TFunction&)
void	**~TFunction**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
class TList*	**GetListOfMethodArgs**( )
virtual const char*	**GetName**( )
Int_t	**GetNargs**( )
Int_t	**GetNargsOpt**( )
const char*	**GetReturnTypeName**( )
const char*	**GetSignature**( )
virtual const char*	**GetTitle**( )
virtual ULong_t	**Hash**( )
void*	**InterfaceMethod**( )
virtual class TClass*	**IsA**( )
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class G__MethodInfo*	**fInfo**	*pointer to CINT function info*
class TString	**fSignature**	*string containing function signature*
class TList*	**fMethodArgs**	*list of function arguments*

See Also: TMethod

Class Description

```
Global functions class (global functions are obtaine from CINT).
This class describes one single global function.
The TROOT class contains a list of all currently defined global
functions (accessible via TROOT::GetListOfGlobalFunctions()).
```

TFunction(G__MethodInfo **info*) : **TDictionary**( )

```
Default TFunction ctor. TFunctions are constructed in TROOT via
a call to TCint::UpdateListOfGlobalFunctions().
```

~TFunction( )

```
TFunction dtor deletes adopted G__MethodInfo.
```

void **CreateSignature**( )

```
Using the CINT method arg information to create a complete signature string.
```

TList* **GetListOfMethodArgs**( )

```
Return list containing the TMethodArgs of a TFunction.
```

const char* **GetName**( ) const

```
Get function name.
```

const char* **GetTitle**( ) const

```
Get function description string (comment).
```

const char* **GetReturnTypeName**( ) const
```
Get full type description of function return type, e,g.: "class TDirectory*".
```

Int_t **GetNargs**( ) const
```
Number of function arguments.
```

Int_t **GetNargsOpt**( ) const
```
Number of function optional (default) arguments.
```

Int_t **Compare**(TObject **obj*)
```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )
```
Return hash value for TFunction based on its name.
```

Long_t **Property**( ) const
```
Get property description word. For meaning of bits see EProperty.
```

void* **InterfaceMethod**( ) const
```
Return pointer to the interface method. Using this pointer we
can find which TFunction belongs to a G__MethodInfo object.
Both need to have the same InterfaceMethod pointer.
```

Author: Fons Rademakers 07/02/97
Last update: 1.03/02 02/09/97 17.44.44 by René Brun

TGaxis

class **TGaxis** : public TLine, public TAttText

public:

class TGaxis	**TGaxis**( )
class TGaxis	**TGaxis**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Float_t *wmin,* Float_t *wmax,* Int_t *ndiv = 510,* Option_t* *chopt,* Float_t *gridlength = 0*)
class TGaxis	**TGaxis**(class TGaxis&)
void	**~TGaxis**( )
virtual void	**AdjustBinSize**(Float_t *A1,* Float_t *A2,* Int_t *nold,* Float_t& *BinLow,* Float_t& *BinHigh,* Int_t& *nbins,* Float_t& *BinWidth*)
class TClass*	**Class**( )
virtual void	**Draw**(Option_t* *chopt*)
virtual void	**DrawAxis**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Float_t *wmin,* Float_t *wmax,* Int_t *ndiv = 510,* Option_t* *chopt,* Float_t *gridlength = 0*)
Float_t	**GetGridLength**( )
Float_t	**GetLabelOffset**( )
Float_t	**GetLabelSize**( )
virtual const Text_t*	**GetName**( )
Int_t	**GetNdiv**( )
Float_t	**GetTickSize**( )
virtual const Text_t*	**GetTitle**( )
Float_t	**GetTitleOffset**( )
Float_t	**GetWmax**( )
Float_t	**GetWmin**( )
virtual class TClass*	**IsA**( )
void	**LabelsLimits**(char* *label,* Int_t& *first,* Int_t& *last*)
virtual void	**Optimize**(Float_t *A1,* Float_t *A2,* Int_t *nold,* Float_t& *BinLow,* Float_t& *BinHigh,* Int_t& *nbins,* Float_t& *BWID*)
virtual void	**Paint**(Option_t* *chopt*)
virtual void	**PaintAxis**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Float_t& *wmin,* Float_t& *wmax,* Int_t& *ndiv,* Option_t* *chopt,* Float_t *gridlength = 0*)
virtual void	**Rotate**(Float_t *X,* Float_t *Y,* Float_t *CFI,* Float_t *SFI,* Float_t *XT,* Float_t *YT,* Float_t& *U,* Float_t& *V*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
void	**SetGridLength**(Float_t *gridlength*)
void	**SetLabelOffset**(Float_t *labeloffset*)
void	**SetLabelSize**(Float_t *labelsize*)
virtual void	**SetName**(Text_t* *name*)
void	**SetOption**(Option_t* *option*)
void	**SetTickSize**(Float_t *ticksize*)
virtual void	**SetTitle**(Text_t* *title*)
void	**SetTitleOffset**(Float_t *titleoffset = 1*)
void	**SetWmax**(Float_t *wmax*)
void	**SetWmin**(Float_t *wmin*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

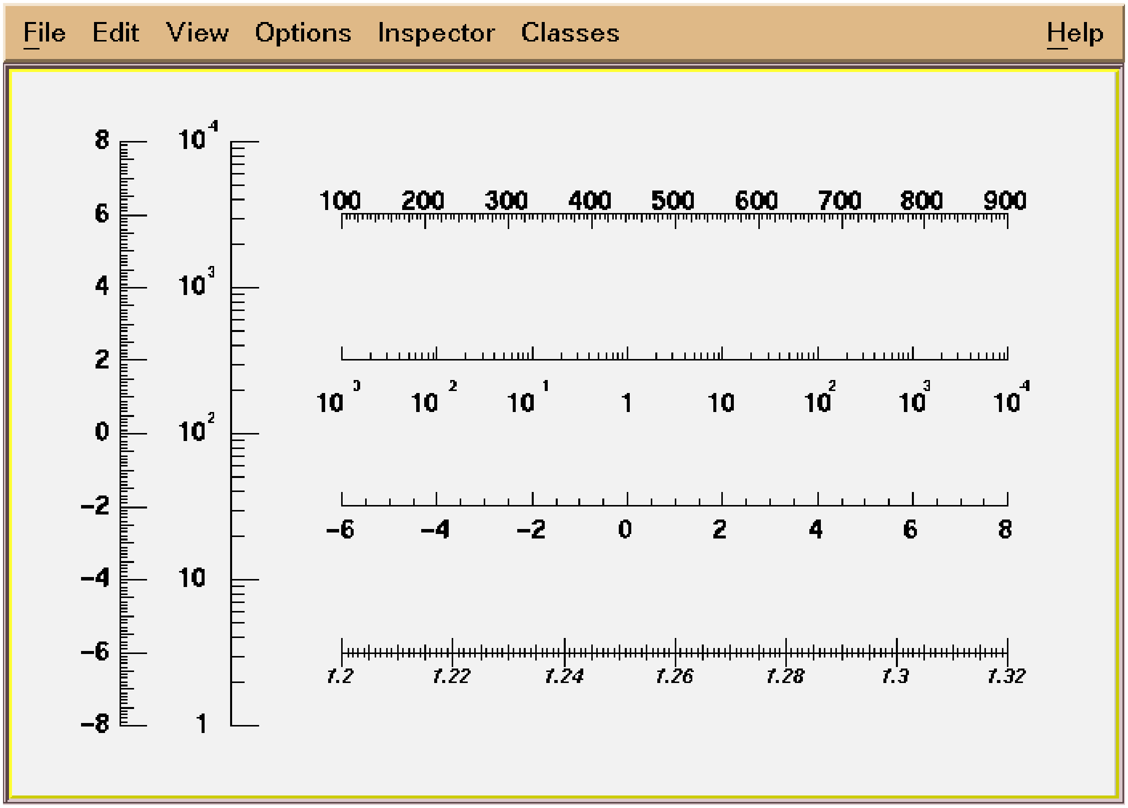
private:

Int_t	**fNdiv**	*Number of divisions*
Float_t	**fWmin**	*Lowest value on the axis*
Float_t	**fWmax**	*Highest value on the axis*

Float_t	**fGridLength**	*Length of the grid in NDC*
Float_t	**fTickSize**	*Size of primary tick mark in NDC*
Float_t	**fLabelOffset**	*Offset of label wrt axis*
Float_t	**fLabelSize**	*Size of labels in NDC*
Float_t	**fTitleOffset**	*Offset of title wrt axis*
char	**fChopt**	*Axis options*
class TString	**fName**	*axis name*
class TString	**fTitle**	*axis title*

Class Description

```
Service class for the graphical representation of axis.
Instances of this class are generated by the histogram classes and TGraph.

The picture below has been generated by the following macro.
{

gROOT.Reset();

c1 = new TCanvas("c1","Examples of Gaxis",10,10,700,500);

c1->Range(-10,-1,10,1);

TGaxis *axis1 = new TGaxis(-4,-0.2,8,-0.2,-6,8,510,"");
axis1->Draw();

TGaxis *axis2 = new TGaxis(-4,0.2,8,0.2,0.001,10000,510,"G");
axis2->Draw();

TGaxis *axis3 = new TGaxis(-8,-0.8,-8,0.8,-8,8,50510,"");
axis3->Draw();

TGaxis *axis4 = new TGaxis(-6,-0.8,-6,0.8,1,10000,50510,"G");
axis4->Draw();

TGaxis *axis5 = new TGaxis(-4,-0.6,8,-0.6,1.2,1.32,80506,"-+");
axis5->SetLabelSize(0.03);
axis5->SetTextFont(72);
axis5->SetLabelOffset(0.025);

axis5->Draw();

TGaxis *axis6 = new TGaxis(-4,0.6,8,0.6,100,900,50510,"-");
axis6->Draw();
}
```



TGaxis( ): **TLine**( ), **TAttText**(*11, 0, 1, 62,* 0*.040*)

    Gaxis default constructor

TGaxis(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,*Float_t wmin, Float_t wmax, Int_t ndiv, Option_t *chopt,Float_t *gridlength*): **TLine**(*xmin, ymin, xmax, ymax*), **TAttText**(*11, 0, 1, 62,* 0.040)

    Gaxis normal constructor
    See explanation of parameters in PaintAxis

~**TGaxis**( )

    Gaxis default destructor

void **Draw**(Option_t **option*)

    Draw this axis with its current attributes

void **DrawAxis**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,*Float_t wmin, Float_t wmax, Int_t ndiv, Option_t *chopt,Float_t *gridlength*)

    Draw this axis with new attributes

void **Paint**(Option_t **option*)

    Draw this axis with its current attributes

void **PaintAxis**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,*Float_t &wmin, Float_t &wmax, Int_t &ndiv, Option_t *chopt,Float_t *gridlength*)

```
Control function to draw an axis
           largely modified and converted to C++ class by René Brun

_Input parameters:

xmin      : X origin coordinate in WC space.
xmax      : X end axis coordinate in WC space.
ymin      : Y origin coordinate in WC space.
ymax      : Y end axis coordinate in WC space.
wmin      : Lowest value for the tick mark
           labels written on the axis.
```

```
wmax      : Highest value for the tick mark labels
             written on the axis.
ndiv      : Number of divisions.

    ndiv=N1 + 100*N2 + 10000*N3
    N1=number of 1st divisions.
    N2=number of 2nd divisions.
    N3=number of 3rd divisions.
        e.g.:
        nndi=0 --> no tick marks.
        nndi=2 --> 2 divisions, one tick mark in the middle
                   of the axis.

chopt :  Options (see below).

    chopt='G': loGarithmic scale, default is linear.
    chopt='B': Blank axis. Useful to superpose axis.

Orientation of tick marks on axis.
----------------------------------

Tick marks are normally drawn on the positive side of the axis,
however, if X0=X1, then negative.

    chopt='+': tick marks are drawn on Positive side. (default)
    chopt='-': tick mark are drawn on the negative side.
    i.e: '+-' --> tick marks are drawn on both sides of the axis.
    chopt='U': Unlabeled axis, default is labeled.

Position of labels on axis.
---------------------------

Labels are normally drawn on side opposite to tick marks.
However:

    chopt='=': on Equal side

Orientation of labels on axis.
------------------------------

Labels are normally drawn parallel to the axis.
However if X0=X1, then Orthogonal
        if Y0=Y1, then Parallel

Position of labels on tick marks.
---------------------------------

Labels are centered on tick marks.
However , if X0=X1, then they are right adjusted.

    chopt='R': labels are Right adjusted on tick mark.
                 (default is centered)
    chopt='L': labels are Left adjusted on tick mark.
    chopt='C': labels are Centered on tick mark.
    chopt='M': In the Middle of the divisions.

Format of labels.
-----------------

Blank characters are stripped, and then the
label is correctly aligned. the dot, if last
character of the string, is also stripped.
```

```
In the following, we have some parameters, like
tick marks length and characters height (in percentage
of the length of the axis (WC))
The default values are as follows:

Primary tick marks: 3.0 %
Secondary tick marks: 1.5 %
Third order tick marks: .75 %
Characters height for labels: 4%

Labels offset: 1.0 %

Optional grid.
--------------

    chopt='W': cross-Wire

Axis bining optimization.
------------------------

By default the axis bining is optimized .

    chopt='N': No bining optimization
    chopt='I': Integer labelling
```

void **Optimize**(Float_t *A1,* Float_t *A2,* Int_t nold, Float_t &BinLow, Float_t &BinHigh, Int_t &nbins, Float_t *&BinWidth*)

```
Reasonable Axis labels optimisation
Get reasonable values for tick marks & ensure they are
not plotted beyond allowed limits

_Input parameters:

A1,A2 : Old WMIN,WMAX .
BinLow,BinHigh : New WMIN,WMAX .
nold   : Old NDIV .
nbins    : New NDIV .
```

void **AdjustBinSize**(Float_t *A1,* Float_t *A2,* Int_t nold, Float_t &BinLow, Float_t &BinHigh, Int_t &nbins, Float_t *&BinWidth*)

```
Axis labels optimisation
This routine adjusts the bining of the axis
in order to have integer values for the labels

_Input parameters:

A1,A2    : Old WMIN,WMAX .
BinLow,BinHigh : New WMIN,WMAX .
nold     : Old NDIV (primary divisions)
nbins    : New NDIV .
```

void **LabelsLimits**(char **label,* Int_t *&first,* Int_t *&last*)

```
Find first and last character of a label
```

void **Rotate**(Float_t *X,* Float_t *Y,* Float_t *CFI,* Float_t SFI, Float_t XT, Float_t YT, Float_t &U, Float_t *&V*)

```
Rotate axis coordinates
```

void **SavePrimitive**(ofstream *&out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetName**(Text_t **name*)

```
Change the name of the axis
```

void **SetOption**(Option_t **option*)

```
To set axis options
```

void **SetTitle**(const Text_t **title*)

```
Change the title of the axis
```

Author: René Brun, Olivier Couet 12/12/94
Last update: 1.03/07 07/11/97 13.00.30 by René Brun

TGeometry

class **TGeometry** : public TNamed

public:

class TGeometry	**TGeometry**( )
class TGeometry	**TGeometry**(Text_t* *name,* Text_t* *title*)
class TGeometry	**TGeometry**(class TGeometry&)
void	**~TGeometry**( )
virtual void	**Browse**(class TBrowser* *b*)
virtual void	**cd**(Text_t* *path = 0*)
class TClass*	**Class**( )
virtual void	**Draw**(Option_t* *option*)
Float_t	**GetBomb**( )
class TNode*	**GetCurrentNode**( )
class THashList*	**GetListOfMaterials**( )
class THashList*	**GetListOfMatrices**( )
class TList*	**GetListOfNodes**( )
class THashList*	**GetListOfShapes**( )
class TMaterial*	**GetMaterial**(Text_t* *name*)
class TMaterial*	**GetMaterialByNumber**(Int_t *number*)
class TNode*	**GetNode**(Text_t* *name*)
class TRotMatrix*	**GetRotMatrix**(Text_t* *name*)
class TRotMatrix*	**GetRotMatrixByNumber**(Int_t *number*)
class TShape*	**GetShape**(Text_t* *name*)
class TShape*	**GetShapeByNumber**(Int_t *number*)
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual void	**ls**(Option_t* *option = rsn2*)
virtual void	**Node**(Text_t* *name,* Text_t* *title,* Text_t* *shapename,* Double_t *x = 0,* Double_t *y = 0,* Double_t *z = 0,* Text_t* *matrixname,* Option_t* *option*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**SetBomb**(Float_t *bomb* = 1.*4*)
virtual void	**SetCurrentNode**(class TNode* *node*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

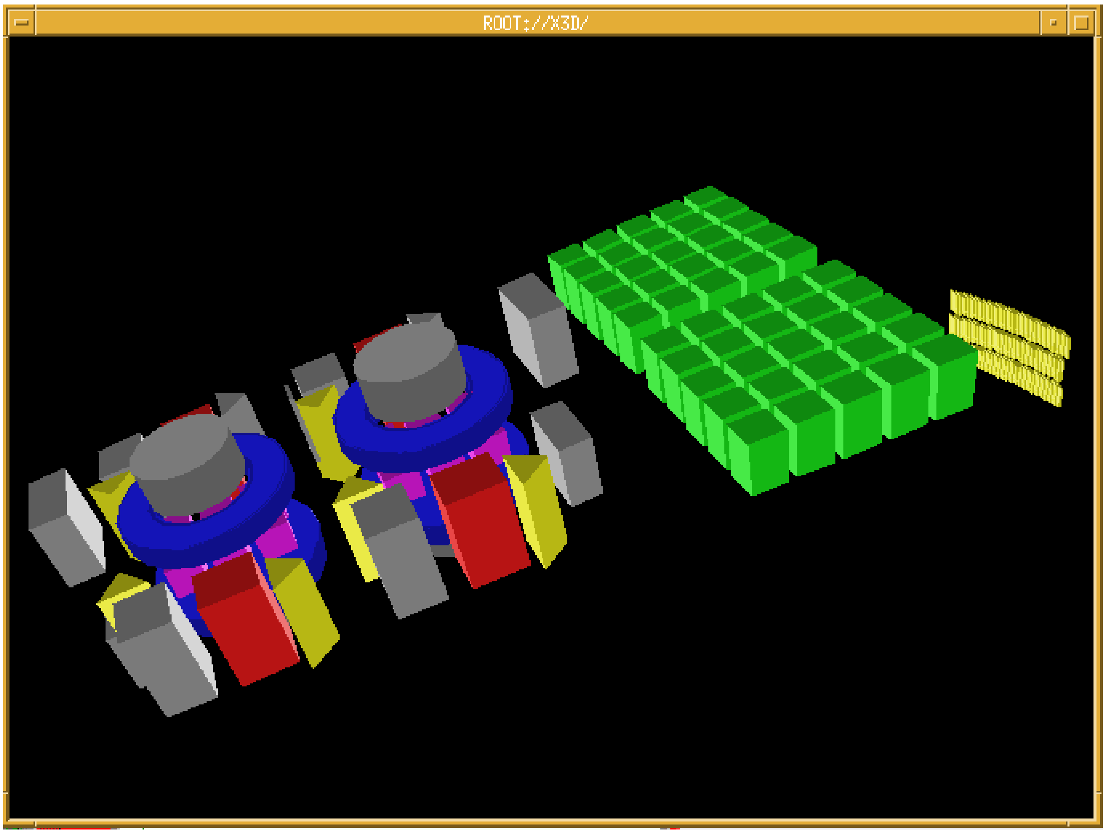
private:

class THashList*	**fMaterials**	*Collection of materials*
class THashList*	**fMatrices**	*Collection of rotation matrices*
class THashList*	**fShapes**	*Collection of shapes*
class TList*	**fNodes**	*Collection of nodes*
class TNode*	**fCurrentNode**	*Pointer to current node*
class TMaterial**	**fMaterialPointer**	*Pointers to materials*
class TRotMatrix**	**fMatrixPointer**	*Pointers to rotation matrices*
class TShape**	**fShapePointer**	*Pointers to shapes*
Float_t	**fBomb**	*Bomb factor for exploded geometry*

Class Description

```
T G E O M E T R Y  description
 The Geometry class describes the geometry of a detector.
 The current implementation supports the GEANT3 style description.
 A special program provided in the ROOT utilities (toroot) can be used
 to automatically translate a GEANT detector geometry into a ROOT geometry.

a Geometry object is entered into the list of geometries into the
```

```
  ROOT main object (see TROOT description) when the TGeometry
  constructor is invoked.
Several geometries may coexist in memory.

A Geometry object consist of the following linked lists:
     - the TMaterial list (material definition only).
     - the TRotmatrix list (Rotation matrices definition only).
     - the TShape list (volume definition only).
     - the TNode list assembling all detector elements.

Only the Build and Draw functions for a geometry are currently supported.

The conversion program from Geant to Root has been added in the list
of utilities in utils directory.(see g2root)
The executable module of g2root can be found in $ROOTSYS/bin/g2root.

To use this conversion program, type the shell command:
     g2root  geant_rzfile macro_name

for example
     g2root na49.geom na49.C
will convert the GEANT RZ file na49.geom into a ROOT macro na49.C

To generate the Geometry structure within Root, do:
 Root > .x na49.C
 Root > na49.Draw()
 Root > wh.x3d()    (this invokes the 3-d Root viewver)
 Root > TFile gna49("na49.root","NEW")  //open a new root file
 Root > na49.Write()                    //Write the na49 geometry structure
 Root > gna49.Write()                   //Write all keys (in this case only
one)
Note: all keys are also written on closing of the file, gna49.Close or
when the program exits, Root closes all open files correctly.
Once this file has been written, in a subsequent session, simply do:
 Root > TFile gna49("na49.root")
 Root > na49.Draw()

The figure below shows the geometry above using the x3d viewer.
This x3d viewver is invoked by selecting "View x3d" in the View menu
of a canvas (See example of this tool bar in TCanvas).
```



TGeometry( )
```
Geometry default constructor
```

TGeometry(Text_t **name,* Text_t **title*) : **TNamed**(*name,* title)
```
Geometry normal constructor
```

~**TGeometry**( )
```
Geometry default destructor
```

void **cd**(const Text_t **path*)
```
Change Current Geometry to this
```

void **Draw**(Option_t **option*)
```
Draw this Geometry
```

TMaterial* **GetMaterial**(Text_t **name*)
```
Return pointer to Material with name
```

TMaterial* **GetMaterialByNumber**(Int_t *number*)
```
Return pointer to Material with number
```

TNode* **GetNode**(Text_t **name*)
```
Return pointer to node with name in the geometry tree
```

TRotMatrix* **GetRotMatrix**(Text_t **name*)
```
Return pointer to RotMatrix with name
```

TRotMatrix* **GetRotMatrixByNumber**(Int_t *number*)
```
Return pointer to RotMatrix with number
```

TShape* **GetShape**(Text_t **name*)
```
Return pointer to Shape with name
```

TShape* **GetShapeByNumber**(Int_t *number*)

```
Return pointer to Shape with number
```

void **ls**(Option_t **option*)

```
List this geometry
```

void **Node**(Text_t **name,* Text_t **title,* Text_t **shapename,* Double_t *x,* Double_t *y,* Double_t *z,* Text_t **matrixname,* Option_t **option*)

```
Add a node to the current node in this geometry
```

void **RecursiveRemove**(TObject **obj*)

```
Recursively remove object from a Geometry list
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 22/09/95
Last update: 1.03/07 29/10/97 13.29.17 by René Brun

TGlobal

class **TGlobal** : public TDictionary

public:

class TGlobal	**TGlobal**(class TGlobal&)
class TGlobal	**TGlobal**(class G__DataMemberInfo* *info = 0*)
void	**~TGlobal**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
void*	**GetAddress**( )
Int_t	**GetArrayDim**( )
const char*	**GetFullTypeName**( )
Int_t	**GetMaxIndex**(Int_t *dim*)
virtual const char*	**GetName**( )
virtual const char*	**GetTitle**( )
const char*	**GetTypeName**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class G__DataMemberInfo* **fInfo** *pointer to CINT data member info*

Class Description

```
Global variables class (global variables are obtained from CINT).
This class describes the attributes of a global variable.
The TROOT class contains a list of all currently defined global
variables (accessible via TROOT::GetListOfGlobals()).
```

TGlobal(G__DataMemberInfo **info*) : **TDictionary**( )

```
Default TGlobal ctor. TGlobals are constructed in TROOT via
a call to TCint::UpdateListOfGlobals().
```

~TGlobal( )

```
TGlobal dtor deletes adopted G__DataMemberInfo object.
```

void* **GetAddress**( ) const

```
Return address of global.
```

Int_t **GetArrayDim**( ) const

```
Return number of array dimensions.
```

Int_t **GetMaxIndex**(Int_t *dim*) const

```
Return maximum index for array dimension "dim".
```

const char* **GetTypeName**( ) const

```
Get type of global variable, e,g.: "class TDirectory*" -> "TDirectory".
Result needs to be used or copied immediately.
```

const char* **GetFullTypeName**( ) const

```
Get full type description of global variable, e,g.: "class TDirectory*".
```

const char* **GetName**( ) const

```
Get global variable name.
```

const char* **GetTitle**( ) const

`Get global variable description string (comment).`

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )

`Return hash value for TGlobal based on its name.`

Long_t **Property**( ) const

`Get property description word. For meaning of bits see EProperty.`

Author: René Brun 13/11/95
Last update: 1.03/02 02/09/97 17.44.44 by René Brun

TGraph

class **TGraph** : public TNamed, public TAttLine, public TAttFill, public TAttMarker

public:

class TGraph	**TGraph**(Int_t *n,* Float_t* *x,* Float_t* *y*)
class TGraph	**TGraph**(class TGraph&)
class TGraph	**TGraph**( )
void	**~TGraph**( )
class TClass*	**Class**( )
void	**ComputeLogs**(Int_t *npoints,* Int_t *opt*)
virtual void	**ComputeRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *xmax,* Float_t& *ymax*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *chopt*)
virtual void	**DrawGraph**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**DrawPanel**( )
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual void	**Fit**(Text_t* *formula,* Option_t* *option,* Option_t* *goption*)
virtual void	**FitPanel**( )
virtual Float_t	**GetErrorX**(Int_t *bin*)
virtual Float_t	**GetErrorY**(Int_t *bin*)
class TH1F*	**GetHistogram**( )
class TList*	**GetListOfFunctions**( )
Int_t	**GetN**( )
virtual void	**GetPoint**(Int_t *i,* Float_t& *x,* Float_t& *y*)
Float_t*	**GetX**( )
class TAxis*	**GetXaxis**( )
Float_t*	**GetY**( )
class TAxis*	**GetYaxis**( )
virtual void	**InitExpo**( )
virtual void	**InitGaus**( )
virtual void	**InitPolynom**( )
virtual class TClass*	**IsA**( )
virtual void	**LeastSquareFit**(Int_t *n,* Int_t *m,* Double_t* *a*)
virtual void	**LeastSquareLinearFit**(Int_t *ndata,* Double_t& *a0,* Double_t& *a1,* Int_t& *ifail*)
virtual void	**Paint**(Option_t* *chopt*)
virtual void	**PaintGraph**(Int_t *npoints,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintGrapHist**(Int_t *npoints,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**Print**(Option_t* *chopt*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetMaximum**(Float_t *maximum = -1111*)
virtual void	**SetMinimum**(Float_t *minimum = -1111*)
virtual void	**SetPoint**(Int_t *i,* Float_t *x,* Float_t *y*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Smooth**(Int_t *npoints,* Float_t* *x,* Float_t* *y,* Int_t *drawtype*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Zero**(Int_t& *k,* Float_t *AZ,* Float_t *BZ,* Float_t *E2,* Float_t& *X,* Float_t& *Y,* Int_t *maxiterations*)

Data Members

protected:

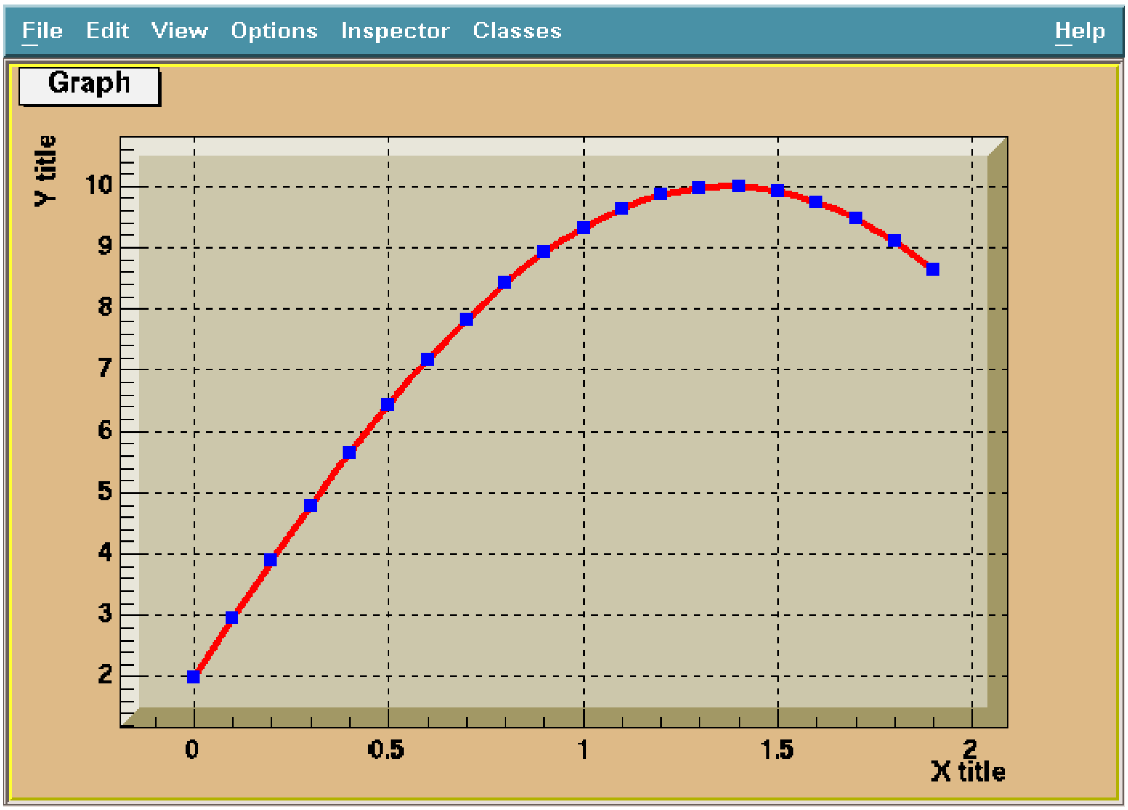
Int_t	**fNpoints**	*Number of points*
Float_t*	**fX**	*array of X points*

Float_t*	**fY**	*array of Y points*
Option_t*	**fOption**	*Axis options*
class TList*	**fFunctions**	*Pointer to list of functions (fits and user)*
class TH1F*	**fHistogram**	*Pointer to histogram used for drawing axis*
Float_t	**fMaximum**	*Maximum value for plotting along y*
Float_t	**fMinimum**	*Minimum value for plotting along y*

See Also: TCutG, TGraphErrors

Class Description

```
A Graph is a graphics object made of two arrays X and Y
with npoints each.
This class supports essentially two graph categories:
  - General case with non equidistant points
  - Special case with equidistant points
The various format options to draw a Graph are explained in
  TGraph::PaintGraph  and TGraph::PaintGrapHist
These two functions are derived from the HIGZ routines IGRAPH and IGHIST
and many modifications.

The picture below has been generated by the following macro:
------------------------------------------------------------------
{
TCanvas *c1 = new TCanvas("c1","A Simple Graph Example",200,10,700,500);
Float_t x[100], y[100];
Int_t n = 20;
for (Int_t i=0;i<n;i++) {
  x[i] = i*0.1;
  y[i] = 10*sin(x[i]+0.2);
}
gr = new TGraph(n,x,y);
gr->Draw("AC*");
}
```



TGraph( ): **TNamed**( ), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
Graph default constructor
```

TGraph(Int_t *n*, Float_t **x*, Float_t **y*): **TNamed**("Graph", "Graph"), **TAttLine**( ), **TAttFill**( ), **TAttMarker**( )

```
Graph normal constructor
```

~**TGraph**( )

```
Graph default destructor
```

void **ComputeRange**(Float_t &*xmin*, Float_t &*ymin*, Float_t &*xmax*, Float_t &*ymax*)

```
this function is dummy in TGraph, but redefined by TGraphErrors
```

void **Draw**(Option_t **option*)

```
Draw this graph with its current attributes
Options to draw a graph are described in TGraph::PaintGraph
```

Int_t **DistancetoPrimitive**(Int_t *px*, Int_t *py*)

```
Compute distance from point px,py to a graph
Compute the closest distance of approach from point px,py to this line.
The distance is computed in pixels units.
```

void **DrawGraph**(Int_t *n*, Float_t **x*, Float_t **y*, Option_t **option*)

```
Draw this graph with new attributes
```

void **DrawPanel**( )

```
Display a panel with all graph drawing options
```

void **ExecuteEvent**(Int_t *event*, Int_t *px*, Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a graph is clicked with the locator

If Left button clicked on one of the line end points, this point
  follows the cursor until button is released.
```

```
if Middle button clicked, the line is moved parallel to itself
  until the button is released.
```

void **Fit**(Text_t **fname,* Option_t **option,* Option_t **goption*)

```
Fit this graph with function fname
fname is the name of an already predefined function created by TF1
Predefined functions such as Gaus, Expo and Poln are automatically
created by ROOT.

The list of fit options is given in parameter option.
   option = "W"  Set all errors to 1
          = "Q" Quiet mode (minimum printing)
          = "V" Verbose mode (default is between Q and V)
          = "B" Some parameters are bound
          = "R" Use the Range specified in the function range
          = "N" Do not store the graphics function, do not draw
          = "0" Do not plot the result of the fit. By default the fitted func-
tion
                is drawn unless the option"N" above is specified.
          = "+" Add this new fitted function to the list of fitted functions
                (by default, any previous function is deleted)

When the fit is drawn (by default), the parameter goption may be used
to specify a list of graphics options. See TGraph::Paint for a complete
list of these options.

In order to use the Range option, one must first create a function
with the expression to be fitted. For example, if your graph
has a defined range between -4 and 4 and you want to fit a gaussian
only in the interval 1 to 3, you can do:
     TF1 *f1 = new TF1("f1","gaus",1,3);
     graph->Fit("f1","R");
```

void **FitPanel**( )

```
Display a panel with all graph fit options
See class TFitPanel for example
```

Float_t **GetErrorX**(Int_t *i*)

```
This function is called by GraphFitChisquare.
It always returns a negative value. Real implementation in TGraphErrors
```

Float_t **GetErrorY**(Int_t *i*)

```
This function is called by GraphFitChisquare.
It always returns a negative value. Real implementation in TGraphErrors
```

void **GetPoint**(Int_t *i,* Float_t &*x,* Float_t &*y*)

```
Get x and y values for point number i
```

TAxis* **GetXaxis**( )

```
Get x axis of the graph.
```

TAxis* **GetYaxis**( )

```
Get y axis of the graph.
```

void **InitGaus**( )

```
Compute Initial values of parameters for a gaussian
```

void **InitExpo**( )

```
Compute Initial values of parameters for an exponential
```

void **InitPolynom**( )

```
Compute Initial values of parameters for a polynom
```

void **LeastSquareFit**(Int_t *n,* Int_t *m,* Double_t **a*)

```
Least squares lpolynomial fitting without weights
n   number of points to fit
m   number of parameters
a   array of parameters

based on CERNLIB routine LSQ: Translated to C++ by René Brun
(E.Keil.  revised by B.Schorr, 23.10.1981.)
```

void **LeastSquareLinearFit**(Int_t *ndata,* Double_t &*a0,* Double_t &*a1,* Int_t &*ifail*)

```
Least square linear fit without weights
extracted from CERNLIB LLSQ: Translated to C++ by René Brun
(added to LSQ by B. Schorr, 15.02.1982.)
```

void **Paint**(Option_t **option*)

```
Draw this graph with its current attributes
```

void **PaintGraph**(Int_t *npoints,* Float_t **x,* Float_t **y,* Option_t **chopt*)

```
Control function to draw a graph
Draws one dimensional graphs. The aspect of the graph is done
according to the value of the chopt.

_Input parameters:

npoints : Number of points in X or in Y.
X(N) or X(2) : X coordinates or (XMIN,XMAX) (WC space).
Y(N) or Y(2) : Y coordinates or (YMIN,YMAX) (WC space).
chopt : Option.

chopt='L' :  A simple polyline beetwen every points is drawn

chopt='F' :  A fill area is drawn ('CF' draw a smooth fill area)

chopt='A' :  Axis are drawn around the graph

chopt='C' :  A smooth Curve is drawn

chopt='*' :  A Star is plotted at each point

chopt='P' :  Idem with the current marker

chopt='B' :  A Bar chart is drawn at each point

chopt='1' :  ylow=rwymin
```

void **PaintGrapHist**(Int_t *npoints,* Float_t **x,* Float_t **y,* Option_t **chopt*)

```
Control function to draw a graphistogram
Draws one dimensional graphs. The aspect of the graph is done
according to the value of the chopt.

_Input parameters:

npoints : Number of points in X or in Y.
X(N) or x[1] : X coordinates or (XMIN,XMAX) (WC space).
Y(N) or y[1] : Y coordinates or (YMIN,YMAX) (WC space).
chopt : Option.

chopt='R' :  Graph is drawn horizontaly, parallel to X axis.
            (default is vertically, parallel to Y axis)
            If option R is selected the user must give:
              2 values for Y (y[0]=YMIN and y[1]=YMAX)
              N values for X, one for each channel.
            Otherwise the user must give:
```

```
               N values for Y, one for each channel.
               2 values for X (x[0]=XMIN and x[1]=XMAX)

chopt='L' :  A simple polyline beetwen every points is drawn

chopt='H' :  An Histogram with equidistant bins is drawn
            as a polyline.

chopt='F' :  An histogram with equidistant bins is drawn
            as a fill area. Contour is not drawn unless
            chopt='H' is also selected..

chopt='N' :  Non equidistant bins (default is equidistant)
            If N is the number of channels array X and Y
            must be dimensionned as follow:
            If option R is not selected (default) then
            the user must give:
              (N+1) values for X (limits of channels).
               N values for Y, one for each channel.
            Otherwise the user must give:
              (N+1) values for Y (limits of channels).
               N values for X, one for each channel.

chopt='F1':  Idem as 'F' except that fill area is no more
            reparted arround axis X=0 or Y=0 .

chopt='C' :  A smooth Curve is drawn.

chopt='*' :  A Star is plotted at the center of each bin.

chopt='P' :  Idem with the current marker

chopt='B' :  A Bar chart with equidistant bins is drawn as fill
            areas (Contours are drawn).
```

void **ComputeLogs**(Int_t *npoints,* Int_t *opt*)

```
Convert WC from Log scales
Take the LOG10 of xwork and ywork according to the value of Options
and put it in xworkl and yworkl.

npoints : Number of points in xwork and in ywork.
```

void **Print**(Option_t **option*)

```
Print graph values
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetPoint**(Int_t *i,* Float_t *x,* Float_t *y*)

```
Set x and y values for point number i
```

void **Smooth**(Int_t *npoints,* Float_t **x,* Float_t **y,* Int_t *drawtype*)

```
Smooth a curve given by N points
Underlaying routine for Draw based on the CERN GD3 routine TVIPTE

  Author - Marlow etc.   Modified by - P. Ward     Date -  3.10.1973

This routine draws a smooth tangentially continuous curve through
the sequence of data points P(I) I=1,N where P(I)=(X(I),Y(I))
the curve is approximated by a polygonal arc of short vectors .
the data points can represent open curves, P(1) != P(N) or closed
curves P(2) == P(N) . If a tangential discontinuity at P(I) is
required , then set P(I)=P(I+1) . loops are also allowed .
```

```
Reference Marlow and Powell,Harwell report No.R.7092.1972
MCCONALOGUE,Computer Journal VOL.13,NO4,NOV1970PP392 6

_Input parameters:

npoints   : Number of data points.
x         : Abscissa
y         : Ordinate

delta is the accuracy required in constructing the curve.
if it is zero then the routine calculates a value other-
wise it uses this value. (default is 0.0)
```

void **Streamer**(TBuffer &*b*)

```
Stream an object of class TGraph.
```

void **Zero**(Int_t &*k*, Float_t *AZ*, Float_t *BZ*, Float_t *E2*, Float_t &*X*, Float_t &Y, Int_t *maxiterations*)

```
Find zero of a continuous function
Underlaying routine for PaintGraph
This function finds a real zero of the continuous real
function Y(X) in a given interval (A,B). See accompanying
notes for details of the argument list and calling sequence
```

Author: René Brun, Olivier Couet 12/12/94
Last update: 1.03/07 08/11/97 10.12.41 by René Brun

TGraphErrors

class **TGraphErrors** : public TGraph

public:

class TGraphErrors	**TGraphErrors**(Int_t *n,* Float_t* *x,* Float_t* *y,* Float_t* *ex,* Float_t* *ey*)
class TGraphErrors	**TGraphErrors**( )
class TGraphErrors	**TGraphErrors**(class TGraphErrors&)
void	**~TGraphErrors**( )
class TClass*	**Class**( )
virtual void	**ComputeRange**(Float_t& *xmin,* Float_t& *ymin,* Float_t& *xmax,* Float_t& *ymax*)
Float_t	**GetErrorX**(Int_t *bin*)
Float_t	**GetErrorY**(Int_t *bin*)
Float_t*	**GetEX**( )
Float_t*	**GetEY**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *chopt*)
virtual void	**Print**(Option_t* *chopt*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetPointError**(Int_t *i,* Float_t *ex,* Float_t *ey*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

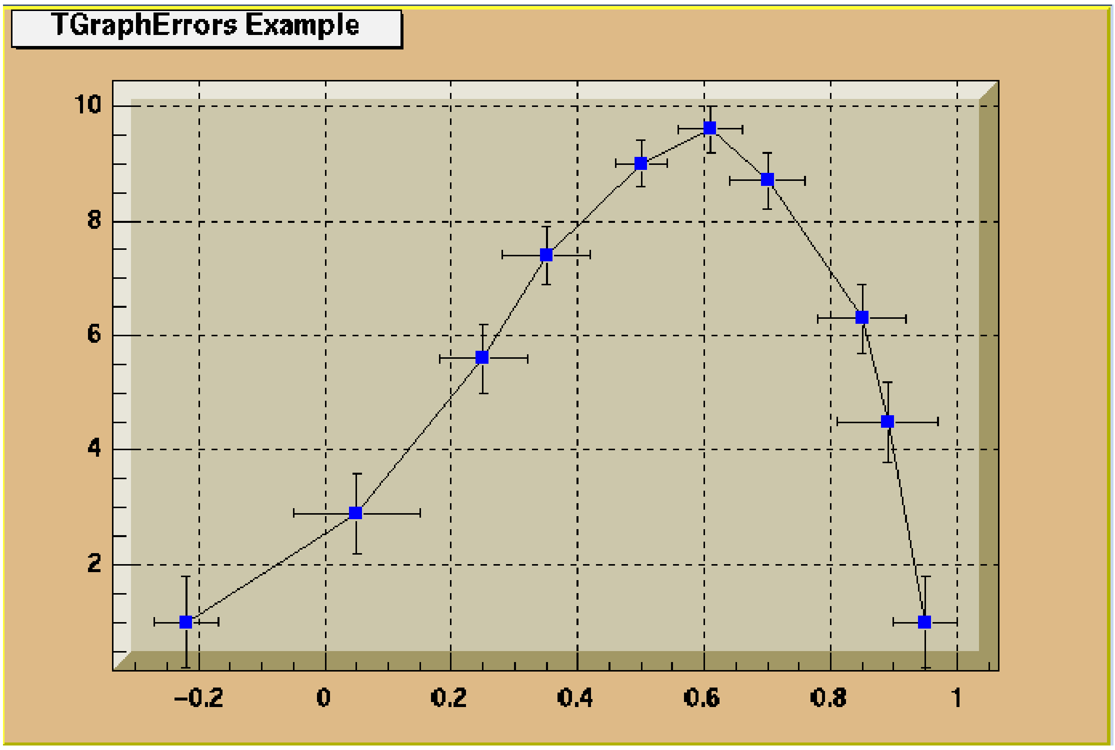
Data Members

protected:

Float_t*	**fEX**	*array of X errors*
Float_t*	**fEY**	*array of Y errors*

Class Description

```
A TGraphErrors is a TGraph with error bars.
The various format options to draw a TGraphErrors are explained in
  TGraphErrors::Paint.

The picture below has been generated by the following macro:
------------------------------------------------------------------------
{
gROOT->Reset();
c1 = new TCanvas("c1","A Simple Graph with error bars",200,10,700,500);

c1->SetFillColor(42);
c1->SetGrid();
c1->GetFrame()->SetFillColor(21);
c1->GetFrame()->SetBorderSize(12);

Int_t n = 10;
Float_t x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5, 0.61,0.7,0.85,0.89,0.95};
Float_t y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
gr = new TGraphErrors(n,x,y,ex,ey);
gr->SetTitle("TGraphErrors Example");
gr->SetMarkerColor(4);
gr->SetMarkerStyle(21);
gr->Draw("ALP");

c1->Update();
}
```



TGraphErrors( ): **TGraph**( )
```
TGraphErrors default constructor
```

TGraphErrors(Int_t *n,* Float_t **x,* Float_t **y,* Float_t **ex,* Float_t **ey*): **TGraph**(*n, x, y*)
```
TGraphErrors normal constructor
```

~**TGraphErrors**( )
```
TGraphErrors default destructor
```

Float_t **GetErrorX**(Int_t *i*)
```
 This function is called by GraphFitChisquare.
 It returns the error along X at point i.
```

Float_t **GetErrorY**(Int_t *i*)
```
 This function is called by GraphFitChisquare.
 It returns the error along Y at point i.
```

void **Paint**(Option_t **option*)
```
Draw this TGraphErrors with its current attributes
```

void **Print**(Option_t **option*)
```
Print graph and errors values
```

void **SavePrimitive**(ofstream *&out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

void **SetPointError**(Int_t *i,* Float_t *ex,* Float_t *ey*)
```
Set ex and ey values for point number i
```

void **Streamer**(TBuffer *&b*)
```
Stream an object of class TGraphErrors.
```

Author: René Brun 15/09/96
Last update: 1.03/05 14/10/97 17.45.32 by René Brun

TGroupButton

class **TGroupButton** : public TButton

public:

class TGroupButton	**TGroupButton**(Text_t* *groupname,* Text_t* *title,* Text_t* *method,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TGroupButton	**TGroupButton**( )
class TGroupButton	**TGroupButton**(class TGroupButton& *button*)
void	**~TGroupButton**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *button*)
virtual void	**DisplayColorTable**(Text_t* *action,* Float_t *x0,* Float_t *y0,* Float_t *wc,* Float_t *hc*)
virtual void	**ExecuteAction**( )
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
A TGroupButton object is a specialized TButton used in a group of Buttons.
When a button from a group of TGroupButtons is selected, all other buttons
from the group with the same name are disabled.

For examples of use of TGroupButton objects, see:
 TAttFillCanvas, TAttLineCanvas, TAttTextCanvas and TAttMarkerCanvas.
```

TGroupButton( ): **TButton**( )

```
GroupButton default constructor
```

TGroupButton(const Text_t **groupname,* const Text_t **title,* const Text_t **method,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TButton**(*title, method, x1, y1, x2, y2*)

```
GroupButton normal constructor
```

~TGroupButton( )

```
GroupButton default destructor
```

void **Copy**(TObject &*button*)

```
Copy this button to button
```

void **DisplayColorTable**(const Text_t **action,* Float_t *x0,* Float_t *y0,* Float_t *wc,* Float_t *hc*)

```
Display Color Table in an attribute canvas
```

void **ExecuteAction**( )

```
Execute action of this button
If an object has been selected before executing the APPLY button
in the control canvas, The member function and its parameters
for this object is executed via the interpreter.
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a Button object is clicked.
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 01/07/96
Last update: 1.03/01 17/10/97 17.48.11 by Piotr Golonka

TGTRA

class **TGTRA** : public TBRIK

public:

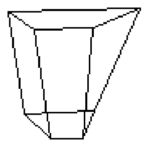
class TGTRA	**TGTRA**( )
class TGTRA	**TGTRA**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dz,* Float_t *theta,* Float_t *phi,* Float_t *twist,* Float_t *h1,* Float_t *bl1,* Float_t *tl1,* Float_t *alpha1,* Float_t *h2,* Float_t *bl2,* Float_t *tl2,* Float_t *alpha2*)
class TGTRA	**TGTRA**(class TGTRA&)
void	**~TGTRA**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fTwist**	*twisting parameter*
Float_t	**fH1**	*half length in y at low z*
Float_t	**fBl1**	*half length in x at low z and y low edge*
Float_t	**fTl1**	*half length in x at low z and y high edge*
Float_t	**fAlpha1**	*angle w.r.t. the y axis*
Float_t	**fH2**	*half length in y at high z*
Float_t	**fBl2**	*half length in x at high z and y low edge*
Float_t	**fTl2**	*half length in x at high z and y high edge*
Float_t	**fAlpha2**	*angle w.r.t. the y axis*

Class Description



```
GTRA is a general twisted trapezoid. The faces perpendicular to z are trapezia
and their centres are not necessarily on a line parallel to the z axis as the
TRAP; additionally, the faces may be twisted so that none of their edges are
parallel. It is a TRAP shape, except that it is twisted in the x-y plane as a
function of z. The parallel sides perpendicular to the z axis are rotated with
respect to the x axis by an angle TWIST, which is one of the parameters. The
shape is defined by the eight corners and is assumed to be constructed of
straight lines joining points on the boundary of the trapezoidal face at z=-DZ
to the corresponding points on the face at z=DZ. Divisions are not allowed.
It has 15 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dZ         half-length along the z axis
  - theta      polar angle of the line joining the centre of the face
               at -DZ to the centre of the one at +DZ
  - phi        azimuthal angle of the line joining the centre of
               the face at -DZ to the centre of the one at +DZ
  - twist      twist angle of the faces parallel to the x-y plane
               at z = +/- DZ around an axis parallel to z passing
```

```
                 through their centre
- h1             half-length along y of the face at -DZ
- bl1            half-length along x of the side at -H1 in y of
                 the face at -DZ in z
- tl1            half-length along x of the side at +H1 in y of the face
                 at -DZ in z
- alpha1         angle with respect to the y axis from the centre of
                 the side at -H1 in y to the centre of the side at
                 +H1 in y of the face at -DZ in z
- h2             half-length along y of the face at +DZ
- bL2            half-length along x of the side at -H2 in y of the face at
                 +DZ in z

- tl2            half-length along x of the side at +H2 in y of the face
                 at +DZ in z

- alpha2         angle with respect to the y axis from the centre of the side
                 at -H2 in y to the centre of the side at +H2 in y of the
                 face at +DZ in z
```

TGTRA( )

```
GTRA shape default constructor
```

TGTRA(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dz,* Float_t *theta,*Float_t phi, Float_t twist, Float_t h1, Float_t bl1, Float_t tl1, Float_t alpha1,Float_t h2, Float_t bl2, Float_t tl2, Float_t *alpha2*): **TBRIK**(*name, title, material, theta, phi, dz*)

```
GTRA shape normal constructor
```

~TGTRA( )

```
GTRA shape default destructor
```

void **SetPoints**(Float_t **buff*)

```
Create GTRA points
```

Author: Nenad Buncic 19/09/95
Last update: 1.00/07 27/03/97 17.53.27 by René Brun

TGuiFactory

class **TGuiFactory** : public TNamed

public:

class TGuiFactory	**TGuiFactory**(char* *name* = *Batch,* char* *title* = Batch GUI *Factory*)
class TGuiFactory	**TGuiFactory**(class TGuiFactory&)
void	**~TGuiFactory**( )
class TClass*	**Class**( )
virtual class TApplicationImp*	**CreateApplicationImp**(char* *classname,* int* *argc,* char** *argv,* void* *option,* Int_t *numOptions*)
virtual class TBrowserImp*	**CreateBrowserImp**(class TBrowser* *b,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
virtual class TBrowserImp*	**CreateBrowserImp**(class TBrowser* *b,* char* *title,* UInt_t *width,* UInt_t *height*)
virtual class TCanvasImp*	**CreateCanvasImp**(class TCanvas* *c,* char* *title,* UInt_t *width,* UInt_t *height*)
virtual class TCanvasImp*	**CreateCanvasImp**(class TCanvas* *c,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
virtual class TContextMenuImp*	**CreateContextMenuImp**(class TContextMenu* *c,* char* *name,* char* *title*)
virtual class TControlBarImp*	**CreateControlBarImp**(class TControlBar* *c,* char* *name,* char* *title*)
virtual class TIconImp*	**CreateIconImp**(class TIcon* *i*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TMotifGuiFactory

Class Description

```
TGuiFactory

This ABC is a factory for GUI components. Depending on which
factory is active one gets either Motif, Windows or Mac components.
In case there is no platform dependent implementation on can run in
batch mode directly using an instance of this base class.
```

TGuiFactory(const char **name,* const char **title*): **TNamed**(*name, title*)

```
TGuiFactory ctor only called by derived classes.
```

TApplicationImp* **CreateApplicationImp**(const char **classname,* int **argc,* char ***argv,* void **options,* Int_t *numOptions*)

```
Create a batch version of TApplicationImp
```

TCanvasImp* **CreateCanvasImp**(TCanvas **c,* const char **title,* UInt_t *width,* UInt_t *height*)

```
Create a batch version of TCanvasImp
```

TCanvasImp* **CreateCanvasImp**(TCanvas **c,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)

```
Create a batch version of TCanvasImp
```

TBrowserImp* **CreateBrowserImp**(TBrowser **b,* const char **title,* UInt_t *width,* UInt_t *height*)

```
Create a batch version of TBrowserImp
```

TBrowserImp* **CreateBrowserImp**(TBrowser **b,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)

```
Create a batch version of TBrowserImp
```

TContextMenuImp* **CreateContextMenuImp**(TContextMenu **c,* const char **name,* const char **title*)

```
Create a batch version of TContextMenuImp
```

TControlBarImp* **CreateControlBarImp**(TControlBar **c,* const char **name,* const char **title*)

```
Create a batch version of TControlBarImp
```

TIconImp* **CreateIconImp**(TIcon **i*)

```
Create a version of TIconImp
```

Author: Fons Rademakers 15/11/95
Last update: 1.01/05 06/06/97 16.30.36 by Fons Rademakers

TGX11

class **TGX11** : public TGXW

private:

- void **ClearPixmap**(Drawable* *pix*)
- void **CloseWindow1**( )
- void **CopyWindowtoPixmap**(Drawable* *pix,* int *xpos,* int *ypos*)
- void **GetWindowID**(int *wkid,* Window& *idg*)
- void **PutImage**(int *offset,* int *itran,* int *x0,* int *y0,* int *nx,* int *ny,* int *xmin,* int *ymin,* int *xmax,* int *ymax,* unsigned char* *image*)
- void **RemovePixmap**(Drawable* *pix*)
- void **RestoreAddress**( )
- void **SaveAddress**( )
- void **SetColor**(GC *gc,* int *ci*)
- void **SetFillStyleIndex**(Int_t *style,* Int_t *fasi*)
- void **SetInput**(int *inp*)
- void **SetMarkerType**(int *type,* int *n,* XPoint* *xy*)

public:

- class TGX11 **TGX11**( )
- class TGX11 **TGX11**(Text_t* *name,* Text_t* *title*)
- class TGX11 **TGX11**(class TGX11&)
- void **~TGX11**( )
- class TClass* **Class**( )
- void **ClearWindow**( )
- void **ClosePixmap**( )
- void **CloseWindow**( )
- void **CopyPixmap**(int *wid,* int *xpos,* int *ypos*)
- const char* **DisplayName**( )
- void **DrawBox**(int *x1,* int *y1,* int *x2,* int *y2,* enum TGXW::EBoxMode *mode*)
- void **DrawCellArray**(int *x1,* int *y1,* int *x2,* int *y2,* int *nx,* int *ny,* int* *ic*)
- void **DrawFillArea**(int n, (*unknown*) TPoint* xy)
- void **DrawLine**(int *x1,* int *y1,* int *x2,* int *y2*)
- void **DrawPolyLine**(int n, (*unknown*) TPoint* xy)
- void **DrawPolyMarker**(int n, (*unknown*) TPoint* xy)
- void **DrawText**(int *x,* int *y,* float *angle,* float *mgn,* char* *text,* enum TGXW::ETextMode *mode*)
- void **GetCharacterUp**(Float_t& *chupx,* Float_t& *chupy*)
- Int_t **GetDoubleBuffer**(int *wid*)
- void **GetGeometry**(int *wid,* int& *x,* int& *y,* unsigned int& *w,* unsigned int& *h*)
- void **GetPlanes**(int& *nplanes*)
- void **GetRGB**(int *index,* float& *r,* float& *g,* float& *b*)
- void **GetTextExtent**(int& *w,* int& *h,* char* *mess*)
- Float_t **GetTextMagnitude**( )
- Bool_t **Init**(void* *display*)
- Int_t **InitWindow**(ULong_t *window*)
- virtual class TClass* **IsA**( )
- void **MoveWindow**(int *wid,* int *x,* int *y*)
- Int_t **OpenDisplay**(Display* *display*)
- Int_t **OpenPixmap**(int *w,* int *h*)
- void **QueryPointer**(int& *ix,* int& *iy*)
- void **ReadGIF**(int *x0,* int *y0,* char* *file*)
- Int_t **RequestLocator**(int *mode,* int *ctyp,* int& *x,* int& *y*)
- Int_t **RequestString**(int *x,* int *y,* char* *text*)
- void **RescaleWindow**(int *wid,* int *w,* int *h*)
- Int_t **ResizePixmap**(int *wid,* int *w,* int *h*)
- void **ResizeWindow**(int *wid*)

void	**SelectWindow**(int *wid*)
void	**SetCharacterUp**(Float_t *chupx,* Float_t *chupy*)
void	**SetClipOFF**(int *wid*)
void	**SetClipRegion**(int *wid,* int *x,* int *y,* int *w,* int *h*)
void	**SetCursor**(int win, (*unknown*) ECursor cursor)
void	**SetDoubleBuffer**(int *wid,* int *mode*)
void	**SetDoubleBufferOFF**( )
void	**SetDoubleBufferON**( )
void	**SetDrawMode**(enum TGXW::EDrawMode *mode*)
void	**SetFillColor**(Color_t *cindex*)
void	**SetFillStyle**(Style_t *style*)
void	**SetLineColor**(Color_t *cindex*)
void	**SetLineStyle**(Style_t *linestyle*)
void	**SetLineType**(int *n,* int* *dash*)
void	**SetLineWidth**(Width_t *width*)
void	**SetMarkerColor**(Color_t *cindex*)
void	**SetMarkerSize**(Float_t *markersize*)
void	**SetMarkerStyle**(Style_t *markerstyle*)
void	**SetRGB**(int *cindex,* float *r,* float *g,* float *b*)
void	**SetTextAlign**(Short_t *talign = 11*)
void	**SetTextColor**(Color_t *cindex*)
void	**SetTextFont**(Int_t *fontnumber*)
Int_t	**SetTextFont**(char* *fontname,* enum TGXW::ETextSetMode *mode*)
void	**SetTextMagnitude**(Float_t *mgn = 1*)
void	**SetTextSize**(Float_t *textsize*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Sync**(int *mode*)
void	**UpdateWindow**(int *mode*)
void	**Warp**(int *ix,* int *iy*)
void	**WriteGIF**(char* *name*)
void	**WritePixmap**(int *wid,* int *w,* int *h,* char* *pxname*)

Data Members

private:

Bool_t	**fDisplayOpened**	*X11 display opened*
Int_t	**fMaxNumberOfWindows**	*Maximum number of windows*
Window_t*	**fWindows**	*List of windows*
Cursor	**fCursors**	*List of cursors*
Int_t	**fTextAlignH**	*Text Alignment Horizontal*
Int_t	**fTextAlignV**	*Text Alignment Vertical*
Float_t	**fCharacterUpX**	*Character Up vector along X*
Float_t	**fCharacterUpY**	*Character Up vector along Y*
Float_t	**fTextMagnitude**	*Text Magnitude*

Class Description

```
The  G  X  1 1  class
Basic interface to the X11 graphics system

This code was initially developped in the context of HIGZ and PAW
by Olivier Couet. (package X11INT)
It has been converted to a C++ class by René Brun.
```

TGX11( )

```
Default Constructor
```

TGX11(Text_t **name,* Text_t **title*) : **TGXW**(*name,* title)

```
Normal Constructor
```

~TGX11( )
```
Default Destructor
```

Bool_t **Init**(void **display*)
```
Initialize X11 system. Returns kFALSE in case of failure.
```

void **ClearPixmap**(Drawable **pix*)
```
Clear the pixmap PIX
pix : Pixmap address
```

void **ClearWindow**( )
```
Clear current window
```

void **ClosePixmap**( )
```
Close the current opened pixmap
```

void **CloseWindow**( )
```
Delete current window
```

void **CloseWindow1**( )
```
Delete current window
```

void **CopyPixmap**(int *wid,* int *xpos,* int *ypos*)
```
Copy the pixmap PIX at the position XPOS YPOS
wid        : Pixmap identifier
xpos,ypos  : Pixmap Position

Window root;
int xx, yy;
unsigned int ww, hh, border, depth;
```

void **CopyWindowtoPixmap**(Drawable **pix,* int *xpos,* int *ypos*)
```
Copy area in current window in the pixmap PIX
pix         : Pixmap address
xpos, ypos  : Position in the current window
The area copied has the size of the pixmap PIX.
```

void **DrawBox**(int *x1,* int *y1,* int *x2,* int *y2,* EBoxMode *mode*)
```
Draw a box
x1,y1        : left down corner
x2,y2        : right up corner
mode         : drawing mode
 mode=0 hollow  (kHollow)
 mode=1 solid   (kSolid)
```

void **DrawCellArray**(int *x1,* int *y1,* int *x2,* int *y2,* int *nx,* int *ny,* int **ic*)
```
Draw a cell array
x1,y1        : left down corner
x2,y2        : right up corner
nx,ny        : array size
ic           : array

Draw a cell array. The drawing is done with the pixel presicion
if (X2-X1)/NX (or Y) is not a exact pixel number the position of
the top rigth corner may be wrong.
```

void **DrawFillArea**(int *n,* TPoint **xyt*)
```
Fill area described by polygon
n         : number of points
xy(2,n)   : list of points
```

void **DrawLine**(int *x1,* int *y1,* int *x2,* int *y2*)
```
Draw a line
```

```
x1,y1        : begin of line
x2,y2        : end of line
```

void **DrawPolyLine**(int *n,* TPoint **xyt*)

```
Draw a line through all points
n         : number of points
xy        : list of points
```

void **DrawPolyMarker**(int *n,* TPoint **xyt*)

```
Draw n markers with the current attributes at position x, y
n    : number of markers to draw
xy   : x,y coordinates of markers
```

void **DrawText**(int *x,* int *y,* float *angle,* float *mgn,*const char *text, ETextMode *mode*)

```
Draw a text string using current font
mode       : drawing mode
mode=0     : the background is not drawn (kClear)
mode=1     : the background is drawn (kSolid)
x,y        : text position
angle      : text angle
mgn        : magnification factor
text       : text string
```

void **GetCharacterUp**(Float_t &*chupx,* Float_t &*chupy*)

```
Return character up vectors
```

Int_t **GetDoubleBuffer**(int *wid*)

```
Query the double buffer value for the window WID
wid  : Window identifier.
```

void **GetGeometry**(int *wid,* int &*x,* int &*y,* unsigned int &*w,* unsigned int &*h*)

```
Return position and size of window wid
wid        : window identifier
x,y        : window position (output)
w,h        : window size (output)
  if wid < 0 the size of the display is returned
```

const char* **DisplayName**( )

```
Return hostname on which the display is opened.
```

void **GetPlanes**(int &*nplanes*)

```
Get maximum number of planes
nplanes     : number of bit planes
```

void **GetTextExtent**(int &*w,* int &*h,* char **mess*)

```
Return the size of a character string
iw          : text width
ih          : text height
mess        : message
```

void **GetWindowID**(int *wkid,* Window &*idg*)

```
Return the X11 window identifier
wkid     : Workstation identifier (input)
idg      : Window identifier (output)
   Return the X11 window identifier according to the Window id.
```

void **MoveWindow**(int *wid,* int *x,* int *y*)

```
Move the window WID
wid  : Window identifier.
x    : x new window position
y    : y new window position
```

Int_t **OpenDisplay**(Display **disp*)

```
Open the display. Return -1 if the opening fails
host        : host name
```

Int_t **OpenPixmap**(int *w*, int *h*)

```
Open a new pixmap
w,h : Width and height of the pixmap.
```

Int_t **InitWindow**(ULong_t *win*)

```
Open window and returns window number
Init window and return window number,
return -1 if window initialization fails.
```

void **QueryPointer**(int &*ix*, int &*iy*)

```
Query pointer position
ix       : X coordinate of pointer
iy       : Y coordinate of pointer
 (both coordinates are relative to the origin of the root window)
```

void **RemovePixmap**(Drawable **pix*)

```
Remove the pixmap PIX
pix         : Pixmap address
```

Int_t **RequestLocator**(int *mode*, int *ctyp*, int &*x*, int &*y*)

```
Request Locatorr position
x,y       : cursor position at moment of button press (output)
ctyp      : cursor type (input)

 ctyp=1 tracking cross
 ctyp=2 cross-hair
 ctyp=3 rubber circle
 ctyp=4 rubber band
 ctyp=5 rubber rectangle

mode      : input mode

 mode=0 request
 mode=1 sample

Request locator:
return button number  1 = left is pressed
                     2 = middle is pressed
                     3 = right is pressed
     in sample mode:
                    11 = left is released
                    12 = middle is released
                    13 = right is released
                    -1 = nothing is pressed or released
                    -2 = leave the window
                  else = keycode (keyboard is pressed)
```

Int_t **RequestString**(int *x*, int *y*, char **text*)

```
Request string
x,y         : position where text is displayed
text        : text displayed (input), edited text (output)

Request string:
text is displayed and can be edited with Emacs-like keybinding
return termination code (0 for ESC, 1 for RETURN)
```

void **RescaleWindow**(int *wid*, int *w*, int *h*)

```
Rescale the window WID
wid  : Window identifier.
w    : Width
```

```
h    : Heigth
```

int **ResizePixmap**(int *wid,* int *w,* int *h*)
```
Resize a pixmap
wid : pixmap to be resized
w,h : Width and height of the pixmap.
```

void **ResizeWindow**(int *wid*)
```
Resize the current window if necessary
wid     : window to be resized
```

void **RestoreAddress**( )
```
Restore the adress of the saved window
```

void **SaveAddress**( )
```
Save the adress of the current active window
```

void **SelectWindow**(int *wid*)
```
Select window to which subsequent output is directed
wid   : window number returned by OpenWindow
```

void **SetCharacterUp**(Float_t *chupx,* Float_t *chupy*)
```
Set character up vectors
```

void **SetClipOFF**(int *wid*)
```
Switch off the clipping for the window wid
wid        : Window indentifier
```

void **SetClipRegion**(int *wid,* int *x,* int *y,* int *w,* int *h*)
```
Set clipping region for the window wid
wid        : Window indentifier
x,y        : origin of clipping rectangle
w,h        : size of clipping rectangle;
```

void **SetColor**(GC *gc,* int *ci*)
```
set the foreground color in GC
```

void **SetCursor**(int *wid,* ECursor *cursor*)
```
set the cursor
```

void **SetDoubleBuffer**(int *wid,* int *mode*)
```
Set the double buffer on/off on window WID
wid  : Window identifier.
            999 means all the opened windows.
mode : 1 double buffer is on
     0 double buffer is off
```

void **SetDrawMode**(EDrawMode *mode*)
```
Set the drawing mode
mode : drawing mode

 mode=1 copy
 mode=2 xor
 mode=3 invert
 mode=4 set the suitable mode for cursor echo according to
        the vendor.
```

void **SetFillColor**(Color_t *cindex*)
```
Set color index for fill areas
cindex : color index defined by SetColor
```

void **SetFillStyle**(Style_t *fstyle*)
```
Set fill area style
```

```
fstyle   : compound fill area interior style

fstyle = 1000*interiorstyle + styleindex
```

void **SetFillStyleIndex**(Int_t *style,* Int_t *fasi*)

```
Set fill area style index
style   : fill area interior style hollow or solid
```

void **SetInput**(int *inp*)

```
Set input on or off
```

void **SetLineColor**(Color_t *cindex*)

```
Set color index for lines
cindex    : color index
```

void **SetLineType**(int *n,* int **dash*)

```
Set line type
n         : length of dash list
dash(n)   : dash segment lengths

 if N.LE.0 use solid lines
 if N.GT.0 use dashed lines described by DASH(N)
 e.g. N=4,DASH=(6,3,1,3) gives a dashed-dotted line with dash length 6
 and a gap of 7 between dashes
```

void **SetLineStyle**(Style_t *lstyle*)

```
Set line style
```

void **SetLineWidth**(Width_t *width*)

```
Set line width
width   : line width in pixels
```

void **SetMarkerColor**(Color_t *cindex*)

```
Set color index for markers
cindex : color index defined by SetColor
```

void **SetMarkerSize**(Float_t *msize*)

```
Set marker size index for markers
msize  : marker scale factor
```

void **SetMarkerType**(int *type,* int *n,* XPoint **xy*)

```
Set marker type
type      : marker type
n         : length of marker description
xy        : list of points describing marker shape

  if N.EQ.0 marker is a single point
  if TYPE.EQ.0 marker is hollow circle of diameter N
  if TYPE.EQ.1 marker is filled circle of diameter N
  if TYPE.EQ.2 marker is a hollow polygon describe by line XY
  if TYPE.EQ.3 marker is a filled polygon describe by line XY
  if TYPE.EQ.4 marker is described by segmented line XY
  e.g. TYPE=4,N=4,XY=(-3,0,3,0,0,-3,0,3) sets a plus shape of 7x7 pixels
```

void **SetMarkerStyle**(Style_t *markerstyle*)

```
Set marker style
```

void **SetRGB**(int *cindex,* float *r,* float *g,* float *b*)

```
Set color intensities for given color index
cindex     : color index
r,g,b      : red, green, blue intensities between 0.0 and 1.0
```

void **SetTextAlign**(Short_t *talign*)

```
Set text alignment
txalh   : horizontal text alignment
txalv   : vertical text alignment
```

void **SetTextColor**(Color_t *cindex*)

```
Set color index for text
cindex    : color index defined by SetColor
```

Int_t **SetTextFont**(char **fontname,* ETextSetMode *mode*)

```
Set text font to specified name
mode       : loading flag
mode=0     : search if the font exist (kCheck)
mode=1     : search the font and load it if it exists (kLoad)
font       : font name

Set text font to specified name. This function returns 0 if
the specified font is found, 1 if not.
```

void **SetTextFont**(Int_t *fontnumber*)

```
Set current text font number
```

void **SetTextSize**(Float_t *textsize*)

```
Set current text size
```

void **Sync**(int *mode*)

```
Set synchronisation on or off
mode : synchronisation on/off
  mode=1  on
  mode<>0 off
```

void **UpdateWindow**(int *mode*)

```
Update display
mode : (1) update
       (0) sync

Synchronise client and server once (not permanent).
Copy the pixmap cws->drawing on the window cws->window
if the double buffer is on.
```

void **Warp**(int *ix,* int *iy*)

```
Set pointer position
ix       : New X coordinate of pointer
iy       : New Y coordinate of pointer
 (both coordinates are relative to the origin of the current window)
```

void **WritePixmap**(int *wid,* int *w,* int *h,* char **pxname*)

```
Write the pixmap IPX in the bitmap file PXNAME
pix         : Pixmap address
w,h         : Width and height of the pixmap.
lenname     : pixmap name length
pxname      : pixmap name
```

void **WriteGIF**(char **name*)

```
writes the current window into GIF files
```

void **ReadGIF**(int *x0,* int *y0,* const char **file*)

```
Load the gif a file in the current active window.
```

Author: René Brun, Olivier Couet 28/11/94
Last update: 1.03/01 22/08/97 12.40.16 by René Brun

TGXW

class **TGXW** : public TNamed, public TAttLine, public TAttFill, public TAttText, public TAttMarker

public:

class TGXW	**TGXW**( )
class TGXW	**TGXW**(Text_t* *name,* Text_t* *title*)
class TGXW	**TGXW**(class TGXW&)
void	**~TGXW**( )
class TClass*	**Class**( )
virtual void	**ClearWindow**( )
virtual void	**ClosePixmap**( )
virtual void	**CloseWindow**( )
virtual void	**CopyPixmap**(int *wid,* int *xpos,* int *ypos*)
virtual void	**CreateOpenGLContext**(int *wid = 0*)
virtual void	**DeleteOpenGLContext**(int *wid = 0*)
virtual const char*	**DisplayName**( )
virtual void	**DrawBox**(int *x1,* int *y1,* int *x2,* int *y2,* enum TGXW::EBoxMode *mode*)
virtual void	**DrawCellArray**(int *x1,* int *y1,* int *x2,* int *y2,* int *nx,* int *ny,* int* *ic*)
virtual void	**DrawFillArea**(int n, (*unknown*) TPoint* xy)
virtual void	**DrawLine**(int *x1,* int *y1,* int *x2,* int *y2*)
virtual void	**DrawPolyLine**(int n, (*unknown*) TPoint* xy)
virtual void	**DrawPolyMarker**(int n, (*unknown*) TPoint* xy)
virtual void	**DrawText**(int *x,* int *y,* float *angle,* float *mgn,* char* *text,* enum TGXW::ETextMode *mode*)
virtual void	**GetCharacterUp**(Float_t& *chupx,* Float_t& *chupy*)
virtual Int_t	**GetDoubleBuffer**(int *wid*)
enum TGXW::EDrawMode	**GetDrawMode**( )
virtual void	**GetGeometry**(int *wid,* int& *x,* int& *y,* unsigned int& *w,* unsigned int& *h*)
virtual void	**GetPlanes**(int& *nplanes*)
virtual void	**GetRGB**(int *index,* float& *r,* float& *g,* float& *b*)
virtual void	**GetTextExtent**(int& *w,* int& *h,* char* *mess*)
virtual Float_t	**GetTextMagnitude**( )
virtual Bool_t	**Init**(void* *display = 0*)
virtual Int_t	**InitWindow**(ULong_t *window*)
virtual class TClass*	**IsA**( )
virtual void	**MoveWindow**(int *wid,* int *x,* int *y*)
virtual Int_t	**OpenPixmap**(int *w,* int *h*)
virtual void	**QueryPointer**(int& *ix,* int& *iy*)
virtual void	**ReadGIF**(int *x0,* int *y0,* char* *file*)
virtual Int_t	**RequestLocator**(int *mode,* int *ctyp,* int& *x,* int& *y*)
virtual Int_t	**RequestString**(int *x,* int *y,* char* *text*)
virtual void	**RescaleWindow**(int *wid,* int *w,* int *h*)
virtual Int_t	**ResizePixmap**(int *wid,* int *w,* int *h*)
virtual void	**ResizeWindow**(int *wid*)
virtual void	**SelectWindow**(int *wid*)
virtual void	**SetCharacterUp**(Float_t *chupx,* Float_t *chupy*)
virtual void	**SetClipOFF**(int *wid*)
virtual void	**SetClipRegion**(int *wid,* int *x,* int *y,* int *w,* int *h*)
virtual void	**SetCursor**(int win, (*unknown*) ECursor cursor)
virtual void	**SetDoubleBuffer**(int *wid,* int *mode*)
virtual void	**SetDoubleBufferOFF**( )
virtual void	**SetDoubleBufferON**( )
virtual void	**SetDrawMode**(enum TGXW::EDrawMode *mode*)
virtual void	**SetFillColor**(Color_t *cindex*)
virtual void	**SetFillStyle**(Style_t *style*)

virtual void **SetLineColor**(Color_t *cindex*)
virtual void **SetLineStyle**(Style_t *linestyle*)
virtual void **SetLineType**(int *n,* int* *dash*)
virtual void **SetLineWidth**(Width_t *width*)
virtual void **SetMarkerColor**(Color_t *cindex*)
virtual void **SetMarkerSize**(Float_t *markersize*)
virtual void **SetMarkerStyle**(Style_t *markerstyle*)
virtual void **SetRGB**(int *cindex,* float *r,* float *g,* float *b*)
virtual void **SetTextAlign**(Short_t *talign = 11*)
virtual void **SetTextColor**(Color_t *cindex*)
virtual Int_t **SetTextFont**(char* *fontname,* enum TGXW::ETextSetMode *mode*)
virtual void **SetTextFont**(Int_t *fontnumber*)
virtual void **SetTextMagnitude**(Float_t *mgn*)
virtual void **SetTextSize**(Float_t *textsize*)
virtual void **ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void **Streamer**(class TBuffer& *b*)
virtual void **UpdateWindow**(int *mode*)
virtual void **Warp**(int *ix,* int *iy*)
virtual void **WriteGIF**(char* *name*)
virtual void **WritePixmap**(int *wid,* int *w,* int *h,* char* *pxname*)

Data Members

protected:
enum TGXW::EDrawMode **fDrawMode** *Drawing mode*
public:
static const enum TGXW::EDrawMode**kCopy**
static const enum TGXW::EDrawMode**kXor**
static const enum TGXW::EDrawMode**kInvert**
static const enum TGXW::EBoxMode**kHollow**
static const enum TGXW::EBoxMode**kFilled**
static const enum TGXW::ETextMode**kClear**
static const enum TGXW::ETextMode**kOpaque**
static const enum TGXW::ETextSetMode**kCheck**
static const enum TGXW::ETextSetMode**kLoad**

See Also: TGX11

Class Description

```
TGXW

Semi-Abstract base class defining a generic interface to the
underlying, low level, graphics system (X11, Win32, MacOS).
An instance of TGXW itself defines a batch interface to the graphics
system.
```

TGXW(Text_t *name, Text_t **title*) : **TNamed**(name, title),**TAttLine**(*1, 1, 1*), **TAttFill**(*1,* 1), **TAttText**(*11, 0, 1, 62,* 0.01), **TAttMarker**(*1, 1,* 1)

```
Ctor of ABC
```

Author: Fons Rademakers 3/12/95
Last update: 1.00/06 23/03/97 11.17.21 by René Brun

TH1

class **TH1** : public TNamed, public TAttLine, public TAttFill, public TAttMarker

private:

Int_t	**AxisChoice**(Option_t* *axis*)
void	**Build**( )
Int_t	**FitOptionsMake**(Option_t* *option*)
void	**PaintArrows**( )
void	**PaintBoxes**( )
void	**PaintColorLevels**( )
void	**PaintContour**( )
Int_t	**PaintContourLine**(Float_t *elev1,* Int_t *icont1,* Float_t *x1,* Float_t *y1,* Float_t *elev2,* Int_t *icont2,* Float_t *x2,* Float_t *y2,* Float_t* *xarr,* Float_t* *yarr,* Int_t* *itarr,* Int_t *ncont*)
void	**PaintDate**( )
void	**PaintErrors**( )
void	**PaintFile**( )
void	**PaintFrame**( )
void	**PaintFunction**( )
void	**PaintH3**(Option_t* *option*)
void	**PaintHist**( )
void	**PaintLego**( )
void	**PaintLegoAxis**(class TGaxis* *axis,* Float_t *ang*)
void	**PaintLive**( )
void	**PaintScatterPlot**( )
void	**PaintSurface**( )
void	**PaintTable**( )
void	**PaintTitle**( )
void	**PaintValues**( )
void	**RecalculateRange**( )
Int_t	**TableInit**( )
void	**UpdateOptions**( )

protected:

virtual void	**Copy**(class TObject& *hnew*)

public:

class TH1	**TH1**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1	**TH1**( )
class TH1	**TH1**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)
class TH1	**TH1**(class TH1&)
void	**~TH1**( )
virtual void	**Add**(class TH1* *h1,* class TH1* *h2,* Float_t *c1 = 1,* Float_t *c2 = 1*)
virtual void	**Add**(class TH1* *h1,* Float_t *c1 = 1*)
virtual void	**AddBinContent**(Int_t *bin,* Stat_t *w*)
virtual void	**AddBinContent**(Int_t *bin*)
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Divide**(class TH1* *h1,* class TH1* *h2,* Float_t *c1 = 1,* Float_t *c2 = 1,* Option_t* *option*)
virtual void	**Divide**(class TH1* *h1*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**DrawPanel**( )
virtual void	**Eval**(class TF1* *f1,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual void	**Fill**(Axis_t *x,* Stat_t *w*)

virtual void	**Fill**(Axis_t *x*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)
virtual void	**Fill12**(Axis_t *x,* Axis_t *y*)
virtual void	**FillN**(Int_t *ntimes,* Axis_t* *x,* Axis_t* *y,* Double_t* *w*)
virtual void	**FillN**(Int_t *ntimes,* Axis_t* *x,* Double_t* *w*)
void	**FillRandom**(char* *fname,* Int_t *ntimes = 5000*)
virtual void	**Fit**(Text_t* *formula,* Option_t* *option,* Option_t* *goption,* Float_t *xmin = 0,* Float_t *xmax = 0*)
virtual void	**FitPanel**( )
virtual Color_t	**GetAxisColor**(Option_t* *axis = X*)
virtual Float_t	**GetBarOffset**( )
virtual Float_t	**GetBarWidth**( )
virtual Int_t	**GetBin**(Int_t *binx,* Int_t *biny = 0,* Int_t *binz = 0*)
virtual Axis_t	**GetBinCenter**(Int_t *bin*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual Stat_t	**GetBinError**(Int_t *bin*)
virtual Text_t*	**GetBinLabel**(Int_t *bin*)
virtual Axis_t	**GetBinLowEdge**(Int_t *bin*)
virtual Axis_t	**GetBinWidth**(Int_t *bin*)
virtual Stat_t	**GetCellContent**(Int_t *binx,* Int_t *biny*)
virtual Stat_t	**GetCellError**(Int_t *binx,* Int_t *biny*)
virtual void	**GetCenter**(Axis_t* *center*)
virtual Int_t	**GetContour**(Float_t* *levels = 0*)
virtual Float_t	**GetContourLevel**(Int_t *level*)
virtual Int_t	**GetDimension**( )
class TDirectory*	**GetDirectory**( )
virtual Stat_t	**GetEntries**( )
virtual class TF1*	**GetFunction**(Text_t* *name*)
virtual Stat_t	**GetIntegral**( )
virtual Stat_t	**GetIntegral**(Int_t *binx1,* Int_t *binx2,* Int_t *biny1,* Int_t *biny2*)
virtual Stat_t	**GetIntegral**(Int_t *binx1,* Int_t *binx2*)
virtual Stat_t	**GetIntegral**(Int_t *binx1,* Int_t *binx2,* Int_t *biny1,* Int_t *biny2,* Int_t *binz1,* Int_t *binz2*)
virtual void	**GetLabel**(Text_t* *label*)
virtual Color_t	**GetLabelColor**(Option_t* *axis = X*)
virtual Style_t	**GetLabelFont**(Option_t* *axis = X*)
virtual Float_t	**GetLabelOffset**(Option_t* *axis = X*)
virtual Float_t	**GetLabelSize**(Option_t* *axis = X*)
class TList*	**GetListOfFunctions**( )
virtual void	**GetLowEdge**(Axis_t* *edge*)
virtual Float_t	**GetMaximum**( )
virtual Stat_t	**GetMean**( )
virtual Float_t	**GetMinimum**( )
virtual Int_t	**GetNbinsX**( )
virtual Int_t	**GetNbinsY**( )
virtual Int_t	**GetNbinsZ**( )
virtual Int_t	**GetNdivisions**(Option_t* *axis = X*)
virtual Text_t*	**GetObjectInfo**(Int_t *px,* Int_t *py*)
const Option_t*	**GetOption**( )
virtual Stat_t	**GetRMS**( )
virtual Stat_t	**GetSumOfWeights**( )
virtual Float_t	**GetTickLength**(Option_t* *axis = X*)
virtual Float_t	**GetTitleOffset**(Option_t* *axis = X*)
virtual class TAxis*	**GetXaxis**( )
virtual class TAxis*	**GetYaxis**( )
virtual class TAxis*	**GetZaxis**( )
virtual class TClass*	**IsA**( )

virtual Int_t	**LocateBinary**(Double_t *value,* Double_t* *array,* Int_t *nbins*)
virtual Int_t	**MakeChopt**(Option_t* *option*)
virtual void	**Multiply**(class TH1* *h1,* class TH1* *h2,* Float_t *c1 = 1,* Float_t *c2 = 1,* Option_t* *option*)
virtual void	**Multiply**(class TH1* *h1*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintAxis**( )
virtual Int_t	**PaintInit**( )
virtual void	**PaintPalette**( )
virtual void	**PaintStat**(Int_t *dostat,* class TF1* *fit*)
virtual void	**Print**(Option_t* *option*)
class TProfile*	**ProfileX**(char* *name = _pfx,* Int_t *firstybin = 0,* Int_t *lastybin = 0,* Option_t* *option*)
class TProfile*	**ProfileY**(char* *name = _pfy,* Int_t *firstxbin = 0,* Int_t *lastxbin = 0,* Option_t* *option*)
class TH1D*	**ProjectionX**(char* *name = _px,* Int_t *firstybin = 0,* Int_t *lastybin = 0,* Option_t* *option*)
class TH1D*	**ProjectionY**(char* *name = _py,* Int_t *firstxbin = 0,* Int_t *lastxbin = 0,* Option_t* *option*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**Scale**(Float_t *c1 = 1*)
virtual void	**SetAxisColor**(Color_t *color = 1,* Option_t* *axis = X*)
virtual void	**SetBarOffset**(Float_t *offset = 0.25*)
virtual void	**SetBarWidth**(Float_t *width = 0.5*)
virtual void	**SetBinContent**(Int_t *bin,* Stat_t *content*)
virtual void	**SetBinError**(Int_t *bin,* Stat_t *error*)
virtual void	**SetBinLabel**(Int_t *bin,* Text_t* *label*)
virtual void	**SetBins**(Int_t *nx,* Float_t *xmin,* Float_t *xmax,* Int_t *ny,* Float_t *ymin,* Float_t *ymax*)
virtual void	**SetBins**(Int_t *nx,* Float_t *xmin,* Float_t *xmax*)
virtual void	**SetCellContent**(Int_t *binx,* Int_t *biny,* Stat_t *content*)
virtual void	**SetCellError**(Int_t *binx,* Int_t *biny,* Stat_t *content*)
virtual void	**SetContent**(Stat_t* *content*)
virtual void	**SetContour**(Int_t *nlevels*)
virtual void	**SetContour**(Int_t *nlevels,* Float_t* *levels*)
virtual void	**SetContourLevel**(Int_t *level,* Float_t *value*)
virtual void	**SetDirectory**(class TDirectory* *dir*)
virtual void	**SetEntries**(Stat_t *n*)
virtual void	**SetError**(Stat_t* *error*)
virtual void	**SetLabel**(Text_t* *label*)
virtual void	**SetLabelColor**(Color_t *color = 1,* Option_t* *axis = X*)
virtual void	**SetLabelFont**(Style_t *font = 62,* Option_t* *axis = X*)
virtual void	**SetLabelOffset**(Float_t *offset = 0.005,* Option_t* *axis = X*)
virtual void	**SetLabelSize**(Float_t *size = 0.02,* Option_t* *axis = X*)
virtual void	**SetMaximum**(Float_t *maximum = -1111*)
virtual void	**SetMinimum**(Float_t *minimum = -1111*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetNdivisions**(Int_t *n = 510,* Option_t* *axis = X*)
virtual void	**SetOption**(Option_t* *option* =)
virtual void	**SetTickLength**(Float_t *length = 0.02,* Option_t* *axis = X*)
virtual void	**SetTitleOffset**(Float_t *offset = 1,* Option_t* *axis = X*)
virtual void	**SetXTitle**(Text_t* *title*)
virtual void	**SetYTitle**(Text_t* *title*)
virtual void	**SetZTitle**(Text_t* *title*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Smooth**(Option_t* *option*)
virtual void	**Streamer**(class TBuffer& *b*)

virtual void	**Sumw2**( )
void	**UseCurrentStyle**( )

Data Members

protected:

Int_t	**fNcells**	*number of bins(1D*), cells (2D) +U/Overflows
class TAxis	**fXaxis**	*X axis descriptor*
class TAxis	**fYaxis**	*Y axis descriptor*
class TAxis	**fZaxis**	*Z axis descriptor*
Short_t	**fBarOffset**	*(1000*offset*) for bar charts or legos
Short_t	**fBarWidth**	*(1000*width*) for bar charts or legos
Stat_t	**fEntries**	*Number of entries*
Stat_t	**fTsumw**	*Total Sum of weights*
Stat_t	**fTsumw2**	*Total Sum of squares of weights*
Stat_t	**fTsumwx**	*Total Sum of weight*X*
Stat_t	**fTsumwx2**	*Total Sum of weight*X*X*
Float_t	**fMaximum**	*Maximum value for plotting*
Float_t	**fMinimum**	*Minimum value for plotting*
Float_t	**fNormfactor**	*Normalization factor*
class TArrayF	**fContour**	*Array to display contour levels*
class TArrayD	**fSumw2**	*Array of sum of squares of weights*
class TString	**fOption**	*histogram options*
class TList*	**fFunctions**	*Pointer to list of functions (fits and user*)
class TDirectory*	**fDirectory**	*Pointer to directory holding this histogram*
Int_t	**fDimension**	*Histogram dimension (1, 2 or 3 dim*)

See Also: TH1C, TH1D, TH1F, TH1S

Class Description

```
The H I S T O G R A M   Classes
ROOT supports the following histogram types:

1-D histograms:
   TH1C : histograms with one byte per channel. Maximum bin content = 255
   TH1S : histograms with one short per channel. Maximum bin content = 65535
   TH1F : histograms with one float per channel. Maximum precision 7 digits
   TH1D : histograms with one double per channel. Maximum precision 14 digits

2-D histograms:
   TH2C : histograms with one byte per channel. Maximum bin content = 255
   TH2S : histograms with one short per channel. Maximum bin content = 65535
   TH2F : histograms with one float per channel. Maximum precision 7 digits
   TH2D : histograms with one double per channel. Maximum precision 14 digits

3-D histograms:
   TH3C : histograms with one byte per channel. Maximum bin content = 255
   TH3S : histograms with one short per channel. Maximum bin content = 65535
   TH3F : histograms with one float per channel. Maximum precision 7 digits
   TH3D : histograms with one double per channel. Maximum precision 14 digits

Profile histograms: See class TProfile

All histogram classes are derived from the base class TH1
The TH*C classes also inherit from the array class TArrayC.
The TH*S classes also inherit from the array class TArrayS.
The TH*F classes also inherit from the array class TArrayF.
The TH*D classes also inherit from the array class TArrayD.
```

TH1( ): **TNamed**( )

```
Histogram default constructor
```

~TH1( )
```
Histogram default destructor
```

TH1(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*):**TNamed**(*name, title*)
```
Normal constructor for fix bin size histograms
  Creates the main histogram structure:
     name   : name of histogram (avoid blanks)
     title  : histogram title
     nbins  : number of bins
     xlow   : low edge of first bin
     xup    : upper edge of last bin (not included in last bin)

   When an histogram is created, it is automatically added to the list
   of special objects in the current directory.
   To find the pointer to this histogram in the current directory
   by its name, do:
   TH1F *h1 = (TH1F*)gDirectory->FindObject(name);
```

TH1(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t **xbins*):**TNamed**(*name, title*)
```
Normal constructor for variable bin size histograms
Creates the main histogram structure:
  name   : name of histogram (avoid blanks)
  title  : histogram title
  nbins  : number of bins
  xbins  : array of low-edges for each bin
           This is an array of size nbins+1
```

void **Build**( )
```
Creates histogram basic data structure
```

void **Add**(TH1 **h1,* Float_t *c1*)
```
Performs the operation: this = this + c1*h1
if errors are defined (see TH1::Sumw2), errors are also recalculated
```

void **Add**(TH1 **h1,* TH1 **h2,* Float_t *c1,* Float_t *c2*)
```
Replace contents of this histogram by the addition of h1 and h2
this = c1*h1 + c2*h2
if errors are defined (see TH1::Sumw2), errors are also recalculated
```

void **AddBinContent**(Int_t *bin*)
```
Increment bin content by 1
```

void **AddBinContent**(Int_t *bin,* Stat_t *w*)
```
Increment bin content by a weight w
```

void **Copy**(TObject &*obj*)
```
Copy this histogram structure to newth1
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a line
Compute the closest distance of approach from point px,py to elements
of an histogram.
The distance is computed in pixels units.

Algorithm:
Currently, this simple model computes the distance from the mouse
to the histogram contour only.
```

void **Divide**(TH1 **h1*)
```
Divide this histogram by h1
this = this/h1
if errors are defined (see TH1::Sumw2), errors are also recalculated
If option B is given, binomial errors are computed
```

void **Divide**(TH1 **h1*, TH1 **h2*, Float_t *c1*, Float_t *c2*, Option_t **option*)

```
Replace contents of this histogram by the division of h1 by h2
this = c1*h1/(c2*h2)

if errors are defined (see TH1::Sumw2), errors are also recalculated
If option B is given, binomial errors are computed
```

void **Draw**(Option_t **option*)

```
Draw this histogram with options
This histogram is added in the list of objects to be drawn in the
current pad.
It is important to note that only a reference to this histogram is added.
This means that if the histogram contents change between pad redraws,
the current histogram contents will be shown.
Use function DrawCopy to force a copy of the histogram at the time
of the Draw operation in the Pad list.

The following options are supported on all types:
 "SAME"   : Superimpose on previous picture in the same pad
 "CYL"    : Use Cylindrical coordinates
 "POL"    : Use Polar coordinates
 "SPH"    : Use Spherical coordinates
 "PSR"    : Use PseudoRapidity/Phi coordinates
 "LEGO"   : Draw a lego plot with hidden line removal
 "LEGO1"  : Draw a lego plot with hidden surface removal
 "LEGO2"  : Draw a lego plot using colors to show the cell contents
 "SURF"   : Draw a surface plot with hidden line removal
 "SURF1"  : Draw a surface plot with hidden surface removal
 "SURF2"  : Draw a surface plot using colors to show the cell contents
 "SURF3"  : same as SURF with in addition a contour view drawn on the top
 "SURF4"  : Draw a surface using Gouraud shading

The following options are supported for 1-D types:
 "A"      : Do not draw the axis labels and tick marks
 "B"      : Bar chart option
 "C"      : Draw a smooth Curve througth the histogram bins
 "E"      : Draw error bars
 "E0"     : Draw error bars including bins with o contents
 "E1"     : Draw error bars with perpendicular lines at the edges
 "E2"     : Draw error bars with rectangles
 "E3"     : Draw a fill area througth the end points of the vertical error bars
 "E4"     : Draw a smoothed filled area through the end points of the error bars
 "L"      : Draw a line througth the bin contents
 "P"      : Draw current marker at each bin
 "*"      : Draw a * at each bin

The following options are supported for 2-D types:
 "ARR"    : arrow mode. Shows gradient between adjacent cells
 "BOX"    : a box is drawn for each cell with surface proportional to contents
 "COL"    : a box is drawn for each cell with a color scale varying with con-
tents
 "CONT"   : Draw a contour plot (same as CONT3)
 "CONT0"  : Draw a contour plot using colors to distinguish contours
 "CONT1"  : Draw a contour plot using line styles to distinguish contours
 "CONT2"  : Draw a contour plot using the same line style for all contours
 "CONT3"  : Draw a contour plot using fill area colors
 "FB"     : With LEGO or SURFACE, suppress the Front-Box
 "BB"     : With LEGO or SURFACE, suppress the Back-Box
 "SCAT"   : Draw a scatter-plot (default)

Note that most options above can be concatenated, example:
h-Draw("e1same");
Options are case insensitive
```

void **DrawCopy**(Option_t **option*)

```
Copy this histogram and Draw in the current pad
Once the histogram is drawn into the pad, any further modification
using graphics input will be made on the copy of the histogram,
and not to the original object.

