17-19 Oct, 2007 @ RCNS, Tohoku Univ., Sendai Day2 16:40-17:00 (20min)

Geant4 Tutorial @ Japan 2007

Histograming and Analysis using ROOT

Go Iwai, KEK/CRC

このセッションについて

- はじめてのROOT
  - HELLO WORLD
- Geant4からのROOTを使う
  - ここまでのおさらい
  - どこで何をするべきか
- オフラインでの表示
  - 対話的
  - マクロ

はじめてのROOT

Go Iwai, KEK/CRC

平成19年10月18日(木)

ROOTについて

- CERNのROOTチームが開発した解析ソフトウェア
  - 高エネルギー界隈ではPAW世代以降の人々によく使われている
- C++による開発
  - 同じくC++で記述されたGeant4との相性が良い
- ROOTで出来ること
  - ヒストグラムの作成・表示、他
    - これを1) マクロ、2) 対話モード、3) プログラムから実行することが出来る
- ROOTで出来ないこと
  - 日本語の表示
- ちょっと不便なこと
  - マクロにしろ、対話モードで使うにしろ、C++で命令しないと動かない
    - ただファイルを読んで点を打つだけなのに、20行くらい書かないといけない
  - 一方で、C++を学習するには良い教材とも言える
- PythonとRubyの言語バインディングも用意

HELLO WORLD (C++)

ソースコード

コマンドライン

hello.cc

```
#include <iostream>

int main()
{
  std::cout << “HELLO WORLD” << std::endl;
  return 0;
}
```

```
$ g++ -o hello.exe hello.cc    # コンパイルとリンク
$ ./hello.exe                  # プログラムの実行
HELLO WORLD                    # 実行結果
```

HELLO WORLD (CINT)

ソースコード

コマンドライン

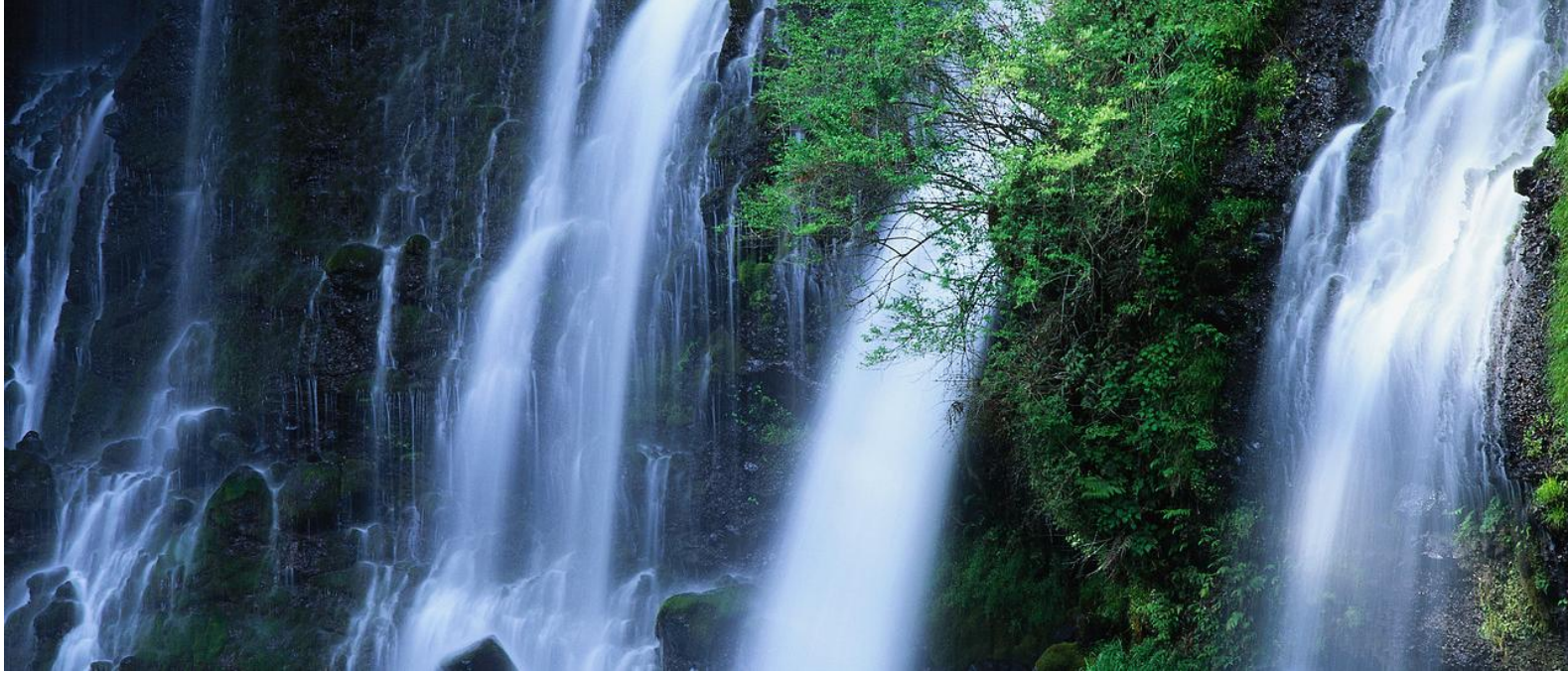
```
$ root                                                    # ROOTの起動
root [0] std::cout << “HELLO WORLD” << std::endl;         // 命令の実行
HELLO WORLD                                               // 実行結果
root [1] .q                                               // ROOTの終了
```

