New Infrastructure Features Since ROOT 2001

Fons Rademakers

Plug-in Manager

- Where are plug-ins used?

```
TFile *rf = TFile::Open("rfio://castor.cern.ch/alice/aap.root")
TFile *df = TFile::Open("dcache://main.desy.de/h1/run2001.root")
```

- For example, to extend the base class TFile to be able to read RFIO files one needs to load the plug-in library libRFIO.so which defines the TRFIOFile class
- Protocol part of the file name URI triggers loading of plug-in. In these cases TRFIOFile and TDCacheFile objects are used, which both derive from TFile

Plug-in Manager

- Previously dependent on “magic strings” in source, e.g. in TFile.cxx:

```
if (!strncmp(name, "rfio:", 5)) {
   if (gROOT->LoadClass("TRFIOFile", "RFIO")) return 0;
   f = (TFile*) gROOT->ProcessLineFast(Form("new
                  TRFIOFile(\"%s\",\"%s\",\"%s\",%d)",
                  name, option, ftitle, compress));
} else if (!strncmp(name, "dcache:", 6)) {
```

- Adding case or changing strings requires code change and recompilation
- Not user customizable

Plug-in Manager (cont.)

- Plug-in manager solves these problems:

```
TPluginHandler *h;
if ((h = gROOT->GetPluginManager()->FindHandler("TFile", name))) {
   if (h->LoadPlugin() == -1) return 0;
   f = (TFile*) h->ExecPlugin(4, name, option, ftitle, compress);
}
```

- Single if-statement to handle all cases
- No magic strings in code anymore

Plug-in Manager (cont.)

- Magic strings moved to system.rootrc file

```
# base class    regexp     plugin class  plugin lib  ctor or factory
Plugin.TFile:   ^rfio:     TRFIOFile     RFIO        "TRFIOFile(const
                                 char*,Option_t*,const char*,Int_t)"
+Plugin.TFile: ^dcache:   TDCacheFile   DCache      "TDCacheFile(const
                                 char*,Option_t*,const char*,Int_t)"
```

- Adding plug-in or changing strings does not require code change and recompilation
- Can be customized by user in private .rootrc file

Plug-in Manager (cont.)

- Currently 29 plug-ins are defined for 20 different (abstract) base classes
- Plug-in handlers can also be registered at run time, e.g.:
  - ```
    gROOT->GetPluginManager()->AddHandler("TSQLServer",
       "^sapdb:","TSapDBServer", "SapDB",
       "TSapDBServer(const char*)");
    ```
- A list of currently defined handlers can be printed using:
  - ```
    gROOT->GetPluginManager()->Print();
    ```

ROOT Build System

- To build ROOT on any platform do:
  - ./configure <platform>; make; make install
- We don’t use autoconf and automake since most platform ifdef’s are already in the source, and we already have figured out how to build shared libraries on all platforms, but the idea is the same
- The configure script tries to discover the right versions of the external libraries needed by the system
- If a right external library is not found the corresponding component is not build

Makefile Structure

- The ROOT Makefile has been structured as described in the paper: "Recursive Make Considered Harmful“
  - **http://www.tip.net.au/~millerp/rmch/recu-make-cons-harm.html**
- The main philosophy is that it is better to have a single large Makefile describing the entire project than many small Makefiles, one for each sub-project, that are recursively called from the main Makefile. By cleverly using the include mechanism the single Makefile solution is as modular as the recursive approach without the problems of incomplete dependency graphs.

Makefile Features

- The single Makefile is FAST
  - about 0.5 sec to check if anything needs to be recompiled on a 2GHz P4 (for 60 directories and 1400 files)
- The Makefile supports parallel builds
  - make –j 24 on FermiLab's SGI's
- The ROOTBUILD shell variable can be used to set debug build option:
  - export ROOTBUILD=debug
  - can also be set via --build option in configure

Important Makefile Targets

- make all (default)
- make install (install to path specified in ./configure)
- make dist (binary tar.gz distribution)
- make redhat (build binary rpm, by Christian Holm)
- make debian (build binary pkg, by Christian Holm)
- make distsrc (source tar.gz)
- make distclean (clean everything except configure info)
- make maintainer-clean (distclean + remove configure info)
- make cintdlls (build all CINT add-on dll’s)
- make html (generate HTML documentation of classes)