See Draw for the list of options
```

void **DrawPanel**( )

```
Display a panel with all histogram drawing options
See class TDrawPanelHist for example
```

void **Eval**(TF1 **f1*, Option_t **option*)

```
Evaluate function f1 at the center of bins of this histogram
If option "R" is specified, the function is evaluated only
for the bins included in the function range.
If option "A" is specified, the value of the function is added to the
existing bin contents
```

void **ExecuteEvent**(Int_t *event*, Int_t *px*, Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a histogram is clicked with the locator

If Left button clicked on the bin top value, then the content of this bin
is modified according to the new position of the mouse when it is released.
```

void **Fill**(Axis_t *x*)

```
Increment bin with abscissa X by 1
if x is less than the low-edge of the first bin, the Underflow bin is incre-
mented
if x is greater than the upper edge of last bin, the Overflow bin is incremented

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by 1 in the bin corresponding to x.
```

void **Fill**(Axis_t *x*, Stat_t *w*)

```
Increment bin with abscissa X with a weight w
if x is less than the low-edge of the first bin, the Underflow bin is incre-
mented
if x is greater than the upper edge of last bin, the Overflow bin is incremented

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the bin corresponding to x.
```

void **FillN**(Int_t *ntimes*, Axis_t **x*, Double_t **w*)

```
Fill this histogram with an array x and weights w
ntimes:  number of entries in arrays x and w
x:       array of values to be histogrammed
w:       array of weighs

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w[i]^2 in the bin corresponding to x[i].
if w is NULL each entry is assumed a weight=1
```

void **Fill12**(Axis_t *x*, Axis_t *y*)

```
Increment cell defined by x,y by 1
if x or/and y is less than the low-edge of the corresponding axis first bin,
the Underflow cell is incremented.
if x or/and y is greater than the upper edge of corresponding axis last bin,
the Overflow cell is incremented.
```

```
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by 1in the cell corresponding to x,y.
```

void **Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)

```
Increment cell defined by x,y by a weight w
if x or/and y is less than the low-edge of the corresponding axis first bin,
the Underflow cell is incremented.
if x or/and y is greater than the upper edge of corresponding axis last bin,
the Overflow cell is incremented.

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y.
```

void **Fill**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)

```
Increment cell defined by x,y,z by a weight w
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y,z.
```

void **FillN**(Int_t *ntimes,* Axis_t **x,* Axis_t **y,* Double_t **w*)

```
Fill a 2-D histogram with an array of values and weights
ntimes:  number of entries in arrays x and w
x:       array of x values to be histogrammed
y:       array of y values to be histogrammed
w:       array of weights

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w[i]^2 in the cell corresponding to x[i],y[i].
if w is NULL each entry is assumed a weight=1
```

void **FillRandom**(char **fname,* Int_t *ntimes*)

```
Fill histogram following distribution in function fname
The distribution contained in the function fname (TF1) is integrated
over the channel contents.
It is normalized to 1.
Getting one random number implies:
  - Generating a random number between 0 and 1 (say r1)
  - Look in which bin in the normalized integral r1 corresponds to
  - Fill histogram channel
ntimes random numbers are generated
```

void **Fit**(const Text_t **fname,* Option_t **option,* Option_t **goption,* Float_t *xxmin,* Float_t *xxmax*)

```
Fit histogram with function fname
fname is the name of an already predefined function created by TF1 or TF2
Predefined functions such as Gaus, Expo and Poln are automatically
created by ROOT.

The list of fit options is given in parameter option.
   option = "W"  Set all errors to 1
          = "I" Use integral of function in bin instead of value at bin center
          = "Q" Quiet mode (minimum printing)
          = "V" Verbose mode (default is between Q and V)
          = "B" Some parameters are bound
          = "E" Perform better Errors estimation using Minos technique
          = "R" Use the Range specified in the function range
          = "N" Do not store the graphics function, do not draw
          = "0" Do not plot the result of the fit. By default the fitted func-
tion
                is drawn unless the option"N" above is specified.
```

```
     = "+" Add this new fitted function to the list of fitted functions
            (by default, any previous function is deleted)

When the fit is drawn (by default), the parameter goption may be used
to specify a list of graphics options. See TH1::Draw for a complete
list of these options.

In order to use the Range option, one must first create a function
with the expression to be fitted. For example, if your histogram
has a defined range between -4 and 4 and you want to fit a gaussian
only in the interval 1 to 3, you can do:
     TF1 *f1 = new TF1("f1","gaus",1,3);
     histo->Fit("f1","R");

In a similar way, to use the Bound option, you must set the limits
for the bound parameters:
     f1->SetParLimits(p_number, parmin, parmax);

Note that option "I" gives better resultes but is slower.
```

void **FitPanel**( )

```
Display a panel with all histogram fit options
See class TFitPanel for example
```

Text_t* **GetObjectInfo**(Int_t *px,* Int_t *py*)

```
Redefines TObject::GetObjectInfo.
Displays the histogram info (bin number, contents, integral up to bin
corresponding to cursor position px,py
```

Int_t **MakeChopt**(Option_t **choptin*)

```
Decode string chopt and fill Hoption structure
```

Int_t **FitOptionsMake**(Option_t **choptin*)

```
Decode string choptin and fill Foption structure
```

Int_t **GetBin**(Int_t *binx,* Int_t *biny,* Int_t *binz*)

```
Return Global bin number corresponding to binx,y,z
2-D and 3-D histograms are represented with a one dimensional
structure.
This has the advantage that all existing functions, such as
  GetBinContent, GetBinError, GetBinFunction work for all dimensions.
```

Stat_t **GetBinContent**(Int_t *bin*)

```
Return content of global bin number bin
```

Int_t **LocateBinary**(Double_t *value,* Double_t **array,* Int_t *nbins*)

```
Binary search in an array of nbins to locate value
array is supposed  to be sorted prior to this call.
If match is found, function returns position of element.
If no match found, function gives nearest element smaller than value.
```

void **Multiply**(TH1 **h1*)

```
Multiply this histogram by h1
this = this*h1

If errors of this are available (TH1::Sumw2), errors are recalculated
```

void **Multiply**(TH1 **h1,* TH1 **h2,* Float_t *c1,* Float_t *c2,* Option_t **option*)

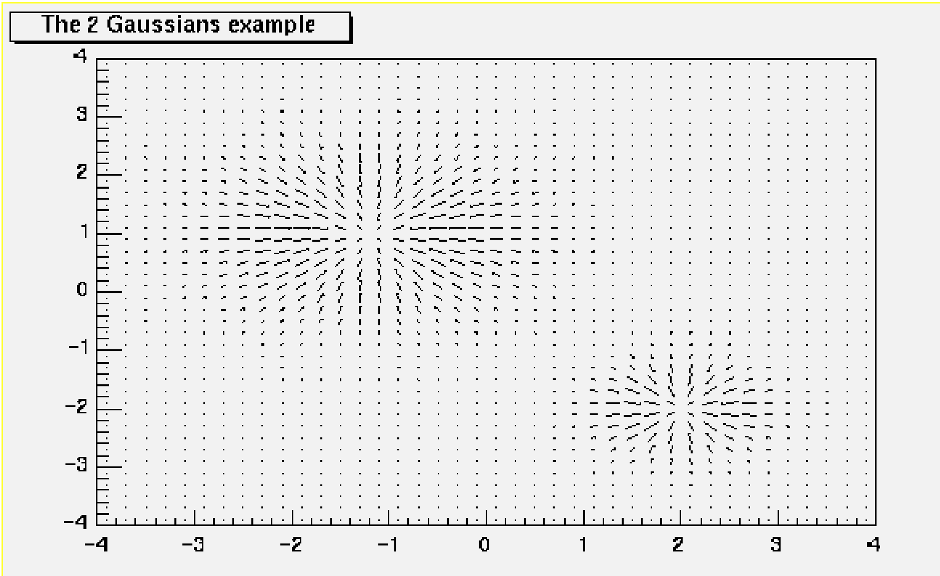
```
Replace contents of this histogram by multiplication of h1 by h2
this = (c1*h1)*(c2*h2)

If errors of this are available (TH1::Sumw2), errors are recalculated
```

void **Paint**(Option_t **option*)

```
Control routine to paint any kind of histograms
This function is automatically called by TCanvas::Update.
(see TH1::Draw for the list of options)
```

void **PaintArrows**( )

```
Control function to draw a table as an arrow plot
 For each cell (i,j) an arrow is drawn
 The orientation of the arrow follows the cell gradient
```

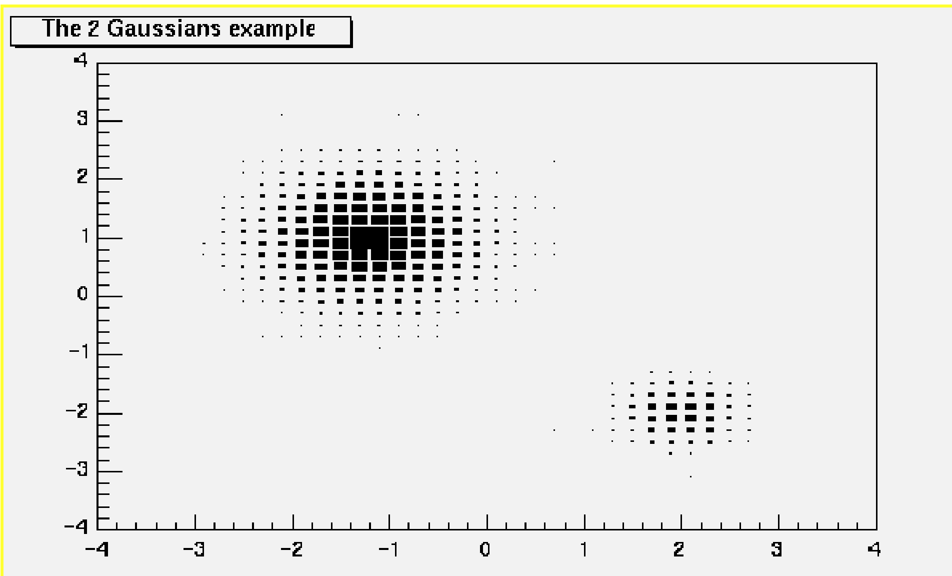


void **PaintAxis**( )

```
Draw axis of an histogram
```

void **PaintBoxes**( )

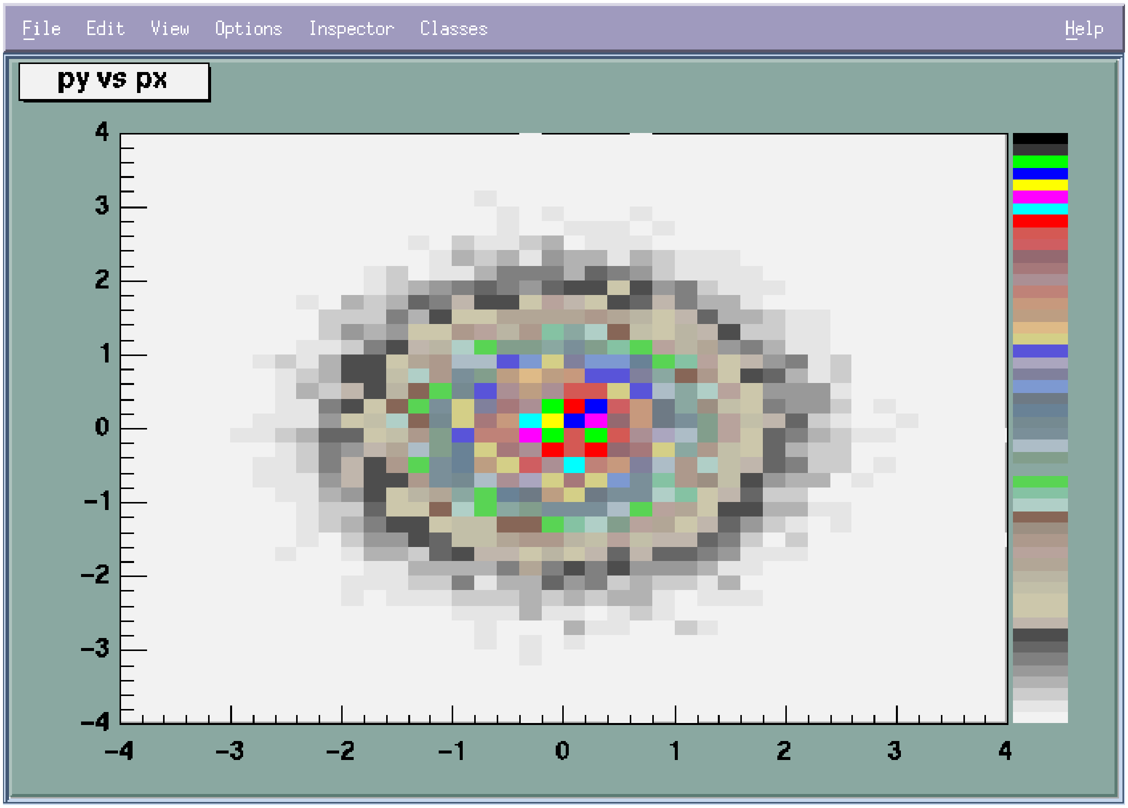
```
Control function to draw a table as a box plot
 For each cell (i,j) a box is drawn.
 The size of the box is proportional to the cell content.
```



void **PaintColorLevels**( )

```
Control function to draw a table as a color plot
```

```
For each cell (i,j) a box is drawn with a color proportional
to the cell content.
The color table used is defined in the current style (gStyle).
The color palette in TStyle can be modified via TStyle::SetPalette.
```

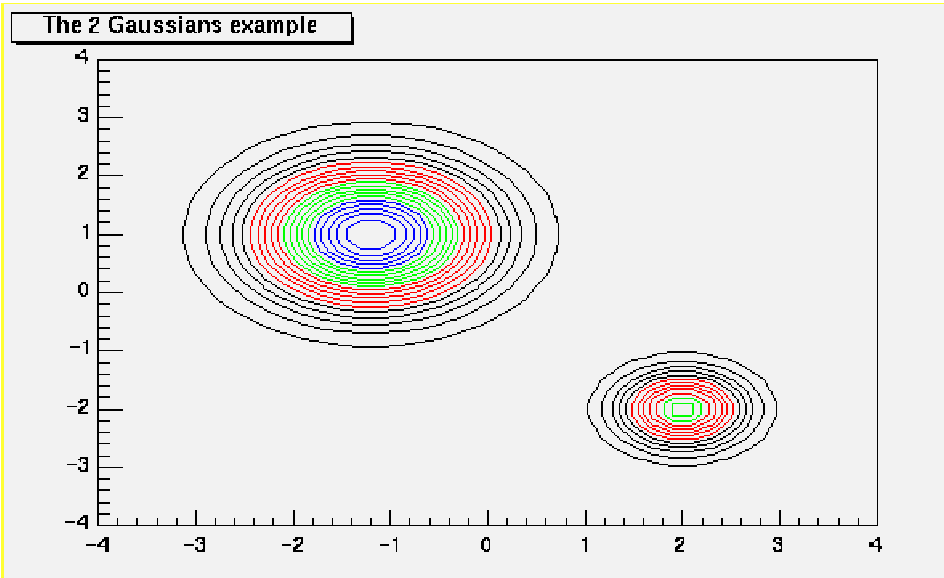


void **PaintPalette**( )

```
Paint the color palette on the right side of the pad
```

void **PaintContour**( )

```
Control function to draw a table as a contour plot
Hoption.Contour may have the following values:
  1  The contour is drawn with filled colour levels.
     The levels are equidistant.
 11  Use colour to distinguish contours.
 12  Use line style to distinguish contours.
 13  Line style and colour are the same for all contours.
```



Int_t **PaintContourLine**(Float_t *elev1,* Int_t *icont1,* Float_t *x1,* Float_t *y1,*Float_t elev2, Int_t icont2, Float_t x2, Float_t y2,Float_t *xarr, Float_t *yarr, Int_t *itarr, Int_t *ncontour*)

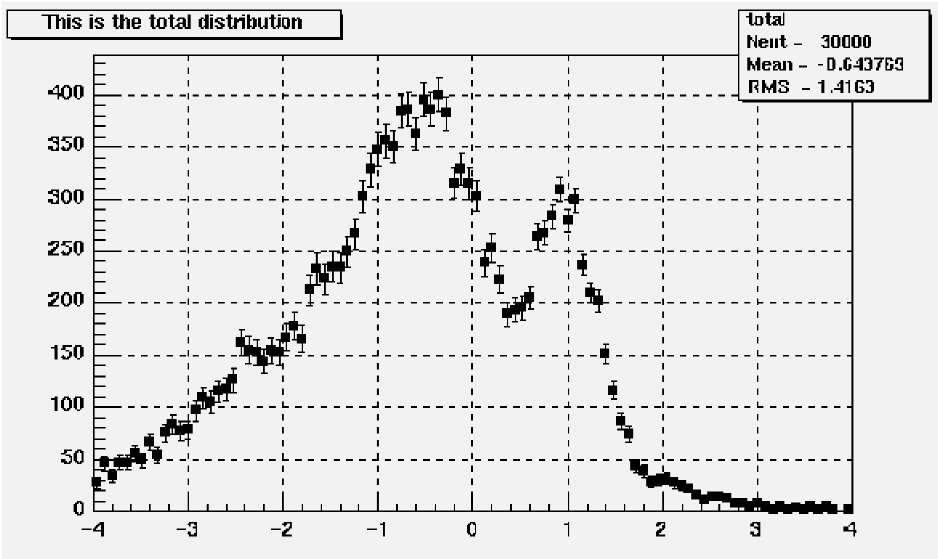
```
Fill the matrix XARR YARR for Contour Plot
```

Int_t **PaintInit**( )

```
Compute histogram parameters used by the drawing routines
```

void **PaintErrors**( )

```
Draw histogram error bars

Draws error bars for the current histogram. The current polymarker
is drawn at the centre of the errors according to CHOPT:

' ' Coordinates are expressed in histogram coordinates
    (of the last drawn histogram). Error bars are drawn.
'1' Small lines are drawn at the end of the error bars.
'2' Error rectangles are drawn.
'3' A filled area is drawn through the end points of the vertical
    error bars.
'4' A smoothed filled area is drawn through the end points of the
    vertical error bars.
'0' Turn off the symbols clipping.
```



void **PaintFrame**( )

```
Calculate range and clear pad (canvas)
```

void **PaintFunction**( )

```
Paint functions associated to an histogram
An associated function is created by TH1::Fit. Note that more than
on fitted function can be associated with one histogram (see TH1::Fit).

A TF1 object can be added to the list of associated functions directly
by a user without calling TH1::Fit.
To add a new function to the list of associated functions, do
  h->GetListOfFunctions()->Add(f1);
To retrieve a function by name from this list, do:
  TF1 *f1 = (TF1*)h->GetListOfFunctions()->FindObject(name);
or
  TF1 *f1 = h->GetFunction(name);
```

void **PaintHist**( )

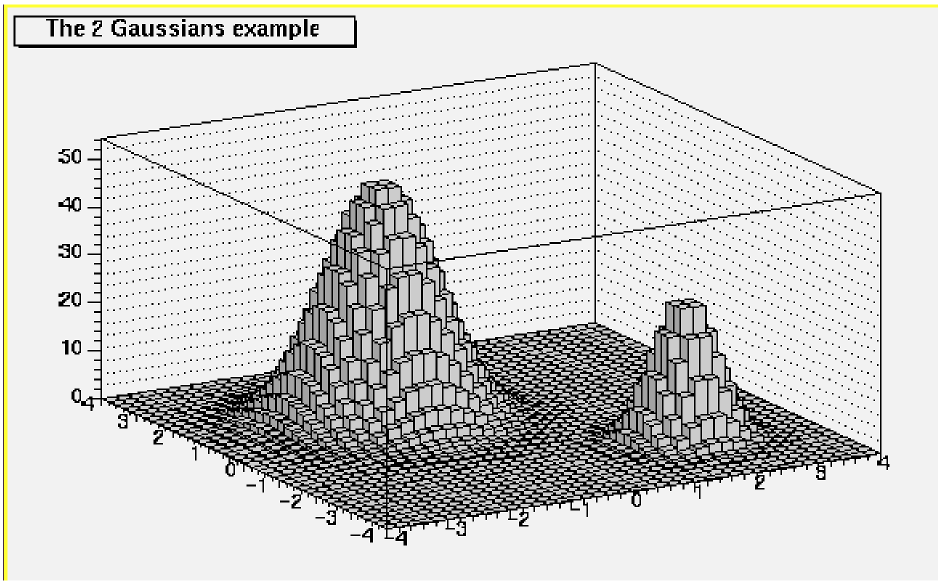
```
Control routine to draw an histogram
```

void **PaintH3**(Option_t **option*)

```
Control function to draw a 3-D histogram
```

void **PaintLego**( )

```
Control function to draw a table as a lego plot
  In a lego plot, cell contents are represented as 3-d boxes.
  The height of the box is proportional to the cell content.

 A lego plot can be represented in several coordinate systems.
 Default system is Cartesian coordinates.
 Possible systems are CYL,POL,SPH,PSR.

See TH1::Draw for the list of Lego options.
See TLego for more examples of lego options.
```

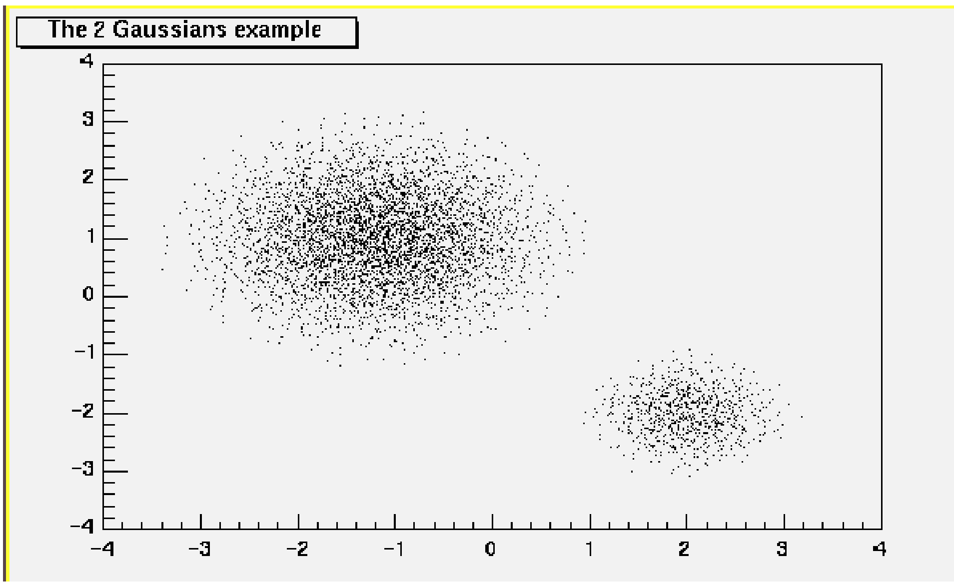


void **PaintLegoAxis**(TGaxis *axis*, Float_t *ang*)

```
Draw the axis for legos and surface plots
```

void **PaintScatterPlot**( )

```
Control function to draw a table as a scatter plot
 For each cell (i,j) a number of points proportional to the cell
 content is drawn.
 A maximum of 500 points per cell is drawn. If the maximum is above 500
 contents are normalized to 500.
```



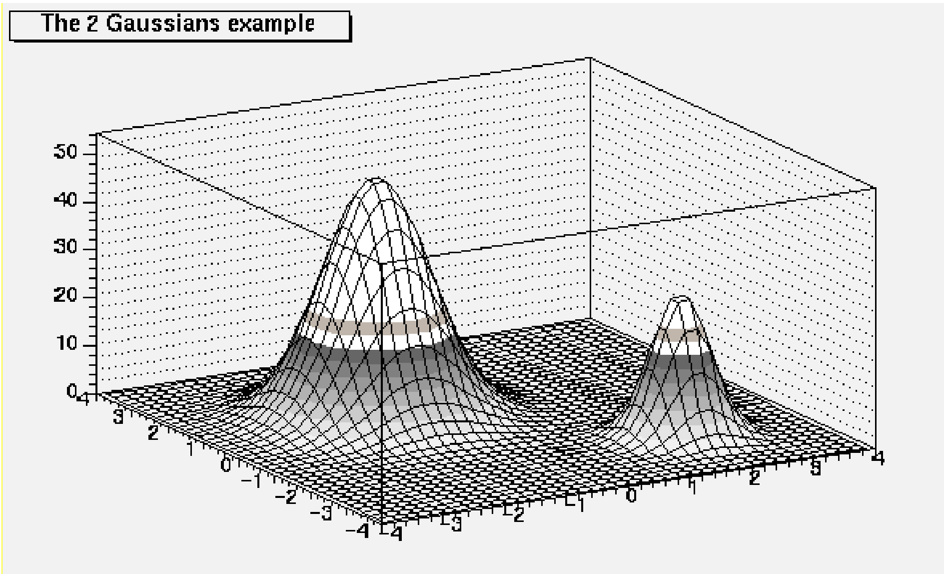
void **PaintStat**(Int_t *dostat*, TF1 **fit*)

```
Draw the statistics box
The type of information printed in the histogram statistics box
can be selected via gStyle->SetOptStat(mode).
The parameter mode can be = nemruo  (default = 111100)
 n = 1;  name of histogram is printed
 e = 1;  number of entries printed
 m = 1;  mean value printed
 r = 1;  rms printed
 u = 1;  number of underflows printed
```

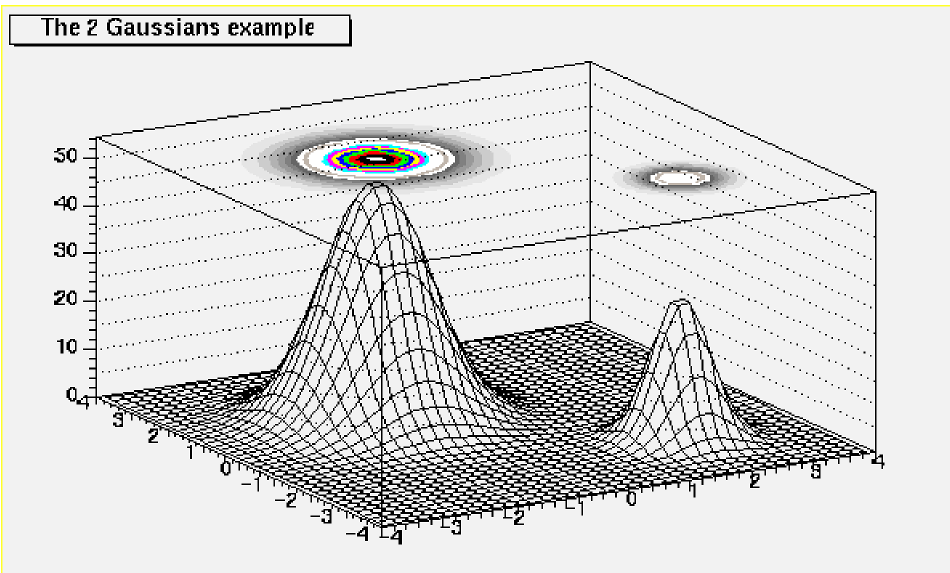
```
 o = 1;  number of overflows printed
Example: gStyle->SetOptStat(11);
        print only name of histogram and number of entries.
```

void **PaintSurface**( )

```
Control function to draw a table as a surface plot
  In a surface plot, cell contents are represented as a mesh.
  The height of the mesh is proportional to the cell content.

 A surface plot can be represented in several coordinate systems.
 Default system is Cartesian coordinates.
 Possible systems are CYL,POL,SPH,PSR.

See TH1::Draw for a list of Surface options
The following picture is generated with option SURF1.
```




```
The following picture is generated with option SURF3.
```



void **PaintTable**( )

```
Control function to draw 2-D/3-D tables
```

TProfile* **ProfileX**(const char **name,* Int_t *firstybin,* Int_t *lastybin,* Option_t **option*)

```
Project a 2-D histogram into a profile histogram along X
The projection is made from the channels along the Y axis
ranging from firstybin to lastybin included.
```

TProfile* **ProfileY**(const char **name,* Int_t *firstxbin,* Int_t *lastxbin,* Option_t **option*)

```
Project a 2-D histogram into a profile histogram along Y
The projection is made from the channels along the X axis
ranging from firstxbin to lastxbin included.
```

TH1D* **ProjectionX**(const char **name,* Int_t *firstybin,* Int_t *lastybin,* Option_t **option*)

```
Project a 2-D histogram into a 1-D histogram along X
The projection is always of the type TH1F.
The projection is made from the channels along the Y axis
ranging from firstybin to lastybin included.

if option 'E" is specified, the errors are computed.
```

TH1D* **ProjectionY**(const char **name,* Int_t *firstxbin,* Int_t *lastxbin,* Option_t **option*)

```
Project a 2-D histogram into a 1-D histogram along Y
The projection is always of the type TH1D.
The projection is made from the channels along the X axis
ranging from firstxbin to lastxbin included.

if option 'E" is specified, the errors are computed.
```

void **Scale**(Float_t *c1*)

```
Multiply contents of this histogram by a constant c1
this = c1*this
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Int_t **TableInit**( )

```
Initialize various options to draw tables
```

void **PaintTitle**( )

```
Draw the histogram title
```

void **Print**(Option_t **option*)

```
Print some global quantities for this histogram
If option "all" is given, bin contents and errors are also printed.
```

void **RecalculateRange**( )

```
Recompute the histogram range following graphics operations
```

void **Reset**(Option_t **option*)

```
Reset this histogram: contents, errors, etc
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **UpdateOptions**( )

```
Change Drawing options of an already drawn histogram
```

void **UseCurrentStyle**( )

```
Replace current attributes by current style
```

void **Smooth**(Option_t **option*)

```
Smooth contents of this histogram
```

Stat_t **GetIntegral**( )

```
Return integral of bin contents (all bins)
```

Stat_t **GetIntegral**(Int_t *binx1,* Int_t *binx2*)
```
Return integral of bin contents between binx1 and binx2 for a 1-D histogram
```

Stat_t **GetIntegral**(Int_t *binx1,* Int_t *binx2,* Int_t *biny1,* Int_t *biny2*)
```
Return integral of bin contents in range [binx1,binx2],[biny1,biny2]
for a 2-D histogram
```

Stat_t **GetIntegral**(Int_t *binx1,* Int_t *binx2,* Int_t *biny1,* Int_t *biny2,* Int_t *binz1,* Int_t *binz2*)
```
Return integral of bin contents in range
[binx1,binx2],[biny1,biny2],[binz1,binz2]
for a 3-D histogram
```

Stat_t **GetMean**( )
```
Return mean value of this histogram along the X axis
```

Stat_t **GetRMS**( )
```
Return the Root Mean Square value of this histogram
```

Stat_t **GetSumOfWeights**( )
```
Return the sum of weights excluding under/overflows
```

void **SetContent**(Stat_t **content*)
```
Replace bin contents by the contents of array content
```

Int_t **GetContour**(Float_t **levels*)
```
Return contour values into array levels
The number of contour levels can be returned by getContourLevel
```

Float_t **GetContourLevel**(Int_t *level*)
```
Return the number of contour levels
```

void **SetContour**(Int_t *nlevels,* Float_t **levels*)
```
Set the number and values of contour levels
By default the number of contour levels is set to 20.

if argument levels = 0 or issing, equidistant contours are computed
```

void **SetContour**(Int_t *nlevels*)
```
Set number of contour levels
Values for the nlevels contours is automatically computed by linear
interpolation between minimum and maximum value.
```

void **SetContourLevel**(Int_t *level,* Float_t *value*)
```
Set value for one contour level
```

Float_t **GetMaximum**( )
```
Return maximum value of all bins
```

Float_t **GetMinimum**( )
```
Return minimum value of all bins
```

void **SetBins**(Int_t *nx,* Float_t *xmin,* Float_t *xmax*)
```
Redefine  x axis parameters
```

void **SetBins**(Int_t *nx,* Float_t *xmin,* Float_t *xmax,* Int_t *ny,* Float_t *ymin,* Float_t *ymax*)
```
Redefine  x and y axis parameters
```

void **SetMaximum**(Float_t *maximum*)
```
Set the maximum value for the Y axis
By default the maximum value is automatically set to the maximum
bin content plyus a margin of 10 per cent.
```

void **SetMinimum**(Float_t *minimum*)

```
Set the minimum value for the Y axis
By default the maximum value is automatically set to zero if all bin contents
are positive or the minimum - 10 per cent otherwise.
```

void **SetDirectory**(TDirectory **dir*)

```
Remove reference to this histogram from current directory and add
reference to new directory dir. dir can be 0 in which case the
histogram does not belong to any directory.
```

void **SetError**(Stat_t **error*)

```
Replace bin errors by values in array error
```

void **SetName**(const Text_t **name*)

```
Change the name of this histogram
```

void **Sumw2**( )

```
Create structure to store sum of squares of weights
if histogram is already filled, the sum of squares of weights
is filled with the existing bin contents
```

TF1* **GetFunction**(const Text_t **name*)

```
Return pointer to function with name
```

Stat_t **GetBinError**(Int_t *bin*)

```
Return value of error associated to bin number bin
if the sum of squares of weights has been defined (via Sumw2),
this function returns the sqrt(sum of w2).
otherwise it returns the sqrt(contents) for this bin.
```

void **SetBinContent**(Int_t *bin,* Stat_t *content*)

Author: René Brun 26/12/94
Last update: 08/11/97 10.42.50 by René Brun

TH1C

class **TH1C** : public TH1, public TArrayC

protected:

class TH1C	**TH1C**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1C	**TH1C**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)

public:

class TH1C	**TH1C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1C	**TH1C**( )
class TH1C	**TH1C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)
class TH1C	**TH1C**(class TH1C& *h1c*)
void	**~TH1C**( )
virtual void	**AddBinContent**(Int_t *bin,* Stat_t *w*)
virtual void	**AddBinContent**(Int_t *bin*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual class TClass*	**IsA**( )
class TH1C&	***operator=***(class TH1C& *h1*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetBinContent**(Int_t *bin,* Stat_t *content*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TH2C, TH3C

Class Description

```
The H I S T O G R A M   Classes
ROOT supports the following histogram types:

1-D histograms:
   TH1C : histograms with one byte per channel. Maximum bin content = 255
   TH1S : histograms with one short per channel. Maximum bin content = 65535
   TH1F : histograms with one float per channel. Maximum precision 7 digits
   TH1D : histograms with one double per channel. Maximum precision 14 digits

2-D histograms:
   TH2C : histograms with one byte per channel. Maximum bin content = 255
   TH2S : histograms with one short per channel. Maximum bin content = 65535
   TH2F : histograms with one float per channel. Maximum precision 7 digits
   TH2D : histograms with one double per channel. Maximum precision 14 digits

3-D histograms:
   TH3C : histograms with one byte per channel. Maximum bin content = 255
   TH3S : histograms with one short per channel. Maximum bin content = 65535
   TH3F : histograms with one float per channel. Maximum precision 7 digits
   TH3D : histograms with one double per channel. Maximum precision 14 digits

Profile histograms: See class TProfile

All histogram classes are derived from the base class TH1
The TH*C classes also inherit from the array class TArrayC.
The TH*S classes also inherit from the array class TArrayS.
```

```
The TH*F classes also inherit from the array class TArrayF.
The TH*D classes also inherit from the array class TArrayD.
```

TH1C(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*)

```
Create a 1-Dim histogram with fix bins of type char (one byte per channel)
                 (see TH1::TH1 for explanation of parameters)
```

TH1C(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t **xbins*)

```
Create a 1-Dim histogram with variable bins of type char (one byte per channel)
                 (see TH1::TH1 for explanation of parameters)
```

void **Reset**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 08/11/97 10.42.50 by René Brun

TH1D

class **TH1D** : public TH1, public TArrayD

protected:

class TH1D	**TH1D**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1D	**TH1D**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)

public:

class TH1D	**TH1D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1D	**TH1D**( )
class TH1D	**TH1D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)
class TH1D	**TH1D**(class TH1D& *h1d*)
void	**~TH1D**( )
virtual void	**AddBinContent**(Int_t *bin,* Stat_t *w*)
virtual void	**AddBinContent**(Int_t *bin*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual class TClass*	**IsA**( )
class TH1D&	***operator=***(class TH1D& *h1*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetBinContent**(Int_t *bin,* Stat_t *content*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TH2D, TH3D, TProfile

Class Description

```
The H I S T O G R A M   Classes
ROOT supports the following histogram types:

1-D histograms:
   TH1C : histograms with one byte per channel. Maximum bin content = 255
   TH1S : histograms with one short per channel. Maximum bin content = 65535
   TH1F : histograms with one float per channel. Maximum precision 7 digits
   TH1D : histograms with one double per channel. Maximum precision 14 digits

2-D histograms:
   TH2C : histograms with one byte per channel. Maximum bin content = 255
   TH2S : histograms with one short per channel. Maximum bin content = 65535
   TH2F : histograms with one float per channel. Maximum precision 7 digits
   TH2D : histograms with one double per channel. Maximum precision 14 digits

3-D histograms:
   TH3C : histograms with one byte per channel. Maximum bin content = 255
   TH3S : histograms with one short per channel. Maximum bin content = 65535
   TH3F : histograms with one float per channel. Maximum precision 7 digits
   TH3D : histograms with one double per channel. Maximum precision 14 digits

Profile histograms: See class TProfile

All histogram classes are derived from the base class TH1
The TH*C classes also inherit from the array class TArrayC.
The TH*S classes also inherit from the array class TArrayS.
```

```
The TH*F classes also inherit from the array class TArrayF.
The TH*D classes also inherit from the array class TArrayD.
```

TH1D(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*):**TArrayD**( ), **TH1**(*name, title, nbins, xlow, xup*)

```
Create a 1-Dim histogram with fix bins of type double
       (see TH1::TH1 for explanation of parameters)
```

TH1D(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t **xbins*):**TArrayD**( ), **TH1**(*name, title, nbins, xbins*)

```
Create a 1-Dim histogram with variable bins of type double
       (see TH1::TH1 for explanation of parameters)
```

void **Reset**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 08/11/97 10.42.50 by René Brun

TH1F

class **TH1F** : public TH1, public TArrayF

protected:

class TH1F	**TH1F**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1F	**TH1F**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)

public:

class TH1F	**TH1F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1F	**TH1F**( )
class TH1F	**TH1F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)
class TH1F	**TH1F**(class TH1F& *h1f*)
void	**~TH1F**( )
virtual void	**AddBinContent**(Int_t *bin,* Stat_t *w*)
virtual void	**AddBinContent**(Int_t *bin*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**FillN**(Int_t *ntimes,* Axis_t* *x,* Double_t* *w*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual class TClass*	**IsA**( )
class TH1F&	***operator=***(class TH1F& *h1*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetBinContent**(Int_t *bin,* Stat_t *content*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TH2F, TH3F

Class Description

```
The H I S T O G R A M   Classes
ROOT supports the following histogram types:

1-D histograms:
   TH1C : histograms with one byte per channel. Maximum bin content = 255
   TH1S : histograms with one short per channel. Maximum bin content = 65535
   TH1F : histograms with one float per channel. Maximum precision 7 digits
   TH1D : histograms with one double per channel. Maximum precision 14 digits

2-D histograms:
   TH2C : histograms with one byte per channel. Maximum bin content = 255
   TH2S : histograms with one short per channel. Maximum bin content = 65535
   TH2F : histograms with one float per channel. Maximum precision 7 digits
   TH2D : histograms with one double per channel. Maximum precision 14 digits

3-D histograms:
   TH3C : histograms with one byte per channel. Maximum bin content = 255
   TH3S : histograms with one short per channel. Maximum bin content = 65535
   TH3F : histograms with one float per channel. Maximum precision 7 digits
   TH3D : histograms with one double per channel. Maximum precision 14 digits

Profile histograms: See class TProfile

All histogram classes are derived from the base class TH1
The TH*C classes also inherit from the array class TArrayC.
```

```
The TH*S classes also inherit from the array class TArrayS.
The TH*F classes also inherit from the array class TArrayF.
The TH*D classes also inherit from the array class TArrayD.
```

TH1F(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*):**TArrayF**( ), **TH1**(*name, title, nbins, xlow, xup*)

```
 Create a 1-Dim histogram with fix bins of type float
        (see TH1::TH1 for explanation of parameters)
```

TH1F(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t **xbins*):**TArrayF**( ), **TH1**(*name, title, nbins, xbins*)

```
 Create a 1-Dim histogram with variable bins of type float
        (see TH1::TH1 for explanation of parameters)
```

void **FillN**(Int_t *ntimes,* Axis_t **x,* Double_t **w*)

```
Fill this TH1F histogram with an array x and weights w
ntimes:  number of entries in arrays x and w
x:       array of values to be histogrammed
w:       array of weights

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w[i]^2 in the bin corresponding to x[i].
if w is NULL each entry is assumed a weight=1
```

void **Reset**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 08/11/97 10.42.50 by René Brun

TH1S

class **TH1S** : public TH1, public TArrayS

protected:

class TH1S	**TH1S**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1S	**TH1S**(Int_t *dim,* Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)

public:

class TH1S	**TH1S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup*)
class TH1S	**TH1S**( )
class TH1S	**TH1S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins*)
class TH1S	**TH1S**(class TH1S& *h1s*)
void	**~TH1S**( )
virtual void	**AddBinContent**(Int_t *bin,* Stat_t *w*)
virtual void	**AddBinContent**(Int_t *bin*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual class TClass*	**IsA**( )
class TH1S&	***operator=***(class TH1S& *h1*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetBinContent**(Int_t *bin,* Stat_t *content*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TH2S, TH3S

Class Description

```
The H I S T O G R A M   Classes
ROOT supports the following histogram types:

1-D histograms:
   TH1C : histograms with one byte per channel. Maximum bin content = 255
   TH1S : histograms with one short per channel. Maximum bin content = 65535
   TH1F : histograms with one float per channel. Maximum precision 7 digits
   TH1D : histograms with one double per channel. Maximum precision 14 digits

2-D histograms:
   TH2C : histograms with one byte per channel. Maximum bin content = 255
   TH2S : histograms with one short per channel. Maximum bin content = 65535
   TH2F : histograms with one float per channel. Maximum precision 7 digits
   TH2D : histograms with one double per channel. Maximum precision 14 digits

3-D histograms:
   TH3C : histograms with one byte per channel. Maximum bin content = 255
   TH3S : histograms with one short per channel. Maximum bin content = 65535
   TH3F : histograms with one float per channel. Maximum precision 7 digits
   TH3D : histograms with one double per channel. Maximum precision 14 digits

Profile histograms: See class TProfile

All histogram classes are derived from the base class TH1
The TH*C classes also inherit from the array class TArrayC.
The TH*S classes also inherit from the array class TArrayS.
```

```
The TH*F classes also inherit from the array class TArrayF.
The TH*D classes also inherit from the array class TArrayD.
```

TH1S(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup*):**TArrayS**( ), **TH1**(*name, title, nbins, xlow, xup*)

```
 Create a 1-Dim histogram with fix bins of type short
        (see TH1::TH1 for explanation of parameters)
```

TH1S(const Text_t **name,* const Text_t **title,* Int_t *nbins,* Axis_t **xbins*):**TArrayS**( ), **TH1**(*name, title, nbins, xbins*)

```
 Create a 1-Dim histogram with variable bins of type short
        (see TH1::TH1 for explanation of parameters)
```

void **Reset**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 08/11/97 10.42.50 by René Brun

TH2

class **TH2**

public:

class TH2	**TH2**(class TH2&)
class TH2	**TH2**( )
void	**~TH2**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TH2& *hnew*)
virtual void	**Fill20**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill20**(Axis_t *x,* Axis_t *y*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Stat_t	**fScalefactor**	*Scale factor*
Stat_t	**fTsumwy**	*Total Sum of weight*Y*
Stat_t	**fTsumwy2**	*Total Sum of weight*Y*Y*
Stat_t	**fTsumwxy**	*Total Sum of weight*X*Y*

See Also: TH2C, TH2D, TH2F, TH2S

Class Description

```
Service class for 2-Dim histogram classes

TH2C a 2-D histogram with one byte per cell (char)
TH2S a 2-D histogram with two bytes per cell (short integer)
TH2F a 2-D histogram with four bytes per cell (float)
TH2D a 2-D histogram with eight bytes per cell (double)
```

void **Fill20**(Axis_t *x,* Axis_t *y,* Stat_t *w*)

Author: René Brun 26/12/94
Last update: 1.02/00 09/07/97 21.15.42 by René Brun

TH2C

class **TH2C** : public TH1C, public TH2

public:

class TH2C	**TH2C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2C	**TH2C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2C	**TH2C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2C	**TH2C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2C	**TH2C**( )
class TH2C	**TH2C**(class TH2C& *h2c*)
void	**~TH2C**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y*)
virtual class TClass*	**IsA**( )
class TH2C&	***operator=***(class TH2C& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
Service class for 2-Dim histogram classes

TH2C a 2-D histogram with one byte per cell (char)
TH2S a 2-D histogram with two bytes per cell (short integer)
TH2F a 2-D histogram with four bytes per cell (float)
TH2D a 2-D histogram with eight bytes per cell (double)
```

void **DrawCopy**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 1.02/00 09/07/97 21.15.42 by René Brun

TH2D

class **TH2D** : public TH1D, public TH2

public:

class TH2D	**TH2D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2D	**TH2D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2D	**TH2D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2D	**TH2D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2D	**TH2D**( )
class TH2D	**TH2D**(class TH2D& *h2d*)
void	**~TH2D**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y*)
virtual class TClass*	**IsA**( )
class TH2D&	***operator=***(class TH2D& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
Service class for 2-Dim histogram classes

TH2C a 2-D histogram with one byte per cell (char)
TH2S a 2-D histogram with two bytes per cell (short integer)
TH2F a 2-D histogram with four bytes per cell (float)
TH2D a 2-D histogram with eight bytes per cell (double)
```

void **DrawCopy**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 1.02/00 09/07/97 21.15.42 by René Brun

TH2F

class **TH2F** : public TH1F, public TH2

public:

class TH2F	**TH2F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2F	**TH2F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2F	**TH2F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2F	**TH2F**( )
class TH2F	**TH2F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2F	**TH2F**(class TH2F& *h2f*)
void	**~TH2F**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y*)
virtual void	**FillN**(Int_t *ntimes,* Axis_t* *x,* Axis_t* *y,* Double_t* *w*)
virtual class TClass*	**IsA**( )
class TH2F&	***operator=***(class TH2F& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
Service class for 2-Dim histogram classes

TH2C a 2-D histogram with one byte per cell (char)
TH2S a 2-D histogram with two bytes per cell (short integer)
TH2F a 2-D histogram with four bytes per cell (float)
TH2D a 2-D histogram with eight bytes per cell (double)
```

void **FillN**(Int_t *ntimes,* Axis_t **x,* Axis_t **y,* Double_t **w*)

```
Fill a 2-D histogram with an array of values and weights
ntimes:  number of entries in arrays x and w
x:       array of x values to be histogrammed
y:       array of y values to be histogrammed
w:       array of weights

If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w[i]^2 in the cell corresponding to x[i],y[i].
if w is NULL each entry is assumed a weight=1
```

Author: René Brun 26/12/94
Last update: 1.02/00 09/07/97 21.15.42 by René Brun

TH2S

class **TH2S** : public TH1S, public TH2

public:

class TH2S	**TH2S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2S	**TH2S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup*)
class TH2S	**TH2S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2S	**TH2S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins*)
class TH2S	**TH2S**( )
class TH2S	**TH2S**(class TH2S& *h2s*)
void	**~TH2S**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y*)
virtual class TClass*	**IsA**( )
class TH2S&	***operator=***(class TH2S& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
Service class for 2-Dim histogram classes

TH2C a 2-D histogram with one byte per cell (char)
TH2S a 2-D histogram with two bytes per cell (short integer)
TH2F a 2-D histogram with four bytes per cell (float)
TH2D a 2-D histogram with eight bytes per cell (double)
```

void **DrawCopy**(Option_t **option*)

Author: René Brun 26/12/94
Last update: 1.02/00 09/07/97 21.15.42 by René Brun

TH3

class **TH3**

public:

class TH3	**TH3**( )
class TH3	**TH3**(class TH3&)
void	**~TH3**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TH3& *hnew*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TH3C, TH3D, TH3F, TH3S

Class Description

```
The 3-D histogram classes derived from the 1-D histogram classes.
all operations are supported (fill, fit).
Drawing is currently restricted to one single option.
A cloud of points is drawn. The number of points is proportional to
cell content.

TH3C a 3-D histogram with one byte per cell (char)
TH3S a 3-D histogram with two bytes per cell (short integer)
TH3F a 3-D histogram with four bytes per cell (float)
TH3D a 3-D histogram with eight bytes per cell (double)
```

void **Copy**(TH3 &*newth3*)

Author: René Brun 27/10/95
Last update: 1.00/07 24/03/97 12.36.35 by René Brun

TH3C

class **TH3C** : public TH1C, public TH3

public:

class TH3C	**TH3C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup,* Int_t *nbinsz,* Axis_t *zlow,* Axis_t *zup*)
class TH3C	**TH3C**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins,* Int_t *nbinsz,* Axis_t* *zbins*)
class TH3C	**TH3C**( )
class TH3C	**TH3C**(class TH3C& *h3c*)
void	**~TH3C**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
class TH3C&	***operator=***(class TH3C& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
The 3-D histogram classes derived from the 1-D histogram classes.
all operations are supported (fill, fit).
Drawing is currently restricted to one single option.
A cloud of points is drawn. The number of points is proportional to
cell content.

TH3C a 3-D histogram with one byte per cell (char)
TH3S a 3-D histogram with two bytes per cell (short integer)
TH3F a 3-D histogram with four bytes per cell (float)
TH3D a 3-D histogram with eight bytes per cell (double)
```

TH3C(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t xup, Int_t nbinsy, Axis_t ylow, Axis_t yup, Int_t nbinsz, Axis_t zlow, Axis_t *zup*):**TH1C**(3, name, title, nbinsx, xlow, *xup*), **TH3**( )

```
Normal constructor for fix bin size 3-D histograms
```

TH3C(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xbins, Int_t nbinsy, Axis_t *ybins, Int_t nbinsz, Axis_t **zbins*):**TH1C**(3, name, title, nbinsx, *xbins*), **TH3**( )

```
Normal constructor for variable bin size 3-D histograms
```

void **Copy**(TObject &*newth3*)

```
Copy this 3-D histogram structure to newth3
```

void **Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)

```
Increment cell defined by x,y,z by a weight w
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y,z.
```

Author: René Brun 27/10/95
Last update: 1.00/07 24/03/97 12.36.35 by René Brun

TH3D

class **TH3D** : public TH1D, public TH3

public:

class TH3D	**TH3D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup,* Int_t *nbinsz,* Axis_t *zlow,* Axis_t *zup*)
class TH3D	**TH3D**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins,* Int_t *nbinsz,* Axis_t* *zbins*)
class TH3D	**TH3D**( )
class TH3D	**TH3D**(class TH3D& *h3d*)
void	**~TH3D**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
class TH3D&	***operator=***(class TH3D& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
The 3-D histogram classes derived from the 1-D histogram classes.
all operations are supported (fill, fit).
Drawing is currently restricted to one single option.
A cloud of points is drawn. The number of points is proportional to
cell content.

TH3C a 3-D histogram with one byte per cell (char)
TH3S a 3-D histogram with two bytes per cell (short integer)
TH3F a 3-D histogram with four bytes per cell (float)
TH3D a 3-D histogram with eight bytes per cell (double)
```

TH3D(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t xup, Int_t nbinsy, Axis_t ylow, Axis_t yup, Int_t nbinsz, Axis_t zlow, Axis_t *zup*):**TH1D**(3, name, title, nbinsx, xlow, *xup*), **TH3**( )

```
Normal constructor for fix bin size 3-D histograms
```

TH3D(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xbins, Int_t nbinsy, Axis_t *ybins, Int_t nbinsz, Axis_t **zbins*):**TH1D**(3, name, title, nbinsx, *xbins*), **TH3**( )

```
Normal constructor for variable bin size 3-D histograms
```

void **Copy**(TObject &*newth3*)

```
Copy this 3-D histogram structure to newth3
```

void **Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)

```
Increment cell defined by x,y,z by a weight w
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y,z.
```

Author: René Brun 27/10/95
Last update: 1.00/07 24/03/97 12.36.35 by René Brun

TH3F

class **TH3F** : public TH1F, public TH3

public:

class TH3F	**TH3F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup,* Int_t *nbinsz,* Axis_t *zlow,* Axis_t *zup*)
class TH3F	**TH3F**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins,* Int_t *nbinsz,* Axis_t* *zbins*)
class TH3F	**TH3F**( )
class TH3F	**TH3F**(class TH3F& *h3f*)
void	**~TH3F**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
class TH3F&	***operator=***(class TH3F& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
The 3-D histogram classes derived from the 1-D histogram classes.
all operations are supported (fill, fit).
Drawing is currently restricted to one single option.
A cloud of points is drawn. The number of points is proportional to
cell content.

TH3C a 3-D histogram with one byte per cell (char)
TH3S a 3-D histogram with two bytes per cell (short integer)
TH3F a 3-D histogram with four bytes per cell (float)
TH3D a 3-D histogram with eight bytes per cell (double)
```

TH3F(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t xup, Int_t nbinsy, Axis_t ylow, Axis_t yup, Int_t nbinsz, Axis_t zlow, Axis_t *zup*):**TH1F**(3, name, title, nbinsx, xlow, *xup*), **TH3**( )

```
Normal constructor for fix bin size 3-D histograms
```

TH3F(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xbins, Int_t nbinsy, Axis_t *ybins, Int_t nbinsz, Axis_t **zbins*):**TH1F**(3, name, title, nbinsx, *xbins*), **TH3**( )

```
Normal constructor for variable bin size 3-D histograms
```

void **Copy**(TObject &*newth3*)

```
Copy this 3-D histogram structure to newth3
```

void **Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)

```
Increment cell defined by x,y,z by a weight w
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y,z.
```

void **Sizeof3D**( ) const

```
Return total size of this 3-D shape with its attributes
```

Author: René Brun 27/10/95
Last update: 1.00/07 24/03/97 12.36.35 by René Brun

TH3S

class **TH3S** : public TH1S, public TH3

public:

class TH3S	**TH3S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Int_t *nbinsy,* Axis_t *ylow,* Axis_t *yup,* Int_t *nbinsz,* Axis_t *zlow,* Axis_t *zup*)
class TH3S	**TH3S**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Int_t *nbinsy,* Axis_t* *ybins,* Int_t *nbinsz,* Axis_t* *zbins*)
class TH3S	**TH3S**( )
class TH3S	**TH3S**(class TH3S& *h3s*)
void	**~TH3S**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
class TH3S&	***operator=***(class TH3S& *h1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
The 3-D histogram classes derived from the 1-D histogram classes.
all operations are supported (fill, fit).
Drawing is currently restricted to one single option.
A cloud of points is drawn. The number of points is proportional to
cell content.

TH3C a 3-D histogram with one byte per cell (char)
TH3S a 3-D histogram with two bytes per cell (short integer)
TH3F a 3-D histogram with four bytes per cell (float)
TH3D a 3-D histogram with eight bytes per cell (double)
```

TH3S(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t xup, Int_t nbinsy, Axis_t ylow, Axis_t yup, Int_t nbinsz, Axis_t zlow, Axis_t *zup*):**TH1S**(3, name, title, nbinsx, xlow, *xup*), **TH3**( )

```
Normal constructor for fix bin size 3-D histograms
```

TH3S(const Text_t **name,* const Text_t **title,* Int_t *nbinsx,* Axis_t *xbins, Int_t nbinsy, Axis_t *ybins, Int_t nbinsz, Axis_t **zbins*):**TH1S**(3, name, title, nbinsx, *xbins*), **TH3**( )

```
Normal constructor for variable bin size 3-D histograms
```

void **Copy**(TObject &*newth3*)

```
Copy this 3-D histogram structure to newth3
```

void **Fill3**(Axis_t *x,* Axis_t *y,* Axis_t *z,* Stat_t *w*)

```
Increment cell defined by x,y,z by a weight w
If the storage of the sum of squares of weights has been triggered,
via the function Sumw2, then the sum of the squares of weights is incremented
by w^2 in the cell corresponding to x,y,z.
```

Author: René Brun 27/10/95
Last update: 1.00/07 24/03/97 12.36.35 by René Brun

THashList

class **THashList** : public TList

public:

class THashList	**THashList**(Int_t *capacity* = TCollection::*kInitHashTableCapacity,* Int_t *rehash = 0*)
class THashList	**THashList**(class TObject* *parent,* Int_t *capacity* = TCollection::*kInitHashTableCapacity,* Int_t *rehash = 0*)
class THashList	**THashList**(class THashList&)
void	**~THashList**( )
void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
void	**AddAt**(class TObject* *obj,* Int_t *idx*)
void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
void	**AddFirst**(class TObject* *obj*)
void	**AddFirst**(class TObject* *obj,* Option_t* *opt*)
void	**AddLast**(class TObject* *obj*)
void	**AddLast**(class TObject* *obj,* Option_t* *opt*)
Float_t	**AverageCollisions**( )
class TClass*	**Class**( )
void	**Clear**(Option_t* *option*)
void	**Delete**(Option_t* *option*)
class TObject*	**FindObject**(Text_t* *name*)
class TObject*	**FindObject**(class TObject* *obj*)
virtual class TClass*	**IsA**( )
void	**Rehash**(Int_t *newCapacity*)
class TObject*	**Remove**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

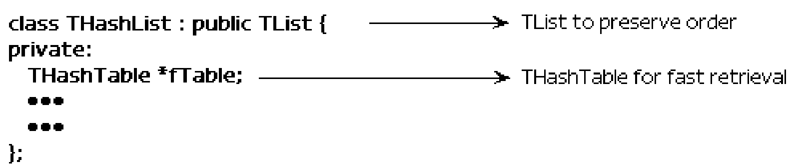
Data Members

protected:

class THashTable*	**fTable**	*Hashtable used for quick lookup of objects*

Class Description

```
THashList

THashList implements a hybrid collection class consisting of a
hash table and a list to store TObject's. The hash table is used for
quick access and lookup of objects while the list allows the objects
to be ordered. The hash value is calculated using the value returned
by the TObject's Hash() function. Each class inheriting from TObject
can override Hash() as it sees fit.
```



THashList(Int_t *capacity,* Int_t *rehash*)

```
Create a THashList object. Capacity is the initial hashtable capacity
(i.e. number of slots), by default kInitHashTableCapacity = 17, and
rehash is the value at which a rehash will be triggered. I.e. when the
average size of the linked lists at a slot becomes longer than rehash
then the hashtable will be resized and refilled to reduce the collision
```

```
rate to about 1. The higher the collision rate, i.e. the longer the
linked lists, the longer lookup will take. If rehash=0 the table will
NOT automatically be rehashed. Use Rehash() for manual rehashing.
```

THashList(TObject **parent,* Int_t *capacity,* Int_t *rehash*): **TList**(*parent*)

```
Create a THashList object. Capacity is the initial hashtable capacity
(i.e. number of slots), by default kInitHashTableCapacity = 17, and
rehash is the value at which a rehash will be triggered. I.e. when the
average size of the linked lists at a slot becomes longer than rehash
then the hashtable will be resized and refilled to reduce the collision
rate to about 1. The higher the collision rate, i.e. the longer the
linked lists, the longer lookup will take. If rehash=0 the table will
NOT automatically be rehashed. Use Rehash() for manual rehashing.
```

~THashList( )

```
Delete a hashlist. All objects will be removed from the list before the
list will be deleted.
```

void **AddFirst**(TObject **obj*)

```
Add object at the beginning of the list.
```

void **AddFirst**(TObject **obj,* const Option_t **opt*)

```
Add object at the beginning of the list and also store option.
Storing an option is useful when one wants to change the behaviour
of an object a little without having to create a complete new
copy of the object. This feature is used, for example, by the Draw()
method. It allows the same object to be drawn in different ways.
```

void **AddLast**(TObject **obj*)

```
Add object at the end of the list.
```

void **AddLast**(TObject **obj,* const Option_t **opt*)

```
Add object at the end of the list and also store option.
Storing an option is useful when one wants to change the behaviour
of an object a little without having to create a complete new
copy of the object. This feature is used, for example, by the Draw()
method. It allows the same object to be drawn in different ways.
```

void **AddBefore**(TObject **before,* TObject **obj*)

```
Insert object before object before in the list.
```

void **AddAfter**(TObject **after,* TObject **obj*)

```
Insert object after object after in the list.
```

void **AddAt**(TObject **obj,* Int_t *idx*)

```
Insert object at location idx in the list.
```

Float_t **AverageCollisions**( ) const

```
Return the average collision rate. The higher the number the longer
the linked lists in the hashtable, the slower the lookup. If the number
is high, or lookup noticeably too slow, perfrom a Rehash().
```

void **Clear**(Option_t **option*)

```
Remove all objects from the list. Does not delete the objects.
```

void **Delete**(Option_t **option*)

```
Remove all objects from the list AND delete all heap based objects.
```

TObject* **FindObject**(const Text_t **name*) const

```
Find object using its name. Uses the hash value returned by the
TString::Hash() after converting name to a TString.
```

TObject* **FindObject**(TObject **obj*) const

```
Find object using its hash value (returned by its Hash() member).
```

void **Rehash**(Int_t *newCapacity*)

```
Rehash the hashlist. If the collision rate becomes too high (i.e.
the average size of the linked lists become too long) then lookup
efficiency decreases since relatively long lists have to be searched
every time. To improve performance rehash the hashtable. This resizes
the table to newCapacity slots and refills the table. Use
AverageCollisions() to check if you need to rehash.
```

TObject* **Remove**(TObject **obj*)

```
Remove object from the list.
```

Author: Fons Rademakers 10/08/95
Last update: 1.03/04 29/09/97 19.08.23 by Fons Rademakers

THashTable

class **THashTable** : public TCollection

private:

Int_t	**GetHashValue**(class TObject* *obj*)
Int_t	**GetHashValue**(class TString& *s*)
Int_t	**GetHashValue**(char* *str*)

public:

class THashTable	**THashTable**(Int_t *capacity* = TCollection::*kInitHashTableCapacity,* Int_t *rehash = 0*)
class THashTable	**THashTable**(class THashTable&)
void	**~THashTable**( )
virtual void	**Add**(class TObject* *obj*)
Float_t	**AverageCollisions**( )
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
Int_t	**Collisions**(char* *name*)
Int_t	**Collisions**(class TObject* *obj*)
virtual void	**Delete**(Option_t* *option*)
virtual class TObject*	**FindObject**(class TObject* *obj*)
virtual class TObject*	**FindObject**(Text_t* *name*)
virtual Int_t	**GetSize**( )
virtual class TClass*	**IsA**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir* = *kIterForward*)
void	**Rehash**(Int_t *newCapacity,* Bool_t *checkObjValidity* = *kTRUE*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

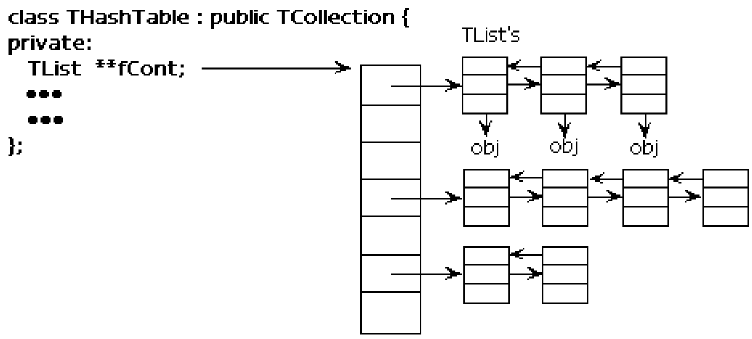
Data Members

private:

class TList**	**fCont**	*Hash table (table of lists)*
Int_t	**fEntries**	*Number of objects in table*
Int_t	**fUsedSlots**	*Number of used slots*
Int_t	**fRehashLevel**	*Average collision rate which triggers rehash*

Class Description

```
THashTable

THashTable implements a hash table to store TObject's. The hash
value is calculated using the value returned by the TObject's
Hash() function. Each class inheriting from TObject can override
Hash() as it sees fit.
THashTable does not preserve the insertion order of the objects.
If the insertion order is important AND fast retrieval is needed
use THashList instead.
```



THashTable(Int_t *capacity,* Int_t *rehashlevel*)

Create a THashTable object. Capacity is the initial hashtable capacity (i.e. number of slots), by default kInitHashTableCapacity = 17, and rehashlevel is the value at which a rehash will be triggered. I.e. when the average size of the linked lists at a slot becomes longer than rehashlevel then the hashtable will be resized and refilled to reduce the collision rate to about 1. The higher the collision rate, i.e. the longer the linked lists, the longer lookup will take. If rehashlevel=0 the table will NOT automatically be rehashed. Use Rehash() for manual rehashing.

~THashTable( )

Delete a hashtable. All objects will be removed from the table before the table will be deleted.

void **Add**(TObject **obj*)

Add object to the hash table. Its position in the table will be determined by the value returned by its Hash() function.

void **Clear**(Option_t **option*)

Remove all objects from the table. Does not delete the objects.

Int_t **Collisions**(const char **name*) const

Returns the number of collisions for an object with a certain name (i.e. number of objects in same slot in the hash table, i.e. length of linked list).

Int_t **Collisions**(TObject **obj*) const

Returns the number of collisions for an object (i.e. number of objects in same slot in the hash table, i.e. length of linked list).

void **Delete**(Option_t **option*)

Remove all objects from the table AND delete all heap based objects.

TObject* **FindObject**(const Text_t **name*) const

Find object using its name. Uses the hash value returned by the TString::Hash() after converting name to a TString.

TObject* **FindObject**(TObject **obj*) const

Find object using its hash value (returned by its Hash() member).

TIterator* **MakeIterator**(Bool_t *dir*) const

Returns a hash table iterator.

void **Rehash**(Int_t *newCapacity,* Bool_t *checkObjValidity*)

Rehash the hashtable. If the collision rate becomes too high (i.e. the average size of the linked lists become too long) then lookup efficiency decreases since relatively long lists have to be searched every time. To improve performance rehash the hashtable. This resizes

```
the table to newCapacity slots and refills the table. Use
AverageCollisions() to check if you need to rehash. Set checkObjValidity
to kFALSE if you know that all objects in the table are still valid
(i.e. have not been deleted from the system in the meanwhile).
```

TObject* **Remove**(TObject **obj*)

```
Remove object from the hashtable.
```

Author: Fons Rademakers 27/09/95
Last update: 1.03/04 29/09/97 19.12.26 by Fons Rademakers

THashTableIter

class **THashTableIter** : public TIterator

private:

class THashTableIter	**THashTableIter**( )
Int_t	**NextSlot**( )

public:

class THashTableIter	**THashTableIter**(class THashTable* *ht,* Bool_t *dir* = *kIterForward*)
class THashTableIter	**THashTableIter**(class THashTableIter&)
void	**~THashTableIter**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
virtual void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

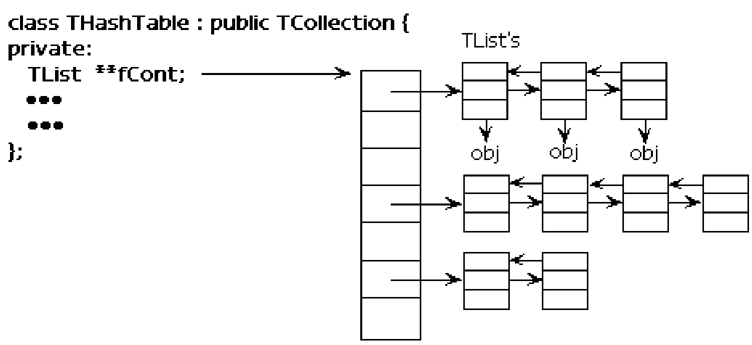
Data Members

private:

const class THashTable*	**fTable**	*hash table being iterated*
Int_t	**fCursor**	*current position in table*
class TListIter*	**fListCursor**	*current position in collision list*
Bool_t	**fDirection**	*iteration direction*

Class Description

```
THashTable

THashTable implements a hash table to store TObject's. The hash
value is calculated using the value returned by the TObject's
Hash() function. Each class inheriting from TObject can override
Hash() as it sees fit.
THashTable does not preserve the insertion order of the objects.
If the insertion order is important AND fast retrieval is needed
use THashList instead.
```



THashTableIter(const THashTable **ht,* Bool_t *dir*)

```
Create a hashtable iterator. By default the iteration direction
is kIterForward. To go backward use kIterBackward.
```

~THashTableIter( )

```
Delete hashtable iterator.
```

TObject* **Next**( )

```
Return next object in hashtable. Returns 0 when no more objects in table.
```

Int_t **NextSlot**( )

```
Returns index of next slot in table containing list to be iterated.
```

void **Reset**( )

```
Reset the hashtable iterator. Either to beginning or end, depending on
the initial iteration direction.
```

Author: Fons Rademakers 27/09/95
Last update: 1.03/04 29/09/97 19.12.26 by Fons Rademakers

THelpDialog

class **THelpDialog** : public TComponent

private:

void	**DestroyCB**(*Widget, XtPointer, XtPointer*)

protected:

class THelpDialog	**THelpDialog**( )

public:

class THelpDialog	**THelpDialog**(char* *name*)
class THelpDialog	**THelpDialog**(class THelpDialog&)
void	**~THelpDialog**( )
void	**AddHelpText**(char* *helpText*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**Post**(class TSimpleWindow* *parent*)
void	**SetHelpText**(char* *helpText*)
void	**SetTitle**(char* *title*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

const char*	**fHelpText**
const char*	**fDialogShellTitle**
Widget	**fHelpShell**
Widget	**fHelpTextWidget**

Class Description

```
THelpDialog

This class handles help text dialogs.
```

void **DestroyCB**(*Widget,* XtPointer *cd, XtPointer*)

Author: Fons Rademakers 01/04/95
Last update: 1.02/00 22/08/97 11.38.49 by Fons Rademakers

THelpPane

class **THelpPane** : public TSubMenu

private:

void	**ClickForHelp**( )
void	**ClickForHelpCallback**(*Widget, XtPointer, XtPointer*)
void	**CreateHelpPane**(MenuDesc_t* *desc,* XtPointer *defaultClientData*)
void	**Index**( )
void	**IndexCallback**(*Widget, XtPointer, XtPointer*)
Widget	**IsParentAShell**( )
void	**Keys**( )
void	**KeysCallback**(*Widget, XtPointer, XtPointer*)
void	**Overview**( )
void	**OverviewCallback**(*Widget, XtPointer, XtPointer*)
void	**ShowVersion**( )
void	**ShowVersionCallback**(*Widget, XtPointer, XtPointer*)

public:

class THelpPane	**THelpPane**(Widget *parent,* char* *name = helpMenu,* MenuDesc_t* *desc = NULL,* XtPointer *defaultClientData = NULL*)
class THelpPane	**THelpPane**(class THelpPane&)
class THelpPane	**THelpPane**(char* *name = helpMenu,* MenuDesc_t* *desc = NULL,* XtPointer *defaultClientData = NULL*)
void	**~THelpPane**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Cursor	**fHelpCursor**
static const String	**fgDefaultResources**

Class Description

```
THelpPane

This class puts help menu items to the application.
```

~THelpPane( )

```
empty
```

void **KeysCallback**(*Widget,* XtPointer *clientData, XtPointer*)

Author: Fons Rademakers 02/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

THtml

class **THtml** : public TObject

private:

void	**Class2Html**(class TClass* *classPtr,* Bool_t *force = kFALSE*)
void	**ClassDescription**(class ofstream& *out,* class TClass* *classPtr,* Bool_t& *flag*)
void	**ClassTree**(class TVirtualPad* *canvas,* class TClass* *classPtr,* Bool_t *force = kFALSE*)
Bool_t	**CopyHtmlFile**(Char_t* *sourceName,* Char_t* *destName*)
void	**CreateIndex**(Char_t** *classNames,* Int_t *numberOfClasses*)
void	**CreateIndexByTopic**(Char_t** *filenames,* Int_t *numberOfNames,* Int_t *maxLen*)
void	**CreateListOfTypes**( )
void	**DerivedClasses**(class ofstream& *out,* class TClass* *classPtr*)
void	**ExpandKeywords**(class ofstream& *out,* Char_t* *text,* class TClass* *ptr2class,* Bool_t& *flag,* Char_t* *dir*)
void	**ExpandPpLine**(class ofstream& *out,* Char_t* *line*)
const Char_t*	**GetFileName**(Char_t* *filename*)
Char_t*	**GetHtmlFileName**(class TClass* *classPtr*)
Bool_t	**IsModified**(class TClass* *classPtr,* Int_t *type*)
Bool_t	**isname**(Int_t *c*)
Bool_t	**isword**(Int_t *c*)
void	**ReplaceSpecialChars**(class ofstream& *out,* Char_t *c*)
void	**ReplaceSpecialChars**(class ofstream& *out,* Char_t* *string*)
void	**SortNames**(Char_t** *strings,* Int_t *num,* Bool_t *type = 0*)
Char_t*	**StrDup**(Char_t* *s1,* Int_t *n = 1*)
void	**WriteHtmlFooter**(class ofstream& *out,* Char_t* *dir,* Char_t* *lastUpdate,* Char_t* *author,* Char_t* *copyright*)
void	**WriteHtmlHeader**(class ofstream& *out,* Char_t* *title*)

public:

class THtml	**THtml**( )
class THtml	**THtml**(class THtml&)
void	**~THtml**( )
class TClass*	**Class**( )
void	**Convert**(Text_t* *filename,* Text_t* *title,* Text_t* *dirname*)
const Char_t*	**GetOutputDir**( )
const Char_t*	**GetSourceDir**( )
virtual class TClass*	**IsA**( )
void	**MakeAll**(Bool_t *force = kFALSE*)
void	**MakeClass**(Text_t* *className,* Bool_t *force = kFALSE*)
void	**MakeIndex**( )
void	**MakeTree**(Text_t* *className,* Bool_t *force = kFALSE*)
void	**SetOutputDir**(Char_t* *dir*)
void	**SetSourceDir**(Char_t* *dir*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

const Char_t*	**fSourceDir**	*source path*
const Char_t*	**fOutputDir**	*output directory*
Char_t*	**fLine**	*current line*
Int_t	**fLen**	*maximum line length*
Char_t*	**fCounter**	*counter string*

Class Description

```
The HyperText Markup Language (HTML) is a simple data format used to
create hypertext documents that are portable from one platform to another.
HTML documents are SGML documents with generic semantics that are
appropriate for representing information from a wide range of domains.

The THtml class is designed to provide an easy way for converting ROOT
classes, and files as well, into HTML documents. Here is the few rules
and suggestions for a configuration, coding and usage.

Configuration:
-------------

The output directory could be specified using the Root.Html.OutputDir
environment variable ( default value: "html/" ). Also it is necessary to
define Root.Html.SourceDir to point to directories containing .cxx and .h
files ( see: TEnv ).

    Examples:
             Root.Html.OutputDir: html
             Root.Html.SourceDir: src:include:.:/usr/user/source
             Root.Html.Root:      http://root.cern.ch/root/html

During the conversion, THtml will look for the certain number of
user defined strings, i.e. author's name, copyright note, etc.
This could be defined with following environment variables:

    Root.Html.Author     ( default: //*-- Author : )
    Root.Html.LastUpdate ( default: //*CMZ : )
    Root.Html.Copyright  ( default: // Copyright )

Coding rules:
------------

A class description block, which must be placed before the first
member function, has a following form:

    //
    // TMyClass
    //
    // This is the description block.
    //

The environment variable Root.Html.Description ( see: TEnv ) contents
the delimiter string ( default value: //_________________ ). It means
that you can also write your class description block like this:

    //_____________________________________________________________
    // A description of the class starts with the line above, and
    // will take place here !

Note that EVERYTHING until the first non-commented line is considered
as a valid class description block.

A member function description block starts immediately after '{'
and looks like this:

    void TWorld::HelloWorldFunc( string *text )
    {
    // This is an example of description for the
    // TWorld member function

       helloWorld.Print( text );
```

```
   }

Like in a class description block, EVERYTHING until the first
non-commented line is considered as a valid member function
description block.

Usage:
-----

  Root> gHtml.MakeAll                // invoke a make for all classes
  Root> gHtml.MakeClass( TMyClass ) // create a HTML files for that class only
  Root> gHtml.MakeIndex()           // creates an index files only
  Root> gHtml.MakeTree( TMyClass )  // creates an inheritance tree for a class

  Root> gHtml.Convert( hist1.mac, "Histogram example" )

Environment variables:
---------------------

Root.Html.OutputDir    ( default: html/ )
Root.Html.SourceDir    ( default: src/ )
Root.Html.Author       ( default: //*-- Author : )
Root.Html.LastUpdate   ( default: //*CMZ : )
Root.Html.Copyright    ( default: // Copyright )
Root.Html.Description  ( default: //____________________ )
Root.Html.HomePage     ( URL to the user defined home page )
Root.Html.SearchEngine ( link to the search engine )
```

THtml( )

```
Create a THtml object. Use object directly or via the global
pointer gHtml. In case output directory does not exist an error
will be printed and gHtml stays 0 also zombie bit will be set.
```

~**THtml**( )

```
Default destructor
```

void **Class2Html**(TClass **classPtr,* Bool_t *force*)

```
It creates HTML file for a single class

Input: classPtr - pointer to the class
```

void **ClassDescription**(ofstream &*out,* TClass **classPtr,* Bool_t &*flag*)

```
This function builds the description of the class

Input: out      - output file stream
     classPtr - pointer to the class
     flag     - this is a '/' flag
```

void **ClassTree**(TVirtualPad **psCanvas,* TClass **classPtr,* Bool_t *force*)

```
It makes a class tree

Input: psCanvas - pointer to the current canvas
     classPtr - pointer to the class
```

void **Convert**(Text_t **filename,* Text_t **title,* Text_t **dirname*)

```
It converts a single text file to HTML

Input: filename - name of the file to convert
     title    - title which will be placed at the top of the HTML file
     dirname  - optional parameter, if it's not specified, output will
                be placed in html/examples directory.

NOTE: Output file name is the same as filename, but with extension .html
```

Bool_t **CopyHtmlFile**(const Char_t **sourceName,* const Char_t **destName*)

```
Copy file to HTML directory

Input: sourceName - source file name
      destName   - optional destination name, if not
                   specified it would be the same
                   as the source file name

Output: TRUE if file is successfully copied, or
      FALSE if it's not

NOTE: The destination directory is always fOutputDir
```

void **CreateIndex**(const Char_t ***classNames,* Int_t *numberOfClasses*)

```
Create an index

Input: classNames      - pointer to an array of class names
     numberOfClasses - number of elements
```

void **CreateIndexByTopic**(Char_t ***fileNames,* Int_t *numberOfNames,* Int_t *maxLen*)

```
It creates several index files

Input: fileNames     - pointer to an array of file names
     numberOfNames - number of elements in the fileNames array
     maxLen        - maximum length of a single name
```

void **CreateListOfTypes**( )

```
Create list of all data types
```

void **DerivedClasses**(ofstream *&out,* TClass **classPtr*)

```
It creates a list of derived classes

Input: out      - output file stream
     classPtr - pointer to the class
```

void **ExpandKeywords**(ofstream *&out,* Char_t **text,* TClass **ptr2class,*Bool_t &flag, Char_t **dir*)

```
Find keywords in text & create URLs

Input: out       - output file stream
     text      - pointer to the array of the characters to process
     ptr2class - pointer to the class
     flag      - this is a 'html_begin/html_end' flag
     dir       - usually "" or "../", depends of current file
                 directory position
```

void **ExpandPpLine**(ofstream *&out,* Char_t **line*)

```
Expand preprocessor statements

Input: out  - output file stream
     line - pointer to the array of characters,
            usually one line from the source file

NOTE: Looks for the #include statements and
     creates link to the corresponding file
     if such file exists
```

const Char_t* **GetFileName**(const Char_t **filename*)

```
It discards any directory information inside filename

Input: filename - pointer to the file name

Output: pointer to the string containing just a file name
      without any other directory information, i.e.
```

```
'/usr/root/test.dat' will return 'test.dat'
```

Char_t* **GetHtmlFileName**(TClass **classPtr*)

```
Return real HTML filename

Input: classPtr - pointer to a class

Output: pointer to the string containing a full name
      of the corresponding HTML file
```

Bool_t **IsModified**(TClass **classPtr,* const Int_t *type*)

```
Check if file is modified

Input: classPtr - pointer to the class
      type     - file type to compare with
                 values: kSource, kInclude, kTree

Output: TRUE     - if file is modified since last time
      FALSE    - if file is up to date
```

Bool_t **isname**(Int_t *c*)

```
Check if c is a valid C++ name character

Input: c - a single character

Output: TRUE if c is a valid C++ name character
      and FALSE if it's not.

NOTE: Valid name characters are [a..zA..Z0..9_],
```

Bool_t **isword**(Int_t *c*)

```
Check if c is a valid first character for C++ name

Input: c - a single character

Output: TRUE if c is a valid first character for C++ name,
      and FALSE if it's not.

NOTE: Valid first characters are [a..zA..Z_]
```

void **MakeAll**(Bool_t *force*)

```
It makes everything
```

void **MakeClass**(Text_t **className,* Bool_t *force*)

```
Make HTML files for a single class

Input: className - name of the class to process
```

void **MakeIndex**( )

```
It makes an index files
```

void **MakeTree**(Text_t **className,* Bool_t *force*)

```
Make an inheritance tree

Input: className - name of the class to process
```

void **ReplaceSpecialChars**(ofstream &*out,* const Char_t *c*)

```
Replace ampersand, less-than and greater-than character

Input: out - output file stream
     c   - single character
```

void **ReplaceSpecialChars**(ofstream &*out,* const Char_t **string*)

```
Replace ampersand, less-than and greater-than characters

Input: out    - output file stream
     string - pointer to an array of characters
```

void **SortNames**(const Char_t ***strings,* Int_t *num,* Bool_t *type*)

```
Sort strings

Input: strings - pointer to an array of strings
     type    - sort type
               values : kCaseInsensitive, kCaseSensitive
               default: kCaseInsensitive
```

Char_t* **StrDup**(const Char_t **s1,* Int_t *n*)

```
Returns a pointer to a new string which is a duplicate
of the string to which 's1' points.  The space for the
new string is obtained using the 'new' operator. The new
string has the length of 'strlen(s1) + n'.
```

void **WriteHtmlHeader**(ofstream *&out,* const Char_t **title*)

```
Write HTML header

Input: out   - output file stream
     title - title for the HTML page
```

void **WriteHtmlFooter**(ofstream *&out,* Char_t **dir,* const Char_t **lastUpdate,*Char_t *author, Char_t **copyright*)

```
Write HTML footer

Input: out        - output file stream
     dir        - usually equal to "" or "../", depends of
                  current file directory position, i.e. if
                  file is in the fOutputDir, then dir will be ""
     lastUpdate - last update string
     author     - author's name
     copyright  - copyright note
```

Author: Nenad Buncic 18/10/95
Last update: 06/11/97 18.11.32 by Fons Rademakers

TIcon

class **TIcon** : public TNamed

private:

class TIcon	**TIcon**( )

public:

class TIcon	**TIcon**(char* *name,* struct IconDesc_t* *desc,* Bool_t *autoInstall* = *kTRUE*)
class TIcon	**TIcon**(class TIcon&)
void	**~TIcon**( )
class TClass*	**Class**( )
Char_t*	**GetIconBits**((*unknown*) EIconType t)
UInt_t	**GetIconHeight**((*unknown*) EIconType t)
UInt_t	**GetIconWidth**((*unknown*) EIconType t)
Bool_t	**Install**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TIconImp*	**fImp**	*TIcon implementation*

protected:

struct IconDesc_t*	**fDesc**	*Icon description*

Class Description

```
TIcon
```

Bool_t **Install**( )

Author: Nenad Buncic 26/11/96
Last update: 1.00/06 23/03/97 11.17.22 by René Brun

TIconImp

class **TIconImp**

public:

class TIconImp	**TIconImp**(class TIcon* *i = 0*)
class TIconImp	**TIconImp**(class TIconImp&)
void	**~TIconImp**( )
class TClass*	**Class**( )
virtual Bool_t	**Install**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TIcon*	**fIcon**	*TIcon associated with this implementation*

See Also: TMotifIcon

Class Description

```
ABC describing GUI independent icon
```

Author: Nenad Buncic 28/11/96
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TInetAddress

class **TInetAddress** : public TObject

private:

class TInetAddress	**TInetAddress**( )
class TInetAddress	**TInetAddress**(char* *host,* UInt_t *addr,* Int_t *family,* Int_t *port = -1*)

public:

class TInetAddress	**TInetAddress**(class TInetAddress& *adr*)
void	**~TInetAddress**( )
class TClass*	**Class**( )
UInt_t	**GetAddress**( )
UChar_t*	**GetAddressBytes**( )
Int_t	**GetFamily**( )
const char*	**GetHostAddress**( )
const char*	**GetHostName**( )
Int_t	**GetPort**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsValid**( )
class TInetAddress&	***operator*=**(class TInetAddress& *rhs*)
void	**Print**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TString	**fHostname**	*full qualified hostname*
UInt_t	**fAddress**	*IP address in host byte order*
Int_t	**fFamily**	*address family*
Int_t	**fPort**	*port through which we are connected*

Class Description

```
TInetAddress

This class represents an Internet Protocol (IP) address.
```

TInetAddress( )

```
Private default ctor. Used in case of unknown host. Not a valid address.
```

TInetAddress(const char **host,* UInt_t *addr,* Int_t *family,* Int_t *port*)

```
Create TInetAddress. Private ctor. TInetAddress objects can only
be created via the friend classes TSystem, TServerSocket and TSocket.
Use the IsValid() method to check the validity of a TInetAddress.
```

TInetAddress(const TInetAddress *&adr*)

```
TInetAddress copy ctor.
```

UChar_t* **GetAddressBytes**( ) const

```
Returns the raw IP address in host byte order. The highest
order byte position is in addr[0]. To be prepared for 64-bit
IP addresses an array of bytes is returned.
```

const char* **GetHostAddress**( ) const

```
Returns the IP address string "%d.%d.%d.%d".
Copy string immediately, it will be reused.
```

void **Print**(Option_t *)

```
Print internet address as string.
```

Author: Fons Rademakers 16/12/96
Last update: 1.03/05 09/10/97 13.33.32 by Fons Rademakers

TInfoDialog

class **TInfoDialog** : public TDialogManager

protected:

class TInfoDialog	**TInfoDialog**( )

public:

class TInfoDialog	**TInfoDialog**(char* *name*)
class TInfoDialog	**TInfoDialog**(class TInfoDialog&)
void	**~TInfoDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TInfoDialog

This class handles info dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TInput

class **TInput** : public TCallbackObject

private:

void	**InputHandler**(XtPointer *clientData,* int* *source,* XtInputId* *id*)

public:

class TInput	**TInput**( )
class TInput	**TInput**(class TInput&)
void	**~TInput**( )
void	**Attach**(int *fd,* XtInputMask *mask*)
class TClass*	**Class**( )
int	**Fd**( )
int	**Id**( )
virtual class TClass*	**IsA**( )
void	**Remove**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

XtInputId	**fId**
int	**fFd**

public:

static const char *const	**fgInputCallback**

Class Description

```
TInput

This class encapsulates input callback from file descriptors.
```

void **InputHandler**(XtPointer *clientData,* int *, XtInputId *)

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TInterpreter

class **TInterpreter** : public TNamed

public:

void	**~TInterpreter**( )
virtual Int_t	**Calc**(char* *line*)
class TClass*	**Class**( )
virtual void	**CreateListOfBaseClasses**(class TClass* *cl*)
virtual void	**CreateListOfDataMembers**(class TClass* *cl*)
virtual void	**CreateListOfMethodArgs**(class TFunction* *m*)
virtual void	**CreateListOfMethods**(class TClass* *cl*)
virtual void	**CreateListOfTypes**( )
virtual Int_t	**DeleteGlobal**(void* *obj*)
virtual void	**EndOfLineAction**( )
virtual void	**Execute**(Text_t* *function,* Text_t* *params*)
virtual void	**Execute**(class TObject* *obj,* class TClass* *cl,* Text_t* *method,* Text_t* *params*)
virtual void	**Execute**(class TObject* *obj,* class TClass* *cl,* class TMethod* *method,* class TObjArray* *params*)
virtual Int_t	**ExecuteMacro**(char* *filename*)
virtual void*	**GetInterfaceMethod**(class TClass* *cl,* Text_t* *method,* Text_t* *params*)
virtual void*	**GetInterfaceMethodWithPrototype**(class TClass* *cl,* Text_t* *method,* Text_t* *proto*)
virtual Int_t	**GetMore**( )
virtual char*	**GetPrompt**( )
virtual Int_t	**InitializeDictionaries**( )
virtual class TClass*	**IsA**( )
virtual void	**LoadMacro**(char* *filename*)
virtual void	**PrintIntro**( )
virtual Int_t	**ProcessLine**(char* *line*)
virtual Int_t	**ProcessLineSynch**(char* *line*)
virtual void	**Reset**( )
virtual void	**ResetAll**( )
virtual void	**ResetGlobals**( )
virtual void	**SaveContext**( )
virtual void	**SaveGlobalsContext**( )
virtual void	**SetClassInfo**(class TClass* *cl*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual const char*	**TypeName**(char* *s*)
virtual void	**UpdateListOfGlobalFunctions**( )
virtual void	**UpdateListOfGlobals**( )

Data Members

See Also: TCint

Class Description

```
TInterpreter

This class defines an abstract interface to a generic command line
interpreter.
```

Author: Fons Rademakers 01/03/96
Last update: 1.00/06 23/03/97 11.32.07 by René Brun

TIter

class **TIter**

protected:

class TIter	**TIter**( )

public:

class TIter	**TIter**(class TCollection* *col,* Bool_t *dir* = *kIterForward*)
class TIter	**TIter**(class TIterator* *it*)
class TIter	**TIter**(class TIter&)
void	**~TIter**( )
class TClass*	**Class**( )
const Option_t*	**GetOption**( )
virtual class TClass*	**IsA**( )
class TObject*	**Next**( )
class TObject*	**operator( )**( )
void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TIterator*	**fIterator**	*collection iterator*

Class Description

```
Collection abstract base class. This class describes the base
protocol all collection classes have to implement. The ROOT
collection classes always store pointers to objects that inherit
from TObject. They never adopt the objects. Therefore, it is the
user's responsability to take care of deleting the actual objects
once they are not needed anymore. In exceptional cases, when the
user is 100% sure nothing else is referencing the objects in the
collection, one can delete all objects and the collection at the
same time using the Delete() function.

Collections can be iterated using an iterator object (see
TIterator). Depending on the concrete collection class there may be
some additional methods of iterating. See the repective classes.

TCollection inherits from TObject since we want to be able to have
collections of collections.

In a later release the collections may become templatized.
```

Author: Fons Rademakers 13/08/95
Last update: 1.03/04 30/09/97 18.04.38 by Fons Rademakers

TIterator

class **TIterator**

public:

void	**~TIterator**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual const Option_t*	**GetOption**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
class TObject*	**operator( )**( )
virtual void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TBtreeIter, THashTableIter, TListIter, TMapIter, TObjArrayIter, TOrdCollectionIter

Class Description

```
TIterator

Iterator abstract base class. This base class provides the interface
for collection iterators.
```

Author: Fons Rademakers 13/08/95
Last update: 1.00/06 23/03/97 11.05.27 by René Brun

TKey

class **TKey** : public TNamed

protected:

virtual void	**Create**(Int_t *nbytes*)

public:

class TKey	**TKey**(Text_t* *name,* Text_t* *title,* class TClass* *cl,* Int_t *nbytes*)
class TKey	**TKey**(class TString& *name,* class TString& *title,* class TClass* *cl,* Int_t *nbytes*)
class TKey	**TKey**(class TObject* *obj,* Text_t* *name,* Int_t *bufsize*)
class TKey	**TKey**( )
class TKey	**TKey**(Seek_t *pointer,* Int_t *nbytes*)
class TKey	**TKey**(class TKey&)
void	**~TKey**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Delete**(Option_t* *option*)
virtual void	**DeleteBuffer**( )
virtual void	**FillBuffer**(char*& *buffer*)
virtual char*	**GetBuffer**( )
class TBuffer*	**GetBufferRef**( )
virtual const Text_t*	**GetClassName**( )
Short_t	**GetCycle**( )
Short_t	**GetKeep**( )
Int_t	**GetKeylen**( )
Int_t	**GetNbytes**( )
Int_t	**GetObjlen**( )
virtual Seek_t	**GetSeekKey**( )
virtual Seek_t	**GetSeekPdir**( )
Int_t	**GetVersion**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual void	**Keep**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**Read**(class TObject* *obj*)
virtual void*	**Read**(Option_t* *option*)
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**ReadFile**( )
virtual void	**SetBuffer**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**WriteFile**(Int_t *cycle = 1*)

Data Members

protected:

Int_t	**fVersion**	*Key version identifier*
Int_t	**fNbytes**	*Number of bytes for the object on file*
Int_t	**fObjlen**	*Length of uncompressed object in bytes*
class TDatime	**fDatime**	*Date/Time of insertion in file*
Short_t	**fKeylen**	*Number of bytes for the key itself*
Short_t	**fCycle**	*Cycle number*
Seek_t	**fSeekKey**	*Location of object on file*
Seek_t	**fSeekPdir**	*Location of parent directory on file*

class TString	**fClassName**	*Object Class name*
Int_t	**fLeft**	*Number of bytes left in current segment*
char*	**fBuffer**	*Object buffer*
class TBuffer*	**fBufferRef**	*Pointer to the TBuffer object*

See Also: TBasket

Class Description

```
The TKey class includes functions to book space on a file,
to create I/O buffers, to fill these buffers
to compress/uncompress data buffers.

Before saving (making persistent) an object on a file, a key must
be created. The key structure contains all the information to
uniquely identify a persistent object on a file.
  fNbytes    = number of bytes for the compressed object+key
  fObjlen    = Length of uncompressed object
  fDatime    = Date/Time when the object was written
  fKeylen    = number of bytes for the key structure
  fCycle     = cycle number of the object
  fSeekKey   = Address of the object on file (points to fNbytes)
               This is a redundant information used to cross-check
               the data base integrity.
  fSeekPdir  = Pointer to the directory supporting this object
  fClassName = Object class name
  fName      = name of the object
  fTitle     = title of the object

The TKey class is used by ROOT to:
 - to write an object in the Current Directory
 - to write a new ntuple buffer

The structure of a file is shown in TFile::TFile
The structure of a directory is shown in TDirectory::TDirectory
The TKey class is used by the TBasket class
  See also TTree
```

TKey( ) : **TNamed**( )

```
TKey default constructor
```

TKey(Seek_t *pointer,* Int_t *nbytes*) : **TNamed**( )

```
Create a TKey object to read keys
Constructor called by TDirectory::ReadKeys and by TFile::TFile
A TKey object is created to read the keys structure itself
```

TKey(TObject **obj,* const Text_t **name,* const Int_t *bufsize*): **TNamed**(name, obj->**GetTitle**( ))

```
Create a TKey object and fill output buffer
```

void **Create**(Int_t *nbytes*)

```
Create a TKey object
```

~TKey( )

```
TKey default destructor
```

void **Delete**(Option_t **option*)

```
Delete an object from the file
```

void **DeleteBuffer**( )

```
Delete key buffer(s)
```

Short_t **GetCycle**( )

```
Return cycle number associated to this key
```

Short_t **GetKeep**( )
```
Returns the "KEEP" status
```

void **FillBuffer**(char *&*buffer*)
```
Encode key header into output buffer
```

void **Keep**( )
```
Set the "KEEP" status
When the KEEP flag is set to 1 the object cannot be purged
```

void **ls**(Option_t **option*)
```
List Key contents
```

void **Print**(Option_t **option*)
```
Print key contents
```

void* **Read**(Option_t **option*)
```
To read an object from the file
The object associated to this key is read from the file into memory
Once the key structure is read (via Streamer) the class identifier
of the object is known.
Using the class identifier we find the TClass object for this class.
A TClass object contains a full description (i.e. dictionary) of the
associated class. In particular the TClass object can create a new
object of the class type it describes. This new object now calls its
Streamer function to rebuilt itself.
```

void **Read**(TObject **obj*)
```
To read an object from the file
The object associated to this key is read from the file into memory
Before invoking this function, obj has been created via the
default constructor.
```

void **ReadBuffer**(char *&*buffer*)
```
Decode input buffer
```

void **ReadFile**( )
```
Read the key structure from the file
```

Int_t **Sizeof**( ) const
```
Return the size in bytes of the key header structure
```

void **Streamer**(TBuffer &*b*)
```
Stream a class object
```

void **WriteFile**(Int_t *cycle*)
```
Write the encoded object supported by this key
```

Author: René Brun 28/12/94
Last update: 1.03/04 29/09/97 15.38.11 by René Brun

TKeyMapFile

class **TKeyMapFile** : public TNamed

public:

class TKeyMapFile	**TKeyMapFile**( )
class TKeyMapFile	**TKeyMapFile**(Text_t* *name,* Text_t* *classname,* class TMapFile* *mapfile*)
class TKeyMapFile	**TKeyMapFile**(class TKeyMapFile&)
void	**~TKeyMapFile**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TMapFile*	**fMapFile**	*pointer to map file*

Class Description

```
Utility class for browsing TMapFile objects.
When the browser is invoked for a TMapFile, a TKeyMapFile object
is created for each object in the mapped file.
When a TKeyMapFile object is clicked in the browser, a new copy
of this object is copied into the local directory and the action
corresponding to object->Browse is executed (typically Draw).
```

void **Browse**(TBrowser **b*)

Author: René Brun 23/07/97
Last update: 1.02/00 23/07/97 18.41.34 by René Brun

TLeaf

class **TLeaf** : public TNamed

public:

class TLeaf	**TLeaf**( )
class TLeaf	**TLeaf**(Text_t* *name,* Text_t* *type*)
class TLeaf	**TLeaf**(class TLeaf&)
void	**~TLeaf**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
class TBranch*	**GetBranch**( )
virtual class TLeaf*	**GetLeafCount**( )
virtual class TLeaf*	**GetLeafCounter**(Int_t& *countval*)
virtual Int_t	**GetLen**( )
virtual Int_t	**GetLenStatic**( )
virtual Int_t	**GetLenType**( )
virtual Int_t	**GetMaximum**( )
virtual Int_t	**GetMinimum**( )
virtual Int_t	**GetNdata**( )
virtual Int_t	**GetOffset**( )
virtual const Text_t*	**GetTypeName**( )
virtual Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsRange**( )
virtual Bool_t	**IsUnsigned**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**SetBranch**(class TBranch* *branch*)
virtual void	**SetLen**(Int_t *len = 1*)
virtual void	**SetOffset**(Int_t *offset = 0*)
virtual void	**SetRange**(Bool_t *range = kTRUE*)
virtual void	**SetUnsigned**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fLen**	*Number of fixed length elements*
Int_t	**fLenType**	*Number of bytes for this data type*
Int_t	**fNdata**	*Number of elements in fAddress data buffer*
Int_t	**fOffset**	*Offset in ClonesArray object (if one)*
class TLeaf*	**fLeafCount**	*Pointer to Leaf count if variable length*
Bool_t	**fIsRange**	*(=kTRUE if leaf has a range, kFALSE otherwise)*
Bool_t	**fIsUnsigned**	*(=kTRUE if unsigned, kFALSE otherwise)*
class TBranch*	**fBranch**	*Pointer to supporting branch*

See Also: TLeafB, TLeafC, TLeafD, TLeafF, TLeafI, TLeafObject, TLeafS

Class Description

```
A TLeaf describes individual elements of a TBranch
```

```
    See TBranch structure in TTree.
```