hello.C

```
void hello(const char* greeting = “HELLO WORLD”)
{
  std::cout << greeting << std::endl;
  return;
}
```

```
root [0] .x hello.C            // gROOT->Macro(“hello.C”)
HELLO WORLD                    // 実行結果
root [1] .q                    // ROOTの終了
```

```
root [0] .L hello.C            // gROOT->LoadMacro(“hello.C”)で代替可能
root [1] hello(“HELLO ROOT”)   // hello()関数の呼び出し
HELLO ROOT                     // 実行結果
root [2] .q                    // ROOTの終了
```

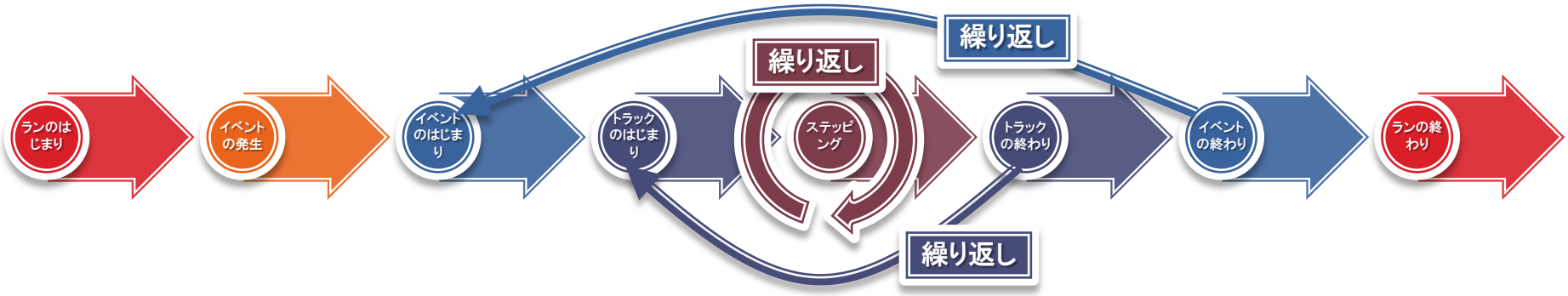


GEANT4からROOTを使う

Go Iwai, KEK/CRC

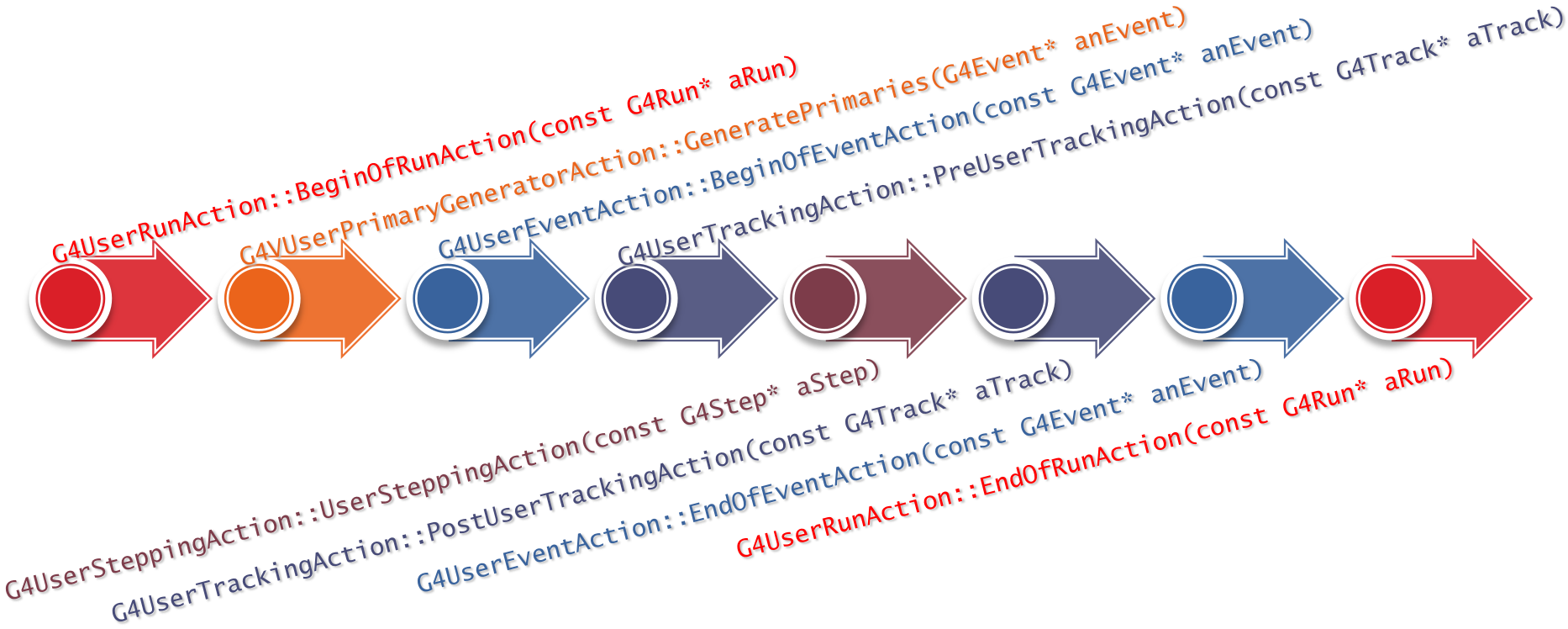
平成19年10月18日(木)