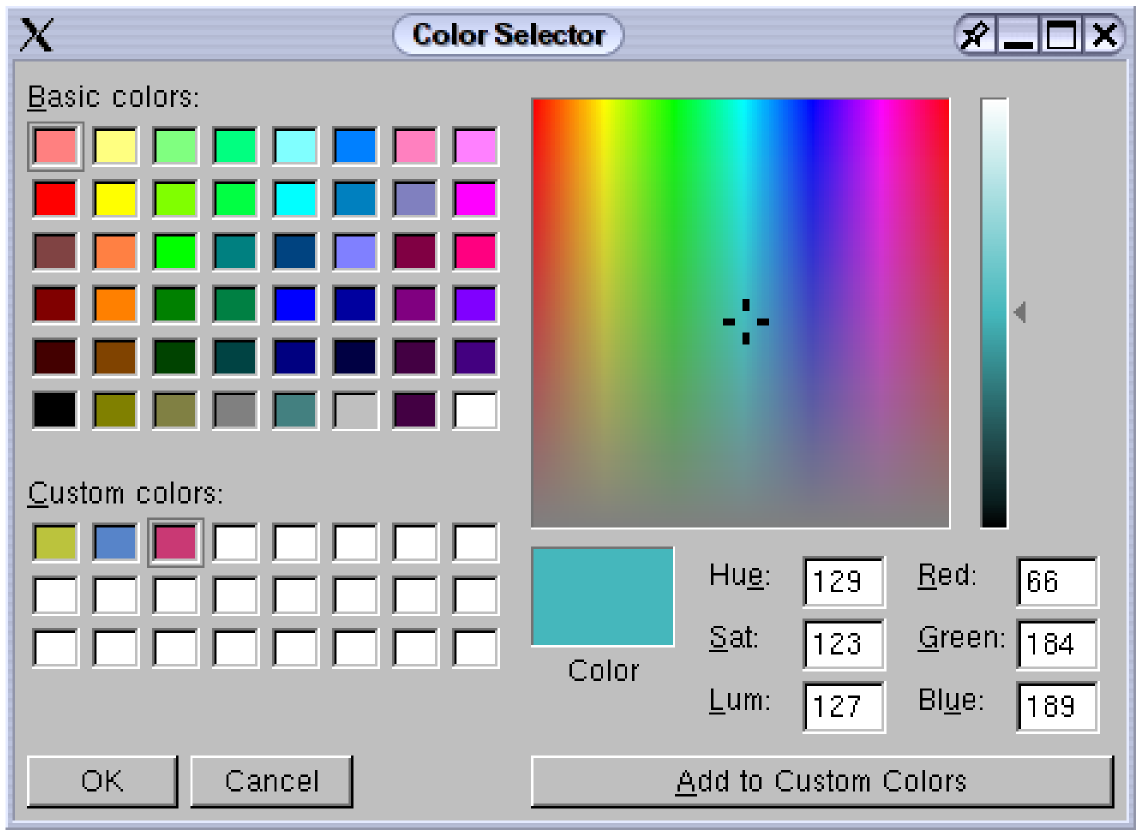
- make all-<module> (builds everything for specified module)
- make distclean-<module> (clean everything for specified module)

Supported Platforms

- New OS's and CPU's since last year:
  - MacOS X
  - GNU/Hurd
  - Itanium 1 and 2
- New compilers since last year:
  - Intel's icc for ia-32 and ecc for ia-64
    - Remarkable compiler: about 30% faster than gcc for ROOT
    - For Linux the C/C++ and Fortran compilers can be downloaded for free as "Non-commercial Unsupported Software". See: http://developer.intel.com/software/products/compilers/ and http://developer.intel.com/software/products/eval/
- Total of 10 different CPU's, 12 OS's and 11 comp.

