TDataFrame

expressive and powerful ROOT data analyses

E. Guiraud D. Piparo
for the ROOT team

New Interfaces meeting 01/03/2017

Improving on current interfaces

```
TTreeReader reader(tree);
// typed proxies for branches x,y,z
TTreeReaderValue<A> x(reader,"x");
TTreeReaderValue<B> y(reader,"y");
TTreeReaderValue<C> z(reader,"z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

Improving on current interfaces

```
TTreeReader reader(tree);
TTreeReaderValue<A> x(reader,"x");
TTreeReaderValue<B> y(reader,"y");
TTreeReaderValue<C> z(reader,"z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

- users have full control over the event-loop
- needs some boilerplate
- running the event-loop in parallel is not trivial
- users implement trivial operations again and again

Improving on current interfaces

```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

```
TDataFrame data(tree, {"x","y","z"});


data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

- users have full control over the event-loop
- ✔ needs some boilerplate
- ✔ running the event-loop in parallel is not trivial
- ✔ users implement trivial operations again and again

Improving on current interfaces

```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

```
ROOT::EnableImplicitMT();
TDataFrame data(tree, {"x","y","z"});

data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

- users have full control over the event-loop

✔ needs some boilerplate

✔ running the event-loop in parallel is not trivial

✔ users implement trivial operations again and again

Improving on current interfaces

```
TFile f("f.root");
TTree *t = nullptr;
f.GetObject("myTree", t);
t->Draw("v1 >> h", "v2 > 0");
TH1F *h = nullptr;
gDirectory->GetObject("h", h);
```

Improving on current interfaces

```
TFile f("f.root");
TTree *t = nullptr;
f.GetObject("myTree", t);
t->Draw("v1 >> h", "v2 > 0");
TH1F *h = nullptr;
gDirectory->GetObject("h", h);
```

- powerful domain-specific language
- no compilation errors or warnings on analysis logic
- cannot run a debugger on the analysis
- running the event-loop in parallel is not trivial

Improving on current interfaces

```
TFile f("f.root");
TTree *t = nullptr;
f.GetObject("myTree", t);
t->Draw("v1 >> h", "v2 > 0");
TH1F *h = nullptr;
gDirectory->GetObject("h", h);
```

```
TDataFrame d("myTree", "f.root");
auto IsPositive = [](int n) { return n>0; };
auto h = d.Filter(IsPositive, {"v2"})
          .Histo1D("v1");
```

- ● powerful domain-specific language
- ✔ no compilation errors or warnings on analysis logic
- ✔ cannot run a debugger on the analysis
- ✔ running the event-loop in parallel is not trivial

Improving on current interfaces

```
TFile f("f.root");
TTree *t = nullptr;
f.GetObject("myTree", t);
t->Draw("v1 >> h", "v2 > 0");
TH1F *h = nullptr;
gDirectory->GetObject("h", h);
```

```
ROOT::EnableImplicitMT();
TDataFrame d("myTree", "f.root");
auto IsPositive = [](int n) { return n>0; };
auto h = d.Filter(IsPositive, {"v2"})
          .Histo1D("v1");
```

- ● powerful domain-specific language
- ✔ no compilation errors or warnings on analysis logic
- ✔ cannot run a debugger on the analysis
- ✔ running the event-loop in parallel is not trivial

TDataFrame goals

simple yet powerful way to analyse data with modern C++

provide high level features, e.g.
less typing, better expressivity, abstraction of complex operations

allow transparent optimisations, e.g.
multi-thread parallelisation and caching

C++ type-safe functional chains

1. build a data-frame object by specifying your data-set
2. apply a series of transformations to your data
   - filter (e.g. apply some cuts) or
   - create new columns
3. apply actions to the transformed data to produce results (e.g. fill histograms, profiles and more)

Overview: cut and fill

```
auto IsPos = [](double x) { return x > 0.; };
TDataFrame d("tree", "data2017_*.root");
auto h = d.Filter(IsPos,{"theta"}).Histo1D("pt");
h->Draw(); // event loop is run here
```

Overview: cut and fill

event-loop is run lazily, upon first access to the results

```
auto IsPos = [](double x) { return x > 0.; };
TDataFrame d("tree", "data2017_*.root");
auto h = d.Filter(IsPos,{"theta"}).Histo1D("pt");
h->Draw(); // event loop is run here
```

Overview: cuts and fills

```cpp
bool IsPos(double x) { return x > 0.; }
bool IsNeg(double x) { return x < 0.; }
TDataFrame d("tree", "file.root");
auto h1 = d.Filter(IsPos, {"theta"}).Histo1D("pt");
auto h2 = d.Filter(IsNeg, {"theta"}).Histo1D("pt");
h1->Draw();            // event loop is run once here
h2->Draw("SAME"); // no need to run loop again here
```

Overview: cuts and fills

all actions are executed in the same event-loop

```
bool IsPos(double x) { return x > 0.; }
bool IsNeg(double x) { return x < 0.; }
TDataFrame d("tree", "file.root");
auto h1 = d.Filter(IsPos, {"theta"}).Histo1D("pt");
auto h2 = d.Filter(IsNeg, {"theta"}).Histo1D("pt");
h1->Draw();          // event loop is run once here
h2->Draw("SAME"); // no need to run loop again here
```