おさらい:ランの構成

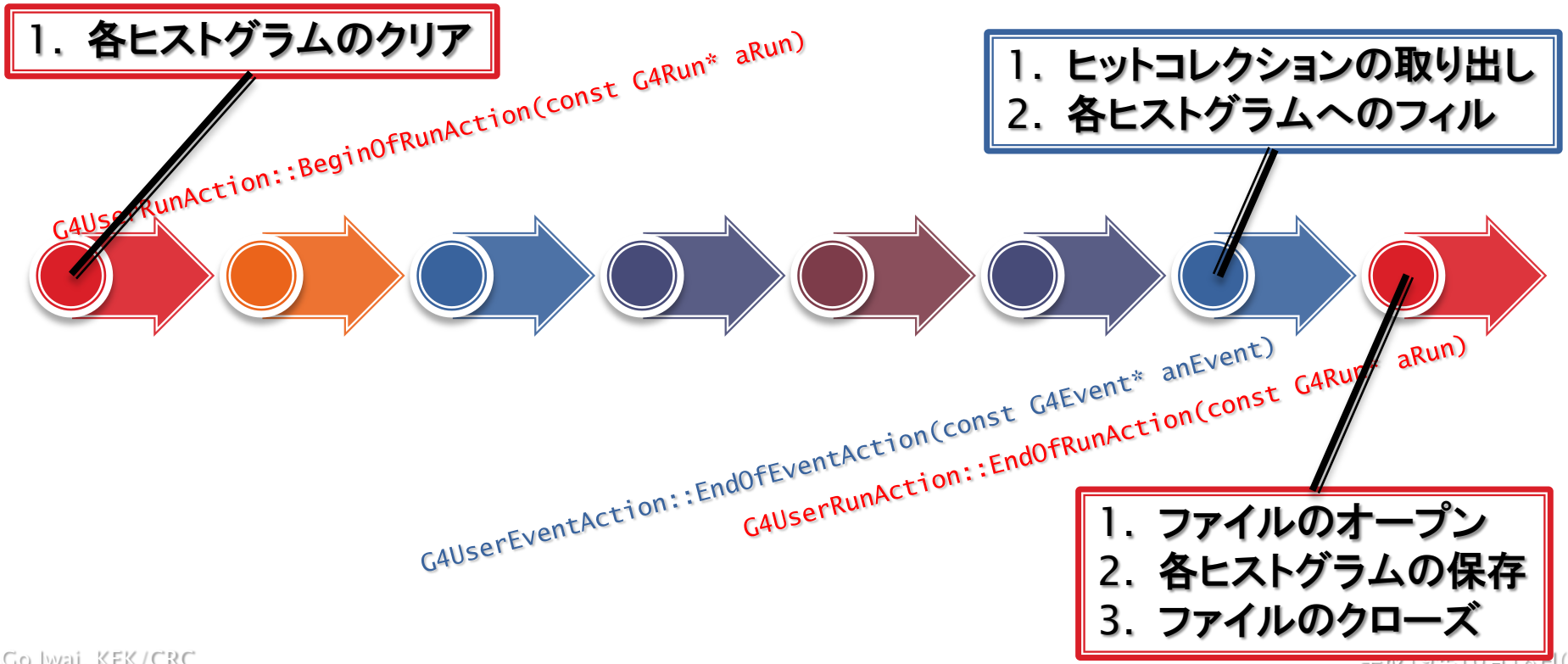


- いくつものステップがひとつのトラックを結び
- 複数のトラックによりイベントが構成され
- 複数のイベントがランを構成する
- Run = Stepping × i ~ Tracking × j ~ Event × k

おさらい：フック関数


G4UserRunAction::BeginOfRunAction(const G4Run* aRun)
G4VUserPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
G4UserEventAction::BeginOfEventAction(const G4Event* anEvent)
G4UserTrackingAction::PreUserTrackingAction(const G4Track* aTrack)
G4UserSteppingAction::UserSteppingAction(const G4Step* aStep)
G4UserTrackingAction::PostUserTrackingAction(const G4Track* aTrack)
G4UserEventAction::EndOfEventAction(const G4Event* anEvent)
G4UserRunAction::EndOfRunAction(const G4Run* aRun)

今回のポイント


1．各ヒストグラムのクリア
G4UserRunAction::BeginOfRunAction(const G4Run* aRun)
1．ヒットコレクションの取り出し
2．各ヒストグラムへのフィル
G4UserEventAction::EndOfEventAction(const G4Event* anEvent)
G4UserRunAction::EndOfRunAction(const G4Run* aRun)
1．ファイルのオープン
2．各ヒストグラムの保存
3．ファイルのクローズ

Geant4提供クラス

G4RunManager

```
G4UserRunAction* userRunAction
G4UserEventAction* userEventAction

void SetUserAction(G4UserRunAction*)
void SetUserAction(G4UserEventAction*)
```

登録はG4RunManagerに対して行う

G4UserRunAction

```
G4Run* GenerateRun()
void BeginOfRunAction(const G4Run*)
void EndOfRunAction(const G4Run*)
```

G4UserEventAction

```
void BeginOfEventAction(const G4Event*)
void EndOfEventAction(const G4Event*)
```

提供されるフック関数を派生先で上書き(オーバーライド)してアプリケーションの振る舞いを定義する(しなくともよい)

MyUserRunAction

```
G4Run* GenerateRun()

void BeginOfRunAction(const G4Run*)
void EndOfRunAction(const G4Run*)
```

ユーザ定義クラス

MyUserEventAction

```
void BeginOfEventAction(const G4Event*)

void EndOfEventAction(const G4Event*)
```

