



Published on *ROOT* (<http://root.cern.ch/drupal>)

[Home](#) > [Printer-friendly PDF](#) > [Printer-friendly PDF](#)

Architectural Overview

[architecture](#) ^[1] [overview](#) ^[2]

The backbone of the ROOT architecture is a layered class hierarchy with, currently, around [1200 classes](#) ^[3] grouped in about 60 frameworks (libraries) divided in [19 main categories](#) ^[4] (modules). This hierarchy is organized in a mostly single-rooted class library, that is, most of the classes inherit from a common base class [TObject](#) ^[5]. While this organization is not universally popular in C++, it has proven to be well suited for our needs (and indeed for almost all successful class libraries: Java, Smalltalk, MFC, etc.). It enables the implementation of some essential infrastructure inherited by all descendants of [TObject](#). However, we also can have classes not inheriting from [TObject](#) when appropriate (e.g., classes that are used as built-in types, like [TString](#) ^[6]).

The Class Categories

The classes in the [ROOT base category](#) ^[7] provide the most low-level building blocks of ROOT. For example, the [TObject](#) class, which implements common behaviour for all ROOT classes. The class [TClass](#) ^[8] and its helper classes that provide support for extended runtime type information. The storage manager [TStorage](#) ^[9] which handles all memory allocation and de-allocation operations and performs basic error checking (memory overwrites, etc.). The class [TFile](#) ^[10] which provides a hierarchical sequential and direct access persistent object store. The operating system abstraction layer [TSystem](#) ^[11] and the concrete OS interfaces [TUnixSystem](#) ^[12] and [TWinNTSystem](#) ^[13] concentrate all OS dependent behavior, like file system access, dynamic loading, error handling and networking for the different platforms supported by ROOT.

The classes in the [container category](#) ^[14] provide general purpose data structures like, [arrays](#) ^[15], [lists](#) ^[16], [sets](#) ^[17], [B-trees](#) ^[18], [maps](#) ^[19], etc., which are heavily used in the implementation of ROOT itself.

The [physics category](#) ^[20] provides classes that implement the [Feldman Cousins](#) ^[21] algorithm, [N-body phase space generator](#) ^[22], [Lorentz rotation](#) ^[23], [Lorentz vector](#) ^[24], [2-D vectors](#) ^[25], [3-D vectors](#) ^[26], etc.

The [matrix category](#) ^[27] provides classes for [lazy matrices](#) ^[28], [general matrices](#) ^[29] and [vectors](#) ^[30].

The [histogram category](#) ^[31] provides classes for advanced statistical data analysis, like [1D](#) ^[32], [2D](#) ^[33] and [3D](#) ^[34] histogramming of short, long, float or double values, with fixed or variable bin sizes, profile histograms and [formula](#) ^[35] evaluation.

The [minimization category](#) ^[36] provides classes that provide interfaces to [Minuit](#) ^[37], [Fumili](#) ^[38] and [linear fitter](#) ^[39] algorithms. ^[40]

The [tree and ntuple category](#) ^[41] contains the [tree system](#) ^[42]. The row-wise and column-wise ntuples have been one of the major strengths of the PAW system. Trees extend the concept of ntuples to all complex objects and data structures found in raw data, ESD's and AOD.s. The idea is that the same data model, same language, same style of queries can be used on all data sets in an experiment. Trees are designed to support not only complex objects, but also a very large number of them in a large number of files. Ntuples are simple trees with only simple types (int, float, double, etc.).

The [2D graphics category](#) ^[43] provides classes for low-level graphics primitives, like [lines](#) ^[44], [arrows](#) ^[45], [rectangles](#) ^[46], [ellipses](#) ^[47], [text](#) ^[48], [legends](#) ^[49], [annotations](#) ^[50], [text with Latex notation](#) ^[51], [splines](#) ^[52], etc., but also the higher level constructs like [pads](#) ^[53] and [canvases](#) ^[54]. They also handle basic [style](#) ^[55] and attribute management.

The [3D graphics category](#) ^[56] provides basic 3D graphics primitives, like [3D polylines](#) ^[57] and [3D polymarkers](#) ^[58] as well as higher level geometrical shapes ([boxes](#) ^[59], [cones](#) ^[60], [polygons](#) ^[61], [tubes](#) ^[62], etc.) which can be efficiently assembled into complex [geometries](#) ^[63].

The [image processing category](#) ^[64] provides classes such as [TASImage](#) ^[65] for image processing.

The [detector geometry category](#) ^[66] provides classes for building, browsing, tracking and visualizing detector geometries.