Overview: parallelism

ImplicitMT should always be enabled when using TDF

```
ROOT::EnableImplicitMT();
bool IsPos(double x) { return x > 0.; }
bool IsNeg(double x) { return x < 0.; }
TDataFrame d("tree", "file.root");
auto h1 = d.Filter(IsPos, {"theta"}).Histo1D("pt");
auto h2 = d.Filter(IsNeg, {"theta"}).Histo1D("pt");
h1->Draw();            // event loop is run once here
h2->Draw("SAME"); // no need to run loop again here
```

Overview: AddColumn

```
double SlowCalc(double, double);
TDataFrame d("tree", "file.root");
double m = d.Filter(Cut, {"x","y"})
            .AddColumn("z", SlowCalc, {"x","y"})
            .Mean("z");
```

Overview: AddColumn

new quantities can be used as if they were tree branches

```
double SlowCalc(double, double);
TDataFrame d("tree", "file.root");
double m = d.Filter(Cut, {"x","y"})
            .AddColumn("z", SlowCalc, {"x","y"})
            .Mean("z");
```

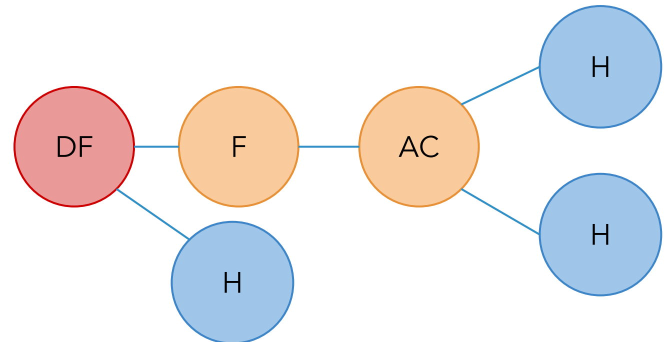
Overview: branching

```
TDataFrame d("tree", "file.root", {"x"});
// store a filtered data-frame with a new column
auto f = d.Filter([](double a) { return a > 0.})
          .AddColumn("z", Sum, {"x","y"});
// make multiple histograms out of it
auto hz = f.Histo1D("z");
auto hxy = f.Histo2D("x","y")
```

Overview: branching

not just functional chains, but functional *graphs*

```
TDataFrame d("tree", "file.root", {"x"});
// store a filtered data-frame with a new column
auto f = d.Filter([](double a) { return a > 0.})
          .AddColumn("z", Sum, {"x","y"});
// make multiple histograms out of it
auto hz = f.Histo1D("z");
auto hxy = f.Histo2D("x","y");
auto hy = d.Profile1D("x","y");
```



Performance

Multi-thread scaling

1 socket, 4 cores, 2 threads/core
Fill of TH1F and a TH2F w/ 10^7 evts
(each evt contained 200 ints for TH2F)

# workers	time (s)
0 (no IMT)	113.5 ± 9.4
2	50.3 ± 3.6
3	38.2 ± 0.6
4	30.8 ± 2.8
6	24.5 ± 0.9
8	24.5 ± 0.6

Fill 1 histogram

TTree::Draw	0.061 s
TDataFrame	0.056 s
TDataFrame x4	0.030 s

Fill 3 histograms

TTree::Draw (must do three loops)	0.190 s
TDataFrame	0.063 s
TDataFrame x4	0.036 s

Planned work

Short term

- entries ranges
- internal usage of jitting
- TDataFrame as tree generator

Mid term

- writing out processed trees
- string filter expressions
- pyROOT integration

Long term

- reading other data formats
  (GSoC project on Apache Parquet)
- df.Filter().AddColumn().TrainDNN()

Resources

TDataFrame user guide -> ROOT docs

https://root.cern.ch/doc/master/classROOT_1_1Experimental_1_1TDataFrame.html

Overview

Here is a quick overview of what actions are present and what they do. Each one is described in more detail in the reference guide.

In the following, whenever we say an action "returns" something, we always mean it returns a smart pointer to it. Also note that all actions are only executed for events that pass all preceding filters.

Lazy actions	Description
Count	Return the number of events processed.
Take	Build a collection of values of a branch.
Histo	Fill a histogram with the values of a branch that passed all filters.
Max	Return the maximum of processed branch values.
Mean	Return the mean of processed branch values.
Min	Return the minimum of processed branch values.
Reduce	Reduce (e.g. sum, merge) entries using the function (lambda, functor...) passed as argument. The function must have signature `T(T,T)` where `T` is the type of the branch. Return the final result of the reduction operation. An optional parameter allows initialization of the result object to non-default values.

Instant actions	Description
Foreach	Execute a user-defined function on each entry. Users are responsible for the thread-safety of this lambda when executing with implicit multi-threading enabled.
ForeachSlot	Same as `Foreach`, but the user-defined function must take an extra `unsigned int slot` as its first parameter. `slot` will take a different value, `0` to `nThreads - 1`, for each thread of execution. This is meant as a helper in writing thread-safe `Foreach` actions when using `TDataFrame` after `ROOT::EnableImplicitMT()`. `ForeachSlot` works just as well with single-thread execution: in that case `slot` will always be `0`.

Extra	Description
Report	This is not properly an action, since when `Report` is called it does not book an operation to be performed on each entry. Instead, it interrogates the data-frame directly to print a cutflow report, i.e. statistics on how many entries have been accepted and rejected by the filters. See the section on named filters for a more detailed explanation.

Resources

Tutorials

$ROOTSYS/tutorials/dataframe

discussion -> ROOT forum

https://root.cern/forum

bugs -> ROOT Jira

https://root.cern/bugs

contributions -> ROOT github

https://github.com/root-mirror/root

Thank you!