TLeaf( ): **TNamed**( )

```
Default constructor for Leaf
```

TLeaf(const Text_t **name,* const Text_t **type*):**TNamed**(*name, name*)

```
Create a Leaf
  See the TTree and TBranch constructors for explanation of parameters.
```

~**TLeaf**( )

```
Default destructor for a Leaf
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

TLeaf* **GetLeafCounter**(Int_t &*countval*)

```
Return Pointer to counter of referenced Leaf
If leaf name has the forme var[nelem], where nelem is alphanumeric, then
  If nelem is a leaf name, return countval = 0 and the pointer to leaf.
If leaf name has the forme var[nelem], where nelem is a digit, then
  return countval = nelemr and a null pointer.
Otherwise return countval=1 and a null pointer.
```

Int_t **GetLen**( ) const

```
Return the number of effective elements of this leaf
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 12/01/96
Last update: 1.02/00 21/08/97 08.39.48 by René Brun

TLeafB

class **TLeafB** : public TLeaf

public:

class TLeafB	**TLeafB**(Text_t* *name,* Text_t* *type*)
class TLeafB	**TLeafB**( )
class TLeafB	**TLeafB**(class TLeafB&)
void	**~TLeafB**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Char_t	**fMinimum**	*Minimum value if leaf range is specified*
Char_t	**fMaximum**	*Maximum value if leaf range is specified*
Char_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for an 8 bit Integer data type.
```

TLeafB( ): **TLeaf**( )

```
Default constructor for LeafB
```

TLeafB(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafB
```

~TLeafB( )

```
Default destructor for a LeafB
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

const Text_t* **GetTypeName**( ) const

```
Returns name of leaf type
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 12/01/96
Last update: 1.01/04 02/06/97 12.32.44 by René Brun

TLeafC

class **TLeafC** : public TLeaf

public:

class TLeafC	**TLeafC**(Text_t* *name,* Text_t* *type*)
class TLeafC	**TLeafC**( )
class TLeafC	**TLeafC**(class TLeafC&)
void	**~TLeafC**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
virtual Int_t	**GetMaximum**( )
virtual Int_t	**GetMinimum**( )
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
char*	**GetValueString**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fMinimum**	*Minimum value if leaf range is specified*
Int_t	**fMaximum**	*Maximum value if leaf range is specified*
Char_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for a variable length string.
```

TLeafC( ): **TLeaf**( )

```
Default constructor for LeafC
```

TLeafC(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafC
```

~TLeafC( )

```
Default destructor for a LeafC
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

const Text_t* **GetTypeName**( ) const

```
Returns name of leaf type
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 17/03/97
Last update: 1.01/04 02/06/97 12.33.25 by René Brun

TLeafD

class **TLeafD** : public TLeaf

public:

class TLeafD	**TLeafD**(Text_t* *name,* Text_t* *type*)
class TLeafD	**TLeafD**( )
class TLeafD	**TLeafD**(class TLeafD&)
void	**~TLeafD**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Double_t	**fMinimum**	*Minimum value if leaf range is specified*
Double_t	**fMaximum**	*Maximum value if leaf range is specified*
Double_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for a 64 bit floating point data type.
```

TLeafD( ): **TLeaf**( )

```
Default constructor for LeafD
```

TLeafD(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafD
```

~TLeafD( )

```
Default destructor for a LeafD
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 12/01/96
Last update: 1.01/04 02/06/97 12.31.14 by René Brun

TLeafF

class **TLeafF** : public TLeaf

public:

class TLeafF	**TLeafF**(Text_t* *name,* Text_t* *type*)
class TLeafF	**TLeafF**( )
class TLeafF	**TLeafF**(class TLeafF&)
void	**~TLeafF**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fMinimum**	*Minimum value if leaf range is specified*
Float_t	**fMaximum**	*Maximum value if leaf range is specified*
Float_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for a 32 bit floating point data type.
```

TLeafF( ): **TLeaf**( )

```
Default constructor for LeafF
```

TLeafF(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafF
```

~TLeafF( )

```
Default destructor for a LeafF
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 12/01/96
Last update: 1.01/04 02/06/97 11.24.03 by René Brun

TLeafI

class **TLeafI** : public TLeaf

public:

class TLeafI	**TLeafI**(Text_t* *name,* Text_t* *type*)
class TLeafI	**TLeafI**( )
class TLeafI	**TLeafI**(class TLeafI&)
void	**~TLeafI**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
virtual Int_t	**GetMaximum**( )
virtual Int_t	**GetMinimum**( )
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**SetMaximum**(Int_t *max*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fMinimum**	*Minimum value if leaf range is specified*
Int_t	**fMaximum**	*Maximum value if leaf range is specified*
Int_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for an Integer data type.
```

TLeafI( ): **TLeaf**( )

```
Default constructor for LeafI
```

TLeafI(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafI
```

~TLeafI( )

```
Default destructor for a LeafI
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

const Text_t* **GetTypeName**( ) const

```
Returns name of leaf type
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 12/01/96
Last update: 1.02/00 18/07/97 12.17.20 by René Brun

TLeafObject

class **TLeafObject** : public TLeaf

public:

class TLeafObject	**TLeafObject**(Text_t* *name,* Text_t* *type*)
class TLeafObject	**TLeafObject**( )
class TLeafObject	**TLeafObject**(class TLeafObject&)
void	**~TLeafObject**( )
class TClass*	**Class**( )
virtual void	**FillBasket**(class TBuffer& *b*)
class TClass*	**GetClass**( )
class TMethodCall*	**GetMethodCall**(char* *name*)
class TObject*	**GetObject**( )
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i,* class TTreeFormula* *f = 0*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TClass*	**fClass**	*pointer to class*
void**	**fObjAddress**	*Address of Pointer to object*

Class Description

```
A TLeaf for a general object derived from TObject.
```

TLeafObject( ): **TLeaf**( )

```
Default constructor for LeafObject
```

TLeafObject(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafObject
```

~TLeafObject( )

```
Default destructor for a LeafObject
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

TMethodCall* **GetMethodCall**(char **name*)

```
Returns pointer to method corresponding to name
 name is a string with the general form  "method(list of params)"
If list of params is omitted, () is assumed;
```

const Text_t* **GetTypeName**( ) const

```
Returns name of leaf type
```

Float_t **GetValue**(Int_t *i,* TTreeFormula **f*)

```
Returns result of leaf method
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

void **Streamer**(TBuffer &*b*)

```
Stream an object of class TLeafObject.
```

Author: René Brun 27/01/96
Last update: 07/11/97 12.08.39 by René Brun

TLeafS

class **TLeafS** : public TLeaf

public:

class TLeafS	**TLeafS**(Text_t* *name,* Text_t* *type*)
class TLeafS	**TLeafS**( )
class TLeafS	**TLeafS**(class TLeafS&)
void	**~TLeafS**( )
class TClass*	**Class**( )
virtual void	**Export**(class TClonesArray* *list*)
virtual void	**FillBasket**(class TBuffer& *b*)
const Text_t*	**GetTypeName**( )
Float_t	**GetValue**(Int_t *i = 0,* class TTreeFormula* *f = 0*)
virtual void*	**GetValuePointer**( )
virtual void	**Import**(class TClonesArray* *list,* Int_t *n*)
virtual class TClass*	**IsA**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBasket**(class TBuffer& *b*)
virtual void	**SetAddress**(void* *add = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Short_t	**fMinimum**	*Minimum value if leaf range is specified*
Short_t	**fMaximum**	*Maximum value if leaf range is specified*
Short_t*	**fValue**	*Pointer to data buffer*

Class Description

```
A TLeaf for a 16 bit Integer data type.
```

TLeafS( ): **TLeaf**( )

```
Default constructor for LeafS
```

TLeafS(const Text_t **name,* const Text_t **type*):**TLeaf**(*name, type*)

```
Create a LeafS
```

~TLeafS( )

```
Default destructor for a LeafS
```

void **Export**(TClonesArray **list*)

```
Export element from local leaf buffer to ClonesArray
```

void **FillBasket**(TBuffer &*b*)

```
Pack leaf elements in Basket output buffer
```

const Text_t* **GetTypeName**( ) const

```
Returns name of leaf type
```

void **Import**(TClonesArray **list,* Int_t *n*)

```
Import element from ClonesArray into local leaf buffer
```

void **Print**(Option_t **option*)

```
Print a description of this leaf
```

void **ReadBasket**(TBuffer &*b*)

```
Read leaf elements from Basket input buffer
```

void **SetAddress**(void **add*)

```
Set leaf buffer data address
```

Author: René Brun 12/01/96
Last update: 1.01/04 02/06/97 12.32.04 by René Brun

TLego

class **TLego** : public TObject, public TAttLine, public TAttFill

public:

class TLego	**TLego**(Float_t* *rmin,* Float_t* *rmax,* Int_t *system = 1*)
class TLego	**TLego**(class TLego&)
class TLego	**TLego**( )
void	**~TLego**( )
void	**BackBox**(Float_t *ang*)
class TClass*	**Class**( )
void	**ClearRaster**( )
void	**ColorFunction**(Int_t *nl,* Float_t* *fl,* Int_t* *icl,* Int_t& *irep*)
void	**DrawFaceMode1**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *t*)
void	**DrawFaceMode2**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *t*)
void	**DrawFaceMode3**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *t*)
void	**DrawFaceMove1**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *tt*)
void	**DrawFaceMove2**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *tt*)
void	**DrawFaceRaster1**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *tt*)
void	**DrawFaceRaster2**(Int_t* *icodes,* Float_t* *xyz,* Int_t *np,* Int_t* *iface,* Float_t* *tt*)
void	**FillPolygon**(Int_t *n,* Float_t* *p,* Float_t* *f*)
void	**FillPolygonBorder**(Int_t *nn,* Float_t* *xy*)
void	**FindLevelLines**(Int_t *np,* Float_t* *f,* Float_t* *t*)
void	**FindPartEdge**(Float_t* *p1,* Float_t* *p2,* Float_t *f1,* Float_t *f2,* Float_t *fmin,* Float_t *fmax,* Int_t& *kpp,* Float_t* *pp*)
void	**FindVisibleDraw**(Float_t* *r1,* Float_t* *r2*)
void	**FindVisibleLine**(Float_t* *p1,* Float_t* *p2,* Int_t *ntmax,* Int_t& *nt,* Float_t* *t*)
void	**FrontBox**(Float_t *ang*)
void	**GouraudFunction**(Int_t *ia,* Int_t *ib,* Float_t* *f,* Float_t* *t*)
void	**InitMoveScreen**(Float_t *xmin,* Float_t *xmax*)
void	**InitRaster**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Int_t *nx,* Int_t *ny*)
virtual class TClass*	**IsA**( )
void	**LegoCartesian**(Float_t *ang,* Int_t *nx,* Int_t *ny,* char* *chopt*)
void	**LegoCylindrical**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)
void	**LegoFunction**(Int_t *ia,* Int_t *ib,* Int_t& *nv,* Float_t* *ab,* Float_t* *vv,* Float_t* *t*)
void	**LegoPolar**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)
void	**LegoSpherical**(Int_t *ipsdr,* Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)
void	**LightSource**(Int_t *nl,* Float_t *yl,* Float_t *xscr,* Float_t *yscr,* Float_t *zscr,* Int_t& *irep*)
void	**Luminosity**(Float_t* *anorm,* Float_t& *flum*)
void	**ModifyScreen**(Float_t* *r1,* Float_t* *r2*)
void	**SetColorDark**(Color_t *color,* Int_t *n = 0*)
void	**SetColorMain**(Color_t *color,* Int_t *n = 0*)
void	**SetDrawFace**(DrawFaceFunc_t *pointer*)
void	**SetLegoFunction**(LegoFunc_t *pointer*)
void	**SetMesh**(Int_t *mesh = 1*)
void	**SetSurfaceFunction**(SurfaceFunc_t *pointer*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)

void	**SideVisibilityDecode**(Float_t *val,* Int_t& *iv1,* Int_t& *iv2,* Int_t& *iv3,* Int_t& *iv4,* Int_t& *iv5,* Int_t& *iv6,* Int_t& *ir*)
void	**SideVisibilityEncode**(Int_t *iopt,* Float_t *phi1,* Float_t *phi2,* Float_t& *val*)
void	**Spectrum**(Int_t *nl,* Float_t *fmin,* Float_t *fmax,* Int_t *ic,* Int_t *idc,* Int_t& *irep*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**SurfaceCartesian**(Float_t *ang,* Int_t *nx,* Int_t *ny,* char* *chopt*)
void	**SurfaceCylindrical**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)
void	**SurfaceFunction**(Int_t *ia,* Int_t *ib,* Float_t* *f,* Float_t* *t*)
void	**SurfacePolar**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)
void	**SurfaceProperty**(Float_t *qqa,* Float_t *qqd,* Float_t *qqs,* Int_t *nnqs,* Int_t& *irep*)
void	**SurfaceSpherical**(Int_t *ipsdr,* Int_t *iordr,* Int_t *na,* Int_t *nb,* char* *chopt*)

Data Members

private:

Int_t	**fSystem**	*Coordinate system*
Int_t	**fNT**	
Float_t	**fX0**	
Float_t	**fDX**	
Float_t	**fRmin**	*Lower limits of lego*
Float_t	**fRmax**	*Upper limits of lego*
Float_t	**fU**	
Float_t	**fD**	
Float_t	**fT**	
Int_t	**fNlevel**	*Number of color levels*
Float_t	**fFunLevel**	*Function levels corresponding to colors*
Int_t	**fColorLevel**	*Color levels corresponding to functions*
Int_t	**fColorMain**	
Int_t	**fColorDark**	
Int_t	**fColorTop**	
Int_t	**fColorBottom**	
Int_t	**fMesh**	(*=1 if mesh to draw, o otherwise*)
Int_t	**fNlines**	
Int_t	**fLevelLine**	
Float_t	**fPlines**	
Float_t	**fAphi**	
Int_t	**fLoff**	
Float_t	**fYdl**	
Float_t	**fYls**	
Float_t	**fVls**	
Float_t	**fQA**	
Float_t	**fQD**	
Float_t	**fQS**	
Int_t	**fNqs**	
Int_t	**fNxrast**	
Int_t	**fNyrast**	
Float_t	**fXrast**	
Float_t	**fYrast**	
Float_t	**fDXrast**	
Float_t	**fDYrast**	
Int_t	**fIfrast**	
Int_t*	**fRaster**	*pointer to raster buffer*
Int_t	**fJmask**	
Int_t	**fMask**	
DrawFaceFunc_t	**fDrawFace**	*pointer to face drawing function*
LegoFunc_t	**fLegoFunction**	*pointer to lego function*
SurfaceFunc_t	**fSurfaceFunction**	*pointer to surface function*

Class Description

```
Legos and Surfaces package
This package was originally written by Evgueni Tcherniaev
from IHEP/Protvino.

The original Fortran implementation was adapted to HIGZ/PAW
by Olivier Couet and  Evgueni Tcherniaev.

This View class is a subset of the original system
It has been converted to a C++ class  by René Brun
```

TLego( ): **TObject**( ), **TAttLine**(*1, 1, 1*), **TAttFill**(*1,* 0)

```
Lego default constructor
```

TLego(Float_t **rmin,* Float_t **rmax,* Int_t *system*): **TObject**( ), **TAttLine**(*1, 1, 1*), **TAttFill**(*1,* 0)

```
Normal default constructor
rmin[3], rmax[3] are the limits of the lego object depending on
the selected coordinate system
```

~TLego( )

```
Lego default destructor
```

void **BackBox**(Float_t *ang*)

```
Draw back surfaces of surrounding box
 Input  ANG     - angle between X and Y axis

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
          NP        - number of nodes in face
          IFACE(NP) - face
          T(NP)     - additional function
```

void **ClearRaster**( )

```
Clear screen
```

void **ColorFunction**(Int_t *nl,* Float_t **fl,* Int_t **icl,* Int_t *&irep*)

```
Set correspondance between function and color levels
 Input: NL        - number of levels
        FL(NL)    - function levels
        ICL(NL+1) - colors for levels

 Output: IREP     - reply: 0 O.K.
                          -1 error in parameters:
                      illegal number of levels
                      function levels must be in increasing order
                      negative color index
```

void **DrawFaceMode1**(Int_t **icodes,* Float_t **xyz,* Int_t *np,* Int_t **iface,* Float_t **t*)

```
Draw face - 1st variant
 Function: Draw face - 1st variant
          (2 colors: 1st for external surface, 2nd for internal)

 References: WCtoNDC

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
        XYZ(3,*)  - coordinates of nodes
        NP        - number of nodes
        IFACE(NP) - face
        T(NP)     - additional function defined on this face
                    (not used in this routine)
```

void **DrawFaceMode2**(Int_t **icodes,* Float_t **xyz,* Int_t *np,* Int_t **iface,* Float_t **t*)

```
Draw face - 2nd option
 Function: Draw face - 2nd option
           (fill in correspondance with function levels)

 References: WCtoNDC, FillPolygon

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
        XYZ(3,*)  - coordinates of nodes
        NP        - number of nodes
        IFACE(NP) - face
        T(NP)     - additional function defined on this face
```

void **DrawFaceMode3**(Int_t **icodes,* Float_t **xyz,* Int_t *np,* Int_t **iface,* Float_t **t*)

```
Draw face - 3rd option
 Function: Draw face - 3rd option
           (draw face for stacked lego plot)

 References: WCtoNDC

 Input: ICODES(*) - set of codes for the line
          ICODES(1) - IX coordinate of the line cell
          ICODES(2) - IY coordinate of the line cell
          ICODES(3) - lego number
          ICODES(4) - side: 1-face,2-right,3-back,4-left,
                            5-bottom, 6-top
          XYZ(3,*)  - coordinates of nodes
          NP        - number of nodes
          IFACE(NP) - face
          T(*)      - additional function (not used here)
```

void **DrawFaceMove1**(Int_t **icodes,* Float_t **xyz,* Int_t *np,* Int_t **iface,* Float_t **tt*)

```
Draw face - 1st variant for "MOVING SCREEN" algorithm
 Function: Draw face - 1st variant for "MOVING SCREEN" algorithm
           (draw face with level lines)

 References: FindLevelLines, WCtoNDC,
             FindVisibleDraw, ModifyScreen

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
        XYZ(3,*)  - coordinates of nodes
        NP        - number of nodes
        IFACE(NP) - face
        TT(NP)    - additional function defined on this face
                    (not used in this routine)
```

void **DrawFaceMove2**(Int_t **icodes,* Float_t **xyz,* Int_t *np,* Int_t **iface,* Float_t **tt*)

```
Draw face - 2nd variant for "MOVING SCREEN" algorithm
 Function: Draw face - 2nd variant for "MOVING SCREEN" algorithm
           (draw face for stacked lego plot)

 References: FindLevelLines, WCtoNDC,
             FindVisibleDraw, ModifyScreen

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
          ICODES(3) - line code (N of lego)
        XYZ(3,*)  - coordinates of nodes
```

```
NP        - number of nodes
IFACE(NP) - face
TT(NP)    - additional function defined on this face
            (not used in this routine)
```

void **DrawFaceRaster1**(Int_t **icodes*, Float_t **xyz*, Int_t *np*, Int_t **iface*, Float_t **tt*)

```
Draw face - 1st variant for "RASTER SCREEN" algorithm
 Function: Draw face - 1st variant for "RASTER SCREEN" algorithm
           (draw face with level lines)

 References: FindLevelLines, WCtoNDC,
             FindVisibleLine, FillPolygonBorder

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
        XYZ(3,*)  - coordinates of nodes
        NP        - number of nodes
        IFACE(NP) - face
        TT(NP)    - additional function defined on this face
                    (not used in this routine)
```

void **DrawFaceRaster2**(Int_t **icodes*, Float_t **xyz*, Int_t *np*, Int_t **iface*, Float_t **tt*)

```
Draw face - 2nd variant for "RASTER SCREEN" algorithm
 Function: Draw face - 2nd variant for "RASTER SCREEN" algorithm
           (draw face for stacked lego plot)

 References: WCtoNDC, FindVisibleLine, FillPolygonBorder

 Input: ICODES(*) - set of codes for the line (not used)
          ICODES(1) - IX
          ICODES(2) - IY
          ICODES(3) - line code (N of lego)
        XYZ(3,*)  - coordinates of nodes
        NP        - number of nodes
        IFACE(NP) - face
        TT(NP)    - additional function defined on this face
                    (not used in this routine)
```

void **FillPolygon**(Int_t *n*, Float_t **p*, Float_t **f*)

```
Fill polygon with function values at vertexes
 Input: N      - number of vertexes
        P(3,*) - polygon
        F(*)   - function values at nodes

 Errors: - illegal number of vertexes in polygon
         - illegal call of FillPolygon: no levels
```

void **FillPolygonBorder**(Int_t *nn*, Float_t **xy*)

```
Fill a polygon including border ("RASTER SCREEN")
 Input: NN      - number of polygon nodes
        XY(2,*) - polygon nodes
```

void **FindLevelLines**(Int_t *np*, Float_t **f*, Float_t **t*)

```
Find level lines for face
 Input: NP      - number of nodes
        F(3,NP) - face
        T(NP)   - additional function

 Error: number of points for line not equal 2
```

void **FindPartEdge**(Float_t **p1*, Float_t **p2*, Float_t *f1*, Float_t *f2*, Float_t *fmin*, Float_t *fmax*, Int_t &*kpp*, Float_t **pp*)

```
Find part of edge
 Function: Find part of edge where function defined on this edge
           has value from FMIN to FMAX

 Input: P1(3) - 1st point
        P2(3) - 2nd point
        F1    - function value at 1st point
        F2    - function value at 2nd point
        FMIN  - min value of layer
        FMAX  - max value of layer

 Output: KPP - current number of point
         PP(3,*) - coordinates of new face
```

void **FindVisibleDraw**(Float_t **r1*, Float_t **r2*)

```
Find visible parts of line (draw line)
 Input: R1(3)  - 1-st point of the line
        R2(3)  - 2-nd point of the line
```

void **FindVisibleLine**(Float_t **p1*, Float_t **p2*, Int_t *ntmax*, Int_t &*nt*, Float_t **t*)

```
Find visible part of a line ("RASTER SCREEN")
 Input: P1(2) - 1st point of the line
        P2(2) - 2nd point of the line
        NTMAX - max allowed number of visible segments

 Output: NT     - number of visible segments of the line
         T(2,*) - visible segments
```

void **FrontBox**(Float_t *ang*)

```
Draw forward faces of surrounding box & axes
 Function: Draw forward faces of surrounding box & axes

 References: AxisVertex, Gaxis

 Input  ANG     - angle between X and Y axis

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
          NP        - number of nodes in face
          IFACE(NP) - face
          T(NP)     - additional function
```

void **GouraudFunction**(Int_t *ia*, Int_t *ib*, Float_t **face*, Float_t **t*)

```
Find part of surface with luminosity in the corners
           This routine is used for Gouraud shading
```

void **InitMoveScreen**(Float_t *xmin*, Float_t *xmax*)

```
Initialize "MOVING SCREEN" method
 Input: XMIN - left boundary
        XMAX - right boundary
```

void **InitRaster**(Float_t *xmin*, Float_t *ymin*, Float_t *xmax*, Float_t *ymax*, Int_t *nx*, Int_t *ny*)

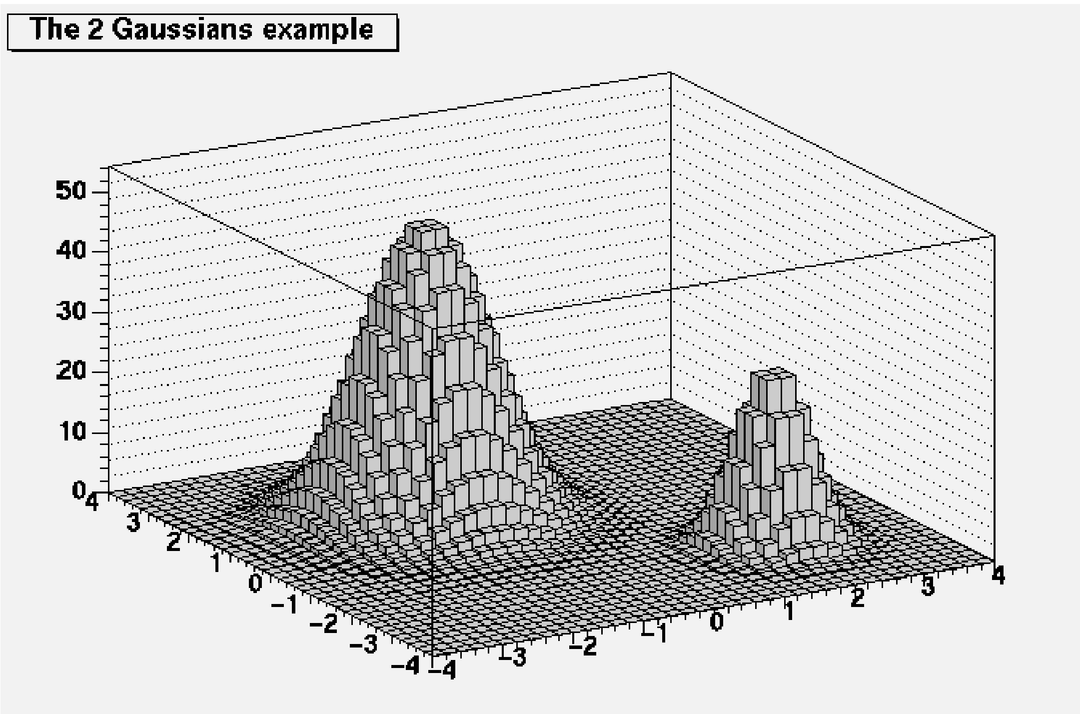
```
Initialize hidden lines removal algorithm (RASTER SCREEN)
 Input: XMIN - Xmin in the normalized coordinate system
        YMIN - Ymin in the normalized coordinate system
        XMAX - Xmax in the normalized coordinate system
        YMAX - Ymax in the normalized coordinate system
        NX   - number of pixels along X
        NY   - number of pixels along Y
```

void **LegoFunction**(Int_t *ia*, Int_t *ib*, Int_t &*nv*, Float_t **ab*, Float_t **vv*, Float_t **t*)

```
Service function for Legos
```

void **LegoCartesian**(Float_t *ang,* Int_t *nx,* Int_t *ny,* char **chopt*)

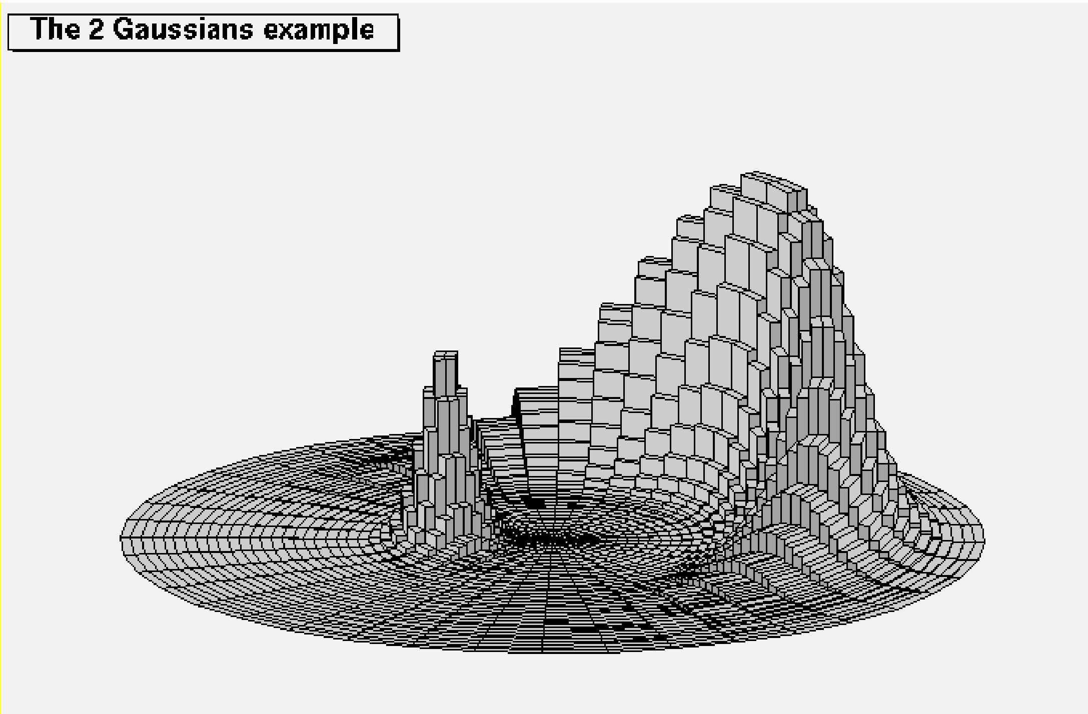
```
Draw stack of lego-plots in cartesian coordinates
 Input: ANG      - angle between X ang Y
        NX       - number of cells along X
        NY       - number of cells along Y

        FUN(IX,IY,NV,XY,V,T) - external routine
          IX     - X number of the cell
          IY     - Y number of the cell
          NV     - number of values for given cell
          XY(2,4)- coordinates of the cell corners
          V(NV)  - cell values
          T(4,NV)- additional function (for example: temperature)

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this line
            ICODES(1) - IX
            ICODES(2) - IY
            ICODES(3) - IV
            ICODES(4) - side: 1-face,2-right,3-back,4-left,
                              5-bottom, 6-top
            XYZ(3,*)  - coordinates of nodes
            NP        - number of nodes
            IFACE(NP) - face
             T(4)   - additional function (here Z-coordinate)

        CHOPT - options: 'BF' - from BACK to FRONT
                         'FB' - from FRONT to BACK
```



void **LegoPolar**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char **chopt*)

```
Draw stack of lego-plots in polar coordinates
 Input: IORDR - order of variables (0 - R,PHI; 1 - PHI,R)
        NA    - number of steps along 1st variable
        NB    - number of steps along 2nd variable

        FUN(IA,IB,NV,AB,V,TT) - external routine
          IA      - cell number for 1st variable
```

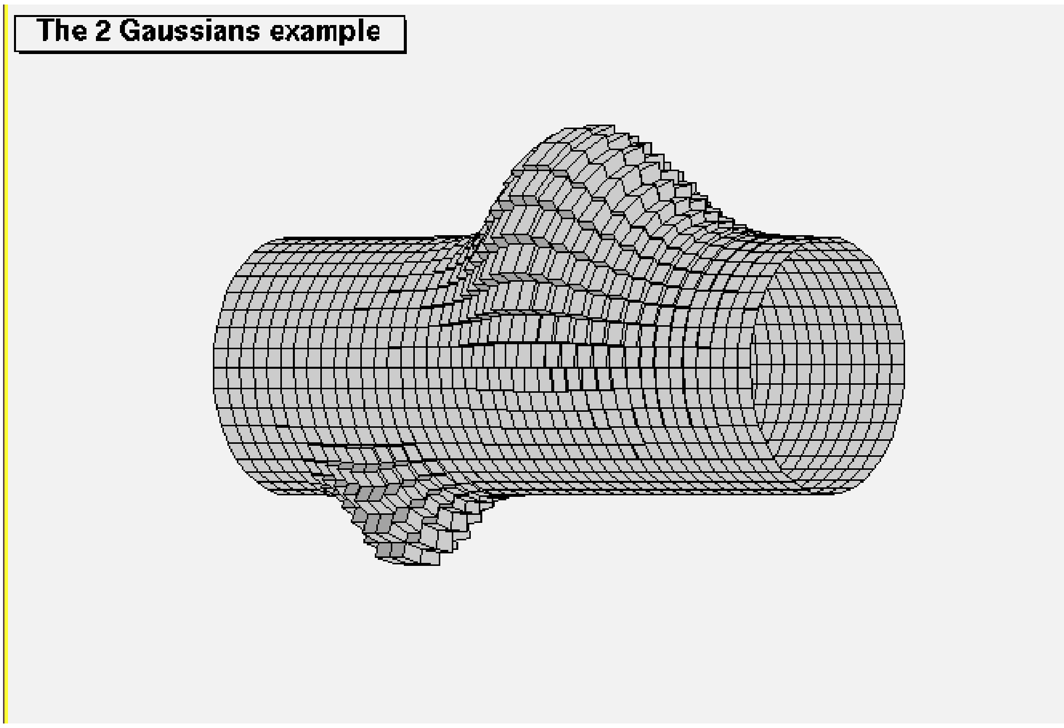
```
  IB      - cell number for 2nd variable
  NV      - number of values for given cell
  AB(2,4) - coordinates of the cell corners
  V(NV)   - cell values
  TT(4,*) - additional function

DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
  ICODES(*) - set of codes for this face
    ICODES(1) - IA
    ICODES(2) - IB
    ICODES(3) - IV
    ICODES(4) - side: 1-internal,2-right,3-external,4-left
                      5-bottom, 6-top
  XYZ(3,*)  - coordinates of nodes
  NP        - number of nodes in face
  IFACE(NP) - face
  T(NP)     - additional function

 CHOPT       - options: 'BF' - from BACK to FRONT
                        'FB' - from FRONT to BACK
```



void **LegoCylindrical**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char **chopt*)

```
Draw stack of lego-plots in cylindrical coordinates
 Input: IORDR - order of variables (0 - Z,PHI; 1 - PHI,Z)
        NA    - number of steps along 1st variable
        NPHI  - number of steps along 2nd variable

        FUN(IA,IB,NV,AB,V,TT) - external routine
          IA      - cell number for 1st variable
          IB      - cell number for 2nd variable
          NV      - number of values for given cell
          AB(2,4) - coordinates of the cell corners
          V(NV)   - cell values
          TT(4,*) - additional function

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
```

```
    ICODES(1) - IA
    ICODES(2) - IB
    ICODES(3) - IV
    ICODES(4) - side: 1,2,3,4 - ordinary sides
                      5-bottom,6-top
  XYZ(3,*)  - coordinates of nodes
  NP        - number of nodes in face
  IFACE(NP) - face
  T(NP)     - additional function

CHOPT       - options: 'BF' - from BACK to FRONT
                       'FB' - from FRONT to BACK
```



void **LegoSpherical**(Int_t *ipsdr,* Int_t *iordr,* Int_t *na,* Int_t *nb,* char **chopt*)

```
Draw stack of lego-plots spheric coordinates
 Input: IPSDR - pseudo-rapidity flag
        IORDR - order of variables (0 - THETA,PHI; 1 - PHI,THETA)
        NA    - number of steps along 1st variable
        NB    - number of steps along 2nd variable

        FUN(IA,IB,NV,AB,V,TT) - external routine
          IA      - cell number for 1st variable
          IB      - cell number for 2nd variable
          NV      - number of values for given cell
          AB(2,4) - coordinates of the cell corners
          V(NV)   - cell values
          TT(4,*) - additional function

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
            ICODES(1) - IA
            ICODES(2) - IB
            ICODES(3) - IV
            ICODES(4) - side: 1,2,3,4 - ordinary sides
                              5-bottom,6-top
          XYZ(3,*)  - coordinates of nodes
          NP        - number of nodes in face
```

```
              IFACE(NP) - face
              T(NP)     - additional function

            CHOPT       - options: 'BF' - from BACK to FRONT
                                   'FB' - from FRONT to BACK
```

void **LightSource**(Int_t *nl,* Float_t *yl,* Float_t *xscr,* Float_t *yscr,* Float_t *zscr,* Int_t *&irep*)

```
Set light source
 Input: NL   - source number: -1 off all light sources
                               0 set diffused light
        YL   - intensity of the light source
        XSCR \
        YSCR  - direction of the light (in respect of the screen)
        ZSCR /

 Output: IREP   - reply : 0 - O.K.
                         -1 - error in light sources definition:
                              negative intensity
                              source number greater than max
                              light source is placed at origin
```

void **Luminosity**(Float_t **anorm,* Float_t *&flum*)

```
Find surface luminosity at given point
                                     --
 Lightness model formula: Y = YD*QA + > YLi*(QD*cosNi+QS*cosRi)
                                     --

         B1     = VN(3)*VL(2) - VN(2)*VL(3)
         B2     = VN(1)*VL(3) - VN(3)*VL(1)
         B3     = VN(2)*VL(1) - VN(1)*VL(2)
         B4     = VN(1)*VL(1) + VN(2)*VL(2) + VN(3)*VL(3)
         VR(1)  = VN(3)*B2 - VN(2)*B3 + VN(1)*B4
         VR(2)  =-VN(3)*B1 + VN(1)*B3 + VN(2)*B4
         VR(3)  = VN(2)*B1 - VN(1)*B2 + VN(3)*B4
         S      = SQRT(VR(1)*VR(1)+VR(2)*VR(2)+VR(3)*VR(3))
         VR(1)  = VR(1)/S
         VR(2)  = VR(2)/S
         VR(3)  = VR(3)/S
         COSR   = VR(1)*0. + VR(2)*0. + VR(3)*1.

 References: WCtoNDC

 Input: ANORM(3) - surface normal at given point

 Output: FLUM - luminosity
```

void **ModifyScreen**(Float_t **r1,* Float_t **r2*)

```
Modify SCREEN
 Input: R1(3) - 1-st point of the line
        R2(3) - 2-nd point of the line
```

void **SetDrawFace**(DrawFaceFunc_t *drface*)

```
Store pointer to current algorithm to draw faces
```

void **SetLegoFunction**(LegoFunc_t *fun*)

```
Store pointer to current lego function
```

void **SetSurfaceFunction**(SurfaceFunc_t *fun*)

```
Store pointer to current surface function
```

void **SetColorDark**(Color_t *color,* Int_t *n*)

```
Store dark color for stack number n
```

void **SetColorMain**(Color_t *color,* Int_t *n*)
```
Store color for stack number n
```

void **SideVisibilityDecode**(Float_t *val,* Int_t &*iv1,* Int_t &*iv2,* Int_t &*iv3,* Int_t &*iv4,* Int_t &*iv5,* Int_t &*iv6,* Int_t &*ir*)
```
Decode side visibilities and order along R for sector
 Input: VAL - encoded value

 Output: IV1 ... IV6  - visibility of the sides
         IR           - increment along R
```

void **SideVisibilityEncode**(Int_t *iopt,* Float_t *phi1,* Float_t *phi2,* Float_t &*val*)
```
Encode side visibilities and order along R for sector
 Input: IOPT - options: 1 - from BACK to FRONT 'BF'
                        2 - from FRONT to BACK 'FB'
        PHI1 - 1st phi of sector
        PHI2 - 2nd phi of sector

 Output: VAL - encoded value
```

void **Spectrum**(Int_t *nl,* Float_t *fmin,* Float_t *fmax,* Int_t *ic,* Int_t *idc,* Int_t &*irep*)
```
Set Spectrum
 Input: NL   - number of levels
        FMIN - MIN fuction value
        FMAX - MAX fuction value
        IC   - initial color index (for 1st level)
        IDC  - color index increment

 Output: IREP - reply: 0 O.K.
                      -1 error in parameters
                         F_max less than F_min
                         illegal number of levels
                         initial color index is negative
                         color index increment must be positive
```

void **SurfaceCartesian**(Float_t *ang,* Int_t *nx,* Int_t *ny,* char **chopt*)
```
Draw surface in cartesian coordinate system
 Input: ANG      - angle between X ang Y
        NX       - number of steps along X
        NY       - number of steps along Y

        FUN(IX,IY,F,T) - external routine
          IX     - X number of the cell
          IY     - Y number of the cell
          F(3,4) - face which corresponds to the cell
          T(4)   - additional function (for example: temperature)

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
            ICODES(1) - IX
            ICODES(2) - IY
          NP        - number of nodes in face
          IFACE(NP) - face
          T(NP)     - additional function

        CHOPT - options: 'BF' - from BACK to FRONT
                         'FB' - from FRONT to BACK
```

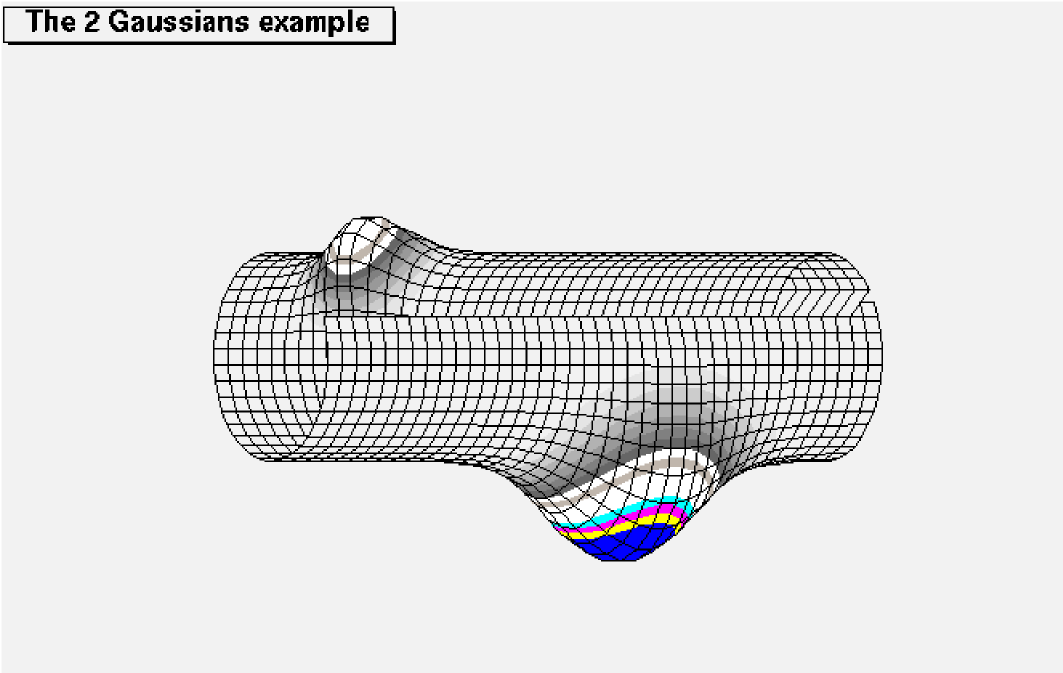
void **SurfaceFunction**(Int_t *ia,* Int_t *ib,* Float_t **f,* Float_t **t*)
```
Service function for Surfaces
```

void **SurfacePolar**(Int_t *iordr,* Int_t *na,* Int_t *nb,*char **chopt*)
```
Draw surface in polar coordinates
```

```
Input: IORDR - order of variables (0 - R,PHI, 1 - PHI,R)
       NA    - number of steps along 1st variable
       NB    - number of steps along 2nd variable

       FUN(IA,IB,F,T) - external routine
         IA     - cell number for 1st variable
         IB     - cell number for 2nd variable
         F(3,4) - face which corresponds to the cell
           F(1,*) - A
           F(2,*) - B
           F(3,*) - Z
         T(4)   - additional function (for example: temperature)

       DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
         ICODES(*) - set of codes for this face
           ICODES(1) - IA
           ICODES(2) - IB
         XYZ(3,*)  - coordinates of nodes
         NP        - number of nodes in face
         IFACE(NP) - face
         T(NP)     - additional function

       CHOPT       - options: 'BF' - from BACK to FRONT
                              'FB' - from FRONT to BACK
```

void **SurfaceCylindrical**(Int_t *iordr,* Int_t *na,* Int_t *nb,* char **chopt*)

```
Draw surface in cylindrical coordinates
 Input: IORDR - order of variables (0 - Z,PHI, 1 - PHI,Z)
        NA    - number of steps along 1st variable
        NB    - number of steps along 2nd variable

        FUN(IA,IB,F,T) - external routine
          IA     - cell number for 1st variable
          IB     - cell number for 2nd variable
          F(3,4) - face which corresponds to the cell
            F(1,*) - A
            F(2,*) - B
            F(3,*) - R
          T(4)   - additional function (for example: temperature)

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
            ICODES(1) - IA
            ICODES(2) - IB
          XYZ(3,*)  - coordinates of nodes
          NP        - number of nodes in face
          IFACE(NP) - face
          T(NP)     - additional function

        CHOPT       - options: 'BF' - from BACK to FRONT
                               'FB' - from FRONT to BACK
```



void **SurfaceSpherical**(Int_t *ipsdr,* Int_t *iordr,* Int_t *na,* Int_t *nb,* char **chopt*)

```
Draw surface in spheric coordinates
 Input: IPSDR - pseudo-rapidity flag
        IORDR - order of variables (0 - THETA,PHI; 1 - PHI,THETA)
        NA    - number of steps along 1st variable
        NB    - number of steps along 2nd variable

        FUN(IA,IB,F,T) - external routine
          IA      - cell number for 1st variable
          IB      - cell number for 2nd variable
          F(3,4) - face which corresponds to the cell
            F(1,*) - A
            F(2,*) - B
            F(3,*) - R
          T(4)    - additional function (for example: temperature)

        DRFACE(ICODES,XYZ,NP,IFACE,T) - routine for face drawing
          ICODES(*) - set of codes for this face
            ICODES(1) - IA
            ICODES(2) - IB
          XYZ(3,*)  - coordinates of nodes
          NP        - number of nodes in face
          IFACE(NP) - face
          T(NP)     - additional function

        CHOPT       - options: 'BF' - from BACK to FRONT
                               'FB' - from FRONT to BACK
```

void **SurfaceProperty**(Float_t *qqa,* Float_t *qqd,* Float_t *qqs,* Int_t *nnqs,* Int_t *&irep*)

```
Set surface property coefficients
 Input: QQA  - diffusion coefficient for diffused light  [0.,1.]
        QQD  - diffusion coefficient for direct light    [0.,1.]
        QQS  - diffusion coefficient for reflected light [0.,1.]
        NNCS - power coefficient for reflected light     (.GE.1)

                                    --
 Lightness model formula: Y = YD*QA + > YLi*(QD*cosNi+QS*cosRi)
                                    --
```

```
Output: IREP   - reply : 0 - O.K.
                        -1 - error in cooefficients
```

Author: René Brun, Evgueni Tcherniaev, Olivier Couet 12/12/94
Last update: 1.01/05 11/06/97 16.54.57 by Fons Rademakers

TLine

class **TLine** : public TObject, public TAttLine

protected:

virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

public:

class TLine	**TLine**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
class TLine	**TLine**( )
class TLine	**TLine**(class TLine& *line*)
void	**~TLine**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *line*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawLine**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
Coord_t	**GetX1**( )
Coord_t	**GetX2**( )
Coord_t	**GetY1**( )
Coord_t	**GetY2**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintLine**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
virtual void	**PaintLineNDC**(Coord_t *u1,* Coord_t *v1,* Coord_t *u2,* Coord_t *v2*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**((*unknown*) ofstream& out, Option_t* option)
virtual void	**SetX1**(Coord_t *x1*)
virtual void	**SetX2**(Coord_t *x2*)
virtual void	**SetY1**(Coord_t *y1*)
virtual void	**SetY2**(Coord_t *y2*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Coord_t	**fX1**	*X of 1st point*
Coord_t	**fY1**	*Y of 1st point*
Coord_t	**fX2**	*X of 2nd point*
Coord_t	**fY2**	*Y of 2nd point*

See Also: TArrow, TGaxis

Class Description

```
A Graphical line
```

TLine( ): **TObject**( ), **TAttLine**( )

```
Line default constructor
```

TLine(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*):**TObject**( ), **TAttLine**( )

```
Line normal constructor
```

~**TLine**( )

```
Line default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this line to line
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a line
```

void **Draw**(Option_t **option*)
```
Draw this line with its current attributes
```

void **DrawLine**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
```
Draw this line with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function is called when a line is clicked with the locator

If Left button clicked on one of the line end points, this point
  follows the cursor until button is released.

if Middle button clicked, the line is moved parallel to itself
  until the button is released.
```

void **ls**(Option_t **option*)
```
List this line with its attributes
```

void **Paint**(Option_t **option*)
```
Paint this line with its current attributes
```

void **PaintLine**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
```
Draw this line with new coordinates
```

void **PaintLineNDC**(Coord_t *u1,* Coord_t *v1,* Coord_t *u2,* Coord_t *v2*)
```
Draw this line with new coordinates in NDC
```

void **Print**(Option_t **option*)
```
Dump this line with its attributes
```

void **SavePrimitive**(ofstream *&out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 12/12/94
Last update: 1.03/04 30/09/97 16.00.30 by Fons Rademakers

TLink

class **TLink** : public TText

public:

class TLink	**TLink**(Coord_t *x*, Coord_t *y*, void* *pointer*)
class TLink	**TLink**( )
class TLink	**TLink**(class TLink&)
void	**~TLink**( )
class TClass*	**Class**( )
virtual void	**ExecuteEvent**(Int_t *event*, Int_t *px*, Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

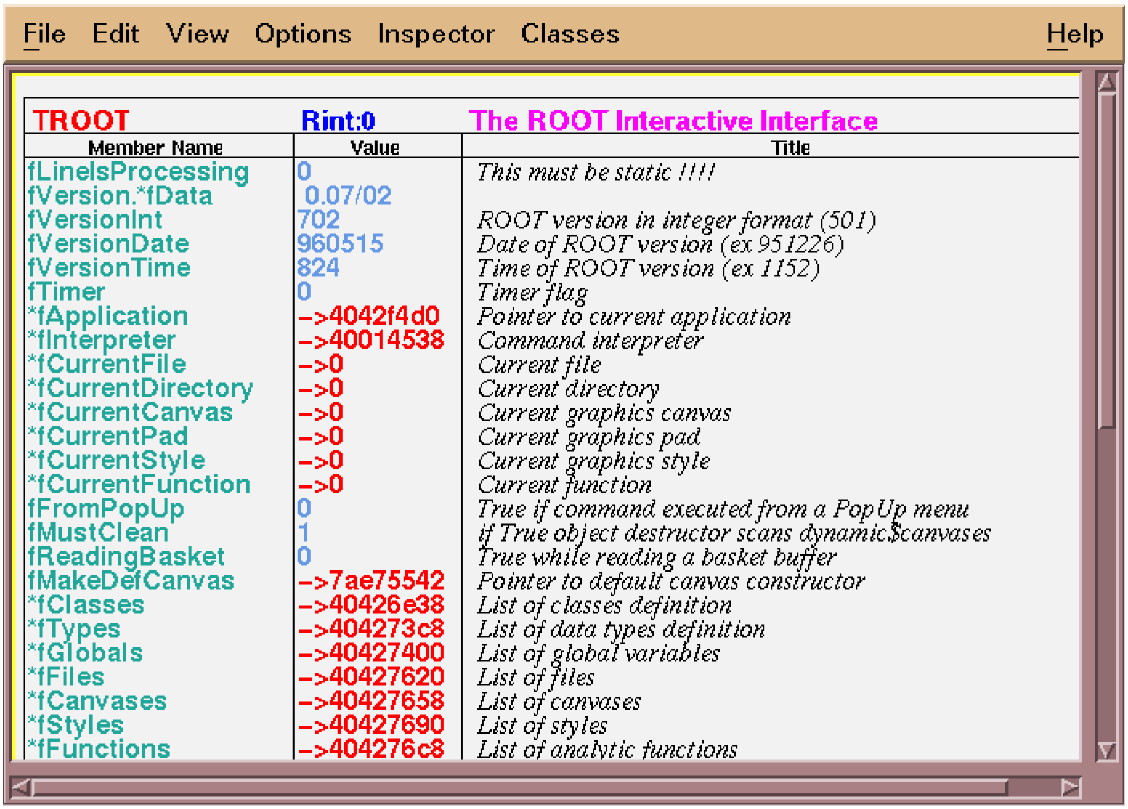
Data Members

protected:

void*	**fLink**	*pointer to object*

Class Description

```
Special TText object used to show hyperlinks.
In the example below created by TObject::Inspect, TLinks are used
to show pointers to other objects.
Clicking on one link, inspect the corresponding object.
```



TLink( ) : **TText**( )

```
Link default constructor
```

TLink(Float_t *x*, Float_t *y*, void * *pointer*): **TText**(*x*, *y*, "")

```
Constructor to define a link object
```

```
pointer points to any kind of object.
```

~TLink( )

```
Link default destructor
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a link is clicked with the locator

If mouse is clicked on a link text, the object pointed by the link
is Inspected
```

Author: René Brun 05/03/95
Last update: 1.02/00 09/07/97 22.53.25 by René Brun

TList

class **TList** : public TSeqCollection

protected:

virtual void	**DeleteLink**(class TObjLink* *lnk*)
class TObjLink*	**FindLink**(class TObject* *obj,* Int_t& *idx*)
class TObjLink*	**LinkAt**(Int_t *idx*)
virtual class TObjLink*	**NewLink**(class TObject* *obj,* class TObjLink* *prev = 0*)
virtual class TObjLink*	**NewOptLink**(class TObject* *obj,* Option_t* *opt,* class TObjLink* *prev = 0*)

public:

class TList	**TList**(class TObject* *parent*)
class TList	**TList**(class TList&)
class TList	**TList**( )
void	**~TList**( )
virtual void	**Add**(class TObject* *obj*)
virtual void	**Add**(class TObject* *obj,* Option_t* *opt*)
virtual void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
virtual void	**AddAt**(class TObject* *obj,* Int_t *idx*)
virtual void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj,* Option_t* *opt*)
virtual void	**AddLast**(class TObject* *obj*)
virtual void	**AddLast**(class TObject* *obj,* Option_t* *opt*)
virtual class TObject*	**After**(class TObject* *obj*)
virtual class TObject*	**At**(Int_t *idx*)
virtual class TObject*	**Before**(class TObject* *obj*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Delete**(Option_t* *option*)
virtual class TObject*	**FindObject**(Text_t* *name*)
virtual class TObject*	**FindObject**(class TObject* *obj*)
virtual class TObject*	**First**( )
virtual class TObjLink*	**FirstLink**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Last**( )
virtual class TObjLink*	**LastLink**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir = kIterForward*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

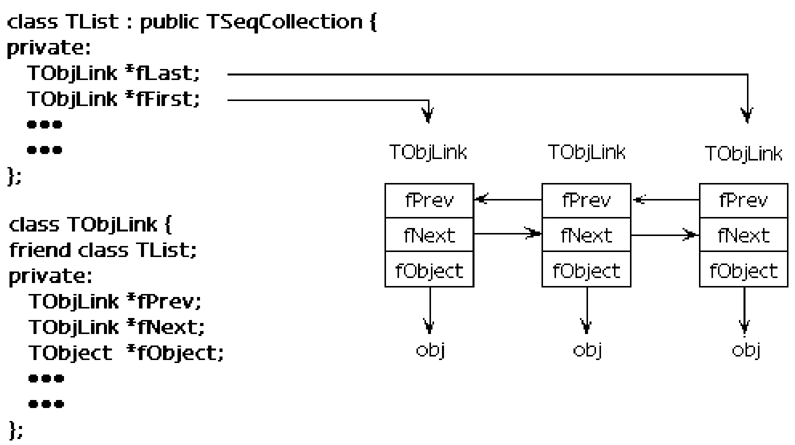
class TObjLink*	**fFirst**	*pointer to first entry in linked list*
class TObjLink*	**fLast**	*pointer to last entry in linked list*
class TObjLink*	**fCache**	*cache to speedup sequential calling of Before( ) and After( ) functions*

See Also: THashList, TSortedList

Class Description

```
TList

A doubly linked list. All classes inheriting from TObject can be
inserted in a TList. Before being inserted into the list the object
```

```
pointer is wrapped in a TObjLink object which contains, besides
the object pointer also a previous and next pointer.

There are basically four ways to iterate over a TList (in order
of preference, if not forced by other constraints):
 1) Using the ForEach macro:
      GetListOfPrimitives()->ForEach(TObject,Paint)(option);

 2) Using the TList iterator TListIter (via the wrapper class
    TIter):
      TIter next(GetListOfPrimitives());
      while (TObject *obj = next())
         obj->Draw(next.GetOption());

 3) Using the TObjLink list entries (that wrap the TObject*):
      TObjLink *lnk = GetListOfPrimitives()->FirstLink();
      while (lnk) {
         lnk->GetObject()->Draw(lnk->GetOption());
         lnk = lnk->Next();
      }

 4) Using the TList's After() and Before() member functions:
      TFree *idcur = this;
      while (idcur) {
         ...
         ...
         idcur = (TFree*)GetListOfFree()->After(idcur);
      }

Methods 2, 3 and 4 can also easily iterate backwards using either
a backward TIter (using argument kIterBackward) or by using
LastLink() and lnk->Prev() or by using the Before() member.
```



~TList( )

```
Delete the list. Objects are not deleted.
```

void **AddFirst**(TObject **obj*)

```
Add object at the beginning of the list.
```

void **AddFirst**(TObject **obj,* const Option_t **opt*)

```
Add object at the beginning of the list and also store option.
Storing an option is useful when one wants to change the behaviour
of an object a little without having to create a complete new
copy of the object. This feature is used, for example, by the Draw()
method. It allows the same object to be drawn in different ways.
```

void **AddLast**(TObject **obj*)
```
Add object at the end of the list.
```

void **AddLast**(TObject **obj,* const Option_t **opt*)
```
Add object at the end of the list and also store option.
Storing an option is useful when one wants to change the behaviour
of an object a little without having to create a complete new
copy of the object. This feature is used, for example, by the Draw()
method. It allows the same object to be drawn in different ways.
```

void **AddBefore**(TObject **before,* TObject **obj*)
```
Insert object before object before in the list.
```

void **AddAfter**(TObject **after,* TObject **obj*)
```
Insert object after object after in the list.
```

void **AddAt**(TObject **obj,* Int_t *idx*)
```
Insert object at position idx in the list.
```

TObject* **After**(TObject **obj*) const
```
Returns the object after object obj. Returns 0 if obj is last in list.
```

TObject* **At**(Int_t *idx*) const
```
Returns the object at position idx. Returns 0 if idx is out of range.
```

TObject* **Before**(TObject **obj*) const
```
Returns the object before object obj. Returns 0 if obj is first in list.
```

void **Clear**(Option_t **option*)
```
Remove all objects from the list. Does not delete the objects.
```

void **Delete**(Option_t **option*)
```
Remove all objects from the list AND delete all heap based objects.
```

void **DeleteLink**(TObjLink **lnk*)
```
Delete a TObjLink object.
```

TObject* **FindObject**(const Text_t **name*) const
```
Find an object in this list using its name. Requires a sequential
scan till the object has been found. Returns 0 if object with specified
name is not found. This method overrides the generic FindObject()
of TCollection for efficiency reasons.
```

TObject* **FindObject**(TObject **obj*) const
```
Find an object in this list using the object's IsEqual()
member function. Requires a sequential scan till the object has
been found. Returns 0 if object is not found.
This method overrides the generic FindObject() of TCollection for
efficiency reasons.
```

TObjLink* **FindLink**(TObject *obj, Int_t &*idx*) const
```
Returns the TObjLink object that contains object obj. In idx it returns
the position of the object in the list.
```

TObject* **First**( ) const
```
Return the first object in the list. Returns 0 when list is empty.
```

TObject* **Last**( ) const
```
Return the last object in the list. Returns 0 when list is empty.
```

TObjLink* **LinkAt**(Int_t *idx*) const
```
Return the TObjLink object at index idx.
```

TIterator* **MakeIterator**(Bool_t *dir*) const

```
Return a list iterator.
```

TObjLink* **NewLink**(TObject **obj,* TObjLink **prev*)

```
Return a new TObjLink.
```

TObjLink* **NewOptLink**(TObject **obj,* const Option_t **opt,* TObjLink **prev*)

```
Return a new TObjOptLink (a TObjLink that also stores the option).
```

TObject* **Remove**(TObject **obj*)

```
Remove object from the list.
```

Author: Fons Rademakers 10/08/95
Last update: 1.03/04 29/09/97 19.13.18 by Fons Rademakers

TListIter

class **TListIter** : public TIterator

private:

class TListIter	**TListIter**( )

public:

class TListIter	**TListIter**(class TList* *l,* Bool_t *dir* = *kIterForward*)
class TListIter	**TListIter**(class TListIter&)
void	**~TListIter**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual const Option_t*	**GetOption**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
virtual void	**Reset**( )
void	**SetOption**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

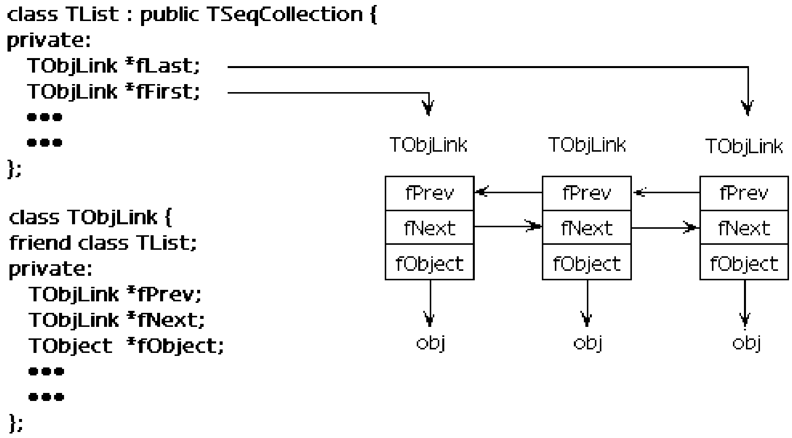
const class TList*	**fList**	*list being iterated*
class TObjLink*	**fCurCursor**	*current position in list*
class TObjLink*	**fCursor**	*next position in list*
Bool_t	**fDirection**	*iteration direction*
Bool_t	**fStarted**	*iteration started*

Class Description

```
TList

A doubly linked list. All classes inheriting from TObject can be
inserted in a TList. Before being inserted into the list the object
pointer is wrapped in a TObjLink object which contains, besides
the object pointer also a previous and next pointer.

There are basically four ways to iterate over a TList (in order
of preference, if not forced by other constraints):
 1) Using the ForEach macro:
      GetListOfPrimitives()->ForEach(TObject,Paint)(option);

 2) Using the TList iterator TListIter (via the wrapper class
    TIter):
      TIter next(GetListOfPrimitives());
      while (TObject *obj = next())
         obj->Draw(next.GetOption());

 3) Using the TObjLink list entries (that wrap the TObject*):
      TObjLink *lnk = GetListOfPrimitives()->FirstLink();
      while (lnk) {
         lnk->GetObject()->Draw(lnk->GetOption());
         lnk = lnk->Next();
      }

 4) Using the TList's After() and Before() member functions:
      TFree *idcur = this;
      while (idcur) {
         ...
```

```
        ...
        idcur = (TFree*)GetListOfFree()->After(idcur);
    }
```

```
Methods 2, 3 and 4 can also easily iterate backwards using either
a backward TIter (using argument kIterBackward) or by using
LastLink() and lnk->Prev() or by using the Before() member.
```



TListIter(const TList **l,* Bool_t *dir*): **fList**(*l*), **fCurCursor**(0), **fCursor**(0), **fDirection**(dir), **fStarted**(kFALSE)

```
Create a new list iterator. By default the iteration direction
is kIterForward. To go backward use kIterBackward.
```

TObject* **Next**( )

```
Return next object in the list. Returns 0 when no more objects in list.
```

const Option_t* **GetOption**( ) const

```
Returns the object option stored in the list.
```

void **SetOption**(const Option_t **option*)

```
Sets the object option stored in the list.
```

Author: Fons Rademakers 10/08/95
Last update: 1.03/04 29/09/97 19.13.18 by Fons Rademakers

TMap

class **TMap** : public TCollection

public:

class TMap	**TMap**(Int_t *capacity* = TCollection::*kInitHashTableCapacity,* Int_t *rehash* = *0*)
class TMap	**TMap**(class TMap&)
void	**~TMap**( )
virtual void	**Add**(class TObject* *obj*)
void	**Add**(class TObject* *key,* class TObject* *value*)
Float_t	**AverageCollisions**( )
Int_t	**Capacity**( )
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
Int_t	**Collisions**(char* *keyname*)
Int_t	**Collisions**(class TObject* *key*)
virtual void	**Delete**(Option_t* *option*)
void	**DeleteAll**( )
virtual class TObject*	**FindObject**(class TObject* *key*)
virtual class TObject*	**FindObject**(Text_t* *keyname*)
class TObject*	**GetValue**(class TObject* *key*)
virtual class TClass*	**IsA**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir* = *kIterForward*)
void	**Rehash**(Int_t *newCapacity,* Bool_t *checkObjValidity* = *kTRUE*)
virtual class TObject*	**Remove**(class TObject* *key*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class THashTable*	**fTable**	*Hash table used to store TAssociation's*

Class Description

```
TMap

TMap implements an associative array of (key,value) pairs using a
THashTable for efficient retrieval (therefore TMap does not conserve
the order of the entries). The hash value is calculated
using the value returned by the keys Hash() function. Both key and
value need to inherit from TObject.
```

```
class TMap : public TCollection {
private:
  THashTable *fTable;        THashTable used to store TAssoc's
  •••                        the hash value is provided by the
  •••                        Hash() method of the fKey object
};

class TAssoc : public TObject {
private:
  TObject *fKey;
  TObject *fValue;
  •••
  •••
};
```

TMap(Int_t *capacity,* Int_t *rehashlevel*)

TMap ctor. See THashTable for a description of the arguments.

~TMap( )

TMap dtor.

void **Add**(TObject **obj*)

This function may not be used (but we need to provide it since it is a pure virtual in TCollection). Use Add(key,value) instead.

void **Add**(TObject **key,* TObject **value*)

Add a (key,value) pair to the map.

Float_t **AverageCollisions**( ) const

Return the ratio of entries vs occupied slots.

Int_t **Capacity**( ) const

Return number of slots in the hashtable. Use GetSize() to get the number of objects stored in the TMap.

void **Clear**(Option_t **option*)

Remove all (key,value) pairs from the map but DO NOT delete the keys and/or values.

Int_t **Collisions**(const char **keyname*) const

Returns the number of collisions for a key with a certain name (i.e. number of objects in same slot in the hash table, i.e. length of linked list).

Int_t **Collisions**(TObject **key*) const

Returns the number of collisions for a key (i.e. number of objects in same slot in the hash table, i.e. length of linked list).

void **Delete**(Option_t **option*)

Remove all (key,value) pairs from the map AND delete the keys when they are allocated on the heap.

void **DeleteAll**( )

Remove all (key,value) pairs from the map AND delete the keys AND values when they are allocated on the heap.

TObject* **FindObject**(const Text_t **keyname*) const

Check if a (key,value) pair exists with keyname as name of the key. Returns a TAssoc* (need to downcast from TObject). Use Key() and Value() to get the pointers to the key and value, respectively. Returns 0 if not found.

TObject* **FindObject**(TObject **key*) const

Check if a (key,value) pair exists with key as key. Returns a TAssoc* (need to downcast from TObject). Use Key() and Value() to get the pointers to the key and value, respectively. Returns 0 if not found.

TObject* **GetValue**(TObject **key*) const

Returns a pointer to the value associated with key.

TIterator* **MakeIterator**(Bool_t *dir*) const

Create an iterator for TMap.

void **Rehash**(Int_t *newCapacity,* Bool_t *checkObjValidity*)

Rehash the underlaying THashTable (see THashTable::Rehash()).

TObject* **Remove**(TObject **key*)

Remove the (key,value) pair with key from the map.

void **Streamer**(TBuffer &*b*)

```
Stream all key/value pairs in the map to or from the I/O buffer.
```

Author: Fons Rademakers 12/11/95
Last update: 1.03/04 29/09/97 19.14.24 by Fons Rademakers

TMapFile

class **TMapFile** : public TObject

protected:

class TMapFile	**TMapFile**(Text_t* *name,* Text_t* *title,* Option_t* *option,* Int_t *size,* class TMapFile*& *newMapFile*)
class TMapFile	**TMapFile**( )
class TMapFile	**TMapFile**(class TMapFile& *f,* Long_t *offset = 0*)
Int_t	**AcquireSemaphore**( )
void	**CreateSemaphore**(Int_t *pid = 0*)
void	**DeleteSemaphore**( )
class TMapFile*	**FindShadowMapFile**( )
void	**InitDirectory**( )
void*	**MapToAddress**( )
void	***operator=***(class TMapFile& *rhs*)
Int_t	**ReleaseSemaphore**( )

public:

void	**~TMapFile**( )
void	**Add**(class TObject* *obj,* char* *name*)
void	**Browse**(class TBrowser* *b*)
void	**cd**(Text_t* *path = 0*)
class TClass*	**Class**( )
void	**Close**(Option_t* *option*)
class TMapFile*	**Create**(Text_t* *name,* Option_t* *option,* Int_t *size = kDefaultMapSize,* Text_t* *title*)
class TObject*	**Get**(char* *name,* class TObject* *retObj = 0*)
void*	**GetBaseAddr**( )
class TDirectory*	**GetDirectory**( )
Int_t	**GetFd**( )
class TMapRec*	**GetFirst**( )
class TMapRec*	**GetLast**( )
void*	**GetMmallocDesc**( )
const char*	**GetName**( )
const char*	**GetOption**( )
Int_t	**GetSize**( )
const char*	**GetTitle**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
Bool_t	**IsWritable**( )
void	**ls**(Option_t* *option*)
void	**operator delete**(void* *vp*)
void*	**OrgAddress**(void* *addr*)
void	**Print**(Option_t* *option*)
void	**Remove**(class TObject* *obj*)
void	**RemoveAll**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Update**(class TObject* *obj = 0*)
class TMapFile*	**WhichMapFile**(void* *addr*)

Data Members

private:

Int_t	**fFd**	*Descriptor of mapped file*
Int_t	**fVersion**	*ROOT version (or -1 for shadow map file)*
char*	**fName**	*Name of mapped file*

char*	**fTitle**	*Title of mapped file*
char*	**fOption**	*Directory creation options*
void*	**fMmallocDesc**	*Pointer to mmalloc descriptor*
ULong_t	**fBaseAddr**	*Base address of mapped memory region*
Int_t	**fSize**	*Size of memory mapped region*
class TMapRec*	**fFirst**	*List of streamed objects is shared memory*
class TMapRec*	**fLast**	*Last object in list of shared objects*
Long_t	**fOffset**	*Offset in bytes for region mapped by reader*
class TDirectory*	**fDirectory**	*pointer to directory associated to this mapfile*
class TList*	**fBrowseList**	*list of KeyMapFile objects*
Bool_t	**fWritable**	*TRUE if mapped file opened in RDWR mode*
Int_t	**fSemaphore**	*Modification semaphore (or getpid( ) for WIN32)*
HANDLE	**fhSemaphore**	*HANDLE of WIN32 Mutex object to implement semaphore*

public:

static const enum TObject:: **kDefaultMapSize**

Class Description

```
TMapFile

This class implements a shared memory region mapped to a file.
Objects can be placed into this shared memory area using the Add()
member function. Whenever the mapped object(s) change(s) call
Update() to put a fresh copy in the shared memory. This extra
step is necessary since it is not possible to share objects with
virtual pointers between processes (the vtbl ptr points to the
originators unique address space and can not be used by the
consumer process(es)). Consumer processes can map the memory region
from this file and access the objects stored in it via the Get()
method (which returns a copy of the object stored in the shared
memory with correct vtbl ptr set). Only objects of classes with a
Streamer() member function defined can be shared.
```

TMapFile( )

```
Default ctor. Does not much except setting some basic values.
```

TMapFile(const Text_t **name,* const Text_t **title,* Option_t **option,*Int_t size, TMapFile *&*newMapFile*)

```
Create a memory mapped file. This opens a file (to which the
memory will be mapped) and attaches a memory region to it.
Option can be either: "NEW", "CREATE", "RECREATE", "UPDATE" or
"READ" (see TFile). The default open mode is "READ". The size
argument specifies the maximum size of shared memory file in bytes.
This protected ctor is called via the static Create() method.
```

TMapFile(const TMapFile &*f,* Long_t *offset*)

```
Private copy ctor. Used by the the ctor to create a new version
of TMapFile in the memory mapped heap. It's main purpose is to
correctly create the string data members.
```

~TMapFile( )

```
TMapFiles may not be deleted, since we want to keep the complete
TMapFile object in the mapped file for later re-use. To enforce this
the delete operator has been made private. Use Close() to properly
terminate a TMapFile (also done via the TROOT dtor).
```

void **InitDirectory**( )

```
Create the directory associated to this mapfile
```

void **Add**(TObject **obj,* const char **name*)

```
Add an object to the list of objects to be stored in shared memory
and copy it to shared memory.
```

void **Update**(TObject **obj*)

```
Update an object (or all objects, if obj == 0) in shared memory.
```

void **Remove**(TObject **obj*)

```
Remove object from shared memory.
```

void **RemoveAll**( )

```
Remove all objects from shared memory.
```

TObject* **Get**(const char **name,* TObject **delObj*)

```
Return pointer to object retrieved from shared memory. The object must
be deleted after use. If delObj is a pointer to a previously allocated
object it will be deleted. Returns 0 in case object with the given
name does not exist.
```

void **CreateSemaphore**(int *pid*)

```
Create semaphore used for synchronizing access to shared memory.
```

void **DeleteSemaphore**( )

```
Delete the semaphore.
```

Int_t **AcquireSemaphore**( )

```
Acquire semaphore. Returns 0 of OK, -1 on error.
```

Int_t **ReleaseSemaphore**( )

```
Release semaphore. Reutnrs 0 if OK, -1 on error.
```

void **Close**(Option_t **option*)

```
Close a mapped file. First detach mapped memory then close file.
TMapFiles can not be "deleted", they can only be Close()'ed.
No member functions of a TMapFile that was opened in write mode
may be called after Close().
```

TMapFile* **FindShadowMapFile**( )

```
Returns shadow map file.
```

void **Print**(Option_t **option*)

```
Print some info about the mapped file.
```

Bool_t **IsFolder**( )

```
Returns kTRUE in case object is a folder (i.e. contains browsable lists).
```

void **Browse**(TBrowser **b*)

```
Browse contents of TMapFile.
```

void **ls**(Option_t **option*)

```
List contents of TMapFile.
```

TMapFile* **Create**(const Text_t **name,* Option_t **option,* Int_t *size,*const Text_t **title*)

```
Create a memory mapped file. This opens a file (to which the
memory will be mapped) and attaches a memory region to it.
Option can be either: "NEW", "CREATE", "RECREATE", "UPDATE"
or "READ" (see TFile). The default open mode is "READ". The size
argument specifies the maximum size of shared memory file in bytes.
TMapFile's can only be created via this method. Create() enforces that
a TMapFile is always on the memory mapped heap (when in "NEW", "CREATE"
or "RECREATE" are used).
```

void* **MapToAddress**( )

```
Return the base address at which we would like the next TMapFile's
mapped data to start.

For now, we let the system decide (start address 0). There are
```

```
a lot of issues to deal with here to make this work reasonably,
including:

- Avoid memory collisions with existing mapped address spaces

- Reclaim address spaces when their mmalloc heaps are unmapped

- When mmalloc heaps are shared between processes they have to be
mapped at the same addresses in each

Once created, a mmalloc heap that is to be mapped back in must be
mapped at the original address.  I.E. each TMapFile will expect
to be remapped at it's original address. This becomes a problem if
the desired address is already in use.
```

Author: Fons Rademakers 08/07/97
Last update: 1.02/00 23/07/97 19.37.21 by René Brun

TMapIter

class **TMapIter** : public TIterator

private:

class TMapIter	**TMapIter**( )

public:

class TMapIter	**TMapIter**(class TMap* *map*, Bool_t *dir* = *kIterForward*)
class TMapIter	**TMapIter**(class TMapIter&)
void	**~TMapIter**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
virtual void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

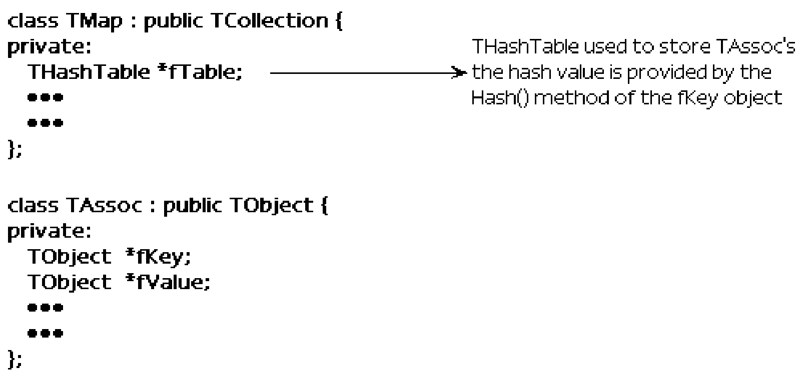
private:

const class TMap*	**fMap**	*map being iterated*
class THashTableIter*	**fCursor**	*current position in map*
Bool_t	**fDirection**	*iteration direction*

Class Description

TMap

TMap implements an associative array of (key,value) pairs using a THashTable for efficient retrieval (therefore TMap does not conserve the order of the entries). The hash value is calculated using the value returned by the keys Hash() function. Both key and value need to inherit from TObject.



TMapIter(const TMap **m*, Bool_t *dir*)

Create a map iterator. Use dir to specify the desired iteration direction.

~TMapIter( )

Map iterator dtor.

TObject* **Next**( )

Returns the next key from a map. Use TMap::GetValue() to get the value associated with the key. Returns 0 when no more items in map.

void **Reset**( )

```
Reset the map iterator.
```

Author: Fons Rademakers 12/11/95
Last update: 1.03/04 29/09/97 19.14.24 by Fons Rademakers

TMarker

class **TMarker** : public TObject, public TAttMarker

public:

class TMarker	**TMarker**(Coord_t *x,* Coord_t *y,* Int_t *marker*)
class TMarker	**TMarker**( )
class TMarker	**TMarker**(class TMarker& *marker*)
void	**~TMarker**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *marker*)
virtual void	**DisplayMarkersTypes**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawMarker**(Coord_t *x,* Coord_t *y*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Coord_t	**GetX**( )
Coord_t	**GetY**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintMarker**(Coord_t *x,* Coord_t *y*)
virtual void	**PaintMarkerNDC**(Coord_t *u,* Coord_t *v*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetX**(Coord_t *x*)
virtual void	**SetY**(Coord_t *y*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

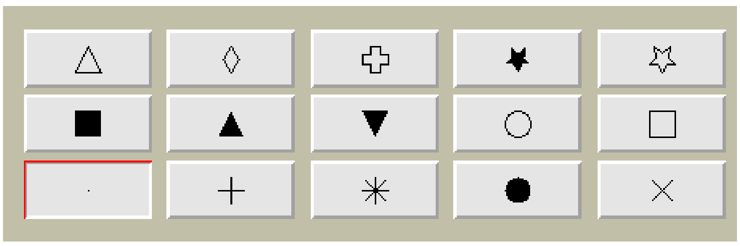
Data Members

protected:

Coord_t	**fX**	*X position of marker (left, center, etc..)*
Coord_t	**fY**	*Y position of marker (left, center, etc..)*

Class Description

```
Manages Markers. Marker attributes are managed by TAttMarker.
The list of standard ROOT markers is shown in this picture
```



TMarker( ): **TObject**( ), **TAttMarker**( )

```
Marker default constructor
```

TMarker(Coord_t *x,* Coord_t *y,* Int_t *marker*):**TObject**( ), **TAttMarker**( )

```
Marker normal constructor
```

~**TMarker**( )

```
Marker default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this marker to marker
```

void **DisplayMarkersTypes**( )

```
Display the table of markers with their numbers
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a marker
Compute the closest distance of approach from point px,py to this marker.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw this marker with its current attributes
```

void **DrawMarker**(Coord_t *x,* Coord_t *y*)

```
Draw this marker with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a marker is clicked with the locator

If Left button is clicked on a marker, the marker is moved to
  a new position when the mouse button is released.
```

void **ls**(Option_t **option*)

```
List this marker with its attributes
```

void **Paint**(Option_t **option*)

```
Paint this marker with its current attributes
```

void **PaintMarker**(Coord_t *x,* Coord_t *y*)

```
Draw this marker with new coordinates
```

void **PaintMarkerNDC**(Coord_t *u,* Coord_t *v*)

```
Draw this marker with new coordinates in NDC
```

void **Print**(Option_t **option*)

```
Dump this marker with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 12/05/95
Last update: 1.02/00 09/07/97 22.26.33 by René Brun

TMarker3DBox

class **TMarker3DBox** : public TObject, public TAttLine, public TAttFill

protected:

virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Local2Master**(Float_t* *local,* Float_t* *master*)
virtual void	**PaintGLPoints**(Float_t* *buff*)

public:

class TMarker3DBox	**TMarker3DBox**(Float_t *x,* Float_t *y,* Float_t *z,* Float_t *dx,* Float_t *dy,* Float_t *dz,* Float_t *cx,* Float_t *cy,* Float_t *cz*)
class TMarker3DBox	**TMarker3DBox**( )
class TMarker3DBox	**TMarker3DBox**(class TMarker3DBox&)
void	**~TMarker3DBox**( )
class TClass*	**Class**( )
virtual void	**GetDirection**(Float_t& *cx,* Float_t& *cy,* Float_t& *cz*)
virtual void	**GetPosition**(Float_t& *x,* Float_t& *y,* Float_t& *z*)
class TObject*	**GetRefObject**( )
virtual void	**GetSize**(Float_t& *dx,* Float_t& *dy,* Float_t& *dz*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintShape**(X3DBuffer* *buff*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetDirection**(Float_t *cx,* Float_t *cy,* Float_t *cz*)
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**SetPosition**(Float_t *x,* Float_t *y,* Float_t *z*)
virtual void	**SetRefObject**(class TObject* *obj = 0*)
virtual void	**SetSize**(Float_t *dx,* Float_t *dy,* Float_t *dz*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fDx**	*half length in x*
Float_t	**fDy**	*half length in y*
Float_t	**fDz**	*half length in z*
Float_t	**fTranslation**	*The coordinats of the centre of the box*
Float_t	**fDirCos**	*The orientation of this box*
class TObject*	**fRefObject**	*Pointer to an object*

Class Description

```
Marker3DBox is a special 3-D marker designed for event display.
It has the following parameters:
 fDx;              half length in X
 fDy;              half length in Y
 fDz;              half length in Z
 fTranslation[3];  the coordinates of the center of the box
 fDirCos[3];       the direction cosinus defining the orientation
 fRefObject;       A reference to an object
```

TMarker3DBox( )

```
Marker3DBox  default constructor
```

TMarker3DBox(Float_t *x,* Float_t *y,* Float_t *z,*Float_t dx, Float_t dy, Float_t dz,Float_t cx, Float_t cy, Float_t *cz*):**TAttLine**(*1, 1, 1*), **TAttFill**(*1,* 0)

```
Marker3DBox normal constructor
```

~TMarker3DBox( )

```
Marker3DBox shape default destructor
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a Marker3DBox
Compute the closest distance of approach from point px,py to each corner
point of the Marker3DBox.
```

void **Local2Master**(Float_t **local,* Float_t **master*)

```
Convert one point from local system to master reference system
```

void **Paint**(Option_t **option*)

```
Paint this 3-D box marker with its current attributes
```

void **PaintGLPoints**(Float_t **vertex*)

```
Direct copy of TBRIK::PaintGLPoints
```

void **PaintShape**(X3DBuffer **buff*)

```
Paint 3-D marker in current pad with its current attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetPoints**(Float_t **buff*)

```
Direct copy of TBRIK::SetPoints
```

void **Sizeof3D**( ) const

```
Return total X3D size of this marker
```

Author: "Valery fine" 31/10/97
Last update: 1.03/07 08/11/97 11.13.50 by René Brun

TMaterial

class **TMaterial** : public TNamed, public TAttFill

public:

class TMaterial	**TMaterial**( )
class TMaterial	**TMaterial**(Text_t* *name,* Text_t* *title,* Float_t *a,* Float_t *z,* Float_t *density*)
class TMaterial	**TMaterial**(class TMaterial&)
void	**~TMaterial**( )
class TClass*	**Class**( )
virtual Float_t	**GetA**( )
virtual Float_t	**GetDensity**( )
virtual Int_t	**GetNumber**( )
virtual Float_t	**GetZ**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNumber**	*Material matrix number*
Float_t	**fA**	*A of Material*
Float_t	**fZ**	*Z of Material*
Float_t	**fDensity**	*Material density in gr/cm3*

See Also: TMixture

Class Description

```
Manages a detector material. See class TGeometry
```

TMaterial( )

```
Material default constructor
```

TMaterial(Text_t **name,* Text_t **title,* Float_t *a,* Float_t *z,* Float_t *density*):**TNamed**(*name, title*), **TAttFill**(*0,* 1)

```
Material normal constructor
```

~TMaterial( )

```
Material default destructor
```

void **Streamer**(TBuffer &*R__b*)

```
Stream an object of class TMaterial.
```

Author: René Brun 03/10/95
Last update: 1.01/07 25/06/97 16.54.30 by René Brun

TMemberInspector

class **TMemberInspector** : public TObject

public:

void	**~TMemberInspector**( )
class TClass*	**Class**( )
virtual void	**Inspect**(class TClass* *cl,* char* *parent,* char* *name,* void* *addr*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TMemberInspector

Abstract base class for accessing the datamembers of a class.
Classes derived from this class can be given as argument to the
ShowMembers() methods of ROOT classes. This feature facilitates
the writing of class browsers and inspectors.
```

Author: Fons Rademakers 15/07/96
Last update: 1.00/06 23/03/97 11.01.57 by René Brun

TMenu

class **TMenu** : public TMenuItem

public:

void	**~TMenu**( )
class TMenuItem*	**ActivateItem**(char* *name*)
void	**Add**(class TMenuItem* *item,* int *position = -1*)
class TMenuAction*	**AddAction**(char* *name,* XtCallbackProc *actionCallback,* XtCallbackProc *undoCallback,* XtPointer *clientData,* int *position = -1*)
class TMenuAction*	**AddAction**(char* *name,* XtCallbackProc *actionCallback = NULL,* XtPointer *clientData = NULL,* int *position = -1*)
class TMenuConfirmFirstAction*	**AddConfirmFirstAction**(char* *name,* XtCallbackProc *actionCallback = NULL,* XtPointer *clientData = NULL,* int *position = -1*)
class TMenuLabel*	**AddLabel**(char* *name,* int *position = -1*)
class TRadioSubMenu*	**AddRadioSubmenu**(char* *name,* MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL,* int *position = -1*)
class TRadioSubMenu*	**AddRadioSubmenu**(char* *name,* int *position = -1*)
class TRadioSubMenu*	**AddRadioSubmenu**(class TRadioSubMenu* *submenu,* int *position = -1*)
class TMenuSeparator*	**AddSeparator**(char* *name = separator,* int *position = -1*)
class TSubMenu*	**AddSubmenu**(char* *name,* int *position = -1*)
class TSubMenu*	**AddSubmenu**(class TSubMenu* *submenu,* int *position = -1*)
class TSubMenu*	**AddSubmenu**(char* *name,* MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL,* int *position = -1*)
class TMenuToggle*	**AddToggle**(char* *name,* XtCallbackProc *actionCallback = NULL,* XtPointer *clientData = NULL,* int *state = -1,* int *position = -1*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
class TMenuItem*	**DeactivateItem**(char* *name*)
class TMenuItem*	**FindNamedItem**(char* *name,* Boolean *caseless = FALSE*)
int	**GetItemPosition**(char* *name*)
int	**GetItemPosition**(class TMenuItem* *item*)
int	**GetItemPosition**(Widget *w*)
virtual class TClass*	**IsA**( )
int	**NumItems**( )
class TMenuItem*	**operator[]**(int *index*)
class TMenuItem*	**RemoveItem**(char* *name*)
class TMenuItem*	**Replace**(char* *name,* class TMenuItem* *newItem*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

int	**fNumberOfElements**
int	**fMaxNumberOfElements**
XtPointer	**fDefaultClientData**
class TMenuItem**	**fElements**

See Also: TMenuBar, TOptionMenu, TPopupMenu, TSubMenu

Class Description

```
TMenu

This is the abstract menu class. It forms the basis for many menu's
that can be instantiated in the application.
```

TMenuItem* **RemoveItem**(const char **name*)

```
Remove item from menu. Returns removed item. Item must be deleted
using delete by the user.
```

const char* **ClassName**( )

Author: Fons Rademakers 30/03/95
Last update: 1.03/03 04/09/97 19.11.11 by Piotr Golonka

TMenuAction

class **TMenuAction** : public TMenuItem

protected:

class TMenuAction	**TMenuAction**( )

public:

class TMenuAction	**TMenuAction**(char* *name*)
class TMenuAction	**TMenuAction**(class TMenuAction&)
void	**~TMenuAction**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
Boolean	**HasUndo**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Undo**( )

Data Members

public:

XtCallbackProc	**fActionCallback**
XtPointer	**fClientData**

See Also: TMenuConfirmFirstAction, TMenuToggle, TMenuUndoManager

Class Description

```
TMenuAction

The TMenuAction class is implemented as a push button gadget within
the menu.
```

Boolean **IsContainer**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuBar

class **TMenuBar** : public TMenu

public:

class TMenuBar	**TMenuBar**(char* *name,* Boolean *showHelpPane = TRUE*)
class TMenuBar	**TMenuBar**(MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL,* Boolean *showHelpPane = TRUE*)
class TMenuBar	**TMenuBar**(Boolean *showHelpPane = TRUE*)
class TMenuBar	**TMenuBar**(MenuDesc_t* *menuDesc,* MenuDesc_t* *helpDesc,* XtPointer *defaultClientData = NULL*)
class TMenuBar	**TMenuBar**(class TMenuBar&)
class TMenuBar	**TMenuBar**(char* *name,* MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL,* Boolean *showHelpPane = TRUE*)
void	**~TMenuBar**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
class THelpPane*	**HelpPane**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class THelpPane*	**fHelpPane**

Class Description

```
TMenuBar

This class allows the user to assign menu bars to TWindows that make
up the interface.
```

void **SetLabel**(const char */*str*/)

```
Does nothing now...
```

Boolean **IsContainer**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuConfirmFirstAction

class **TMenuConfirmFirstAction** : public TMenuAction

private:
void **ConfirmCallback**(*Widget, XtPointer, XtPointer*)
protected:
class TMenuConfirmFirstAction**TMenuConfirmFirstAction**( )
public:
class TMenuConfirmFirstAction**TMenuConfirmFirstAction**(char* *name*)
class TMenuConfirmFirstAction**TMenuConfirmFirstAction**(class TMenuConfirmFirstAction&)
void **~TMenuConfirmFirstAction**( )
class TClass* **Class**( )
const char* **ClassName**( )
Widget **Instantiate**(Widget *parent*)
virtual class TClass* **IsA**( )
Boolean **IsContainer**( )
EMenuItemType **MenuType**( )
virtual void **ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void **Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TMenuConfirmFirstAction

This class handles menu actions that require user confirmation
execution.
```

Boolean **IsContainer**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuItem

class **TMenuItem** : public TComponent

public:

void	**~TMenuItem**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TMenu, TMenuAction, TMenuLabel

Class Description

```
TMenuItem

This class defines the menu items.
```

~TMenuItem( )

```
empty
```

const char* **ClassName**( )

Author: Fons Rademakers 30/03/95
Last update: 1.03/03 04/09/97 21.52.15 by Piotr Golonka

TMenuLabel

class **TMenuLabel** : public TMenuItem

protected:

class TMenuLabel	**TMenuLabel**( )

public:

class TMenuLabel	**TMenuLabel**(char* *name*)
class TMenuLabel	**TMenuLabel**(class TMenuLabel&)
void	**~TMenuLabel**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TMenuSeparator

Class Description

```
TMenuLabel

This class defines menu labels, which are implemented as simple
label gadgets.
```

TMenuLabel(const char **name*) : **TMenuItem**(name)

```
empty for now
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuSeparator

class **TMenuSeparator** : public TMenuLabel

public:

class TMenuSeparator	**TMenuSeparator**(char* *name* = *separator*)
class TMenuSeparator	**TMenuSeparator**(class TMenuSeparator&)
void	**~TMenuSeparator**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
Boolean	**IsContainer**( )
EMenuItemType	**MenuType**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TMenuSeparator

This class defines menu separators, which are implemented as simple
separator gadgets.
```

TMenuSeparator(char **name*) : **TMenuLabel**(name)

```
empty for now
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuToggle

class **TMenuToggle** : public TMenuAction, public TToggle

protected:

class TMenuToggle	**TMenuToggle**( )

public:

class TMenuToggle	**TMenuToggle**(class TMenuToggle&)
class TMenuToggle	**TMenuToggle**(char* *name*)
void	**~TMenuToggle**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual Bool_t	**GetVisualState**( )
Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
Boolean	**IsContainer**( )
EMenuItemType	**MenuType**( )
void	**SetLabel**(char* *str*)
virtual void	**SetStateAndNotify**(Boolean *state*)
virtual void	**SetVisualState**(Boolean *state*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Boolean	**fInitialState**

Class Description

```
TMenuToggle

This class defines menu toggle buttons, which are implemented as
simple toggle button gadgets.
```

void **SetLabel**(const char **str*)

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuUndoManager

class **TMenuUndoManager** : public TMenuAction

private:

void	**RegisterAction**((*unknown*) TAction*)
void	**RegisterAction**(class TMenuAction*)

protected:

class TMenuUndoManager	**TMenuUndoManager**( )

public:

class TMenuUndoManager	**TMenuUndoManager**(char* *name*)
class TMenuUndoManager	**TMenuUndoManager**(class TMenuUndoManager&)
void	**~TMenuUndoManager**( )
void	**Add**(char* *name,* XtCallbackProc *undoCallback,* XtPointer *clientData*)
class TClass*	**Class**( )
const char*	**ClassName**( )
class TComponentList*	**HistoryList**( )
Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
Boolean	**IsContainer**( )
EMenuItemType	**MenuType**( )
void	**MultiLevel**(Boolean *flag*)
void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TComponentList*	**fHistory**
Boolean	**fIsMultiLevel**

Class Description

```
TMenuUndoManager

This class registers actions and provides an undo mechanism.
```

void **RegisterAction**(TMenuAction*)

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMenuWindow

class **TMenuWindow** : public TSimpleWindow

protected:

class TMenuWindow	**TMenuWindow**( )

public:

class TMenuWindow	**TMenuWindow**(class TMenuWindow&)
class TMenuWindow	**TMenuWindow**(char* *name,* ArgList *args = NULL,* Cardinal *argCount = 0*)
void	**~TMenuWindow**( )
class TSubMenu*	**AddMenuPane**(char* *name,* MenuDesc_t* *menudesc*)
class TSubMenu*	**AddMenuPane**(char* *name*)
class TRadioSubMenu*	**AddRadioMenuPane**(char* *name,* MenuDesc_t* *menudesc*)
class TRadioSubMenu*	**AddRadioMenuPane**(char* *name*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual class TClass*	**IsA**( )
virtual class TMenuBar*	**Menu**( )
void	**SetMenuBar**(MenuDesc_t* *menuDesc*)
void	**SetMenuBar**(class TMenuBar* *menuObj*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TMenuBar*	**fMenuBar**
Boolean	**fMenuBarInstantiated**

See Also: TMotifBrowser, TMotifCanvas

Class Description

```
TMenuWindow

This class differs from TSimpleWindow in that you can put menu bars
into TMenuWindows but not into TSimpleWindows.
```

TRadioSubMenu* **AddRadioMenuPane**(const char **name,*MenuDesc_t **menudesc*)

Author: Fons Rademakers 30/03/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TMessage

class **TMessage** : public TBuffer

private:

class TMessage	**TMessage**(void* *buf,* Int_t *bufsize*)
void	**SetLength**( )

public:

class TMessage	**TMessage**(UInt_t *what* = *kMESS_ANY*)
class TMessage	**TMessage**(class TMessage&)
void	**~TMessage**( )
class TClass*	**Class**( )
class TClass*	**GetClass**( )
virtual class TClass*	**IsA**( )
void	**Reset**(UInt_t *what*)
void	**Reset**( )
void	**SetWhat**(UInt_t *what*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
UInt_t	**What**( )

Data Members

private:

UInt_t	**fWhat**	*message type*
class TClass*	**fClass**	*if message is kMESS_OBJECT, pointer to object's class*

Class Description

```
TMessage

Message buffer class used for serializing objects and sending them
over a network. This class inherits from TBuffer the basic I/O
serializer.
```

TMessage(UInt_t *what*) : **TBuffer**(kWrite)

```
Create a TMessage object for storing objects. The "what" integer
describes the type of message. Predifined ROOT system message types
can be found in MessageTypes.h. Make sure your own message types are
unique from the ROOT defined message types (i.e. 0 - 10000 are
reserved by ROOT). In case you OR "what" with kMESS_ACK, the message
will wait for an acknowledgement from the remote side. This makes
the sending process synchronous.
```

TMessage(void **buf,* Int_t *bufsize*) : **TBuffer**(*kRead, bufsize,* buf)

```
Create a TMessage object for reading objects. The objects will be
copied in to buf. Use the What() method to get the message type.
```

void **Reset**( )

```
Reset the message buffer so we can use (i.e. fill) it again.
```

void **SetLength**( ) const

```
Set the message length at the beginning of the message buffer.
This method is only called by TSocket::Send().
```

void **SetWhat**(UInt_t *what*)

```
Using this method one can change the message type a posteriory.
```

Author: Fons Rademakers 19/12/96

Last update: 1.03/01 15/08/97 18.08.32 by Fons Rademakers

TMethod

class **TMethod** : public TFunction

private:

void	**CreateSignature**( )

public:

class TMethod	**TMethod**(class G__MethodInfo* *info = 0,* class TClass* *cl = 0*)
class TMethod	**TMethod**(class TMethod&)
void	**~TMethod**( )
class TClass*	**Class**( )
virtual class TDataMember*	**FindDataMember**( )
class TClass*	**GetClass**( )
virtual const Text_t*	**GetCommentString**( )
virtual class TList*	**GetListOfMethodArgs**( )
virtual class TClass*	**IsA**( )
enum EMenuItemKind	**IsMenuItem**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TClass*	**fClass**	*pointer to the class*
enum EMenuItemKind	**fMenuItem**	*type of menuitem in context menu*