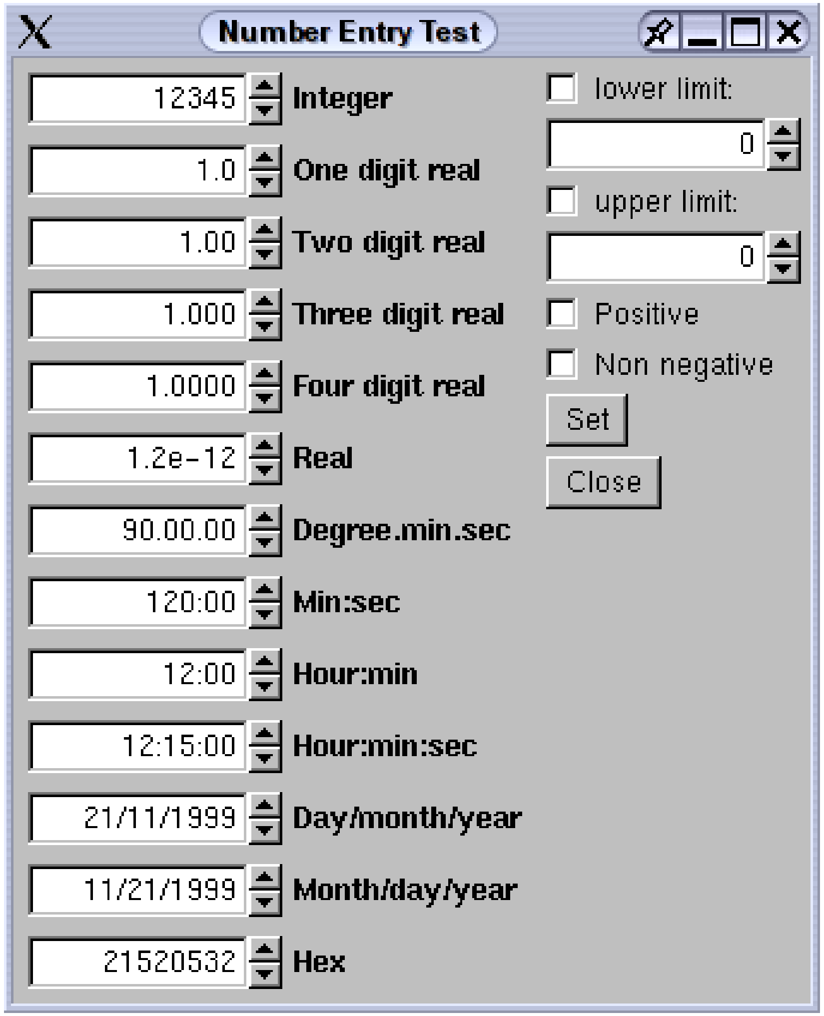
New ROOT GUI Widgets

- Color selector dialog: TGColorDialog



New ROOT GUI Widgets

- Number entry widget: TGNumberEntry



Support for HSM Systems

- Two popular HSM systems are now supported:
  - CASTOR
    - developed by CERN, file access via RFIO API and remote rfiod
  - dCache
    - developed by DESY, files access via dCache API and remote dcached

```
TFile *rf = TFile::Open("rfio://castor.cern.ch/alice/aap.root")
TFile *df = TFile::Open("dcache://main.desy.de/h1/run2001.root")
```

TGrid – Abstract Interface to GRIDs

```
class TGrid : public TObject {
public:
   virtual Int_t        AddFile(const char *lfn, const char *pfn) = 0;
   virtual Int_t        DeleteFile(const char *lfn) = 0;
   virtual TGridResult *GetPhysicalFileNames(const char *lfn) = 0;
   virtual Int_t        AddAttribute(const char *lfn,
                                     const char *attrname,
                                     const char *attrval) = 0;
   virtual Int_t        DeleteAttri

   virtual TGridResult *GetAttribu
   virtual void         Close(Opti

   virtual TGridResult *Query(cons

   static TGrid *Connect(const cha
                          const cha   pw = 0);

   ClassDef(TGrid,0)  // ABC defining interface to GRID services
};
```

Class TAlien concrete implementation for AliEn (http://alien.cern.ch)

Authentication Issues

- Support for kerberos 5 authentication for the rootd and proofd daemons
  - Implemented by Johannes Muelmenstaedt of MIT on special request by FermiLab
- Adding support for GRID authentication services will be next
  - Gerri Ganis, LCG project