The code is independent from the Monte Carlo simulation packages and therefore it does not contain any constraints related to physics. However, the package defines a number of hooks for tracking, such as materials, magnetic field or track state flags, in order to allow interfacing to tracking MC's. The final goal is to be able to use the same geometry for several purposes, such as tracking, reconstruction or visualization, taking advantage of the ROOT features related to bookkeeping, I/O, histogramming, browsing and GUI's. The geometrical modeler [67] is the most important component of the package and it provides answers to the basic questions like "Where am I ?" or "How far am I from the next boundary ?", but also to more complex ones like "How far am I from the closest surface ?" or "Which is the next crossing along a helix ?".

The neural network category [68] provides classes for multi layer perceptrons [69], etc.

The graphical user interface category [70] provides all the classes needed to build a modern, cross-platform, GUI. There are classes for many basic widgets such as buttons [71], windows [72], dialogs [73] and menus [74] and higher level widgets.

[75]

The C++ interpreter [76], CINT, allows the construction of applications in which the user has to learn only one language, C++, to communicate with the system. The command language, macro language and programming language are all one and the same.

The networking category [77] provides classes for client [78] and server sockets [79] that allow to easily construct client/server applications.

The SQL interface [80] provides a simple, but powerful abstract interface to different SQL database servers (MySQL, SAPDB, PostgreSQL, Oracle).

The documentation classes [81] allow the creation of hyperized C++ header and source files, inheritance trees, class indices, macro's and session transcripts. Thanks to this facility almost everything in the ROOT system can be automatically documented and cross-referenced.

The TObject Class

Most ROOT classes are derived from the `TObject` class. The `TObject` class defines protocols (abstract methods) for comparing objects, for object inspection, for object I/O, for graphics hit detection and for object notification, to name just the most important ones.

The ROOT object I/O facility [82] supports the streaming of arbitrarily complex polymorphic data structures from memory to a buffer. This buffer can then be stored in a ROOT binary machine-independent file [10], an XML file [83] or send over the network. This functionality is based on the abstract `Streamer` method, which is overridden in subclasses to stream an object's instance variables. Circular structures are linearized, and multiple references to the same object are restored properly. Storing pointers is implemented by an object table, which assigns a unique identifier to each transmitted object. This identifier can be transferred to other address spaces or to permanent storage.

The Class Dictionary and Object Run-time Support

Even with the upcoming run-time type identification (RTTI) extension for C++, the run-time system does not provide any information about the class structure, the instance variables or the member functions of an object. Consequently, an additional mechanism had to be introduced to gather this information, in order to support `InheritsFrom`, `Inspect` and `Dump` methods, the object I/O facility and the automatic documentation system. ROOT uses the approach of associating with each class (via a static pointer) a special object describing its structure. These descriptors are instances of the class `TClass` [8] which is itself a subclass of `TObject`. The `TClass` objects store the following information about a class:

- The name and title of a class.
- The size of an instance in bytes.
- Its parent class(es).
- The names, types and descriptions of its instance variables.
- The names and signatures of its member functions.
- A source code reference to the definition and implementation part of the class.
- The address of the class' object factory method used to create a new object.

Because the C++ run-time system gives no access to type and structure information, the ROOT system uses a dictionary generator called **CINT** [76]. **CINT** parses the class header files and generates a dictionary (in the form of a C++ function). To link the CINT generated dictionary function to a class the programmer only has to add two pre-processor macros to the code. One macro, `ClassDef` [84], must be placed in the class definition file and the other macro, `ClassImp` [84], in the implementation file.

Source URL: <http://root.cern.ch/drupal/content/architectural-overview>

Links:

- [1] <http://root.cern.ch/drupal/category/package-context/architecture>
- [2] <http://root.cern.ch/drupal/category/package-context/overview>
- [3] <http://root.cern.ch/root/html/ClassIndex.html>
- [4] <http://root.cern.ch/root/html/index.html>
- [5] <http://root.cern.ch/root/html/TObject.html>
- [6] <http://root.cern.ch/root/html/TString.html>
- [7] http://root.cern.ch/root/html/CORE_BASE_Index.html
- [8] <http://root.cern.ch/root/html/TClass.html>
- [9] <http://root.cern.ch/root/html/TStorage.html>
- [10] <http://root.cern.ch/root/html/TFile.html>
- [11] <http://root.cern.ch/root/html/TSystem.html>
- [12] <http://root.cern.ch/root/html/TUnixSystem.html>
- [13] <http://root.cern.ch/root/html/TWinNTSystem.h>
- [14] http://root.cern.ch/root/html/CORE_CONT_Index.html
- [15] <http://root.cern.ch/root/html/TObjArray.html>
- [16] <http://root.cern.ch/root/html/TList.html>
- [17] <http://root.cern.ch/root/html/THashTable.html>
- [18] <http://root.cern.ch/root/html/TBtree.html>
- [19] <http://root.cern.ch/root/html/TMap.html>
- [20] http://root.cern.ch/root/html/MATH_PHYSICS_Index.html
- [21] <http://root.cern.ch/root/html/TFeldmanCousins.html>
- [22] <http://root.cern.ch/root/html/TGenPhaseSpace.html>
- [23] <http://root.cern.ch/root/html/TLorentzRotation.html>
- [24] <http://root.cern.ch/root/html/TLorentzVector.html>
- [25] <http://root.cern.ch/root/html/TVector2.html>
- [26] <http://root.cern.ch/root/html/TVector3.html>
- [27] http://root.cern.ch/root/html/MATH_MATRIX_Index.html
- [28] <http://root.cern.ch/root/html/TMatrixDLazy.html>
- [29] <http://root.cern.ch/root/html/TMatrixD.html>
- [30] <http://root.cern.ch/root/html/TVectorD.html>
- [31] http://root.cern.ch/root/html/HIST_HIST_Index.html
- [32] <http://root.cern.ch/root/html/TH1.html>
- [33] <http://root.cern.ch/root/html/TH2.html>
- [34] <http://root.cern.ch/root/html/TH3.html>
- [35] <http://root.cern.ch/root/html/TF1.html>
- [36] http://root.cern.ch/root/html/MATH_MINUIT_Index.html
- [37] <http://root.cern.ch/root/html/TMinuit.html>
- [38] <http://root.cern.ch/root/html/TFumili.html>
- [39] <http://root.cern.ch/root/html/TLinearFitter.html>
- [40] http://root.cern.ch/drupal/html/QUADP_Index.html
- [41] http://root.cern.ch/root/html/TREE_TREE_Index.html
- [42] <http://root.cern.ch/root/html/TTree.html#TTree:description>
- [43] http://root.cern.ch/root/html/GRAF2D_Index.html
- [44] <http://root.cern.ch/root/html/TLine.html>
- [45] <http://root.cern.ch/root/html/TArrow.html>
- [46] <http://root.cern.ch/root/html/TBox.html>
- [47] <http://root.cern.ch/root/html/TEllipse.html>
- [48] <http://root.cern.ch/root/html/TText.html>
- [49] <http://root.cern.ch/root/html/TLegend.html>
- [50] <http://root.cern.ch/root/html/TPaveLabel.html>
- [51] <http://root.cern.ch/root/html/TLatex.html>
- [52] <http://root.cern.ch/root/html/TSpline.html>
- [53] <http://root.cern.ch/root/html/TPad.html>
- [54] <http://root.cern.ch/root/html/TCanvas.html>
- [55] <http://root.cern.ch/root/html/TStyle.html>
- [56] http://root.cern.ch/root/html/GRAF3D_Index.html
- [57] <http://root.cern.ch/root/html/TPolyLine3D.html>
- [58] <http://root.cern.ch/root/html/TPolyMarker3D.html>
- [59] <http://root.cern.ch/root/html/TBRIK.html>
- [60] <http://root.cern.ch/root/html/TCONE.html>
- [61] <http://root.cern.ch/root/html/TPGON.html>
- [62] <http://root.cern.ch/root/html/TTUBE.html>
- [63] <http://root.cern.ch/root/html/TGeometry.html>
- [64] http://root.cern.ch/root/html/GRAF2D_ASIMAGE_Index.html
- [65] <http://root.cern.ch/root/html/TASImage.html>
- [66] http://root.cern.ch/root/html/GEOM_Index.html
- [67] <http://root.cern.ch/root/html/TGeoManager.html>
- [68] http://root.cern.ch/root/html/MATH_MLP_Index.html
- [69] <http://root/html/TMLPAnalyzer.html>
- [70] http://root.cern.ch/root/html/GUI_Index.html
- [71] <http://root.cern.ch/root/html/TGButton.html>
- [72] <http://root.cern.ch/root/html/TGWindow.html>
- [73] <http://root.cern.ch/root/html/TGTransientFrame.html>
- [74] <http://root.cern.ch/root/html/TGMenuBar.html>
- [75] http://root.cern.ch/drupal/html/GED_Index.html
- [76] http://root.cern.ch/root/html/CINT_Index.html
- [77] http://root.cern.ch/root/html/NET_Index.html
- [78] <http://root.cern.ch/root/html/TSocket.html>

[79] <http://root.cern.ch/root/html/TServerSocket.html>
[80] <http://root.cern.ch/root/html/TSQLServer.html>
[81] http://root.cern.ch/root/html/HTML_Index.html
[82] <http://root.cern.ch/drupal/inputoutput>
[83] <http://root.cern.ch/root/html/TXMLFile.html>
[84] <http://root.cern.ch/drupal/how-write-objects-file>