Class Description

```
Each ROOT class (see TClass) has a linked list of methods.
This class describes one single method (member function).
The method info is obtained via the CINT api.
  see class TCint.

The method information is used a.o. by the THml class and by the
TTree class.
```

TMethod(G__MethodInfo **info,* TClass **cl*) : **TFunction**(info)

```
Default TMethod ctor. TMethods are constructed in TClass.
Comment strings are pre-parsed to find out whether the method is
a contxt-menu item.
```

const Text_t* **GetCommentString**( )

```
returns a comment string from the class declaration
```

void **CreateSignature**( )

```
Using the CINT method arg information create a complete signature string.
```

TDataMember* **FindDataMember**( )

```
Tries to guess DataMember from comment string
and Method's name <==(only if 1 Argument!)
If more then one argument=> returns pointer to the last argument.
It also sets MethodArgs' pointers to point to specified data members.

The form of comment string defining arguments is:
void XXX(Int_t x1, Float_t y2) //*ARGS={x1=>fX1,y2=>fY2}
where fX1, fY2 are data fields in the same class.
("pointers" to data members)
```

TList* **GetListOfMethodArgs**( )

```
Returns methodarg list and additionally updates fDataMember in TMethod by
```

```
Calling FindDataMember();
```

Author: René Brun 09/02/95
Last update: 1.03/03 08/09/97 13.19.42 by Piotr Golonka

TMethodArg

class **TMethodArg** : public TDictionary

public:

class TMethodArg	**TMethodArg**((*unknown*) G__MethodArgInfo* info = 0, class TFunction* method = 0)
class TMethodArg	**TMethodArg**(class TMethodArg&)
void	**~TMethodArg**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
class TDataMember*	**GetDataMember**( )
const char*	**GetDefault**( )
const char*	**GetFullTypeName**( )
class TFunction*	**GetMethod**( )
virtual const char*	**GetName**( )
class TList*	**GetOptions**( )
virtual const char*	**GetTitle**( )
const char*	**GetTypeName**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
virtual Long_t	**Property**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class G__MethodArgInfo*	**fInfo**	*pointer to CINT method argument info*
class TFunction*	**fMethod**	*pointer to the method or global function*
class TDataMember*	**fDataMember**	*TDataMember pointed by this arg, to get values and options from.*

Class Description

```
Each ROOT method (see TMethod) has a linked list of its arguments.
This class describes one single method argument.
The method argument info is obtained via the CINT api.
See class TCint.

The method argument information is used a.o. in the TContextMenu
and THtml classes.
```

TMethodArg(G__MethodArgInfo **info,* TFunction **method*) : **TDictionary**( )

```
Default TMethodArg ctor. TMethodArgs are constructed in TFunction
via a call to TCint::CreateListOfMethodArgs().
```

~TMethodArg( )

```
TMethodArg dtor deletes adopted G__MethodArgInfo object.
```

const char* **GetDefault**( ) const

```
Get default value of method argument.
```

const char* **GetName**( ) const

```
Get method argument name (can be empty string or 0).
```

const char* **GetTypeName**( ) const

```
Get type of method argument, e.g.: "class TDirectory*" -> "TDirectory"
Result needs to be used or copied immediately.
```

const char* **GetFullTypeName**( ) const

```
Get full type description of method argument, e.g.: "class TDirectory*".
```

const char* **GetTitle**( ) const

```
Get method argument description. This method always returns 0.
However, for backward compatibility, we currently return
GetFullTypeName().
```

Int_t **Compare**(TObject **obj*)

```
Compare to other object. Returns 0<, 0 or >0 depending on
whether "this" is lexicographically less than, equal to, or
greater than obj.
```

ULong_t **Hash**( )

```
Return hash value for TDataType based on its name.
```

Long_t **Property**( ) const

```
Get property description word. For meaning of bits see EProperty.
```

TList* **GetOptions**( )

```
Returns list of possible options - according to pointed datamember.
If there is no datamember field assigned to this methodarg - returns 0.
```

TDataMember* **GetDataMember**( )

```
return DataMember pointed by this methodArg.
If you want to specify list of options or current value for your
MethodArg (i.e. it is used as initial values in argument-asking dialogs
popped up from context-meny),you can get this value from one of data members of
the class.
The only restriction is , that this DataMember object must have its
Getter/Setter methods set-up correctly - for details look at TDataMember.
To learn how to specify the data member to which the argument should
"point" , look at TMethod. This is TMethod which sets up fDataMember,
so  it could work correctly.
```

Author: René Brun 04/02/95
Last update: 1.03/03 08/09/97 11.31.14 by Piotr Golonka

TMethodCall

class **TMethodCall** : public TObject

public:

class TMethodCall	**TMethodCall**(class TClass* *cl,* char* *method,* char* *params*)
class TMethodCall	**TMethodCall**( )
class TMethodCall	**TMethodCall**(char* *function,* char* *params*)
class TMethodCall	**TMethodCall**(class TMethodCall&)
void	**~TMethodCall**( )
void	**CallDtorOnly**(Bool_t *set* = *kTRUE*)
class TClass*	**Class**( )
void	**Execute**(void* *object,* char* *params,* Double_t& *retDouble*)
void	**Execute**(void* *object,* char* *params,* Text_t** *retText*)
void	**Execute**( )
void	**Execute**(char* *params*)
void	**Execute**(Long_t& *retLong*)
void	**Execute**(char* *params,* Long_t& *retLong*)
void	**Execute**(void* *object*)
void	**Execute**(void* *object,* char* *params*)
void	**Execute**(Double_t& *retDouble*)
void	**Execute**(void* *object,* Long_t& *retLong*)
void	**Execute**(char* *params,* Double_t& *retDouble*)
void	**Execute**(void* *object,* char* *params,* Long_t& *retLong*)
void	**Execute**(void* *object,* Double_t& *retDouble*)
class TFunction*	**GetMethod**( )
const Text_t*	**GetMethodName**( )
const Text_t*	**GetParams**( )
const Text_t*	**GetProto**( )
void	**Init**(class TClass* *cl,* char* *method,* char* *params*)
void	**Init**(char* *function,* char* *params*)
void	**InitWithPrototype**(char* *function,* char* *proto*)
void	**InitWithPrototype**(class TClass* *cl,* char* *method,* char* *proto*)
virtual class TClass*	**IsA**( )
enum EReturnType	**ReturnType**( )
void	**SetParamPtrs**(void* *paramArr*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class G__CallFunc*	**fFunc**	*CINT method invocation environment*
Long_t	**fOffset**	*offset added to object pointer before method invocation*
class TClass*	**fClass**	*pointer to the class info*
class TFunction*	**fMetPtr**	*pointer to the method or function info*
class TString	**fMethod**	*method name*
class TString	**fParams**	*argument string*
class TString	**fProto**	*prototype string*
Bool_t	**fDtorOnly**	*call only dtor and not delete when calling ~xxx*
enum EReturnType	**fRetType**	*method return type*

Class Description

```
TMethodCall

Method or function calling interface. Objects of this class contain
```

```
the (CINT) environment to call a global function or a method for an
object of a specific class with the desired arguments. This class is
espicially useful when a method has to be called more times for
different objects and/or with different arguments. If a function or
method needs to be called only once one better uses
TInterpreter::Execute().
```

TMethodCall( )

```
Default TMethodCall ctor. Use Init() to initialize the method call
environment.
```

TMethodCall(TClass **cl,* const char **method,* const char **params*)

```
Create a method invocation environment for a specific class, method and
parameters. The parameter string has the form: "\"aap\", 3, 4.35".
To execute the method call TMethodCall::Execute(object,...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

TMethodCall(const char **function,* const char **params*)

```
Create a global function invocation environment. The parameter
string has the form: "\"aap\", 3, 4,35". To execute the
function call TMethodCall::Execute(...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

~**TMethodCall**( )

```
TMethodCall dtor.
```

void **Init**(TClass **cl,* const char **method,* const char **params*)

```
Initialize the method invocation environment. Necessary input
information: the class, method name and the parameter string
of the form "\"aap\", 3, 4.35".
To execute the method call TMethodCall::Execute(object,...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

void **Init**(const char **function,* const char **params*)

```
Initialize the function invocation environment. Necessary input
information: the function name and the parameter string of
the form "\"aap\", 3, 4.35".
To execute the method call TMethodCall::Execute(...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

void **InitWithPrototype**(TClass **cl,* const char **method,* const char **proto*)

```
Initialize the method invocation environment. Necessary input
information: the class, method name and the prototype string of
the form: "char*,int,float".
To execute the method call TMethodCall::Execute(object,...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

void **InitWithPrototype**(const char **function,* const char **proto*)

```
Initialize the function invocation environment. Necessary input
information: the function name and the prototype string of
the form: "char*,int,float".
To execute the method call TMethodCall::Execute(...).
This two step method is much more efficient than calling for
every invocation TInterpreter::Execute(...).
```

TFunction* **GetMethod**( )

```
Returns the TMethod describing the method to be executed. This takes
all overriding and overloading into account (call TClass::GetMethod()).
```

```
Since finding the method is expensive the result is cached.
```

void **Execute**(void **object*)
```
Execute the method (with preset arguments) for the specified object.
```

void **Execute**(void **object,* const char **params*)
```
Execute the method for the specified object and argument values.
```

void **Execute**(void **object,* Long_t &*retLong*)
```
Execute the method (with preset arguments) for the specified object.
```

void **Execute**(void **object,* const char **params,* Long_t &*retLong*)
```
Execute the method for the specified object and argument values.
```

void **Execute**(void **object,* Double_t &*retDouble*)
```
Execute the method (with preset arguments) for the specified object.
```

void **Execute**(void **object,* const char **params,* Double_t &*retDouble*)
```
Execute the method for the specified object and argument values.
```

void **Execute**(void **object,* Text_t ***retText*)
```
Execute the method (with preset arguments) for the specified object.
```

void **Execute**(void **object,* const char **params,* Text_t ***retText*)
```
Execute the method for the specified object and argument values.
```

EReturnType **ReturnType**( )
```
Returns the return type of the method. Either (unsigned) long,
int, short and char, or float and double or anything else.
Since finding the return type is expensive the result is cached.
```

void **SetParamPtrs**(void **paramArr*)
```
ParamArr is an array containing the addresses where to take the
function parameters. At least as many pointers should be present in
the array as there are required arguments (all arguments - default args).
```

Author: Fons Rademakers 13/06/96
Last update: 1.03/03 22/08/97 16.53.12 by Piotr Golonka

TMinuit

class **TMinuit** : public TNamed

private:

class TMinuit	**TMinuit**(class TMinuit& *m*)

public:

class TMinuit	**TMinuit**( )
class TMinuit	**TMinuit**(Int_t *maxpar*)
void	**~TMinuit**( )
void	**BuildArrays**(Int_t *maxpar = 15*)
class TClass*	**Class**( )
void	**DeleteArrays**( )
Int_t	**GetMaxIterations**( )
class TMethodCall*	**GetMethodCall**( )
class TObject*	**GetObjectFit**( )
virtual class TClass*	**IsA**( )
void	**mnamin**( )
void	**mnbins**(Double_t *a1,* Double_t *a2,* Int_t *naa,* Double_t& *bl,* Double_t& *bh,* Int_t& *nb,* Double_t& *bwid*)
void	**mncalf**(Double_t* *pvec,* Double_t& *ycalf*)
void	**mncler**( )
void	**mncntr**(Int_t *ke1,* Int_t *ke2,* Int_t& *ierrf*)
void	**mncomd**(class TString *crdbin,* Int_t& *icondn*)
void	**mncont**(Int_t *ke1,* Int_t *ke2,* Int_t *nptu,* Double_t* *xptu,* Double_t* *yptu,* Int_t& *ierrf*)
void	**mncrck**(class TString& *crdbuf,* Int_t *maxcwd,* class TString& *comand,* Int_t& *lnc,* Int_t *mxp,* Double_t* *plist,* Int_t& *llist,* Int_t& *ierr,* Int_t *isyswr*)
void	**mncros**(Double_t& *aopt,* Int_t& *iercr*)
void	**mncuve**( )
void	**mnderi**( )
void	**mndxdi**(Double_t *pint,* Int_t *ipar,* Double_t& *dxdi*)
void	**mneig**(Double_t* *a,* Int_t *ndima,* Int_t *n,* Int_t *mits,* Double_t* *work,* Double_t *precis,* Int_t& *ifault*)
void	**mnemat**(Double_t* *emat,* Int_t *ndim*)
void	**mnerrs**(Int_t *number,* Double_t& *eplus,* Double_t& *eminus,* Double_t& *eparab,* Double_t& *gcc*)
void	**mneval**(Double_t *anext,* Double_t& *fnext,* Int_t& *ierev*)
void	**mnexcm**(class TString *comand,* Double_t* *plist,* Int_t *llist,* Int_t& *ierflg*)
void	**mnexin**(Double_t* *pint*)
void	**mnfixp**(Int_t *iint,* Int_t& *ierr*)
void	**mnfree**(Int_t *k*)
void	**mngrad**( )
void	**mnhelp**(class TString *comd*)
void	**mnhes1**( )
void	**mnhess**( )
void	**mnimpr**( )
void	**mninex**(Double_t* *pint*)
void	**mninit**(Int_t *i1,* Int_t *i2,* Int_t *i3*)
void	**mnlims**( )
void	**mnline**(Double_t* *start,* Double_t *fstart,* Double_t* *step,* Double_t *slope,* Double_t *toler*)
void	**mnmatu**(Int_t *kode*)
void	**mnmigr**( )
void	**mnmnos**( )
void	**mnmnot**(Int_t *ilax,* Int_t *ilax2,* Double_t& *val2pl,* Double_t& *val2mi*)
void	**mnparm**(Int_t *k,* class TString *cnamj,* Double_t *uk,* Double_t *wk,* Double_t

	a, Double_t *b,* Int_t& *ierflg*)
void	**mnpars**(class TString& *crdbuf,* Int_t& *icondn*)
void	**mnpfit**(Double_t* *parx2p,* Double_t* *pary2p,* Int_t *npar2p,* Double_t* *coef2p,* Double_t& *sdev2p*)
void	**mnpint**(Double_t& *pexti,* Int_t *i,* Double_t& *pinti*)
void	**mnplot**(Double_t* *xpt,* Double_t* *ypt,* class TString* *chpt,* Int_t *nxypt,* Int_t *nunit,* Int_t *npagwd,* Int_t *npagln*)
void	**mnpout**(Int_t *iuext,* class TString *chnam,* Double_t& *val,* Double_t& *err,* Double_t& *xlolim,* Double_t& *xuplim,* Int_t& *iuint*)
void	**mnprin**(Int_t *inkode,* Double_t *fval*)
void	**mnpsdf**( )
void	**mnrazz**(Double_t *ynew,* Double_t* *pnew,* Double_t* *y,* Int_t& *jh,* Int_t& *jl*)
void	**mnrn15**(Double_t& *val,* Int_t& *inseed*)
void	**mnrset**(Int_t *iopt*)
void	**mnsave**( )
void	**mnscan**( )
void	**mnseek**( )
void	**mnset**( )
void	**mnsimp**( )
void	**mnstat**(Double_t& *fmin,* Double_t& *fedm,* Double_t& *errdef,* Int_t& *npari,* Int_t& *nparx,* Int_t& *istat*)
void	**mntiny**(Double_t *epsp1,* Double_t& *epsbak*)
Bool_t	**mnunpt**(class TString& *cfname*)
void	**mnvert**(Double_t* *a,* Int_t *l,* Int_t *m,* Int_t *n,* Int_t& *ifail*)
void	**mnwarn**(char* *copt,* char* *corg,* char* *cmes*)
void	**mnwerr**( )
void	**SetFCN**(void (*)(Int_t &, Double_t *, Double_t &f, Double_t *, *Int_t*) fcn)
void	**SetFCN**(void* *fcn*)
void	**SetMaxIterations**(Int_t *maxiter = 500*)
void	**SetObjectFit**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

public:

Int_t	**fEmpty**	*Initialization flag (1=Minuit initialized*)
Int_t	**fMaxpar**	*Maximum number of parameters*
class TString*	**fCpnam**	*Array of parameters names*
Double_t*	**fU**	*External (visible to user in FCN*) value of parameters
Double_t*	**fAlim**	*Lower limits for parameters. If zero no limits*
Double_t*	**fBlim**	*Upper limits for parameters*
Double_t*	**fErp**	*Positive Minos errors if calculated*
Double_t*	**fErn**	*Negative Minos errors if calculated*
Double_t*	**fWerr**	*External parameters error (standard deviation, defined by UP*)
Double_t*	**fGlobcc**	*Global Correlation Coefficients*
Int_t*	**fNvarl**	*parameters flag (-1=undefined, 0=constant..)*
Int_t*	**fNiofex**	*Internal parameters number, or zero if not currently variable*
Int_t*	**fNexofi**	*External parameters number for currently variable parameters*
Double_t*	**fX**	*Internal parameters values*
Double_t*	**fXt**	*Internal parameters values X saved as Xt*
Double_t*	**fDirin**	*(Internal*) step sizes for current step
Double_t*	**fXs**	*Internal parameters values saved for fixed params*
Double_t*	**fXts**	*Internal parameters values X saved as Xt for fixed*

	params	
Double_t*	**fDirins**	*(Internal)* step sizes for current step for fixed params
Double_t*	**fGrd**	*First derivatives*
Double_t*	**fG2**	
Double_t*	**fGstep**	*Step sizes*
Double_t*	**fGin**	
Double_t*	**fDgrd**	*Uncertainties*
Double_t*	**fGrds**	
Double_t*	**fG2s**	
Double_t*	**fGsteps**	
Int_t*	**fIpfix**	*List of fixed parameters*
Int_t	**fNpfix**	*Number of fixed parameters*
Double_t*	**fVhmat**	*(Internal)* error matrix stored as Half MATrix, since it is symmetric
Double_t*	**fVthmat**	*VHMAT is sometimes saved in VTHMAT, especially in MNMNOT*
Double_t*	**fP**	
Double_t*	**fPstar**	
Double_t*	**fPstst**	
Double_t*	**fPbar**	
Double_t*	**fPrho**	*Minimum point of parabola*
Int_t	**fMaxint**	*Maximum number of internal parameters*
Int_t	**fNpar**	*Number of parameters*
Int_t	**fMaxext**	*Maximum number of external parameters*
Int_t	**fMaxIterations**	*Maximum number of iterations*
Int_t	**fNu**	
Int_t	**fIsysrd**	*standardInput unit*
Int_t	**fIsyswr**	*standard output unit*
Int_t	**fIsyssa**	
Int_t	**fNpagwd**	*Page width*
Int_t	**fNpagln**	*Number of lines per page*
Int_t	**fNewpag**	
Int_t	**fIstkrd**	
Int_t	**fNstkrd**	
Int_t	**fIstkwr**	
Int_t	**fNstkwr**	
class TString	**fCfrom**	
class TString	**fCstatu**	
class TString	**fCtitl**	
class TString	**fCword**	
class TString	**fCundef**	
class TString	**fCvrsn**	
class TString	**fCovmes**	
Int_t	**fISW**	*Array of switches*
Int_t	**fIdbg**	*Array of internal debug switches*
Int_t	**fNblock**	*Number of Minuit data blocks*
Int_t	**fIcomnd**	*Number of commands*
Double_t	**fAmin**	*Minimum value found for FCN*
Double_t	**fUp**	*FCN+-UP defines errors (for chisquare fits UP=1)*
Double_t	**fEDM**	*Estimated vertical distance to the minimum*
Double_t	**fFval3**	
Double_t	**fEpsi**	
Double_t	**fApsi**	
Double_t	**fDcovar**	*Relative change in covariance matrix*
Int_t	**fNfcn**	*Number of calls to FCN*
Int_t	**fNfcnmx**	*Maximum number of calls to FCN*
Int_t	**fNfcnlc**	

Int_t	**fNfcnfr**	
Int_t	**fItaur**	
Int_t	**fIstrat**	
Int_t	**fNwrmes**	
Double_t*	**fWord7**	
Bool_t	**fLwarn**	*true if warning messges are to be put out (default=true*)
Bool_t	**fLrepor**	*true if exceptional conditions are put out (default=false*)
Bool_t	**fLimset**	*true if a parameter is up against limits (for MINOS*)
Bool_t	**fLnolim**	*true if there are no limits on any parameters (not yet used*)
Bool_t	**fLnewmn**	*true if the previous process has unexpectedly improved FCN*
Bool_t	**fLphead**	*true if a heading should be put out for the next parameter definition*
Double_t	**fEpsmac**	*machine precision for floating points*:
Double_t	**fEpsma2**	*sqrt(fEpsmac*)
Double_t	**fVlimlo**	
Double_t	**fVlimhi**	
Double_t	**fUndefi**	*Undefined number = -54321*
Double_t	**fBigedm**	*Big EDM = 123456*
Double_t	**fUpdflt**	
Double_t*	**fXpt**	*X array of points for contours*
Double_t*	**fYpt**	*Y array of points for contours*
class TString*	**fChpt**	*Character to be plotted at the X, Y contour positions*
Double_t	**fXmidcr**	
Double_t	**fYmidcr**	
Double_t	**fXdircr**	
Double_t	**fYdircr**	
Int_t	**fKe1cr**	
Int_t	**fKe2cr**	
class TString*	**fOrigin**	
class TString*	**fWarmes**	
Int_t	**fNfcwar**	
Int_t	**fIcirc**	
class TObject*	**fObjectFit**	*Pointer to object being fitted*
class TMethodCall*	**fMethodCall**	*Pointer to MethodCall in case of interpreted function*

void (*)(Int_t &npar, Double_t *gin, Double_t &f, Double_t *u, Int_t *flag*)**fFCN**

Class Description

```
The Minimization package
This package was originally written in Fortran by Fred James
and part of PACKLIB (patch D506)

It has been converted to a C++ class  by R.Brun
The current implementation in C++ is a straightforward conversion
of the original Fortran version: The main changes are:

- The variables in the various Minuit labelled common blocks
  have been changed to the TMinuit class data members.
- The include file Minuit.h has been commented as much as possible
  using existing comments in the code or the printed documentation
- The original Minuit subroutines are now member functions.
- Constructors and destructor have been added.
- Instead of passing the FCN  function in the argument
  list, the addresses of this function is stored as pointer
  in the data members of the class. This is by far more elegant
  and flexible in an interactive environment.
```

```
  The member function SetFCN can be used to define this pointer.
- The ROOT static function Printf is provided to replace all
  format statements and to print on currently defined output file.
- The derived class TMinuitOld contains obsolete routines from
  the Fortran based version.
```

Basic concepts of MINUIT

The MINUIT package acts on a multiparameter Fortran function to which one must give the generic name `FCN`. In the ROOT implementation, the function `FCN` is defined via the MINUIT SetFCN member function when an Histogram.Fit command is invoked. The value of `FCN` will in general depend on one or more variable parameters.

To take a simple example, in case of ROOT histograms (classes TH1C,TH1S,TH1F,TH1D) the Fit function defines the Minuit fitting function as being H1FitChisquare or H1FitLikelihood depending on the options selected. H1FitChisquare calculates the chisquare between the user fitting function (gaussian, polynomial, user defined,etc) and the data for given values of the parameters. It is the task of MINUIT to find those values of the parameters which give the lowest value of chisquare.

Basic concepts - The transformation for parameters with limits.

For variable parameters with limits, MINUIT uses the following transformation:

```
P    = arcsin(2((P    -a)/(b- a))-1)              P     = a+((b- a)/(2))(sinP
   +1)
int             ext                               ext                        int
```

so that the internal value P can take on any value, while the external int value P can take on values only between the lower limit a and the ext upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the `FCN` value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if he puts limits of 10 (0,10) on a parameter, then the values 0.0 and 1. 0 will be indistinguishable to the accuracy of most machines.

The transformation also affects the parameter error matrix, of course, so MINUIT does a transformation of the error matrix (and the ''parabolic'' parameter errors) when there are parameter limits. Users should however realize that the transformation is only a linear approximation, and that it cannot give a meaningful result if one or more parameters is very close to a limit, where partial P /partial P #0. Therefore, it is ext int recommended that:

1. Limits on variable parameters should be used only when needed in order to prevent the parameter from taking on unphysical values.
2. When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits.

How to get the right answer from MINUIT.

MINUIT offers the user a choice of several minimization algorithms. The MIGRAD algorithm is in general the best minimizer for nearly all functions. It is a variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness. Its main weakness is that it depends heavily on knowledge of the first derivatives, and fails miserably if they are very inaccurate.

If parameter limits are needed, in spite of the side effects, then the user should be aware of the following techniques to alleviate problems caused by limits:

Getting the right minimum with limits.

If MIGRAD converges normally to a point where no parameter is near one of its limits, then the existence of limits has probably not prevented MINUIT from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimizer has become ''blocked'' at a limit. This may normally happen only if the parameter is so close to a limit (internal value at an odd multiple of #((pi)/(2)) that MINUIT prints a warning to this effect when it prints the parameter values.

The minimizer can become blocked at a limit, because at a limit the derivative seen by the minimizer partial F/ partial P is zero no matter int what the real derivative partial F/partial P is. ext

((partial F)/(partial P))= ((partial F)/(partial P))((partial P)/(partial P)) =((partial F)/(partial P))= 0 int ext ext int ext

Getting the right parameter errors with limits.

In the best case, where the minimum is far from any limits, MINUIT will correctly transform the error matrix, and the parameter errors it reports should be accurate and very close to those you would have got without limits. In other cases (which should be more common, since otherwise you wouldn't need limits), the very meaning of parameter errors becomes problematic. Mathematically, since the limit is an absolute constraint on the parameter, a parameter at its limit has no error, at least in one direction. The error matrix, which can assign only symmetric errors, then becomes essentially meaningless.

Interpretation of Parameter Errors:

There are two kinds of problems that can arise: the reliability of MINUIT's error estimates, and their statistical interpretation, assuming they are accurate.

Statistical interpretation:

For discussuion of basic concepts, such as the meaning of the elements of the error matrix, or setting of exact confidence levels (see [bib-EADIE]).

Reliability of MINUIT error estimates.

MINUIT always carries around its own current estimates of the parameter errors, which it will print out on request, no matter how accurate they are at any given point in the execution. For example, at initialization, these estimates are just the starting step sizes as specified by the user. After a HESSE step, the errors are usually quite accurate, unless there has been a problem. MINUIT, when it prints out error values, also gives some indication of how reliable it thinks they are. For example, those marked `CURRENT GUESS ERROR` are only working values not to be believed, and `APPROXIMATE ERROR` means that they have been calculated but there is reason to believe that they may not be accurate.

If no mitigating adjective is given, then at least MINUIT believes the errors are accurate, although there is always a small chance that MINUIT has been fooled. Some visible signs that MINUIT may have been fooled are:

3. Warning messages produced during the minimization or error analysis.
4. Failure to find new minimum.
5. Value of `EDM` too big (estimated Distance to Minimum).
6. Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others.
7. Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parameterized so that individual errors are not very meaningful because they are so highly correlated.
8. Parameter at limit. This condition, signalled by a MINUIT warning message, may make both the function minimum and parameter errors unreliable. See the discussion above "Getting the right parameter errors with limits".

The best way to be absolutely sure of the errors, is to use "independent" calculations and compare them, or compare the calculated errors with a picture of the function. Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum, although it may not be so for all points far away from the minimum, even for a well-determined physical problem. Therefore, if MIGRAD reports that it has found a non-positive-definite covariance matrix, this may be a sign of one or more of the following:

A non-physical region:

On its way to the minimum, MIGRAD may have traversed a region which has unphysical behaviour, which is of course not a serious problem as long as it recovers and leaves such a region.

An underdetermined problem:

If the matrix is not positive-definite even at the minimum, this may mean that the solution is not well-defined, for example that there are more unknowns than there are data points, or that the parameterization of the fit contains a linear dependence. If this is the case, then MINUIT (or any other program) cannot solve your problem uniquely, and the error matrix will necessarily be largely meaningless, so the user must remove the underdeterminedness by reformulating the parameterization. MINUIT cannot do this itself.

Numerical inaccuracies:

It is possible that the apparent lack of positive-definiteness is in fact only due to excessive roundoff errors in numerical calculations in the user function or not enough precision. This is unlikely in general, but becomes more likely if the number of free parameters is very large, or if the parameters are badly scaled (not all of the same order of magnitude), and correlations are also large. In any case, whether the non-positive-definiteness is real or only numerical is largely irrelevant, since in both cases the error matrix will be unreliable and the minimum suspicious.

An ill-posed problem:

For questions of parameter dependence, see the discussion above on positive-definiteness.

Possible other mathematical problems are the following:

Excessive numerical roundoff:

Be especially careful of exponential and factorial functions which get big very quickly and lose accuracy.

Starting too far from the solution:

The function may have unphysical local minima, especially at infinity in some variables.

TMinuit( ): **TNamed**("MINUIT", "The Minimization package")

```
Minuit normal constructor
```

TMinuit(Int_t *maxpar*): **TNamed**("MINUIT", "The Minimization package")

```
Minuit normal constructor
```

TMinuit(const TMinuit &)

```
Private TMinuit copy ctor. TMinuit can not be copied.
```

~**TMinuit**( )

```
Minuit default destructor
```

void **BuildArrays**(Int_t *maxpar*)

```
Create internal Minuit arrays for the maxpar parameters
```

void **DeleteArrays**( )

```
Delete internal Minuit arrays
```

void **SetFCN**(**void**(**fcn*)(Int_t &, Double_t *, Double_t &f, Double_t *, Int_t))

```
To set the address of the minimization function
```

void **SetFCN**(void **fcn*)

```
To set the address of the minimization function
  this function is called by CINT instead of the function above
```

void **mnamin**( )

```
Initialize AMIN
C        Called  from many places.  Initializes the value of AMIN by
C        calling the user function. Prints out the function value and
C        parameter values if Print Flag value is high enough.
```

void **mnbins**(Double_t *a1,* Double_t *a2,* Int_t *naa,* Double_t &*bl,* Double_t &*bh,* Int_t &*nb,* Double_t &*bwid*)

```
Compute reasonable histogram intervals
     Function TO DETERMINE REASONABLE HISTOGRAM INTERVALS
     GIVEN ABSOLUTE UPPER AND LOWER BOUNDS  A1 AND A2
     AND DESIRED MAXIMUM NUMBER OF BINS NAA
     PROGRAM MAKES REASONABLE BINNING FROM BL TO BH OF WIDTH BWID
```

```
          F. JAMES,   AUGUST, 1974 , stolen for Minuit, 1988
```

void **mncalf**(Double_t **pvec,* Double_t &*ycalf*)

```
     Transform FCN to find further minima
          Called only from MNIMPR.  Transforms the function FCN
          by dividing out the quadratic part in order to find further
          minima.    Calculates  ycalf = (f-fmin)/(x-xmin)*v*(x-xmin)
```

void **mncler**( )

```
     Resets the parameter list to UNDEFINED
          Called from MINUIT and by option from MNEXCM
```

void **mncntr**(Int_t *ke1,* Int_t *ke2,* Int_t &*ierrf*)

```
     Print function contours in two variables, on line printer
                  input arguments: parx, pary, devs, ngrid
```

void **mncomd**(TString *crdbin,* Int_t &*icondn*)

```
     Reads a command string and executes
          Called by user.  'Reads' a command string and executes.
       Equivalent to MNEXCM except that the command is given as a
            character string.

       ICONDN = 0: command executed normally
                1: command is blank, ignored
                2: command line unreadable, ignored
                3: unknown command, ignored
                4: abnormal termination (e.g., MIGRAD not converged)
                5: command is a request to read PARAMETER definitions
                6: 'SET INPUT' command
                7: 'SET TITLE' command
                8: 'SET COVAR' command
                9: reserved
               10: END command
               11: EXIT or STOP command
               12: RETURN command
```

void **mncont**(Int_t *ke1,* Int_t *ke2,* Int_t *nptu,* Double_t **xptu,* Double_t **yptu,* Int_t &*ierrf*)

```
     Find points along a contour where FCN is minimum
         Find NPTU points along a contour where the function
               FMIN (X(KE1),X(KE2)) =  AMIN+UP
         where FMIN is the minimum of FCN with respect to all
         the other NPAR-2 variable parameters (if any).
     IERRF on return will be equal to the number of points found:
       NPTU if normal termination with NPTU points found
       -1   if errors in the calling sequence (KE1, KE2 not variable)
        0   if less than four points can be found (using MNMNOT)
       n>3  if only n points can be found (n < NPTU)

                    input arguments: parx, pary, devs, ngrid
```

void **mncrck**(TString &*crdbuf,* Int_t *maxcwd,* TString &*comand,* Int_t &*lnc,*Int_t mxp, Double_t *plist, Int_t &llist, Int_t &ierr, Int_t *isyswr*)

```
     Cracks the free-format input
         Called from MNREAD.
         Cracks the free-format input, expecting zero or more
           alphanumeric fields (which it joins into COMAND(1:LNC))
           followed by one or more numeric fields separated by
           blanks and/or one comma.  The numeric fields are put into
           the LLIST (but at most MXP) elements of PLIST.
        IERR = 0 if no errors,
             = 1 if error(s).
        Diagnostic messages are written to ISYSWR
```

void **mncros**(Double_t *&aopt,* Int_t *&iercr*)

```
Find point where MNEVAL=AMIN+UP
    Find point where MNEVAL=AMIN+UP, along the line through
    XMIDCR,YMIDCR with direction XDIRCR,YDIRCR,   where X and Y
    are parameters KE1CR and KE2CR.  If KE2CR=0 (from MINOS),
    only KE1CR is varied.  From MNCONT, both are varied.
    Crossing point is at
     (U(KE1),U(KE2)) = (XMID,YMID) + AOPT*(XDIR,YDIR)
```

void **mncuve**( )

```
Makes sure that the current point is a local minimum
     Makes sure that the current point is a local
     minimum and that the error matrix exists,
     or at least something good enough for MINOS and MNCONT
```

void **mnderi**( )

```
Calculates the first derivatives of FCN (GRD)
     Calculates the first derivatives of FCN (GRD),
     either by finite differences or by transforming the user-
     supplied derivatives to internal coordinates,
     according to whether ISW(3) is zero or one.
```

void **mndxdi**(Double_t *pint,* Int_t *ipar,* Double_t *&dxdi*)

```
Calculates the transformation factor between ext/internal values
     calculates the transformation factor between external and
     internal parameter values.     this factor is one for
     parameters which are not limited.     called from MNEMAT.
```

void **mneig**(Double_t **a,* Int_t *ndima,* Int_t *n,* Int_t *mits,* Double_t **work,* Double_t *precis,* Int_t *&ifault*)

```
Compute matrix eigen values
```

void **mnemat**(Double_t **emat,* Int_t *ndim*)

```
Calculates the external error matrix from the internal matrix
     Calculates the external error matrix from the internal
     to be called by user.
```

void **mnerrs**(Int_t *number,* Double_t *&eplus,* Double_t *&eminus,* Double_t *&eparab,* Double_t *&gcc*)

```
Utility routine to get MINOS errors
 Called by user.
 NUMBER is the parameter number
 values returned by MNERRS:
    EPLUS, EMINUS are MINOS errors of parameter NUMBER,
    EPARAB is 'parabolic' error (from error matrix).
              (Errors not calculated are set = 0)
    GCC is global correlation coefficient from error matrix
```

void **mneval**(Double_t *anext,* Double_t *&fnext,* Int_t *&ierev*)

```
Evaluates the function being analyzed by MNCROS
   Evaluates the function being analyzed by MNCROS, which is
   generally the minimum of FCN with respect to all remaining
   variable parameters.  The class data members contains the
   data necessary to know the values of U(KE1CR) and U(KE2CR)
   to be used, namely     U(KE1CR) = XMIDCR + ANEXT*XDIRCR
   and (if KE2CR .NE. 0)  U(KE2CR) = YMIDCR + ANEXT*YDIRCR
```

void **mnexcm**(TString *comand,* Double_t **plist,* Int_t *llist,* Int_t *&ierflg*)

```
Interprets a command and takes appropriate action
     either directly by skipping to the corresponding code in
     MNEXCM, or by setting up a call to a function

recognized MINUIT commands:
obsolete commands:
   IERFLG is now (94.5) defined the same as ICONDN in MNCOMD
```

```
= 0: command executed normally
  1: command is blank, ignored
  2: command line unreadable, ignored
  3: unknown command, ignored
  4: abnormal termination (e.g., MIGRAD not converged)
  9: reserved
 10: END command
 11: EXIT or STOP command
 12: RETURN command
```

void **mnexin**(Double_t **pint*)

```
Transforms the external parameter values U to internal values
     Transforms the external parameter values U to internal
     values in the dense array PINT.
```

void **mnfixp**(Int_t *iint1,* Int_t *&ierr*)

```
Removes parameter IINT from the internal parameter list
     and arranges the rest of the list to fill the hole.
```

void **mnfree**(Int_t *k*)

```
Restores one or more fixed parameter(s) to variable status
     Restores one or more fixed parameter(s) to variable status
     by inserting it into the internal parameter list at the
     appropriate place.

     K = 0 means restore all parameters
     K = 1 means restore the last parameter fixed
     K = -I means restore external parameter I (if possible)
     IQ = fix-location where internal parameters were stored
     IR = external number of parameter being restored
     IS = internal number of parameter being restored
```

void **mngrad**( )

```
Interprets the SET GRAD command
    Called from MNSET
    Interprets the SET GRAD command, which informs MINUIT whether
    the first derivatives of FCN will be calculated by the user
    inside FCN.  It can check the user derivative calculation
    by comparing it with a finite difference approximation.
```

void **mnhelp**(TString *comd*)

```
HELP routine for MINUIT interactive commands
   COMD ='*   '  prints a global help for all commands
   COMD =Command_name: print detailed help for one command.
      Note that at least 3 characters must be given for the command
      name.

  Author: René Brun
          comments extracted from the MINUIT documentation file.

command name ASSUMED to be in upper case
```

void **mnhess**( )

```
Calculates the full second-derivative matrix of FCN
     by taking finite differences. When calculating diagonal
     elements, it may iterate so that step size is nearly that
     which gives function change= UP/10. The first derivatives
     of course come as a free side effect, but with a smaller
     step size in order to obtain a known accuracy.
```

void **mnhes1**( )

```
Calculate first derivatives (GRD) and uncertainties (DGRD)
      and appropriate step sizes GSTEP
```

```
    Called from MNHESS and MNGRAD
```

void **mnimpr**( )

```
Attempts to improve on a good local minimum
     Attempts to improve on a good local minimum by finding a
     better one.   The quadratic part of FCN is removed by MNCALF
     and this transformed function is minimized using the simplex
     method from several random starting points.
     ref. -- Goldstein and Price, Math.Comp. 25, 569 (1971)
```

void **mninex**(Double_t **pint*)

```
Transforms from internal coordinates (PINT) to external (U)
     The minimizing routines which work in
     internal coordinates call this routine before calling FCN.
```

void **mninit**(Int_t *i1,* Int_t *i2,* Int_t *i3*)

```
Main initialization member function for MINUIT
  It initializes some constants
           (including the logical I/O unit nos.),
```

void **mnlims**( )

```
Interprets the SET LIM command, to reset the parameter limits
    Called from MNSET
```

void **mnline**(Double_t **start,* Double_t *fstart,* Double_t **step,* Double_t *slope,* Double_t *toler*)

```
Perform a line search from position START
     along direction STEP, where the length of vector STEP
                gives the expected position of minimum.
     FSTART is value of function at START
     SLOPE (if non-zero) is df/dx along STEP at START
     TOLER is initial tolerance of minimum in direction STEP

SLAMBG and ALPHA control the maximum individual steps allowed.
The first step is always =1. The max length of second step is SLAMBG.
The max size of subsequent steps is the maximum previous successful
step multiplied by ALPHA + the size of most recent successful step,
but cannot be smaller than SLAMBG.
```

void **mnmatu**(Int_t *kode*)

```
Prints the covariance matrix v when KODE=1
     always prints the global correlations, and
     calculates and prints the individual correlation coefficients
```

void **mnmigr**( )

```
Performs a local function minimization
     Performs a local function minimization using basically the
     method of Davidon-Fletcher-Powell as modified by Fletcher
     ref. -- Fletcher, Comp.J. 13,317 (1970)   "switching method"
```

void **mnmnos**( )

```
Performs a MINOS error analysis
     Performs a MINOS error analysis on those parameters for
     which it is requested on the MINOS command by calling
     MNMNOT for each parameter requested.
```

void **mnmnot**(Int_t *ilax,* Int_t *ilax2,* Double_t *&val2pl,* Double_t *&val2mi*)

```
Performs a MINOS error analysis on one parameter
     The parameter ILAX is varied, and the minimum of the
     function with respect to the other parameters is followed
     until it crosses the value FMIN+UP.
```

void **mnparm**(Int_t *k1,* TString *cnamj,* Double_t *uk,* Double_t *wk,* Double_t *a,* Double_t *b,* Int_t *&ierflg*)

```
Implements one parameter definition
```

```
    Called from MNPARS and user-callable
 Implements one parameter definition, that is:
      K     (external) parameter number
      CNAMK parameter name
      UK    starting value
      WK    starting step size or uncertainty
      A, B  lower and upper physical parameter limits
 and sets up (updates) the parameter lists.
 Output: IERFLG=0 if no problems
               >0 if MNPARM unable to implement definition
```

void **mnpars**(TString &*crdbuf,* Int_t &*icondn*)

```
Implements one parameter definition
     Called from MNREAD and user-callable
 Implements one parameter definition, that is:
    parses the string CRDBUF and calls MNPARM

output conditions:
     ICONDN = 0    all OK
     ICONDN = 1    error, attempt to define parameter is ignored
     ICONDN = 2    end of parameter definitions
```

void **mnpfit**(Double_t **parx2p,* Double_t **pary2p,* Int_t *npar2p,* Double_t **coef2p,* Double_t &*sdev2p*)

```
To fit a parabola to npar2p points
npar2p   no. of points
parx2p(i)   x value of point i
pary2p(i)   y value of point i

coef2p(1...3)  coefficients of the fitted parabola
y=coef2p(1) + coef2p(2)*x + coef2p(3)*x**2
sdev2p= variance
method : chi**2 = min equation solved explicitly
```

void **mnpint**(Double_t &*pexti,* Int_t *i1,* Double_t &*pinti*)

```
Calculates the internal parameter value PINTI
     corresponding  to the external value PEXTI for parameter I.
```

void **mnplot**(Double_t **xpt,* Double_t **ypt,* TString **chpt,* Int_t *nxypt,* Int_t *nunit,* Int_t *npagwd,* Int_t *npagln*)

```
Plots points in array xypt onto one page with labelled axes
     NXYPT is the number of points to be plotted
     XPT(I) = x-coord. of ith point
     YPT(I) = y-coord. of ith point
     CHPT(I) = character to be plotted at this position
     the input point arrays XPT, YPT, CHPT are destroyed.
```

void **mnpout**(Int_t *iuext1,* TString *chnam,* Double_t &*val,* Double_t &*err,* Double_t &*xlolim,* Double_t &*xuplim,* Int_t &*iuint*)

```
Provides the user with information concerning the current status
       of parameter number IUEXT. Namely, it returns:
     CHNAM: the name of the parameter
     VAL: the current (external) value of the parameter
     ERR: the current estimate of the parameter uncertainty
     XLOLIM: the lower bound (or zero if no limits)
     XUPLIM: the upper bound (or zero if no limits)
     IUINT: the internal parameter number (or zero if not variable,
        or negative if undefined).
Note also:  If IUEXT is negative, then it is -internal parameter
        number, and IUINT is returned as the EXTERNAL number.
  Except for IUINT, this is exactly the inverse of MNPARM
  User-called
```

void **mnprin**(Int_t *inkode,* Double_t *fval*)

```
Prints the values of the parameters at the time of the call
```

```
     also prints other relevant information such as function value,
     estimated distance to minimum, parameter errors, step sizes.

      According to the value of IKODE, the printout is:
 IKODE=INKODE= 0    only info about function value
               1    parameter values, errors, limits
               2    values, errors, step sizes, internal values
               3    values, errors, step sizes, first derivs.
               4    values, parabolic errors, MINOS errors
 when INKODE=5, MNPRIN chooses IKODE=1,2, or 3, according to ISW(2)
```

void **mnpsdf**( )

```
Calculates the eigenvalues of v to see if positive-def
     if not, adds constant along diagonal to make positive.
```

void **mnrazz**(Double_t *ynew,* Double_t **pnew,* Double_t **y,* Int_t &*jh,* Int_t &*jl*)

```
Called only by MNSIMP (and MNIMPR) to add a new point
     and remove an old one from the current simplex, and get the
     estimated distance to minimum.
```

void **mnrn15**(Double_t &*val,* Int_t &*inseed*)

```
This is a super-portable random number generator
      It should not overflow on any 32-bit machine.
      The cycle is only ~10**9, so use with care!
      Note especially that VAL must not be undefined on input.
                 Set Default Starting Seed
```

void **mnrset**(Int_t *iopt*)

```
Resets function value and errors to UNDEFINED
 If IOPT=1,
 If IOPT=0, sets only MINOS errors to undefined
     Called from MNCLER and whenever problem changes, for example
     after SET LIMITS, SET PARAM, CALL FCN 6
```

void **mnsave**( )

```
Writes current parameter values and step sizes onto file ISYSSA
       in format which can be reread by Minuit for restarting.
    The covariance matrix is also output if it exists.
```

void **mnscan**( )

```
Scans the values of FCN as a function of one parameter
     and plots the resulting values as a curve using MNPLOT.
     It may be called to scan one parameter or all parameters.
     retains the best function and parameter values found.
```

void **mnseek**( )

```
Performs a rough (but global) minimization by monte carlo search
     Each time a new minimum is found, the search area is shifted
     to be centered at the best value.  Random points are chosen
     uniformly over a hypercube determined by current step sizes.
The Metropolis algorithm accepts a worse point with probability
   exp(-d/UP), where d is the degradation.  Improved points
   are of course always accepted.  Actual steps are random
   multiples of the nominal steps (DIRIN).
```

void **mnset**( )

```
Interprets the commands that start with SET and SHOW
     Called from MNEXCM
     file characteristics for SET INPUT
    'SET ' or 'SHOW',  'ON ' or 'OFF', 'SUPPRESSED' or 'REPORTED  '
     explanation of print level numbers -1:3  and strategies 0:2
     identification of debug options
     things that can be set or shown
```

```
        options not intended for normal users
```

void **mnsimp**( )

```
    Minimization using the simplex method of Nelder and Mead
        Performs a minimization using the simplex method of Nelder
        and Mead (ref. -- Comp. J. 7,308 (1965)).
```

void **mntiny**(Double_t *epsp1,* Double_t *&epsbak*)

```
    To find the machine precision
        Compares its argument with the value 1.0, and returns
        the value .TRUE. if they are equal.  To find EPSMAC
        safely by foiling the Fortran optimizer
```

Bool_t **mnunpt**(TString *&cfname*)

```
    Returns .TRUE. if CFNAME contains unprintable characters
```

void **mnvert**(Double_t **a,* Int_t *l,* Int_t *m,* Int_t *n,* Int_t *&ifail*)

```
    Inverts a symmetric matrix
        inverts a symmetric matrix.   matrix is first scaled to
        have all ones on the diagonal (equivalent to change of units)
        but no pivoting is done since matrix is positive-definite.
```

void **mnwarn**(const char **copt1,* const char **corg1,* const char **cmes1*)

```
    Prints Warning messages
      If COPT='W', CMES is a WARning message from CORG.
      If COPT='D', CMES is a DEBug message from CORG.
          If SET WARnings is in effect (the default), this routine
              prints the warning message CMES coming from CORG.
          If SET NOWarnings is in effect, the warning message is
              stored in a circular buffer of length MAXMES.
          If called with CORG=CMES='SHO', it prints the messages in
              the circular buffer, FIFO, and empties the buffer.
```

void **mnwerr**( )

```
    Calculates the WERR, external parameter errors
       and the global correlation coefficients, to be called
       whenever a new covariance matrix is available.
```

Author: René Brun, Frederick James 12/08/95
Last update: 03/11/97 09.54.51 by René Brun

TMixture

class **TMixture** : public TMaterial

public:

class TMixture	**TMixture**( )
class TMixture	**TMixture**(Text_t* *name,* Text_t* *title,* Int_t *nmixt*)
class TMixture	**TMixture**(class TMixture&)
void	**~TMixture**( )
class TClass*	**Class**( )
virtual void	**DefineElement**(Int_t *n,* Float_t *a,* Float_t *z,* Float_t *w*)
Float_t*	**GetAmixt**( )
virtual Int_t	**GetNumber**( )
Float_t*	**GetWmixt**( )
Float_t*	**GetZmixt**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNmixt**	*Number of elements in mixture*
Float_t*	**fAmixt**	*Array of A of mixtures*
Float_t*	**fZmixt**	*Array of Z of mixtures*
Float_t*	**fWmixt**	*Array of relative weights*

Class Description

```
Manages a detector mixture. See class TGeometry.
```

TMixture( )

```
Mixture default constructor
```

TMixture(Text_t **name,* Text_t **title,* Int_t *nmixt*):**TMaterial**(*name, title, 0, 0, 0*)

```
Mixture normal constructor
    Defines mixture OR COMPOUND as composed by
    the basic nmixt materials defined later by DefineElement.

    If nmixt > 0 then Wmixt contains the PROPORTION BY WEIGHTS
    of each basic material in the mixture.

    If nmixt < 0 then Wmixt contains the number of atoms
    of a given kind into the molecule of the COMPOUND
    In this case, Wmixt is changed to relative weights.

    nb : the radiation length is computed according
         the EGS manual slac-210 uc-32 June-78
                        formula  2-6-8 (37)
```

~TMixture( )

```
Mixture default destructor
```

void **DefineElement**(Int_t *n,* Float_t *a,* Float_t *z,* Float_t *w*)

```
Define one mixture element
```

void **Streamer**(TBuffer *&b*)

```
Stream a class object
```

Author: René Brun 03/10/95

Last update: 1.02/00 21/07/97 17.29.46 by Fons Rademakers

TMonitor

class **TMonitor** : public TObject

public:

class TMonitor	**TMonitor**( )
class TMonitor	**TMonitor**(class TMonitor&)
void	**~TMonitor**( )
void	**Activate**(class TSocket* *sock*)
void	**ActivateAll**( )
void	**Add**(class TSocket* *sock*)
class TClass*	**Class**( )
void	**DeActivate**(class TSocket* *sock*)
void	**DeActivateAll**( )
Int_t	**GetActive**( )
Int_t	**GetDeActive**( )
virtual class TClass*	**IsA**( )
void	**Remove**(class TSocket* *sock*)
class TSocket*	**Select**(Long_t *timeout*)
class TSocket*	**Select**( )
void	**SetReady**(class TSocket* *sock*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TList*	**fActive**	*list of sockets to monitor*
class TList*	**fDeActive**	*list of (temporary*) disabled sockets
class TSocket*	**fReady**	*socket which is ready to be read*

Class Description

```
TMonitor

This class monitors activity on a number of network sockets.
The actual monitoring is done by TSystem::DispatchOneEvent().
Typical usage: create a TMonitor object. Register a number of
TSocket objects and call TMonitor::Select(). Select() returns the
socket object which has data waiting. TSocket objects can be added,
removed, (temporary) enabled or disabled.
```

TMonitor( )

```
Create a monitor object.
```

~TMonitor( )

```
Cleanup the monitor object. Does not delete socket being monitored.
```

void **Add**(TSocket **sock*)

```
Add socket to the monitor's active list.
```

void **Remove**(TSocket **sock*)

```
Remove a socket from the monitor.
```

void **Activate**(TSocket **sock*)

```
Activate a de-activated socket.
```

void **ActivateAll**( )

```
Activate all de-activated sockets.
```

void **DeActivate**(TSocket **sock*)
```
De-activate a socket.
```

void **DeActivateAll**( )
```
De-activate all activated sockets.
```

TSocket* **Select**( )
```
Return pointer to socket for which an event is waiting.
```

TSocket* **Select**(Long_t *timeout*)
```
Return pointer to socket for which an event is waiting. Wait a maximum
of timeout milliseconds. If return is due to timeout it returns
(TSocket *)-1.
```

void **SetReady**(TSocket **sock*)
```
Called by TSocketHandler::Notify() to signal which socket is ready
to be read. User should not call this routine. The ready socket will
be returned via the Select() user function.
```

Int_t **GetActive**( ) const
```
Return number of sockets in the active list.
```

Int_t **GetDeActive**( ) const
```
Return number of sockets in the de-active list.
```

Author: Fons Rademakers 09/01/97
Last update: 1.00/06 23/03/97 11.34.51 by René Brun

TMotifApplication

class **TMotifApplication** : public TComponent, public TApplicationImp

protected:

class TMotifApplication	**TMotifApplication**( )

public:

class TMotifApplication	**TMotifApplication**(class TMotifApplication&)
class TMotifApplication	**TMotifApplication**(char* *appClassName,* int* *argc,* char** *argv,* void* *options = 0,* int *numOptions = 0*)
void	**~TMotifApplication**( )
class TDialogManager*	**AboutDialog**( )
XtAppContext	**AppContext**( )
char*	**ApplicationClassName**( )
virtual char*	**ApplicationName**( )
Widget	**BaseWidget**( )
virtual void	**Busy**(char* *msg = NULL,* class TSimpleWindow* *window = NULL*)
virtual Cursor	**BusyCursor**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
Display*	**Disp**( )
virtual void	**HandlePendingEvents**( )
virtual void	**HandleRawEvent**(XEvent* *event*)
virtual Bool_t	**HandleXInput**( )
virtual void	**Hide**( )
virtual void	**Iconify**( )
virtual void	**Init**( )
virtual class TClass*	**IsA**( )
virtual void	**Lower**( )
class TSimpleWindow*	**MainWindow**( )
virtual Cursor	**NormalCursor**( )
virtual void	**NotBusy**( )
virtual void	**Open**( )
virtual void	**Progressing**(char* *msg = NULL*)
virtual void	**QuitYourself**( )
virtual void	**Raise**( )
void	**RegisterWindow**(class TSimpleWindow* *win*)
void	**SetAboutDialog**(class TDialogManager* *dialog*)
void	**SetBusyCursor**((*unknown*) TCursorList* animatedCursor)
void	**SetBusyCursor**(Cursor *c*)
void	**SetBusyDialog**(class TBusyDialog* *dialog*)
void	**SetMainWindow**(class TSimpleWindow* *window*)
void	**SetNormalCursor**(Cursor *c*)
char*	**ShellGeometry**( )
virtual void	**Show**( )
void	**ShowCursor**(Cursor *c*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**StartupIconified**(Boolean *flag*)
Boolean	**StartupIconified**( )
virtual void	**Streamer**(class TBuffer& *b*)
void	**UnregisterWindow**(class TSimpleWindow* *win*)

Data Members

private:

XtAppContext	**fTheContext**
Display*	**fTheDisplay**

char*	**fAppName**	
char*	**fAppClassName**	
class TSimpleWindow*	**fMainWindow**	
Cursor	**fCurrentCursor**	
Cursor	**fBusyCursor**	
class TBusyDialog*	**fBusyDialog**	
class TDialogManager*	**fAboutDialog**	
ECursorType	**fCursorType**	
class TCursorList*	**fBusyCursorList**	
int	**fBusyDepth**	
Boolean	**fStartupIconified**	
Boolean	**fDumpResourceFile**	
static String	**fgDefaultResources**	
static XtResource	**fgResourceList**	
protected:		
class TComponentList	**fWinList**	*!List of windows in application*

Class Description

```
TMotifApplication

This class creates the Motif version of the ROOT application
environment.
```

void **UnregisterWindow**(TSimpleWindow **win*)

Author: Fons Rademakers 30/03/95
Last update: 1.02/00 22/08/97 11.38.49 by Fons Rademakers

TMotifBrowser

class **TMotifBrowser** : public TBrowserImp, public TMenuWindow

protected:

class TMotifBrowser	**TMotifBrowser**( )
void	**Chdir**(ListTreeItem* *item*)
void	**CloseCB**(*Widget, XtPointer, XtPointer*)
void	**CreateWindow**( )
void	**HandleIconBoxInput**(*XtPointer*)
void	**HandleIconBoxInputCB**(*Widget, XtPointer, XtPointer*)
void	**HandlerCB**(Widget *w,* XtPointer *cd,* XEvent* *e,* Boolean* *c*)
void	**HelpAboutCB**(*Widget, XtPointer, XtPointer*)
void	**HelpBrowserCB**(*Widget, XtPointer, XtPointer*)
void	**ListTreeActivate**(XtPointer *cd*)
void	**ListTreeActivateCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)
void	**ListTreeHighlight**(XtPointer *cd*)
void	**ListTreeHighlightCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)
void	**ListTreeMenu**(XtPointer *cd*)
void	**ListTreeMenuCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)
void	**NewCB**(*Widget, XtPointer, XtPointer*)
void	**Options**(Char_t* *item*)
void	**OptionsCB**(*Widget, XtPointer, XtPointer*)
void	**OptionsShowAs**(Char_t* *item*)
void	**OptionsShowAsCB**(*Widget, XtPointer, XtPointer*)
void	**OptionsSortBy**(Char_t* *item*)
void	**OptionsSortByCB**(*Widget, XtPointer, XtPointer*)
void	**PopupMenu**(Int_t *x,* Int_t *y*)
void	**PrintCB**(*Widget, XtPointer, XtPointer*)
virtual void	**ResizeWindow**(XConfigureEvent* *ev*)
void	**ResizeWindowCB**(Widget *w,* XConfigureEvent* *ev,* char**, int*)
void	**RootExec**(char* *cmd*)
void	**SaveAsCB**(*Widget, XtPointer, XtPointer*)
void	**SaveCB**(*Widget, XtPointer, XtPointer*)

public:

class TMotifBrowser	**TMotifBrowser**(class TBrowser* *b,* char* *title,* UInt_t *width,* UInt_t *height*)
class TMotifBrowser	**TMotifBrowser**(class TBrowser* *b,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TMotifBrowser	**TMotifBrowser**(class TMotifBrowser&)
void	**~TMotifBrowser**( )
void	**Add**(class TObject* *obj,* char* *name = 0*)
void	**AddToBox**(class TObject* *obj,* char* *name*)
void	**AddToTree**(class TObject* *obj,* char* *name*)
class TClass*	**Class**( )
const char*	**ClassName**( )
void	**Iconify**( )
virtual class TClass*	**IsA**( )
void	**RecursiveRemove**(class TObject* *obj*)
void	**Refresh**( )
void	**ResizeBrowser**( )
void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Widget	**fForm**	*Form containing lists and iconbox*
Widget	**fObjScrolledWindow**	*Window containing iconbox*
Widget	**fObjBox**	*Iconbox containing objects*
Widget	**fListTree**	*ListTree window*
ListTreeItem*	**fListLevel**	*Current ListTree level*
UInt_t	**fBwidth**	*Width of fScrolledWindow in pixels*
UInt_t	**fBheight**	*Height of fScrolledWindow in pixels*
Bool_t	**fTreeLock**	*True when we want to lock ListTree*

protected:

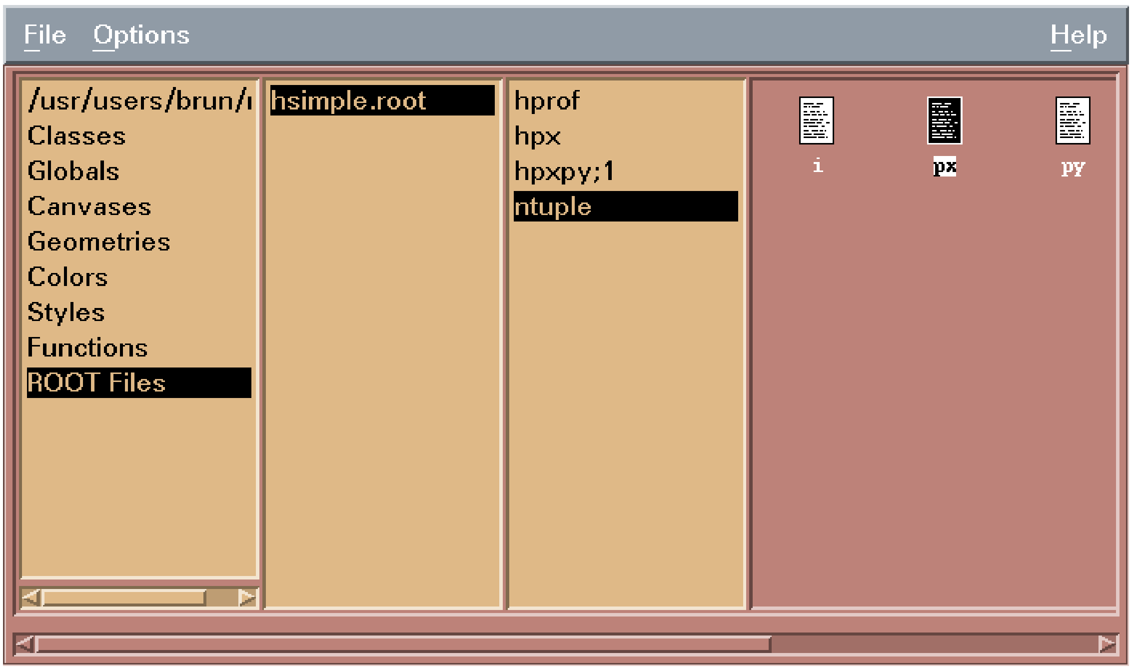
static MenuDesc_t	**fgMenuPane**
static MenuDesc_t	**fgFilePane**
static MenuDesc_t	**fgOptionsPane**
static MenuDesc_t	**fgShowAsPane**
static MenuDesc_t	**fgSortByPane**
static MenuDesc_t	**fgHelpPane**
static String	**fgDefaultResources**

public:

static const enum	**kListDefaultWidth**
static const enum	**kObjWindowMinSize**

Class Description

```
TMotifBrowser

This class creates a main window with menubar, scrollbars and a
list and a drawing area.
The list displays all browsable ROOT classes and the drawing area
contains the objects in the browsable classes. Selecting an object
can create a new list and show the contents of the object in the
drawing area, and so on.
The picture below shows a MotifBrowser on a ROOT file containing
histograms and ntuples.
```



TMotifBrowser(TBrowser **b,* const char **title,* UInt_t *width,* UInt_t *height*): **TBrowserImp**(*b*), **TMenuWindow**(title)

```
Create browser with a specified width and height.
```

TMotifBrowser(TBrowser **b,* const char **title,* Int_t *x,* Int_t *y,*UInt_t width, UInt_t *height*): **TBrowserImp**(*b*), **TMenuWindow**(title)

    Create browser with a specified width and height and at position x, y.

~TMotifBrowser( )

    Browser destructor.

void **Add**(TObject **obj,* const char **name*)

    Add items to the browser. This function has to be called
    by the Browse() member function of objects when they are
    called by a browser.

void **AddToBox**(TObject **obj,* const char **name*)

    Add items to the iconbox of the browser.

void **AddToTree**(TObject **obj,* const char **name*)

    Add items to the current ListTree of the browser.

void **CreateWindow**( )

    Create the actual browser window.

void **HandleIconBoxInput**(XtPointer *cd*)

    Handle iconbox events.

void **Iconify**( )

    Iconify the browser.

void **OptionsShowAs**(const Char_t **item*)

    Set icon type, either Big Icons, Small Icons or Text

void **OptionsSortBy**(const Char_t **item*)

    Set sort type, either by name or by type

void **PopupMenu**(Int_t *x,* Int_t *y*)

    It displays ContextMenu for selected icon

void **Refresh**( )

    Refresh the browser.

void **ResizeBrowser**( )

    Resize the iconbox (lists have fixed width).

void **ResizeWindow**(XConfigureEvent **ev*)

    Handle scrolledwindow resize events.

void **RootExec**(const char **cmd*)

    Process ROOT commands.

void **Show**( )

    Show (i.e. popup) the browser.

void **CloseCB**(*Widget,* XtPointer *cd, XtPointer*)

    Static callback for menu item: Close.

void **HandleIconBoxInputCB**(*Widget,* XtPointer *cd,* XtPointer *cbs*)

    Static callback to handle iconbox events.

void **HandlerCB**(Widget *w,* XtPointer *clientData,* XEvent **event,* Boolean **b*)

    Static ContextMenu handler

void **ListTreeActivateCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)

    Static callback to handle ListTree events.

void **ListTreeHighlightCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)
```
Static callback to handle ListTree events.
```

void **ListTreeMenuCB**(Widget *w,* XtPointer *cd,* XtPointer *cbs*)
```
Static callback to handle ListTree events.
```

void **NewCB**(*Widget, XtPointer, XtPointer*)
```
Static callback for menu item: New.
```

void **OptionsCB**(Widget *w,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: Options->Show As...
```

void **OptionsShowAsCB**(Widget *w,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: Options->Show As...
```

void **OptionsSortByCB**(Widget *w,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: SortBy.
```

void **PrintCB**(*Widget,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: Print.
```

void **ResizeWindowCB**(Widget *w,* XConfigureEvent **ev,* char **, int *)
```
Static callback to handle scrolledwindow resize events.
```

void **SaveCB**(*Widget,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: Save.
```

void **SaveAsCB**(*Widget,* XtPointer *cd, XtPointer*)
```
Static callback for menu item: SaveAs.
```

void **HelpBrowserCB**(*Widget,* XtPointer *cd, XtPointer*)

Author: Fons Rademakers 25/10/95
Last update: 1.03/04 26/09/97 18.40.12 by Fons Rademakers

TMotifButton

class **TMotifButton** : public TDialogItem

protected:
class TMotifButton **TMotifButton**( )
public:
class TMotifButton **TMotifButton**(char* *name*, Bool_t *isGadget = 1*, Text_t* *initData = NULL*)
class TMotifButton **TMotifButton**(class TMotifButton&)
void **~TMotifButton**( )
class TClass* **Class**( )
virtual const char* **ClassName**( )
void **DefaultCB**(Widget *w*, XtPointer *cd*, XtPointer *call_data*)
virtual enum EDialogItemType**DialogItemType**( )
virtual Widget **Instantiate**(Widget *parent*, Arg* *args = 0*, Int_t *numargs = 0*)
virtual class TClass* **IsA**( )
virtual Bool_t **IsGadget**( )
virtual void **SetLabel**(char* *str*)
virtual void **ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void **Streamer**(class TBuffer& *b*)

Data Members

private:
Bool_t **fIsGadget**
public:
XtCallbackProc **fActionCallback**
XtPointer **fClientData**

Class Description

```
TMotifButton

A Motif PushButton widget used in TDialogs.
```

void **DefaultCB**(Widget *w*, XtPointer *cd*, XtPointer *call_data*)

Author: Piotr Golonka 19/08/97
Last update: 1.03/04 19/09/97 16.47.07 by Fons Rademakers

TMotifCanvas

class **TMotifCanvas** : public TCanvasImp, public TMenuWindow

protected:

class TMotifCanvas	**TMotifCanvas**( )
virtual void	**AfterRealizeHook**( )
void	**AutoFitCanvasCB**(*Widget, XtPointer, XtPointer*)
void	**BrowserCB**(*Widget, XtPointer, XtPointer*)
void	**CanEditHistogramsCB**(*Widget, XtPointer, XtPointer*)
void	**ClearCanvasCB**(*Widget,* const *XtPointer, XtPointer*)
void	**ClearPadCB**(*Widget, XtPointer, XtPointer*)
void	**CloseCB**(*Widget, XtPointer, XtPointer*)
void	**ColorsCB**(*Widget, XtPointer, XtPointer*)
virtual void	**CreateWindow**( )
void	**EditorCB**(*Widget, XtPointer, XtPointer*)
void	**EventStatusCB**(*Widget, XtPointer, XtPointer*)
virtual void	**ExposeOrResize**(*XtPointer*)
void	**ExposeOrResizeCB**(*Widget, XtPointer, XtPointer*)
void	**FitCanvasCB**(*Widget, XtPointer, XtPointer*)
void	**FontsCB**(*Widget, XtPointer, XtPointer*)
void	**FullTreeCB**(*Widget, XtPointer, XtPointer*)
virtual void	**HandleEnter**( )
void	**HandleEnterCB**(*Widget, XtPointer,* XEvent*)
virtual void	**HandleInput**(*XtPointer*)
void	**HandleInputCB**(*Widget, XtPointer, XtPointer*)
virtual void	**HandleMove**(XEvent*)
void	**HandleMoveCB**(*Widget, XtPointer,* XEvent*)
virtual void	**HandleWMDeleteMessage**( )
virtual void	**HandleWMQuitMessage**( )
void	**HelpAboutCB**(*Widget, XtPointer, XtPointer*)
void	**HelpBrowserCB**(*Widget, XtPointer, XtPointer*)
void	**HelpCanvasCB**(*Widget, XtPointer, XtPointer*)
void	**HelpGraphicsEditorCB**(*Widget, XtPointer, XtPointer*)
void	**HelpObjectsCB**(*Widget, XtPointer, XtPointer*)
void	**HelpPostscriptCB**(*Widget, XtPointer, XtPointer*)
void	**HelpPullDownMenusCB**(*Widget, XtPointer, XtPointer*)
void	**IconifyCB**(*Widget, XtPointer, XtPointer*)
void	**MarkersCB**(*Widget, XtPointer, XtPointer*)
virtual void	**NewCanvas**( )
void	**NewCB**(*Widget, XtPointer, XtPointer*)
void	**OpenCB**(*Widget, XtPointer, XtPointer*)
void	**OpenGLViewCB**(*Widget, XtPointer, XtPointer*)
void	**OptFitCB**(*Widget, XtPointer, XtPointer*)
void	**OptStatCB**(*Widget, XtPointer, XtPointer*)
void	**OptTitleCB**(*Widget, XtPointer, XtPointer*)
void	**PartialTreeCB**(*Widget, XtPointer, XtPointer*)
void	**PrintCB**(*Widget, XtPointer, XtPointer*)
void	**QuitCB**(*Widget, XtPointer, XtPointer*)
void	**RefreshCB**(*Widget, XtPointer, XtPointer*)
virtual void	**ResizeWindow**(XConfigureEvent* *ev*)
void	**ResizeWindowCB**(Widget *w,* XConfigureEvent* *ev,* char**, int*)
virtual void	**RootExec**(char* *cmd*)
void	**ROOTInspectCB**(*Widget, XtPointer, XtPointer*)
void	**SaveAsCB**(*Widget, XtPointer, XtPointer*)
void	**SaveCB**(*Widget, XtPointer, XtPointer*)
void	**SaveGIFCB**(*Widget, XtPointer, XtPointer*)

void	**SaveROOTCB**(*Widget, XtPointer, XtPointer*)
void	**SaveSourceCB**(*Widget, XtPointer, XtPointer*)
void	**UndoCB**(*Widget, XtPointer, XtPointer*)
void	**X3DViewCB**(*Widget, XtPointer, XtPointer*)
public:	
class TMotifCanvas	**TMotifCanvas**(class TCanvas* *c,* char* *name,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TMotifCanvas	**TMotifCanvas**(class TMotifCanvas&)
class TMotifCanvas	**TMotifCanvas**(class TCanvas* *c,* char* *name,* UInt_t *width,* UInt_t *height*)
void	**~TMotifCanvas**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
void	**FitCanvas**( )
void	**ForceUpdate**( )
UInt_t	**GetCheight**( )
UInt_t	**GetCwidth**( )
void	**Iconify**( )
Int_t	**InitWindow**( )
virtual class TClass*	**IsA**( )
void	**ResizeCanvas**( )
void	**SetCanvas**(UInt_t *w,* UInt_t *h*)
void	**SetCanvasTitle**(Text_t* *newTitle*)
void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**ShowMenuBar**(Bool_t *show* = *kTRUE*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Widget	**fScrolledWindow**	*Scrolled window containing canvas*
Widget	**fDrawingArea**	*Canvas widget*
UInt_t	**fCwidth**	*Width of canvas in fUnits units*
UInt_t	**fCheight**	*Height of canvas in fUnits units*
EUnits	**fUnits**	*Canvas dimension units*
Int_t	**fButton**	*Currently pressed button*
Bool_t	**fAutoFit**	*if TRUE automatically fit canvas*
static MenuDesc_t	**fgMenuPane**	
static MenuDesc_t	**fgFilePane**	
static MenuDesc_t	**fgEditPane**	
static MenuDesc_t	**fgViewPane**	
static MenuDesc_t	**fgOptionPane**	
static MenuDesc_t	**fgInspectorPane**	
static MenuDesc_t	**fgClassesPane**	
static MenuDesc_t	**fgHelpPane**	
static String	**fgDefaultResources**	

Class Description

```
TMotifCanvas

This class creates a main window with menubar, scrollbars and a
drawing area.
```

void **AfterRealizeHook**( )

```
if no dimensions for the canvas are set, make canvas same size as window
```

void **ResizeCanvas**( )

```
change size of drawing area widget
```

void **HandleWMDeleteMessage**( )

```
Printf("Closing canvas %s", Canvas()->GetName());
```

void **HelpPostscriptCB**(*Widget,* XtPointer *cd, XtPointer*)

Author: Fons Rademakers 02/04/95
Last update: 1.03/01 04/09/97 13.37.26 by René Brun

TMotifContextMenu

class **TMotifContextMenu** : public TContextMenuImp, public TComponent

public:

void	**~TMotifContextMenu**( )
void	**ActionCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
void	**CancelCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
class TClass*	**Class**( )
const Char_t*	**ClassName**( )
virtual void	**CreateMenu**(class TObject* *object,* class TPopupMenu* *popup*)
virtual void	**Dialog**(class TObject* *object,* class TMethod* *method*)
virtual void	**DisplayPopup**(Int_t *x,* Int_t *y*)
class TMotifDialog*	**GetDialog**( )
Char_t*	**GetParameters**( )
virtual class TClass*	**IsA**( )
void	**OkCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
void	**OptionSelectCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**ToggleListCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
void	**ToggleMethodCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

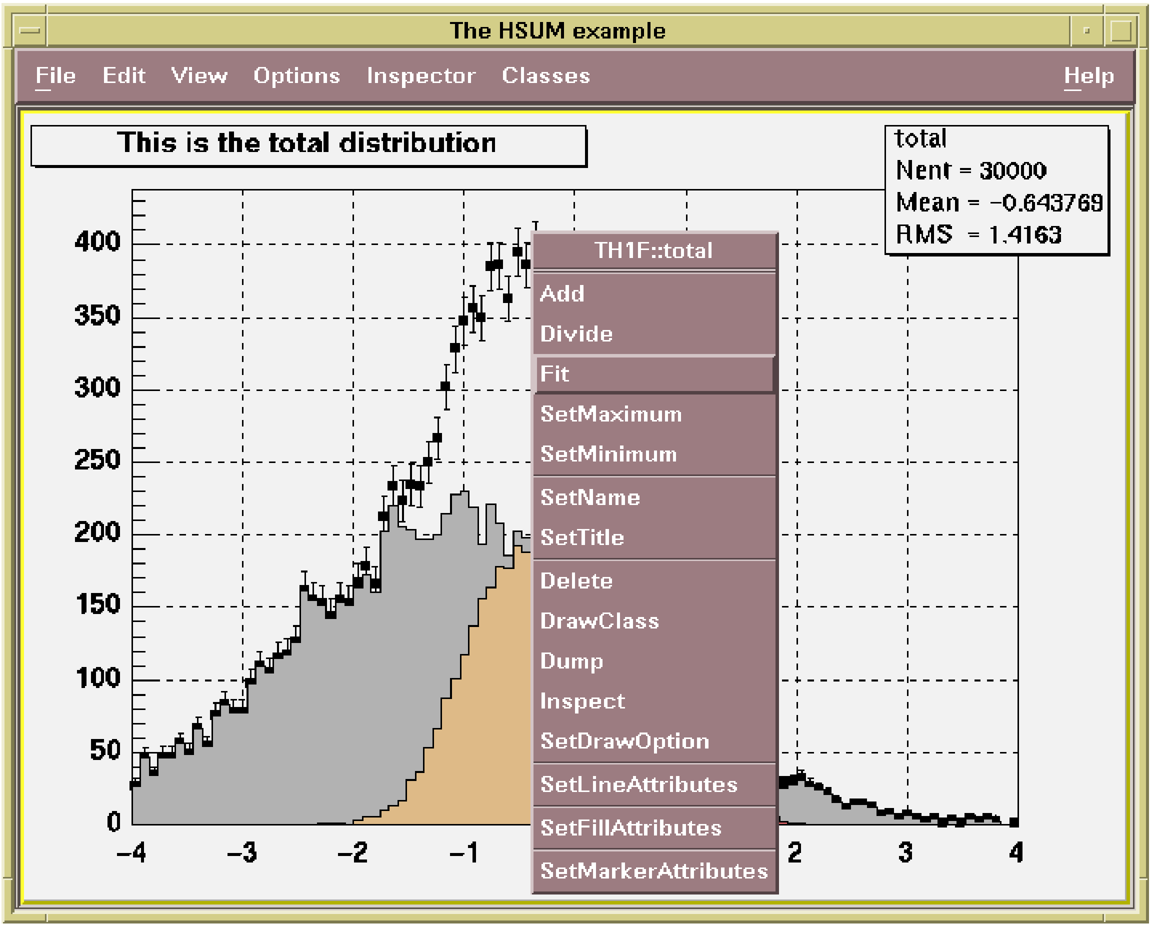
Data Members

protected:

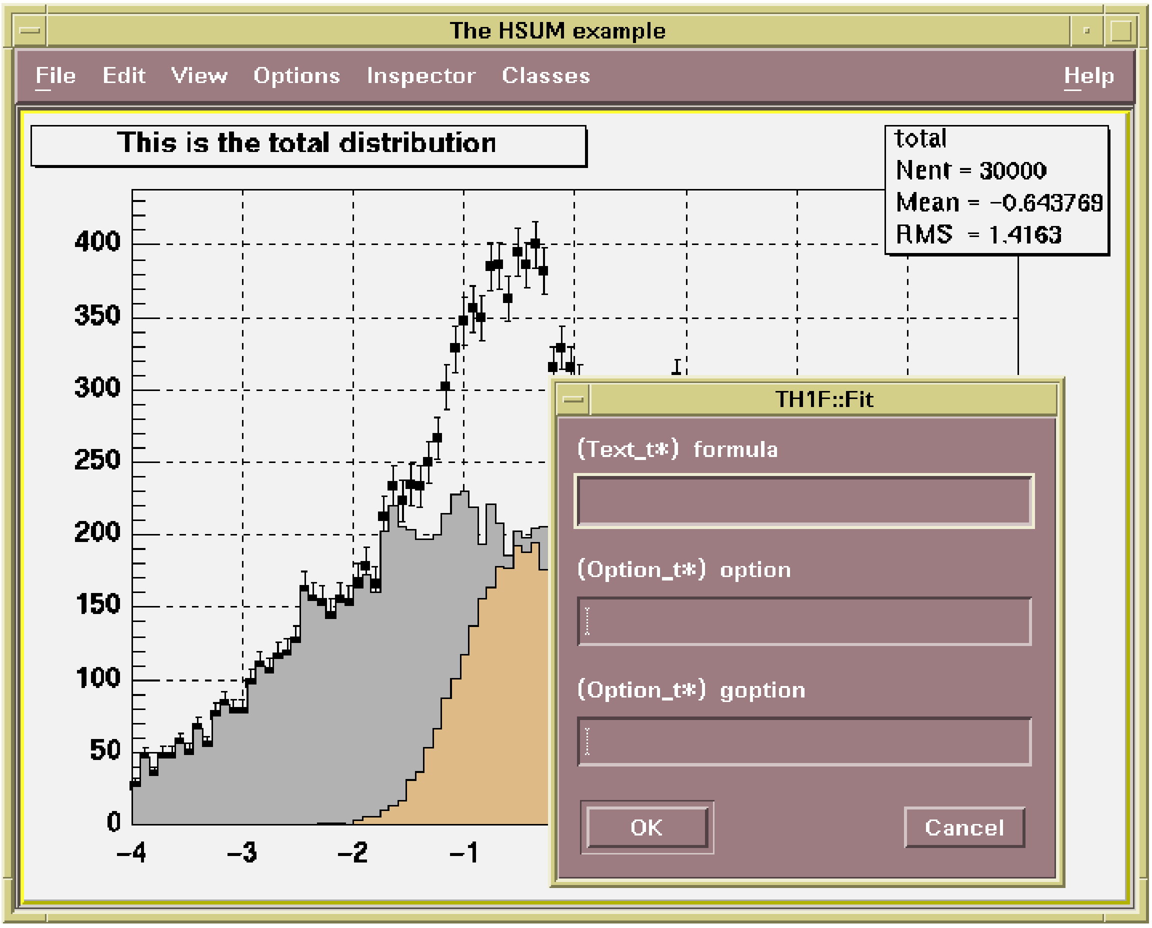
class TMotifDialog*	**fDialog**	*dialog*
class TPopupMenu*	**fPopup**	*popup menu*

Class Description

```
TMotifContextMenu .

This class provides an interface to  context sensitive popup menus.
These menus pop up when the user hits  the right mouse button,  and
are destroyed when the menu pops downs.
The picture below shows a canvas with a pop-up menu.
```



The picture below shows a canvas with a pop-up menu and a dialog box.



void **OkCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

Author: Nenad Buncic 08/02/96
Last update: 1.03/05 16/10/97 14.37.45 by René Brun

TMotifControlBar

class **TMotifControlBar** : public TControlBarImp, public TComponent

private:

virtual void	**Create**( )
virtual void	**Create**(Widget *parent*, Position *x = 0*, Position *y = 0*)
Widget	**CreateButton**(Widget *w*, Char_t* *label*, XtTranslations& *translations*)
Widget	**CreateDrawnButton**(Widget *w*, Char_t* *pixmapName*, XtTranslations& *translations*)

public:

void	**~TMotifControlBar**( )
void	**ActionCB**(Widget *w*, XtPointer *cd*, XtPointer *call_data*)
class TClass*	**Class**( )
const Text_t*	**ClassName**( )
void	**CloseCB**(Widget *w*, XtPointer *cd*, XtPointer *call_data*)
Widget	**CreateHintWidget**( )
Widget	**GetControlBarWidget**( )
Widget	**GetHintWidget**( )
Widget	**GetParentWidget**( )
void	**GetPosition**(Widget *w*, Position& *x*, Position& *y*, Int_t *vAlign = kTop*, Int_t *hAlign = kLeft*)
Widget	**GetSubmenuWidget**(Widget *w*)
virtual void	**Hide**( )
void	**HideHint**(Widget *w*, XEvent* *event*, String* *params*, Cardinal* *num_params*)
virtual class TClass*	**IsA**( )
virtual void	**Show**( )
void	**ShowHint**(Widget *w*, XEvent* *event*, String* *params*, Cardinal* *num_params*)
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**SubmenuCB**(Widget *w*, XtPointer *cd*, XtPointer *call_data*)

Data Members

private:

Pixmap*	**fPixmap**	*array of pixmaps*
Int_t	**fNPixmap**	*number of pixmaps*
static const String	**fgDefaultResources**	
static XtActionsRec	**fgActs**	
static char	**fgAddXlations**	

protected:

Widget	**fParentWidget**	*parent control bar widget*
Widget	**fControlBarWidget**	*control bar widget*
Widget	**fHintWidget**	*hint widget*

public:

static const enum	**kTop**
static const enum	**kLeft**
static const enum	**kCenter**
static const enum	**kBottom**
static const enum	**kRight**

Class Description

```
TMotifControlBar

This is the Motif implementation of TControlBar. Note that to be able
```

to display multi-coloured icons (.XPM) you have to have Motif library version 1.2.5 and above, otherwise only bitmaps (.XBM) can be used. TMotifControlBar looks for the pixmaps along a search path specified by the XBMLANGPATH environment variable or by a default search path, which varies depending on whether or not the XAPPLRESDIR environment variable is set. For more information about setting these variables see the XmGetPixmap() man page.

Of course the user has the capability of customizing the MotifControlBar to suit personal preferences using primary defaults files: an app-defaults file and a user-specific file named .Xdefaults.
You can have an app-defaults file in your home directory or you can tell the resource manager to look for an app-defaults file in a specific directory by specifying that directory in one of the environment variables XFILESEARCHPATH, XUSERFILESEARCHPATH, or XAPPLRESDIR.
The .Xdefaults file should be in the user's home directory.

```
X Defaults
------------------------------------------
<YourApplicationName>*controlBar*marginWidth:              0
<YourApplicationName>*controlBar*marginHeight:             0
<YourApplicationName>*controlBar*spacing:                  0
<YourApplicationName>*controlBar*drawnButton*marginWidth:  0
<YourApplicationName>*controlBar*drawnButton*marginHeight: 0
<YourApplicationName>*controlBar*drawnButton*shadowThickness: 2
<YourApplicationName>*controlBar*pushButton*marginWidth:   2
<YourApplicationName>*controlBar*pushButton*marginHeight:  2
<YourApplicationName>*controlBar*pushButton*shadowThickness:  2
<YourApplicationName>*controlBar*foreground:          #ffffff
<YourApplicationName>*controlBar*background:          #9e9c9b
<YourApplicationName>*controlBar*hint*foreground:     #ffffff
<YourApplicationName>*controlBar*hint*background:     #82807e
```

~TMotifControlBar( )
The destructor, of course.

void **Create**( )
A private member function. It creates a main control bar.

void **Create**(Widget *parent,* Position *x,* Position *y*)
A private member function. It creates a control bar.

Widget **CreateButton**(Widget *w,* Char_t **label,* XtTranslations &*translations*)
A private member function. This function is called by Create() and it creates a single push button.

Widget **CreateDrawnButton**(Widget *w,* Char_t **pixmapName,* XtTranslations &*translations*)
A private member function. This function is called by Create() and it creates a single drawn button.

Widget **CreateHintWidget**( )
This function is called by ShowHint(). It creates a hint widget.

void **GetPosition**(Widget *w,* Position &*x,* Position &*y,*Int_t vAlign, Int_t *hAlign*)
This function returns the button position on the screen. The values returned in 'x' and 'y' are calculated corresponding to an alignment which is specified by 'vAlign' and 'hAlign'. Valid entries for 'vAlign' and 'hAlign' are:

```
 vAlign: kTop       hAlign: kLeft
         kCenter            kCenter
         kBottom            kRight
```

```
(default: vAlign=kTop, hAlign=kLeft)
```

Widget **GetSubmenuWidget**(Widget *w*)

```
It returns widget for the next level control bar, if exist,
otherwise it returns 0.
```

void **Hide**( )

```
This function unmanages a previously managed control bar widget.
```

void **Show**( )

```
It manages a control bar widget. If a control bar widget doesn't exist
than it calls Create().
```

void **ActionCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

```
Execute an action associated with that widget
```

void **CloseCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

```
If the widget is a main control bar widget - execute the destructor,
otherwise do nothing.
```

void **SubmenuCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

```
Get the submenu widget and manage it.
```

void **ShowHint**(Widget *w,* XEvent **event,* String **params,* Cardinal **num_params*)

```
If a hint widget doesn't exist - create it, otherwise just change the hint
text massage, and manage it.
```

void **HideHint**(Widget *w,* XEvent **event,* String **params,* Cardinal **num_params*)

```
Simply unmanage a hint widget if previously managed.
```

Author: Nenad Buncic 20/02/96
Last update: 1.00/07 22/08/97 11.38.49 by Fons Rademakers

TMotifDialog

class **TMotifDialog** : public TComponent

protected:

class TMotifDialog	**TMotifDialog**( )

public:

class TMotifDialog	**TMotifDialog**(char* *name,* Bool_t *OkB = 1,* Bool_t *CanB = 1,* Bool_t *AplB = 0*)
class TMotifDialog	**TMotifDialog**(class TMotifDialog&)
void	**~TMotifDialog**( )
virtual void	**Add**(class TDialogItem* *i*)
virtual void	**AddApplyCallback**(XtCallbackProc *AplCB,* XtPointer *Cd*)
virtual void	**AddCancelCallback**(XtCallbackProc *CanCB,* XtPointer *Cd*)
virtual void	**AddOKCallback**(XtCallbackProc *OKCB,* XtPointer *Cd*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual class TMotifButton*	**GetApplyButton**( )
virtual class TMotifButton*	**GetCancelButton**( )
virtual class TList*	**GetItemList**( )
virtual class TMotifButton*	**GetOKButton**( )
virtual Widget	**Instantiate**(Widget *parent,* Arg* *arg = 0,* Int_t *numarg = 0*)
virtual Widget	**InstantiateDialog**(Widget *parent,* Arg* *arg = 0,* Int_t *numarg = 0*)
virtual class TClass*	**IsA**( )
virtual void	**ManageDialog**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TList*	**fItemList**	
class TMotifButton*	**fOKButton**	
class TMotifButton*	**fCancelButton**	
class TMotifButton*	**fApplyButton**	*by default not exist!*

Class Description

```
TMotifDialog

This class defines a Motif dialog class .
```

void **AddCancelCallback**(XtCallbackProc *CanCB,* XtPointer *Cd*)

Author: Piotr Golonka 20/08/97
Last update: 1.03/04 19/09/97 16.51.56 by Fons Rademakers

TMotifGuiFactory

class **TMotifGuiFactory** : public TGuiFactory

public:

class TMotifGuiFactory	**TMotifGuiFactory**(char* *name = Motif,* char* *title* = Motif GUI *Factory*)
class TMotifGuiFactory	**TMotifGuiFactory**(class TMotifGuiFactory&)
void	**~TMotifGuiFactory**( )
class TClass*	**Class**( )
class TApplicationImp*	**CreateApplicationImp**(char* *classname,* int* *argc,* char** *argv,* void* *option,* Int_t *numOptions*)
class TBrowserImp*	**CreateBrowserImp**(class TBrowser* *b,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TBrowserImp*	**CreateBrowserImp**(class TBrowser* *b,* char* *title,* UInt_t *width,* UInt_t *height*)
class TCanvasImp*	**CreateCanvasImp**(class TCanvas* *c,* char* *title,* UInt_t *width,* UInt_t *height*)
class TCanvasImp*	**CreateCanvasImp**(class TCanvas* *c,* char* *title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)
class TContextMenuImp*	**CreateContextMenuImp**(class TContextMenu* *c,* char* *name,* char* *title*)
class TControlBarImp*	**CreateControlBarImp**(class TControlBar* *c,* char* *name = MotifControlBar,* char* *title = MotifControlBar*)
class TIconImp*	**CreateIconImp**(class TIcon* *c*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TMotifGuiFactory

This class is a factory for Motif GUI components. It overrides
the member functions of the ABS TGuiFactory.
```

TMotifGuiFactory(const char *name, const char *title): **TGuiFactory**(*name, title*)

```
TMotifGuiFactory ctor.
```

TApplicationImp* **CreateApplicationImp**(const char **classname,* int **argc,* char ***argv,* void **options,* Int_t *numOptions*)

```
Create a Motif version of TApplicationImp
```

TCanvasImp* **CreateCanvasImp**(TCanvas **c,* const char **title,* UInt_t *width,* UInt_t *height*)

```
Create a Motif version of TCanvasImp
```

TCanvasImp* **CreateCanvasImp**(TCanvas **c,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)

```
Create a Motif version of TCanvasImp
```

TBrowserImp* **CreateBrowserImp**(TBrowser **b,* const char **title,* UInt_t *width,* UInt_t *height*)

```
Create a Motif version of TBrowserImp
```

TBrowserImp* **CreateBrowserImp**(TBrowser **b,* const char **title,* Int_t *x,* Int_t *y,* UInt_t *width,* UInt_t *height*)

```
Create a Motif version of TBrowserImp
```

TContextMenuImp* **CreateContextMenuImp**(TContextMenu **c,* const char **name,* const char **title*)

```
Create a Motif version of TContextMenuImp
```

TControlBarImp* **CreateControlBarImp**(TControlBar **c,* const char **name,* const char **title*)

```
Create a Motif version of TControlBarImp
```

TIconImp* **CreateIconImp**(TIcon **c*)

```
Create a Motif version of TIcon
```

Author: Fons Rademakers 15/11/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TMotifIcon

class **TMotifIcon** : public TIconImp

private:

class TMotifIcon	**TMotifIcon**( )

public:

class TMotifIcon	**TMotifIcon**(class TIcon* *i*)
class TMotifIcon	**TMotifIcon**(class TMotifIcon&)
void	**~TMotifIcon**( )
class TClass*	**Class**( )
struct _XImage*	**GetXImage**((*unknown*) EIconType t)
Bool_t	**Install**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

struct _XImage*	**fXImage**	*XImage structures*

Class Description

```
TMotifIcon
```

~TMotifIcon( )

```
MotifIcon destructor
```

Bool_t **Install**( )

```
It stores an image in an image cache that can later be used to
generate a pixmap. Returns kTRUE when successful, otherwise kFALSE.
```

Author: Nenad Buncic 26/11/96
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TMotifOptionMenu

class **TMotifOptionMenu** : public TDialogItem

protected:

class TMotifOptionMenu	**TMotifOptionMenu**( )

public:

class TMotifOptionMenu	**TMotifOptionMenu**(Text_t* *label*)
class TMotifOptionMenu	**TMotifOptionMenu**(class TMotifOptionMenu&)
void	**~TMotifOptionMenu**( )
virtual void	**AddItem**(Text_t* *label,* Text_t* *data*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual enum EDialogItemType	**DialogItemType**( )
virtual Text_t*	**GetData**( )
virtual Widget	**Instantiate**(Widget *parent,* Arg* *args = 0,* Int_t *numargs = 0*)
virtual class TClass*	**IsA**( )
void	**SelectedItemCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)
virtual void	**SetData**(Text_t* *data*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TList*	**fOptionList**	*list of items <=>PushButtons*
Widget	**fPullDown**	
class TDialogItem*	**fSelected**	

Class Description

```
TMotifOptionMenu

This class defines an option menu used in dialogs.
```

Text_t* **GetData**( )

Author: Piotr Golonka 20/08/97
Last update: 1.03/03 08/09/97 13.45.32 by Piotr Golonka

TMotifTextField

class **TMotifTextField** : public TDialogItem

protected:

class TMotifTextField	**TMotifTextField**( )

public:

class TMotifTextField	**TMotifTextField**(char* *name,* Text_t* *initData = 0*)
class TMotifTextField	**TMotifTextField**(class TMotifTextField&)
void	**~TMotifTextField**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual enum EDialogItemType	**DialogItemType**( )
virtual Text_t*	**GetData**( )
virtual Widget	**Instantiate**(Widget *parent,* Arg* *args = 0,* Int_t *numargs = 0*)
virtual class TClass*	**IsA**( )
virtual void	**SetData**(Text_t* *data*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**TextChangedCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

Data Members

private:

Widget	**fLabelWidget**

Class Description

```
TMotifTextField

A Motif TextField widget used in TDialogs.
```

void **TextChangedCB**(Widget *w,* XtPointer *cd,* XtPointer *call_data*)

Author: Piotr Golonka 21/08/97
Last update: 1.03/03 08/09/97 13.37.51 by Piotr Golonka

TNamed

class **TNamed** : public TObject

public:

class TNamed	**TNamed**(Text_t* *name,* Text_t* *title*)
class TNamed	**TNamed**(class TString& *name,* class TString& *title*)
class TNamed	**TNamed**( )
class TNamed	**TNamed**(class TNamed& *named*)
void	**~TNamed**( )
class TClass*	**Class**( )
virtual Int_t	**Compare**(class TObject* *obj*)
void	**Copy**(class TObject& *named*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**FillBuffer**(char*& *buffer*)
virtual const Text_t*	**GetName**( )
virtual const Text_t*	**GetTitle**( )
virtual ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsSortable**( )
virtual void	**ls**(Option_t* *option*)
virtual class TNamed&	***operator=***(class TNamed& *rhs*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetObject**(Text_t* *Name,* Text_t* *title*)
virtual void	**SetTitle**(Text_t* *title*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TString	**fName**	*object identifier*
class TString	**fTitle**	*object title*

See Also: TAxis, TBenchmark, TBranch, TBrowser, TChainElement, TColor, TContextMenu, TControlBar-Button, TCut, TDirectory, TEventList, TFileType, TFormula, TGXW, TGeometry, TGraph, TGui-Factory, TH1, TIcon, TInterpreter, TKey, TKeyMapFile, TLeaf, TMaterial, TMinuit, TNode, TPostScript, TRandom, TRotMatrix, TSelector, TServerSocket, TShape, TSocket, TStyle, TSys-tem, TSystemFile, TText, TToggle, TToggleGroup, TTree

TNamed( ) : **TObject**( )

```
Default TNamed constructor
```

TNamed(const Text_t **name,* const Text_t **title*): **TObject**( ), **fName**(*name*), **fTitle**(title)

```
TNamed constructor
```

TNamed(const TString &*name,* const TString &*title*): **TObject**( ), **fName**(*name*), **fTitle**(title)

```
TNamed constructor
```

~TNamed( )

```
TNamed destructor
```

void **Copy**(TObject &*obj*)

```
Copy TNamed parameters to another object
```

void **Draw**(Option_t **option*)
    `Draw TNamed`

void **FillBuffer**(char *&*buffer*)
    `Encode TNamed into output buffer`

void **ls**(Option_t **option*)
    `List TNamed parameters`

void **Paint**(Option_t **option*)
    `Paint object`

void **Print**(Option_t **option*)
    `Print TNamed parameters`

void **ReadBuffer**(char *&*buffer*)
    `Decode TNamed from input buffer`

void **SetName**(const Text_t **name*)
    `Change the name of the object`

void **SetObject**(const Text_t **name,* const Text_t **title*)
    `Set all the object parameters`

void **SetTitle**(const Text_t **title*)
    `Change the title of the object`

Int_t **Sizeof**( ) const
    `Return size of the TNamed part of the object`

Author: René Brun 26/12/94
Last update: 03/11/97 12.42.52 by René Brun

TNetFile

class **TNetFile** : public TFile

private:

class TNetFile	**TNetFile**( )
Bool_t	**Authenticate**( )
Bool_t	**CheckNetrc**(char*& *user,* char*& *passwd*)
char*	**GetPasswd**( )
char*	**GetUser**( )
void	**Init**(Bool_t *create*)
void	**Print**(Option_t* *option*)
void	**PrintError**(char* *where,* Int_t *err*)
Int_t	**Recv**(Int_t& *status,* enum EMessageTypes& *kind*)

public:

class TNetFile	**TNetFile**(char* *url,* Option_t* *option,* Text_t* *ftitle,* Int_t *compress = 1*)
class TNetFile	**TNetFile**(class TNetFile&)
void	**~TNetFile**( )
class TClass*	**Class**( )
void	**Close**( )
void	**Flush**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsOpen**( )
Bool_t	**ReadBuffer**(char* *buf,* int *len*)
void	**Seek**(Seek_t *offset,* enum TFile::ERelativeTo *pos = kBeg*)
void	**SetPasswd**(char* *passwd*)
void	**SetUser**(char* *user*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
Bool_t	**WriteBuffer**(char* *buf,* int *len*)

Data Members

private:

class TUrl	**fUrl**	*URL of file*
class TString	**fUser**	*remote user name*
Seek_t	**fOffset**	*seek offset*
class TSocket*	**fSocket**	*connection to rootd server*
static char*	**fgUser**	
static char*	**fgPasswd**	

Class Description

```
TNetFile

A TNetFile is like a normal TFile except that it reads and writes
its data via a rootd server (for more on the rootd daemon see the
source files ROOTD_*.cxx). TNetFile file names are in standard URL
format with protocol "root". The following are valid TNetFile URL's:

 root://hpsalo/files/aap.root
 root://hpbrun.cern.ch/root/hsimple.root
 root://pcna49a:5151/~na49/data/run821.root
 root://pcna49d.cern.ch:5050//v1/data/run810.root

The only difference with the well known httpd URL's is that the root
of the remote file tree is the user's home directory. Therefore an
absolute pathname requires a // after the host or port specifier
(see last example). Further the expansion of the standard shell
```

```
characters, like ~, $, .., are handled as expected.
TNetFile (actually TUrl) uses 432 as default port for rootd.