Work on Thread Safety

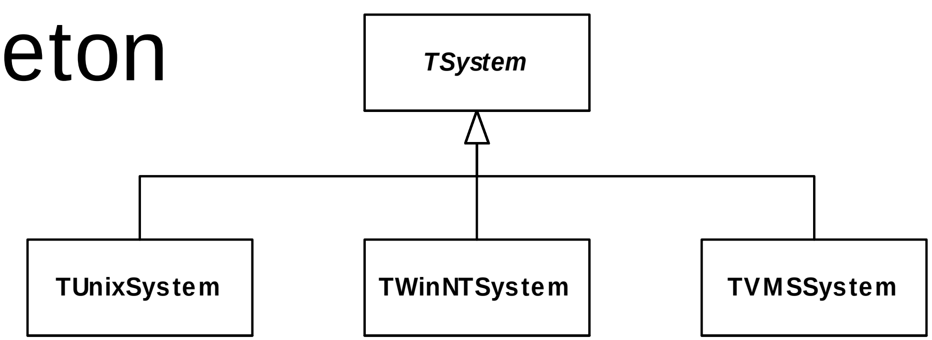
- Introduction of TVirtualMutex and TLockGuard classes in libCore
- Introduction of two global mutexes:
  - gCINTMutex and gContainerMutex
- After loading of libThread they will point to real TMutex objects, 0 otherwise
- Mutexes placed with TLockGuard via zero-cost macro (when not compiled with thread support)
- Work done by Mathieu de Naurois

New Infrastructure Classes

- Class TMD5
  - Implements the MD5 message-digest algorithm. Used to generate checksums of a bunch of bytes (like files or strings)
- Class TUUID
  - Implements a UUID (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier). A UUID is 128 bits long, and if generated according to this algorithm, is either guaranteed to be different from all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to be different

Refresh of ROOT Core System Services

- Many examples of user code not portable due to direct usage of Unix/Win32 OS system services
- Interface to operating system is provided via an abstract base class: TSystem
- Accessible via the gSystem singleton



TSystem Services

- TSystem provides:
  - System event handling
    - signal handling (TSignalHandler)
    - file and socket handling (TFileHandler)
    - timer handling (sync, async) (TTimer)
    - event processing and dispatching
  - Process control
    - fork, exec, wait, ...
  - File system access
    - file creation and manipulation
    - directory creation, reading, manipulation

More TSystem Services

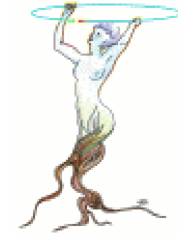
- Environment variable manipulation
  - getenv, putenv, unsetenv
- System logging
  - syslog interface
- Dynamic loading
  - load, unload, find symbol, ...
- RPC primivitves
  - open, close, option setting, read, write, ...

- Please check TSystem carefully for the right methods. Keep your code portable.

Refresh of Signals and Slots

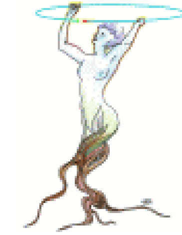
- Integration of signal and slot mechanism into the ROOT core
  - TQObject, TQConnection, TQClass, ...
- Signal and slots were pioneered by Trolltech in their Qt GUI toolkit
- This mechanism facilitates component programming since it allows a total decoupling of the interacting classes

Signals and Slots Example: Emitting a Signal



```
class A {
RQ_OBJECT(“A”)
private:
   Int_t  fValue;
public:
   A() { fValue = 0; }
   Int_t  GetValue() const { return fValue; }
   void   SetValue(Int_t);      //*SIGNAL*
};
```

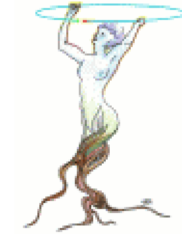
Signals and Slots Example: Emitting a Signal



```
void A::SetValue(Int_t v)
{
   if (v != fValue) {
      fValue = v;
      Emit("SetValue(Int_t)", v);
   }
}
```

```
void TGButton::Clicked()
{
   Emit(“Clicked()");
}
```

Signals and Slots Example: Connecting a Signal to a Slot



```
A *a = new A();
A *b = new A();
a->Connect("SetValue(Int_t)", "A", b, "SetValue(Int_t)");


a->SetValue(79);
b->GetValue();          // this is now 79
```

```
fButton->Connect("Clicked()", "MyFrame", this, "DoButton()");
```

Signals and Slots

- The ROOT signal and slot system uses the dictionary information and interpreter to connect signals to slots
- Many different signals are emitted by:
  - TVirtualPad (TCanvas and TPad)
  - TSysEvtHandler (TTimer, TFileHandler)
  - All GUI widgets
- Let your classes emit signals whenever they change a significant state that others might be interested in