Connecting to a rootd requires the remote user id and password.
TNetFile allows three ways for you to provide your login:
1) Setting it globally via the static functions:
       TNetFile::SetUser() and TNetFile::SetPasswd()
2) Getting it from the ~/.netrc file (same file as used by ftp)
3) Command line prompt
The different methods will be tried in the order given above.
On machines with AFS rootd will authenticate using AFS.

The rootd daemon lives in the directory $ROOTSYS/bin. It can be
started either via inetd or by hand from the command line (no need
to be super user). For more info about rootd see the web page:
http://root.cern.ch/root/NetFile.html.
```

TNetFile(const char **url,* Option_t **option,* const Text_t **ftitle,* Int_t *compress*): **fUrl**(*url*), **TFile**(*url,* "NET", *ftitle,* compress)

```
Create a NetFile object. A net file is the same as a TFile
except that it is being accessed via a rootd server. The url
argument must be of the form: root://host.dom.ain/file.root.
If the file specified in the URL does not exist, is not accessable
or can not be created the kZombie bit will be set in the TNetFile
object. Use IsZombie() to see if the file is accessable.
```

~TNetFile( )

```
TNetFile dtor. Send close message and close socket.
```

Bool_t **Authenticate**( )

```
Authenticate to remote rootd server. Return kTRUE if authentication
succeeded.
```

Bool_t **CheckNetrc**(char *&*user,* char *&*passwd*)

```
Try to get user name and passwd from the ~/.netrc file.
This file will only be used when its access mask is 0600.
Returns kTRUE if user and passwd were found for the machine
specified in the URL. User and passwd must be deleted by
the caller. If kFALSE, user and passwd are 0.
```

void **Close**( )

```
Close remote file.
```

void **Flush**( ) const

```
Flush file to disk.
```

char* **GetUser**( )

```
Get user name to be used for authentication to rootd.
User is asked to type user name.
Returns user name (which must be deleted by caller) or 0.
```

char* **GetPasswd**( )

```
Get passwd to be used for authentication to rootd.
Uses non-echoing command line to get passwd.
Returns passwd (which must de deleted by caller) or 0.
```

void **Init**(Bool_t *create*)

```
Initialize a TNetFile object.
```

Bool_t **IsOpen**( ) const

```
Retruns kTRUE if file is open, kFALSE otherwise.
```

void **Print**(Option_t **option*)

```
Print some info about the net file.
```

void **PrintError**(const char **where,* Int_t *err*)
```
Print error string depending on error code.
```

Bool_t **ReadBuffer**(char **buf,* int *len*)
```
Read specified byte range from remote file via rootd daemon.
Returns kTRUE in case of error.
```

Bool_t **WriteBuffer**(const char **buf,* int *len*)
```
Write specified byte range to remote file via rootd daemon.
Returns kTRUE in case of error.
```

Int_t **Recv**(Int_t *&status,* EMessageTypes *&kind*)
```
Return status from rootd server and message kind. Returns -1 in
case of error otherwise 8 (sizeof 2 words, status and kind).
```

void **Seek**(Seek_t *offset,* ERelativeTo *pos*)
```
Set position from where to start reading.
```

void **SetUser**(const char **user*)
```
Set user name to be used for authentication to rootd.
```

void **SetPasswd**(const char **passwd*)
```
Set passwd to be used for authentication to rootd.
```

Author: Fons Rademakers 14/08/97
Last update: 1.03/05 15/10/97 14.39.37 by Fons Rademakers

TNode

class **TNode** : public TNamed, public TAttLine, public TAttFill

public:

class TNode	**TNode**(Text_t* *name,* Text_t* *title,* Text_t* *shapename,* Double_t *x = 0,* Double_t *y = 0,* Double_t *z = 0,* Text_t* *matrixname,* Option_t* *option*)
class TNode	**TNode**( )
class TNode	**TNode**(Text_t* *name,* Text_t* *title,* class TShape* *shape,* Double_t *x = 0,* Double_t *y = 0,* Double_t *z = 0,* class TRotMatrix* *matrix = 0,* Option_t* *option*)
class TNode	**TNode**(class TNode&)
void	**~TNode**( )
virtual void	**Browse**(class TBrowser* *b*)
virtual void	**BuildListOfNodes**( )
virtual void	**cd**(Text_t* *path = 0*)
class TClass*	**Class**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawOnly**(Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
class TList*	**GetListOfNodes**( )
virtual class TRotMatrix*	**GetMatrix**( )
virtual class TNode*	**GetNode**(Text_t* *name*)
virtual Text_t*	**GetObjectInfo**(Int_t *px,* Int_t *py*)
virtual class TNode*	**GetParent**( )
class TShape*	**GetShape**( )
Int_t	**GetVisibility**( )
virtual Double_t	**GetX**( )
virtual Double_t	**GetY**( )
virtual Double_t	**GetZ**( )
virtual void	**ImportShapeAttributes**( )
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual void	**Local2Master**(Float_t* *local,* Float_t* *master*)
virtual void	**Local2Master**(Double_t* *local,* Double_t* *master*)
virtual void	**ls**(Option_t* *option = 2*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**SetMatrix**(class TRotMatrix* *matrix = 0*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetParent**(class TNode* *parent*)
virtual void	**SetPosition**(Double_t *x = 0,* Double_t *y = 0,* Double_t *z = 0*)
virtual void	**SetVisibility**(Int_t *vis = 1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**UpdateGlobalMatrix**(Double_t* *dx1,* Double_t* *rmat1,* Double_t *x,* Double_t *y,* Double_t *z,* Double_t* *matrix,* Double_t* *dxnew,* Double_t* *rmatnew*)

Data Members

protected:

Double_t	**fX**	*X offset with respect to parent object*
Double_t	**fY**	*Y offset with respect to parent object*

Double_t	**fZ**	*Z offset with respect to parent object*
class TRotMatrix*	**fMatrix**	*Pointer to rotation matrix*
class TShape*	**fShape**	*Pointer to shape definition*
class TNode*	**fParent**	*Pointer to parent positioned volume*
class TList*	**fNodes**	*List of son nodes (if any)*
class TString	**fOption**	*List of options if any*
Int_t	**fVisibility**	*Visibility flag*

See Also: TNodeDiv

Class Description

```
T N O D E  description
 A TNode object is used to build the geometry hierarchy (see TGeometry).
 A node may contain other nodes.

 A geometry node has attributes:
   - name and title
   - pointer to the referenced shape (see TShape).
   - x,y,z offset with respect to the mother node.
   - pointer to the rotation matrix (see TRotMatrix).

 A node can be drawn.
```

TNode( )

```
Node default constructor
```

TNode(Text_t **name,* Text_t **title,* Text_t **shapename,* Double_t *x,* Double_t *y,* Double_t *z,* Text_t **matrixname,* Option_t **option*):**TNamed**(*name, title*), **TAttLine**( ), **TAttFill**( )

```
Node normal constructor
 name    is the name of the node
 title   is title
 shapename is the name of the referenced shape
 x,y,z   are the offsets of the volume with respect to his mother
 matrixname  is the name of the rotation matrix

 This new node is added into the list of sons of the current node
```

TNode(Text_t **name,* Text_t **title,* TShape **shape,* Double_t *x,* Double_t *y,* Double_t *z,* TRotMatrix **matrix,* Option_t **option*):**TNamed**(*name, title*), **TAttLine**( ), **TAttFill**( )

```
Node normal constructor
 name    is the name of the node
 title   is title
 shape   is the pointer to the shape definition
 x,y,z   are the offsets of the volume with respect to his mother
 matrix  is the pointer to the rotation matrix

 This new node is added into the list of sons of the current node

The color "1" - default produces a very bad 3D image with OpenGL
```

~TNode( )

```
Node default destructor
```

void **BuildListOfNodes**( )

```
Create the list to support sons of this node
```

void **cd**(const Text_t **path*)

```
Change Current Reference node to this
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a Node
```

```
Compute the closest distance of approach from point px,py to this node.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw Referenced node with current parameters
```

void **DrawOnly**(Option_t **option*)

```
Draw only Sons of this node
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the window
```

TNode* **GetNode**(Text_t **name*)

```
Return pointer to node with name in the node tree
```

void **ImportShapeAttributes**( )

```
Copy shape attributes as node attributes
```

Bool_t **IsFolder**( )

```
Return TRUE if node contains nodes, FALSE otherwise
```

void **Local2Master**(Double_t **local,* Double_t **master*)

```
Convert one point from local system to master reference system
```

void **Local2Master**(Float_t **local,* Float_t **master*)

```
Convert one point from local system to master reference system
```

void **ls**(Option_t **option*)

```
List Referenced object with current parameters
```

void **Paint**(Option_t **option*)

```
Paint Referenced node with current parameters
vis = 1  (default) shape is drawn
vis = 0  shape is not drawn but its sons may be not drawn
vis = -1 shape is not drawn. Its sons are not drawn
vis = -2 shape is drawn. Its sons are not drawn
```

void **RecursiveRemove**(TObject **obj*)

```
Recursively remove object from the list of nodes of this node
```

void **SetName**(const Text_t **name*)

```
Change the name of this Node
```

void **SetVisibility**(Int_t *vis*)

```
Set visibility for this node and its sons
vis = 3  node is drawn and its sons are drawn
vis = 2  node is not drawn but its sons are drawn
vis = 1  (default) node is drawn
vis = 0  node is not drawn
vis = -1 node is not drawn. Its sons are not drawn
vis = -2 node is drawn. Its sons are not drawn
vis = -3 Only node leaves are drawn
vis = -4 Node is not drawn. Its immediate sons are drawn
```

void **Sizeof3D**( ) const

```
Return total size of this 3-D Node with its attributes
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **UpdateGlobalMatrix**(Double_t **dx,* Double_t *rmat, Double_t x, Double_t y, Double_t z, Double_t *matrix, Double_t *dxnew, Double_t **rmatnew*)

```
Compute new translation vector and global matrix
dx      old translation vector
rmat    old global matrix
x,y,z   offset of new local system with respect to mother
dxnew   new translation vector
rmatnew new global rotation matrix
```

Author: René Brun 14/09/95
Last update: 1.03/07 04/11/97 17.10.42 by René Brun

TNodeDiv

class **TNodeDiv** : public TNode

public:

class TNodeDiv	**TNodeDiv**(Text_t* *name,* Text_t* *title,* Text_t* *shapename,* Int_t *ndiv,* Int_t *axis,* Option_t* *option*)
class TNodeDiv	**TNodeDiv**( )
class TNodeDiv	**TNodeDiv**(Text_t* *name,* Text_t* *title,* class TShape* *shape,* Int_t *ndiv,* Int_t *axis,* Option_t* *option*)
class TNodeDiv	**TNodeDiv**(class TNodeDiv&)
void	**~TNodeDiv**( )
class TClass*	**Class**( )
virtual void	**Draw**(Option_t* *option*)
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNdiv**	*Number of divisions*
Int_t	**fAxis**	*Axis number where object is divided*

TNodeDiv( )

```
NodeDiv default constructor
```

TNodeDiv(Text_t **name,* Text_t **title,* Text_t **shapename,* Int_t *ndiv,* Int_t *axis,* Option_t **option*):**TNode**(*name, title, shapename, 0, 0, 0, 0, option*)

```
NodeDiv normal constructor
 name    is the name of the node
 title   is title
 shapename is the name of the referenced shape
 x,y,z   are the offsets of the volume with respect to his mother
 matrixname  is the name of the rotation matrix

 This new node is added into the list of sons of the current node
```

TNodeDiv(Text_t **name,* Text_t **title,* TShape **shape,* Int_t *ndiv,* Int_t *axis,* Option_t **option*):**TNode**(*name, title, shape, 0, 0, 0, 0, option*)

```
NodeDiv normal constructor
 name    is the name of the node
 title   is title
 shape   is the pointer to the shape definition
 ndiv    number of divisions
 axis    number of the axis for the division

 This new node is added into the list of sons of the current node
```

~TNodeDiv( )

```
NodeDiv default destructor
```

void **Draw**(Option_t **option*)

```
Draw Referenced node with current parameters
```

void **Paint**(Option_t **option*)

```
Paint Referenced node with current parameters
```

Author: René Brun 14/09/95

Last update: 1.00/06 23/03/97 11.26.51 by René Brun

TNtuple

class **TNtuple** : public TTree

public:

class TNtuple	**TNtuple**( )
class TNtuple	**TNtuple**(Text_t* *name,* Text_t* *title,* Text_t* *varlist,* Int_t *bufsize = 32000*)
class TNtuple	**TNtuple**(class TNtuple&)
void	**~TNtuple**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Fill**(Float_t *x0,* Float_t *x1,* Float_t *x2 = 0,* Float_t *x3 = 0,* Float_t *x4 = 0,* Float_t *x5 = 0,* Float_t *x6 = 0,* Float_t *x7 = 0,* Float_t *x8 = 0,* Float_t *x9 = 0,* Float_t *x10 = 0,* Float_t *x11 = 0,* Float_t *x12 = 0,* Float_t *x13 = 0,* Float_t *x14 = 0*)
virtual void	**Fill**(Float_t* *x*)
Float_t*	**GetArgs**( )
virtual Int_t	**GetNvar**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNvar**	*Number of columns*
Float_t*	**fArgs**	*Array of variables*

Class Description

```
TNtuple

A simple tree restricted to a list of float variables only.

Each variable goes to a separate branch.

A Ntuple is created via
  TNtuple(name,title,varlist,bufsize)
It is filled via:
  TNtuple::Fill(*x)  or
  TNtuple::Fill(v1,v2,v3.....)
```

TNtuple( ): **TTree**( )

```
Default constructor for Ntuple
```

TNtuple(const Text_t **name,* const Text_t **title,* const Text_t **varlist,* Int_t *bufsize*):**TTree**(*name, title*)

```
Create an Ntuple
    The parameter varlist describes the list of the ntuple variables
    separated by a colon:
      example:  "x:y:z:energy"
    For each variable in the list a separate branch is created.

    Use TTree to create branches with variables of different data types.
```

~TNtuple( )

```
Default destructor for an Ntuple
```

void **Fill**(Float_t **x*)

```
Fill a Ntuple with an array of floats
```

void **Fill**(Float_t *x0,* Float_t *x1,* Float_t *x2,* Float_t *x3,* Float_t x4, Float_t x5, Float_t x6, Float_t x7, Float_t x8, Float_t x9, Float_t x10, Float_t x11, Float_t x12, Float_t x13, Float_t *x14*)

```
Fill a Ntuple: Each Ntuple item is an argument
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 06/04/96
Last update: 1.00/08 12/04/97 10.30.25 by René Brun

TObjArray

class **TObjArray** : public TSeqCollection

private:

Bool_t	**BoundsOk**(char* *where,* Int_t *at*)
Int_t	**GetAbsLast**( )
void	**Init**(Int_t *s,* Int_t *lowerBound*)
Bool_t	**OutOfBoundsError**(char* *where,* Int_t *i*)

public:

class TObjArray	**TObjArray**(Int_t *s* = TCollection::*kInitCapacity,* Int_t *lowerBound = 0*)
class TObjArray	**TObjArray**(class TObjArray& *a*)
void	**~TObjArray**( )
virtual void	**Add**(class TObject* *obj*)
virtual void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
virtual void	**AddAt**(class TObject* *obj,* Int_t *idx*)
virtual void	**AddAtAndExpand**(class TObject* *obj,* Int_t *idx*)
virtual Int_t	**AddAtFree**(class TObject* *obj*)
virtual void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj*)
virtual void	**AddLast**(class TObject* *obj*)
virtual class TObject*	**After**(class TObject* *obj*)
virtual class TObject*	**At**(Int_t *idx*)
virtual class TObject*	**Before**(class TObject* *obj*)
virtual Int_t	**BinarySearch**(class TObject* *obj,* Int_t *upto* = *kMaxInt*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Compress**( )
virtual void	**Delete**(Option_t* *option*)
virtual void	**Expand**(Int_t *newSize*)
virtual class TObject*	**First**( )
Int_t	**GetEntries**( )
Int_t	**GetEntriesFast**( )
Int_t	**GetLast**( )
Int_t	**IndexOf**(class TObject* *obj*)
virtual class TClass*	**IsA**( )
virtual class TObject*	**Last**( )
Int_t	**LowerBound**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir* = *kIterForward*)
virtual class TObject*&	**operator[]**(Int_t *i*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**RemoveAt**(Int_t *idx*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sort**(Int_t *upto* = *kMaxInt*)
virtual void	**Streamer**(class TBuffer& *b*)
class TObject*	**UncheckedAt**(Int_t *i*)

Data Members

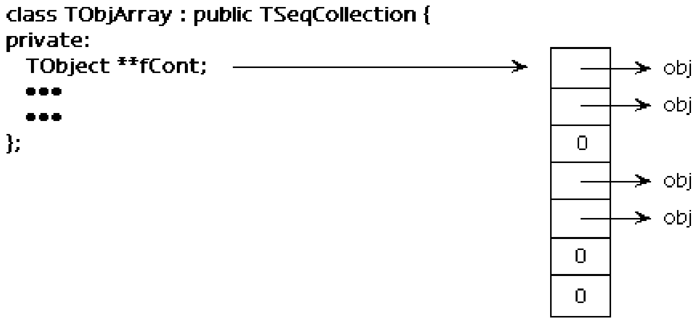
private:

class TObject**	**fCont**	*Array contents*
Int_t	**fLowerBound**	*Lower bound of the array*
Int_t	**fLast**	*Last element in array containing an object*

See Also: TClonesArray

Class Description

```
TObjArray

An array of TObjects. The array expands automatically when
objects are added (shrinking can be done by hand using Expand(),
how nice to have meaningful names -:)).
Use operator[] to have "real" array behaviour.
```



TObjArray(Int_t *s,* Int_t *lowerBound*)
```
Create an object array. Using s one can set the array size (default is
kInitCapacity=16) and lowerBound can be used to set the array lowerbound
index (default is 0).
```

TObjArray(const TObjArray &*a*)
```
Create a copy of TObjArray a.
```

~TObjArray( )
```
Delete an array.
```

void **AddFirst**(TObject **obj*)
```
Add object in the first slot of the array. This will overwrite the
first element that might have been there. To have insertion semantics
use either a TList or a TOrdCollection.
```

void **AddLast**(TObject **obj*)
```
Add object in the next empty slot in the array. Expand the array
if necessary.
```

void **AddBefore**(TObject **before,* TObject **obj*)
```
Add object in the slot before object before. If before=0 add object
in the first slot. Note that this will overwrite any object that
might have already been in this slot. For insertion semantics use
either a TList or a TOrdCollection.
```

void **AddAfter**(TObject **after,* TObject **obj*)
```
Add object in the slot after object after. If after=0 add object in
the last empty slot. Note that this will overwrite any object that
might have already been in this slot. For insertion semantics use
either a TList or a TOrdCollection.
```

void **AddAtAndExpand**(TObject **obj,* Int_t *idx*)
```
Add object at position idx. If idx is larger than the current size
of the array, expand the array (double its size).
```

void **AddAt**(TObject **obj,* Int_t *idx*)
```
Add object at position ids. Give an error when idx is out of bounds
(i.e. the array is not expanded).
```

Int_t **AddAtFree**(TObject **obj*)
```
Return the position of the new object.
```

```
Find the first empty cell or AddLast if there is no empty cell
```

TObject* **After**(TObject **obj*) const

```
Return the object after obj. Returns 0 if obj is last object.
```

TObject* **At**(Int_t *i*) const

```
Return the object at position i. Returns 0 if i is out of bounds.
```

TObject* **Before**(TObject **obj*) const

```
Return the object before obj. Returns 0 if obj is first object.
```

void **Clear**(Option_t **option*)

```
Remove all objects from the array. Does not delete the objects.
```

void **Compress**( )

```
Remove empty slots from array.
```

void **Delete**(Option_t **option*)

```
Remove all objects from the array AND delete all heap based objects.
```

void **Expand**(Int_t *newSize*)

```
Expand or shrink the array to newSize elements.
```

void **Streamer**(TBuffer *&b*)

```
Stream all objects in the array to or from the I/O buffer.
```

TObject* **First**( ) const

```
Return the object in the first slot.
```

TObject* **Last**( ) const

```
Return the object in the last filled slot. Returns 0 if no entries.
```

Int_t **GetEntries**( ) const

```
Return the number of objects in array (i.e. number of non-empty slots).
Attention: use this method ONLY if you want to know the number of
non-empty slots. This function loops over the complete array and
is therefore very slow when applied in a loop. Most of the time you
better use GetLast()+1.
```

Int_t **GetAbsLast**( ) const

```
Return absolute index to last object in array. Returns -1 in case
array is empty.
```

Int_t **GetLast**( ) const

```
Return index of last object in array. Returns lowerBound-1 in case
array is empty.
```

Int_t **IndexOf**(TObject **obj*) const

```
obj != 0 Return index of object in array.
       Returns lowerBound-1 in case array doesn't contain the obj.

obj == 0 Return the index of the first empty slot.
       Returns lowerBound-1 in case array doesn't contain any empty slot.
```

void **Init**(Int_t *s*, Int_t *lowerBound*)

```
Initialize a TObjArray.
```

TIterator* **MakeIterator**(Bool_t *dir*) const

```
Returns an array iterator.
```

Bool_t **OutOfBoundsError**(const char *where, Int_t *i*) const

```
Generate an out-of-bounds error. Always returns false.
```

void **RemoveAt**(Int_t *idx*)

```
Remove object at index idx.
```

TObject* **Remove**(TObject **obj*)

```
Remove object from array.
```

void **Sort**(Int_t *upto*)

```
If objects in array are sortable (i.e. IsSortable() returns true
for all objects) then sort array.
```

Int_t **BinarySearch**(TObject **op,* Int_t *upto*)

```
Find object using a binary search. Array must first have been sorted.
Search can be limited by setting upto to desired index.
```

Author: Fons Rademakers 11/09/95
Last update: 1.03/05 02/10/97 13.17.21 by Fons Rademakers

TObjArrayIter

class **TObjArrayIter** : public TIterator

private:

class TObjArrayIter	**TObjArrayIter**( )

public:

class TObjArrayIter	**TObjArrayIter**(class TObjArray* *arr,* Bool_t *dir* = *kIterForward*)
class TObjArrayIter	**TObjArrayIter**(class TObjArrayIter&)
void	**~TObjArrayIter**( )
class TClass*	**Class**( )
virtual const class TCollection*	**GetCollection**( )
virtual class TClass*	**IsA**( )
virtual class TObject*	**Next**( )
virtual void	**Reset**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

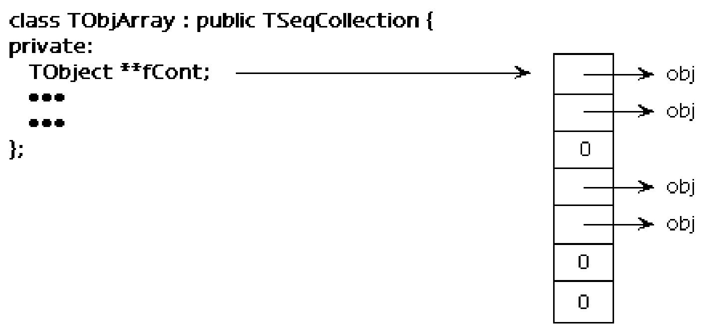
Data Members

private:

const class TObjArray*	**fArray**	*array being iterated*
Int_t	**fCursor**	*current position in array*
Bool_t	**fDirection**	*iteration direction*

Class Description

```
TObjArray

An array of TObjects. The array expands automatically when
objects are added (shrinking can be done by hand using Expand(),
how nice to have meaningful names -:)).
Use operator[] to have "real" array behaviour.
```



TObjArrayIter(const TObjArray **arr,* Bool_t *dir*)

```
Create array iterator. By default the iteration direction
is kIterForward. To go backward use kIterBackward.
```

TObject* **Next**( )

```
Return next object in array. Returns 0 when no more objects in array.
```

void **Reset**( )

```
Reset array iterator.
```

Author: Fons Rademakers 11/09/95
Last update: 1.03/05 02/10/97 13.17.21 by Fons Rademakers

TObject

class **TObject**

protected:

void	**MakeZombie**( )

public:

class TObject	**TObject**( )
class TObject	**TObject**(class TObject& *object*)
void	**~TObject**( )
void	**AbstractMethod**(char* *method*)
void	**AppendDirectory**( )
void	**AppendPad**(Option_t* *option*)
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual const Text_t*	**ClassName**( )
virtual void	**Clear**(Option_t*)
virtual class TObject*	**Clone**( )
virtual void	**Close**(Option_t* *option*)
virtual Int_t	**Compare**(class TObject* *obj*)
virtual void	**Copy**(class TObject& *object*)
Int_t	**DecreaseDirLevel**( )
virtual void	**Delete**(Option_t* *option*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawClass**( )
virtual void	**DrawClone**(Option_t* *option*)
virtual void	**Dump**( )
void	**Error**(char* *method,* char* *msgfmt, int*)
virtual void	**Execute**(class TMethod* *method,* const class TObjArray* *params*)
virtual void	**Execute**(Text_t* *method,* Text_t* *params*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
void	**Fatal**(char* *method,* char* *msgfmt, int*)
Int_t	**GetDirLevel**( )
virtual const Option_t*	**GetDrawOption**( )
virtual const Text_t*	**GetName**( )
virtual Text_t*	**GetObjectInfo**(Int_t *px,* Int_t *py*)
Bool_t	**GetObjectStat**( )
virtual const Option_t*	**GetOption**( )
virtual const Text_t*	**GetTitle**( )
virtual UInt_t	**GetUniqueID**( )
virtual ULong_t	**Hash**( )
Int_t	**IncreaseDirLevel**( )
void	**IndentLevel**( )
virtual Bool_t	**InheritsFrom**(class TClass* *cl*)
virtual Bool_t	**InheritsFrom**(Text_t* *classname*)
virtual void	**Inspect**( )
void	**InvertBit**(UInt_t *f*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsEqual**(class TObject* *obj*)
virtual Bool_t	**IsFolder**( )
virtual Bool_t	**IsModified**( )
Bool_t	**IsOnHeap**( )
virtual Bool_t	**IsSortable**( )
Bool_t	**IsZombie**( )
virtual void	**ls**(Option_t* *option*)

void	**MayNotUse**(char* *method*)
virtual void	**Modified**(Bool_t *flag = kTRUE*)
void	**operator delete**(void* *vp*)
void*	**operator new**(size_t *sz,* void* *vp*)
void*	**operator new**(size_t *sz*)
virtual class TObject&	***operator=***(class TObject& *rhs*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**Pop**( )
virtual void	**Print**(Option_t* *option*)
virtual void	**Read**(Text_t* *name*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
void	**ResetBit**(UInt_t *f*)
virtual void	**SavePrimitive**((*unknown*) ofstream& out, Option_t* option)
void	**SetBit**(UInt_t *f*)
void	**SetBit**(UInt_t *f,* Bool_t *set*)
void	**SetDirLevel**(Int_t *level = 0*)
virtual void	**SetDrawOption**(Option_t* *option*)
void	**SetObjectStat**(Bool_t *stat*)
virtual void	**SetUniqueID**(UInt_t *uid*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)
void	**SysError**(char* *method,* char* *msgfmt, int*)
Bool_t	**TestBit**(UInt_t *f*)
virtual void	**UseCurrentStyle**( )
void	**Warning**(char* *method,* char* *msgfmt, int*)
virtual void	**Write**(Text_t* *name = 0,* Int_t *bufsize = 0*)

Data Members

private:

UInt_t	**fUniqueID**	*object unique identifier*
UInt_t	**fBits**	*bit field status word*
static Int_t	**fgDirLevel**	*indentation level for ls( )*
static Bool_t	**fgObjectStat**	*if true keep track of objects in TObjectTable*

public:

static const enum TObject::	**kIsOnHeap**
static const enum TObject::	**kNotDeleted**
static const enum TObject::	**kZombie**
static const enum TObject::	**kBitMask**

See Also: TApplication, TBox, TBuffer, TClassTable, TCollection, TDialogItem, TDictionary, TEllipse, TEnv, TFree, THtml, TInetAddress, TLego, TLine, TMapFile, TMarker, TMarker3DBox, TMemberInspector, TMethodCall, TMonitor, TNamed, TObjString, TObjectTable, TPacketGenerator, TPolyLine, TPolyLine3D, TPolyMarker, TPolyMarker3D, TProof, TRealData, TSlave, TStopwatch, TSysEvtHandler, TUrl, TView

Class Description

```
TObject

Mother of all ROOT objects.

The TObject class provides default behaviour and protocol for all
objects in the ROOT system. It provides protocol for object I/O,
error handling, sorting, inspection, printing, drawing, etc.
Every object which inherits from TObject can be stored in the
ROOT collection classes.
```

TObject( )

```
TObject constructor. It sets the two data words of TObject to their
initial values. The unique ID is set to 0 and the status word is
set depending if the object is created on the stack or allocated
on the heap. Of the status word the high 8 bits are reserved for
system usage and the low 24 bits are user settable. Depending on
the ROOT environment variable "Root.MemStat" (see TEnv.h) the object
is added to the global TObjectTable for bookkeeping.
```

TObject(const TObject &*obj*)

```
TObject copy ctor.
```

~TObject( )

```
TObject destructor. Removes object from all canvases and object browsers
iff observer bit is on and remove from the global object table.
```

void **AppendDirectory**( )

```
Append object to current directory.
```

void **AppendPad**(Option_t **option*)

```
Append graphics object to current pad. In case no current pad is set
yet, create a default canvas with the name "c1".
```

void **Browse**(TBrowser **b*)

```
Browse object (by default we inspect). May be overridden for  other default
action
```

const Text_t* **ClassName**( ) const

```
Returns name of class to which the object belongs.
```

TObject* **Clone**( )

```
Copy obj using the Streamer facility
```

void **Close**(Option_t **option*)

```
Close abstract method. Must be overridden in case an object needs to
to something in case it is being removed from collections.
```

Int_t **Compare**(TObject **obj*)

```
Compare abstract method. Must be overridden if a class wants to be able
to compare itself with other objects. Must return -1 if this is smaller
than obj, 0 if objects are equal and 1 if this is larger than obj.
```

void **Delete**(Option_t **option*)

```
Delete this object. Typically called as a command via the interpreter.
Normally use "delete" operator.
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

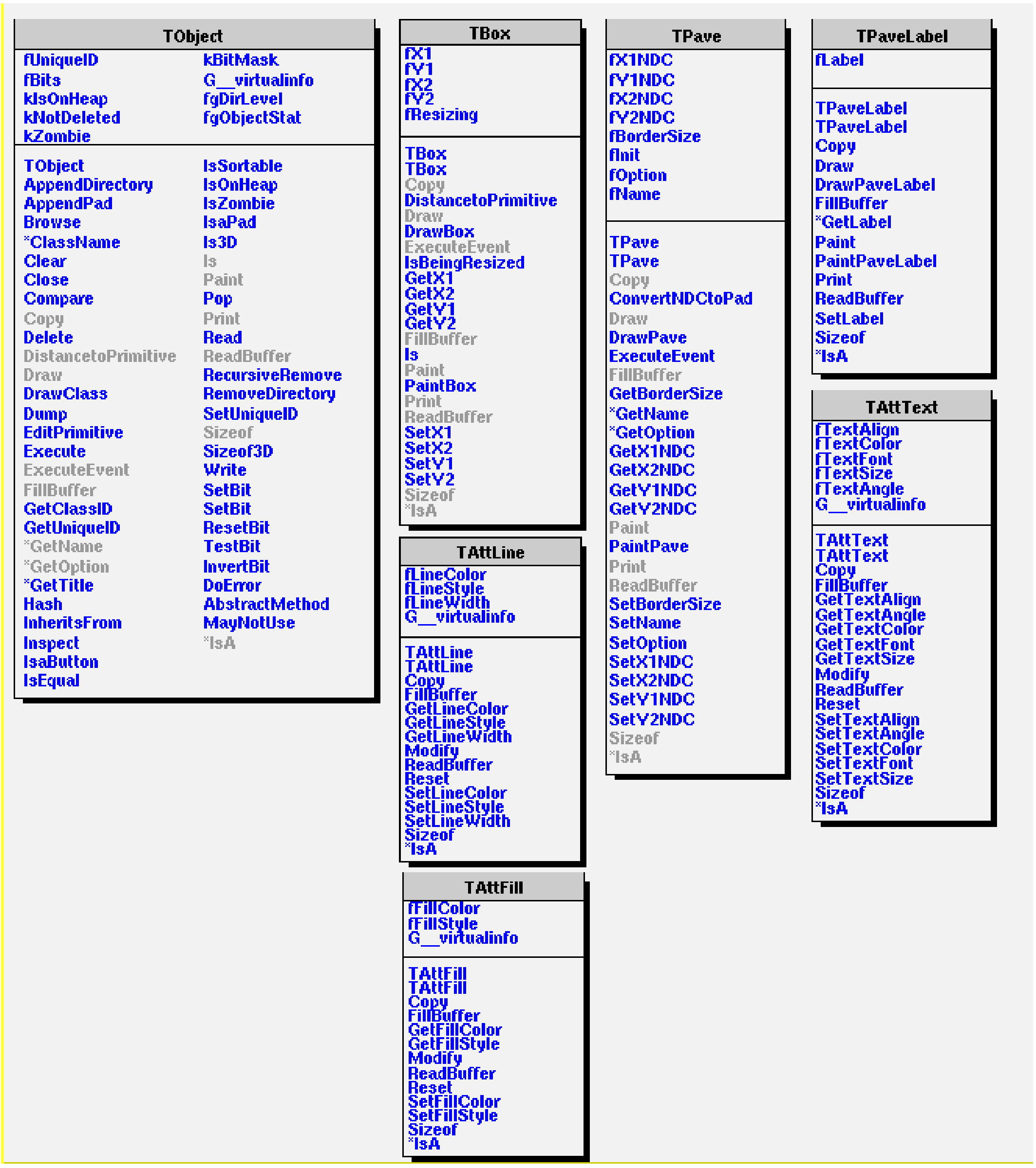
```
Computes distance from point (px,py) to the object.
This member function must be implemented for each graphics primitive.
This default function returns a big number (999999).
```

void **Draw**(Option_t **option*)

```
Default Draw method for all objects
```

void **DrawClass**( )

```
Draw class inheritance tree of the class to which this object belongs.
If a class B inherits from a class A, description of B is drawn
on the right side of description of A.
Member functions overridden by B are shown in class A with a blue line
crossing-out the corresponding member function.
The following picture is the class inheritance tree of class TPaveLabel:

/
```




```
*/
```

void **DrawClone**(Option_t **option*)

```
Draw a clone of this object in the current pad
```

void **Dump**( )

```
Dump contents of object on stdout.
Using the information in the object dictionary (class TClass)
each data member is interpreted.
If a data member is a pointer, the pointer value is printed

The following output is the Dump of a TArrow object:
fAngle                    0           Arrow opening angle (degrees)
fArrowSize                0.2         Arrow Size
fOption.*fData
fX1                       0.1         X of 1st point
fY1                       0.15        Y of 1st point
fX2                       0.67        X of 2nd point
fY2                       0.83        Y of 2nd point
fUniqueID                 0           object unique identifier
fBits                     50331648    bit field status word
fLineColor                1           line color
```

```
fLineStyle              1           line style
fLineWidth              1           line width
fFillColor              19          fill area color
fFillStyle              1001        fill area style
```

void **Execute**(Text_t **method,* Text_t **params*)

```
Execute method on this object with the given parameter string, e.g.
"3.14,1,\"text\"".
```

void **Execute**(TMethod **method,* TObjArray **params*)

```
Execute method on this object with parameters stored in the TObjArray.
The TObjArray should contain an argv vector like:

argv[0] ... argv[n] = the list of TObjString parameters
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to an event at (px,py). This method
must be overridden if an object can react to graphics events.
```

const Option_t* **GetDrawOption**( ) const

```
Get option used by the graphics system to draw this object.
```

const Text_t* **GetName**( ) const

```
Returns name of object. This default method returns the class name.
Classes that give objects a name should override this method.
```

UInt_t **GetUniqueID**( ) const

```
Return the unique object id.
```

Text_t* **GetObjectInfo**(Int_t *px,* Int_t *py*)

```
Returns string containing info about the object at position (px,py).
This method is typically overridden by classes of which the objects
can report peculiarities for different positions.
Returned string will be re-used (lock in MT environment).
```

const Text_t* **GetTitle**( ) const

```
Returns title of object. This default method returns the class title
(i.e. description). Classes that give objects a title should override
this method.
```

ULong_t **Hash**( )

```
Return hash value for this object.
```

Bool_t **InheritsFrom**(const Text_t **classname*)

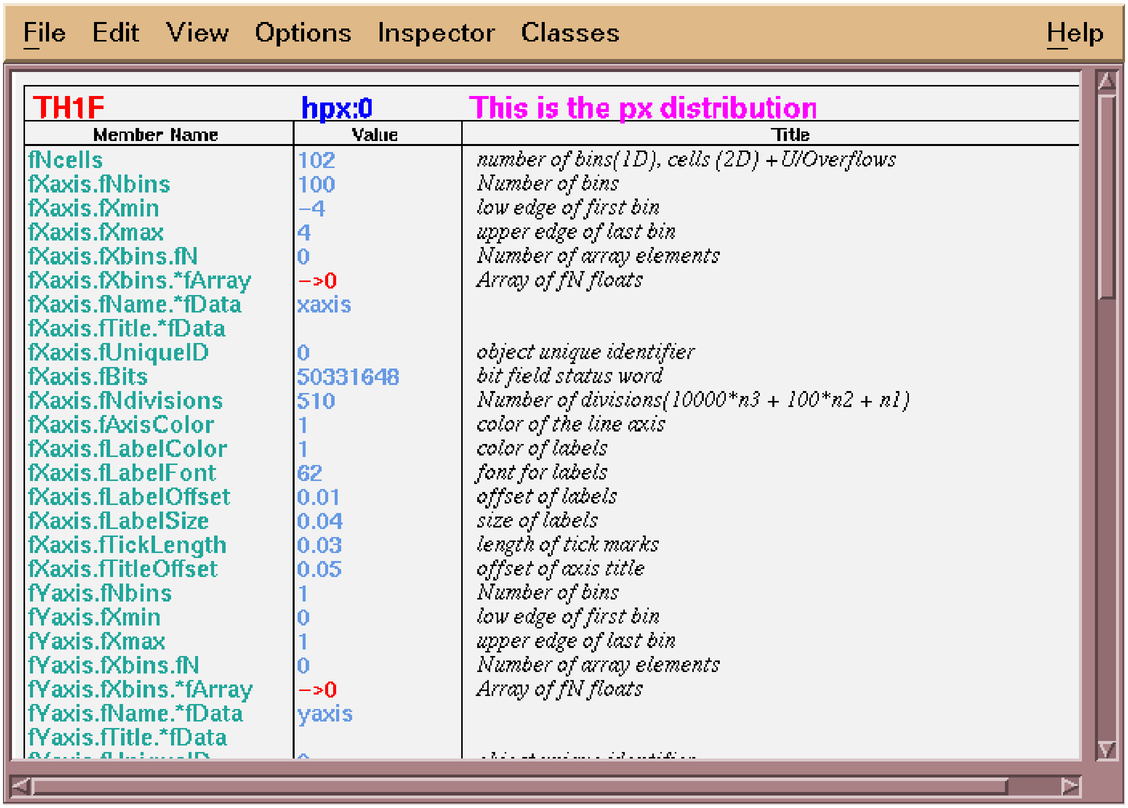
```
Returns kTRUE if object inherits from class "classname".
```

Bool_t **InheritsFrom**(const TClass **cl*)

```
Returns kTRUE if object inherits from TClass cl.
```

void **Inspect**( )

```
Dump contents of this object in a graphics canvas.
Same action as Dump but in a graphical form.
In addition pointers to other objects can be followed.

The following picture is the Inspect of a histogram object:

/
```



Member Name	Value	Title
fNcells	102	number of bins(1D), cells (2D) +U/Overflows
fXaxis.fNbins	100	Number of bins
fXaxis.fXmin	-4	low edge of first bin
fXaxis.fXmax	4	upper edge of last bin
fXaxis.fXbins.fN	0	Number of array elements
fXaxis.fXbins.*fArray	->0	Array of fN floats
fXaxis.fName.*fData	xaxis	
fXaxis.fTitle.*fData		
fXaxis.fUniqueID	0	object unique identifier
fXaxis.fBits	50331648	bit field status word
fXaxis.fNdivisions	510	Number of divisions(10000*n3 + 100*n2 + n1)
fXaxis.fAxisColor	1	color of the line axis
fXaxis.fLabelColor	1	color of labels
fXaxis.fLabelFont	62	font for labels
fXaxis.fLabelOffset	0.01	offset of labels
fXaxis.fLabelSize	0.04	size of labels
fXaxis.fTickLength	0.03	length of tick marks
fXaxis.fTitleOffset	0.05	offset of axis title
fYaxis.fNbins	1	Number of bins
fYaxis.fXmin	0	low edge of first bin
fYaxis.fXmax	1	upper edge of last bin
fYaxis.fXbins.fN	0	Number of array elements
fYaxis.fXbins.*fArray	->0	Array of fN floats
fYaxis.fName.*fData	yaxis	
fYaxis.fTitle.*fData		

*/

Bool_t **IsFolder**( )

    Returns kTRUE in case object contains browsable objects (like containers
    or lists of other objects).

Bool_t **Is3D**( )

    Returns kTRUE in case object is a 3D graphics object, in which case
    it can be displayed in a 3D viewer (like X3D or OpenGL).

Bool_t **IsEqual**(TObject **obj*)

    Default equal comparison (objects are equal if they have the same
    address in memory). More complicated classes might want to override
    this function.

void **ls**(Option_t **option*)

    The ls function lists the contents of a class on stdout. Ls output
    is typically much less verbose then Dump().

void **Paint**(Option_t **option*)

    This method must be overridden if a class wants to paint itself.
    The difference between Paint() and Draw() is that when a object
    draws itself it is added to the display list of the pad in
    which it is drawn (and automatically redrawn whenever the pad is
    redrawn). While paint just draws the object without adding it to
    the pad display list.

void **Pop**( )

    Pop on object drawn in a pad to the top of the display list. I.e. it
    will be drawn last and on top of all other primitives.

void **Print**(Option_t **option*)

    This method must be overridden when a class wants to print itself.

void **Read**(const Text_t **name*)

```
Read contents of object with specified name from the current directory.
First the key with the given name is searched in the current directory,
next the key buffer is deserialized into the object.
The object must have been created before via the default constructor.
See TObject::Write().
```

void **RecursiveRemove**(TObject **obj*)

```
Recursively remove this object from a list. Typically implemented
by classes that can contain mulitple references to a same object.
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save a primitive as a C++ statement(s) on output stream "out".
```

void **SetDrawOption**(const Option_t **option*)

```
Set drawing option for object. This option only affects
the drawing style and is stored in the option field of the
TObjOptLink supporting a TPad's primitive list (TList).
```

void **SetBit**(UInt_t *f,* Bool_t *set*)

```
Set or unset the user status bits as specified in f.
```

void **SetUniqueID**(UInt_t *uid*)

```
Set the unique object id.
```

void **Sizeof3D**( ) const

```
Set total size of this 3D object (used by X3D interface).
```

void **UseCurrentStyle**( )

```
Set current style settings in this object
This function is called when either TCanvas::UseCurrentStyle
or TROOT::ForceStyle have been invoked.
```

void **Write**(const Text_t **name,* Int_t *bufsize*)

```
Write this object to the current directory
The data structure corresponding to this object is serialized.
The corresponding buffer is written to the current directory
with an associated key with name "name".

Writing an object to a file involves the following steps:

-Creation of a support TKey object in the current directory.
The TKey object creates a TBuffer object.

-The TBuffer object is filled via the class::Streamer function.

-If the file is compressed (default) a second buffer is created to
hold the compressed buffer.

-Reservation of the corresponding space in the file by looking
in the TFree list of free blocks of the file.

-The buffer is written to the file.

Bufsize can be given to force a given buffer size to write this object.
By default, the buffersize will be taken from the average buffer size
of all objects written to the current file so far.

If a name is specified, it will be the name of the key.
If name is not given, the name of the key will be the name as returned
by GetName().

An object is read from the file into memory via TKey::Read() or
via TObject::Read().
```

void **Warning**(const char *location, const char ***va_**(*fmt*), ...) const

```
Issue warning message. Use "location" to specify the method where the
warning occured. Accepts standard printf formatting arguments.
```

void **Error**(const char *location, const char ***va_**(*fmt*), ...) const

```
Issue error message. Use "location" to specify the method where the
error occured. Accepts standard printf formatting arguments.
```

void **SysError**(const char *location, const char ***va_**(*fmt*), ...) const

```
Issue system error message. Use "location" to specify the method where
the system error occured. Accepts standard printf formatting arguments.
```

void **Fatal**(const char *location, const char ***va_**(*fmt*), ...) const

```
Issue fatal error message. Use "location" to specify the method where the
fatal error occured. Accepts standard printf formatting arguments.
```

void **AbstractMethod**(const char **method*) const

```
Use this method to implement an "abstract" method that you don't
want to leave purely abstract.
```

void **MayNotUse**(const char **method*) const

```
Use this method to signal that a method (defined in a base class)
may not be called in a derived class (in principle against good
design since a child class should not provide less functionality
than its parent, however, sometimes it is necessary).
```

Bool_t **GetObjectStat**( )

Int_t **GetDirLevel**( )

Int_t **DecreaseDirLevel**( )

Int_t **IncreaseDirLevel**( )

---

void **SetDirLevel**(Int_t *level*)

Author: René Brun 26/12/94
Last update: 1.03/06 24/10/97 17.24.57 by René Brun

TObjectTable

class **TObjectTable** : public TObject

private:

void	**Expand**(int *newsize*)
Int_t	**FindElement**(class TObject* *obj*)
void	**FixCollisions**(int *index*)
Bool_t	**HighWaterMark**( )

public:

class TObjectTable	**TObjectTable**(Int_t *tableSize = 100*)
class TObjectTable	**TObjectTable**(class TObjectTable&)
void	**~TObjectTable**( )
void	**Add**(class TObject* *obj*)
void	**AddObj**(class TObject* *obj*)
void*	**CheckPtrAndWarn**(char* *msg,* void* *vp*)
class TClass*	**Class**( )
void	**Delete**( )
Int_t	**GetSize**( )
Int_t	**Instances**( )
void	**InstanceStatistics**( )
virtual class TClass*	**IsA**( )
void	**Print**(Option_t* *option*)
Bool_t	**PtrIsValid**(class TObject* *obj*)
void	**Remove**(class TObject* *obj*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Statistics**( )
virtual void	**Streamer**(class TBuffer& *b*)
void	**Terminate**( )
void	**UpdateInstCount**( )

Data Members

private:

class TObject**	**fTable**
Int_t	**fSize**
Int_t	**fTally**

Class Description

```
This class registers all instances of TObject and its derived
classes in a hash table. The Add() and Remove() members are called
from the TObject ctor and dtor, repectively. Using the Print()
member one can see all currently active objects in the system.
Using the resource (in .rootrc): Root.ObjectStat one can toggle this
feature on or off.
Using the compile option NOSTATS one can de-active this feature
for the entire system (for maximum performance in highly time
critical applications).

The following output has been produced in a ROOT interactive session
via the command gObjectTable->Print()
class                     cnt    on heap     size    total size    heap size
TKey                        4          4       72           288          288
TClass                     84         84       80          6720         6720
TDataMember               276        276       24          6624         6624
TObject                    11         11       12           132          132
TMethod                  1974       1974       64        126336       126336
TDataType                  34         34       56          1904         1904
TList                    2328       2328       36         83808        83808
```

```
TH1F                       1         1      448         448         448
TText                   2688      2688       56      150528      150528
TGaxis                     1         0      120         120           0
TAxis                      6         3       88         528         264
TBox                      57        57       52        2964        2964
TLine                    118       118       40        4720        4720
TWbox                      1         1       56          56          56
TArrow                     1         1       64          64          64
TPaveText                 59        59      124        7316        7316
TPave                      1         1       92          92          92
TFile                      1         1      136         136         136
TCanvas                    3         3      444        1332        1332
TPad                       1         1      312         312         312
TContextMenu               3         3       48         144         144
TMethodArg              2166      2166       44       95304       95304
TPaveLabel                 1         1      120         120         120
THtml                      1         1       32          32          32
TROOT                      1         0      208         208           0
TApplication               1         1       28          28          28
TFileHandler               1         1       20          20          20
TColor                   163       163       40        6520        6520
TStyle                     1         1      364         364         364
TRealData                117       117       28        3276        3276
TBaseClass                88        88       36        3168        3168
THashList                  5         5       40         200         200
THashTable                 5         5       36         180         180
TGeometry                  1         1       64          64          64
TLink                      7         7       60         420         420
TPostScript                1         1      764         764         764
TMinuit                    1         1      792         792         792
TStopwatch                 1         0       56          56           0
TMotifGuiFactory           1         1       28          28          28
TGX11                      1         1      172         172         172
TUnixSystem                1         1      252         252         252
TSignalHandler             1         1       20          20          20
TOrdCollection             3         3       40         120         120
TEnv                       1         1       24          24          24
TCint                      1         1      208         208         208
TBenchmark                 1         1       52          52          52
TClassTable                1         1       12          12          12
TObjectTable               1         1       12          12          12
-----------------------------------------------------------------------
Total:                 10225     10219     5976      506988      506340
```

TObjectTable(Int_t *tableSize*)

    Create an object table.

~TObjectTable( )

    Delete TObjectTable.

void **Print**(Option_t *)

    Print the object table.

void **Add**(TObject **op*)

    Add an object to the object table.

void **AddObj**(TObject **op*)

    Add an object to the global object table gObjectTable. If the global
    table does not exist create it first. This member function may only
    be used by TObject::TObject. Use Add() to add objects to any other
    TObjectTable object. This is a static function.

void **Delete**( )

`Delete all objects stored in the TObjectTable.`

void **Remove**(TObject **op*)

`Remove an object from the object table.`

void **Terminate**( )

`Deletes the object table (this static class function calls the dtor).`

Int_t **FindElement**(TObject **op*)

```
Find an object in the object table. Returns the slot where to put
the object. To test if the object is actually already in the table
use PtrIsValid().
```

void **FixCollisions**(Int_t *index*)

`Rehash the object table in case an object has been removed.`

void **Expand**(Int_t *newSize*)

`Expand the object table.`

void **InstanceStatistics**( )

`Print the object table.`

void **UpdateInstCount**( )

`Histogram all objects according to their classes.`

void* **CheckPtrAndWarn**(const char **msg,* void **vp*)

```
Issue a warning in case an object still appears in the table
while it should not.
```

Author: Fons Rademakers 11/08/95
Last update: 1.01/05 11/06/97 20.58.27 by Fons Rademakers

TObjString

class **TObjString** : public TObject

public:

class TObjString	**TObjString**(char* *s*)
class TObjString	**TObjString**(class TObjString& *s*)
void	**~TObjString**( )
class TClass*	**Class**( )
Int_t	**Compare**(class TObject* *obj*)
void	**FillBuffer**(char*& *buffer*)
const Text_t*	**GetName**( )
class TString	**GetString**( )
ULong_t	**Hash**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsEqual**(class TObject* *obj*)
Bool_t	**IsSortable**( )
void	**Print**(Option_t* *opt*)
void	**ReadBuffer**(char*& *buffer*)
void	**SetString**(char* *s*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)
class TString&	**String**( )

Data Members

private:

class TString	**fString**	*wrapped TString*

Class Description

```
TObjString

Collectable string class. This is a TObject containing a TString.
```

Bool_t **IsEqual**(TObject **obj*)

Author: Fons Rademakers 12/11/95
Last update: 1.01/04 29/05/97 13.03.03 by Fons Rademakers

TOptionMenu

class **TOptionMenu** : public TMenu

private:

void	**CreateOptionMenu**(Widget *parent,* MenuDesc_t* *memuDesc,* XtPointer *defaultClientDat,* Arg* *geometryManagement = NULL,* Int_t *numArgs = 0*)

public:

void	**~TOptionMenu**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
void	**ForceWidth**(int *width*)
int	**GetIndex**( )
class TMenuItem*	**GetItem**( )
virtual Int_t	**GetSavedValue**( )
virtual Int_t	**GetSelectedValue**( )
void	**Instantiate**(Widget *parent,* Arg* *geometryManagement,* Int_t *numArgs*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
Widget	**Pulldown**( )
void	**Set**(class TMenuItem* *item*)
void	**Set**(char* *name*)
void	**Set**(int *index*)
virtual void	**SetLabel**(char* *str*)
virtual void	**SetSavedValue**(Int_t *value*)
virtual void	**SetSelectedValue**(Int_t *value*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Widget	**fPulldown**
Int_t	**fSelectedValue**
Int_t	**fSavedValue**

Class Description

```
TOptionMenu

This class defines an option menu.
```

Widget **Pulldown**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 04/09/97 22.02.35 by Piotr Golonka

TOrdCollection

class **TOrdCollection** : public TSeqCollection

private:

Bool_t	**IllegalIndex**(char* *method,* Int_t *idx*)
void	**Init**(Int_t *capacity*)
Int_t	**LogIndex**(Int_t *idx*)
Bool_t	**LowWaterMark**( )
void	**MoveGapTo**(Int_t *newGapStart*)
Int_t	**PhysIndex**(Int_t *idx*)
void	**SetCapacity**(Int_t *newCapacity*)

public:

class TOrdCollection	**TOrdCollection**(class TOrdCollection&)
class TOrdCollection	**TOrdCollection**(Int_t *capacity = kDefaultCapacity*)
void	**~TOrdCollection**( )
virtual void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
virtual void	**AddAt**(class TObject* *obj,* Int_t *idx*)
virtual void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj*)
virtual void	**AddLast**(class TObject* *obj*)
virtual class TObject*	**After**(class TObject* *obj*)
virtual class TObject*	**At**(Int_t *idx*)
virtual class TObject*	**Before**(class TObject* *obj*)
Int_t	**BinarySearch**(class TObject* *obj*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Delete**(Option_t* *option*)
virtual class TObject*	**First**( )
virtual Int_t	**IndexOf**(class TObject* *obj*)
virtual class TClass*	**IsA**( )
virtual class TObject*	**Last**( )
virtual class TIterator*	**MakeIterator**(Bool_t *dir = kIterForward*)
void	**PutAt**(class TObject* *obj,* Int_t *idx*)
virtual class TObject*	**Remove**(class TObject* *obj*)
virtual void	**RemoveAt**(Int_t *idx*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Sort**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TObject**	**fCont**
Int_t	**fCapacity**
Int_t	**fGapStart**
Int_t	**fGapSize**

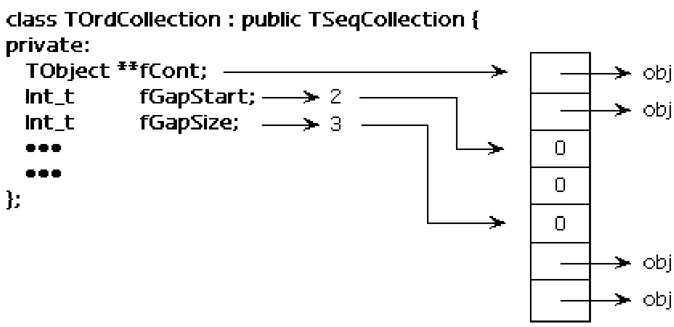
public:

static const enum	**kDefaultCapacity**
static const enum	**kMinExpand**
static const enum	**kShrinkFactor**

Class Description

```
TOrdCollection

Ordered collection. An ordered collection has TList insertion
semantics but is implemented using an array of TObject*'s. It uses
```

less space than a TList (since there is no need for the prev and next pointers), but it is more costly to insert objects (since it has to create a gap by copying object pointers). TOrdCollection is better than TList when objects are only added at the end of the collection since no copying needs to be done.



TOrdCollection(Int_t *capacity*)
  Create an ordered collection.

~TOrdCollection( )
  Delete the collection. Objects are not deleted.

void **AddAt**(TObject **obj,* Int_t *idx*)
  Insert object at position idx in the collection.

void **AddFirst**(TObject **obj*)
  Insert object at beginning of collection.

void **AddLast**(TObject **obj*)
  Add object at the end of the collection.

void **AddBefore**(TObject **before,* TObject **obj*)
  Insert object before object before in the collection.

void **AddAfter**(TObject **after,* TObject **obj*)
  Insert object after object after in the collection.

TObject* **After**(TObject **obj*) const
  Return the object after object obj. Returns 0 if obj is last in collection.

TObject* **At**(Int_t *idx*) const
  Returns the object at position idx. Returns 0 if idx is out of range.

TObject* **Before**(TObject **obj*) const
  Returns the object before object obj. Returns 0 if obj is first in collection.

void **Clear**(Option_t **option*)
  Remove all objects from the collection. Does not delete the objects.

void **Delete**(Option_t **option*)
  Remove all objects from the collection AND delete all heap based objects.

TObject* **First**( ) const
  Return the first object in the collection. Returns 0 when collection is empty.

TObject* **Last**( ) const
  Return the last object in the collection. Returns 0 when collection

`is empty.`

Bool_t **IllegalIndex**(const char *method, Int_t *idx*) const
`Return true when index out of bounds and print error.`

Int_t **IndexOf**(TObject **obj*) const
`Return index of object in collection. Returns -1 when object not found.`
`Uses member IsEqual() to find object.`

void **Init**(Int_t *capacity*)
`Initialize ordered collection.`

TIterator* **MakeIterator**(Bool_t *dir*) const
`Return an ordered collection iterator.`

void **MoveGapTo**(Int_t *start*)
`Move gap to new position. Gap needs to be moved when objects are`
`inserted not at the end.`

void **PutAt**(TObject **obj,* Int_t *idx*)
`Put object at index idx. Overwrites what was at idx before.`

void **RemoveAt**(Int_t *idx*)
`Remove object at index idx.`

TObject* **Remove**(TObject **obj*)
`Remove object from collection.`

void **SetCapacity**(Int_t *newCapacity*)
`Set/change ordered collection capacity.`

void **Sort**( )
`If objects in collection are sortable (i.e. IsSortable() returns true`
`for all objects) then sort collection.`

Int_t **BinarySearch**(TObject **obj*)
`Find object using a binary search. Collection must first have been`
`sorted.`

Author: Fons Rademakers 13/09/95
Last update: 1.03/04 29/09/97 19.15.55 by Fons Rademakers

TOrdCollectionIter

class **TOrdCollectionIter** : public TIterator

private:
class TOrdCollectionIter **TOrdCollectionIter**( )
public:
class TOrdCollectionIter **TOrdCollectionIter**(class TOrdCollection* *col,* Bool_t *dir* = *kIterForward*)
class TOrdCollectionIter **TOrdCollectionIter**(class TOrdCollectionIter&)
void **~TOrdCollectionIter**( )
class TClass* **Class**( )
virtual const class TCollection* **GetCollection**( )
virtual class TClass* **IsA**( )
virtual class TObject* **Next**( )
virtual void **Reset**( )
virtual void **ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void **Streamer**(class TBuffer& *b*)

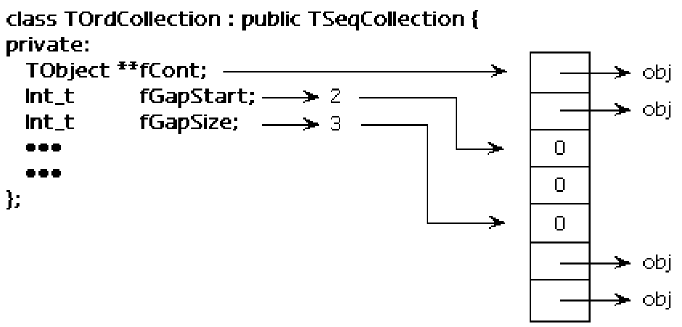
Data Members

private:

const class TOrdCollection*	**fCol**	*collection being iterated*
Int_t	**fCursor**	*current position in collection*
Bool_t	**fDirection**	*iteration direction*

Class Description

```
TOrdCollection

Ordered collection. An ordered collection has TList insertion
semantics but is implemented using an array of TObject*'s. It uses
less space than a TList (since there is no need for the prev and
next pointers), but it is more costly to insert objects (since it
has to create a gap by copying object pointers). TOrdCollection
is better than TList when objects are only added at the end of the
collection since no copying needs to be done.
```



TOrdCollectionIter(const TOrdCollection **col,* Bool_t *dir*)

```
Create collection iterator. By default the iteration direction
is kIterForward. To go backward use kIterBackward.
```

TObject* **Next**( )

```
Return next object in collection. Returns 0 when no more objects in
collection.
```

void **Reset**( )

```
Reset collection iterator.
```

Author: Fons Rademakers 13/09/95
Last update: 1.03/04 29/09/97 19.15.55 by Fons Rademakers

TPacketGenerator

class **TPacketGenerator** : public TObject

private:

class TPacketGenerator	**TPacketGenerator**( )
class TPacketGenerator	**TPacketGenerator**(class TPacketGenerator&)
void	**Init**( )
void	***operator=***(class TPacketGenerator&)

public:

class TPacketGenerator	**TPacketGenerator**(Stat_t *firstEvent,* Stat_t *lastEvent,* class TTree* *tree,* class TList* *slaves*)
class TPacketGenerator	**TPacketGenerator**(Stat_t *firstEvent,* Stat_t *lastEvent,* Int_t *packetSize,* class TList* *slaves*)
void	**~TPacketGenerator**( )
class TClass*	**Class**( )
Stat_t	**GetEventsProcessed**( )
Stat_t	**GetEventsProcessed**(class TSlave* *sl*)
Int_t	**GetInitialPacketSize**( )
Stat_t	**GetLastEvent**( )
Stat_t	**GetNextEvent**( )
Bool_t	**GetNextPacket**(class TSlave* *sl,* Int_t& *nevent,* Stat_t& *first*)
Int_t	**GetNoPackets**( )
class TList*	**GetPackets**( )
virtual class TClass*	**IsA**( )
void	**Print**(Option_t* *option*)
void	**Reset**(Stat_t *firstEvent,* Stat_t *lastEvent,* class TList* *slaves*)
void	**SetMinPacketSize**(Int_t *minps*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Stat_t	**fNextEvent**	*next event to be processed*
Stat_t	**fLastEvent**	*last event to be processed*
Stat_t	**fEventsProcessed**	*number of events processed*
Int_t	**fInitialPacketSize**	*initial packet size*
Int_t	**fMinPacketSize**	*minimum packet size*
Int_t	**fSmallestPacket**	*smallest packet generated*
class TTime	**fMinTimeDiff**	*time difference between last two packets for fastest slave*
Int_t	**fMaxOrd**	*highest ordinal slave number*
class TTree*	**fTree**	*tree or chain being processed*
class TList*	**fSlaves**	*list of slaves*
Stat_t*	**fEventsPerSlave**	*array containing number of events processed per slave*
class TObjArray*	**fLastPackets**	*pointer to the last packet for each slave*
class TList*	**fPackets**	*list of packets that have been processed*

Class Description

```
TPacketGenerator

This class generates packets to be processed on PROOF slave servers.
A packet is an event range (begin event and number of events).
Packets are generated taking into account the performance of the
remote machine, the time it took to process a previous packet on
the remote machine, the locality of the database files, etc.
```

TPacketGenerator( )

```
Private default ctor.
```

TPacketGenerator(Stat_t *firstEvent,* Stat_t *lastEvent,* TTree **tree,* TList **slaves*)

```
Initialize TPacketGenerator using a TTree and a list of slave processors.
```

TPacketGenerator(Stat_t *firstEvent,* Stat_t *lastEvent,* Int_t *packetSize,* TList **slaves*)

```
Initialize TPacketGenerator using the total number of events, the
initial packet size and the list of slaves.
```

~**TPacketGenerator**( )

```
TPacketGenerator dtor. Deletes the list of packets.
```

void **Init**( )

```
Initialize the TPacketGenerator. Called by the ctors.
```

Stat_t **GetEventsProcessed**(TSlave **sl*) const

```
Return number of events processed by slave sl.
```

Bool_t **GetNextPacket**(TSlave **sl,* Int_t *&nevent,* Stat_t *&first*)

```
Get next packet for slave sl. Returns kTRUE and sets first and nevent.
In case no more packets return kFALSE (and set first and nevent == 0).
The packet size for a slave is a linear function of the time diff between
the processing of two packets relative to the fastest slave (i.e. the one
with the smallest time diff). The fastest slave always processes
packets of size fTree->GetPacketSize() (or of the size specified in
the TPacketGenerator ctor).
```

Int_t **GetNoPackets**( ) const

```
Number of packets generated.
```

void **Print**(Option_t **option*)

```
Prints info about the packet generator. If option is "all", print also
info about all packets.
```

void **Reset**(Stat_t *firstEvent,* Stat_t *lastEvent,* TList **slaves*)

```
Reset the packet generator for a new cycle.
```

void **SetMinPacketSize**(Int_t *minps*)

```
Set minimum packet size. The default is 20% of the initial packet size
(in case of TTree’s the initial packet size is set in via
TTree::SetPacketSize() and its default is 100).
```

Author: Fons Rademakers 28/03/97
Last update: 1.00/07 01/04/97 15.42.35 by Fons Rademakers

TPad

class **TPad** : public TVirtualPad

public:

class TPad	**TPad**(Text_t* *name,* Text_t* *title,* Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup,* Color_t *color = -1,* Short_t *bordersize = -1,* Short_t *bordermode = -2*)
class TPad	**TPad**( )
class TPad	**TPad**(class TPad& *pad*)
void	**~TPad**( )
virtual void	**AbsCoordinates**(Bool_t *set*)
virtual Float_t	**AbsPixeltoX**(Int_t *px*)
virtual void	**AbsPixeltoXY**(Int_t *xpixel,* Int_t *ypixel,* Axis_t& *x,* Axis_t& *y*)
virtual Float_t	**AbsPixeltoY**(Int_t *py*)
virtual void	**Browse**(class TBrowser* *b*)
virtual void	**cd**(Int_t *subpadnumber = 0*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Close**(Option_t* *option*)
virtual void	**Copy**(class TObject& *pad*)
virtual void	**CopyPixmap**( )
virtual void	**CopyPixmaps**( )
void	**CreateNewArrow**(Int_t *event,* Int_t *px,* Int_t *py*)
void	**CreateNewEllipse**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)
void	**CreateNewPad**(Int_t *event,* Int_t *px,* Int_t *py*)
void	**CreateNewPave**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)
void	**CreateNewPolyLine**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)
void	**CreateNewText**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual void	**Divide**(Int_t *nx = 1,* Int_t *ny = 1,* Float_t *xmargin* = 0.*01,* Float_t *ymargin* = 0.*01,* Int_t *color = 0*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Text_t* *title*)
virtual void	**DrawLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* const Float_t *y2*)
virtual void	**DrawLineNDC**(Float_t *u1,* Float_t *v1,* Float_t *u2,* Float_t *v2*)
virtual void	**DrawText**(Float_t *x,* Float_t *y,* Text_t* *text*)
virtual void	**DrawTextNDC**(Float_t *u,* Float_t *v,* Text_t* *text*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Float_t	**GetAbsHNDC**( )
virtual Float_t	**GetAbsWNDC**( )
virtual Float_t	**GetAbsXlowNDC**( )
virtual Float_t	**GetAbsYlowNDC**( )
virtual class TCanvas*	**GetCanvas**( )
virtual Int_t	**GetCanvasID**( )
virtual class TFrame*	**GetFrame**( )
virtual Bool_t	**GetGridx**( )
virtual Bool_t	**GetGridy**( )
virtual Color_t	**GetHighLightColor**( )
virtual Float_t	**GetHNDC**( )
virtual class TList*	**GetListOfPrimitives**( )
virtual Int_t	**GetLogx**( )
virtual Int_t	**GetLogy**( )
virtual Int_t	**GetLogz**( )
virtual class TVirtualPad*	**GetMother**( )
virtual const Text_t*	**GetName**( )
virtual Int_t	**GetNumber**( )

virtual Int_t	**GetPadPaint**( )
virtual void	**GetPadPar**(Float_t& *xlow,* Float_t& *ylow,* Float_t& *xup,* Float_t& *yup*)
virtual class TObject*	**GetPadPointer**( )
virtual class TVirtualPad*	**GetPadSave**( )
virtual Float_t	**GetPhi**( )
virtual Int_t	**GetPixmapID**( )
virtual class TObject*	**GetPrimitive**(Text_t* *name*)
virtual void	**GetRange**(Float_t& *x1,* Float_t& *y1,* Float_t& *x2,* Float_t& *y2*)
virtual void	**GetRangeAxis**(Axis_t& *xmin,* Axis_t& *ymin,* Axis_t& *xmax,* Axis_t& *ymax*)
virtual class TObject*	**GetSelected**( )
virtual class TVirtualPad*	**GetSelectedPad**( )
virtual Float_t	**GetTheta**( )
virtual const Text_t*	**GetTitle**( )
virtual class TView*	**GetTView**( )
virtual Float_t	**GetUxmax**( )
virtual Float_t	**GetUxmin**( )
virtual Float_t	**GetUymax**( )
virtual Float_t	**GetUymin**( )
virtual class TView*	**GetView**( )
virtual class TPadView3D*	**GetView3D**( )
virtual UInt_t	**GetWh**( )
virtual Float_t	**GetWNDC**( )
virtual UInt_t	**GetWw**( )
virtual Float_t	**GetXlowNDC**( )
virtual Float_t	**GetYlowNDC**( )
virtual void	**HighLight**(Color_t *col* = *kRed,* Bool_t *set* = *kTRUE*)
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsBatch**( )
virtual Bool_t	**IsFolder**( )
virtual Bool_t	**IsModified**( )
virtual Bool_t	**IsRetained**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Modified**(Bool_t *flag* = *1*)
virtual Bool_t	**OpaqueMoving**( )
virtual Bool_t	**OpaqueResizing**( )
virtual Float_t	**PadtoX**(Axis_t *x*)
virtual Float_t	**PadtoY**(Axis_t *y*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintBox**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Option_t* *option*)
virtual void	**PaintFillArea**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**PaintLine3D**(Float_t* *p1,* Float_t* *p2*)
virtual void	**PaintLineNDC**(Coord_t *u1,* Coord_t *v1,* Coord_t *u2,* Coord_t *v2*)
virtual void	**PaintModified**( )
virtual void	**PaintPadFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
virtual void	**PaintPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintPolyLine3D**(Int_t *n,* Float_t* *p*)
virtual void	**PaintPolyLineNDC**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintText**(Float_t *x,* Float_t *y,* Text_t* *text*)
virtual void	**PaintTextNDC**(Float_t *u,* Float_t *v,* Text_t* *text*)
virtual class TPad*	**Pick**(Int_t px, Int_t py, (*unknown*) TObjLink*& pickobj)
virtual Float_t	**PixeltoX**(Int_t *px*)
virtual void	**PixeltoXY**(Int_t *xpixel,* Int_t *ypixel,* Axis_t& *x,* Axis_t& *y*)
virtual Float_t	**PixeltoY**(Int_t *py*)
virtual void	**Pop**( )
virtual void	**Print**(Text_t* *filename*)

virtual void	**Print**(Text_t* *filename,* Option_t* *option*)
virtual void	**Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**RangeAxis**(Axis_t *xmin,* Axis_t *ymin,* Axis_t *xmax,* Axis_t *ymax*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**ResetView3D**(class TPadView3D* *view = 0*)
virtual void	**ResizePad**(Option_t* *option*)
virtual void	**SaveAs**(Text_t* *filename*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetBorderMode**(Short_t *bordermode*)
virtual void	**SetBorderSize**(Short_t *bordersize*)
virtual void	**SetCanvas**(UInt_t *ww,* UInt_t *wh*)
virtual void	**SetCursor**((*unknown*) ECursor cursor)
virtual void	**SetDoubleBuffer**(Int_t *mode = 1*)
virtual void	**SetDrawOption**(Option_t* *option*)
virtual void	**SetGrid**(Int_t *valuex = 1,* Int_t *valuey = 1*)
virtual void	**SetGridx**(Int_t *value = 1*)
virtual void	**SetGridy**(Int_t *value = 1*)
virtual void	**SetLogx**(Int_t *value = 1*)
virtual void	**SetLogy**(Int_t *value = 1*)
virtual void	**SetLogz**(Int_t *value = 1*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetNumber**(Int_t *number*)
virtual void	**SetPad**(Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup*)
virtual void	**SetPad**(Text_t* *name,* Text_t* *title,* Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup,* Color_t *color = 35,* Short_t *bordersize = 5,* Short_t *bordermode = -1*)
virtual void	**SetPadPointer**(class TObject* *padpointer = 0*)
virtual void	**SetPhi**(Float_t *phi = 30*)
virtual void	**SetSelected**(class TObject* *obj*)
virtual void	**SetTheta**(Float_t *theta = 30*)
virtual void	**SetTitle**(Text_t* *title*)
virtual void	**SetView**(class TView* *view*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Update**( )
virtual void	**UseCurrentStyle**( )
virtual Int_t	**UtoAbsPixel**(Float_t *u*)
virtual Int_t	**UtoPixel**(Float_t *u*)
virtual Int_t	**VtoAbsPixel**(Float_t *v*)
virtual Int_t	**VtoPixel**(Float_t *v*)
virtual void	**x3d**(Option_t* *option*)
virtual Int_t	**XtoAbsPixel**(Axis_t *x*)
virtual Float_t	**XtoPad**(Axis_t *x*)
virtual Int_t	**XtoPixel**(Axis_t *x*)
virtual void	**XYtoAbsPixel**(Axis_t *x,* Axis_t *y,* Int_t& *xpixel,* Int_t& *ypixel*)
virtual void	**XYtoPixel**(Axis_t *x,* Axis_t *y,* Int_t& *xpixel,* Int_t& *ypixel*)
virtual Int_t	**YtoAbsPixel**(Axis_t *y*)
virtual Float_t	**YtoPad**(Axis_t *y*)
virtual Int_t	**YtoPixel**(Axis_t *y*)

Data Members

protected:

Int_t	**fLogx**	*(=0 if X linear scale, =1 if log scale)*
Int_t	**fLogy**	*(=0 if Y linear scale, =1 if log scale)*
Int_t	**fLogz**	*(=0 if Z linear scale, =1 if log scale)*
Float_t	**fXtoAbsPixelk**	*Conversion coefficient for X World to absolute pixel*
Float_t	**fXtoPixelk**	*Conversion coefficient for X World to pixel*
Float_t	**fXtoPixel**	*xpixel = fXtoPixelk + fXtoPixel*xworld*

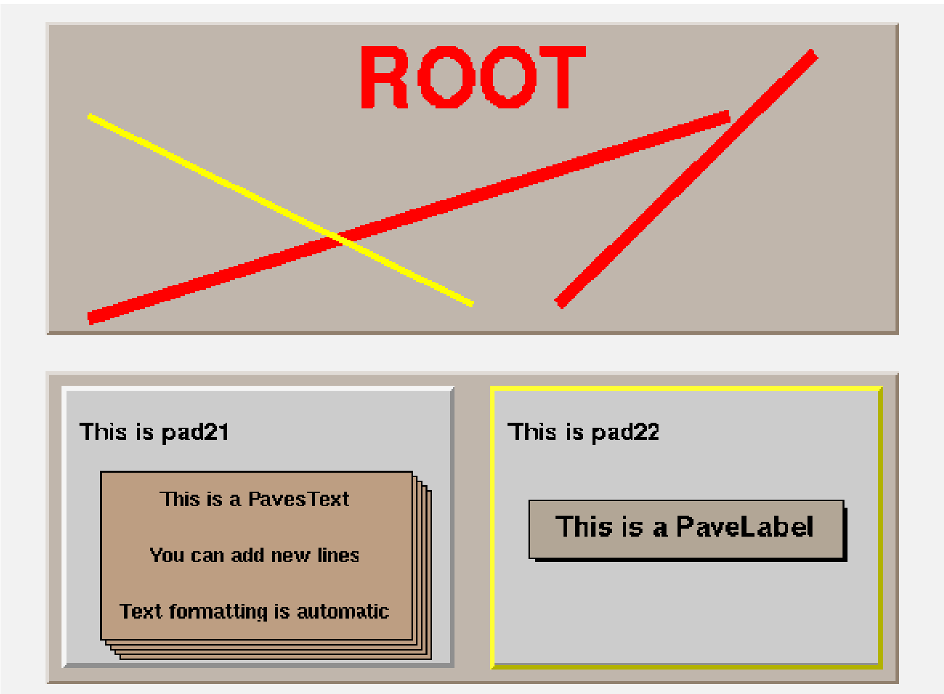
Float_t	**fYtoAbsPixelk**	*Conversion coefficient for Y World to absolute pixel*
Float_t	**fYtoPixelk**	*Conversion coefficient for Y World to pixel*
Float_t	**fYtoPixel**	*ypixel = fYtoPixelk + fYtoPixel*yworld*
Float_t	**fUtoAbsPixelk**	*Conversion coefficient for U NDC to absolute pixel*
Float_t	**fUtoPixelk**	*Conversion coefficient for U NDC to pixel*
Float_t	**fUtoPixel**	*xpixel = fUtoPixelk + fUtoPixel*undc*
Float_t	**fVtoAbsPixelk**	*Conversion coefficient for Y World to absolute pixel*
Float_t	**fVtoPixelk**	*Conversion coefficient for Y World to pixel*
Float_t	**fVtoPixel**	*ypixel = fVtoPixelk + fVtoPixel*vndc*
Float_t	**fAbsPixeltoXk**	*Conversion coefficient for absolute pixel to X World*
Float_t	**fPixeltoXk**	*Conversion coefficient for pixel to X World*
Float_t	**fPixeltoX**	*xworld = fPixeltoXk + fPixeltoX*xpixel*
Float_t	**fAbsPixeltoYk**	*Conversion coefficient for absolute pixel to Y World*
Float_t	**fPixeltoYk**	*Conversion coefficient for pixel to Y World*
Float_t	**fPixeltoY**	*yworld = fPixeltoYk + fPixeltoY*ypixel*
Float_t	**fXlowNDC**	*X bottom left corner of pad in NDC [0, 1]*
Float_t	**fYlowNDC**	*Y bottom left corner of pad in NDC [0, 1]*
Float_t	**fWNDC**	*Width of pad along X in NDC*
Float_t	**fHNDC**	*Height of pad along Y in NDC*
Float_t	**fAbsXlowNDC**	*Absolute X top left corner of pad in NDC [0, 1]*
Float_t	**fAbsYlowNDC**	*Absolute Y top left corner of pad in NDC [0, 1]*
Float_t	**fAbsWNDC**	*Absolute Width of pad along X in NDC*
Float_t	**fAbsHNDC**	*Absolute Height of pad along Y in NDC*
Float_t	**fUxmin**	*Minimum value on the X axis*
Float_t	**fUymin**	*Minimum value on the Y axis*
Float_t	**fUxmax**	*Maximum value on the X axis*
Float_t	**fUymax**	*Maximum value on the Y axis*
class TPad*	**fMother**	*pointer to mother of the list*
class TCanvas*	**fCanvas**	*!Pointer to mother canvas*
Int_t	**fPixmapID**	*Off-screen pixmap identifier*
class TList*	**fPrimitives**	*List of primitives (subpads)*
class TString	**fName**	*Pad name*
class TString	**fTitle**	*Pad title*
Int_t	**fPadPaint**	*Set to 1 while painting the pad*
Bool_t	**fModified**	*Set to true when pad is modified*
Bool_t	**fGridx**	*Set to true if grid along X*
Bool_t	**fGridy**	*Set to true if grid along Y*
class TFrame*	**fFrame**	*Pointer to 2-D frame (if one exists)*
class TView*	**fView**	*Pointer to 3-D view (if one exists)*
Float_t	**fTheta**	*theta angle to view as lego/surface*
Float_t	**fPhi**	*phi angle to view as lego/surface*
class TObject*	**fPadPointer**	*free pointer*
Int_t	**fNumber**	*pad number identifier*
Bool_t	**fAbsCoord**	*Use absolute coordinates*
class TPadView3D*	**fPadView3D**	*3D View of this TPad*

See Also: TButton, TCanvas, TSlider

Class Description

```
The Pad class is the most important graphics class in the ROOT system.
A Pad is contained in a Canvas.
A Pad may contain other pads (unlimited pad hierarchy).
A pad is a linked list of primitives of any type (graphics objects,
histograms, detectors, tracks, etc.).
Adding a new element into a pad is in general performed by the Draw
member function of the object classes.
It is important to realize that the pad is a linked list of references
to the original object.
```

```
For example, in case of an histogram, the histogram.Draw() operation
only stores a reference to the histogram object and not a graphical
representation of this histogram.
When the mouse is used to change (say the bin content), the bin content
of the original histogram is changed !!

The convention used in ROOT is that a Draw operation only adds
a reference to the object. The effective drawing is performed when
when the canvas receives a signal to be painted.
This signal is generally sent when typing carriage return in the
command input or when a graphical operation has been performed on one
of the pads of this canvas.
When a Canvas/Pad is repainted, the member function Paint for all
objects in the Pad linked list is invoked.

When the mouse is moved on the Pad, The member function DistancetoPrimitive
is called for all the elements in the pad. DistancetoPrimitive returns
the distance in pixels to this object.
when the object is within the distance window, the member function
ExecuteEvent is called for this object.
in ExecuteEvent, move, changes can be performed on the object.
For examples of DistancetoPrimitive and ExecuteEvent functions,
see classes TLine::DistancetoPrimitive, TLine::ExecuteEvent
           TBox::DistancetoPrimitive,  TBox::ExecuteEvent
           TH1::DistancetoPrimitive,   TH1::ExecuteEvent

A Pad supports linear and log scales coordinate systems.
The transformation coefficients are explained in TPad::ResizePad.
An example of pads hierarchy is shown below:
```



TPad( ): **TVirtualPad**( )

```
Pad default constructor
```

TPad(const Text_t **name,* const Text_t **title,* Float_t *xlow,*Float_t ylow, Float_t xup, Float_t yup,Color_t color, Short_t bordersize, Short_t *bordermode*): **TVirtualPad**(*name, title, xlow, ylow, xup, yup, color, bordersize, bordermode*)

```
Pad constructor
```

```
A pad is a linked list of primitives.
A pad is contained in a canvas. It may contain other pads.
A pad has attributes. When a pad is created, the attributes
defined in the current style are copied to the pad attributes.

xlow [0,1] is the position of the bottom left point of the pad
           expressed in the mother pad reference system
ylow [0,1] is the Y position of this point.
xup  [0,1] is the x position of the top right point of the pad
           expressed in the mother pad reference system
yup  [0,1] is the Y position of this point.

See TWbox::TWbox for the other parameters.
```

~TPad( )

```
Pad destructor
```

void **cd**(Int_t *subpadnumber*)

```
Set Current Pad
When a canvas/pad is divided via TPad::Divide, one can directly
set the current path to one of the subdivisions.
See TPad::Divide for the convention to number subpads.
For example:
 c1.Divide(2,3); // create 6 pads (2 divisions along x, 3 along y).
 To set the current pad to the bottom right pad, do
 c1.cd(6);
Note that c1.cd() is equivalent to c1.cd(0) and sets the current pad
to c1 itself.
```

void **Clear**(Option_t **option*)

```
Delete all pad primitives
```

void **Close**(Option_t **option*)

```
Delete all primitives in pad and pad itself
NB. Pad cannot be used anymore after this call
```

void **Copy**(TObject &*obj*)

```
Copy content of pad into another pad
```

void **CopyPixmap**( )

```
Copy the pixmap of the pad to the canvas
```

void **CopyPixmaps**( )

```
Copy the sub-pixmaps of the pad to the canvas
```

void **CreateNewArrow**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Create a new arrow in this pad
Function called by TPad::ExecuteEvent when the action "Create NewArrow"
has been selected in the Canvas pulldown menu.

Click left button to indicate arrow starting position.
Release left button to terminate the arrow.
```

void **CreateNewEllipse**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)

```
Create a new arc/ellipse in this pad
Function called by TPad::ExecuteEvent when the action "Create NewEllipse"
has been selected in the Canvas pulldown menu.

Click left button to indicate arrow starting position.
Release left button to terminate the arrow.
```

void **CreateNewPad**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Create a new pad in this pad
```

```
Function called by TPad::ExecuteEvent when the action "Create NewPad"
has been selected in the Canvas pulldown menu.

Click left button to indicate one corner of the pad
Click left button to indicate the opposite corner

The new pad is inserted in the pad where the first point is selected.
```

void **CreateNewPave**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)

```
Create a new pavetext in this pad
Function called by TPad::ExecuteEvent when the action "Create NewPaveText"
has been selected in the Canvas pulldown menu.

Click left button to indicate one corner of the pavelabel.
Release left button at the opposite corner.
```

void **CreateNewPolyLine**(Int_t *event,* Int_t *px,* Int_t *py,* Int_t *mode*)

```
Create a new PolyLine
Click left button to indicate a new point
Click right button to close the polyline
mode = 0 normal polyline
mode = 0 smooth polyline
This function is called when the item NewPolyLine is selected in the
canvas pulldown menu.
```

void **CreateNewText**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Create a new TText at the cursor position
Click left button to indicate the text position
This function is called when the item NewText is selected in the
canvas pulldown menu.
```

void **Divide**(Int_t *nx,* Int_t *ny,* Float_t *xmargin,* Float_t *ymargin,* Int_t *color*)

```
Automatic pad generation by division
The current canvas is divided in nx by ny equal divisions(pads).
xmargin is the space along x between pads in percent of canvas.
ymargin is the space along y between pads in percent of canvas.
color is the color of the new pads. If 0, color is the canvas color.
Pads are automatically named canvasname_n where n is the division number
starting from top left pad.
    Example if canvasname=c1 , nx=2, ny=3

 ...............................................................
 .                                .                             .
 .                                .                             .
 .                                .                             .
 .           c1_1                 .           c1_2              .
 .                                .                             .
 .                                .                             .
 .                                .                             .
 ...............................................................
 .                                .                             .
 .                                .                             .
 .                                .                             .
 .           c1_3                 .           c1_4              .
 .                                .                             .
 .                                .                             .
 .                                .                             .
 ...............................................................
 .                                .                             .
 .                                .                             .
 .                                .                             .
 .           c1_5                 .           c1_6              .
 .                                .                             .
```

```
.                               .                              .
..............................................................

Once a pad is divided into subpads, one can set the current pad
to a subpad with a given division number as illustrated above
with TPad::cd(subpad_number).
For example, to set the current pad to c1_4, one can do:
c1->cd(4)

Note<that c1->cd(0) is equivalent to c1->cd().
This statement sets the current pad to c1 itself.
```

void **Draw**(Option_t **option*)
```
Draw Pad in Current pad (re-parent pad if necessary)
```

void **DrawFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Text_t **title*)
```
Draw a pad frame
Compute real pad range taking into account all margins
Use services of TH1F class
```

void **DrawLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
```
Draw line in CurrentPad World coordinates
```

void **DrawLineNDC**(Float_t *u1,* Float_t *v1,* Float_t *u2,* Float_t *v2*)
```
Draw line in CurrentPad NDC coordinates
```

void **DrawText**(Float_t *x,* Float_t *y,* Text_t **text*)
```
Draw text in CurrentPad World coordinates
```

void **DrawTextNDC**(Float_t *u,* Float_t *v,* Text_t **text*)
```
Draw text in CurrentPad NDC coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the canvas
```

void **GetRange**(Float_t &*x1,* Float_t &*y1,* Float_t &*x2,* Float_t &*y2*)
```
Return pad world coordinates range
```

void **GetRangeAxis**(Axis_t &*xmin,* Axis_t &*ymin,* Axis_t &*xmax,* Axis_t &*ymax*)
```
Return pad axis coordinates range
```

void **ls**(Option_t **option*)
```
List all primitives in pad
```

void **Paint**(Option_t **option*)
```
Paint all primitives in pad
```

void **PaintPadFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
```
Paint histogram/graph frame
```

void **PaintModified**( )
```
Traverse pad hierarchy and (re)paint only modified pads
```

void **PaintBox**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Option_t **option*)
```
Paint box in CurrentPad World coordinates
```

void **PaintFillArea**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)
```
Paint fill area in CurrentPad World coordinates
```

void **PaintLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
```
Paint line in CurrentPad World coordinates
```

void **PaintLineNDC**(Coord_t *u1,* Coord_t *v1,* Coord_t *u2,* Coord_t *v2*)
```
Draw a line with coordinates in NDC
```

void **PaintLine3D**(Float_t **p1,* Float_t **p2*)
```
Paint 3-D line in the CurrentPad
```

void **PaintPolyLine**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)
```
Paint polyline in CurrentPad World coordinates
```

void **PaintPolyLineNDC**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)
```
Paint polyline in CurrentPad NDC coordinates
```

void **PaintPolyLine3D**(Int_t *n,* Float_t **p*)
```
Paint 3-D polyline in the CurrentPad
```

void **PaintPolyMarker**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)
```
Paint polymarker in CurrentPad World coordinates
```

void **PaintText**(Float_t *x,* Float_t *y,* Text_t **text*)
```
Paint text in CurrentPad World coordinates
```

void **PaintTextNDC**(Float_t *u,* Float_t *v,* Text_t **text*)
```
Paint text in CurrentPad NDC coordinates
```

TPad* **Pick**(Int_t *px,* Int_t *py,* TObjLink **&pickobj*)
```
Search for an object at pixel position px,py
Check if point is in this pad.
If yes, check if it is in one of the subpads
If found in the pad, compute closest distance of approach
to each primitive.
If one distance of approach is found to be within the limit Distancemaximum
the corresponding primitive is selected and the routine returns.
```

void **Pop**( )
```
Pop pad to the top of the stack
```

void **Print**(Text_t **filename*)
```
Old interface. Use SaveAs instead
```

void **Print**(Text_t **filename,* Option_t **option*)
```
Save Pad contents on a file in various formats
if option  =  0   - as "ps"
            "ps"  - Postscript file is produced
            "eps" - an Encapsulated Postscript file is produced
            "gif" - a GIF file is produced
            "cxx" - a C++ macro file is produced

  filename = 0 - filename  is defined by the GetName and its
                 extension is defined with the option
```

void **Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
```
Set world coordinate system for the pad
```

void **RangeAxis**(Axis_t *xmin,* Axis_t *ymin,* Axis_t *xmax,* Axis_t *ymax*)
```
Set axis coordinate system for the pad
The axis coordinate system is a subset of the world coordinate system
xmin,ymin are the origin of the current coordinate system
xmax is the end of the X axis
ymax is the end of the Y axis
By default a margin of 10 per cent is left on all sides of the pad
```

void **RecursiveRemove**(TObject **obj*)
```
Recursively remove object from a pad and its subpads
```

void **ResizePad**(Option_t **option*)

```
Compute pad conversion coefficients
Conversion from x to px & y to py
    x - xmin     px - pxlow             xrange  = xmax-xmin
    --------  =  ----------      with
     xrange        pxrange              pxrange = pxmax-pxmin

            pxrange(x-xmin)
==>  px =   ---------------  + pxlow   = fXtoPixelk + fXtoPixel * x
                 xrange

==>  fXtoPixelk = pxlow - pxrange*xmin/xrange
     fXtoPixel  = pxrange/xrange
        where  pxlow   = fAbsXlowNDC*fCw
               pxrange = fAbsWNDC*fCw

    y - ymin     py - pylow             yrange  = ymax-ymin
    --------  =  ----------      with
     yrange        pyrange              pyrange = pymax-pymin

            pyrange(y-ymin)
==>  py =   ---------------  + pylow   = fYtoPixelk + fYtoPixel * y
                 yrange

==>  fYtoPixelk = pylow - pyrange*ymin/yrange
     fYtoPixel  = pyrange/yrange
        where  pylow   = (1-fAbsYlowNDC)*fCh
               pyrange = -fAbsHNDC*fCh

-  Conversion from px to x & py to y
          xrange(px-pxlow)
==>  x =  ----------------  + xmin  = fPixeltoXk + fPixeltoX * px
              pxrange
-
==>  fPixeltoXk = xmin - pxlow*xrange/pxrange
     fPixeltoX  = xrange/pxrange

          yrange(py-pylow)
==>  y =  ----------------  + ymin  = fPixeltoYk + fPixeltoY * py
              pyrange
-
==>  fPixeltoYk = ymin - pylow*yrange/pyrange
     fPixeltoY  = yrange/pyrange

-----------------------------------------------------------------------

Computation of the coefficients in case of LOG scales
A, Conversion from pixel coordinates to world coordinates

    Log(x) - Log(xmin)      Log(x/xmin)       px - pxlow
u = --------------------- =  -------------  =  -----------
   Log(xmax) - Log(xmin)    Log(xmax/xmin)     pxrange

==> Log(x/xmin) = u*Log(xmax/xmin)
   x = xmin*exp(u*Log(xmax/xmin)
Let alfa = Log(xmax/xmin)/fAbsWNDC

   x = xmin*exp(-alfa*pxlow) + exp(alfa*px)
   x = fPixeltoXk*exp(fPixeltoX*px)
==> fPixeltoXk = xmin*exp(-alfa*pxlow)
   fPixeltoX  = alfa

    Log(y) - Log(ymin)      Log(y/ymin)       pylow - py
```

```
v = --------------------- =  -------------  =  -----------
   Log(ymax) - Log(ymin)    Log(ymax/ymin)     pyrange

Let beta = Log(ymax/ymin)/pyrange
   Log(y/ymin) = beta*pylow - beta*py
   y/ymin = exp(beta*pylow - beta*py)
   y = ymin*exp(beta*pylow)*exp(-beta*py)
==> y = fPixeltoYk*exp(fPixeltoY*py)
   fPixeltoYk = ymin*exp(beta*pylow)
   fPixeltoY  = -beta

-  B, Conversion from World coordinates to pixel coordinates

px = pxlow + u*pxrange
  = pxlow + Log(x/xmin)/alfa
  = pxlow -Log(xmin)/alfa  + Log(x)/alfa
  = fXtoPixelk + fXtoPixel*Log(x)
==> fXtoPixelk = pxlow -Log(xmin)/alfa
==> fXtoPixel  = 1/alfa

py = pylow - Log(y/ymin)/beta
  = fYtoPixelk + fYtoPixel*Log(y)
==> fYtoPixelk = pylow - Log(ymin)/beta
   fYtoPixel  = 1/beta
```

void **SaveAs**(Text_t **filename*)

```
Save Pad contents on a  file in various formats
if filename is "", the file produced is padname.ps
if filename starts with a dot, the padname is added in front
if filename contains .eps, an Encapsulated Postscript file is produced
if filename contains .gif, a GIF file is produced
if filename contains .C or .cxx, a C++ macro file is produced
if filename contains .root, a Root file is produced
```

void **SavePrimitive**(ofstream *&out,* Option_t **option*)

```
Save primitives in this pad on the C++ source file out
```

void **SetLogx**(Int_t *value*)

```
Set Lin/Log scale for X
```

void **SetLogy**(Int_t *value*)

```
Set Lin/Log scale for Y
```

void **SetLogz**(Int_t *value*)

```
Set Lin/Log scale for Z
```

void **SetPad**(Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup*)

```
Set canvas range for pad
```

void **SetPad**(const Text_t **name,* const Text_t **title,*Float_t xlow, Float_t ylow, Float_t xup, Float_t yup,Color_t color, Short_t bordersize, Short_t *bordermode*)

```
Set all pad parameters
```

void **Streamer**(TBuffer *&b*)

```
Stream a class object
```

void **UseCurrentStyle**( )

```
Force a copy of current style for all objects in pad
```

void **x3d**(Option_t **option*)

```
Invokes the x3d package to view the content of a pad
```

Author: René Brun 12/12/94

Last update: 1.03/06 17/10/97 13.44.15 by René Brun

TPARA

class **TPARA** : public TBRIK

public:

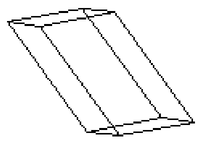
class TPARA	**TPARA**( )
class TPARA	**TPARA**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dx,* Float_t *dy,* Float_t *dz,* Float_t *alpha,* Float_t *theta,* Float_t *phi*)
class TPARA	**TPARA**(class TPARA&)
void	**~TPARA**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fAlpha**	*angle w.r.t. the y axis from the centre of the low y to the high y*
Float_t	**fTheta**	*polar angle from the centre of the low z to the high z*
Float_t	**fPhi**	*polar angle from the centre of the low z to the high z*

Class Description



```
PARA is a parallelepiped. It has 9 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dx         half-length in x
  - dy         half-length in y
  - dz         half-length in z
  - alpha      angle formed by the y axis and by the plane joining the
               centre of the faces parallel to the z-x plane at -DY and +DY
  - theta      polar angle of the line joining the centres of the faces
               at -DZ and +DZ in z
  - phi        azimuthal angle of the line joining the centres of the
               faces at -DZ and +DZ in z
```

TPARA( )

```
PARA shape default constructor
```

TPARA(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dx,* Float_t *dy,* Float_t *dz,*Float_t *alpha,* Float_t *theta,* Float_t *phi*) : **TBRIK**(*name, title, material, dx, dy,* dz)

```
PARA shape normal constructor
```

~TPARA( )

```
PARA shape default destructor
```

void **SetPoints**(Float_t **buff*)

```
Create PARA points
```

Author: Nenad Buncic 19/09/95
Last update: 1.00/07 27/03/97 17.55.23 by René Brun

TPave

class **TPave** : public TBox

public:

class TPave	**TPave**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t *bordersize = 4,* Option_t* *option = br*)
class TPave	**TPave**( )
class TPave	**TPave**(class TPave& *pave*)
void	**~TPave**( )
class TClass*	**Class**( )
virtual void	**ConvertNDCtoPad**( )
void	**Copy**(class TObject& *pave*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPave**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t *bordersize = 4,* Option_t* *option = br*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Int_t	**GetBorderSize**( )
Float_t	**GetCornerRadius**( )
const Option_t*	**GetName**( )
const Option_t*	**GetOption**( )
Float_t	**GetX1NDC**( )
Float_t	**GetX2NDC**( )
Float_t	**GetY1NDC**( )
Float_t	**GetY2NDC**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPave**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t *bordersize = 4,* Option_t* *option = br*)
virtual void	**PaintPaveArc**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t *bordersize = 4,* Option_t* *option = br*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetBorderSize**(Int_t *bordersize = 4*)
virtual void	**SetCornerRadius**(Float_t *rad* = 0.2)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetOption**(Option_t* *option = br*)
virtual void	**SetX1NDC**(Coord_t *x1*)
virtual void	**SetX2NDC**(Coord_t *x2*)
virtual void	**SetY1NDC**(Coord_t *y1*)
virtual void	**SetY2NDC**(Coord_t *y2*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fX1NDC**	*X1 point in NDC coordinates*
Float_t	**fY1NDC**	*Y1 point in NDC coordinates*
Float_t	**fX2NDC**	*X2 point in NDC coordinates*
Float_t	**fY2NDC**	*Y2 point in NDC coordinates*
Int_t	**fBorderSize**	*window box bordersize in pixels*
Int_t	**fInit**	(*=0 if transformation to NDC not yet done*)
Float_t	**fCornerRadius**	*Corner radius in case of option arc*
class TString	**fOption**	*Pave style*
class TString	**fName**	*Pave name*

See Also: TPaveLabel, TPaveText

Class Description

```
a PAVE is a TBox with a bordersize and a shadow option
The corners of a TPave can be rounded (option "arc")
More functional objects like TPavelabel, TPaveText derive from TPave.
```

TPave( ): **TBox**( )

```
pave default constructor
```

TPave(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t bordersize, Option_t **option*):**TBox**(*x1, y1, x2, y2*)

```
pave normal constructor
a PAVE is a box with a bordersize and a shadow option
the bordersize is in pixels
option = "T" Top frame
option = "B" Bottom frame
option = "R" Right frame
option = "L" Left frame
option = "NDC" x1,y1,x2,y2 are given in NDC
option = "ARC" corners are rounded
```

~TPave( )

```
pave default destructor
```

void **ConvertNDCtoPad**( )

```
Convert pave coordinates from NDC to Pad coordinates
```

void **Copy**(TObject &*obj*)

```
Copy this pave to pave
```

void **Draw**(Option_t **option*)

```
Draw this pave with its current attributes
```

void **DrawPave**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t bordersize, Option_t **option*)

```
Draw this pave with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a PAVE object is clicked.
```

void **Paint**(Option_t **option*)

```
Paint this pave with its current attributes
option = "T" Top frame
option = "B" Bottom frame
option = "R" Right frame
option = "L" Left frame
option = "NDC" x1,y1,x2,y2 are given in NDC
option = "ARC" corners are rounded
   In case of option "ARC", the corner radius is specified
   via TPave::SetCornerRadius(rad) where rad is given in percent
   of the pave height (default value is 0.2).
```

void **PaintPave**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t bordersize, Option_t **option*)

```
Draw this pave with new coordinates
```

void **PaintPaveArc**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Int_t bordersize, Option_t **option*)

```
Draw this pave with rounded corners
```

void **Print**(Option_t **option*)

```
Dump this pave with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 16/10/95
Last update: 1.02/00 09/07/97 22.27.48 by René Brun

TPaveLabel

class **TPaveLabel** : public TPave, public TAttText

public:

class TPaveLabel	**TPaveLabel**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t* *label,* Option_t* *option = br*)
class TPaveLabel	**TPaveLabel**( )
class TPaveLabel	**TPaveLabel**(class TPaveLabel& *pavelabel*)
void	**~TPaveLabel**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *pavelabel*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPaveLabel**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t* *label,* Option_t* *option*)
const Text_t*	**GetLabel**( )
const Text_t*	**GetTitle**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPaveLabel**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t* *label,* Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetLabel**(Text_t* *label*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

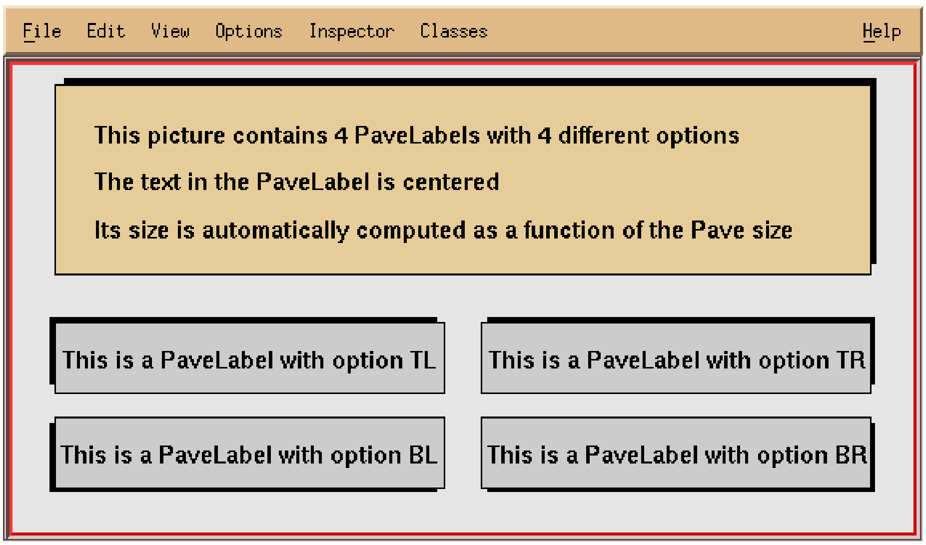
Data Members

protected:

class TString	**fLabel**	*Label written at the center of Pave*

Class Description

```
*   A PaveLabel is a Pave (see TPave) with a text centered in the Pave.
```



TPaveLabel( ): **TPave**( ), **TAttText**( )

```
pavelabel default constructor
```

TPaveLabel(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t **label,* Option_t **option*):**TPave**(*x1, y1, x2, y2, 3, option*), **TAttText**(*22, 0, 1, 62,* 0.7)

```
pavelabel normal constructor
a PaveLabel is a Pave with a label centered in the Pave
The Pave is by default defined bith bordersize=5 and option ="br".
The text size is automatically computed as a function of the pave size.
```

~TPaveLabel( )

```
pavelabel default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this pavelabel to pavelabel
```

void **Draw**(Option_t **option*)

```
Draw this pavelabel with its current attributes
```

void **DrawPaveLabel**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t **label,* Option_t **option*)

```
Draw this pavelabel with new coordinates
```

void **Paint**(Option_t **option*)

```
Paint this pavelabel with its current attributes
```

void **PaintPaveLabel**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Text_t *label, Option_t **option*)

```
Draw this pavelabel with new coordinates
```

void **Print**(Option_t **option*)

```
Dump this pavelabel with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 17/10/95
Last update: 1.02/00 09/07/97 22.28.46 by René Brun

TPavesText

class **TPavesText** : public TPaveText

public:

class TPavesText	**TPavesText**(Coord_t *x1*, Coord_t *y1*, Coord_t *x2*, Coord_t *y2*, Int_t *npaves* = *5*, Option_t* *option* = *br*)
class TPavesText	**TPavesText**( )
class TPavesText	**TPavesText**(class TPavesText& *pavestext*)
void	**~TPavesText**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *pavestext*)
virtual void	**Draw**(Option_t* *option*)
virtual Int_t	**GetNpaves**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out*, Option_t* *option*)
virtual void	**SetNpaves**(Int_t *npaves* = *5*)
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

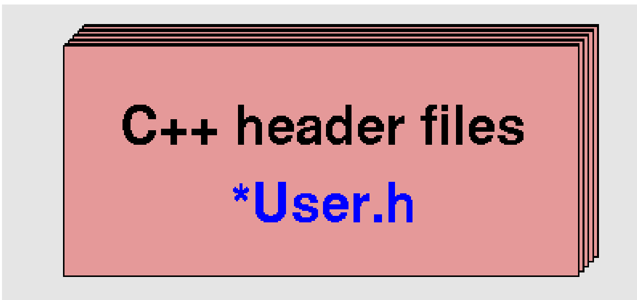
Data Members

protected:

Int_t	**fNpaves**	*Number of stacked paves*

Class Description

```
A PavesText is a PaveText (see TPaveText) with several stacked paves.
```



TPavesText( ): **TPaveText**( )

```
pavestext default constructor
```

TPavesText(Coord_t *x1*, Coord_t *y1*, Coord_t *x2*, Coord_t *y2*, Int_t *npaves*, Option_t **option*):**TPaveText**(*x1*, *y1*, *x2*, *y2*, *option*)

```
pavestext normal constructor
The PavesText is by default defined bith bordersize=1 and option ="br".
```

~TPavesText( )

```
pavestext default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this pavestext to pavestext
```

void **Draw**(Option_t **option*)

```
Draw this pavestext with its current attributes
```

void **Paint**(Option_t **option*)

```
Paint this pavestext with its current attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 19/11/95
Last update: 1.02/00 09/07/97 22.29.24 by René Brun

TPaveText

class **TPaveText** : public TPave, public TAttText

public:

class TPaveText	**TPaveText**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Option_t* *option = br*)
class TPaveText	**TPaveText**( )
class TPaveText	**TPaveText**(class TPaveText& *pavetext*)
void	**~TPaveText**( )
virtual class TBox*	**AddBox**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual class TLine*	**AddLine**(Float_t *x1 = 0,* Float_t *y1 = 0,* Float_t *x2 = 0,* Float_t *y2 = 0*)
virtual class TText*	**AddText**(Float_t *x1,* Float_t *y1,* Text_t* *label*)
virtual class TText*	**AddText**(Text_t* *label*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
void	**Copy**(class TObject& *pavetext*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawFile**(Text_t* *filename,* Option_t* *option*)
virtual class TText*	**GetLine**(Int_t *number*)
virtual class TText*	**GetLineWith**(Text_t* *text*)
virtual class TList*	**GetListOfLines**( )
Float_t	**GetMargin**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPrimitives**(Int_t *mode*)
virtual void	**Print**(Option_t* *option*)
virtual void	**ReadFile**(Text_t* *filename,* Option_t* *option,* Int_t *nlines = 50,* Int_t *fromline = 0*)
virtual void	**SaveLines**(class ofstream& *out,* char* *name*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetAllWith**(Text_t* *text,* Option_t* *option,* Float_t *value*)
virtual void	**SetMargin**(Float_t *margin* = 0.*05*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

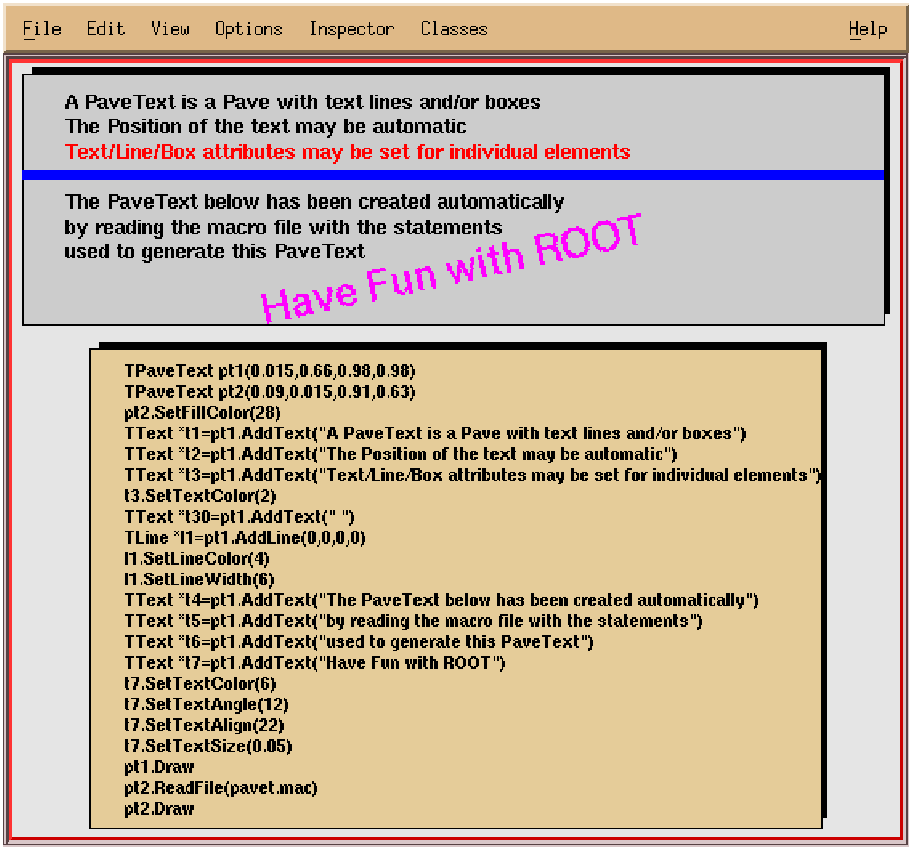
protected:

Int_t	**fLongest**	*Length of the longest line*
Float_t	**fMargin**	*Text margin*
class TList*	**fLines**	*List of labels*

See Also: TDiamond, TPavesText

Class Description

```
A PaveText is a Pave (see TPave) with text,lines or/and boxes.
```



TPaveText( ): **TPave**( ), **TAttText**( )

```
pavetext default constructor
```

TPaveText(Coord_t *x1*, Coord_t *y1*, Coord_t *x2*, Coord_t *y2*, Option_t **option*):**TPave**(*x1, y1, x2, y2, 4, option*), **TAttText**(*22, 0, 1, 62,* 0)

```
pavetext normal constructor
a PaveText is a Pave with several lines of text
The Pave is by default defined bith bordersize=5 and option ="br".
The individual text items are entered via AddText
By default, text items inherits from the default pavetext AttText.
```

~**TPaveText**( )

```
pavetext default destructor
```

TBox* **AddBox**(Float_t *x1*, Float_t *y1*, Float_t *x2*, Float_t *y2*)

```
Add a new graphics box to this pavetext
```

TLine* **AddLine**(Float_t *x1*, Float_t *y1*, Float_t *x2*, Float_t *y2*)

```
Add a new graphics line to this pavetext
```

TText* **AddText**(Float_t *x1*, Float_t *y1*, Text_t **text*)

```
Add a new Text line to this pavetext at given coordinates
```

TText* **AddText**(Text_t **text*)

```
Add a new Text line to this pavetext
```

void **Clear**(Option_t **option*)

```
Clear all lines in this pavetext
```

void **Copy**(TObject &*obj*)

```
Copy this pavetext to pavetext
```

void **Draw**(Option_t **option*)
```
Draw this pavetext with its current attributes
```

void **DrawFile**(Text_t **filename,* Option_t **option*)
```
Draw lines in filename in this pavetext
```

TText* **GetLine**(Int_t *number*)
```
Get Pointer to line number in this pavetext
```

TText* **GetLineWith**(Text_t **text*)
```
Get Pointer to first containing string text in this pavetext
```

void **Paint**(Option_t **option*)
```
Paint this pavetext with its current attributes
```

void **PaintPrimitives**(Int_t *mode*)
```
Paint list of primitives in this pavetext
```

void **Print**(Option_t **option*)
```
Dump this pavetext with its attributes
```

void **ReadFile**(Text_t **filename,* Option_t **option,* Int_t *nlines,* Int_t *fromline*)
```
Read lines of filename in this pavetext
Read from line number fromline a total of nlines

Note that this function changes the default text alignment to left/center
```

void **SaveLines**(ofstream &*out,* char **name*)
```
Save lines of this pavetext as C++ statements on output stream out
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

void **SetAllWith**(Text_t **text,* Option_t **option,* Float_t *value*)
```
Set attribute option for all lines containing string text
Possible options are all the AttText attributes
    Align, Color, Font, Size and Angle
```

Author: René Brun 20/10/95
Last update: 1.02/00 09/07/97 22.31.01 by René Brun

TPCON

class **TPCON** : public TShape

protected:

virtual void	**MakeTableOfCoSin**( )
virtual void	**PaintGLPoints**(Float_t* *vertex*)

public:

class TPCON	**TPCON**( )
class TPCON	**TPCON**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *phi1,* Float_t *dphi1,* Int_t *nz*)
class TPCON	**TPCON**(class TPCON&)
void	**~TPCON**( )
class TClass*	**Class**( )
virtual void	**DefineSection**(Int_t *secNum,* Float_t *z,* Float_t *rmin,* Float_t *rmax*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual Int_t	**GetNumberOfDivisions**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SetNumberOfDivisions**(Int_t *p*)
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

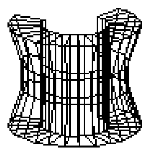
Double_t*	**fSiTab**	*Table of sin(fPhi1*) .... sin(fPhil+fDphi1)
Double_t*	**fCoTab**	*Table of cos(fPhi1*) .... cos(fPhil+fDphi1)

protected:

Float_t	**fPhi1**	*lower phi limit*
Float_t	**fDphi1**	*range in phi*
Int_t	**fNz**	*number of z segments*
Float_t*	**fRmin**	*pointer to array of inside radiuses*
Float_t*	**fRmax**	*pointer to array of outside radiuses*
Float_t*	**fDz**	*pointer to array of half lengths in z*
Int_t	**fNdiv**	*number of divisions*

See Also: TPGON

Class Description



```
PCON is a polycone. It has the following parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - phi1       the azimuthal angle phi at which the volume begins (angles
               are counted counterclockwise)
  - dphi       opening angle of the volume, which extends from
               phi1 to phi1+dphi
```

```
  - nz         number of planes perpendicular to the z axis where
               the dimension of the section is given -- this number
               should be at least 2
  - rmin       array of dimension nz with minimum radius at a given plane
  - rmax       array of dimension nz with maximum radius at a given plane
  - z          array of dimension nz with z position of given plane
```

TPCON( )

```
PCON shape default constructor
```

TPCON(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *phi1,* Float_t *dphi1,* Int_t *nz*): **TShape**(*name, title, material*)

```
PCON shape normal constructor
Parameters of the nz positions must be entered via TPCON::DefineSection.
```

~TPCON( )

```
PCON shape default destructor
```

void **DefineSection**(Int_t *secNum,* Float_t *z,* Float_t *rmin,* Float_t *rmax*)

```
Defines section secNum of the polycone
  - rmin  radius of the inner circle in the cross-section

  - rmax  radius of the outer circle in the cross-section

  - z     z coordinate of the section
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a PCON
Compute the closest distance of approach from point px,py to each
computed outline point of the PCON.
```

void **Paint**(Option_t **option*)

```
Paint this 3-D shape with its current attributes
```

void **PaintGLPoints**(Float_t **vertex*)

```
Paint BRIK via OpenGL
```

void **SetPoints**(Float_t **buff*)

```
Create PCON points
```

void **Sizeof3D**( ) const

```
Return total X3D size of this shape with its attributes
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: Nenad Buncic 29/09/95
Last update: 1.01/07 26/06/97 08.59.56 by René Brun

TPeriodic

class **TPeriodic** : public TCallbackObject

private:

void	**TickTimeOut**(*XtPointer,* XtIntervalId*)

protected:

class TPeriodic	**TPeriodic**( )
virtual void	**Tick**( )

public:

class TPeriodic	**TPeriodic**(class TPeriodic&)
class TPeriodic	**TPeriodic**(int *interval*)
void	**~TPeriodic**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Start**( )
void	**Stop**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

int	**fInterval**
XtIntervalId	**fId**

public:

static const char *const	**fgTimerCallback**

Class Description

```
TPeriodic

This class provides a class wrapper for X toolkit time outs.
```

void **TickTimeOut**(XtPointer *clientData,* XtIntervalId *)

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TPGON

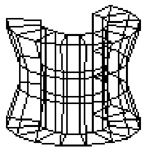
class **TPGON** : public TPCON

public:

class TPGON	**TPGON**( )
class TPGON	**TPGON**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *phi1,* Float_t *dphi1,* Int_t *npdv,* Int_t *nz*)
class TPGON	**TPGON**(class TPGON&)
void	**~TPGON**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description



```
PGON is a polygon. It has the following parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - phi1       the azimuthal angle &phi at which the volume begins (angles
               are counted counterclockwise)
  - dphi1      opening angle of the volume, which extends from phi1 to
               phi1+dphi
  - npdv       number of sides of the cross section between the given
               phi limits
  - nz         number of planes perpendicular to the z axis where
               the dimension of the section is given -- this number
               should be at least 2
  - rmin       array of dimension nz with minimum radius at a given plane
  - rmax       array of dimension nz with maximum radius at a given plane
  - z          array of dimension nz with z position of given plane
```

TPGON( )

```
PGON shape default constructor
```

TPGON(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *phi1,*Float_t dphi1, Int_t npdv, Int_t *nz*):
TPCON(*name, title, material, phi1, dphi1, nz*)

```
PGON shape normal constructor
Parameters of the nz positions must be entered via TPCON::DefineSection.
```

~TPGON( )

```
PGON shape default destructor
```

Author: Nenad Buncic 29/09/95
Last update: 1.00/06 23/03/97 11.26.51 by René Brun

TPoints

class **TPoints**

public:

class TPoints	**TPoints**(Coord_t *xy*)
class TPoints	**TPoints**( )
class TPoints	**TPoints**(Coord_t *x,* Coord_t *y*)
class TPoints	**TPoints**(class TPoints&)
void	**~TPoints**( )
class TClass*	**Class**( )
Coord_t	**GetX**( )
Coord_t	**GetY**( )
virtual class TClass*	**IsA**( )
void	**SetX**(Coord_t *x*)
void	**SetY**(Coord_t *y*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Coord_t	**fX**	*X world coordinate*
Coord_t	**fY**	*Y world coordinate*

Class Description

Author: René Brun 23/02/95
Last update: 1.00/06 23/03/97 19.28.42 by René Brun

TPolyLine

class **TPolyLine** : public TObject, public TAttLine, public TAttFill

public:

class TPolyLine	**TPolyLine**(Int_t *n,* Option_t* *option*)
class TPolyLine	**TPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
class TPolyLine	**TPolyLine**( )
class TPolyLine	**TPolyLine**(class TPolyLine& *polyline*)
void	**~TPolyLine**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *polyline*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Int_t	**GetN**( )
const Option_t*	**GetOption**( )
virtual void	**GetPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
Float_t*	**GetX**( )
Float_t*	**GetY**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintPolyLineNDC**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetOption**(Option_t* *option*)
virtual void	**SetPoint**(Int_t *point,* Float_t *x,* Float_t *y*)
virtual void	**SetPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fN**	*Number of points*
Float_t*	**fX**	*Array of X coordinates*
Float_t*	**fY**	*Array of Y coordinates*
class TString	**fOption**	*options*

Class Description

```
a PolyLine is defined by an array on N points in a 2-D space.
```

TPolyLine( ): **TObject**( )

```
PolyLine default constructor
```

TPolyLine(Int_t *n,* Option_t **option*):**TObject**( ), **TAttLine**( ), **TAttFill**( )

```
PolyLine normal constructor without initialisation
```

TPolyLine(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*):**TObject**( ), **TAttLine**( ), **TAttFill**( )

```
PolyLine normal constructor
```

~TPolyLine( )

```
PolyLine default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this polyline to polyline
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a polyline
Compute the closest distance of approach from point px,py to each segment
of the polyline.
Returns when the distance found is below DistanceMaximum.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw this polyline with its current attributes
```

void **DrawPolyLine**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)

```
Draw this polyline with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the window
```

void **GetPolyLine**(Int_t *n,* Float_t **x,* Float_t **y,* const Option_t **option*)

```
Return current polyline values
```

void **ls**(Option_t **option*)

```
List this polyline with its attributes
```

void **Paint**(Option_t **option*)

```
Paint this polyline with its current attributes
```

void **PaintPolyLine**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)

```
Draw this polyline with new coordinates
```

void **PaintPolyLineNDC**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)

```
Draw this polyline with new coordinates in NDC
```

void **Print**(Option_t **option*)

```
Dump this polyline with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetPoint**(Int_t *point,* Float_t *x,* Float_t *y*)

```
Initialize one point of the polyline
```

void **SetPolyLine**(Int_t *n,* Float_t **x,* Float_t **y,* Option_t **option*)

```
Set new values for this polyline
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 12/12/94
Last update: 1.03/01 15/08/97 10.41.36 by René Brun

TPolyLine3D

class **TPolyLine3D** : public TObject, public TAttLine

public:

class TPolyLine3D	**TPolyLine3D**(Int_t *n,* Option_t* *option*)
class TPolyLine3D	**TPolyLine3D**(Int_t *n,* Float_t* *p,* Option_t* *option*)
class TPolyLine3D	**TPolyLine3D**( )
class TPolyLine3D	**TPolyLine3D**(Int_t *n,* Float_t* *x,* Float_t* *y,* Float_t* *z,* Option_t* *option*)
class TPolyLine3D	**TPolyLine3D**(class TPolyLine3D& *polylin*)
void	**~TPolyLine3D**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *polyline*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPolyLine**(Int_t *n,* Float_t* *p,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Int_t	**GetN**( )
const Option_t*	**GetOption**( )
Float_t*	**GetP**( )
virtual void	**GetPolyLine**(Int_t *n,* Float_t* *p,* Option_t* *option*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPolyLine**(Int_t *n,* Float_t* *p,* Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetOption**(Option_t* *option*)
virtual void	**SetPoint**(Int_t *point,* Float_t *x,* Float_t *y,* Float_t *z*)
virtual void	**SetPolyLine**(Int_t *n,* Float_t* *p,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fN**	*Number of points*
Float_t*	**fP**	*Array of 3-D coordinates (x, y, z)*
class TString	**fOption**	*options*
UInt_t	**fGLList**	*The list number for OpenGL view*

Class Description

```
PolyLine3D is a 3-dimensional polyline. It has 4 different constructors.

First one, without any parameters TPolyLine3D(), we call 'default
constructor' and it's used in a case that just an initialisation is
needed (i.e. pointer declaration).

    Example:
            TPolyLine3D *pl1 = new TPolyLine3D;

Second one is 'normal constructor' with, usually, one parameter
n (number of points), and it just allocates a space for the points.

    Example:
            TPolyLine3D pl1(150);
```

```
Third one allocates a space for the points, and also makes
initialisation from the given array.

    Example:
              TPolyLine3D pl1(150, pointerToAnArray);

Fourth one is, almost, similar to the constructor above, except
initialisation is provided with three independent arrays (array of
x coordinates, y coordinates and z coordinates).

    Example:
              TPolyLine3D pl1(150, xArray, yArray, zArray);
```

TPolyLine3D( )

```
3-D PolyLine default constructor
```

TPolyLine3D(Int_t *n,* Option_t **option*)

```
3-D PolyLine normal constructor without initialisation
If n < 0 the default size (2 points) is set
```

TPolyLine3D(Int_t *n,* Float_t **p,* Option_t **option*)

```
3-D PolyLine normal constructor
If n < 0 the default size (2 points) is set
```

TPolyLine3D(Int_t *n,* Float_t **x,* Float_t **y,* Float_t **z,* Option_t **option*)

```
3-D PolyLine normal constructor
If n < 0 the default size (2 points) is set
```

~TPolyLine3D( )

```
3-D PolyLine default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this polyline to polyline
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a 3-D polyline
Compute the closest distance of approach from point px,py to each segment
of the polyline.
Returns when the distance found is below DistanceMaximum.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)

```
Draw this 3-D polyline with its current attributes
```

void **DrawPolyLine**(Int_t *n,* Float_t **p,* Option_t **option*)

```
Draw this 3-D polyline with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
```

void **GetPolyLine**(Int_t *n,* Float_t **p,* const Option_t **option*)

```
Return current 3-D polyline values
```

void **ls**(Option_t **option*)

```
List this 3-D polyline with its attributes
```

void **Paint**(Option_t **option*)

```
Paint this 3-D polyline with its current attributes
Option could be 'x3d', and it means that output
will be performed by X3D package.
```

void **PaintPolyLine**(Int_t *n,* Float_t **p,* Option_t **option*)

```
Draw this 3-D polyline with new coordinates
```

void **Print**(Option_t **option*)

```
Dump this 3-D polyline with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **SetPoint**(Int_t *n,* Float_t *x,* Float_t *y,* Float_t *z*)

```
Initialize one point of the 3-D polyline
if n is more then the current TPolyLine3D size (n > fN) - re-allocate this
```

void **SetPolyLine**(Int_t *n,* Float_t **p,* Option_t **option*)

```
Set new values for this 3-D polyline
```

void **Sizeof3D**( ) const

```
Return total X3D size of this shape with its attributes
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: Nenad Buncic 17/08/95
Last update: 1.01/07 26/06/97 10.09.08 by Valery Fine(fine@vxcern.cern.ch)

TPolyMarker

class **TPolyMarker** : public TObject, public TAttMarker

public:

class TPolyMarker	**TPolyMarker**(Int_t *n,* Option_t* *option*)
class TPolyMarker	**TPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
class TPolyMarker	**TPolyMarker**( )
class TPolyMarker	**TPolyMarker**(class TPolyMarker& *polymarker*)
void	**~TPolyMarker**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *polymarker*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Int_t	**GetN**( )
const Option_t*	**GetOption**( )
virtual void	**GetPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
Float_t*	**GetX**( )
Float_t*	**GetY**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetPoint**(Int_t *point,* Float_t *x,* Float_t *y*)
virtual void	**SetPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fN**	*number of points*
Float_t*	**fX**	*Array of X coordinates*
Float_t*	**fY**	*Array of Y coordinates*
class TString	**fOption**	*options*

Class Description

```
a PolyMarker is defined by an array on N points in a 2-D space.
At each point x[i], y[i] a marker is drawn.
Marker attributes are managed by TAttMarker.
See TMarker for the list of possible marker types.
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the window
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: René Brun 12/12/94
Last update: 1.03/01 15/08/97 10.41.36 by René Brun

TPolyMarker3D

class **TPolyMarker3D** : public TObject, public TAttMarker

public:

class TPolyMarker3D	**TPolyMarker3D**(Int_t *n,* Marker_t *marker = 1,* Option_t* *option*)
class TPolyMarker3D	**TPolyMarker3D**(Int_t *n,* Float_t* *p,* Marker_t *marker = 1,* Option_t* *option*)
class TPolyMarker3D	**TPolyMarker3D**( )
class TPolyMarker3D	**TPolyMarker3D**(class TPolyMarker3D& *polymarker*)
void	**~TPolyMarker3D**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *polymarker*)
Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawPolyMarker**(Int_t *n,* Float_t* *p,* Marker_t *marker,* Option_t* *option*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Int_t	**GetN**( )
const Option_t*	**GetOption**( )
virtual Float_t*	**GetP**( )
virtual void	**GetPolyMarker**(Int_t *n,* Float_t* *p,* Marker_t *marker,* Option_t* *option*)
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintPolyMarker**(Int_t *n,* Float_t* *p,* Marker_t *marker,* Option_t* *option*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
void	**SetPoint**(Int_t *n,* Float_t *x,* Float_t *y,* Float_t *z*)
virtual void	**SetPolyMarker**(Int_t *n,* Float_t* *p,* Marker_t *marker,* Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fN**	*number of points*
Float_t*	**fP**	*Array of X, Y, Z coordinates*
class TString	**fOption**	*options*
UInt_t	**fGLList**	*The list number for OpenGL view*

Class Description

```
PolyMarker3D is a 3D polymarker. It has three constructors.

First one, without any parameters TPolyMarker3D(), we call 'default
constructor' and it's used in a case that just an initialisation is
needed (i.e. pointer declaration).

    Example:
             TPolyMarker3D *pm = new TPolyMarker3D;

Second one, takes, usually, two parameters, n (number of points) and
marker (marker style). Third parameter is optional.

    Example:
             TPolyMarker3D (150, 1);
```

```
Third one takes, usually, three parameters, n (number of points), *p
(pointer to an array of 3D points), and marker (marker style). Fourth
parameter is optional.

    Example:
              Float_t *ptr = new Float_t [150*3];
                      ... ... ...
                      ... ... ...
                      ... ... ...

              TPolyMarker3D (150, ptr, 1);
```

TPolyMarker3D( ): **TObject**( ), **TAttMarker**( )
`PolyMarker3D default constructor`

TPolyMarker3D(Int_t *n,* Marker_t *marker,* Option_t **option*): **TObject**( ), **TAttMarker**( )
`PolyMarker3D normal constructor`

TPolyMarker3D(Int_t *n,* Float_t **p,* Marker_t *marker,* Option_t **option*): **TObject**( ), **TAttMarker**( )
`PolyMarker3D constructor`

~TPolyMarker3D( )
`PolyMarker3D destructor`

void **Copy**(TObject &*obj*)
`Copy polymarker`

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a 3-D polymarker
Compute the closest distance of approach from point px,py to each segment
of the polyline.
Returns when the distance found is below DistanceMaximum.
The distance is computed in pixels units.
```

void **Draw**(Option_t **option*)
`Draws PolyMarker and adds it into the ListOfPrimitives`

void **DrawPolyMarker**(Int_t *n,* Float_t **p,* Marker_t *marker,* Option_t **option*)
`Draws PolyMarker`

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the window
```

void **GetPolyMarker**(Int_t *n,* Float_t **p,* Marker_t *marker,* const Option_t **option*)
`Returns PolyMarker's parameters and data`

void **ls**(Option_t **option*)
`List PolyMarker's contents`

void **Paint**(Option_t **option*)
`Paint PolyMarker`

void **PaintPolyMarker**(Int_t *n,* Float_t **p,* Marker_t *marker,* Option_t **option*)
`Paint polymarker in CurrentPad World coordinates`

void **Print**(Option_t **option*)
`Print PolyMarker Info`

void **SavePrimitive**(ofstream &*out,* Option_t **option*)
`Save primitive as a C++ statement(s) on output stream out`

void **SetPoint**(Int_t *n*, Float_t *x*, Float_t *y*, Float_t *z*)

```
Set point n to x, y, z
```

void **SetPolyMarker**(Int_t *n*, Float_t **p*, Marker_t *marker*, Option_t **option*)

```
Loads n points from array p
```

void **Sizeof3D**( ) const

```
Return total size of this 3-D shape with its attributes
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

Author: Nenad Buncic 21/08/95
Last update: 1.01/05 11/06/97 16.49.51 by Fons Rademakers

TPopupMenu

class **TPopupMenu** : public TMenu

private:

void	**CreatePopupMenu**(MenuDesc_t* *menuDesc,* XtPointer *defaultClientData*)
void	**Popup**(Widget *w,* XtPointer *clientData,* XEvent* *event,* Boolean*)

public:

class TPopupMenu	**TPopupMenu**(char* *name = PopupMenu,* MenuDesc_t* *menuDesc = NULL,* XtPointer *defaultClientData = NULL*)
class TPopupMenu	**TPopupMenu**(MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL*)
class TPopupMenu	**TPopupMenu**(Widget *parent,* MenuDesc_t* *menuDesc,* XtPointer *defaultClientData = NULL*)
class TPopupMenu	**TPopupMenu**(class TPopupMenu&)
class TPopupMenu	**TPopupMenu**(Widget *parent,* char* *name = PopupMenu,* MenuDesc_t* *menuDesc = NULL,* XtPointer *defaultClientData = NULL*)
void	**~TPopupMenu**( )
virtual void	**Attach**(Widget *w*)
virtual void	**Build**(Widget *parent*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**Show**(XEvent* *buttonPressEvent*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TPopupMenu

This class provides an interface to popup menus. These menus pop
up when the user hits the right mouse button.
```

void **Popup**(*Widget,*XtPointer clientData,XEvent *event,Boolean *)

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 09/09/97 09.02.24 by René Brun

TPostScript

class **TPostScript** : public TNamed, public TAttLine, public TAttFill, public TAttMarker, public TAttText

public:

class TPostScript	**TPostScript**(Text_t* *filename,* Int_t *type = -111*)
class TPostScript	**TPostScript**(class TPostScript&)
class TPostScript	**TPostScript**( )
void	**~TPostScript**( )
class TClass*	**Class**( )
void	**Close**( )
Int_t	**CMtoPS**(Float_t *u*)
void	**DefineFillPatterns**( )
void	**DefineMarkers**( )
void	**DrawBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)
void	**DrawFrame**(Coord_t *xl,* Coord_t *yl,* Coord_t *xt,* Coord_t *yt,* Int_t *mode,* Int_t *border,* Int_t *dark,* Int_t *light*)
void	**DrawHatch**(Float_t *dy,* Float_t *angle,* Int_t *n,* Float_t* *x,* Float_t* *y*)
void	**DrawPolyLine**(Int_t *n,* class TPoints* *xy*)
void	**DrawPolyLineNDC**(Int_t *n,* class TPoints* *uv*)
void	**DrawPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y*)
void	**DrawPS**(Int_t *n,* Float_t* *xw,* Float_t* *yw*)
void	**FontEncode**( )
Color_t	**GetFillColor**( )
Style_t	**GetFillStyle**( )
Color_t	**GetLineColor**( )
Style_t	**GetLineStyle**( )
Width_t	**GetLineWidth**( )
Color_t	**GetMarkerColor**( )
Size_t	**GetMarkerSize**( )
Style_t	**GetMarkerStyle**( )
Short_t	**GetTextAlign**( )
Float_t	**GetTextAngle**( )
Color_t	**GetTextColor**( )
Int_t	**GetTextFont**( )
Float_t	**GetTextSize**( )
void	**Initialize**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
void	**MakeGreek**( )
void	**MovePS**(Int_t *x,* Int_t *y*)
void	**NewPage**( )
void	**Off**( )
void	**On**( )
void	**Open**(Text_t* *filename,* Int_t *type = -111*)
virtual void	**Print**(Text_t* *string*)
virtual void	**Print1**(Text_t* *string*)
virtual void	**PrintFast**(Int_t *nch,* Text_t* *string*)
void	**Range**(Float_t *xrange,* Float_t *yrange*)
void	**Resize**( )
void	**SaveRestore**(Int_t *flag*)
void	**SetColor**(Int_t *color = 1*)
void	**SetFillColor**(Color_t *cindex = 1*)
void	**SetFillStyle**(Style_t *style*)
void	**SetLineColor**(Color_t *cindex = 1*)
void	**SetLineStyle**(Style_t *linestyle = 1*)

void	**SetLineWidth**(Width_t *linewidth = 1*)
void	**SetMarkerColor**(Color_t *cindex = 1*)
void	**SetMarkerSize**(Float_t *markersize*)
void	**SetMarkerStyle**(Style_t *markerstyle*)
void	**SetTextAlign**(Short_t *talign = 11*)
void	**SetTextAngle**(Float_t *tangle = 0*)
void	**SetTextColor**(Color_t *cindex = 1*)
void	**SetTextFont**(Int_t *fontnumber = 62*)
void	**SetTextSize**(Float_t *tsize*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Text**(Float_t *x,* Float_t *y,* Text_t* *string*)
void	**TextNDC**(Float_t *u,* Float_t *v,* Text_t* *string*)
Int_t	**UtoPS**(Float_t *u*)
Int_t	**VtoPS**(Float_t *v*)
void	**WriteInteger**(Int_t *i*)
void	**WriteReal**(Float_t *r*)
Int_t	**XtoPS**(Float_t *x*)
Int_t	**YtoPS**(Float_t *y*)
void	**Zone**( )

Data Members

protected:

Float_t	**fX1v**	*X bottom left corner of paper*
Float_t	**fY1v**	*Y bottom left corner of paper*
Float_t	**fX2v**	*X top right corner of paper*
Float_t	**fY2v**	*Y top right corner of paper*
Float_t	**fX1w**	
Float_t	**fY1w**	
Float_t	**fX2w**	
Float_t	**fY2w**	
Float_t	**fDXC**	
Float_t	**fDYC**	
Float_t	**fXC**	
Float_t	**fYC**	
Float_t	**fFX**	
Float_t	**fFY**	
Float_t	**fXVP1**	
Float_t	**fXVP2**	
Float_t	**fYVP1**	
Float_t	**fYVP2**	
Float_t	**fXVS1**	
Float_t	**fXVS2**	
Float_t	**fYVS1**	
Float_t	**fYVS2**	
Float_t	**fXsize**	*Page size along X*
Float_t	**fYsize**	*Page size along Y*
Float_t	**fMaxsize**	*Largest dimension of X and Y*
Float_t	**fRed**	*Per cent of red*
Float_t	**fGreen**	*Per cent of green*
Float_t	**fBlue**	*Per cent of blue*
Int_t	**fSave**	*Number of gsave for restore*
Int_t	**fNXzone**	*Number of zones along X*
Int_t	**fNYzone**	*Number of zones along Y*
Int_t	**fIXzone**	*Current zone along X*
Int_t	**fIYzone**	*Current zone along Y*
Int_t	**fLenBuffer**	*buffer length*
Float_t	**fMarkerSizeCur**	*current transformed value of marker size*

Int_t	**fCurrentColor**	*current Postscript color index*
Int_t	**fNpages**	*number of pages*
Int_t	**fType**	*PostScript workstation type*
Int_t	**fMode**	*PostScript mode*
Int_t	**fClip**	*Clipping mode*
Bool_t	**fBoundingBox**	*True for Encapsulated PostScript*
Bool_t	**fClear**	*True when page must be cleared*
Bool_t	**fClipStatus**	*Clipping Indicator*
Bool_t	**fPrinted**	*True when a page must be printed*
Bool_t	**fRange**	*True when a range has been defined*
Bool_t	**fZone**	*Zone indicator*
class ofstream*	**fStream**	*File stream identifier*
Text_t	**fBuffer**	*PostScript file buffer*

Class Description

```
The  P O S T S C R I P T  class
Graphics interface to PostScript.

This code was initially developped in the context of HIGZ and PAW
by Olivier Couet and Pierre Juillot.
It has been converted to a C++ class by René Brun.
```

TPostScript( ):**TNamed**( )

```
Default PostScript constructor
```

TPostScript(Text_t **fname,* Int_t *wtype*):**TNamed**(fname, "PostScript"),**TAttLine**( ), **TAttFill**( ), **TAttText**( ), **TAttMarker**( )

```
Initialize the PostScript interface
_Input parameters:

fname : PostScript file name
wtype : PostScript workstation type

The PostScript workstation types have the following format:
                [Format][Nx][Ny][Type]
   Where:
[Format] Is an integer between 0 and 99 defining the format of the
        paper. For example if Format=3 the paper is in the standard
        A3 format. Format=4 and Format=0 are the same and
        define an A4 page. The A0 format is selected by Format=99.
        The US format Letter is selected by Format=100.
        The US format Legal is selected by  Format=200.
        The US format Ledger is selected by Format=300.
[Nx, Ny] Specify respectively the number of zones on the x and y axis.
             Nx and Ny are integers between 1 and 9.
[Type] Can be equal to:
 1  Portrait mode with a small margin at the bottom of the page.
 2  Landscape mode with a small margin at the bottom of the page.
 4  Portrait mode with a large margin at the bottom of the page.
 5  Landscape mode with a large margin at the bottom of the page.

The large margin is useful for some PostScript printers as they
need more space to grip the paper for mechanical reasons.

Note that some PostScript colour printers can also use the so called
"special A4" format permitting the full usage of the A4 area; In this
case larger margins are not necessary and Type=1 or 2 can be used.

 3  Encapsulated PostScript. This Type permits the generation of
        files which can be included in other documents, for example
        in LaTeX files. Note that with this Type, Nx and Ny must
        always be equal to 1, and Format has no meaning.
```

Therefore the workstation type for Encapsulated PostScript is 113.

With Type=1,2,4 and 5 the pictures are centered on the page, and the usable area on paper is proportional to the dimensions of A4 format.

Examples:

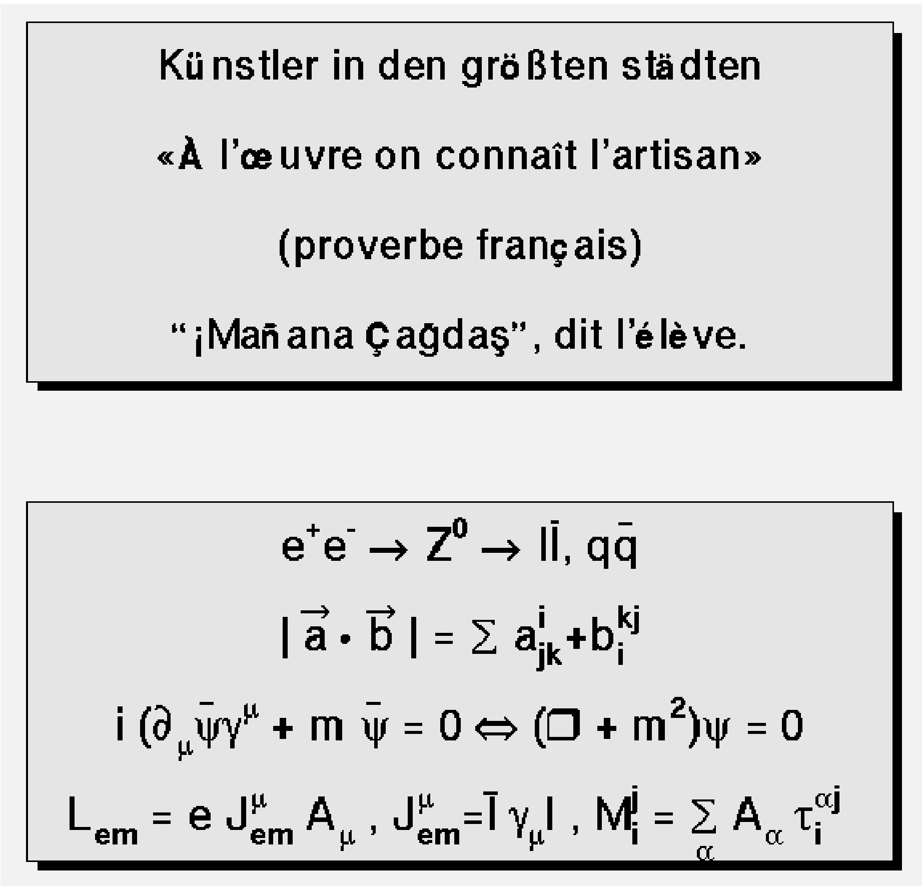
111 or 4111 defines an A4 page not divided.
6322 define an A6 landscape page divided in 3 columns and 2 rows.

```
*********************
*  1   *  2   *  3
*********************
*  4   *  5   *  6
*********************
```

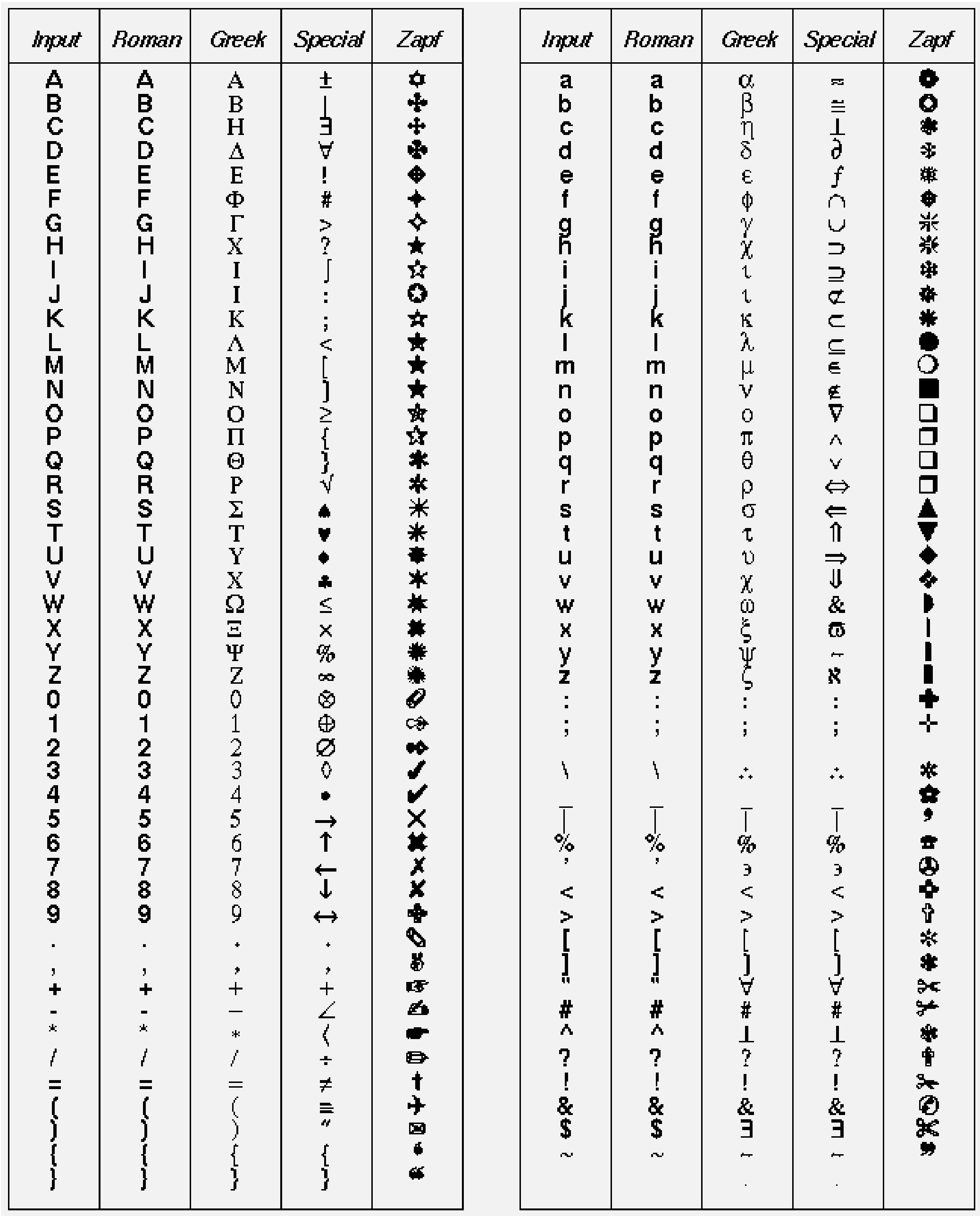
The first picture will be drawn in the area 1. The next image will appear in the next area in the order defined above. If a page is filled, a new page is used with the same grid.
Note that empty pages are not printed in order to save paper.

Ignoring formats smaller than A12, the total number of different PostScript workstation types is: 4 x 9 x 9 x 13 + 1 = 4213 !

The picture below shows fancy text with national accents or subscripts and superscripts. This picture has been generated by the macro psexam.



The two following tables list the correspondance between the typed character and its interpretation using the special characters given in TPostScript::Text. These tables are screen copies. True and better resolution PostScript files can be seen at psexam, pstable1 and pstable2. The macro pstable has been used to generate the two PostScript tables.



Input	Roman	Greek	Special	Zapf
\241	¡	ϒ	ϒ	❡
\242	¢	′	′	❢
\243	£	≤	≤	❣
\244	⁄	⁄	⁄	❤
\245	¥	∞	∞	❥
\246	ƒ	ƒ	ƒ	❦
\247	§	♣	♣	❧
\250	¤	♦	♦	♣
\251	'	♥	♥	♦
\252	“	♠	♠	♥
\253	«	↔	↔	♠
\254	‹	←	←	①
\255	›	↑	↑	②
\256	ﬁ	→	→	③
\257	ﬂ	↓	↓	④
\260	à	°	°	⑤
\261	–	±	±	⑥
\262	†	″	″	⑦
\263	‡	≥	≥	⑧
\264	·	×	×	⑨
\265	À	∝	∝	⑩
\266	¶	∂	∂	❶
\267	•	•	•	❷
\270	‚	÷	÷	❸
\271	„	≠	≠	❹
\272	”	≡	≡	❺
\273	»	≈	≈	❻
\274	…	…	…	❼
\275	‰	⏐	⏐	❽
\276	ã	⎯	⎯	❾
\277	¿	↵	↵	❿
\300	Ã	ℵ	ℵ	➀
\301	`	ℑ	ℑ	➁
\302	´	ℜ	ℜ	➂
\303	ˆ	℘	℘	➃
\304	˜	⊗	⊗	➄
\305	¯	⊕	⊕	➅
\306	˘	∅	∅	➆
\307	˙	∩	∩	➇
\310	¨	∪	∪	➈
\311	ä	⊃	⊃	➉
\312	˚	⊇	⊇	➊
\313		⊄	⊄	➋
\314	Ä	⊂	⊂	➌
\315	˝	⊆	⊆	➍
\316	˛	∈	∈	➎
\317		∉	∉	➏

Input	Roman	Greek	Special	Zapf
\321	ç	∇	∇	➑
\322	Ç	®	®	➒
\323	é	©	©	➓
\324	É	™	™	➔
\325	è	∏	∏	→
\326	È	√	√	↔
\327	ê	⋅	⋅	↕
\330	Ê	¬	¬	➘
\331	ë	∧	∧	➙
\332	Ë	∨	∨	➚
\333	î	⇔	⇔	➛
\334	Î	⇐	⇐	➜
\335	ï	⇑	⇑	➝
\336	Ï	⇒	⇒	➞
\337	ñ	⇓	⇓	➟
\340	Ñ	◊	◊	➠
\341	Æ	〈	〈	➡
\342	ô	®	®	➢
\343	ª	©	©	➣
\344	Ô	™	™	➤
\345	ö	∑	∑	➥
\346	Ö	⎛	⎛	➦
\347	û	⎜	⎜	➧
\340	Ñ	⎝	⎝	➨
\351	Ø	⎡	⎡	➩
\352	Œ	⎢	⎢	➪
\353	º	⎣	⎣	➫
\354	Û	⎧	⎧	➬
\355	ü	⎨	⎨	➭
\356	Ü	⎩	⎩	➮
\357	å	⎪	⎪	➯
\360				
\361	æ	〉	〉	➱
\362	Å	∫	∫	➲
\363	ÿ	⌠	⌠	➳
\364	Ÿ	⎮	⎮	➴
\365	ı	⌡	⌡	➵
\366	á	⎞	⎞	➶
\367	Á	⎟	⎟	➷
\370	ł	⎠	⎠	➸
\371	ø	⎤	⎤	➹
\372	œ	⎥	⎥	➺
\373	ß	⎦	⎦	➻
\374	ù	⎫	⎫	➼
\375	Ù	⎬	⎬	➽
\376		⎭	⎭	➾
\377				

void **Open**(Text_t **fname,* Int_t *wtype*)

    Open a PostScript file

~TPostScript( )

    Default PostScript destructor

void **Close**( )

    Close a PostScript file

void **On**( )

    Activate an already open PostScript file

void **Off**( )

    DeActivate an already open PostScript file

void **DefineFillPatterns**( )

    Define the fill patterns from 1 to 25

void **DefineMarkers**( )

```
Define the markers
```

void **DrawBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2*)

```
Draw a Box
```

void **DrawFrame**(Coord_t *xl,* Coord_t *yl,* Coord_t *xt,* Coord_t *yt,*Int_t mode, Int_t border, Int_t dark, Int_t *light*)

```
Draw a Frame around a box
mode = -1  box looks as it is behind the screen
mode =  1  box looks as it is in front of the screen
border is the bordersize in already precomputed PostScript units
dark  is the color for the dark part of the frame
light is the color for the light part of the frame
```

void **DrawPolyLine**(Int_t *nn,* TPoints **xy*)

```
Draw a PolyLine
Draw a polyline through  the points  xy.
If NN=1 moves only to point x,y.
If NN=0 the x,y are  written  in the PostScript file
  according to the current tranformation.
If NN>0 the line is clipped as a line.
If NN<0 the line is clipped as a fill area.
```

void **DrawPolyLineNDC**(Int_t *nn,* TPoints **xy*)

```
Draw a PolyLine in NDC space
Draw a polyline through  the points  xy.
If NN=1 moves only to point x,y.
If NN=0 the x,y are  written  in the PostScript file
  according to the current tranformation.
If NN>0 the line is clipped as a line.
If NN<0 the line is clipped as a fill area.
```

void **DrawPolyMarker**(Int_t *n,* Float_t **x,* Float_t **y*)

```
Draw markers at the n WC points x, y
```

void **DrawPS**(Int_t *nn,* Float_t **xw,* Float_t **yw*)

```
Draw a PolyLine
Draw a polyline through  the points  XW,YW.
If NN=1 moves only to point XW,YW.
If NN=0 the XW(1) and YW(1) are  written  in the PostScript file
  according to the current NT.
If NN>0 the line is clipped as a line.
If NN<0 the line is clipped as a fill area.
```

void **DrawHatch**(Float_t *dy,* Float_t *angle,* Int_t *n,* Float_t **x,* Float_t **y*)

```
Draw Fill area with hacth styles
```

void **FontEncode**( )

```
Font Reencoding
*-- Author :    P.Juillot   13/08/92
```

void **Initialize**( )

```
PostScript Initialisation
This routine initialize the following PostScript procedures:

+------------+------------------+-----------------------------------+
| Macro Name | Input parameters |            Explanation            |
+------------+------------------+-----------------------------------+
|     l      | x y              | Draw a line to the x y position   |
+------------+------------------+-----------------------------------+
|     m      | x y              | Move to the position x y          |
+------------+------------------+-----------------------------------+
|    box     | dx dy x y        | Define a box                      |
```

bl	dx dy x y	Draw a line box
bf	dx dy x y	Draw a filled box
sw	text	Return string width of text
t	x y	Translate
r	angle	Rotate
rl	i j	Roll the stack
d	x y	Draw a relative line to x y
X	x	Draw a relative line to x (y=0)
Y	y	Draw a relative line to y (x=0)
rm	x y	Move relatively to x y
gr		Restore the graphic context
lw	lwidth	Set line width to lwidth
sd	[] 0	Set dash line define by []
s		Stroke mode
c	r g b	Set rgb color to r g b
cl		Close path
f		Fill the last describe path
mXX	x y	Draw the marker type XX at (x,y)
Zone	ix iy	Define the current zone
black		The color is black
C	dx dy x y	Clipping on
NC		Clipping off
R		repeat

void **ls**(Option_t **option*)

`List current PostScript parameters`

void **MakeGreek**( )

`Reencode the Greek (/Symbol) font into the special font (/Special)`

void **MovePS**(Int_t *ix,* Int_t *iy*)

`Move to a new position`

void **NewPage**( )

`Move to a new PostScript page`

void **Print1**(const Text_t **str*)

`Output the string STR in the output buffer`

void **Print**(const Text_t **str*)
```
Output the string STR in the output buffer
```

void **PrintFast**(Int_t *len,* const Text_t **str*)
```
Fast version of Print
```

void **Range**(Float_t *xsize,* Float_t *ysize*)
```
Set the range for the paper in centimeters
```

void **Resize**( )
```
Compute conversion coefficients
```

void **SaveRestore**(Int_t *flag*)
```
Compute number of gsaves for restore
This allows to write the correct number of grestore at the
end of the PS file.
```

void **SetFillColor**(Color_t *cindex*)
```
Set color index for fill areas
cindex : color index
```

void **SetFillStyle**(Style_t *fstyle*)
```
Set fill area style
fstyle   : compound fill area interior style

fstyle = 1000*interiorstyle + styleindex
```

void **SetLineColor**(Color_t *cindex*)
```
Set color index for lines
cindex    : color index
```

void **SetLineStyle**(Style_t *linestyle*)
```
Change the line style
linestyle = 2 dashed
          = 3  dotted
          = 4  dash-dotted
           else = solid
```

void **SetLineWidth**(Width_t *linewidth*)
```
Change the line width
```

void **SetMarkerColor**(Color_t *cindex*)
```
Set color index for markers
cindex : color index
```

void **SetMarkerSize**(Float_t *msize*)
```
Set marker size index for markers
msize  : marker scale factor
```

void **SetMarkerStyle**(Style_t *markerstyle*)
```
Set marker style
```

void **SetColor**(Int_t *color*)
```
Set the current color
```

void **SetTextAlign**(Short_t *talign*)
```
Set text alignment
txalh   : horizontal text alignment
txalv   : vertical text alignment
```

void **SetTextAngle**(Float_t *tangle*)
```
Set text angle
tangle   : angle in degrees
```

void **SetTextColor**(Color_t *cindex*)
```
Set color index for text
cindex    : color index
```

void **SetTextFont**(Int_t *fontnumber*)
```
Set current text font number
```

void **SetTextSize**(Float_t *textsize*)
```
Set current text size
```

void **Text**(Float_t *xx*, Float_t *yy*, const Text_t **chars*)
```
Write a string of characters
This routine writes the string chars into a PostScript file
 at position xx,yy in world coordinates.

Note the special action of the following special characters:

    ` : go to greek
    ' : go to special
    ~ : go to ZapfDingbats
    ? : go to subscript
    ^ : go to superscript
    ! : go to normal level of script
    & : backspace one character
    # : end of greek or of ZapfDingbats

This special characters have no effect on the screen.
```

void **TextNDC**(Float_t *u*, Float_t *v*, const Text_t **chars*)
```
Write a string of characters in NDC
```

Int_t **UtoPS**(Float_t *u*)
```
Convert U from NDC coordinate to PostScript
```

Int_t **VtoPS**(Float_t *v*)
```
Convert V from NDC coordinate to PostScript
```

void **WriteInteger**(Int_t *n*)
```
Write one Integer to the PostScript file
```

void **WriteReal**(Float_t *z*)
```
Write a Real number to the PostScript file
```

Int_t **XtoPS**(Float_t *x*)
```
Convert X from world coordinate to PostScript
```

Int_t **YtoPS**(Float_t *y*)
```
Convert Y from world coordinate to PostScript
```

void **Zone**( )
```
Initialize the PostScript page in zones
```

Author: René Brun, Olivier Couet, Pierre Juillot 29/11/94
Last update: 1.03/04 17/09/97 13.51.57 by René Brun

TProfile

class **TProfile** : public TH1D

public:

class TProfile	**TProfile**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Option_t* *option*)
class TProfile	**TProfile**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t *xlow,* Axis_t *xup,* Axis_t *ylow,* Axis_t *yup,* Option_t* *option*)
class TProfile	**TProfile**(Text_t* *name,* Text_t* *title,* Int_t *nbinsx,* Axis_t* *xbins,* Option_t* *option*)
class TProfile	**TProfile**( )
class TProfile	**TProfile**(class TProfile& *profile*)
void	**~TProfile**( )
void	**BuildOptions**(Float_t *ymin,* Float_t *ymax,* Option_t* *option*)
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *hnew*)
virtual void	**DrawCopy**(Option_t* *option*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)
virtual void	**Fill**(Axis_t *x,* Axis_t *y*)
virtual void	**FillN**(Int_t *ntimes,* Axis_t* *x,* Axis_t* *y,* Double_t* *w*)
virtual Stat_t	**GetBinContent**(Int_t *bin*)
virtual Stat_t	**GetBinError**(Int_t *bin*)
const Option_t*	**GetErrorOption**( )
virtual class TClass*	**IsA**( )
virtual void	**Reset**(Option_t* *option*)
virtual void	**SetBins**(Int_t *nbins,* Float_t *xmin,* Float_t *xmax*)
virtual void	**SetErrorOption**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

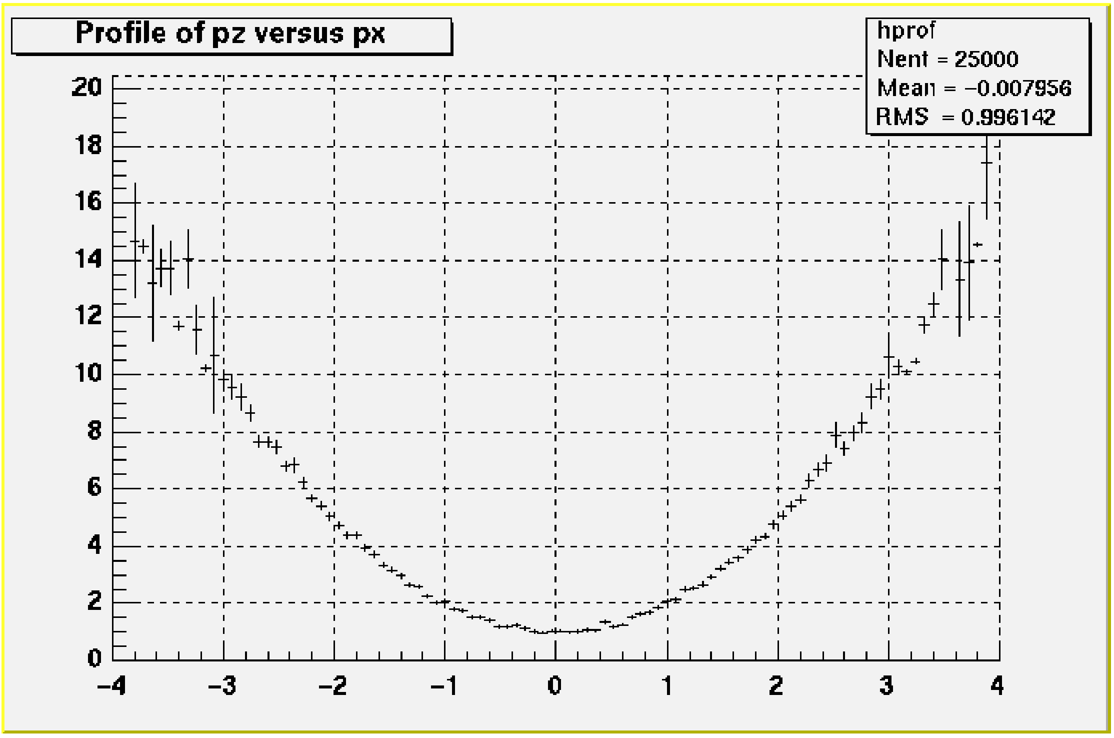
protected:

class TArrayD	**fBinEntries**	*number of entries per bin*
enum EErrorType	**fErrorMode**	*Option to compute errors*
Float_t	**fYmin**	*Lower limit in Y (if set)*
Float_t	**fYmax**	*Upper limit in Y (if set)*

Class Description

```
Profile histograms are used to display the mean
value of Y and its RMS for each bin in X. Profile histograms are in many cases
an
elegant replacement of two-dimensional histograms : the inter-relation of two
measured quantities X and Y can always be visualized by a two-dimensional
histogram or scatter-plot; its representation on the line-printer is not par-
ticularly
satisfactory, except for sparse data. If Y is an unknown (but single-valued)
approximate function of X, this function is displayed by a profile histogram
with
much better precision than by a scatter-plot.

The following formulae show the cumulated contents (capital letters) and the
values
displayed by the printing or plotting routines (small letters) of the elements
for bin J.

                                                         2
   H(J)  =  sum Y                   E(J)  =  sum Y
```

```
   l(J)   =  sum l                   L(J)   =  sum l
   h(J)   =  H(J)/L(J)               s(J)   =  sqrt(E(J)/L(J)- h(J)**2)
   e(J)   =  s(J)/sqrt(L(J))

        Example of a profile histogram with its graphics output
{
TCanvas c1 = new TCanvas("c1","Profile histogram example",200,10,700,500);
hprof  = new TProfile("hprof","Profile of pz versus px",100,-4,4,0,20);
Float_t px, py, pz;
for ( Int_t i=0; i<25000; i++) {
  gRandom->Rannor(&px,&py);
  pz = px*px + py*py;
  hprof->Fill(px,pz,1);
}
hprof->Draw();
}
```



TProfile( ): **TH1D**( )

```
Default constructor for Profile histograms
```

~TProfile( )

```
Default destructor for Profile histograms
```

TProfile(Text_t **name*, Text_t **title*, Int_t *nbins*, Axis_t *xlow*, Axis_t *xup*, Option_t **option*):**TH1D**(*name, title, nbins, xlow, xup*)

```
Normal Constructor for Profile histograms
The first five parameters are similar to TH1D::TH1D.
All values of y are accepted at filling time.
To fill a profile histogram, one must use TProfile::Fill function.

Note that when filling the profile histogram the function Fill
checks if the variable y is betyween fYmin and fYmax.
If a minimum or maximum value is set for the Y scale before filling,
then all values below ymin or above ymax will be discarded.
Setting the minimum or maximum value for the Y scale before filling
has the same effect as calling the special TProfile constructor below
```

```
where ymin and ymax are specified.

H(J) is printed as the channel contents. The errors displayed are s(J) if
CHOPT='S'
(spread option), or e(J) if CHOPT=' ' (error on mean).

     See TProfile::BuildOptions for explanation of parameters
```

TProfile(Text_t **name,* Text_t **title,* Int_t *nbins,* Axis_t **xbins,* Option_t **option*):**TH1D**(*name, title, nbins, xbins*)

```
Constructor for Profile histograms with variable bin size
     See TProfile::BuildOptions for more explanations on errors
```

TProfile(Text_t **name,* Text_t **title,* Int_t *nbins,* Axis_t *xlow,* Axis_t *xup,* Axis_t *ylow,* Axis_t *yup,* Option_t **option*):**TH1D**(*name, title, nbins, xlow, xup*)

```
Constructor for Profile histograms with range in y
The first five parameters are similar to TH1D::TH1D.
Only the values of Y between YMIN and YMAX will be considered at filling time.
ymin and ymax will also be the maximum and minimum values
on the y scale when drawing the profile.

     See TProfile::BuildOptions for more explanations on errors
```

void **BuildOptions**(Float_t *ymin,* Float_t *ymax,* Option_t **option*)

```
Set Profile histogram structure and options
 If a bin has N data points all with the same value Y (especially
 possible when dealing with integers), the spread in Y for that bin
 is zero, and the uncertainty assigned is also zero, and the bin is
 ignored in making subsequent fits. If SQRT(Y) was the correct error
 in the case above, then SQRT(Y)/SQRT(N) would be the correct error here.
 In fact, any bin with non-zero number of entries N but with zero spread
 should have an uncertainty SQRT(Y)/SQRT(N).

 Now, is SQRT(Y)/SQRT(N) really the correct uncertainty?
 that it is only in the case where the Y variable is some sort
 of counting statistics, following a Poisson distribution. This should
 probably be set as the default case. However, Y can be any variable
 from an original NTUPLE, not necessarily distributed "Poissonly".
 The computation of errors is based on the parameter option:
 option:
  ' '  (Default) Errors are Spread/SQRT(N) for Spread.ne.0. ,
                   "     "  SQRT(Y)/SQRT(N) for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
  's'            Errors are Spread  for Spread.ne.0. ,
                   "     "  SQRT(Y)  for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
  'i'            Errors are Spread/SQRT(N) for Spread.ne.0. ,
                   "     "  1./SQRT(12.*N) for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0

 The third case above corresponds to Integer Y values for which the
 uncertainty is +-0.5, with the assumption that the probability that Y
 takes any value between Y-0.5 and Y+0.5 is uniform (the same argument
 goes for Y uniformly distributed between Y and Y+1); this would be
 useful if Y is an ADC measurement, for example. Other, fancier options
 would be possible, at the cost of adding one more parameter to the PROFILE
 command. For example, if all Y variables are distributed according to some
 known Gaussian of standard deviation Sigma, then:
  'G'            Errors are Spread/SQRT(N) for Spread.ne.0. ,
                   "     "  Sigma/SQRT(N) for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
 For example, this would be useful when all Y's are experimental quantities
 measured with the same instrument with precision Sigma.
```

void **Copy**(TObject &*obj*)

```
Copy a Profile histogram to a new profile histogram
```

void **DrawCopy**(Option_t **option*)

```
Draw a copy of this profile histogram
```

void **Fill**(Axis_t *x,* Axis_t *y*)

```
Fill a Profile histogram (no weights)
```

void **Fill**(Axis_t *x,* Axis_t *y,* Stat_t *w*)

```
Fill a Profile histogram with weights
```

void **FillN**(Int_t *ntimes,* Axis_t **x,* Axis_t **y,* Stat_t **w*)

```
Fill a Profile histogram with weights
```

Stat_t **GetBinContent**(Int_t *bin*)

```
Return bin content of a Profile histogram
```

Stat_t **GetBinError**(Int_t *bin*)

```
Return bin error of a Profile histogram
```

const Option_t* **GetErrorOption**( ) const

```
Return option to compute profile errors
```

void **Reset**(Option_t **option*)

```
Reset contents of a Profile histogram
```

void **SetBins**(Int_t *nx,* Float_t *xmin,* Float_t *xmax*)

```
Redefine  x axis parameters
```

void **SetErrorOption**(Option_t **option*)

```
Set option to compute profile errors
 The computation of errors is based on the parameter option:
 option:
  ' '  (Default) Errors are Spread/SQRT(N) for Spread.ne.0. ,
                   "     "  SQRT(Y)/SQRT(N) for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
  's'            Errors are Spread  for Spread.ne.0. ,
                   "     "  SQRT(Y)  for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
  'i'            Errors are Spread/SQRT(N) for Spread.ne.0. ,
                   "     "  1./SQRT(12.*N) for Spread.eq.0,N.gt.0 ,
                   "     "  0.  for N.eq.0
See TProfile::BuildOptions for explanation of all options
```

Author: René Brun 29/09/95
Last update: 1.02/00 09/07/97 21.14.01 by René Brun

TPromptDialog

class **TPromptDialog** : public TDialogManager

protected:

class TPromptDialog	**TPromptDialog**( )

public:

class TPromptDialog	**TPromptDialog**(char* *name*)
class TPromptDialog	**TPromptDialog**(class TPromptDialog&)
void	**~TPromptDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TPromptDialog

This class provides an interface to prompt dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TProof

class **TProof** : public TObject

private:

class TProof	**TProof**( )
class TProof	**TProof**(class TProof&)
Int_t	**Collect**(class TMonitor* *mon*)
void	**ConnectFiles**( )
void	**GetStatus**( )
void	**GetUserInfo**( )
Int_t	**Init**(char* *cluster,* char* *service,* char* *master,* char* *vers,* Int_t *loglevel,* char* *confdir*)
void	**Limits**(class TSocket* *s,* class TMessage& *mess*)
void	**MarkBad**(class TSocket* *s*)
void	**MarkBad**(class TSlave* *sl*)
void	***operator=***(class TProof&)
Int_t	**SendGroupView**( )
Int_t	**SendInitialState**( )

public:

class TProof	**TProof**(char* *cluster,* char* *master* = pcna49a.cern.*ch,* char* *vers* = *kPROOF_Version,* char* *service* = *proofslave,* Int_t *loglevel* = *1,* char* *confdir* = /usr/*proof*)
void	**~TProof**( )
Bool_t	**Active**( )
Int_t	**Broadcast**(class TMessage& *mess,* enum TProof::ESlaves *list* = *kActive*)
Int_t	**Broadcast**(char* *mess,* Int_t *kind* = *kMESS_STRING,* enum TProof::ESlaves *list* = *kActive*)
Int_t	**Broadcast**(Int_t *kind,* enum TProof::ESlaves *list* = *kActive*)
class TClass*	**Class**( )
void	**Close**( )
Int_t	**Collect**(enum TProof::ESlaves *list* = *kActive*)
Int_t	**Collect**(class TSlave* *sl*)
Int_t	**ConnectFile**(class TFile* *file*)
Int_t	**DisConnectFile**(class TFile* *file*)
class TSlave*	**FindSlave**(class TSocket* *s*)
UInt_t	**GetBytesRead**( )
const char*	**GetClusterName**( )
const char*	**GetConfDir**( )
const char*	**GetConfFile**( )
Float_t	**GetCpuTime**( )
class TList*	**GetListOfActiveSlaves**( )
class TList*	**GetListOfBadSlaves**( )
class TList*	**GetListOfSlaves**( )
Int_t	**GetLogLevel**( )
const char*	**GetMaster**( )
Int_t	**GetNumberOfActiveSlaves**( )
Int_t	**GetNumberOfBadSlaves**( )
Int_t	**GetNumberOfSlaves**( )
Int_t	**GetProtocol**( )
Float_t	**GetRealTime**( )
const char*	**GetService**( )
const char*	**GetUser**( )
const char*	**GetVersion**( )
void	**Interrupt**(enum TProof::EUrgent *type,* enum TProof::ESlaves *list* = *kActive*)
virtual class TClass*	**IsA**( )

Bool_t	**IsMaster**( )
Bool_t	**IsValid**( )
void	**Loop**(class TTree* *tree*)
Int_t	**Ping**(enum TProof::ESlaves *list = kActive*)
void	**Print**(Option_t* *option*)
void	**RecvLogFile**(class TSocket* *s*)
Int_t	**SendCommand**(char* *cmd,* enum TProof::ESlaves *list = kActive*)
Int_t	**SendCurrentState**(enum TProof::ESlaves *list = kActive*)
Int_t	**SendObject**(class TObject* *obj,* enum TProof::ESlaves *list = kActive*)
void	**SetLogLevel**(Int_t *level*)
Int_t	**SetParallel**(Int_t *nodes = 9999*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TString	**fCluster**	*name of cluster, name will be used to find config file*
class TString	**fService**	*service we are connected to, either "proofserv" or "proofslave"*
class TString	**fMaster**	*name of master server (in case of "proofserv")*
class TString	**fVersion**	*proof server major version*
class TString	**fConfDir**	*directory containing cluster config information*
class TString	**fConfFile**	*file containing config information*
class TString	**fUser**	*user under which to run*
class TString	**fPasswd**	*user password*
Int_t	**fProtocol**	*protocol level*
Int_t	**fLogLevel**	*server debug logging level*
Bool_t	**fMasterServ**	*true if we are a master server*
class TList*	**fSlaves**	*list of all slave servers as in config file*
class TList*	**fActiveSlaves**	*list of active slaves (subset of all slaves)*
class TList*	**fBadSlaves**	*dead slaves (subset of all slaves)*
class TMonitor*	**fAllMonitor**	*monitor activity on all valid slave sockets*
class TMonitor*	**fActiveMonitor**	*monitor activity on all active slave sockets*
UInt_t	**fBytesRead**	*bytes read by all slaves during the session*
Float_t	**fRealTime**	*realtime spent by all slaves during the session*
Float_t	**fCpuTime**	*CPU time spent by all slaves during the session*
class TTree*	**fTree**	*Object being PROOFed*
Int_t	**fLimits**	*Used by Limits( )*

public:

static const enum TProof::ESlaves**kAll**
static const enum TProof::ESlaves**kActive**
static const enum TProof::EUrgent**kHardInterrupt**
static const enum TProof::EUrgent**kSoftInterrupt**
static const enum TProof::EUrgent**kShutdownInterrupt**

Class Description

```
TProof

This class controls a Parallel ROOT Facility, PROOF, cluster.
It fires the slave servers, it keeps track of how many slaves are
running, it keeps track of the slaves running status, it broadcasts
messages to all slaves, it collects results, etc.
```

TProof(const char **cluster,* const char **master,* const char **version,*const char *service, Int_t loglevel, const char **confdir*)

```
Create a PROOF environment. Starting PROOF involves reading a config
file describing the cluster and firing slave servers on all of the
available nodes.
```

~TProof( )
```
Clean up PROOF environment.
```

Int_t **Init**(const char **cluster,* const char **service,* const char **master,*const char *vers, Int_t loglevel, const char **confdir*)
```
Start the PROOF environment. Starting PROOF involves reading a config
file describing the cluster and firing slave servers on all of the
available nodes.
```

Int_t **ConnectFile**(const TFile **file*)
```
Send message to all slaves to connect "file". This method is
called by the TFile ctor (no user method).
```

void **ConnectFiles**( )
```
Tell all servers to open all files currently opened by the client.
```

void **Close**( )
```
Close all open slave servers.
```

Int_t **DisConnectFile**(const TFile **file*)
```
Send message to all slaves to disconnect "file". This method is
called by the TFile::Close() (no user method).
```

TSlave* **FindSlave**(TSocket **s*) const
```
Find slave that has TSocket s. Returns 0 in case slave is not found.
```

void **GetUserInfo**( )
```
Get user info: user name and password. This info is needed to validate
the user on the PROOF cluster.
```

Int_t **GetNumberOfSlaves**( ) const
```
Return number of slaves as described in the config file.
```

Int_t **GetNumberOfActiveSlaves**( ) const
```
Return number of active slaves, i.e. slaves that are valid and in
the current computing group.
```

Int_t **GetNumberOfBadSlaves**( ) const
```
Return number of bad slaves. This are slaves that we in the config
file, but refused to startup or that died during the PROOF session.
```

void **GetStatus**( )
```
Get the status of the slaves.
```

void **Interrupt**(EUrgent *type,* ESlaves *list*)
```
Send interrupt OOB byte to master or slave servers.
```

Int_t **Broadcast**(const TMessage *&mess,* ESlaves *list*)
```
Broadcast a message to all slaves in the specified list (either
all slaves or only the active slaves). Returns the number of slaves
the message was successfully send to.
```

Int_t **Broadcast**(const char **str,* Int_t *kind,* ESlaves *list*)
```
Broadcast a character string buffer to all slaves in the specified
list (either all slaves or only the active slaves). Use kind to
set the TMessage what field. Returns the number of slaves the message
was send to.
```

Int_t **Collect**(const TSlave **sl*)
```
Collect responses from slave sl. Returns the number of slaves that
responded (=1).
```

Int_t **Collect**(ESlaves *list*)

```
Collect responses from the slave servers. Returns the number of slaves
that responded.
```

Int_t **Collect**(TMonitor **mon*)

```
Collect responses from the slave servers. Returns the number of slaves
that responded.
```

void **Limits**(TSocket **s,* TMessage *&mess*)

```
Calculate histogram limits after TTree::fEstimate events have
been processed.
This function is called via Collect() in response to a kPROOF_LIMITS
message send from a PROOF slave in TTree::TakeEstimate().
```

void **Loop**(TTree **tree*)

```
Handle message comming from the remote TTree method currently being
executed.
```

void **MarkBad**(TSlave **sl*)

```
Add a bad slave server to the bad slave list and remove it from
the active list and from the two monitor objects.
```

void **MarkBad**(TSocket **s*)

```
Add slave with socket s to the bad slave list and remove if from
the active list and from the two monitor objects.
```

Int_t **Ping**(ESlaves *list*)

```
Ping PROOF slaves. Returns the number of slaves that responded.
```

void **Print**(Option_t **option*)

```
Print status of PROOF cluster.
```

void **RecvLogFile**(TSocket **s*)

```
Receive the log file of the slave with socket s.
```

Int_t **SendGroupView**( )

```
Send to all active slaves servers the current slave group size
and their unique id. Returns number of active slaves.
```

Int_t **SendCommand**(const char **cmd,* ESlaves *list*)

```
Send command to be executed on the PROOF master and/or slaves.
Command can be any legal command line command.
```

Int_t **SendCurrentState**(ESlaves *list*)

```
Transfer the current state of the master to the active slave servers
just before starting the TTree loop. The current state includes: the
current working directory, TChain defintion, MaxVirtualSize, Selector,
etc.
```

Int_t **SendInitialState**( )

```
Transfer the initial (i.e. current) state of the master to all
slave servers. The initial state includes: log level, currently open
files.
```

Int_t **SendObject**(const TObject **obj,* ESlaves *list*)

```
Send object to master or slave servers. Returns number slaves message
was sent too, 0 in case of error.
```

void **SetLogLevel**(Int_t *level*)

```
Set server logging level.
```

Int_t **SetParallel**(Int_t *nodes*)

```
Set the number of nodes that should work in parallel. Since the fSlaves
list is sorted by slave performace the active list will contain first
```

```
the most performant nodes.
```

Bool_t **Active**( )

```
Static function that returns kTRUE in case a PROOF connection exists
with more than 1 active slave. When only one active slave we run in
sequential mode.
```

Author: Fons Rademakers 13/02/97
Last update: 1.00/07 01/04/97 01.45.10 by Fons Rademakers

TProofServ

class **TProofServ** : public TApplication

private:

void	**RedirectOutput**( )
void	**Setup**( )

public:

class TProofServ	**TProofServ**(int* *argc,* char** *argv*)
class TProofServ	**TProofServ**(class TProofServ&)
void	**~TProofServ**( )
Bool_t	**Active**( )
class TClass*	**Class**( )
class TObject*	**Get**(char* *namecycle*)
const char*	**GetConfDir**( )
Float_t	**GetCpuTime**( )
Stat_t	**GetEventsProcessed**( )
Int_t	**GetGroupId**( )
Int_t	**GetGroupSize**( )
void	**GetLimits**(Int_t *dim,* Int_t *nevents,* Int_t* *nbins,* Float_t* *vmin,* Float_t* *vmax*)
Int_t	**GetLogLevel**( )
Bool_t	**GetNextPacket**(Int_t& *nevents,* Stat_t& *firstevent*)
void	**GetOptions**(int* *argc,* char** *argv*)
Int_t	**GetOrdinal**( )
Int_t	**GetProtocol**( )
Float_t	**GetRealTime**( )
const char*	**GetService**( )
class TSocket*	**GetSocket**( )
const char*	**GetUser**( )
const char*	**GetVersion**( )
void	**HandleSocketInput**( )
void	**HandleUrgentData**( )
void	**Interrupt**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsMaster**( )
void	**Print**(Option_t* *option*)
void	**Reset**(char* *dir*)
void	**Run**(Bool_t *retrn = kFALSE*)
void	**SendLogFile**( )
void	**SendStatus**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Terminate**(int *status*)

Data Members

private:

class TString	**fService**	*service we are running, either "proofserv" or "proofslave"*
class TString	**fUser**	*user as which we run*
class TString	**fUserPass**	*encoded user and passwd info for slaves*
class TString	**fVersion**	*proof server major version*
class TString	**fConfDir**	*directory containing cluster config information*
class TString	**fLogDir**	*directory containing log files*
class TSocket*	**fSocket**	*socket connection to client*
FILE*	**fLogFile**	*log file*

Int_t	**fProtocol**	*protocol level*
Int_t	**fMasterPid**	*pid of master server*
Int_t	**fOrdinal**	*slaves (i.e. our)* ordinal number, -1 for master
Int_t	**fGroupId**	*our unique id in the active slave group*
Int_t	**fGroupSize**	*size of the active slave group*
Int_t	**fLogLevel**	*debug logging level*
Bool_t	**fMasterServ**	*true if we are a master server*
Int_t	**fNcmd**	*command history number*
Bool_t	**fInterrupt**	*if true macro execution will be stopped*
Float_t	**fRealTime**	*real time spent executing commands*
Float_t	**fCpuTime**	*CPU time spent executing commands*
Stat_t	**fEventsProcessed**	*total number of events processed (obtained via GetNextPacket)*

Class Description

```
TProofServ

TProofServ is the PROOF server. It can act either as the master
server or as a slave server, depending on its startup arguments. It
receives and handles message coming from the client or from the
master server.
```

TProofServ(int **argc,* char ***argv*): **TApplication**("proofserv", *argc, argv*)

```
Create an application environment. The TProofServ environment provides
an eventloop via inheritance of TApplication.
```

~TProofServ( )

```
Cleanup. Not really necessary since after this dtor there is no
live anyway.
```

TObject* **Get**(const char **namecycle*)

```
Get object with name "name;cycle" (e.g. "aap;2") from master or client.
This method is called by TDirectory::Get() in case the object can not
be found locally.
```

void **GetLimits**(Int_t *dim,* Int_t *nevents,* Int_t **nbins,* Float_t **vmin,* Float_t **vmax*)

```
Get limits of histogram from master. This method is called by
TTree::TakeEstimate().
```

Bool_t **GetNextPacket**(Int_t &*nevents,* Stat_t &*firstevent*)

```
Get next range of events to be processed on this server.
```

void **GetOptions**(int **argc,* char ***argv*)

```
Get and handle command line options.
```

void **HandleSocketInput**( )

```
Handle input coming from the client or from the master server.
```

void **HandleUrgentData**( )

```
Handle Out-Of-Band data sent by the master or client.
```

void **Print**(Option_t **option*)

```
Print status of slave server.
```

void **RedirectOutput**( )

```
Redirect stdout to a log file. This log file will be flushed to the
client or master after each command.
```

void **Reset**(const char **dir*)

```
Reset PROOF environment to be ready for execution of next command.
```

void **Run**(Bool_t *retrn*)

```
Main server eventloop.
```

void **SendLogFile**( )

```
Send log file to master.
```

void **SendStatus**( )

```
Send status of slave server to master or client.
```

void **Setup**( )

```
Print the ProofServ logo on standard output.
```

void **Terminate**(int *status*)

```
Terminate the proof server.
```

Bool_t **Active**( )

```
Static function that returns kTRUE in case we are a PROOF server.
```

Author: Fons Rademakers 16/02/97
Last update: 1.00/07 01/04/97 20.07.38 by Fons Rademakers

TQuestionDialog

class **TQuestionDialog** : public TDialogManager

protected:

class TQuestionDialog	**TQuestionDialog**( )

public:

class TQuestionDialog	**TQuestionDialog**(char* *name*)
class TQuestionDialog	**TQuestionDialog**(class TQuestionDialog&)
void	**~TQuestionDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TQuestionDialog

This class provides an interface to question dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TRadioBox

class **TRadioBox** : public TCheckBox

protected:

class TRadioBox	**TRadioBox**( )

public:

class TRadioBox	**TRadioBox**(char* *name,* Widget *parent*)
class TRadioBox	**TRadioBox**(class TRadioBox&)
void	**~TRadioBox**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TRadioBox

This class is a subclass of TCheckBox that enforces radio behavior
(one of many) on it’s toggle buttons.
```

const char* **ClassName**( )

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TRadioSubMenu

class **TRadioSubMenu** : public TSubMenu

protected:

class TRadioSubMenu	**TRadioSubMenu**( )

public:

class TRadioSubMenu	**TRadioSubMenu**(char* *name*, MenuDesc_t* *menuDesc* = *NULL*, XtPointer *defaultClientData* = *NULL*)
class TRadioSubMenu	**TRadioSubMenu**(class TRadioSubMenu&)
void	**~TRadioSubMenu**( )
class TClass*	**Class**( )
const char*	**ClassName**( )
Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
EMenuItemType	**MenuType**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TRadioSubMenu

This class enforces radio-like behavior among menu items in
different hierarchies.
```

Widget **Instantiate**(Widget *parent*)

Author: Fons Rademakers 02/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TRandom

class **TRandom** : public TNamed

public:

class TRandom	**TRandom**( )
class TRandom	**TRandom**(class TRandom&)
void	**~TRandom**( )
class TClass*	**Class**( )
virtual Float_t	**Gaus**(Float_t *mean = 0,* Float_t *sigma = 1*)
virtual UInt_t	**GetSeed**( )
virtual class TClass*	**IsA**( )
virtual void	**Rannor**(Float_t& *a,* Float_t& *b*)
virtual Float_t	**Rndm**(Int_t *i = 0*)
virtual void	**SetSeed**(UInt_t *seed = 12345*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

UInt_t	**fSeed**	*Random number generator seed*

Class Description

```
TRandom

Simple prototype random number generator class.
```

TRandom( ): **TNamed**("Random", "Default Random number generator")

```
default constructor
```

~TRandom( )

```
default destructor
```

Float_t **Gaus**(Float_t *mean,* Float_t *sigma*)

```
Return a number distributed following a gaussian with mean and sigma
```

void **Rannor**(Float_t &*a,* Float_t &*b*)

```
Return 2 numbers distributed following a gaussian with mean=0 and sigma=1
```

Float_t **Rndm**(Int_t *iseed*)

```
Return a random number in the interval [0,1].
```

Author: René Brun 15/12/95
Last update: 1.00/06 23/03/97 11.01.57 by René Brun

TRealData

class **TRealData** : public TObject

public:

class TRealData	**TRealData**( )
class TRealData	**TRealData**(char* *name,* Int_t *offset,* class TDataMember* *datamember*)
class TRealData	**TRealData**(class TRealData&)
void	**~TRealData**( )
class TClass*	**Class**( )
class TDataMember*	**GetDataMember**( )
virtual const Text_t*	**GetName**( )
Int_t	**GetThisOffset**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**WriteRealData**(void* *pointer,* char*& *buffer*)

Data Members

private:

class TDataMember*	**fDataMember**	*pointer to data member descriptor*
Int_t	**fThisOffset**	*offset with the THIS object pointer*
class TString	**fName**	*Concatenated names of this realdata*

Class Description

```
The TRealData class manages the effective list of all data members
for a given class. For example for an object of class TLine that inherits
from TObject and TAttLine, the TRealData object for a line contains the
complete list of all data members of the 3 classes.

The list of TRealData members in TClass is built when functions like
object.Inspect or object.DrawClass are called.
```

TRealData( ) : **TObject**( )

```
RealData default constructor
```

TRealData(const char **name,* Int_t *offset,* TDataMember **datamember*): **TObject**( )

```
Constructor to define one persistent data member
datamember is the pointer to the data member descriptor.
```

~TRealData( )

```
RealData default destructor
```

void **WriteRealData**(void **pointer,* char *&*buffer*)

```
Write one persistent data member on output buffer
pointer points to the current persistent data member
```

Author: René Brun 05/03/95
Last update: 1.00/07 04/04/97 08.45.11 by René Brun

TRepeatButton

class **TRepeatButton** : public TComponent

private:

void	**BeginRepeat**(*XtPointer*)
void	**BeginRepeatCallback**(*Widget, XtPointer, XtPointer*)
void	**EndRepeat**( )
void	**EndRepeatCallback**(*Widget, XtPointer, XtPointer*)
void	**Repeat**( )
void	**RepeatTimeOut**(*XtPointer,* XtIntervalId*)

protected:

class TRepeatButton	**TRepeatButton**( )

public:

class TRepeatButton	**TRepeatButton**(char* *name,* Widget *parent,* ERepeatButtonType *type*)
class TRepeatButton	**TRepeatButton**(class TRepeatButton&)
void	**~TRepeatButton**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual class TClass*	**IsA**( )
void	**SetParameters**(long *initial,* long *repeat*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
ERepeatButtonType	**Type**( )

Data Members

private:

static String	**fgDefaultResources**
static const char *const	**fgButtonCallback**
long	**fInitialDelay**
long	**fRepeatDelay**
XtIntervalId	**fRepeatId**
XmAnyCallbackStruct	**fMyCallbackInfo**
ERepeatButtonType	**fType**

Class Description

```
TRepeatButton

This class creates buttons that repeatedly call callbacks once
armed.
```

const char* **ClassName**( )

Author: Fons Rademakers 02/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TRint

class **TRint** : public TApplication

private:

char*	**GetPrompt**( )
void	**PrintLogo**( )

public:

class TRint	**TRint**(char* *appClassName,* int* *argc,* char** *argv,* void* *options = NULL,* int *numOptions = 0*)
class TRint	**TRint**(class TRint&)
void	**~TRint**( )
class TClass*	**Class**( )
virtual void	**HandleTermInput**( )
void	**Interrupt**( )
virtual class TClass*	**IsA**( )
virtual void	**Run**(Bool_t *retrn = kFALSE*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Terminate**(int *status*)

Data Members

private:

Int_t	**fNcmd**	*Command history number*
char	**fPrompt**	*Interpreter prompt*
Bool_t	**fInterrupt**	*If true macro execution will be stopped*

Class Description

```
Rint

Rint is the ROOT Interactive Interface. It allows interactive access
to the ROOT system via the CINT C/C++ interpreter.
```

TRint(char **appClassName,* int **argc,* char ***argv,* void **options,* int *numOptions*): **TApplication**(*appClassName, argc, argv, options, numOptions*)

```
Create an application environment. The TRint environment provides an
interface to the WM manager functionality and eventloop via inheritance
of TApplication and in addition provides interactive access to
the CINT C++ interpreter via the command line.
```

void **Run**(Bool_t *retrn*)

```
Main application eventloop. First process files given on the command
line and then go into the main application event loop.
```

void **PrintLogo**( )

```
Print the ROOT logo on standard output.
```

char* **GetPrompt**( )

```
Get prompt from interpreter. Either "root [n]" or "end with '}'".
```

void **HandleTermInput**( )

```
Handle input coming from terminal.
```

void **Terminate**(int *status*)

```
Terminate the application. Reset the terminal to sane mode and call
the logoff macro defined via Rint.Logoff environment variable.
```

Author: René Brun 17/02/95

Last update: 1.03/05 10/10/97 12.25.04 by Fons Rademakers

TROOT

class **TROOT** : public TDirectory

protected:

void	**InitSystem**( )
void*	**operator new**(size_t *l*)

public:

void	**~TROOT**( )
void	**Browse**(class TBrowser* *b*)
void	**BrowseAllTypes**(Bool_t *br = kTRUE*)
Bool_t	**BrowseOnlyFileTypes**( )
class TClass*	**Class**( )
void	**DisplayColorTable**( )
class TObject*	**FindObject**(Text_t* *name,* void*& *where*)
class TObject*	**FindObject**(Text_t* *name*)
const Text_t*	**FindObjectClassName**(Text_t* *name*)
void	**ForceStyle**(Bool_t *force = kTRUE*)
Bool_t	**FromPopUp**( )
class TApplication*	**GetApplication**( )
class TClass*	**GetClass**(Text_t* *name,* Bool_t *load = kTRUE*)
class TColor*	**GetColor**(Int_t *color*)
Int_t	**GetCount**(Int_t *i*)
Int_t*	**GetCounter**( )
const Text_t*	**GetDefCanvasName**( )
Bool_t	**GetEditHistograms**( )
Int_t	**GetEditorMode**( )
class TFile*	**GetFile**( )
class TFile*	**GetFile**(Text_t* *name*)
class TFileType*	**GetFileType**(char* *filename*)
Bool_t	**GetForceStyle**( )
class TF1*	**GetFunction**(Text_t* *name*)
class TGeometry*	**GetGeometry**(Text_t* *name*)
class TGlobal*	**GetGlobal**(class TObject* *obj,* Bool_t *load = kFALSE*)
class TGlobal*	**GetGlobal**(Text_t* *name,* Bool_t *load = kFALSE*)
class TFunction*	**GetGlobalFunction**(Text_t* *name,* Text_t* *params = 0,* Bool_t *load = kFALSE*)
class TFunction*	**GetGlobalFunctionWithPrototype**(Text_t* *name,* Text_t* *proto = 0,* Bool_t *load = kFALSE*)
class TList*	**GetListOfBrowsables**( )
class TSeqCollection*	**GetListOfBrowsers**( )
class TSeqCollection*	**GetListOfCanvases**( )
class TSeqCollection*	**GetListOfClasses**( )
class TSeqCollection*	**GetListOfColors**( )
class TSeqCollection*	**GetListOfFiles**( )
class TList*	**GetListOfFileTypes**( )
class TSeqCollection*	**GetListOfFunctions**( )
class TSeqCollection*	**GetListOfGeometries**( )
class TSeqCollection*	**GetListOfGlobalFunctions**(Bool_t *load = kFALSE*)
class TSeqCollection*	**GetListOfGlobals**(Bool_t *load = kFALSE*)
class TSeqCollection*	**GetListOfMappedFiles**( )
class TSeqCollection*	**GetListOfSockets**( )
class TSeqCollection*	**GetListOfSpecials**( )
class TSeqCollection*	**GetListOfStyles**( )
class TSeqCollection*	**GetListOfTypes**( )
const char*	**GetMacroPath**( )
VoidFuncPtr_t	**GetMakeDefCanvas**( )

Int_t	**GetNclasses**( )
Int_t	**GetNtypes**( )
class TVirtualPad*	**GetSelectedPad**( )
class TObject*	**GetSelectedPrimitive**( )
class TStyle*	**GetStyle**(Text_t* *name*)
class TDataType*	**GetType**(Text_t* *name*)
const Text_t*	**GetVersion**( )
Int_t	**GetVersionDate**( )
Int_t	**GetVersionInt**( )
Int_t	**GetVersionTime**( )
void	**Idle**(UInt_t *idleTimeInSec,* Text_t* *command = 0*)
Bool_t	**Initialized**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsBatch**( )
Bool_t	**IsFolder**( )
Bool_t	**IsLineProcessing**( )
void	**LoadMacro**(Text_t* *filename*)
void	**ls**(Option_t* *option*)
Int_t	**Macro**(Text_t* *filename*)
Bool_t	**MustClean**( )
void	**ProcessLine**(Text_t* *line*)
Int_t	**ProcessLineFast**(Text_t* *line*)
void	**Proof**(Text_t* *cluster = proof*)
Bool_t	**ReadingBasket**( )
void	**Reset**(Option_t* *option*)
void	**SaveContext**( )
void	**SetApplication**(class TApplication* *app*)
void	**SetBatch**(Bool_t *batch = kTRUE*)
void	**SetCount**(Int_t *i = -1,* Int_t *value = 0*)
void	**SetDefCanvasName**(Text_t* *name = c1*)
void	**SetEditHistograms**(Bool_t *flag = kTRUE*)
void	**SetEditorMode**(Text_t* *mode*)
void	**SetFileType**(Text_t* *ftype,* Text_t* *classname*)
void	**SetFromPopUp**(Bool_t *flag = kTRUE*)
void	**SetLineHasBeenProcessed**( )
void	**SetLineIsProcessing**( )
void	**SetMakeDefCanvas**(VoidFuncPtr_t *makecanvas*)
void	**SetMustClean**(Bool_t *flag = kTRUE*)
void	**SetReadingBasket**(Bool_t *flag = kTRUE*)
void	**SetSelectedPad**(class TVirtualPad* *pad*)
void	**SetSelectedPrimitive**(class TObject* *obj*)
void	**SetStyle**(Text_t* *stylename = Default*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**Time**(Int_t *casetime = 1*)
Int_t	**Timer**( )

Data Members

private:

static Bool_t	**fgRootInit**	*Singleton initialization flag*
Int_t	**fLineIsProcessing**	*To synchronize multi-threads*

protected:

class TString	**fVersion**	*ROOT version (from CMZ VERSQQ*) ex 0.05/01
Int_t	**fVersionInt**	*ROOT version in integer format (501*)
Int_t	**fVersionDate**	*Date of ROOT version (ex 951226*)
Int_t	**fVersionTime**	*Time of ROOT version (ex 1152*)
Int_t	**fTimer**	*Timer flag*
class TApplication*	**fApplication**	*Pointer to current application*

class TInterpreter*	**fInterpreter**	*Command interpreter*
class TFile*	**fCurrentFile**	*Current file*
class TDirectory*	**fCurrentDirectory**	*Current directory*
class TVirtualPad*	**fCurrentCanvas**	*Current graphics canvas*
class TVirtualPad*	**fCurrentPad**	*Current graphics pad*
class TStyle*	**fCurrentStyle**	*Current graphics style*
void*	**fCurrentFunction**	*Current function*
Bool_t	**fBatch**	*True if session without graphics*
Bool_t	**fEditHistograms**	*True if histograms can be edited with the mouse*
Bool_t	**fFromPopUp**	*True if command executed from a popup menu*
Bool_t	**fMustClean**	*True if object destructor scans canvases*
Bool_t	**fReadingBasket**	*True while reading a basket buffer*
Bool_t	**fBrowseOnlyFileTypes**	*True/false option set by browser/canvas menu*
Bool_t	**fForceStyle**	*Force setting of current style when reading objects*
VoidFuncPtr_t	**fMakeDefCanvas**	*Pointer to default canvas constructor*
Int_t	**fEditorMode**	*Current Editor mode*
Int_t*	**fCounter**	*Array of counters*
class TObject*	**fPrimitive**	*Currently selected primitive*
class TVirtualPad*	**fSelectPad**	*Currently selected pad*
class TSeqCollection*	**fClasses**	*List of classes definition*
class TSeqCollection*	**fTypes**	*List of data types definition*
class TSeqCollection*	**fGlobals**	*List of global variables*
class TSeqCollection*	**fGlobalFunctions**	*List of global functions*
class TSeqCollection*	**fFiles**	*List of files*
class TSeqCollection*	**fMappedFiles**	*List of memory mapped files*
class TSeqCollection*	**fSockets**	*List of network sockets*
class TSeqCollection*	**fCanvases**	*List of canvases*
class TSeqCollection*	**fStyles**	*List of styles*
class TSeqCollection*	**fFunctions**	*List of analytic functions*
class TSeqCollection*	**fProcesses**	*List of connected processes*
class TSeqCollection*	**fColors**	*List of colors*
class TSeqCollection*	**fGeometries**	*List of geometries*
class TSeqCollection*	**fBrowsers**	*List of browsers*
class TSeqCollection*	**fSpecials**	*List of special objects*
class TList*	**fBrowsables**	*List of browsables*
class TList*	**fFileTypes**	*List of special classes associated to file types*
class TString	**fDefCanvasName**	*Name of default canvas*

Class Description

```
R O O T top level object description
 The TROOT object is the entry point to the system.
 The single instance of TROOT is accessible via the global gROOT.
 Using the gROOT pointer one has access to basically every object
 created in a ROOT based program. The TROOT object is essentially a
 "dispatcher" with several lists pointing to the ROOT main objects.

 The following lists are accessible from gROOT object:
    gROOT->GetListOfClasses
    gROOT->GetListOfColors
    gROOT->GetListOfTypes
    gROOT->GetListOfGlobals
    gROOT->GetListOfGlobalFunctions
    gROOT->GetListOfFiles
    gROOT->GetListOfMappedFiles
    gROOT->GetListOfSockets
    gROOT->GetListOfCanvases
    gROOT->GetListOfStyles
    gROOT->GetListOfFunctions
    gROOT->GetListOfSpecials (for example graphical cuts)
```

```
    gROOT->GetListOfGeometries
    gROOT->GetListOfBrowsers

The TROOT class provides also many useful services:
  - Get pointer to an object in any of the lists above
  - Time utilities TROOT::Time

The ROOT object must be created as a static object. An example
of a main program creating an interactive version is shown below:

---------------------Example of a main program--------------------------------

    #include "TROOT.h"
    #include "TRint.h"

    extern void  InitGui();

    int Error;

    VoidFuncPtr_t dictfuncs[] = { InitGui, 0 };

    TROOT root("Rint","The ROOT Interactive Interface", dictfuncs);

    int main(int argc, char **argv)
    {
       TRint *theApp = new TRint("ROOT example", &argc, argv, NULL, 0);

       // Init Intrinsics, build all windows, and enter event loop
       theApp->Run();

       return(0);
    }
-----------------------End of Main program-------------------------------
```

~**TROOT**( )

```
ROOT Default Destructor
```

void **DisplayColorTable**( )

```
Display Color Table in the pad
```

TObject* **FindObject**(Text_t **name,* void *&*where*)

```
Returns address of a ROOT object if it exists
This function looks in the following order in the ROOT lists:
  - List of files
  - List of memory mapped files
  - List of functions
  - List of geometries
  - List of canvases
  - List of styles
  - List of specials
  - List of materials in current geometry
  - List of shapes in current geometry
  - List of matrices in current geometry
  - List of Nodes in current geometry
  - Current Directory in memory
  - Current Directory on file
```

const Text_t* **FindObjectClassName**(Text_t **name*) const

```
Returns class name of a ROOT object including CINT globals
```

TClass* **GetClass**(const Text_t **name1,* Bool_t *load*)

```
Return pointer to class with name
```

TColor* **GetColor**(Int_t *color*)
```
Return address of color with index color
```

TDataType* **GetType**(const Text_t **name*)
```
Return pointer to type with name
```

TFile* **GetFile**(const Text_t **name*)
```
Return pointer to file with name
```

TFileType* **GetFileType**(const Text_t **filename*)
```
Return filetype of filename if type is known
```

TStyle* **GetStyle**(const Text_t **name*)
```
Return pointer to style with name
```

TF1* **GetFunction**(const Text_t **name*)
```
Return pointer to function with name
```

TGlobal* **GetGlobal**(const Text_t **name,* Bool_t *load*)
```
Return pointer to global variable by name. If load is true force
reading of all currently defined globals from CINT (more expensive).
```

TGlobal* **GetGlobal**(TObject **addr,* Bool_t *load*)
```
Return pointer to global variable with address addr. If load is true
force reading of all currently defined globals from CINT (more
expensive).
```

TFunction* **GetGlobalFunction**(const Text_t **function,* const Text_t **params,*Bool_t *load*)
```
Return pointer to global function by name. If params != 0
it will also resolve overloading. If load is true force reading
of all currently defined global functions from CINT (more expensive).
The param string must be of the form: "3189,\"aap\",1.3".
```

TFunction* **GetGlobalFunctionWithPrototype**(const Text_t **function,*const Text_t *proto, Bool_t *load*)
```
Return pointer to global function by name. If proto != 0
it will also resolve overloading. If load is true force reading
of all currently defined global functions from CINT (more expensive).
The proto string must be of the form: "int, char*, float".
```

TGeometry* **GetGeometry**(const Text_t **name*)
```
Return pointer to Geometry with name
```

TSeqCollection* **GetListOfGlobals**(Bool_t *load*)
```
Return list containing the TGlobals currently defined.
Since globals are created and deleted during execution of the
program, we need to update the list of globals every time we
execute this method. However, when calling this function in
a (tight) loop where no interpreter symbols will be created
you can set load=kFALSE (default).
```

TSeqCollection* **GetListOfGlobalFunctions**(Bool_t *load*)
```
Return list containing the TFunctions currently defined.
Since functions are created and deleted during execution of the
program, we need to update the list of functions every time we
execute this method. However, when calling this function in
a (tight) loop where no interpreter symbols will be created
you can set load=kFALSE (default).
```

TSeqCollection* **GetListOfTypes**( )
```
Return list containing all TDataTypes (typedefs) currently defined.
```

void **Idle**(UInt_t *idleTimeInSec,* const Text_t **command*)
```
Execute command when system has been idle for idleTimeInSec seconds.
```

void **ls**(Option_t **option*)

```
To list all objects of the application
Loop on all objects created in the ROOT linked lists
objects may be files and windows or any other object directly
attached to the ROOT linked list.
```

void **LoadMacro**(const Text_t **filename*)

```
Load a macro in the interpreter's memory. Equivalent to the command line
command ".L filename".
```

Int_t **Macro**(const Text_t **filename*)

```
Execute a macro in the interpreter. Equivalent to the command line
command ".x filename".
```

void **ProcessLine**(const Text_t **line*)

```
Process interpreter command via TApplication::ProcessLine().
```

Int_t **ProcessLineFast**(const Text_t **line*)

```
Process interpreter command directly via CINT interpreter.
Only executable statements are allowed (no variable declarations),
In all other cases use TROOT::ProcessLine().
```

void **Proof**(const Text_t **cluster*)

```
Start PROOF session on a specific cluster (default is "proof").
The cluster configuration is defined either in the file ."cluster".conf,
or $HOME/."cluster".conf or /usr/proof/etc/"cluster".conf.
The TProof object can be accessed via the gProof global. Creating a
new TProof object will delete the current one.
```

void **Reset**(Option_t **option*)

```
Delete all global interpreter objects created since the last call to Reset

If option="a" is set reset to startup context (i.e. unload also
all loaded files, classes, structs, typedefs, etc.).

This function is typically used at the beginning (or end) of a macro
to clean the environment.
```

void **SaveContext**( )

```
Save the current interpreter context.
```

void **SetEditorMode**(const Text_t **mode*)

```
Set editor mode
```

void **SetCount**(Int_t *icount,* Int_t *value*)

```
Set counter element icount to zero (or all elements if icount <0)
```

void **SetFileType**(const Text_t **ftype,* const Text_t **classname*)

```
Enter a new file type and its class in the list of file types
```

void **SetMakeDefCanvas**(VoidFuncPtr_t *makecanvas*)

```
Set address of default make canvas function
This address is by default initialized to 0 by the ROOT constructor.
It is set by InitG for the interactive version of ROOT.
```

void **SetStyle**(const Text_t **stylename*)

```
Change current style to style with name stylename
```

const char* **GetMacroPath**( )

```
Get macro search path. Static utility function.
```

Author: René Brun 08/12/94
Last update: 1.03/05 13/10/97 16.24.50 by René Brun

TRotMatrix

class **TRotMatrix** : public TNamed

private:

virtual void	**SetReflection**( )

public:

class TRotMatrix	**TRotMatrix**(Text_t* *name,* Text_t* *title,* Double_t *theta,* Double_t *phi,* Double_t *psi*)
class TRotMatrix	**TRotMatrix**( )
class TRotMatrix	**TRotMatrix**(Text_t* *name,* Text_t* *title,* Double_t *theta1,* Double_t *phi1,* Double_t *theta2,* Double_t *phi2,* Double_t *theta3,* Double_t *phi3*)
class TRotMatrix	**TRotMatrix**(class TRotMatrix&)
void	**~TRotMatrix**( )
class TClass*	**Class**( )
virtual Double_t	**Determinant**( )
virtual Double_t*	**GetGLMatrix**(Double_t* *rGLMatrix*)
virtual Double_t*	**GetMatrix**( )
virtual Int_t	**GetNumber**( )
virtual Double_t	**GetPhi**( )
virtual Double_t	**GetPsi**( )
virtual Double_t	**GetTheta**( )
virtual Int_t	**GetType**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsReflection**( )
virtual void	**SetName**(Text_t* *name*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fNumber**	*Rotation matrix number*
Int_t	**fType**	*Type of matrix (0=identity, 1=reflexion, 2=otherwise*)
Double_t	**fTheta**	*theta angle*
Double_t	**fPhi**	*phi angle*
Double_t	**fPsi**	*psi angle*
Double_t	**fMatrix**	*Rotation matrix*

Class Description

```
Manages a detector rotation matrix. See class TGeometry.
```

TRotMatrix( )

```
RotMatrix default constructor
```

TRotMatrix(Text_t **name,* Text_t **title,* Double_t *theta,* Double_t *phi,* Double_t *psi*):**TNamed**(*name, title*)

```
RotMatrix normal constructor
```

TRotMatrix(Text_t **name,* Text_t **title,* Double_t *theta1,* Double_t phi1, Double_t theta2, Double_t phi2, Double_t theta3, Double_t *phi3*):**TNamed**(*name, title*)

```
RotMatrix normal constructor defined a la GEANT
```

~TRotMatrix( )

```
RotMatrix default destructor
```

Double_t **Determinant**( )

```
Determinant() returns the value of the determiant of this matrix
```

Double_t* **GetGLMatrix**(Double_t **rGLMatrix*)

```
Convert this matrix to the OpenGL [4x4]

[  fMatrix[0]   fMatrix[1]   fMatrix[2]    0  ]
[  fMatrix[3]   fMatrix[4]   fMatrix[5]    0  ]
[  fMatrix[6]   fMatrix[7]   fMatrix[8]    0  ]
[     0              0           0         1  ]

Input:
-----
Double_t *rGLMatrix - pointer to Double_t 4x4 buffer array

Return:
------
Double_t pointer to the input buffer
```

void **SetReflection**( )

```
SetReflection() -  checks whether the determinant of this
                  matrix defines the reflection transformation
                  and set the "reflection" flag if any
```

Author: René Brun 14/09/95
Last update: 1.01/05 06/06/97 10.50.27 by Valery Fine(fine@vxcern.cern.ch)

TSelectionDialog

class **TSelectionDialog** : public TDialogManager

protected:

class TSelectionDialog	**TSelectionDialog**( )

public:

class TSelectionDialog	**TSelectionDialog**(class TSelectionDialog&)
class TSelectionDialog	**TSelectionDialog**(char* *name*)
void	**~TSelectionDialog**( )
void	**AddItem**(char* *item*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
int	**Index**(XmString *item*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

XmString*	**fItems**
int	**fNumberOfItems**
int	**fMaxNumberOfItems**

Class Description

```
TSelectionDialog

This class handles selection dialogs.
```

int **Index**(*XmString*)

```
if (fCachedDialog)
return XmListItemPos(XmSelectionBoxGetChild(fDialogBox, XmDIALOG_LIST),
                     item);
else
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TSelector

class **TSelector** : public TNamed

public:

class TSelector	**TSelector**(Text_t* *name,* Text_t* *title*)
class TSelector	**TSelector**(class TSelector&)
class TSelector	**TSelector**( )
void	**~TSelector**( )
virtual void	**BeginFile**( )
class TClass*	**Class**( )
virtual void	**EndFile**( )
virtual void	**Execute**(class TTree* *tree,* Int_t *event*)
virtual void	**Finish**(Option_t* *option*)
virtual void	**Init**(class TTree* *tree,* Option_t* *option*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Start**(Option_t* *option*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TTree*	**fTree**	*Pointer to current TTree*

Class Description

```
A TSelector object is used by the TTree::Draw, TTree::Scan,
TTree::Loop to navigate in a TTree and make selections.

The following members functions are called by the TTree functions.
 Init:      to initialize the selector
 Start:     called everytime a loop on the tree starts.
            a convenient place to create your histograms.
 Finish:    called at the end of a loop on a TTree.
            a convenient place to draw/fit your histograms.
 BeginFile: called when a TChain is processed at the begining of
            of a new file.
 EndFile:   called when a TChain is processed at the end of a file.//
 Execute:   called for each selected event.
```

TSelector( ): **TNamed**( )

```
Default constructor for a Selector
```

TSelector(const Text_t **name,* const Text_t **title*):**TNamed**(*name, title*)

```
Create a Selector
```

~TSelector( )

```
Default destructor for a selector
```

void **BeginFile**( )

```
called by TTree::Loop when a new file begins in a TChain
```

void **EndFile**( )

```
called by TTree::Loop when a file ends in a TChain
```

void **Execute**(TTree **tree,* Int_t *event*)

```
process event number event
```

void **Finish**(Option_t **option*)

```
called at the end of TTree::Loop
```

void **Init**(TTree **tree,* Option_t **option*)

```
called to initialize/reinitialize the selector.
```

void **Start**(Option_t **option*)

```
called at the begining of TTree::Loop
```

Author: René Brun 05/02/97
Last update: 1.00/06 23/03/97 11.38.38 by René Brun

TSeqCollection

class **TSeqCollection** : public TCollection

protected:

virtual void	**Changed**( )

public:

void	**~TSeqCollection**( )
virtual void	**Add**(class TObject* *obj*)
virtual void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
virtual void	**AddAt**(class TObject* *obj,* Int_t *idx*)
virtual void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
virtual void	**AddFirst**(class TObject* *obj*)
virtual void	**AddLast**(class TObject* *obj*)
virtual class TObject*	**After**(class TObject* *obj*)
virtual class TObject*	**At**(Int_t *idx*)
virtual class TObject*	**Before**(class TObject* *obj*)
class TClass*	**Class**( )
virtual class TObject*	**First**( )
virtual Int_t	**IndexOf**(class TObject* *obj*)
virtual class TClass*	**IsA**( )
Bool_t	**IsSorted**( )
virtual class TObject*	**Last**( )
Int_t	**LastIndex**( )
Int_t	**ObjCompare**(class TObject* *a,* class TObject* *b*)
void	**QSort**(class TObject** *a,* Int_t *first,* Int_t *last*)
void	**QSort**(class TObject** *a,* class TObject** *b,* Int_t *first,* Int_t *last*)
virtual void	**RemoveAfter**(class TObject* *after*)
virtual void	**RemoveAt**(Int_t *idx*)
virtual void	**RemoveBefore**(class TObject* *before*)
virtual void	**RemoveFirst**( )
virtual void	**RemoveLast**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Bool_t	**fSorted**

See Also: TBtree, TList, TObjArray, TOrdCollection

Class Description

```
TSeqCollection

Sequenceable collection abstract base class. TSeqCollection's have
an ordering relation, i.e. there is a first and last element.
```

Int_t **IndexOf**(TObject **obj*) const

```
Return index of object in collection. Returns -1 when object not found.
Uses member IsEqual() to find object.
```

Int_t **ObjCompare**(TObject **a,* TObject **b*)

```
Compare to objects in the collection. Use member Compare() of object a.
```

void **QSort**(TObject ***a,* Int_t *first,* Int_t *last*)

```
Sort array of TObject pointers using a quicksort algorithm.
Uses ObjCompare() to compare objects.
```

void **QSort**(TObject ***a,* TObject ***b,* Int_t *first,* Int_t *last*)

```
Sort array a of TObject pointers using a quicksort algorithm.
Array b will be sorted just like a (a determines the sort).
Uses ObjCompare() to compare objects.
```

Author: Fons Rademakers 04/08/95
Last update: 1.03/05 02/10/97 13.28.42 by Fons Rademakers

TServerSocket

class **TServerSocket** : public TNamed

private:

class TServerSocket	**TServerSocket**( )
class TServerSocket	**TServerSocket**(class TServerSocket&)
void	***operator=***(class TServerSocket&)

public:

class TServerSocket	**TServerSocket**(Int_t *port,* Bool_t *reuse* = *kFALSE,* Int_t *backlog* = *kDefaultBacklog*)
class TServerSocket	**TServerSocket**(char* *service,* Bool_t *reuse* = *kFALSE,* Int_t *backlog* = *kDefaultBacklog*)
void	**~TServerSocket**( )
virtual class TSocket*	**Accept**( )
class TClass*	**Class**( )
virtual void	**Close**( )
virtual Int_t	**GetDescriptor**( )
virtual class TInetAddress	**GetLocalInetAddress**( )
virtual Int_t	**GetLocalPort**( )
virtual Int_t	**GetOption**(enum ESockOptions *opt,* Int_t& *val*)
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsValid**( )
virtual Int_t	**SetOption**(enum ESockOptions *opt,* Int_t *val*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Int_t	**fSocket**	*server socket descriptor*
class TString	**fService**	*name of service (matches port #)*
class TInetAddress	**fAddress**	*local internet address and port #*

public:

static const enum	**kDefaultBacklog**

Class Description

```
TServerSocket

This class implements server sockets. A server socket waits for
requests to come in over the network. It performs some operation
based on that request and then possibly returns a full duplex socket
to the requester. The actual work is done via the TSystem class
(either TUnixSystem, TWin32System or TMacSystem).
```

TServerSocket(const char **service,* Bool_t *reuse,* Int_t *backlog*): **TNamed**("ServerSocket", *service*)

```
Create a server socket object for a named service. Set reuse to true
to force reuse of the server socket (i.e. do not wait for the time
out to pass). Using backlog one can set the desirable queue length
for pending connections. Use IsValid() to check the validity of the
server socket. Every server socket is added to the TROOT sockets list
which will make sure that any open sockets are properly closed on
program termination.
```

TServerSocket(Int_t *port,* Bool_t *reuse,* Int_t *backlog*): **TNamed**("ServerSocket", "")

```
Create a server socket object on a specified port. Set reuse to true
to force reuse of the server socket (i.e. do not wait for the time
out to pass). Using backlog one can set the desirable queue length
for pending connections. Use IsValid() to check the validity of the
```

```
server socket. Every server socket is added to the TROOT sockets list
which will make sure that any open sockets are properly closed on
program termination.
```

TSocket* **Accept**( )

```
Accept a connection on a server socket. Returns a full-duplex
communication TSocket object. If no pending connections are
present on the queue and nonblocking mode has not been enabled
with SetOption(kNoBlock,1) the call blocks until a connection is
present. The returned socket must be deleted by the user. The socket
is also added to the TROOT sockets list which will make sure that
any open sockets are properly closed on program termination.
In case of error 0 is returned and in case non-blocking I/O is
enabled and no connections are available -1 is returned.
```

void **Close**( )

```
Close a TServerSocket. Also called via the dtor.
```

TInetAddress **GetLocalInetAddress**( )

```
Return internet address of host to which the server socket is bound,
i.e. the local host. In case of error TInetAddress::IsValid() returns
kFALSE.
```

Int_t **GetLocalPort**( )

```
Get port # to which server socket is bound. In case of error returns -1.
```

Int_t **SetOption**(ESockOptions *opt,* Int_t *val*)

```
Set server socket options. For server socket mainly used to set
option kNoBlock.
```

Int_t **GetOption**(ESockOptions *opt,* Int_t &*val*)

```
Get server socket options.
```

Author: Fons Rademakers 18/12/96
Last update: 1.01/00 30/04/97 11.35.37 by Fons Rademakers

TShape

class **TShape** : public TNamed, public TAttLine, public TAttFill

public:

class TShape	**TShape**(Text_t* *name,* Text_t* *title,* Text_t* *material*)
class TShape	**TShape**( )
class TShape	**TShape**(class TShape&)
void	**~TShape**( )
class TClass*	**Class**( )
class TMaterial*	**GetMaterial**( )
virtual Int_t	**GetNumber**( )
Int_t	**GetVisibility**( )
virtual Bool_t	**Is3D**( )
virtual class TClass*	**IsA**( )
virtual void	**PaintGLPoints**(Float_t* *vertex*)
virtual void	**PaintShape**(X3DBuffer* *buff*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetVisibility**(Int_t *vis*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

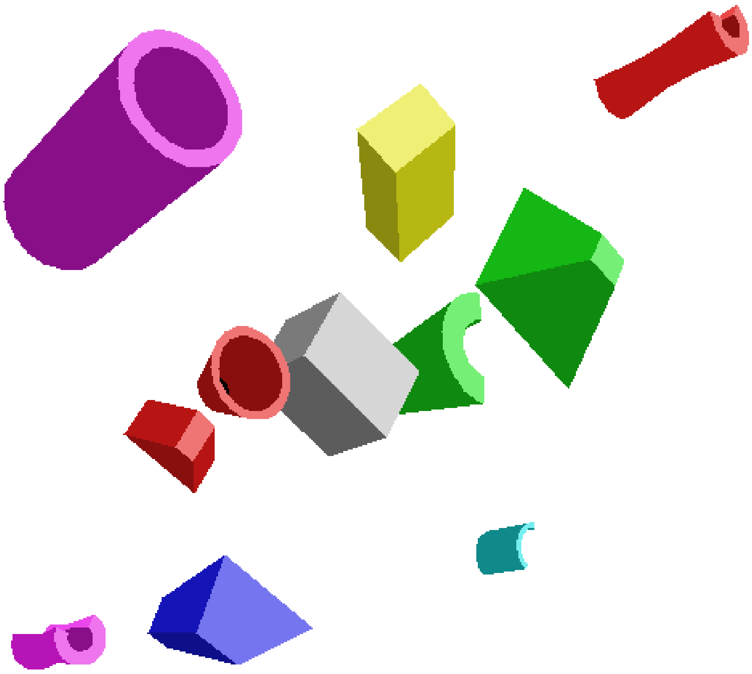
protected:

Int_t	**fNumber**	*Shape number*
Int_t	**fVisibility**	*Visibility flag*
class TMaterial*	**fMaterial**	*Pointer to material*

See Also: TBRIK, TPCON, TTUBE

Class Description

```
This is the base class for all geometry shapes.
The list of shapes currently supported correspond to the shapes
in Geant version 3:
 TBRIK,TCONE,TCONS,TGTRA,TPARA,TPCON,TPGON
,TTRAP,TTRD1,TTRD2,TTUBE and TTUBS.

The figure below shows instances of all these shapes. This figure
is generated by the ROOT 3-D viewer.
```



TShape( )

```
Shape default constructor
```

TShape(Text_t **name,* Text_t **title,* Text_t **materialname*): **TNamed**(*name, title*), **TAttLine**( ), **TAttFill**( )

```
Shape normal constructor
```

~**TShape**( )

```
Shape default destructor
```

void **PaintShape**(X3DBuffer **buff*)

```
Paint 3-D shape in current pad with its current attributes
```

Author: Nenad Buncic 17/09/95
Last update: 1.03/07 04/11/97 17.10.42 by René Brun

TSignalHandler

class **TSignalHandler** : public TSysEvtHandler

protected:

class TSignalHandler	**TSignalHandler**( )

public:

class TSignalHandler	**TSignalHandler**(enum ESignals *sig*, Bool_t *sync* = *kTRUE*)
class TSignalHandler	**TSignalHandler**(class TSignalHandler&)
void	**~TSignalHandler**( )
class TClass*	**Class**( )
void	**Delay**( )
enum ESignals	**GetSignal**( )
void	**HandleDelayedSignal**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsAsync**( )
Bool_t	**IsSync**( )
virtual Bool_t	**Notify**( )
virtual void	**Remove**( )
void	**SetSignal**(enum ESignals *sig*)
virtual void	**ShowMembers**(class TMemberInspector& *insp*, char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

enum ESignals	**fSignal**	*Signal to be handled*
Bool_t	**fSync**	*Synchronous or a-synchronous signal*
Int_t	**fDelay**	*Delay handling of signal (use fDelay in Notify( ))*

Class Description

```
TSysEvtHandler

Abstract base class for handling system events.
```

TSignalHandler(ESignals *sig*, Bool_t *sync*)

```
Create signal event handler.
```

Bool_t **Notify**( )

```
Notify when signal occurs.
```

void **Remove**( )

```
Remove signal handler from system signal handler list.
```

Author: Fons Rademakers 16/09/95
Last update: 1.03/02 25/08/97 16.23.18 by Fons Rademakers

TSimpleWindow

class **TSimpleWindow** : public TComponent, public TBaseWindow

protected:

class TSimpleWindow	**TSimpleWindow**( )

public:

class TSimpleWindow	**TSimpleWindow**(char* *name,* ArgList *args,* Cardinal *argCount*)
class TSimpleWindow	**TSimpleWindow**(class TSimpleWindow&)
class TSimpleWindow	**TSimpleWindow**(char* *name*)
void	**~TSimpleWindow**( )
void	**AddView**(Widget *w*)
void	**AddView**(class TComponent* *component*)
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual void	**Hide**( )
virtual class TClass*	**IsA**( )
void	**RemoveView**( )
virtual Widget	**SetUpInterface**(Widget *parent*)
virtual void	**Show**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual Widget	**ViewWidget**( )

Data Members

protected:

Widget	**fCurrentView**

See Also: TMenuWindow

Class Description

```
TSimpleWindow

This class defines basic window behaviour.
```

Widget **ViewWidget**( ) const

Author: Fons Rademakers 30/03/95
Last update: 1.00/09 22/08/97 11.38.49 by René Brun

TSlave

class **TSlave** : public TObject

private:

class TSlave	**TSlave**( )
class TSlave	**TSlave**(class TSlave&)
class TSlave	**TSlave**(char* *host,* Int_t *ord,* Int_t *perf,* class TProof* *proof*)
void	***operator=***(class TSlave&)

public:

void	**~TSlave**( )
class TClass*	**Class**( )
void	**Close**( )
Int_t	**Compare**(class TObject* *obj*)
UInt_t	**GetBytesRead**( )
Float_t	**GetCpuTime**( )
const char*	**GetName**( )
Int_t	**GetOrdinal**( )
Int_t	**GetPerfIdx**( )
class TProof*	**GetProof**( )
Float_t	**GetRealTime**( )
class TSocket*	**GetSocket**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsSortable**( )
Bool_t	**IsValid**( )
void	**Print**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TString	**fName**	*slave's hostname*
Int_t	**fOrdinal**	*slave's ordinal number*
Int_t	**fPerfIdx**	*performance index, on a scale of 1-200, PPro200=40*
class TSocket*	**fSocket**	*socket to slave*
class TProof*	**fProof**	*proof cluster to which slave belongs*
UInt_t	**fBytesRead**	*bytes read by slave (info is obtained from slave)*
Float_t	**fRealTime**	*real time spent executing commands (info obtained from slave)*
Float_t	**fCpuTime**	*CPU time spent executing commands (info obtained from slave)*

Class Description

```
TSlave

This class describes a PROOF slave server.
It contains information like the slaves host name, ordinal number,
performance index, socket, etc. Objects of this class can only be
created via TProof member functions.
```

TSlave(const char **host,* Int_t *ord,* Int_t *perf,* TProof **proof*)

```
Create a PROOF slave object. Called via the TProof ctor.
```

~TSlave( )

```
Destroy slave.
```

void **Close**( )

```
Close slave socket.
```

Int_t **Compare**(TObject **obj*)

```
Used to sort slaves by performance index.
```

void **Print**(Option_t **option*)

```
Printf info about slave.
```

Author: Fons Rademakers 14/02/97
Last update: 1.00/07 27/03/97 11.02.06 by Fons Rademakers

TSlider

class **TSlider** : public TPad

public:

class TSlider	**TSlider**(Text_t* *name,* Text_t* *title,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color = 16,* Short_t *bordersize = 2,* Short_t *bordermode = -1*)
class TSlider	**TSlider**(class TSlider&)
class TSlider	**TSlider**( )
void	**~TSlider**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *slider*)
Float_t	**GetMaximum**( )
Float_t	**GetMinimum**( )
class TObject*	**GetObject**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetMaximum**(Float_t *max = 1*)
virtual void	**SetMinimum**(Float_t *min = 0*)
void	**SetObject**(class TObject* *obj = 0*)
virtual void	**SetRange**(Float_t *xmin = 0,* Float_t *xmax = 1*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fMinimum**	*Slider minimum value in [0, 1]*
Float_t	**fMaximum**	*Slider maximum value in [0, 1]*
class TObject*	**fObject**	*Pointer to associated object*

Class Description

```
TSlider

A TSlider object is a specialized TPad including a TSliderBox object
The TSliderBox can be moved in the pad.
Slider drawing options include the possibility to change the slider
starting and ending positions or only one of them.

The current slider position can be retrieved via the functions
  TSlider::GetMinimum and TSlider::GetMaximum
These two functions return numbers in the range [0,1].

While the slider is being moved/resized, the object ExecuteEvent
function is called.
```

TSlider( ): **TPad**( )

```
slider default constructor
```

TSlider(const Text_t **name,* const Text_t **title,* Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color,* Short_t *bordersize,* Short_t *bordermode*):**TPad**(*name, title,* 0.*1,* 0.*1,* 0.*9,* 0.*9, color, bordersize, bordermode*)

```
Slider normal constructor
x1,y1,x2,y2 are in pad user coordinates
```

~TSlider( )

```
slider default destructor
```

void **Copy**(TObject &*slider*)
```
Copy this slider to slider
```

void **Paint**(Option_t **option*)
```
Paint this slider with its current attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

void **SetRange**(Float_t *xmin,* Float_t *xmax*)
```
Set Slider range in [0,1]
```

Author: René Brun 23/11/96
Last update: 1.00/06 23/03/97 11.21.52 by René Brun

TSliderBox

class **TSliderBox** : public TWbox

public:

class TSliderBox	**TSliderBox**( )
class TSliderBox	**TSliderBox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color = 18,* Short_t *bordersize* = *2,* Short_t *bordermode* = *-1*)
class TSliderBox	**TSliderBox**(class TSliderBox&)
void	**~TSliderBox**( )
class TClass*	**Class**( )
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
void	**SetSlider**(class TSlider* *slider*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TSlider*	**fSlider**	*Pointer to slider*

Class Description

```
TSliderBox

The moving box in a TSlider
```

TSliderBox( ): **TWbox**( )

```
SliderBox default constructor
```

TSliderBox(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color,* Short_t *bordersize,* Short_t *bordermode*):**TWbox**(*x1, y1, x2, y2, color, bordersize, bordermode*)

```
SliderBox normal constructor
```

~TSliderBox( )

```
SliderBox default destructor
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Interaction with a slider
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 23/11/96
Last update: 1.01/04 20/08/97 11.33.26 by René Brun

TSocket

class **TSocket** : public TNamed

private:

class TSocket	**TSocket**( )
void	***operator=***(class TSocket& *rhs*)
void	**SetDescriptor**(Int_t *desc*)

public:

class TSocket	**TSocket**(class TInetAddress *address,* Int_t *port*)
class TSocket	**TSocket**(char* *host,* char* *service*)
class TSocket	**TSocket**(char* *host,* Int_t *port*)
class TSocket	**TSocket**(Int_t *descriptor*)
class TSocket	**TSocket**(class TInetAddress *address,* char* *service*)
class TSocket	**TSocket**(class TSocket& *s*)
void	**~TSocket**( )
class TClass*	**Class**( )
virtual void	**Close**( )
UInt_t	**GetBytesRecv**( )
UInt_t	**GetBytesSent**( )
virtual Int_t	**GetDescriptor**( )
virtual class TInetAddress	**GetInetAddress**( )
virtual class TInetAddress	**GetLocalInetAddress**( )
virtual Int_t	**GetLocalPort**( )
virtual Int_t	**GetOption**(enum ESockOptions *opt,* Int_t& *val*)
virtual Int_t	**GetPort**( )
UInt_t	**GetSocketBytesRecv**( )
UInt_t	**GetSocketBytesSent**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsValid**( )
virtual Int_t	**Recv**(char* *mess,* Int_t *max*)
virtual Int_t	**Recv**(class TMessage*& *mess*)
virtual Int_t	**Recv**(char* *mess,* Int_t *max,* Int_t& *kind*)
virtual Int_t	**RecvRaw**(void* *buffer,* Int_t *length,* enum ESendRecvOptions *opt* = *kDefault*)
virtual Int_t	**Send**(char* *mess,* Int_t *kind* = *kMESS_STRING*)
virtual Int_t	**Send**(Int_t *kind*)
virtual Int_t	**Send**(class TMessage& *mess*)
virtual Int_t	**SendObject**(class TObject* *obj,* Int_t *kind* = *kMESS_OBJECT*)
virtual Int_t	**SendRaw**(void* *buffer,* Int_t *length,* enum ESendRecvOptions *opt* = *kDefault*)
virtual Int_t	**SetOption**(enum ESockOptions *opt,* Int_t *val*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Int_t	**fSocket**	*socket descriptor*
class TString	**fService**	*name of service (matches remote port #)*
class TInetAddress	**fAddress**	*remote internet address and port #*
class TInetAddress	**fLocalAddress**	*local internet address and port #*
UInt_t	**fBytesSent**	*total bytes sent using this socket*
UInt_t	**fBytesRecv**	*total bytes received over this socket*
static UInt_t	**fgBytesSent**	*total bytes sent by all socket objects*
static UInt_t	**fgBytesRecv**	*total bytes received by all socket objects*

Class Description

```
TSocket

This class implements client sockets. A socket is an endpoint for
communication between two machines.
The actual work is done via the TSystem class (either TUnixSystem,
TWin32System or TMacSystem).
```

TSocket(TInetAddress *addr,* const char **service*): **TNamed**(addr.**GetHostName**( ), *service*)

```
Create a socket. Connect to the named service at address addr.
Returns when connection has been accepted by remote side. Use IsValid()
to check the validity of the socket. Every socket is added to the TROOT
sockets list which will make sure that any open sockets are properly
closed on program termination.
```

TSocket(TInetAddress *addr,* Int_t *port*): **TNamed**(addr.**GetHostName**( ), "")

```
Create a socket. Connect to the specified port # at address addr.
Returns when connection has been accepted by remote side. Use IsValid()
to check the validity of the socket. Every socket is added to the TROOT
sockets list which will make sure that any open sockets are properly
closed on program termination.
```

TSocket(const char **host,* const char **service*): **TNamed**(*host, service*)

```
Create a socket. Connect to named service on the remote host.
Returns when connection has been accepted by remote side. Use IsValid()
to check the validity of the socket. Every socket is added to the TROOT
sockets list which will make sure that any open sockets are properly
closed on program termination.
```

TSocket(const char **host,* Int_t *port*): **TNamed**(*host,* "")

```
Create a socket. Connect to specified port # on the remote host.
Returns when connection has been accepted by remote side. Use IsValid()
to check the validity of the socket. Every socket is added to the TROOT
sockets list which will make sure that any open sockets are properly
closed on program termination.
```

TSocket(Int_t *desc*) : **TNamed**("", "")

```
Create a socket. The socket will use descriptor desc.
```

TSocket(const TSocket *&s*)

```
TSocket copy ctor.
```

void **Close**( )

```
Close the socket. Also called via the dtor.
```

TInetAddress **GetLocalInetAddress**( )

```
Return internet address of local host to which the socket is bound.
In case of error TInetAddress::IsValid() returns kFALSE.
```

Int_t **GetLocalPort**( )

```
Return the local port # to which the socket is bound.
In case of error return -1.
```

Int_t **Send**(Int_t *kind*)

```
Send a single message op code. Use kind (opcode) is used to set the
TMessage what field. Returns the number of bytes that were sent
(always sizeof(Int_t)) and -1 in case of error. In case the kind has
been or'ed with kMESS_ACK, the call will only return after having
received an acknowledgement, making the sending process synchronous.
```

Int_t **Send**(const char **str,* Int_t *kind*)

```
Send a character string buffer. Use kind to set the TMessage what field.
Returns the number of bytes in the string str that were sent and -1 in
```

case of error. In case the kind has been or'ed with kMESS_ACK, the call will only return after having received an acknowledgement, making the sending process synchronous.

Int_t **Send**(const TMessage &*mess*)

Send a TMessage object. Returns the number of bytes in the TMessage that were sent and -1 in case of error. In case the TMessage::What has been or'ed with kMESS_ACK, the call will only return after having received an acknowledgement, making the sending process synchronous.

Int_t **SendObject**(const TObject **obj,* Int_t *kind*)

Send an object. Returns the number of bytes sent and -1 in case of error. In case the "kind" has been or'ed with kMESS_ACK, the call will only return after having received an acknowledgement, making the sending synchronous.

Int_t **SendRaw**(const void **buffer,* Int_t *length,* ESendRecvOptions *opt*)

Send a raw buffer of specified length. Using option kOob one can send OOB data. Returns the number of bytes sent or -1 in case of error.

Int_t **Recv**(char **str,* Int_t *max*)

Receive a character string message of maximum max length. The expected message must be of type kMESS_STRING. Returns length of received string or -1 in case of error.

Int_t **Recv**(char **str,* Int_t *max,* Int_t &*kind*)

Receive a character string message of maximum max length. Returns in kind the message type. Returns length of received string or -1 in case of error.

Int_t **Recv**(TMessage *&*mess*)

Receive a TMessage object. The user must delete the TMessage object. Returns length of message in bytes or -1 in case of error (mess == 0 in that case).

Int_t **RecvRaw**(void **buffer,* Int_t *length,* ESendRecvOptions *opt*)

Receive a raw buffer of specified length bytes. Using option kPeek one can peek at incoming data. Returns -1 in case of error. In case of opt == kOob: -2 means EWOULDBLOCK and -3 EINVAL.

Int_t **SetOption**(ESockOptions *opt,* Int_t *val*)

Set socket options.

Int_t **GetOption**(ESockOptions *opt,* Int_t &*val*)

Get socket options. Returns -1 in case of error.

Author: Fons Rademakers 18/12/96
Last update: 1.03/02 25/08/97 18.36.15 by Fons Rademakers

TSortedList

class **TSortedList** : public TList

public:

class TSortedList	**TSortedList**(Bool_t *order* = *kSortAscending*)
class TSortedList	**TSortedList**(class TSortedList&)
void	**~TSortedList**( )
void	**Add**(class TObject* *obj*)
void	**Add**(class TObject* *obj,* Option_t* *opt*)
void	**AddAfter**(class TObject* *after,* class TObject* *obj*)
void	**AddAt**(class TObject* *obj,* Int_t *idx*)
void	**AddBefore**(class TObject* *before,* class TObject* *obj*)
void	**AddFirst**(class TObject* *obj*)
void	**AddFirst**(class TObject* *obj,* Option_t* *opt*)
void	**AddLast**(class TObject* *obj,* Option_t* *opt*)
void	**AddLast**(class TObject* *obj*)
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsAscending**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Bool_t	**fAscending**

Class Description

```
A sorted doubly linked list. All sortable classes inheriting from
TObject can be inserted in a TSortedList.
```

void **Add**(TObject **obj*)

```
Add object in sorted list. Uses object Compare() member to find right
position.
```

void **Add**(TObject **obj,* const Option_t **opt*)

```
Add object in sorted list. Uses object Compare() member to find right
position and also store option. See TList::Add for explanation of
usage of option.
```

Author: Fons Rademakers 14/09/95
Last update: 1.00/06 23/03/97 11.05.27 by René Brun

TSPHE

class **TSPHE** : public TBRIK

public:

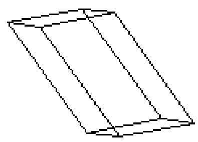
class TSPHE	**TSPHE**( )
class TSPHE	**TSPHE**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *rmin,* Float_t *rmax,* Float_t *themin,* Float_t *themax,* Float_t *phimin,* Float_t *phimax*)
class TSPHE	**TSPHE**(class TSPHE&)
void	**~TSPHE**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fRmin**	*minimum radius*
Float_t	**fRmax**	*maximum radius*
Float_t	**fThemin**	*minimum theta*
Float_t	**fThemax**	*maximum theta*
Float_t	**fPhimin**	*minimum phi*
Float_t	**fPhimax**	*maximum phi*

Class Description



```
SPHE is a Sphere. It has 9 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - rmin       minimum radius
  - rmax       maximum radius
  - themin     theta min
  - themax     theta max
  - phimin     phi min
  - phimax     phi max
```

TSPHE( )
```
SPHE shape default constructor
```

TSPHE(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rmin,* Float_t *rmax,* Float_t *themin,*Float_t *themax,* Float_t *phimin,* Float_t *phimax*) : **TBRIK**(*name, title, material, rmax, rmax,* rmax)
```
SPHE shape normal constructor
```

~TSPHE( )
```
SPHE shape default destructor
```

void **SetPoints**(Float_t **buff*)
```
Create SPHE points
```

Author: René Brun 13/06/97

Last update: 1.01/05 13/06/97 08.31.19 by René Brun

TStopwatch

class **TStopwatch** : public TObject

public:

class TStopwatch	**TStopwatch**(class TStopwatch&)
class TStopwatch	**TStopwatch**( )
void	**~TStopwatch**( )
class TClass*	**Class**( )
void	**Continue**( )
Double_t	**CpuTime**( )
Double_t	**GetCPUTime**( )
Double_t	**GetRealTime**( )
virtual class TClass*	**IsA**( )
void	**Print**(Option_t* *option*)
Double_t	**RealTime**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Start**( )
void	**Stop**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Double_t	**fStartRealTime**	*wall clock start time*
Double_t	**fStopRealTime**	*wall clock stop time*
Double_t	**fStartCpuTime**	*cpu start time*
Double_t	**fStopCpuTime**	*cpu stop time*
enum TStopwatch::EState	**fState**	*stopwatch state*

public:

static const enum TStopwatch::EState**kUndefined**
static const enum TStopwatch::EState**kStopped**
static const enum TStopwatch::EState**kRunning**

Class Description

```
TStopwatch

Stopwatch class. This class returns the real and cpu time between
the start and stop events.
```

TStopwatch( )

```
Create a stopwatch and start it.
```

void **Start**( )

```
Start the stopwatch.
```

void **Stop**( )

```
Stop the stopwatch.
```

void **Continue**( )

```
Resume a stopped stopwatch.
```

Double_t **RealTime**( )

```
Return the realtime passed between the start and stop events. If the
stopwatch was still running stop it first.
```

Double_t **CpuTime**( )

```
Return the cputime passed between the start and stop events. If the
stopwatch was still running stop it first.
```

void **Print**(Option_t **option*)

```
Print the real and cpu time passed between the start and stop events.
```

Author: Fons Rademakers 11/10/95
Last update: 1.00/06 23/03/97 11.01.58 by René Brun

TString

class **TString**

private:

void	**Clone**( )
void	**Clone**(Ssiz_t *nc*)

protected:

class TString	**TString**(char* *a1,* Ssiz_t *n1,* char* *a2,* Ssiz_t *n2*)
Ssiz_t	**AdjustCapacity**(Ssiz_t *nc*)
void	**AssertElement**(Ssiz_t *nc*)
void	**Clobber**(Ssiz_t *nc*)
void	**Cow**( )
void	**Cow**(Ssiz_t *nc*)
void	**InitChar**(char *c*)
class TStringRef*	**Pref**( )

public:

class TString	**TString**(Ssiz_t *ic*)
class TString	**TString**(class TString& *s*)
class TString	**TString**(char* *s*)
class TString	**TString**(char* *s,* Ssiz_t *n*)
class TString	**TString**( )
class TString	**TString**(char *c*)
class TString	**Copy**( )
class TString	**TString**(char *c,* Ssiz_t *s*)
class TString	**TString**(class TSubString& *sub*)
void	**~TString**( )
class TString&	**Append**(char *c,* Ssiz_t *rep = 1*)
class TString&	**Append**(char* *cs,* Ssiz_t *n*)
class TString&	**Append**(class TString& *s,* Ssiz_t *n*)
class TString&	**Append**(char* *cs*)
class TString&	**Append**(class TString& *s*)
Ssiz_t	**Capacity**( )
Ssiz_t	**Capacity**(Ssiz_t *n*)
class TClass*	**Class**( )
int	**CompareTo**(char* cs, (*unknown*) TString::ECaseCompare cmp = kExact)
int	**CompareTo**(class TString& st, (*unknown*) TString::ECaseCompare cmp = kExact)
Bool_t	**Contains**(class TString& pat, (*unknown*) TString::ECaseCompare cmp = kExact)
Bool_t	**Contains**(char* pat, (*unknown*) TString::ECaseCompare cmp = kExact)
const char*	**Data**( )
virtual void	**FillBuffer**(char*& *buffer*)
Ssiz_t	**First**(char* *cs*)
Ssiz_t	**First**(char *c*)
Ssiz_t	**GetInitialCapacity**( )
Ssiz_t	**GetMaxWaste**( )
Ssiz_t	**GetResizeIncrement**( )
unsigned int	**Hash**((*unknown*) TString::ECaseCompare cmp = kExact)
Ssiz_t	**Index**(char* pat, Ssiz_t patlen, Ssiz_t i, (*unknown*) TString::ECaseCompare cmp)
Ssiz_t	**Index**(class TString& s, Ssiz_t patlen, Ssiz_t i, (*unknown*) TString::ECaseCompare cmp)
Ssiz_t	**Index**((*unknown*) TRegexp& pat, Ssiz_t* ext, Ssiz_t i = 0)
Ssiz_t	**Index**((*unknown*) TRegexp& pat, Ssiz_t i = 0)
Ssiz_t	**Index**(char* pat, Ssiz_t i=0, (*unknown*) TString::ECaseCompare cmp=kExact)

Ssiz_t	**Index**(class TString& s, Ssiz_t i=0, (*unknown*) TString::ECaseCompare cmp=kExact)
Ssiz_t	**InitialCapacity**(Ssiz_t *ic = 15*)
class TString&	**Insert**(Ssiz_t *pos,* class TString& *s*)
class TString&	**Insert**(Ssiz_t *pos,* char* *s,* Ssiz_t *extent*)
class TString&	**Insert**(Ssiz_t *pos,* class TString& *s,* Ssiz_t *extent*)
class TString&	**Insert**(Ssiz_t *pos,* char* *s*)
virtual class TClass*	**IsA**( )
Bool_t	**IsAscii**( )
Bool_t	**IsNull**( )
Ssiz_t	**Last**(char *c*)
Ssiz_t	**Length**( )
Ssiz_t	**MaxWaste**(Ssiz_t *mw = 15*)
char*	**operator const char***( )
char	**operator( )**(Ssiz_t *n)*
class TSubString	**operator( )**(Ssiz_t *start,* Ssiz_t *len)*
class TSubString	**operator( )**((*unknown)* TRegexp& re, Ssiz_t start)
class TSubString	**operator( )**((*unknown)* TRegexp& re)
class TString&	**operator+=**(char* *s*)
class TString&	**operator+=**(class TString& *s*)
class TString&	***operator*=**(class TString& *s*)
class TString&	***operator*=**(class TSubString& *s*)
class TString&	***operator*=**(char *s*)
class TString&	***operator*=**(char* *s*)
char	**operator[]**(Ssiz_t *n*)
class TString&	**Prepend**(class TString& *s,* Ssiz_t *n*)
class TString&	**Prepend**(char *c,* Ssiz_t *rep = 1*)
class TString&	**Prepend**(char* *cs,* Ssiz_t *n*)
class TString&	**Prepend**(char* *cs*)
class TString&	**Prepend**(class TString& *s*)
virtual void	**ReadBuffer**(char*& *buffer*)
class istream&	**ReadFile**(class istream& *str*)
class istream&	**ReadLine**(class istream& *str,* Bool_t *skipWhite = kTRUE*)
class istream&	**ReadString**(class istream& *str*)
class istream&	**ReadToDelim**(class istream& *str,* char *delim* = *n*)
class istream&	**ReadToken**(class istream& *str*)
class TString&	**Remove**(Ssiz_t *pos*)
class TString&	**Remove**(Ssiz_t *pos,* Ssiz_t *n*)
class TString&	**Replace**(Ssiz_t *pos,* Ssiz_t *n,* char* *s*)
class TString&	**Replace**(Ssiz_t *pos,* Ssiz_t *n1,* class TString& *s,* Ssiz_t *n2*)
class TString&	**Replace**(Ssiz_t *pos,* Ssiz_t *n,* class TString& *s*)
class TString&	**Replace**(Ssiz_t *pos,* Ssiz_t *n,* char* *s,* Ssiz_t *ns*)
class TString&	**ReplaceAll**(char* *s1,* char* *s2*)
class TString&	**ReplaceAll**(char* *s1,* Ssiz_t *ls1,* char* *s2,* Ssiz_t *ls2*)
class TString&	**ReplaceAll**(char* *s1,* class TString& *s2*)
class TString&	**ReplaceAll**(class TString& *s1,* char* *s2*)
class TString&	**ReplaceAll**(class TString& *s1,* class TString& *s2*)
void	**Resize**(Ssiz_t *n*)
Ssiz_t	**ResizeIncrement**(Ssiz_t *ri = 16*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)
const class TSubString	**Strip**((*unknown*) TString::EStripType s = kTrailing, char c =)
class TSubString	**SubString**(char* pat, Ssiz_t start=0, (*unknown*) TString::ECaseCompare cmp=kExact)
void	**ToLower**( )
void	**ToUpper**( )

Data Members

private:

static Ssiz_t	**fgInitialCapac**	*Initial allocation Capacity*
static Ssiz_t	**fgResizeInc**	*Resizing increment*
static Ssiz_t	**fgFreeboard**	*Max empty space before reclaim*

protected:

char*	**fData**	*ref. counted data (TStringRef is in front)*

public:

static const enum TString::EStripType**kLeading**
static const enum TString::EStripType**kTrailing**
static const enum TString::EStripType**kBoth**
static const enum TString::ECaseCompare**kExact**
static const enum TString::ECaseCompare**kIgnoreCase**

See Also: TStringLong

Class Description

```
TString

Basic string class.
```

Ssiz_t **GetMaxWaste**( )

TString( )

```
TString default ctor.
```

TString(Ssiz_t *ic*)

```
Create TString able to contain ic characters.
```

TString(const char* *cs*)

```
Create TString and initialize it with string cs.
```

TString(const char* *cs,* Ssiz_t *n*)

```
Create TString and initialize it with the first n characters of cs.
```

void **InitChar**(char *c*)

```
Initialize a string with a single character.
```

TString(char *c,* Ssiz_t *n*)

```
Initialize the first n locations of a TString with character c.
```

TString(const TSubString& *substr*)

```
Copy a TSubString in a TString.
```

~TString( )

```
Delete a TString. I.e. decrease its reference count. When 0 free space.
```

Ssiz_t **Capacity**(Ssiz_t *nc*)

```
Return string capacity. If nc != current capacity Clone() the string
in a string with the desired capacity.
```

int **CompareTo**(const char* cs2, ECaseCompare *cmp*) const

```
Compare a string to char *cs2.
```

int **CompareTo**(const TString& str, ECaseCompare *cmp*) const

```
Compare a string to another string.
```

TString **Copy**( ) const

```
Copy a string.
```

unsigned **Hash**(ECaseCompare *cmp*) const
    Return hash value.

Ssiz_t **Index**(const char* *pattern,* Ssiz_t *plen,* Ssiz_t *startIndex,*ECaseCompare *cmp*) const
    Search for a string in the TString. Plen is the length of pattern, startIndex is the index from which to start and cmp selects the type of case-comparison.

void **Resize**(Ssiz_t *n*)
    Resize the string. Truncate or add blanks as necessary.

TSubString **Strip**(EStripType *st,* char *c*)
    Return a substring of self stripped at beginning and/or end.

const TSubString **Strip**(EStripType st, char *c*) const
    Just use the "non-const" version, adjusting the return type.

void **ToLower**( )
    Change string to lower-case.

void **ToUpper**( )
    Change string to upper case.

void **AssertElement**(Ssiz_t *i*) const
    Check to make sure a string index is in range.

TString(const char* *a1,* Ssiz_t *n1,* const char* *a2,* Ssiz_t *n2*)
    Special constructor to initialize with the concatenation of a1 and a2.

Ssiz_t **AdjustCapacity**(Ssiz_t *nc*)
    Calculate a nice capacity greater than or equal to nc.

void **Clobber**(Ssiz_t *nc*)
    Clear string and make sure it has a capacity of nc.

void **Clone**( )
    Make string a distinct copy; preserve previous contents.

void **Clone**(Ssiz_t *nc*)
    Make self a distinct copy with capacity of at least nc.
    Preserve previous contents.

void **FillBuffer**(char *&*buffer*)
    Copy string into I/O buffer.

void **ReadBuffer**(char *&*buffer*)
    Read string from I/O buffer.

Int_t **Sizeof**( ) const
    Returns size string will occupy on I/O buffer.

void **Streamer**(TBuffer &*b*)
    Stream a string object

Ssiz_t **InitialCapacity**(Ssiz_t *ic*)
    Set default initial capacity for all TStrings. Default is 15.

Ssiz_t **ResizeIncrement**(Ssiz_t *ri*)
    Set default resize increment for all TStrings. Default is 16.

Ssiz_t **MaxWaste**(Ssiz_t *mw*)
    Set maximum space that may be wasted in a string before doing a resize.
    Default is 15.

TSubString **SubString**(const char* *pattern,* Ssiz_t *startIndex,*ECaseCompare *cmp*)

```
Returns a substring matching "pattern", or the null substring
if there is no such match.  It would be nice if this could be yet another
overloaded version of operator(), but this would result in a type
conversion ambiguity with operator(Ssiz_t, Ssiz_t).
```

const TSubString **SubString**(const char* *pattern,* Ssiz_t *startIndex,*ECaseCompare *cmp*) const

```
Return sub-string matching pattern, starting at index. Cmp selects
the type of case conversion.
```

Bool_t **IsAscii**( ) const

```
Return true if all characters in string are ascii.
```

Author: Fons Rademakers 04/08/95
Last update: 1.00/06 23/03/97 11.01.58 by René Brun

TStringLong

class **TStringLong** : public TString

public:

class TStringLong	**TStringLong**(Ssiz_t *ic*)
class TStringLong	**TStringLong**(class TString& *s*)
class TStringLong	**TStringLong**(char* *s*)
class TStringLong	**TStringLong**(char* *s,* Ssiz_t *n*)
class TStringLong	**TStringLong**( )
class TStringLong	**TStringLong**(char *c*)
class TStringLong	**TStringLong**(class TStringLong&)
class TStringLong	**TStringLong**(char *c,* Ssiz_t *s*)
class TStringLong	**TStringLong**(class TSubString& *sub*)
void	**~TStringLong**( )
class TClass*	**Class**( )
virtual void	**FillBuffer**(char*& *buffer*)
virtual class TClass*	**IsA**( )
virtual void	**ReadBuffer**(char*& *buffer*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual Int_t	**Sizeof**( )
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TStringLong

The long string class (unlimited number of chars in I/O).

This class redefines only the I/O member functions of TString.
It uses 4 bytes to store the string length (1 byte only for TString).
```

void **Streamer**(TBuffer &*b*)

```
Stream a long (>255 characters) string object.
```

Author: René Brun 15/11/95
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TStyle

class **TStyle** : public TNamed, public TAttLine, public TAttFill, public TAttMarker, public TAttText

public:

class TStyle	**TStyle**( )
class TStyle	**TStyle**(Text_t* *name,* Text_t* *title*)
class TStyle	**TStyle**(class TStyle& *style*)
void	**~TStyle**( )
Int_t	**AxisChoice**(Option_t* *axis*)
virtual void	**Browse**(class TBrowser* *b*)
virtual void	**cd**( )
class TClass*	**Class**( )
virtual void	**Copy**(class TObject& *style*)
Color_t	**GetAxisColor**(Option_t* *axis* = *X*)
Float_t	**GetBarOffset**( )
Float_t	**GetBarWidth**( )
Int_t	**GetCanvasBorderMode**( )
Width_t	**GetCanvasBorderSize**( )
Color_t	**GetCanvasColor**( )
Int_t	**GetColorPalette**(Int_t *i*)
Int_t	**GetDrawBorder**( )
Int_t	**GetErrorMarker**( )
Float_t	**GetErrorMsize**( )
Float_t	**GetErrorX**( )
Int_t	**GetFrameBorderMode**( )
Width_t	**GetFrameBorderSize**( )
Color_t	**GetFrameFillColor**( )
Style_t	**GetFrameFillStyle**( )
Color_t	**GetFrameLineColor**( )
Style_t	**GetFrameLineStyle**( )
Width_t	**GetFrameLineWidth**( )
Color_t	**GetFuncColor**( )
Style_t	**GetFuncStyle**( )
Width_t	**GetFuncWidth**( )
Color_t	**GetHistFillColor**( )
Style_t	**GetHistFillStyle**( )
Color_t	**GetHistLineColor**( )
Style_t	**GetHistLineStyle**( )
Width_t	**GetHistLineWidth**( )
Color_t	**GetLabelColor**(Option_t* *axis* = *X*)
Style_t	**GetLabelFont**(Option_t* *axis* = *X*)
Float_t	**GetLabelOffset**(Option_t* *axis* = *X*)
Float_t	**GetLabelSize**(Option_t* *axis* = *X*)
Float_t	**GetLegoInnerR**( )
Int_t	**GetNdivisions**(Option_t* *axis* = *X*)
Int_t	**GetNumberOfColors**( )
Int_t	**GetOptDate**( )
Int_t	**GetOptFile**( )
Int_t	**GetOptFit**( )
Int_t	**GetOptLogx**( )
Int_t	**GetOptLogy**( )
Int_t	**GetOptLogz**( )
Int_t	**GetOptStat**( )
Int_t	**GetOptTitle**( )
Int_t	**GetPadBorderMode**( )

Width_t	**GetPadBorderSize**( )
Float_t	**GetPadBottomMargin**( )
Color_t	**GetPadColor**( )
Float_t	**GetPadLeftMargin**( )
Float_t	**GetPadRightMargin**( )
Float_t	**GetPadTopMargin**( )
void	**GetPaperSize**(Float_t& *xsize,* Float_t& *ysize*)
Float_t	**GetScreenFactor**( )
Int_t	**GetShowEventStatus**( )
Color_t	**GetStatColor**( )
Float_t	**GetStatH**( )
Float_t	**GetStatW**( )
Float_t	**GetStatX**( )
Float_t	**GetStatY**( )
Float_t	**GetTickLength**(Option_t* *axis = X*)
Color_t	**GetTitleColor**( )
Float_t	**GetTitleH**( )
Float_t	**GetTitleOffset**(Option_t* *axis = X*)
Float_t	**GetTitleW**( )
Float_t	**GetTitleX**( )
Style_t	**GetTitleXFont**( )
Float_t	**GetTitleXOffset**( )
Float_t	**GetTitleXSize**( )
Float_t	**GetTitleY**( )
Style_t	**GetTitleYFont**( )
Float_t	**GetTitleYOffset**( )
Float_t	**GetTitleYSize**( )
virtual class TClass*	**IsA**( )
virtual void	**Reset**(Option_t* *option*)
void	**SetAxisColor**(Color_t *color = 1,* Option_t* *axis = X*)
void	**SetBarOffset**(Float_t *baroff* = 0.*5*)
void	**SetBarWidth**(Float_t *barwidth* = 0.*5*)
void	**SetCanvasBorderMode**(Int_t *mode = 1*)
void	**SetCanvasBorderSize**(Width_t *size = 1*)
void	**SetCanvasColor**(Color_t *color = 19*)
void	**SetDrawBorder**(Int_t *drawborder = 1*)
void	**SetErrorMarker**(Int_t *marker = 21*)
void	**SetErrorMsize**(Float_t *msize* = 0.*05*)
void	**SetErrorX**(Float_t *errorx* = 0.*5*)
void	**SetFrameBorderMode**(Int_t *mode = 1*)
void	**SetFrameBorderSize**(Width_t *size = 1*)
void	**SetFrameFillColor**(Color_t *color = 1*)
void	**SetFrameFillStyle**(Style_t *styl = 0*)
void	**SetFrameLineColor**(Color_t *color = 1*)
void	**SetFrameLineStyle**(Style_t *styl = 0*)
void	**SetFrameLineWidth**(Width_t *width = 1*)
void	**SetFuncColor**(Color_t *color = 1*)
void	**SetFuncStyle**(Style_t *style = 1*)
void	**SetFuncWidth**(Width_t *width = 4*)
void	**SetHistFillColor**(Color_t *color = 1*)
void	**SetHistFillStyle**(Style_t *styl = 0*)
void	**SetHistLineColor**(Color_t *color = 1*)
void	**SetHistLineStyle**(Style_t *styl = 0*)
void	**SetHistLineWidth**(Width_t *width = 1*)
void	**SetLabelColor**(Color_t *color = 1,* Option_t* *axis = X*)
void	**SetLabelFont**(Style_t *font = 62,* Option_t* *axis = X*)
void	**SetLabelOffset**(Float_t *offset* = 0.*005,* Option_t* *axis = X*)
void	**SetLabelSize**(Float_t *size* = 0.*02,* Option_t* *axis = X*)

void	**SetLegoInnerR**(Float_t *rad* = 0.*5*)
void	**SetNdivisions**(Int_t *n = 510,* Option_t* *axis = X*)
void	**SetOptDate**(Int_t *datefl = 1*)
void	**SetOptFile**(Int_t *file = 1*)
void	**SetOptFit**(Int_t *fit = 1*)
void	**SetOptLogx**(Int_t *logx = 1*)
void	**SetOptLogy**(Int_t *logy = 1*)
void	**SetOptLogz**(Int_t *logz = 1*)
void	**SetOptStat**(Int_t *stat = 1*)
void	**SetOptTitle**(Int_t *tit = 1*)
void	**SetPadBorderMode**(Int_t *mode = 1*)
void	**SetPadBorderSize**(Width_t *size = 1*)
void	**SetPadBottomMargin**(Float_t *margin* = 0.*1*)
void	**SetPadColor**(Color_t *color = 19*)
void	**SetPadLeftMargin**(Float_t *margin* = 0.*1*)
void	**SetPadRightMargin**(Float_t *margin* = 0.*1*)
void	**SetPadTopMargin**(Float_t *margin* = 0.*1*)
void	**SetPalette**(Int_t *ncolors = 0,* Int_t* *colors = 0*)
void	**SetPaperSize**(Float_t *xsize = 20,* Float_t *ysize = 26*)
void	**SetScreenFactor**(Float_t *factor = 1*)
void	**SetStatColor**(Int_t *color = 19*)
void	**SetStatH**(Float_t *h* = 0.*1*)
void	**SetStatW**(Float_t *w* = 0.*19*)
void	**SetStatX**(Float_t *x = 0*)
void	**SetStatY**(Float_t *y = 0*)
void	**SetTickLength**(Float_t *length* = 0.*02,* Option_t* *axis = X*)
void	**SetTitleColor**(Int_t *color = 19*)
void	**SetTitleH**(Float_t *h = 0*)
void	**SetTitleOffset**(Float_t *offset = 1,* Option_t* *axis = X*)
void	**SetTitleW**(Float_t *w = 0*)
void	**SetTitleX**(Float_t *x = 0*)
void	**SetTitleXFont**(Style_t *font = 62*)
void	**SetTitleXOffset**(Float_t *offset = 1*)
void	**SetTitleXSize**(Float_t *size* = 0.*02*)
void	**SetTitleY**(Float_t *y = 1*)
void	**SetTitleYFont**(Style_t *font = 62*)
void	**SetTitleYOffset**(Float_t *offset = 1*)
void	**SetTitleYSize**(Float_t *size* = 0.*02*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
void	**ToggleEventStatus**( )

Data Members

private:

class TAttAxis	**fXaxis**	*X axis attributes*
class TAttAxis	**fYaxis**	*Y axis attributes*
class TAttAxis	**fZaxis**	*Z axis attributes*
Float_t	**fBarWidth**	*width of bar for graphs*
Float_t	**fBarOffset**	*offset of bar for graphs*
Int_t	**fDrawBorder**	*flag to draw border(=1*) or not (0)
Int_t	**fOptLogx**	*=1 if log scale in X*
Int_t	**fOptLogy**	*=1 if log scale in y*
Int_t	**fOptLogz**	*=1 if log scale in z*
Int_t	**fOptDate**	*=1 if date option is selected*
Int_t	**fOptStat**	*=1 if option Stat is selected*
Int_t	**fOptTitle**	*=1 if option Title is selected*
Int_t	**fOptFile**	*=1 if option File is selected*
Int_t	**fOptFit**	*=1 if option Fit is selected*

Int_t	**fErrorMarker**	*marker for error bars*
Int_t	**fShowEventStatus**	*Show event status panel*
Float_t	**fErrorMsize**	*marker size for error bars*
Float_t	**fErrorX**	*per cent of bin width for errors along X*
Color_t	**fFuncColor**	*function color*
Style_t	**fFuncStyle**	*function style*
Width_t	**fFuncWidth**	*function line width*
Color_t	**fFrameFillColor**	*pad frame fill color*
Color_t	**fFrameLineColor**	*pad frame line color*
Style_t	**fFrameFillStyle**	*pad frame fill style*
Style_t	**fFrameLineStyle**	*pad frame line style*
Width_t	**fFrameLineWidth**	*pad frame line width*
Width_t	**fFrameBorderSize**	*pad frame border size*
Int_t	**fFrameBorderMode**	*pad frame border mode*
Color_t	**fHistFillColor**	*histogram fill color*
Color_t	**fHistLineColor**	*histogram line color*
Style_t	**fHistFillStyle**	*histogram fill style*
Style_t	**fHistLineStyle**	*histogram line style*
Width_t	**fHistLineWidth**	*histogram line width*
Color_t	**fCanvasColor**	*canvas color*
Width_t	**fCanvasBorderSize**	*canvas border size*
Int_t	**fCanvasBorderMode**	*canvas border mode*
Color_t	**fPadColor**	*pad color*
Width_t	**fPadBorderSize**	*pad border size*
Int_t	**fPadBorderMode**	*pad border mode*
Float_t	**fPadBottomMargin**	*pad bottom margin*
Float_t	**fPadTopMargin**	*pad top margin*
Float_t	**fPadLeftMargin**	*pad left margin*
Float_t	**fPadRightMargin**	*pad right margin*
Float_t	**fPaperSizeX**	*PostScript paper size along X*
Float_t	**fPaperSizeY**	*PostScript paper size along Y*
Float_t	**fScreenFactor**	*Multiplication factor for canvas size and position*
Color_t	**fStatColor**	*stat color*
Float_t	**fStatX**	*X position of top right corner of stat box*
Float_t	**fStatY**	*Y position of top right corner of stat box*
Float_t	**fStatW**	*width of stat box*
Float_t	**fStatH**	*height of stat box*
Color_t	**fTitleColor**	*title color*
Float_t	**fTitleXOffset**	*Offset of title to X axis*
Style_t	**fTitleXFont**	*font for X title*
Float_t	**fTitleXSize**	*size of X title*
Float_t	**fTitleYOffset**	*Offset of title to Y axis*
Style_t	**fTitleYFont**	*font for Y title*
Float_t	**fTitleYSize**	*size of Y title*
Float_t	**fTitleX**	*X position of top left corner of title box*
Float_t	**fTitleY**	*Y position of top left corner of title box*
Float_t	**fTitleW**	*width of title box*
Float_t	**fTitleH**	*height of title box*
Float_t	**fLegoInnerR**	*Inner radius for cylindrical legos*
class TArrayI	**fPalette**	*Color palette*

Class Description

```
TStyle objects may be created to define special styles.
By default ROOT creates a default style that can be accessed via
the gStyle pointer.

This class includes functions to set the following object attributes.
- Canvas
```

```
- Pad
- Histogram axis
- Lines
- Fill areas
- Text
- Markers
- Functions
- Histogram Statistics and Titles
```

void **cd**( )

```
Change current style
```

void **GetPaperSize**(Float_t &*xsize,* Float_t &*ysize*)

```
Set paper size for PostScript output
```

void **SetPaperSize**(Float_t *xsize,* Float_t *ysize*)

```
Set paper size for PostScript output
```

void **SetPalette**(Int_t *ncolors,* Int_t **colors*)

Author: René Brun 12/12/94
Last update: 1.03/05 02/10/97 17.37.28 by René Brun

TSubMenu

class **TSubMenu** : public TMenu

protected:

class TSubMenu	**TSubMenu**( )

public:

class TSubMenu	**TSubMenu**(class TSubMenu&)
class TSubMenu	**TSubMenu**(char* *name,* MenuDesc_t* *menuDesc* = *NULL,* XtPointer *defaultClientData* = *NULL*)
void	**~TSubMenu**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**Instantiate**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual Boolean	**IsContainer**( )
virtual EMenuItemType	**MenuType**( )
virtual Widget	**operator Widget**( )
Widget	**Pulldown**( )
virtual void	**SetLabel**(char* *str*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**ShowTearOff**(Boolean *showIt*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Widget	**fPulldown**
Boolean	**fTearOff**

See Also: THelpPane, TRadioSubMenu

Class Description

```
TSubMenu

This class defines sub menus.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.03/03 22/08/97 11.38.49 by Piotr Golonka

TSysEvtHandler

class **TSysEvtHandler** : public TObject

public:

void	**~TSysEvtHandler**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**Notify**( )
virtual void	**Remove**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TFileHandler, TSignalHandler, TTimer

Class Description

```
TSysEvtHandler

Abstract base class for handling system events.
```

Author: Fons Rademakers 16/09/95
Last update: 1.03/02 25/08/97 16.23.18 by Fons Rademakers

TSystem

class **TSystem** : public TNamed

public:

class TSystem	**TSystem**(char* *name = Generic,* char* *title* = Generic *System*)
class TSystem	**TSystem**(class TSystem&)
void	**~TSystem**( )
virtual void	**Abort**(int *code = 0*)
virtual int	**AcceptConnection**(int *sock*)
virtual Bool_t	**AccessPathName**(char* path, (*unknown*) EAccessMode mode = kFileExists)
virtual void	**AddFileHandler**(class TFileHandler* *fh*)
virtual void	**AddSignalHandler**(class TSignalHandler* *sh*)
virtual void	**AddTimer**(class TTimer* *t*)
virtual int	**AnnounceTcpService**(int *port,* Bool_t *reuse,* int *backlog*)
virtual int	**AnnounceUnixService**(int *port,* int *backlog*)
virtual const char*	**BaseName**(char* *pathname*)
Bool_t	**cd**(char* *path*)
virtual Bool_t	**ChangeDirectory**(char* *path*)
class TClass*	**Class**( )
virtual void	**CloseConnection**(int *sock*)
virtual void	**Closelog**( )
virtual int	**ClosePipe**(FILE* *pipe*)
virtual char*	**ConcatFileName**(char* *dir,* char* *name*)
virtual void	**CreateInspectView**(class TObject* *obj*)
virtual const char*	**DirName**(char* *pathname*)
virtual void	**DispatchOneEvent**( )
virtual char*	**DynamicPathName**(char* *lib*)
virtual Func_t*	**DynFindSymbol**(char* *module,* char* *entry*)
virtual Int_t	**Exec**(char* *shellcmd*)
virtual void	**Exit**(int *code,* Bool_t *mode = kTRUE*)
void	**ExitLoop**( )
virtual char*	**ExpandPathName**(char* *path*)
virtual Bool_t	**ExpandPathName**(class TString& *path*)
virtual void	**FreeDirectory**(void* *dirp*)
virtual const char*	**GetDirEntry**(void* *dirp*)
virtual const char*	**Getenv**(char* *env*)
virtual const char*	**GetError**( )
const char*	**GetErrorStr**( )
virtual class TInetAddress	**GetHostByName**(char* *server*)
virtual int	**GetPathInfo**(char* *path,* Long_t* *id,* Long_t* *size,* Long_t* *flags,* Long_t* *modtime*)
virtual class TInetAddress	**GetPeerName**(int *sock*)
virtual int	**GetPid**( )
virtual int	**GetServiceByName**(char* *service*)
virtual char*	**GetServiceByPort**(int *port*)
virtual class TInetAddress	**GetSockName**(int *sock*)
virtual int	**GetSockOpt**(int *sock,* int *kind,* int* *val*)
virtual const char*	**HomeDirectory**( )
virtual const char*	**HostName**( )
virtual void	**IgnoreInterrupt**(Bool_t *ignore = kTRUE*)
Bool_t	**InControl**( )
virtual Bool_t	**Init**( )
void	**InnerLoop**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsAbsoluteFileName**(char* *dir*)

virtual int	**Link**(char* *from,* char* *to*)
virtual void	**ListLibraries**(char* *regexp*)
virtual void	**ListSymbols**(char* *module,* char* *re*)
virtual int	**Load**(char* *module,* char* *entry*)
virtual int	**MakeDirectory**(char* *name*)
int	**mkdir**(char* *name*)
virtual Long_t	**NextTimeOut**(Bool_t *mode*)
virtual class TTime	**Now**( )
virtual int	**OpenConnection**(char* *server,* int *port*)
virtual void*	**OpenDirectory**(char* *name*)
virtual void	**Openlog**(char* name, Int_t options, (*unknown*) ELogFacility facility)
virtual FILE*	**OpenPipe**(char* *command,* char* *mode*)
const char*	**pwd**( )
virtual int	**RecvBuf**(int *sock,* void* *buffer,* int *length*)
virtual int	**RecvRaw**(int *sock,* void* *buffer,* int *length,* int *flag*)
virtual class TFileHandler*	**RemoveFileHandler**(class TFileHandler* *fh*)
void	**RemoveOnExit**(class TObject* *obj*)
virtual class TSignalHandler*	**RemoveSignalHandler**(class TSignalHandler* *sh*)
virtual class TTimer*	**RemoveTimer**(class TTimer* *t*)
virtual void	**Rename**(char* *from,* char* *to*)
virtual void	**Run**( )
virtual int	**SendBuf**(int *sock,* void* *buffer,* int *length*)
virtual int	**SendRaw**(int *sock,* void* *buffer,* int *length,* int *flag*)
virtual void	**Setenv**(char* *name,* char* *value*)
virtual void	**SetProgname**(char* *name*)
virtual int	**SetSockOpt**(int *sock,* int *kind,* int *val*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sleep**(UInt_t *milliSec*)
virtual void	**StackTrace**( )
virtual void	**Streamer**(class TBuffer& *b*)
virtual int	**Symlink**(char* *from,* char* *to*)
virtual void	**Syslog**((*unknown*) ELogLevel level, char* mess)
virtual int	**Umask**(Int_t *mask*)
virtual const char*	**UnixPathName**(char* *unixpathname*)
virtual int	**Unlink**(char* *name*)
virtual void	**Unload**(char* *module*)
virtual void	**Unsetenv**(char* *name*)
virtual char*	**Which**(char* search, char* file, (*unknown*) EAccessMode mode = kFileExists)
virtual const char*	**WorkingDirectory**( )

Data Members

protected:

class TFdSet	**fReadmask**	*!Files that should be checked for read events*
class TFdSet	**fWritemask**	*!Files that should be checked for write events*
class TFdSet	**fReadready**	*!Files with reads waiting*
class TFdSet	**fWriteready**	*!Files with writes waiting*
class TFdSet	**fSignals**	*!Signals that were trapped*
Int_t	**fNfd**	*Number of fd's in masks*
Int_t	**fMaxrfd**	*Largest fd in read mask*
Int_t	**fMaxwfd**	*Largest fd in write mask*
Int_t	**fSigcnt**	*Number of pending signals*
class TString	**fWdpath**	*Working directory*
class TString	**fHostname**	*Hostname*
Bool_t	**fInsideNotify**	*Used by DispatchTimers( )*
Bool_t	**fInControl**	*True if in eventloop*
Bool_t	**fDone**	*True if eventloop should be finished*
Int_t	**fLevel**	*Level of nested eventloops*

class TString	**fLastErrorString**	*Last system error message*
class TSeqCollection*	**fTimers**	*List of timers*
class TSeqCollection*	**fSignalHandler**	*List of signal handlers*
class TSeqCollection*	**fFileHandler**	*List of file handlers*
class TSeqCollection*	**fOnExitList**	*List of items to be cleaned-up on exit*

See Also: TUnixSystem

Class Description

```
TSystem

Abstract base class defining a generic interface to the underlying
Operating System.
This is not an ABC in the strict sense of the (C++) word. For
every member function their is an implementation (often not more
than a call to AbstractMethod() which prints a warning saying
that the method should be overridden in a derived class), which
allows a simple partial implementation for new OS'es.
```

TSystem(const char **name,* const char **title*) : **TNamed**(*name,* title)

    Create a new OS interface.

~TSystem( )

    Delete the OS interface.

Bool_t **Init**( )

    Initialize the OS interface. Copy the OS name (i.e. Unix) and the
    ROOT name to gSystemName and gRootName, respectively.

void **SetProgname**(const char **name*)

    Set the application name (from command line, argv[0]) and copy it in
    gProgName.

void **CreateInspectView**(TObject **obj*)

    Create an inspect view of the obj to perform TObject::Inspect

const char* **GetError**( )

    Return system error string.

void **RemoveOnExit**(TObject **obj*)

    Objects that should be deleted on exit of the OS interface.

const char* **HostName**( )

    Return the system's host name.

void **Run**( )

    System event loop.

void **ExitLoop**( )

    Exit from event loop.

void **InnerLoop**( )

    Inner event loop.

void **DispatchOneEvent**( )

    Dispatch a single event.

void **Sleep**(UInt_t *milliSec*)

    Sleep milliSec milli seconds.

TTime **Now**( )

    Return current time.

void **AddTimer**(TTimer **ti*)
    `Add timer to list of system timers.`

TTimer* **RemoveTimer**(TTimer **ti*)
    `Remove timer from list of system timers.`

Long_t **NextTimeOut**(Bool_t *mode*)
    `Time when next timer of mode (synchronous=kTRUE or asynchronous=kFALSE) will time-out (in ms).`

void **AddSignalHandler**(TSignalHandler **h*)
    `Add a signal handler to list of system signal handlers.`

TSignalHandler* **RemoveSignalHandler**(TSignalHandler **h*)
    `Remove a signal handler from list of signal handlers.`

void **AddFileHandler**(TFileHandler **h*)
    `Add a file handler to the list of system file handlers.`

TFileHandler* **RemoveFileHandler**(TFileHandler **h*)
    `Remove a file handler from the list of file handlers.`

void **IgnoreInterrupt**(Bool_t *ignore*)
    `Ignore the interrupt signal if ignore == kTRUE else restore previous behaviour. Typically call ignore interrupt before writing to disk.`

int **Exec**(const char*)
    `Execute a command.`

FILE* **OpenPipe**(const char*, const char*)
    `Open a pipe.`

int **ClosePipe**(FILE*)
    `Close the pipe.`

int **GetPid**( )
    `Get process id.`

void **Exit**(*int, Bool_t*)
    `Exit the application.`

void **Abort**(*int*)
    `Abort the application.`

void **StackTrace**( )
    `Print a stack trace.`

int **MakeDirectory**(const char*)
    `Make a directory. Returns 0 in case of success and -1 if the directory could not be created.`

void* **OpenDirectory**(const char*)
    `Open a directory. Returns 0 if directory does not exist.`

void **FreeDirectory**(void*)
    `Free a directory.`

const char* **GetDirEntry**(void*)
    `Get a directory entry. Returns 0 if no more entries.`

Bool_t **ChangeDirectory**(const char*)
    `Change directory.`

const char* **WorkingDirectory**( )
```
Return working directory.
```

const char* **HomeDirectory**( )
```
Return the user's home directory.
```

const char* **BaseName**(const char **name*)
```
Base name of a file name. Base name of /user/root is root.
```

Bool_t **IsAbsoluteFileName**(const char **dir*)
```
Return true if dir is an absolute pathname.
```

const char* **DirName**(const char **pathname*)
```
Return the directory name in pathname. DirName of /user/root is /user.
```

const char* **UnixPathName**(const char **name*)
```
Convert from a Unix pathname to a local pathname. E.g. from /user/root to
\user\root.
```

char* **ConcatFileName**(const char **dir,* const char **name*)
```
Concatenate a directory and a file name.
```

Bool_t **ExpandPathName**(TString&)
```
Expand a pathname getting rid of special shell characaters like ~.$, etc.
```

char* **ExpandPathName**(const char *)
```
Expand a pathname getting rid of special shell characaters like ~.$, etc.
```

Bool_t **AccessPathName**(const char*, *EAccessMode*)
```
Returns FALSE if one can access a file using the specified access mode.
```

void **Rename**(const char **from,* const char **to*)
```
Rename a file.
```

int **Link**(const char **file1,* const char **file2*)
```
Create a link from file1 to file2.
```

int **Symlink**(const char **file1,* const char **file2*)
```
Create a symbolic link from file1 to file2.
```

int **Unlink**(const char **name*)
```
Unlink, i.e. remove, a file.
```

int **GetPathInfo**(const char*, Long_t*, Long_t*, Long_t*, Long_t*)
```
Get info about a file: id, size, flags, modification time.
```

int **Umask**(Int_t *mask*)
```
Set the process file creation mode mask.
```

char* **Which**(const char **search,* const char **file,* EAccessMode *mode*)
```
Find location of file in a search path.
```

void **Setenv**(const char*, const char*)
```
Set environment variable.
```

void **Unsetenv**(const char **name*)
```
Unset environment variable.
```

const char* **Getenv**(const char*)
```
Get environment variable.
```

void **Openlog**(const char **name,* Int_t *options,* ELogFacility *facility*)
```
Open connection to system log daemon. For the use of the options and
```

`facility see the Unix openlog man page.`

void **Syslog**(ELogLevel *level,* const char **mess*)
`Send mess to syslog daemon. Level is the logging level and mess the message that will be written on the log.`

void **Closelog**( )
`Close connection to system log daemon.`

int **Load**(const char **module,* const char **entry*)
`Load a shared library. Returns 0 on successful loading, 1 in case lib was already loaded and -1 in case lib does not exist or in case of error.`

void **Unload**(const char *)
`Unload a shared library.`

void **ListSymbols**(const char *, const char *)
`List symbols in a shared library.`

void **ListLibraries**(const char *)
`List all loaded shared libraries.`

TInetAddress **GetHostByName**(const char *)
`Get Internet Protocol (IP) address of host.`

TInetAddress **GetPeerName**(*int*)
`Get Internet Protocol (IP) address of remote host and port #.`

TInetAddress **GetSockName**(*int*)
`Get Internet Protocol (IP) address of host and port #.`

int **GetServiceByName**(const char *)
`Get port # of internet service.`

char* **GetServiceByPort**(*int*)
`Get name of internet service.`

int **OpenConnection**(const char*, *int*)
`Open a connection to another host.`

int **AnnounceTcpService**(*int, Bool_t, int*)
`Announce TCP/IP service.`

int **AnnounceUnixService**(*int, int*)
`Announce unix domain service.`

int **AcceptConnection**(*int*)
`Accept a connection.`

void **CloseConnection**(*int*)
`Close socket connection.`

int **RecvRaw**(*int,* void *, *int, int*)
`Receive exactly length bytes into buffer. Use opt to receive out-of-band data or to have a peek at what is in the buffer (see TSocket).`

int **SendRaw**(*int,* const void *, *int, int*)
`Send exactly length bytes from buffer. Use opt to send out-of-band data (see TSocket).`

int **RecvBuf**(*int,* void *, *int*)
`Receive a buffer headed by a length indicator.`

int **SendBuf**(*int,* const void *, *int*)
`Send a buffer headed by a length indicator.`

int **SetSockOpt**(*int, int, int*)
`Set socket option.`

int **GetSockOpt**(*int, int,* int*)
`Get socket option.`

Author: Fons Rademakers 15/09/95
Last update: 1.03/06 27/10/97 17.01.08 by Fons Rademakers

TSystemDirectory

class **TSystemDirectory** : public TSystemFile

private:

virtual Bool_t	**CheckDirectory**(char* *name*)

public:

class TSystemDirectory	**TSystemDirectory**(Text_t* *dirname,* Text_t* *path*)
class TSystemDirectory	**TSystemDirectory**( )
class TSystemDirectory	**TSystemDirectory**(class TSystemDirectory&)
void	**~TSystemDirectory**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Edit**( )
virtual void	**Execute**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsFolder**( )
virtual void	**SetDirectory**(Text_t* *name*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TSystemDirectory

Describes an Operating System directory for the browser.

Author: Christian Bormann  30/09/97
      http://www.ikf.physik.uni-frankfurt.de/~bormann/
```

Bool_t **CheckDirectory**(const char* *name*)

```
Check if a file is a directory
```

void **Browse**(TBrowser **b*)

Author: Christian Bormann 13/10/97
Last update: 1.03/05 13/10/97 16.06.01 by René Brun

TSystemFile

class **TSystemFile** : public TNamed

public:

class TSystemFile	**TSystemFile**(Text_t* *filename,* Text_t* *dirname*)
class TSystemFile	**TSystemFile**( )
class TSystemFile	**TSystemFile**(class TSystemFile&)
void	**~TSystemFile**( )
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**Edit**( )
virtual void	**Execute**( )
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

See Also: TSystemDirectory

Class Description

```
TSystemFile describes one operating system file (or directory).
The information is used by the browser (see TBrowser).
```

TSystemFile( ): **TNamed**( )

```
SystemFile default constructor
```

TSystemFile(const Text_t **filename,* const Text_t **dirname*): **TNamed**(*filename, dirname*)

```
SystemFile normal constructor
```

~TSystemFile( )

```
SystemFile default destructor
```

void **Edit**( )

```
Invoke text editor on this file
```

void **Execute**( )

```
Execute this file as a CINT macro
```

Author: René Brun 26/06/96
Last update: 1.03/05 13/10/97 16.01.56 by René Brun

TText

class **TText** : public TNamed, public TAttText

public:

class TText	**TText**(Coord_t *x,* Coord_t *y,* Text_t* *text*)
class TText	**TText**( )
class TText	**TText**(class TText& *text*)
void	**~TText**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *text*)
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawText**(Coord_t *x,* Coord_t *y,* Text_t* *text*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Coord_t	**GetX**( )
Coord_t	**GetY**( )
virtual class TClass*	**IsA**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintText**(Coord_t *x,* Coord_t *y,* Text_t* *text*)
virtual void	**PaintTextNDC**(Coord_t *u,* Coord_t *v,* Text_t* *text*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetText**(Coord_t *x,* Coord_t *y,* Text_t* *text*)
virtual void	**SetX**(Coord_t *x*)
virtual void	**SetY**(Coord_t *y*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Coord_t	**fX**	*X position of text (left, center, etc..)*
Coord_t	**fY**	*Y position of text (left, center, etc..)*

See Also: TLink

Class Description

```
TText is the base class for several text objects.
See TAttText for a list of text attributes or fonts.

In case a text is drawn to a PostScript file (see TPostScript)
the following characters have a special action:

   ` : go to greek
   ' : go to special characters
   ~ : go to ZapfDingbats
   ? : go to subscript
   ^ : go to superscript
   ! : go to normal level of script
   & : backspace one character
   # : end of greek or of ZapfDingbats

This special characters have no effect on the screen.
```

TText( ): **TNamed**( ), **TAttText**( )

```
Text default constructor
```

TText(Coord_t *x,* Coord_t *y,* const Text_t **text*) : **TNamed**("", text), **TAttText**( )
```
Text normal constructor
```

~**TText**( )
```
Text default destructor
```

void **Copy**(TObject &*obj*)
```
Copy this text to text
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)
```
Compute distance from point px,py to a string
Compute the closest distance of approach from point px,py to this text.
The rectangle surrounding this string is evaluated.
if point is in rectangle, the distance is set to zero.
```

void **Draw**(Option_t **option*)
```
Draw this text with its current attributes
```

void **DrawText**(Coord_t *x,* Coord_t *y,* const Text_t **text*)
```
Draw this text with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
```
Execute action corresponding to one event
This member function must be implemented to realize the action
corresponding to the mouse click on the object in the window
```

void **ls**(Option_t **option*)
```
List this text with its attributes
```

void **Paint**(Option_t **option*)
```
Paint this text with its current attributes
```

void **PaintText**(Coord_t *x,* Coord_t *y,* const Text_t **text*)
```
Draw this text with new coordinates
```

void **PaintTextNDC**(Coord_t *u,* Coord_t *v,* const Text_t **text*)
```
Draw this text with new coordinates in NDC
```

void **Print**(Option_t **option*)
```
Dump this text with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)
```
Save primitive as a C++ statement(s) on output stream out
```

Author: Nicolas Brun 12/12/94
Last update: 1.02/00 09/07/97 22.33.22 by René Brun

TTime

class **TTime**

public:

class TTime	**TTime**(Long_t *msec*)
class TTime	**TTime**(class TTime& *t*)
class TTime	***operator*=**(class TTime& *t*)
class TTime	**operator+=**(class TTime& *t*)
class TTime	**operator-=**(class TTime& *t*)
class TTime	**operator*=**(class TTime& *t*)
class TTime	**TTime**( )
class TTime	**operator/=**(class TTime& *t*)
void	**~TTime**( )
const char*	**AsString**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
long	**operator long**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

Long_t	**fMilliSec**

Class Description

```
TTime

Basic time type with millisecond precision.
```

const char* **AsString**( ) const

Author: Fons Rademakers 28/11/96
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TTimer

class **TTimer** : public TSysEvtHandler

public:

class TTimer	**TTimer**(Long_t *milliSec,* Bool_t *mode* = *kTRUE*)
class TTimer	**TTimer**(class TTimer&)
void	**~TTimer**( )
Bool_t	**CheckTimer**(class TTime& *now*)
class TClass*	**Class**( )
class TTime	**GetAbsTime**( )
class TTime	**GetTime**( )
UInt_t	**GetTimerID**( )
Bool_t	**HasTimedOut**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsAsync**( )
Bool_t	**IsSync**( )
virtual Bool_t	**Notify**( )
virtual void	**Remove**( )
void	**Reset**( )
void	**SetTimerID**(UInt_t *id* = *0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TTime	**fTime**	*time out time in ms*
class TTime	**fAbsTime**	*absolute time out time in ms*
Bool_t	**fTimeout**	*true if timer has timed out*
Bool_t	**fSync**	*true if synchrounous timer*
UInt_t	**fTimeID**	*The System ID of this timed (for WIN32*)

Class Description

```
TTimer

Handles synchronous and a-synchronous timer events. To make use of
this class one has to sub-class TTimer and implement Notify() and
Remove() (if timer has not been added to the gSystem timer list).
Without sub-classing one can use the HasTimedOut() method.
Use Reset() to reset the timer after expiration. To disable a timer
remove it using Remove() or destroy it.
```

TTimer(Long_t *ms,* Bool_t *mode*) : **fTime**(ms)

```
Create timer that times out in ms milliseconds. If mode == kTRUE then
the timer is synchronous else a-synchronous. The default is synchronous.
Add a timer to the system eventloop by doing: gSystem->AddTimer().
```

Bool_t **CheckTimer**(const TTime &*now*)

```
Check if timer timed out.
```

Bool_t **Notify**( )

```
Notify when timer times out
```

void **Remove**( )

```
Remove timer from system timer list. This requires that a timer
has been placed in the system timer list (using gSystem->AddTimer()).
If a TTimer subclass is placed on another list, override Remove() to
remove the timer from the correct list.
```

void **Reset**( )

```
Reset the timer.
```

Author: Fons Rademakers 28/11/96
Last update: 1.00/06 23/03/97 11.01.59 by René Brun

TToggle

class **TToggle** : public TNamed

public:

class TToggle	**TToggle**(class TToggle&)
class TToggle	**TToggle**( )
void	**~TToggle**( )
class TClass*	**Class**( )
virtual Long_t	**GetOffValue**( )
virtual Long_t	**GetOnValue**( )
virtual Bool_t	**GetState**( )
virtual Int_t	**GetValue**( )
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsInitialized**( )
virtual void	**SetOffValue**(Long_t *lof*)
virtual void	**SetOnValue**(Long_t *lon*)
virtual void	**SetState**(Bool_t *state*)
virtual void	**SetToggledObject**(class TObject* *obj,* class TMethod* *anymethod*)
virtual void	**SetToggledVariable**(Int_t& *var*)
virtual void	**SetValue**(Long_t *val*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Toggle**( )

Data Members

private:

Bool_t	**fState**	*object's state - "a local copy"*
Long_t	**fOnValue**	*Value recognized as switched ON (Def=1)*
Long_t	**fOffValue**	*Value recognized as switched OFF(Def=0)*
Long_t	**fValue**	*local copy of a value returned by called function*

protected:

Bool_t	**fInitialized**	*True if either SetToggledObject or SetToggledVariable called - enables Toggle( ) method.*
class TObject*	**fObject**	*The object this Toggle belongs to*
class TMethodCall*	**fGetter**	*Method to Get a value of fObject;*
class TMethodCall*	**fSetter**	*Method to Set a value of fObject;*
Int_t*	**fTglVariable**	*Alternatively*: pointer to an integer value to be Toggled instead of TObjectl

See Also: TMenuToggle

Class Description

```
TToggle

This class defines toggling facility for both - object's method or
variables.
Assume that user provides an object with a two-state field , and
methods to Get/Set value of this field. This object enables to switch
values via this method when the only thing you know about the field
is the name of the method (or method itself) which sets the field.
This facility is required in context Pop-Up menu, when the only
information about how to toggle a field is a name of methhod which
sets it.
This Object may be also used for toggling an integer variable ,
which may be important while building universal objects...
When user provides a "set-method" of name SetXXX this object tries
```

```
automaticaly find a matching "get-method" by lookin for a method
with name GetXXX or IsXXXfor given object.
```

TToggle( )

```
TToggle default constructor. You have to initialize it before using
by making a call to SetToggledVariable() or SetToggledObject()
```

void **SetToggledVariable**(Int_t &*var*)

```
initializes object for use with a variable - you pass it via reference
so it will be modified by Toggle.
```

Bool_t **GetState**( )

```
Returns the state of Toggle according to its current value and
fOnValue ; Returns true if they match.
```

void **SetState**(Bool_t *state*)

```
Sets the value of toggle to fOnValue or fOffValue according to passed
argument.
```

void **SetValue**(Long_t *val*)

```
Sets the value of toggle and modifies its state according to whether
the value is equal to fOnValue.
```

void **Toggle**( )

```
Toggles the Values and State of this object and connected data!
```

void **SetToggledObject**(TObject **obj,* TMethod **anymethod*)

```
Initializes it to toggle an object’s datamember using this object’s method.
```

Author: Piotr Golonka 30/07/97
Last update: 1.03/03 08/09/97 14.11.34 by Piotr Golonka

TToggleGroup

class **TToggleGroup** : public TNamed

public:

class TToggleGroup	**TToggleGroup**( )
class TToggleGroup	**TToggleGroup**(class TToggleGroup&)
void	**~TToggleGroup**( )
virtual Int_t	**Add**(class TToggle* *t,* Bool_t *select = 1*)
virtual class TToggle*	**At**(Int_t *idx*)
class TClass*	**Class**( )
virtual void	**DeleteAll**( )
virtual class TToggle*	**First**( )
virtual Int_t	**GetTogglesCount**( )
virtual Int_t	**IndexOf**(class TToggle* *t*)
virtual Int_t	**InsertAt**(class TToggle* *t,* Int_t *pos,* Bool_t *select = 1*)
virtual class TClass*	**IsA**( )
virtual class TToggle*	**Last**( )
virtual void	**Remove**(class TToggle* *t*)
virtual void	**Remove**(Int_t *pos*)
virtual void	**Select**(class TToggle* *t*)
virtual void	**Select**(Int_t *idx*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TOrdCollection*	**fToggles**	*list of TToggle objects*
class TToggle*	**fSelected**	*currently selected object*

Class Description

```
TToggleGroup

This class defines check-box facility for TToggle objects
It is used in context menu "selectors" for picking up a value.
```

~TToggleGroup( )

```
deletes togglegroup but does not disposes toggled objects!
```

void **DeleteAll**( )

```
disposes of all objects and clears array
```

Author: Piotr Golonka 31/07/97
Last update: 1.03/03 08/09/97 10.32.40 by Piotr Golonka

TTRAP

class **TTRAP** : public TBRIK

public:

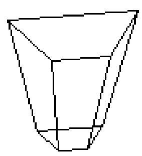
class TTRAP	**TTRAP**( )
class TTRAP	**TTRAP**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dz,* Float_t *theta,* Float_t *phi,* Float_t *h1,* Float_t *bl1,* Float_t *tl1,* Float_t *alpha1,* Float_t *h2,* Float_t *bl2,* Float_t *tl2,* Float_t *alpha2*)
class TTRAP	**TTRAP**(class TTRAP&)
void	**~TTRAP**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fH1**	*half length in y at low z*
Float_t	**fBl1**	*half length in x at low z and y low edge*
Float_t	**fTl1**	*half length in x at low z and y high edge*
Float_t	**fAlpha1**	*angle w.r.t. the y axis*
Float_t	**fH2**	*half length in y at high z*
Float_t	**fBl2**	*half length in x at high z and y low edge*
Float_t	**fTl2**	*half length in x at high z and y high edge*
Float_t	**fAlpha2**	*angle w.r.t. the y axis*

Class Description



```
TRAP is a general trapezoid. The faces perpendicular to z are trapezia and
their centres are not necessarily on a line parallel to the z axis.
This shape has 14 parameters.

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dz         half-length along the z axis
  - theta      polar angle of the line joining the centre of the face
               at -DZ to the centre of the one at +DZ
  - phi        azimuthal angle of the line joining the centre of the face
               at -DZ to the centre of the one at +DZ
  - h1         half-length along y of the face at -DZ
  - bl1        half-length along x of the side at -H1 in y of the face
               at -DZ in z
  - tl1        half-length along x of the side at +H1 in y of the face
               at -DZ in z
  - alpha1     angle with respect to the y axis from the centre of the
               side at -H1 in y to the centre of the side at +H1 in y
               of the face at -DZ in z
  - h2         half-length along y of the face at +DZ
  - bl2        half-length along x of the side at -H2 in y of the
```

```
                  face at +DZ in z
      - tl2       half-length along x of the side at +H2 in y of the face
                  at +DZ in z
      - alpha2    angle with respect to the y axis from the centre of the side
                  at -H2 in y to the centre of the side at +H2 in y of the
                  face at +DZ in z
```

TTRAP( )

    TRAP shape default constructor

TTRAP(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dz,* Float_t *theta,* Float_t *phi,*Float_t h1, Float_t bl1, Float_t tl1, Float_t alpha1, Float_t h2,Float_t *bl2,* Float_t *tl2,* Float_t *alpha2*) : **TBRIK**(*name, title, material, theta, phi,* dz)

    TRAP shape normal constructor

~TTRAP( )

    TRAP shape default destructor

void **SetPoints**(Float_t **buff*)

    Create TRAP points

Author: Nenad Buncic 19/09/95
Last update: 1.00/07 27/03/97 17.56.06 by René Brun

TTRD1

class **TTRD1** : public TBRIK

public:

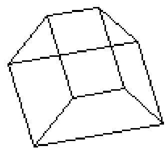
class TTRD1	**TTRD1**( )
class TTRD1	**TTRD1**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dx1,* Float_t *dx2,* Float_t *dy,* Float_t *dz*)
class TTRD1	**TTRD1**(class TTRD1&)
void	**~TTRD1**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fDx2**	*half length in x at the high z surface*

Class Description



```
TRD1 is a trapezoid with the x dimension varying along z.
It has 7 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dx1        half-length along x at the z surface positioned at -DZ
  - dx2        half-length along x at the z surface positioned at +DZ
  - dy         half-length along the y-axis
  - dz         half-length along the z-axis
```

TTRD1( )
    TRD1 shape default constructor

TTRD1(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dx1,* Float_t *dx2,* Float_t *dy,* Float_t *dz*): **TBRIK**(*name, title, material, dx1, dy, dz*)
    TRD1 shape normal constructor

~TTRD1( )
    TRD1 shape default destructor

void **SetPoints**(Float_t **buff*)
    Create TRD1 points

Author: Nenad Buncic 17/09/95
Last update: 1.00/06 23/03/97 11.26.52 by René Brun

TTRD2

class **TTRD2** : public TBRIK

public:

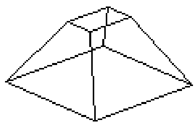
class TTRD2	**TTRD2**( )
class TTRD2	**TTRD2**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *dx1,* Float_t *dx2,* Float_t *dy1,* Float_t *dy2,* Float_t *dz*)
class TTRD2	**TTRD2**(class TTRD2&)
void	**~TTRD2**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Float_t	**fDx2**	*half length in x at the high z surface*
Float_t	**fDy2**	*half length in y at the high z surface*

Class Description



```
TRD2 is a trapezoid with both x and y dimensions varying along z.
It has 8 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - dx1        half-length along x at the z surface positioned at -DZ
  - dx2        half-length along x at the z surface positioned at +DZ
  - dy1        half-length along y at the z surface positioned at -DZ
  - dy2        half-length along y at the z surface positioned at +DZ
  - dz         half-length along the z-axis
```

TTRD2( )

`TRD2 shape default constructor`

TTRD2(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *dx1,* Float_t *dx2,* Float_t *dy1,*Float_t *dy2,* Float_t *dz*) : **TBRIK**(*name, title, material, dx1, dy1,* dz)

`TRD2 shape normal constructor`

~TTRD2( )

`TRD2 shape default destructor`

void **SetPoints**(Float_t **buff*)

`Create TRD2 points`

Author: Nenad Buncic 13/09/95
Last update: 1.00/06 23/03/97 11.26.52 by René Brun

TTree

class **TTree** : public TNamed, public TAttLine, public TAttFill, public TAttMarker

protected:

const Text_t*	**GetNameByIndex**(class TString& *varexp,* Int_t* *index,* Int_t *colindex*)
virtual Int_t	**MakeChopt**(Option_t* *option*)
virtual Int_t	**MakeFitOptions**(Option_t* *option*)
virtual void	**MakeIndex**(class TString& *varexp,* Int_t* *index*)
virtual void	**SetDirectory**(class TDirectory* *dir*)
void	**TakeAction**(Int_t *nfill,* Int_t& *npoints,* Int_t& *action,* class TObject* *obj,* Option_t* *option*)
void	**TakeEstimate**(Int_t *nfill,* Int_t& *npoints,* Int_t *action,* class TObject* *obj,* Option_t* *option*)

public:

class TTree	**TTree**(Text_t* *name,* Text_t* *title,* Int_t *maxvirtualsize = 0*)
class TTree	**TTree**( )
class TTree	**TTree**(class TTree&)
void	**~TTree**( )
virtual void	**AddTotBytes**(Int_t *tot*)
virtual void	**AddZipBytes**(Int_t *zip*)
virtual void	**AutoSave**( )
class TBranch*	**Branch**(Text_t* *name,* void* *address,* Text_t* *leaflist,* Int_t *bufsize = 32000*)
class TBranch*	**Branch**(Text_t* *name,* Text_t* *classname,* void* *addobj,* Int_t *bufsize = 32000,* Int_t *splitlevel = 1*)
class TBranch*	**Branch**(Text_t* *name,* class TClonesArray* *list,* Int_t *bufsize = 32000,* Int_t *splitlevel = 1*)
virtual void	**Browse**(class TBrowser* *b*)
class TClass*	**Class**( )
virtual void	**ClearFormula**( )
virtual class TTree*	**CloneTree**(Int_t *nevents = 0*)
virtual void	**CompileVariables**(Text_t* *varexp,* Text_t* *selection*)
virtual void	**CreatePacketGenerator**(Int_t *nevents,* Stat_t *firstEvent*)
virtual void	**Draw**(class TCut *varexp,* class TCut *selection,* Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual void	**Draw**(Text_t* *varexp,* Text_t* *selection,* Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
void	**DropBuffers**(Int_t *nbytes*)
virtual void	**EstimateLimits**(Int_t *estimate,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual void	**EventLoop**(Int_t& *action,* class TObject* *obj,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0,* Option_t* *option*)
virtual Int_t	**Fill**( )
void	**FindGoodLimits**(Int_t *nbins,* Int_t& *newbins,* Float_t& *xmin,* Float_t& *xmax*)
virtual void	**Fit**(Text_t* *formula,* Text_t* *varexp,* Text_t* *selection,* Option_t* *option,* Option_t* *goption,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual class TBranch*	**GetBranch**(Text_t* *name*)
class TFile*	**GetCurrentFile**( )
class TDirectory*	**GetDirectory**( )
virtual Stat_t	**GetEntries**( )
virtual Int_t	**GetEstimate**( )
virtual Int_t	**GetEvent**(Int_t *event = 0,* Int_t *getall = 0*)
class TEventList*	**GetEventList**( )
virtual class TLeaf*	**GetLeaf**(Text_t* *name*)
virtual class TList*	**GetListOfActiveBranches**( )

virtual class TObjArray*	**GetListOfBranches**( )
virtual class TObjArray*	**GetListOfLeaves**( )
virtual Int_t	**GetMaxEventLoop**( )
virtual Float_t	**GetMaximum**(Text_t* *columname*)
virtual Int_t	**GetMaxVirtualSize**( )
virtual Float_t	**GetMinimum**(Text_t* *columname*)
class TTreeFormula*	**GetMultiplicity**( )
virtual Int_t	**GetNbranches**( )
virtual void	**GetNextPacket**(class TSlave* *sl,* Int_t& *nevents,* Stat_t& *firstevent,* Stat_t& *processed*)
virtual Int_t	**GetOldEventNumber**( )
class TPacketGenerator*	**GetPacketGenerator**( )
virtual Int_t	**GetPacketSize**( )
virtual Int_t	**GetReadEvent**( )
virtual Int_t	**GetScanField**( )
class TTreeFormula*	**GetSelect**( )
virtual Int_t	**GetSelectedRows**( )
class TSelector*	**GetSelector**( )
virtual Stat_t	**GetTotBytes**( )
class TTreeFormula*	**GetVar1**( )
class TTreeFormula*	**GetVar2**( )
class TTreeFormula*	**GetVar3**( )
class TTreeFormula*	**GetVar4**( )
virtual Stat_t	**GetZipBytes**( )
virtual void	**IncrementTotalBuffers**(Int_t *nbytes*)
virtual class TClass*	**IsA**( )
Bool_t	**IsFolder**( )
virtual Int_t	**LoadTree**(Int_t *event*)
virtual void	**Loop**(Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual Int_t	**MakeCode**(char* *filename = 0*)
Bool_t	**MemoryFull**(Int_t *nbytes*)
virtual void	**Print**(Option_t* *option*)
virtual void	**Project**(Text_t* *hname,* Text_t* *varexp,* Text_t* *selection,* Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual void	**Reset**(Option_t* *option*)
virtual void	**Scan**(Text_t* *varexp,* Text_t* *selection,* Option_t* *option,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual class TEventList*	**Select**(Text_t* *selection,* Int_t *nevents = 1000000000,* Int_t *firstevent = 0*)
virtual void	**SetAutoSave**(Int_t *autosave = 10000000*)
virtual void	**SetBranchAddress**(Text_t* *bname,* void* *add*)
virtual void	**SetBranchStatus**(Text_t* *bname,* Bool_t *status = 1*)
virtual void	**SetEstimate**(Int_t *nevents = 10000*)
virtual void	**SetEventList**(class TEventList* *list*)
virtual void	**SetMaxEventLoop**(Int_t *maxev = 1000000000*)
virtual void	**SetMaxVirtualSize**(Int_t *size = 0*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetPacketSize**(Int_t *size = 100*)
virtual void	**SetScanField**(Int_t *n = 50*)
virtual void	**SetSelector**(class TSelector* *selector = 0*)
virtual void	**SetSelector**(Text_t* *macroname*)
virtual void	**SetUpdate**(Int_t *freq = 0*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Int_t	**fScanField**	*Number of runs before prompting in Scan*
Int_t	**fDraw**	*Last event loop number when object was drawn*

Int_t	**fUpdate**	*Update frequency for EventLoop*
Int_t	**fMaxEventLoop**	*Maximum number of events to process*
Int_t	**fMaxVirtualSize**	*Maximum total size of buffers kept in memory*
Int_t	**fAutoSave**	*Autosave tree when fAutoSave bytes produced*
Stat_t	**fEntries**	*Number of entries*
Stat_t	**fTotBytes**	*Total number of bytes in all branches before compression*
Stat_t	**fZipBytes**	*Total number of bytes in all branches after compression*
Stat_t	**fSavedBytes**	*Number of autosaved bytes*
class TTreeFormula*	**fVar1**	*Pointer to first variable formula*
class TTreeFormula*	**fVar2**	*Pointer to second variable formula*
class TTreeFormula*	**fVar3**	*Pointer to third variable formula*
class TTreeFormula*	**fVar4**	*Pointer to fourth variable formula*
class TTreeFormula*	**fSelect**	*Pointer to selection formula*
class TTreeFormula*	**fMultiplicity**	*Pointer to formula giving ndata per event*
Int_t	**fOldEventNumber**	*Number of the previously read event*
Int_t	**fReadEvent**	*Number of the event being processed*
Int_t	**fTotalBuffers**	*Total number of bytes in branch buffers*
Int_t	**fEstimate**	*Number of events to estimate histogram limits*
Int_t	**fDimension**	*Dimension of the current expression*
Int_t	**fSelectedRows**	*Number of selected events*
Int_t	**fPacketSize**	*Number of events in one packet for parallel root*
Int_t	**fNbins**	*Number of bins per dimension*
Float_t	**fVmin**	*Minima of varexp columns*
Float_t	**fVmax**	*Maxima of varexp columns*
Float_t*	**fV1**	*Local buffer for variable 1*
Float_t*	**fV2**	*Local buffer for variable 2*
Float_t*	**fV3**	*Local buffer for variable 3*
Double_t*	**fW**	*Local buffer for weights*
class TDirectory*	**fDirectory**	*Pointer to directory holding this tree*
class TObjArray	**fBranches**	*List of Branches*
class TObjArray	**fLeaves**	*Direct pointers to individual branch leaves*
class TList*	**fActiveBranches**	*List of active branches*
class TEventList*	**fEventList**	*Pointer to event selection list (if one)*
class TSelector*	**fSelector**	*Pointer to current selector*
class TPacketGenerator*	**fPacketGen**	*Packet generator*
Int_t	**fNfill**	*Local for EventLoop*

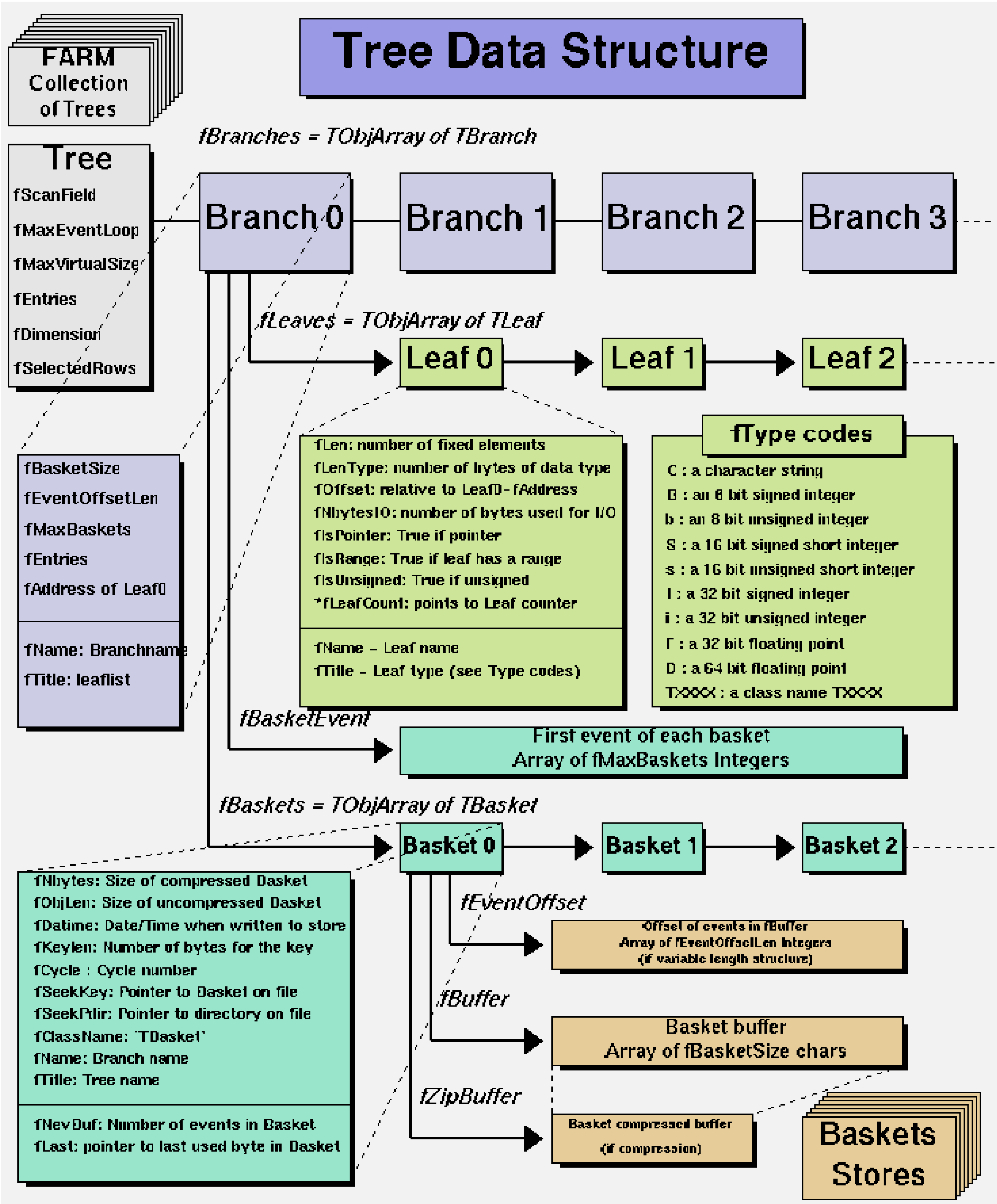
See Also: TChain, TNtuple

Class Description

```
TTree

a TTree object has a header with a name and a title.
It consists of a list of independent branches (TBranch). Each branch
has its own definition and list of buffers. Branch buffers may be
automatically written to disk or kept in memory until the Tree attribute
fMaxVirtualSize is reached.
Variables of one branch are written to the same buffer.
A branch buffer is automatically compressed if the file compression
attribute is set (default).

The ROOT user can decide to make one single branch and serialize one
object into one single I/O buffer or to make several branches.
Making one single branch and one single buffer can be the right choice
when one wants to process only a subset of all entries in the tree.
(you know for example the list of event numbers you want to process).
Making several branches is particularly interesting in the data analysis
```

```
phase, when one wants to histogram some attributes of an object (event)
without reading all the attributes.

==> TTree *tree = new TTree(name, title, maxvirtualsize)
  Creates a Tree with name and title. Maxvirtualsize is by default 64Mbytes,
  maxvirtualsize = 64000000(default) means: Keeps as many buffers in memory
until
  the sum of all buffers is greater than 64 Megabyte. When this happens,
  memory buffers are written to disk and deleted until the size of all
  buffers is again below the threshold.
  maxvirtualsize = 0 means: keep only one buffer in memory.

  Various kinds of branches can be added to a tree:
    A - simple structures or list of variables. (may be for C or Fortran struc-
tures)
    B - any object (inheriting from TObject). (we expect this option be the most
frequent)
    C - a ClonesArray. (a specialized object for collections of same class
objects)

==> Case A
  TBranch *branch = tree->Branch(branchname,address, leaflist, bufsize)
    * address is the address of the first item of a structure
    * leaflist is the concatenation of all the variable names and types
      separated by a colon character :
      The variable name and the variable type are separated by a slash (/).
      The variable type may be 0,1 or 2 characters. If no type is given,
      the type of the variable is assumed to be the same as the previous
      variable. If the first variable does not have a type, it is assumed
      of type F by default. The list of currently supported types is given
below:
         - C : a character string terminated by the 0 character
         - B : an 8 bit signed integer (Char_t)
         - b : an 8 bit unsigned integer (UChar_t)
         - S : a 16 bit signed integer (Short_t)
         - s : a 16 bit unsigned integer (UShort_t)
         - I : a 32 bit signed integer (Int_t)
         - i : a 32 bit unsigned integer (UInt_t)
         - F : a 32 bit floating point (Float_t)
         - D : a 64 bit floating point (Double_t)

==> Case B
  TBranch *branch = tree->Branch(branchname,object, bufsize, splitlevel)
       object is a pointer to an existing object (derived from TObject).
     if splitlevel=1 (default), this branch will automatically be split
       into subbranches, with one subbranch for each data member or object
       of the object itself. In case the object member is a TClonesArray,
       the mechanism described in case C is applied to this array.
     if splitlevel=0, the object is serialized in the branch buffer.

==> Case C
  TBranch *branch = tree->Branch(branchname,clonesarray, bufsize, splitlevel)
      clonesarray is a pointer to a TClonesArray.
      The TClonesArray is a direct access list of objects of the same class.
      For example, if the TClonesArray is an array of TTrack objects,
      this function will create one subbranch for each data member of
      the object TTrack.

==> branch->SetAddress(Void *address)
   In case of dynamic structures changing with each event for example, one must
   redefine the branch address before filling the branch again.
   This is done via the TBranch::SetAddress member function.
```

```
==> tree->Fill()
   loops on all defined branches and for each branch invokes the Fill function.

      See also the class TNtuple (a simple Tree with only one branch)
```




```
A simple example with histograms and a tree
This program creates :
 - a one dimensional histogram
 - a two dimensional histogram
 - a profile histogram
 - a tree
```

These objects are filled with some random numbers and saved on a file.

```
#include "TROOT.h"
#include "TFile.h"
#include "TH1.h"
#include "TH2.h"
#include "TProfile.h"
#include "TRandom.h"
#include "TTree.h"

int Error;

TROOT simple("simple","Histograms and trees");

//
______________________________________________________________________________
main(int argc, char **argv)
{
// Create a new ROOT binary machine independent file.
// Note that this file may contain any kind of ROOT objects, histograms,trees
// pictures, graphics objects, detector geometries, tracks, events, etc..
// This file is now becoming the current directory.
TFile hfile("htree.root","RECREATE","Demo ROOT file with histograms & trees");

// Create some histograms and a profile histogram
TH1F *hpx   = new TH1F("hpx","This is the px distribution",100,-4,4);
TH2F *hpxpy = new TH2F("hpxpy","py ps px",40,-4,4,40,-4,4);
TProfile *hprof = new TProfile("hprof","Profile of pz versus px",100,-
4,4,0,20);

// Define some simple structures
typedef struct {Float_t x,y,z;} POINT;
typedef struct {
   Int_t ntrack,nseg,nvertex;
   UInt_t flag;
   Float_t temperature;
} EVENTN;
static POINT point;
static EVENTN eventn;

// Create a ROOT Tree
TTree *tree = new TTree("T","An example of ROOT tree with a few branches");
tree->Branch("point",&point,"x:y:z");
tree->Branch("eventn",&eventn,"ntrack/I:nseg:nvertex:flag/i:temperature/F");
tree->Branch("hpx","TH1F",&hpx,128000,0);

Float_t px,py,pz;
static Float_t p[3];

//--------------------Here we start a loop on 1000 events
for ( Int_t i=0; i<1000; i++) {
   gRandom->Rannor(px,py);
   pz = px*px + py*py;
   Float_t random = gRandom->::Rndm(1);

//         Fill histograms
   hpx->Fill(px);
   hpxpy->Fill(px,py,1);
   hprof->Fill(px,pz,1);

//         Fill structures
   p[0] = px;
   p[1] = py;
```

```
      p[2] = pz;
      point.x = 10*(random-1);;
      point.y = 5*random;
      point.z = 20*random;
      eventn.ntrack  = Int_t(100*random);
      eventn.nseg    = Int_t(2*eventn.ntrack);
      eventn.nvertex = 1;
      eventn.flag    = Int_t(random+0.5);
      eventn.temperature = 20+random;

   //        Fill the tree. For each event, save the 2 structures and 3 objects
   //      In this simple example, the objects hpx, hprof and hpxpy are slightly
   //      different from event to event. We expect a big compression factor!
      tree->Fill();
   }
   //--------------End of the loop

   tree->Print();

   // Save all objects in this file
   hfile.Write();

   // Close the file. Note that this is automatically done when you leave
   // the application.
   hfile.Close();

   return 0;
   }
```

TTree( ): **TNamed**( )

```
Default Tree constructor
```

TTree(const Text_t **name,* const Text_t **title,* Int_t *maxvirtualsize*):**TNamed**(*name, title*)

```
Normal Tree constructor
```

~**TTree**( )

```
Tree destructor
```

void **AutoSave**( )

```
AutoSave tree header every fAutoSave bytes
When large Trees are produced, it is safe to activate the AutoSave
procedure. Some branches may have buffers holding many events.
AutoSave is automatically called by TTree::Fill when the number of bytes
generated since the previous AutoSave is greater than fAutoSave bytes.
This function may also be invoked by the user, for example every
N events.
Each AutoSave generates a new key on the file.

In case your program crashes before closing the file holding this tree,
the file will be automatically recovered when you will connect the file
in UPDATE mode.
The Tree will be recovered at the status corresponding to the last AutoSave.
```

TBranch* **Branch**(const Text_t **name,* void **address,* const Text_t **leaflist,* Int_t *bufsize*)

```
Create a new TTree Branch
  This Branch constructor is provided to support non-objects in
  a Tree. The variables described in leaflist may be simple variables
  or structures.
 See the two following constructors for writing objects in a Tree.
```

TBranch* **Branch**(const Text_t **name,* const Text_t **classname,* void **addobj,* Int_t *bufsize,* Int_t *splitlevel*)

```
Create a new TTree BranchObject
 Build a TBranchObject for an object of class classname.
```

```
addobj is the address of a pointer to an object of class classname.

This option requires access to the library where the corresponding class
is defined. Accessing one single data member in the object implies
reading the full object.
See the next Branch constructor for a more efficient storage
in case the event consists of arrays of identical objects.
```

TBranch* **Branch**(const Text_t **name,* TClonesArray **list,* Int_t *bufsize,* Int_t *splitlevel*)

```
Create a new TTree BranchClones
 name:    global name of this BranchClones
 list:    pointer to a TClonesArray
 bufsize: buffersize in bytes of each individual data member buffer

 This Tree option is provided in case each event consists of one
 or more arrays of same class objects (tracks, hits,etc).
 This function creates as many branches as there are public data members
 in the objects pointed by the TClonesArray.

 BranchClones have the following advantages compared to the two other
 solutions (Branch and BranchObject).
   - When reading the Tree data, it is possible to read selectively
     a subset of one object (may be just one single data member).
   - This solution minimizes the number of objects created/destructed.
   - Data members of the same type are consecutive in the basket buffers,
     therefore optimizing the compression algorithm.
   - Array processing notation becomes possible in the query language.
```

void **ClearFormula**( )

```
Delete internal buffers
```

TTree* **CloneTree**(Int_t *nevents*)

```
Create a clone of this tree and copy nevents
By default nevents = 0
```

void **CompileVariables**(const Text_t **varexp,* const Text_t **selection*)

```
Compile input variables and selection expression
varexp is an expression of the general form e1:e2:e3
 where e1,etc is a formula referencing a combination of the columns
Example:
  varexp = x     simplest case: draw a 1-Dim distribution of column named x
         = sqrt(x)            : draw distribution of sqrt(x)
         = x*y/z
         = y:sqrt(x) 2-Dim dsitribution of y versus sqrt(x)

selection is an expression with a combination of the columns
Example:
   selection = "x<y && sqrt(z)>3.2"
    in a selection all the C++ operators are authorized
```

void **CreatePacketGenerator**(Int_t *nevents,* Stat_t *firstEvent*)

```
Create or reset the packet generator.
```

void **Draw**(TCut *varexp,* TCut *selection,* Option_t **option,* Int_t *nevents,* Int_t *firstevent*)

```
Draw expression varexp for specified events
   This function accepts TCut objects as arguments.
   Useful to use the string operator +
      example:
         ntuple.Draw("x",cut1+cut2+cut3);
```

void **Draw**(const Text_t **varexp0,* const Text_t **selection,* Option_t **option,* Int_t *nevents,* Int_t *firstevent*)

```
Draw expression varexp for specified events
varexp is an expression of the general form e1:e2:e3
```

```
 where e1,etc is a formula referencing a combination of the columns
Example:
  varexp = x     simplest case: draw a 1-Dim distribution of column named x
         = sqrt(x)            : draw distribution of sqrt(x)
         = x*y/z
         = y:sqrt(x) 2-Dim dsitribution of y versus sqrt(x)

selection is an expression with a combination of the columns
Example:
   selection = "x<y && sqrt(z)>3.2"
    in a selection all the C++ operators are authorized

option is the drawing option
   see TH1::Draw for the list of all drawing options.
   If option contains the string "goff", no graphics is generated.

nevents is the number of events to process (default is all)
first is the first event to process (default is 0)

By default the temporary histogram created is called htemp.
If varexp contains >>hnew , the new histogram created is called hnew
and it is kept in the current directory.
Example:
 tree.Draw("sqrt(x)>>hsqrt","y>0")
 will draw sqrt(x) and save the histogram as "hsqrt" in the current
 directory.

By default, the specified histogram is reset.
To continue to append data to an existing histogram, use "+" in front
of the histogram name;
 tree.Draw("sqrt(x)>>+hsqrt","y>0")
   will not reset hsqrt, but will continue filling.
```

void **DropBuffers**(Int_t *nbytes*)

```
Drop branch buffers to accomodate nbytes below MaxVirtualsize
```

void **EstimateLimits**(Int_t *estimate,* Int_t *nevents,* Int_t *firstevent*)

```
Estimate histogram limits for conditions of input parameters
The estimation of the limits is based on estimate events
The first estimate/2 and the last esstimate/2 events are used.
This algorithm is in general good enough to make a reasonable estimation
with a small number of events.

nevents is the number of events to process (default is all)
first is the first event to process (default is 0)
```

void **EventLoop**(Int_t *&action,* TObject **obj,* Int_t *nevents,* Int_t *firstevent,* Option_t **option*)

```
Loop on all events
nevents is the number of events to process (default is all)
first is the first event to process (default is 0)

action =  1  Fill 1-D histogram obj
      =  2  Fill 2-D histogram obj
      =  3  Fill 3-D histogram obj
      =  4  Fill Profile histogram obj
      = 11  Estimate Limits
      = 12  Fill 2-D PolyMarker obj
      = 13  Fill 3-D PolyMarker obj
action < 0   Evaluate Limits for case abs(action)
```

Int_t **Fill**( )

```
Fill all branches of a Tree
This function loops on all the branches of this tree.
```

```
For each branch, it copies to the branch buffer (basket) the current
values of the leaves data types.
If a leaf is a simple data type, a simple conversion to a machine
independent format has to be done.
```

void **FindGoodLimits**(Int_t *nbins,* Int_t &*newbins,* Float_t &*xmin,* Float_t &*xmax*)

```
Find reasonable bin values
```

void **Fit**(const Text_t **formula,* const Text_t **varexp,* const Text_t **selection,* Option_t **option,* Option_t **goption,* Int_t *nevents,* Int_t *firstevent*)

```
Fit  a projected item(s) from a Tree
formula is a TF1 expression.

See TTree::Draw for explanations of the other parameters.

By default the temporary histogram created is called htemp.
If varexp contains >>hnew , the new histogram created is called hnew
and it is kept in the current directory.
Example:
 tree.Fit(pol4,sqrt(x)>>hsqrt,y>0)
 will fit sqrt(x) and save the histogram as "hsqrt" in the current
 directory.
```

TBranch* **GetBranch**(const Text_t **name*)

```
Return pointer to the branch with name
```

TFile* **GetCurrentFile**( )

```
Return pointer to the current file
```

Int_t **GetEvent**(Int_t *event,* Int_t *getall*)

```
Read all branches of event and return total number of bytes
  If event = 0 take current event number + 1
  If event < 0 reset event number to 0
  getall = 0 : get only active branches
  getall = 1 : get all branches
```

TLeaf* **GetLeaf**(const Text_t **name*)

```
Return pointer to the 1st Leaf named name in any Branch
```

Float_t **GetMaximum**(Text_t **columname*)

```
Return maximum of column with name columname
```

Float_t **GetMinimum**(Text_t **columname*)

```
Return minimum of column with name columname
```

const Text_t* **GetNameByIndex**(TString &*varexp,* Int_t **index,* Int_t *colindex*)

```
Return name corresponding to colindex in varexp
varexp is a string of names separated by :
index is an array with pointers to the start of name[i] in varexp
```

void **GetNextPacket**(TSlave **sl,* Int_t &*nevents,* Stat_t &*firstevent,* Stat_t &*processed*)

```
Return in nevents and firstevent the optimal range of events (packet)
to be processed by slave sl. See TPacketGenerator for the algorithm
used to get the packet size.
```

TSelector* **GetSelector**( )

```
Return current selector or create default selector
```

Int_t **LoadTree**(Int_t *event*)

```
Set current Tree event
```

void **Loop**(Option_t **option,* Int_t *nevents,* Int_t *firstevent*)

```
Loop on nevents of this tree starting at firstevent
```

Int_t **MakeChopt**(Option_t **option*)
```
Fill Hoption structure with the codes in option
```

Int_t **MakeCode**(const char **filename*)
```
Generate skeleton function for this Tree
The function code is written on filename
if filename is NULL, filename will be nameoftree.C

The generated code includes the following:
   - Identification of the original Tree and Input file name
   - Connection of the Tree file
   - Declaration of Tree variables
   - Setting of branches addresses
   - a skeleton for the event loop

To use this function:
   - connect your Tree file (eg: TFile f("myfile.root");)
   - T->MakeCode("anal.C");
 where T is the name of the Tree in file myfile.root
 and anal.C the name of the file created by this function.

       Author: René Brun
```

Int_t **MakeFitOptions**(Option_t **option*)
```
Fill Foption structure with the codes in option
```

void **MakeIndex**(TString &*varexp,* Int_t **index*)
```
Build Index array for names in varexp
```

Bool_t **MemoryFull**(Int_t *nbytes*)
```
Check if adding nbytes to memory we are still below MaxVirtualsize
```

void **Print**(Option_t **option*)
```
Print a summary of the Tree contents. In case options are "p" or "pa"
print information about the TPacketGenerator ("pa" is equivalent to
TPacketGenerator::Print("all")).
```

void **Project**(Text_t **hname,* Text_t **varexp,* Text_t **selection,* Option_t **option,* Int_t *nevents,* Int_t *firstevent*)
```
Make a projection of a Tree using selections
Depending on the value of varexp (described in Draw) a 1-D,2-D,etc
projection of the Tree will be filled in histogram hname.
Note that the dimension of hname must match with the dimension of varexp.
```

void **Reset**(Option_t **option*)
```
Reset buffers and entries count in all branches/leaves
```

void **Scan**(const Text_t **varexp,* const Text_t **selection,* Option_t **option,* Int_t *nevents,* Int_t *firstevent*)
```
Loop on Tree & print events following selection
```

TEventList* **Select**(const Text_t **selection,* Int_t *nevents,* Int_t *firstevent*)
```
Return EventList corresponding to selection
```

void **SetBranchAddress**(const Text_t **bname,* void **add*)
```
Set branch address
   If object is a TTree, this function is only an interface to TBranch::SetAd-
dress
   Function overloaded by TChain.
```

void **SetBranchStatus**(const Text_t **bname,* Bool_t *status*)
```
Set branch status Process or DoNotProcess
   bname is the name of a branch. if bname="*", apply to all branches.
   status = 1  branch will be processed
          = 0  branch will not be processed
```

void **SetDirectory**(TDirectory **dir*)

```
Remove reference to this tree from current directory and add
reference to new directory dir. dir can be 0 in which case the tree
does not belong to any directory.
```

void **SetEstimate**(Int_t *n*)

```
Set number of events to estimate variable limits
```

void **SetName**(const Text_t **name*)

```
Change the name of this Tree
```

void **SetPacketSize**(Int_t *size*)

```
Set number of events per packet for parallel root
```

void **SetSelector**(TSelector **selector*)

```
Set current selector to user selector
```

void **SetSelector**(const Text_t **macroname*)

```
Set current selector to macroname
```

void **Streamer**(TBuffer &*b*)

```
Stream a class object
```

void **TakeAction**(Int_t *nfill,* Int_t &*npoints,* Int_t &*action,* TObject **obj,* Option_t **option*)

```
Execute action for object obj nfill times
```

void **TakeEstimate**(Int_t *nfill,* Int_t &*npoints,* Int_t *action,* TObject **obj,* Option_t **option*)

```
Estimate limits for 1-D, 2-D or 3-D objects
```

Author: René Brun 12/01/96
Last update: 07/11/97 10.48.51 by René Brun

TTreeFormula

class **TTreeFormula** : public TFormula

public:

class TTreeFormula	**TTreeFormula**( )
class TTreeFormula	**TTreeFormula**(char* *name,* char* *formula,* class TTree* *tree*)
class TTreeFormula	**TTreeFormula**(class TTreeFormula&)
void	**~TTreeFormula**( )
class TClass*	**Class**( )
virtual Int_t	**DefinedVariable**(class TString& *variable*)
virtual Double_t	**EvalInstance**(Int_t *i = 0*)
virtual class TLeaf*	**GetLeaf**(Int_t *n*)
class TMethodCall*	**GetMethodCall**(Int_t *code*)
virtual Int_t	**GetMultiplicity**( )
virtual Int_t	**GetNdata**( )
virtual class TClass*	**IsA**( )
virtual char*	**PrintValue**(Int_t *mode = 0*)
virtual void	**SetTree**(class TTree* *tree*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

class TTree*	**fTree**	*pointer to Tree*
Short_t	**fCodes**	*List of leaf numbers referenced in formula*
Int_t	**fNcodes**	*Number of leaves referenced in formula*
Int_t	**fMultiplicity**	*Number of array elements in leaves in case of a TClonesArray*
Int_t	**fInstance**	*Instance number for GetValue*
Int_t	**fNindex**	*Size of fIndex*
Int_t*	**fIndex**	*array of instances numbers*
class TObjArray	**fMethods**	*List of leaf method calls*

Class Description

```
  A TreeFormula is used to pass a selection expression
  to the Tree drawing routine. See TTree::Draw

A TreeFormula can contain any arithmetic expression including
standard operators and mathematical functions separated by operators.
Examples of valid expression:
       "x<y && sqrt(z)>3.2"
```

TTreeFormula( ): **TFormula**( )

```
Tree Formula default constructor
```

TTreeFormula(const char **name,* const char **expression,* TTree **tree*):**TFormula**( )

```
Normal Tree Formula constructor
```

~TTreeFormula( )

```
Tree Formula default destructor
```

Int_t **DefinedVariable**(TString &*name*)

```
Check if name is in the list of Tree/Branch leaves
This member function redefines the function in TFormula
If a leaf has a name corresponding to the argument name, then
returns a new code.
A TTreeFormula may contain more than one variable.
```

```
For each variable referenced, the pointers to the corresponding
branch and leaf is stored in the object arrays fBranches and fLeaves.

name can be :
   - Leaf_Name (simple variable or data member of a ClonesArray)
   - Branch_Name.Leaf_Name
   - Branch_Name.Method_Name
```

Double_t **EvalInstance**(Int_t *instance*)

```
Evaluate this treeformula
```

TLeaf* **GetLeaf**(Int_t *n*)

```
Return leaf corresponding to serial number n
```

TMethodCall* **GetMethodCall**(Int_t *code*)

```
Return methodcall corresponding to code
function called by TLeafObject::GetValue
with the value of fIndex computed in TTreeFormula::Definedvariable
```

Int_t **GetNdata**( )

```
Return number of data words in the leaf
```

char* **PrintValue**(Int_t *mode*)

```
Return value of variable as a string
   mode = -2 : Print line with **
   mode = -1 : Print column names
   mode = 0  : Print column values
```

void **Streamer**(TBuffer &*R__b*)

```
Stream an object of class TTreeFormula.
```

Author: René Brun 19/01/96
Last update: 03/11/97 15.55.56 by René Brun

TTUBE

class **TTUBE** : public TShape

protected:

virtual void	**MakeTableOfCoSin**( )
virtual void	**PaintGLPoints**(Float_t* *vertex*)

public:

class TTUBE	**TTUBE**( )
class TTUBE	**TTUBE**(Text_t* *name,* Text_t* *title,* Text_t* *material,* Float_t *rmin,* Float_t *rmax,* Float_t *dz*)
class TTUBE	**TTUBE**(class TTUBE&)
void	**~TTUBE**( )
class TClass*	**Class**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual Int_t	**GetNumberOfDivisions**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SetNumberOfDivisions**(Int_t *p*)
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

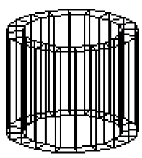
Data Members

protected:

Float_t	**fRmin**	*inside radius*
Float_t	**fRmax**	*outside radius*
Float_t	**fDz**	*half length in z*
Int_t	**fNdiv**	*number of segments (precision*)
Double_t*	**fSiTab**	*Table of sin(fPhi1*) .... sin(fPhil+fDphi1)
Double_t*	**fCoTab**	*Table of cos(fPhi1*) .... cos(fPhil+fDphi1)

See Also: TCONE, TELTU, TTUBS

Class Description



```
TUBE is a tube. It has 6 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - rmin       inside radius
  - rmax       outside radius
  - dz         half length in z
```

TTUBE( )

```
TUBE shape default constructor
```

TTUBE(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rmin,* Float_t *rmax,* Float_t *dz*): **TShape**(*name, title, material*)

```
TUBE shape normal constructor
```

~TTUBE( )

```
TUBE shape default destructor
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a TUBE
Compute the closest distance of approach from point px,py to each
computed outline point of the TUBE.
```

void **Paint**(Option_t **option*)

```
Paint this 3-D shape with its current attributes
```

void **PaintGLPoints**(Float_t **vertex*)

```
Paint BRIK via OpenGL
```

void **SetPoints**(Float_t **buff*)

```
Create TUBE points
```

void **Sizeof3D**( ) const

```
Return total X3D size of this shape with its attributes
```

Author: Nenad Buncic 18/09/95
Last update: 1.01/05 04/11/97 15.37.06 by Fons Rademakers

TTUBS

class **TTUBS** : public TTUBE

protected:

virtual void	**MakeTableOfCoSin**( )
virtual void	**PaintGLPoints**(Float_t* *vertex*)

public:

class TTUBS	**TTUBS**( )
class TTUBS	**TTUBS**(Text_t* *name,* Text_t* *title,* Text_t* *material,* const Float_t *rmin,* Float_t *rmax,* Float_t *dz,* Float_t *phi1,* Float_t *phi2*)
class TTUBS	**TTUBS**(class TTUBS&)
void	**~TTUBS**( )
class TClass*	**Class**( )
virtual Int_t	**DistancetoPrimitive**(Int_t *px,* Int_t *py*)
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**SetPoints**(Float_t* *buff*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Sizeof3D**( )
virtual void	**Streamer**(class TBuffer& *b*)

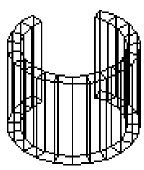
Data Members

protected:

Float_t	**fPhi1**	*first phi limit*
Float_t	**fPhi2**	*second phi limit*

See Also: TCONS, TCTUB

Class Description



```
TUBS is a segment of a tube. It has 8 parameters:

  - name       name of the shape
  - title      shape's title
  - material  (see TMaterial)
  - rmin       inside radius
  - rmax       outside radius
  - dz         half length in z
  - phi1       starting angle of the segment
  - phi2       ending angle of the segment

NOTE: phi1 should be smaller than phi2. If this is not the case,
    the system adds 360 degrees to phi2.
```

TTUBS( )

```
TUBS shape default constructor
```

TTUBS(Text_t **name,* Text_t **title,* Text_t **material,* Float_t *rmin,*Float_t rmax, Float_t dz, Float_t phi1, Float_t *phi2*): **TTUBE**(*name, title, material, rmin, rmax, dz*)

```
TUBS shape normal constructor
```

~TTUBS( )

```
TUBS shape default destructor
```

Int_t **DistancetoPrimitive**(Int_t *px,* Int_t *py*)

```
Compute distance from point px,py to a TUBE
Compute the closest distance of approach from point px,py to each
computed outline point of the TUBE.
```

void **Paint**(Option_t **option*)

```
Paint this 3-D shape with its current attributes
```

void **PaintGLPoints**(Float_t **vertex*)

```
Paint BRIK via OpenGL
```

void **SetPoints**(Float_t **buff*)

```
Create TUBS points
```

void **Sizeof3D**( ) const

```
Return total X3D size of this shape with its attributes
```

Author: Nenad Buncic 18/09/95
Last update: 1.01/07 17/06/97 13.16.11 by Fons Rademakers

TUnixSystem

class **TUnixSystem** : public TSystem

protected:

char*	**DynamicPathName**(char* *lib*)
void*	**FindDynLib**(char* *lib*)
const char*	**GetDynamicPath**( )
Func_t*	**UnixDynFindSymbol**(char* *lib,* char* *entry*)
void	**UnixDynListLibs**(char* *lib*)
void	**UnixDynListSymbols**(char* *lib,* char* *re*)
int	**UnixDynLoad**(char* *lib*)
void	**UnixDynUnload**(char* *lib*)
int	**UnixFilestat**(char* *path,* Long_t* *id,* Long_t* *size,* Long_t* *flags,* Long_t* *modtime*)
const char*	**UnixGetdirentry**(void* *dir*)
const char*	**UnixHomedirectory**(char* *user = 0*)
int	**UnixMakedir**(char* *name*)
Long_t	**UnixNow**( )
void*	**UnixOpendir**(char* *name*)
int	**UnixRecv**(int *sock,* void* *buf,* int *len,* int *flag*)
int	**UnixSelect**(UInt_t *nfds,* class TFdSet* *readready,* class TFdSet* *writeready,* Long_t *timeout*)
int	**UnixSend**(int *sock,* void* *buf,* int *len,* int *flag*)
int	**UnixSetitimer**(Long_t *ms*)
void	**UnixSignal**(enum ESignals *sig,* SigHandler_t *h*)
char*	**UnixSigname**(enum ESignals *sig*)
int	**UnixTcpConnect**(char* *hostname,* int *port*)
int	**UnixTcpService**(int *port,* Bool_t *reuse,* int *backlog*)
int	**UnixUnixConnect**(int *port*)
int	**UnixUnixService**(int *port,* int *backlog*)
int	**UnixWaitchild**( )

public:

class TUnixSystem	**TUnixSystem**(class TUnixSystem&)
class TUnixSystem	**TUnixSystem**( )
void	**~TUnixSystem**( )
void	**Abort**(int *code = 0*)
int	**AcceptConnection**(int *sock*)
Bool_t	**AccessPathName**(char* path, (*unknown*) EAccessMode mode = kFileExists)
void	**AddFileHandler**(class TFileHandler* *fh*)
void	**AddSignalHandler**(class TSignalHandler* *sh*)
void	**AddTimer**(class TTimer* *ti*)
int	**AnnounceTcpService**(int *port,* Bool_t *reuse,* int *backlog*)
int	**AnnounceUnixService**(int *port,* int *backlog*)
Bool_t	**ChangeDirectory**(char* *path*)
void	**CheckChilds**( )
Bool_t	**CheckSignals**(Bool_t *sync*)
class TClass*	**Class**( )
void	**CloseConnection**(int *sock*)
void	**Closelog**( )
int	**ClosePipe**(FILE* *pipe*)
char*	**ConcatFileName**(char* *dir,* char* *name*)
int	**ConnectService**(char* *server,* int *port*)
void	**DispatchOneEvent**( )
void	**DispatchSignals**(enum ESignals *sig*)
Bool_t	**DispatchTimers**(Bool_t *mode*)

Func_t*	**DynFindSymbol**(char* *module,* char* *entry*)
int	**Exec**(char* *shellcmd*)
void	**Exit**(int *code,* Bool_t *mode = kTRUE*)
char*	**ExpandPathName**(char* *path*)
Bool_t	**ExpandPathName**(class TString& *patbuf*)
void	**FreeDirectory**(void* *dirp*)
const char*	**GetDirEntry**(void* *dirp*)
const char*	**Getenv**(char* *name*)
const char*	**GetError**( )
class TInetAddress	**GetHostByName**(char* *server*)
int	**GetPathInfo**(char* *path,* Long_t* *id,* Long_t* *size,* Long_t* *flags,* Long_t* *modtime*)
class TInetAddress	**GetPeerName**(int *sock*)
int	**GetPid**( )
int	**GetServiceByName**(char* *service*)
char*	**GetServiceByPort**(int *port*)
class TInetAddress	**GetSockName**(int *sock*)
int	**GetSockOpt**(int *sock,* int *option,* int* *val*)
const char*	**HomeDirectory**( )
const char*	**HostName**( )
void	**IgnoreInterrupt**(Bool_t *ignore = kTRUE*)
Bool_t	**Init**( )
virtual class TClass*	**IsA**( )
int	**Link**(char* *from,* char* *to*)
void	**ListLibraries**(char* *regexp*)
void	**ListSymbols**(char* *module,* char* *re*)
int	**Load**(char* *module,* char* *entry*)
int	**MakeDirectory**(char* *name*)
class TTime	**Now**( )
int	**OpenConnection**(char* *server,* int *port*)
void*	**OpenDirectory**(char* *name*)
void	**Openlog**(char* name, Int_t options, (*unknown*) ELogFacility facility)
FILE*	**OpenPipe**(char* *shellcmd,* char* *mode*)
int	**RecvBuf**(int *sock,* void* *buffer,* int *length*)
int	**RecvRaw**(int *sock,* void* *buffer,* int *length,* int *flag*)
class TFileHandler*	**RemoveFileHandler**(class TFileHandler* *fh*)
class TSignalHandler*	**RemoveSignalHandler**(class TSignalHandler* *sh*)
class TTimer*	**RemoveTimer**(class TTimer* *ti*)
void	**Rename**(char* *from,* char* *to*)
int	**SendBuf**(int *sock,* void* *buffer,* int *length*)
int	**SendRaw**(int *sock,* void* *buffer,* int *length,* int *flag*)
void	**Setenv**(char* *name,* char* *value*)
void	**SetProgname**(char* *name*)
int	**SetSockOpt**(int *sock,* int *option,* int *val*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
void	**Sleep**(UInt_t *milliSec*)
void	**StackTrace**( )
virtual void	**Streamer**(class TBuffer& *b*)
int	**Symlink**(char* *from,* char* *to*)
void	**Syslog**((*unknown*) ELogLevel level, char* mess)
int	**Umask**(Int_t *mask*)
int	**Unlink**(char* *name*)
void	**Unload**(char* *module*)
char*	**Which**(char* search, char* file, (*unknown*) EAccessMode mode = kFileExists)
const char*	**WorkingDirectory**( )

Data Members

Class Description

TUnixSystem

Class providing an interface to the UNIX Operating System.

Bool_t **Init**( )

Initialize Unix system interface.

void **SetProgname**(const char **name*)

Set the application name (from command line, argv[0]) and copy it in gProgName. Copy the application pathname in gProgPath.

const char* **GetError**( )

Return system error string.

const char* **HostName**( )

Return the system's host name.

void **AddFileHandler**(TFileHandler **h*)

Add a file handler to the list of system file handlers.

TFileHandler* **RemoveFileHandler**(TFileHandler **h*)

Remove a file handler from the list of file handlers.

void **AddSignalHandler**(TSignalHandler **h*)

Add a signal handler to list of system signal handlers.

TSignalHandler* **RemoveSignalHandler**(TSignalHandler **h*)

Remove a signal handler from list of signal handlers.

void **IgnoreInterrupt**(Bool_t *ignore*)

Ignore the interrupt signal if ignore == kTRUE else restore previous behaviour. Typically call ignore interrupt before writing to disk.

void **DispatchOneEvent**( )

Dispatch a single event.

void **Sleep**(UInt_t *milliSec*)

Sleep milliSec milliseconds.

void **DispatchSignals**(ESignals *sig*)

Handle and dispatch signals.

Bool_t **CheckSignals**(Bool_t *sync*)

Check if some signals were raised and call their Notify() member.

void **CheckChilds**( )

Check if childs have finished.

int **MakeDirectory**(const char **name*)

Make a Unix file system directory. Returns 0 in case of success and -1 if the directory could not be created.

void* **OpenDirectory**(const char **name*)

Open a Unix file system directory. Returns 0 if directory does not exist.

void **FreeDirectory**(void **dirp*)

Close a Unix file system directory.

const char* **GetDirEntry**(void **dirp*)

Get next Unix file system directory entry. Returns 0 if no more entries.

Bool_t **ChangeDirectory**(const char **path*)

```
Change directory. Returns kTRUE in case of success, kFALSE otherwise.
```

const char* **WorkingDirectory**( )

```
Return working directory.
```

const char* **HomeDirectory**( )

```
Return the user's home directory.
```

char* **ConcatFileName**(const char **dir,* const char **name*)

```
Concatenate a directory and a file name. Returned string must be
deleted by user.
```

Bool_t **AccessPathName**(const char **path,* EAccessMode *mode*)

```
Returns FALSE if one can access a file using the specified access mode.
Mode is the same as for the Unix access(2) function.
```

void **Rename**(const char **f,* const char **t*)

```
Rename a file.
```

int **GetPathInfo**(const char **path,* Long_t **id,* Long_t **size,*Long_t *flags, Long_t **modtime*)

```
Get info about a file: id, size, flags, modification time.
Id      is (statbuf.st_dev << 24) + statbuf.st_ino
Size    is the file size
Flags   is file type: bit 1 set executable, bit 2 set directory,
                    bit 3 set regular file
Modtime is modification time
```

int **Link**(const char **from,* const char **to*)

```
Create a link from file1 to file2. Returns 0 when successful,
-1 in case of failure.
```

int **Symlink**(const char **from,* const char **to*)

```
Create a symlink from file1 to file2. Returns 0 when succesfull,
-1 in case of failure.
```

int **Unlink**(const char **name*)

```
Unlink, i.e. remove, a file or directory.
```

Bool_t **ExpandPathName**(TString *&patbuf0*)

```
Expand a pathname getting rid of special shell characters like ~.$, etc.
```

char* **ExpandPathName**(const char **path*)

```
Expand a pathname getting rid of special shell characaters like ~.$, etc.
User must delete returned string.
```

int **Umask**(Int_t *mask*)

```
Set the process file creation mode mask.
```

char* **Which**(const char **search,* const char **file,* EAccessMode *mode*)

```
Find location of file in a search path.
User must delete returned string.
```

void **Setenv**(const char **name,* const char **value*)

```
Set environment variable. The string passed will be owned by
the environment and can not be reused till a "name" is set
again. The solution below will lose the space for the string
in that case, but if this functions is not called thousands
of times that should not be a problem.
```

const char* **Getenv**(const char **name*)

```
Get environment variable.
```

int **Exec**(const char **shellcmd*)

Execute a command.

FILE* **OpenPipe**(const char **command,* const char **mode*)
Open a pipe.

int **ClosePipe**(FILE **pipe*)
Close the pipe.

int **GetPid**( )
Get process id.

void **Exit**(int *code,* Bool_t *mode*)
Exit the application.

void **Abort**(*int*)
Abort the application.

void **StackTrace**( )
Print a stack trace.

void **Openlog**(const char **name,* Int_t *options,* ELogFacility *facility*)
Open connection to system log daemon. For the use of the options and facility see the Unix openlog man page.

void **Syslog**(ELogLevel *level,* const char **mess*)
Send mess to syslog daemon. Level is the logging level and mess the message that will be written on the log.

void **Closelog**( )
Close connection to system log daemon.

int **Load**(const char **module,* const char **entry*)
Load a shared library. Returns 0 on successful loading, 1 in case lib was already loaded and -1 in case lib does not exist or in case of error.

void **Unload**(const char **module*)
Unload a shared library.

void **ListSymbols**(const char **module,* const char **regexp*)
List symbols in a shared library.

void **ListLibraries**(const char **regexp*)
List all loaded shared libraries.

TTime **Now**( )
Return current time.

Bool_t **DispatchTimers**(Bool_t *mode*)
Handle and dispatch timers. If mode = kTRUE dispatch synchronous timers else a-synchronous timers.

void **AddTimer**(TTimer **ti*)
Add timer to list of system timers.

TTimer* **RemoveTimer**(TTimer **ti*)
Remove timer from list of system timers.

TInetAddress **GetHostByName**(const char **hostname*)
Get Internet Protocol (IP) address of host.

TInetAddress **GetSockName**(int *sock*)
Get Internet Protocol (IP) address of host and port #.

TInetAddress **GetPeerName**(int *sock*)
    Get Internet Protocol (IP) address of remote host and port #.

int **GetServiceByName**(const char **servicename*)
    Get port # of internet service.

char* **GetServiceByPort**(int *port*)
    Get name of internet service.

int **ConnectService**(const char **servername,* int *port*)
    Connect to service servicename on server servername.

int **OpenConnection**(const char **server,* int *port*)
    Open a connection to a service on a server. Try 3 times with an interval of 1 second.

int **AnnounceTcpService**(int *port,* Bool_t *reuse,* int *backlog*)
    Announce TCP/IP service.

int **AnnounceUnixService**(int *port,* int *backlog*)
    Announce unix domain service.

int **AcceptConnection**(int *sock*)
    Accept a connection. In case of an error return -1. In case non-blocking I/O is enabled and no connections are available return -2.

void **CloseConnection**(int *sock*)
    Close socket.

int **RecvBuf**(int *sock,* void **buf,* int *length*)
    Receive a buffer headed by a length indicator. Lenght is the size of the buffer. Returns the number of bytes received in buf or -1 in case of error.

int **SendBuf**(int *sock,* const void **buf,* int *length*)
    Send a buffer headed by a length indicator. Returns length of sent buffer or -1 in case of error.

int **RecvRaw**(int *sock,* void **buf,* int *length,* int *opt*)
    Receive exactly length bytes into buffer. Use opt to receive out-of-band data or to have a peek at what is in the buffer (see TSocket). Buffer must be able to store at least lenght bytes. Returns the number of bytes received or -1 in case of error, -2 in case of MSG_OOB and errno == EWOULDBLOCK, -3 in case of MSG_OOB and errno == EINVAL and -4 in case of kNonBlock and errno == EWOULDBLOCK.

int **SendRaw**(int *sock,* const void **buf,* int *length,* int *opt*)
    Send exactly length bytes from buffer. Use opt to send out-of-band data (see TSocket). Returns the number of bytes sent or -1 in case of error.

int **SetSockOpt**(int *sock,* int *opt,* int *val*)
    Set socket option.

int **GetSockOpt**(int *sock,* int *opt,* int **val*)
    Get socket option.

void **UnixSignal**(ESignals *sig,* SigHandler_t *handler*)
    Set a signal handler for a signal.

char* **UnixSigname**(ESignals *sig*)
    Return the signal name associated with a signal.

Long_t **UnixNow**( )

```
Get current time in milliseconds since 0:00 Jan 1 1995.
```

int **UnixSetitimer**(Long_t *ms*)

```
Set interval timer to time-out in ms milliseconds.
```

int **UnixSelect**(UInt_t *nfds,* TFdSet **readready,* TFdSet **writeready,*Long_t *timeout*)

```
Wait for events on the file descriptors specified in the readready and
writeready masks or for timeout (in milliseconds) to occur.
```

const char* **UnixHomedirectory**(const char **name*)

```
Returns the user's home directory.
```

int **UnixMakedir**(const char **dir*)

```
Make a Unix file system directory. Returns 0 in case of success and
-1 if the directory could not be created.
```

void* **UnixOpendir**(const char **dir*)

```
Open a directory.
```

const char* **UnixGetdirentry**(void **dirp1*)

```
Returns the next directory entry.
```

int **UnixFilestat**(const char **path,* Long_t **id,* Long_t **size,*Long_t *flags, Long_t **modtime*)

```
Get info about a file: id, size, flags, modification time.
Id      is (statbuf.st_dev << 24) + statbuf.st_ino
Size    is the file size
Flags   is file type: bit 0 set executable, bit 1 set directory,
                    bit 2 set regular file
Modtime is modification time
The function returns 0 in case of success and 1 if the file could
not be stat'ed.
```

int **UnixWaitchild**( )

```
Wait till child is finished.
```

int **UnixTcpConnect**(const char **hostname,* int *port*)

```
Open a TCP/IP connection to server and connect to a service (i.e. port).
Is called via the TSocket constructor.
```

int **UnixUnixConnect**(int *port*)

```
Connect to a Unix domain socket.
```

int **UnixTcpService**(int *port,* Bool_t *reuse,* int *backlog*)

```
Open a socket, bind to it and start listening for TCP/IP connections
on the port. Tries 20 times to bind to the socket with 10 second
intervals. Returns socket fd or -1.
```

int **UnixUnixService**(int *port,* int *backlog*)

```
Open a socket, bind to it and start listening for Unix domain connections
to it. Returns socket fd or -1.
```

int **UnixRecv**(int *sock,* void **buffer,* int *length,* int *flag*)

```
Receive exactly length bytes into buffer. Returns number of bytes
received. Returns -1 in case of error, -2 in case of MSG_OOB
and errno == EWOULDBLOCK, -3 in case of MSG_OOB and errno == EINVAL
and -4 in case of kNonBlock and errno == EWOULDBLOCK.
```

int **UnixSend**(int *sock,* const void **buffer,* int *length,* int *flag*)

```
Send exactly length bytes from buffer. Returns -1 in case of error,
otherwise number of sent bytes.
```

const char* **GetDynamicPath**( )

```
Get shared library search path. Static utility function.
```

char* **DynamicPathName**(const char **lib*)
```
Returns the path of a shared library (searches for library in the
shared library search path). If no file name extension is provided
it first tries .so, .sl and then .dl. Returned string must be deleted.
```

void* **FindDynLib**(const char **lib*)
```
Returns the handle to a loaded shared library. Returns 0 when library
not loaded.
```

int **UnixDynLoad**(const char **lib*)
```
Load a shared library. Returns 0 on successful loading, 1 in
case lib was already loaded and -1 in case lib does not exist
or in case of error.
```

Func_t **UnixDynFindSymbol**(const char **lib,* const char **entry*)
```
Finds and returns a function pointer to a symbol in the shared library.
Returns 0 when symbol not found.
```

void **UnixDynListSymbols**(const char **lib,* const char **regexp*)
```
List symbols in a shared library. One can use wildcards to list only
the intresting symbols.
```

void **UnixDynListLibs**(const char **lib*)
```
List all loaded shared libraries.
```

void **UnixDynUnload**(const char **lib*)
```
Unload a shared library.
```

Author: Fons Rademakers 15/09/95
Last update: 04/11/97 17.34.00 by Fons Rademakers

TUrl

class **TUrl** : public TObject

private:

class TUrl	**TUrl**( )

public:

class TUrl	**TUrl**(char* *url*)
class TUrl	**TUrl**(class TUrl& *url*)
void	**~TUrl**( )
class TClass*	**Class**( )
const char*	**GetAnchor**( )
const char*	**GetFile**( )
const char*	**GetHost**( )
const char*	**GetOptions**( )
Int_t	**GetPort**( )
const char*	**GetProtocol**( )
const char*	**GetUrl**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsValid**( )
class TUrl&	***operator=***(class TUrl& *rhs*)
void	**Print**(Option_t* *option*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TString	**fUrl**	*full URL*
class TString	**fProtocol**	*protocol*: http, ftp, news, root
class TString	**fHost**	*remote host*
class TString	**fFile**	*remote object*
class TString	**fAnchor**	*anchor in object*
class TString	**fOptions**	*options (after ?)*
Int_t	**fPort**	*port through which to contact remote server*

Class Description

```
TUrl

This class represents a WWW compatible URL.
It provides member functions to returns the different parts of
an URL.
```

TUrl(const char **url*)

```
Parse url character string and split in its different subcomponents.
Use IsValid() to check if URL is legal.

url: [proto://]host[:port][/file.ext][#anchor][?options]

Allowed protocols: http, root, ftp, news, file (default http)
Default ports: http=80, root=432, ftp=20, news=119.
```

TUrl(const TUrl &*url*)

```
TUrl copt ctor.
```

const char* **GetUrl**( )

```
Return full URL.
```

void **Print**(Option_t *)

```
Print URL on stdout.
```

Author: Fons Rademakers 17/01/97
Last update: 1.03/05 09/10/97 13.33.13 by Fons Rademakers

TView

class **TView** : public TObject

protected:

void	**ResetView**(Float_t *longitude,* Float_t *latitude,* Float_t *psi,* Int_t& *irep*)

public:

class TView	**TView**(Int_t *system*)
class TView	**TView**( )
class TView	**TView**(Float_t* *rmin,* Float_t* *rmax,* Int_t *system = 1*)
class TView	**TView**(class TView&)
void	**~TView**( )
virtual void	**AxisVertex**(Float_t *ang,* Float_t* *av,* Int_t& *ix1,* Int_t& *ix2,* Int_t& *iy1,* Int_t& *iy2,* Int_t& *iz1,* Int_t& *iz2*)
class TClass*	**Class**( )
virtual void	**DefineViewDirection**(Float_t* *s,* Float_t* *c,* Double_t *cosphi,* Double_t *sinphi,* Double_t *costhe,* Double_t *sinthe,* Double_t *cospsi,* Double_t *sinpsi,* Float_t* *tnorm,* Float_t* *tback*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual void	**ExecuteRotateView**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual void	**FindNormal**(Float_t *x,* Float_t *y,* Float_t *z,* Float_t& *zn*)
virtual void	**FindPhiSectors**(Int_t *iopt,* Int_t& *kphi,* Float_t* *aphi,* Int_t& *iphi1,* Int_t& *iphi2*)
virtual void	**FindScope**(Float_t* *scale,* Float_t* *center,* Int_t& *irep*)
virtual void	**FindThetaSectors**(Int_t *iopt,* Float_t *phi,* Int_t& *kth,* Float_t* *ath,* Int_t& *ith1,* Int_t& *ith2*)
Bool_t	**GetAutoRange**( )
virtual Int_t	**GetDistancetoAxis**(Int_t *axis,* Int_t *px,* Int_t *py*)
Float_t	**GetLatitude**( )
Float_t	**GetLongitude**( )
class TSeqCollection*	**GetOutline**( )
Float_t	**GetPsi**( )
virtual void	**GetRange**(Float_t* *min,* Float_t* *max*)
Float_t*	**GetRmax**( )
Float_t*	**GetRmin**( )
Int_t	**GetSystem**( )
Float_t*	**GetTN**( )
Float_t*	**GetTnorm**( )
virtual class TClass*	**IsA**( )
virtual void	**NormalWCtoNDC**(Float_t* *pw,* Float_t* *pn*)
virtual void	**PadRange**(Float_t *rback*)
virtual void	**SetAutoRange**(Bool_t *autorange = kTRUE*)
virtual void	**SetAxisNDC**(Float_t* *x1,* Float_t* *x2,* Float_t* *y1,* Float_t* *y2,* Float_t* *z1,* Float_t* *z2*)
void	**SetLatitude**(Float_t *latitude*)
void	**SetLongitude**(Float_t *longitude*)
virtual void	**SetOutlineToCube**( )
void	**SetPsi**(Float_t *psi*)
virtual void	**SetRange**(Float_t *x0,* Float_t *y0,* Float_t *z0,* Float_t *x1,* Float_t *y1,* Float_t *z1,* Int_t *flag = 0*)
virtual void	**SetRange**(Float_t* *min,* Float_t* *max*)
virtual void	**SetSystem**(Int_t *system*)
virtual void	**SetView**(Float_t *longitude,* Float_t *latitude,* Float_t *psi,* Int_t& *irep*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**WCtoNDC**(Float_t* *pw,* Float_t* *pn*)

Data Members

protected:

Int_t	**fSystem**	*Coordinate system*
Float_t	**fLatitude**	*View angle latitude*
Float_t	**fLongitude**	*View angle longitude*
Float_t	**fPsi**	*View angle psi*
Float_t	**fTN**	
Float_t	**fTB**	
Float_t	**fRmax**	*Upper limits of object*
Float_t	**fRmin**	*Lower limits of object*
Float_t	**fTnorm**	*Transformation matrix*
Float_t	**fTback**	*Back transformation matrix*
Float_t	**fX1**	*First coordinate of X axis*
Float_t	**fX2**	*Second coordinate of X axis*
Float_t	**fY1**	*First coordinate of Y axis*
Float_t	**fY2**	*Second coordinate of Y axis*
Float_t	**fZ1**	*First coordinate of Z axis*
Float_t	**fZ2**	*Second coordinate of Z axis*
class TSeqCollection*	**fOutline**	*Collection of outline's objects*
Bool_t	**fDefaultOutline**	*Set to TRUE if outline is default cube*
Bool_t	**fAutoRange**	*Set to TRUE if range computed automatically*

Class Description

```
The V I E W class
This package was originally written by Evgueni Tcherniaev
from IHEP/Protvino.

The original Fortran implementation was adapted to HIGZ/PAW
by Olivier Couet and  Evgueni Tcherniaev.

This View class is a subset of the original system
It has been converted to a C++ class  by René Brun
```

TView( )

```
View default constructor
```

~TView( )

```
View default destructor
```

TView(Int_t *system*)

```
View constructor
 Creates a 3-D view in the current pad
 By default pad range in 3-D view is (-1,-1,1,1), so ...

 Before drawing a 3-D object in a pad, a 3-D view must be created.
 Note that a view is automatically created when drawing legos or surfaces.

The coordinate system is selected via system:
 system = 1  Cartesian
 system = 2  Polar
 system = 3  Cylindrical
 system = 4  Spherical
 system = 5  PseudoRapidity/Phi
```

TView(Float_t **rmin,* Float_t **rmax,* Int_t *system*)

```
View constructor
 Creates a 3-D view in the current pad
rmin[3], rmax[3] are the limits of the object depending on
the selected coordinate system
```

```
Before drawing a 3-D object in a pad, a 3-D view must be created.
Note that a view is automatically created when drawing legos or surfaces.

The coordinate system is selected via system:
 system = 1  Cartesian
 system = 2  Polar
 system = 3  Cylindrical
 system = 4  Spherical
 system = 5  PseudoRapidity/Phi
```

void **AxisVertex**(Float_t *ang,* Float_t **av,* Int_t &*ix1,* Int_t &*ix2,* Int_t &*iy1,* Int_t &*iy2,* Int_t &*iz1,* Int_t &*iz2*)

```
Define axis  vertices
 Input  ANG     - angle between X and Y axis

 Output: AV(3,8) - axis vertixes
         IX1     - 1st point of X-axis
         IX2     - 2nd point of X-axis
         IY1     - 1st point of Y-axis
         IY2     - 2nd point of Y-axis
         IZ1     - 1st point of Z-axis
         IZ2     - 2nd point of Z-axis

              8                        6
             / \                      /|\
          5 /   \ 7                5 / | \ 7
           |\   /|                  |  |  |
THETA < 90  | \6/ |     THETA > 90   | /2\ |
(Top view)  |  |  |   (Bottom view)  |/   \|
          1 \ | /3                 1 \   /3
             \|/                      \ /
              2                        4
```

void **DefineViewDirection**(Float_t **s,* Float_t **c,*Double_t cosphi, Double_t sinphi,Double_t costhe, Double_t sinthe,Double_t cospsi, Double_t sinpsi,Float_t *tnorm, Float_t **tback*)

```
Define view direction (in spherical coordinates)
           Compute transformation matrix from world coordinates
           to normalised coordinates (-1 to +1)

 Input: S(3)    - scale factors
        C(3)    - centre of scope
        COSPHI  - longitude COS
        SINPHI  - longitude SIN
        COSTHE  - latitude COS (angle between +Z and view direc.)
        SINTHE  - latitude SIN
        COSPSI  - screen plane rotation angle COS
        SINPSI  - screen plane rotation angle SIN
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
```

void **ExecuteRotateView**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a object is clicked with the locator

If Left button clicked in the object area, while the button is kept down
the cube representing the surrounding frame for the corresponding
new latitude and longitude position is drawn.
```

void **FindNormal**(Float_t *x,* Float_t *y,* Float_t *z,* Float_t &*zn*)

```
Find Z component of NORMAL in normalized coordinates
 Input: X - X-component of NORMAL
        Y - Y-component of NORMAL
        Z - Z-component of NORMAL
```

```
 Output: ZN - Z-component of NORMAL in normalized coordinates
```

void **FindPhiSectors**(Int_t *iopt,* Int_t &*kphi,* Float_t **aphi,* Int_t &*iphi1,* Int_t &*iphi2*)

```
Find critical PHI sectors
 Input: IOPT    - options: 1 - from BACK to FRONT 'BF'
                           2 - from FRONT to BACK 'FB'
        KPHI    - number of phi sectors
        APHI(*) - PHI separatrices

 Output: IPHI1  - initial sector
         IPHI2  - final sector
```

void **FindThetaSectors**(Int_t *iopt,* Float_t *phi,* Int_t &*kth,* Float_t **ath,* Int_t &*ith1,* Int_t &*ith2*)

```
Find critical THETA sectors for given PHI sector
 Input: IOPT        - options: 1 - from BACK to FRONT 'BF'
                               2 - from FRONT to BACK 'FB'
        PHI         - PHI sector
        KTH         - number of THETA sectors
        ATH(*)      - THETA separatrices

 Output: ITH1  - initial sector
         ITH2  - final sector
```

void **FindScope**(Float_t **scale,* Float_t **center,* Int_t &*irep*)

```
Find centre of a MIN-MAX scope and scale factors
 Output: SCALE(3)  - scale factors
         CENTER(3) - centre
         IREP      - reply (-1 if error in min-max)
```

Int_t **GetDistancetoAxis**(Int_t *axis,* Int_t *px,* Int_t *py*)

```
Return distance to axis from point px,py
Algorithm:

 A(x1,y1)         P                             B(x2,y2)
 ------------------------------------------------
                  I
                  I
                  I
                  I
                 M(x,y)

Let us call  a = distance AM     A=a**2
            b = distance BM     B=b**2
            c = distance AB     C=c**2
            d = distance PM     D=d**2
            u = distance AP     U=u**2
            v = distance BP     V=v**2     c = u + v

D = A - U
D = B - V  = B -(c-u)**2
  ==> u = (A -B +C)/2c
```

void **GetRange**(Float_t **min,* Float_t **max*)

```
Get Range function
```

void **NormalWCtoNDC**(Float_t **pw,* Float_t **pn*)

```
Transfer vector of NORMAL from word to normalized coodinates
 Input: PW(3) - vector of NORMAL in word coordinate system
        PN(3) - vector of NORMAL in normalized coordinate system
```

void **PadRange**(Float_t *rback*)

```
Set the correct window size for lego and surface plots
```

```
Set the correct window size for lego and surface plots.
And draw the background if necessary.

 Input parameters:

RBACK : Background colour
```

void **SetAxisNDC**(Float_t **x1*, Float_t **x2*, Float_t **y1*, Float_t **y2*, Float_t **z1*, Float_t **z2*)

```
Store axis coordinates in the NDC system
```

void **SetOutlineToCube**( )

```
This is a function which creates default outline
   x = fRmin[0]        X = fRmax[0]
   y = fRmin[1]        Y = fRmax[1]
   z = fRmin[2]        Z = fRmax[2]

         (x,Y,Z) +---------+ (X,Y,Z)
                /         /|
               /         / |
              /         /  |
     (x,y,Z) +---------+   |
             |         |   + (X,Y,z)
             |         |  /
             |         | /
             |         |/
             +---------+
          (x,y,z)   (X,y,z)
```

void **SetRange**(Float_t **min*, Float_t **max*)

```
Set Range function
```

void **SetRange**(Float_t *x0*, Float_t *y0*, Float_t *z0*, Float_t *x1*, Float_t *y1*, Float_t *z1*, Int_t *flag*)

```
Set 3-D View range
Input:  x0, y0, z0 are minimum coordinates
      x1, y1, z1 are maximum coordinates

      flag values are: 0 (set always) <- default
                       1 (shrink view)
                       2 (expand view)
```

void **ResetView**(Float_t *longitude*, Float_t *latitude*, Float_t *psi*, Int_t &*irep*)

```
Set view direction (in spherical coordinates)
 Input  PHI     - longitude
        THETA   - latitude (angle between +Z and view direction)
        PSI     - rotation in screen plane

 Output: IREP   - reply (-1 if error in min-max)

 Errors: error in min-max scope
```

void **WCtoNDC**(Float_t **pw*, Float_t **pn*)

```
Transfer point from world to normalized coordinates
 Input: PW(3) - point in world coordinate system
        PN(3) - point in normalized coordinate system
```

Author: René Brun, Nenad Buncic, Evgueni Tcherniaev, Olivier Couet 18/08/95
Last update: 1.03/07 04/11/97 17.10.41 by René Brun

TVirtualPad

class **TVirtualPad** : public TWbox, public TAttPad

public:

void	**~TVirtualPad**( )
virtual void	**AbsCoordinates**(Bool_t *set*)
virtual Float_t	**AbsPixeltoX**(Int_t *px*)
virtual void	**AbsPixeltoXY**(Int_t *xpixel,* Int_t *ypixel,* Axis_t& *x,* Axis_t& *y*)
virtual Float_t	**AbsPixeltoY**(Int_t *py*)
virtual void	**cd**(Int_t *subpadnumber = 0*)
class TClass*	**Class**( )
virtual void	**Clear**(Option_t* *option*)
virtual void	**Close**(Option_t* *option*)
virtual void	**CopyPixmap**( )
virtual void	**CopyPixmaps**( )
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax,* Text_t* *title*)
virtual void	**DrawLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**DrawLineNDC**(Float_t *u1,* Float_t *v1,* Float_t *u2,* Float_t *v2*)
virtual void	**DrawText**(Float_t *x,* Float_t *y,* Text_t* *text*)
virtual void	**DrawTextNDC**(Float_t *u,* Float_t *v,* Text_t* *text*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
virtual Float_t	**GetAbsHNDC**( )
virtual Float_t	**GetAbsWNDC**( )
virtual Float_t	**GetAbsXlowNDC**( )
virtual Float_t	**GetAbsYlowNDC**( )
virtual class TCanvas*	**GetCanvas**( )
virtual Int_t	**GetCanvasID**( )
virtual class TFrame*	**GetFrame**( )
virtual Bool_t	**GetGridx**( )
virtual Bool_t	**GetGridy**( )
virtual Color_t	**GetHighLightColor**( )
virtual Float_t	**GetHNDC**( )
virtual class TList*	**GetListOfPrimitives**( )
virtual Int_t	**GetLogx**( )
virtual Int_t	**GetLogy**( )
virtual Int_t	**GetLogz**( )
virtual class TVirtualPad*	**GetMother**( )
virtual const Text_t*	**GetName**( )
virtual Int_t	**GetNumber**( )
virtual Int_t	**GetPadPaint**( )
virtual void	**GetPadPar**(Float_t& *xlow,* Float_t& *ylow,* Float_t& *xup,* Float_t& *yup*)
virtual class TObject*	**GetPadPointer**( )
virtual class TVirtualPad*	**GetPadSave**( )
virtual Float_t	**GetPhi**( )
virtual Int_t	**GetPixmapID**( )
virtual class TObject*	**GetPrimitive**(Text_t* *name*)
virtual void	**GetRange**(Float_t& *x1,* Float_t& *y1,* Float_t& *x2,* Float_t& *y2*)
virtual void	**GetRangeAxis**(Axis_t& *xmin,* Axis_t& *ymin,* Axis_t& *xmax,* Axis_t& *ymax*)
virtual class TObject*	**GetSelected**( )
virtual class TVirtualPad*	**GetSelectedPad**( )
virtual Float_t	**GetTheta**( )
virtual const Text_t*	**GetTitle**( )
virtual class TView*	**GetTView**( )

virtual Float_t	**GetUxmax**( )
virtual Float_t	**GetUxmin**( )
virtual Float_t	**GetUymax**( )
virtual Float_t	**GetUymin**( )
virtual class TView*	**GetView**( )
virtual class TPadView3D*	**GetView3D**( )
virtual UInt_t	**GetWh**( )
virtual Float_t	**GetWNDC**( )
virtual UInt_t	**GetWw**( )
virtual Float_t	**GetXlowNDC**( )
virtual Float_t	**GetYlowNDC**( )
virtual void	**HighLight**(Color_t *col = kRed,* Bool_t *set = kTRUE*)
virtual class TClass*	**IsA**( )
virtual Bool_t	**IsBatch**( )
virtual Bool_t	**IsModified**( )
virtual Bool_t	**IsRetained**( )
virtual void	**ls**(Option_t* *option*)
virtual void	**Modified**(Bool_t *flag = 1*)
virtual Bool_t	**OpaqueMoving**( )
virtual Bool_t	**OpaqueResizing**( )
virtual Float_t	**PadtoX**(Axis_t *x*)
virtual Float_t	**PadtoY**(Axis_t *y*)
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintBox**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2,* Option_t* *option*)
virtual void	**PaintFillArea**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintLine**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**PaintLine3D**(Float_t* *p1,* Float_t* *p2*)
virtual void	**PaintLineNDC**(Coord_t *u1,* Coord_t *v1,* Coord_t *u2,* Coord_t *v2*)
virtual void	**PaintModified**( )
virtual void	**PaintPadFrame**(Float_t *xmin,* Float_t *ymin,* Float_t *xmax,* Float_t *ymax*)
virtual void	**PaintPolyLine**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintPolyLine3D**(Int_t *n,* Float_t* *p*)
virtual void	**PaintPolyLineNDC**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintPolyMarker**(Int_t *n,* Float_t* *x,* Float_t* *y,* Option_t* *option*)
virtual void	**PaintText**(Float_t *x,* const Float_t *y,* Text_t* *text*)
virtual void	**PaintTextNDC**(Float_t *u,* Float_t *v,* Text_t* *text*)
virtual Float_t	**PixeltoX**(Int_t *px*)
virtual void	**PixeltoXY**(Int_t *xpixel,* Int_t *ypixel,* Axis_t& *x,* Axis_t& *y*)
virtual Float_t	**PixeltoY**(Int_t *py*)
virtual void	**Pop**( )
virtual void	**Print**(Text_t* *filename,* Option_t* *option*)
virtual void	**Print**(Text_t* *filename*)
virtual void	**Range**(Float_t *x1,* Float_t *y1,* Float_t *x2,* Float_t *y2*)
virtual void	**RangeAxis**(Axis_t *xmin,* Axis_t *ymin,* Axis_t *xmax,* Axis_t *ymax*)
virtual void	**RecursiveRemove**(class TObject* *obj*)
virtual void	**ResetView3D**(class TPadView3D* *view = 0*)
virtual void	**ResizePad**(Option_t* *option*)
virtual void	**SaveAs**(Text_t* *filename*)
virtual void	**SetBorderMode**(Short_t *bordermode*)
virtual void	**SetBorderSize**(Short_t *bordersize*)
virtual void	**SetCanvas**(UInt_t *ww,* UInt_t *wh*)
virtual void	**SetCursor**((*unknown*) ECursor cursor)
virtual void	**SetDoubleBuffer**(Int_t *mode = 1*)
virtual void	**SetGrid**(Int_t *valuex = 1,* Int_t *valuey = 1*)
virtual void	**SetGridx**(Int_t *value = 1*)
virtual void	**SetGridy**(Int_t *value = 1*)
virtual void	**SetLogx**(Int_t *value = 1*)
virtual void	**SetLogy**(Int_t *value = 1*)

virtual void	**SetLogz**(Int_t *value = 1*)
virtual void	**SetName**(Text_t* *name*)
virtual void	**SetNumber**(Int_t *number*)
virtual void	**SetPad**(Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup*)
virtual void	**SetPad**(Text_t* *name,* Text_t* *title,* Float_t *xlow,* Float_t *ylow,* Float_t *xup,* Float_t *yup,* Color_t *color = 35,* Short_t *bordersize = 5,* Short_t *bordermode = -1*)
virtual void	**SetPadPointer**(class TObject* *padpointer = 0*)
virtual void	**SetPhi**(Float_t *phi = 30*)
virtual void	**SetSelected**(class TObject* *obj*)
virtual void	**SetTheta**(Float_t *theta = 30*)
virtual void	**SetTitle**(Text_t* *title*)
virtual void	**SetView**(class TView* *view*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)
virtual void	**Update**( )
virtual Int_t	**UtoAbsPixel**(Float_t *u*)
virtual Int_t	**UtoPixel**(Float_t *u*)
virtual Int_t	**VtoAbsPixel**(Float_t *v*)
virtual Int_t	**VtoPixel**(Float_t *v*)
virtual void	**x3d**(Option_t* *option*)
virtual Int_t	**XtoAbsPixel**(Axis_t *x*)
virtual Float_t	**XtoPad**(Axis_t *x*)
virtual Int_t	**XtoPixel**(Axis_t *x*)
virtual void	**XYtoAbsPixel**(Axis_t *x,* Axis_t *y,* Int_t& *xpixel,* Int_t& *ypixel*)
virtual void	**XYtoPixel**(Axis_t *x,* Axis_t *y,* Int_t& *xpixel,* Int_t& *ypixel*)
virtual Int_t	**YtoAbsPixel**(Axis_t *y*)
virtual Float_t	**YtoPad**(Axis_t *y*)
virtual Int_t	**YtoPixel**(Axis_t *y*)

Data Members

See Also: TPad

Class Description

```
TVirtualPad is an abstract base class for the Pad and Canvas classes.
```

~TVirtualPad( )

```
VirtualPad destructor
```

Author: René Brun 05/12/95
Last update: 1.03/03 04/09/97 17.14.36 by Victor Perev

TWarningDialog

class **TWarningDialog** : public TDialogManager

protected:

class TWarningDialog	**TWarningDialog**( )

public:

class TWarningDialog	**TWarningDialog**(char* *name*)
class TWarningDialog	**TWarningDialog**(class TWarningDialog&)
void	**~TWarningDialog**( )
class TClass*	**Class**( )
virtual const char*	**ClassName**( )
virtual Widget	**CreateDialog**(Widget *parent*)
virtual class TClass*	**IsA**( )
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

Class Description

```
TWarningDialog

This class provides an interface to warning dialogs.
```

const char* **ClassName**( )

Author: Fons Rademakers 01/04/95
Last update: 1.00/06 22/08/97 11.38.49 by René Brun

TWbox

class **TWbox** : public TBox

public:

class TWbox	**TWbox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color = 18,* Short_t *bordersize = 5,* Short_t *bordermode = 1*)
class TWbox	**TWbox**( )
class TWbox	**TWbox**(class TWbox& *wbox*)
void	**~TWbox**( )
class TClass*	**Class**( )
void	**Copy**(class TObject& *wbox*)
virtual void	**Draw**(Option_t* *option*)
virtual void	**DrawWbox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color = 33,* Short_t *bordersize = 5,* Short_t *bordermode = -1*)
virtual void	**ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)
Short_t	**GetBorderMode**( )
Short_t	**GetBorderSize**( )
Int_t	**GetDarkColor**( )
Int_t	**GetLightColor**( )
virtual class TClass*	**IsA**( )
virtual void	**Paint**(Option_t* *option*)
virtual void	**PaintFrame**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color,* Short_t *bordersize,* Short_t *bordermode,* Bool_t *tops*)
virtual void	**PaintWbox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t *color = 33,* Short_t *bordersize = 5,* Short_t *bordermode = -1*)
virtual void	**Print**(Option_t* *option*)
virtual void	**SavePrimitive**(class ofstream& *out,* Option_t* *option*)
virtual void	**SetBorderMode**(Short_t *bordermode*)
virtual void	**SetBorderSize**(Short_t *bordersize*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

protected:

Short_t	**fBorderSize**	*window box bordersize in pixels*
Short_t	**fBorderMode**	*Bordermode (-1=down, 0=no border, 1=up)*

See Also: TFrame, TSliderBox, TVirtualPad

Class Description

```
a TWbox is a TBox with a bordersize and a bordermode.
```

TWbox( ): **TBox**( )

```
wbox default constructor
```

TWbox(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t color, Short_t bordersize, Short_t *bordermode*):**TBox**(*x1, y1, x2, y2*)

```
wbox normal constructor
a WBOX is a box with a bordersize and a bordermode
the bordersize is in pixels
bordermode = -1 box looks as it is behind the screen
bordermode = 0  no special effects
bordermode = 1  box looks as it is in front of the screen
```

~TWbox( )

```
wbox default destructor
```

void **Copy**(TObject &*obj*)

```
Copy this wbox to wbox
```

void **Draw**(Option_t **option*)

```
Draw this wbox with its current attributes
```

void **DrawWbox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t color, Short_t bordersize, Short_t *bordermode*)

```
Draw this wbox with new coordinates
```

void **ExecuteEvent**(Int_t *event,* Int_t *px,* Int_t *py*)

```
Execute action corresponding to one event
This member function is called when a WBOX object is clicked.
```

void **Paint**(Option_t **option*)

```
Paint this wbox with its current attributes
```

void **PaintWbox**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,*Color_t color, Short_t bordersize, Short_t *bordermode*)

```
Draw this wbox with new coordinates
```

void **PaintFrame**(Coord_t *x1,* Coord_t *y1,* Coord_t *x2,* Coord_t *y2,* Color_t color, Short_t bordersize, Short_t bordermode,Bool_t *tops*)

```
Paint a 3D frame around a box.
```

void **Print**(Option_t **option*)

```
Dump this wbox with its attributes
```

void **SavePrimitive**(ofstream &*out,* Option_t **option*)

```
Save primitive as a C++ statement(s) on output stream out
```

Author: René Brun 12/12/94
Last update: 1.03/01 20/08/97 09.09.13 by Piotr Golonka

TWebFile

class **TWebFile** : public TFile

private:

void	**Init**( )

public:

class TWebFile	**TWebFile**(char* *url*)
class TWebFile	**TWebFile**(class TUrl *url*)
class TWebFile	**TWebFile**(class TWebFile&)
void	**~TWebFile**( )
class TClass*	**Class**( )
virtual class TClass*	**IsA**( )
Bool_t	**IsOpen**( )
Bool_t	**ReadBuffer**(char* *buf,* int *len*)
void	**Seek**(Seek_t *offset,* enum TFile::ERelativeTo *pos* = *kBeg*)
virtual void	**ShowMembers**(class TMemberInspector& *insp,* char* *parent*)
virtual void	**Streamer**(class TBuffer& *b*)

Data Members

private:

class TUrl	**fUrl**	*URL of file*
Seek_t	**fOffset**	*seek offset*

Class Description

```
TWebFile

A TWebFile is like a normal TFile except that it reads its data
via a (slightly modified) apache web server. A TWebFile is a
read-only file.
```

TWebFile(const char **url*) : **fUrl**(url), **TFile**(*url,* "WEB")

```
Create a Web file object. A web file is the same as a read-only
TFile except that it is being read via a HTTP server. The url
argument must be of the form: http://host.dom.ain/file.root.
If the file specified in the URL does not exist or is not accessible
the kZombie bit will be set in the TWebFile object. Use IsZombie()
to see if the file is accessible.
```

TWebFile(TUrl *url*) : **fUrl**(url), **TFile**(url.**GetUrl**( ), "WEB")

```
Create a Web file object. A web file is the same as a read-only
TFile except that it is being read via a HTTP server. Make sure url
is a valid TUrl object.
If the file specified in the URL does not exist or is not accessible
the kZombie bit will be set in the TWebFile object. Use IsZombie()
to see if the file is accessible.
```

void **Init**( )

```
Initialize a TWebFile object.
```

Bool_t **IsOpen**( ) const

```
A TWebFile that has been correctly constructed is always considered open.
```

Bool_t **ReadBuffer**(char **buf,* int *len*)

```
Read specified byte range from remote file via HTTP daemon. This
routine connects to the remote host, sends the request and returns
the buffer. Returns kTRUE in case of error.
```

void **Seek**(Seek_t *offset,* ERelativeTo *pos*)

```
Set position from where to start reading.
```

Author: Fons Rademakers 17/01/97
Last update: 1.03/02 02/09/97 17.44.44 by René Brun

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z