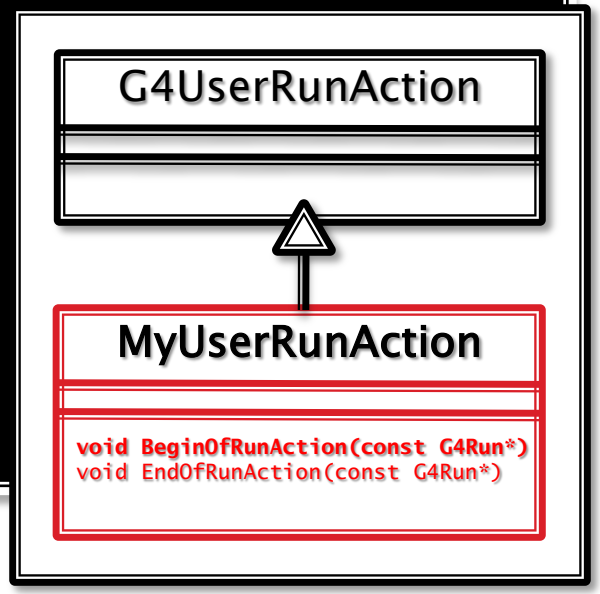
実装例（ランのはじまり）

ソースコード

コマンドライン

```
void MyRunAction::MyRunAction()
{
  theEdepHist = new TH1D("edep", "TITLE", 100, 0.0, 50.0);
  // edepはオブジェクト識別名(後述)
  // TITLEはヒストグラムのタイトル(後述)
  // 表示範囲は0から50まで、これを100ビンで区切る
}

void MyRunAction::BeginOfRunAction(const G4Run* aRun)
{
  theEdepHist->Reset();        // ヒストグラムのクリア
  return;
}
```



- ヒストグラムの器は一度だけ用意して使い回す
  - ラン毎に「作って～消して」を繰り返しても良い
- 誰(どのクラス)にヒストグラムオブジェクトを持たせるかは設計(趣味)の問題
- 他のクラスから参照される場合はアクセス手段(GetXXX関数)を用意しておく

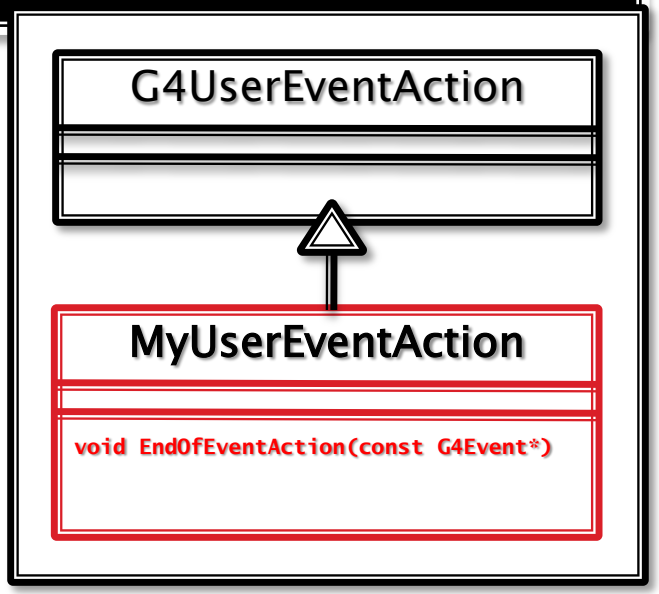
実装例（イベントの終わり）

ソースコード

コマンドライン

```
void MyEventAction::EndOfEventAction(const G4Event* anEvent)
{
  theEdepHist->Fill(total_edep);                    // ヒストグラムへのフィル
  return;
}
```

- ヒットコレクションから興味のある物理量を取り出し、ヒストグラムに対してフィルしてやる
- この例ではMyRunActionがヒストグラムオブジェクト持っているのでランマネージャ～ランアクションを経由してヒストグラムを取得する必要がある
  - G4RunManager::GetRunManager()->GetUserRunAction()
  - 必要に応じてMyRunActionへのキャストも行う
    - MyRunActionで追加された関数を使用する場合など



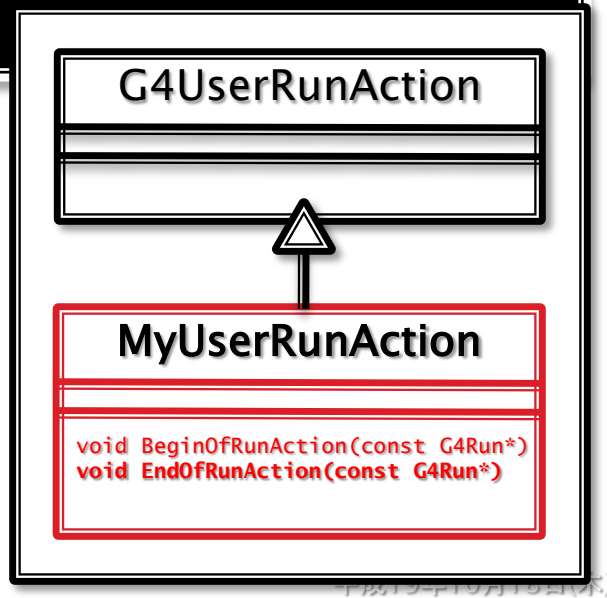
実装例（ランの終わり）

ソースコード

コマンドライン

```
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  TFile* f = new TFile("mydata.root", "RECREATE");      // ファイルのオープン
  theEdepHist->Write();                                 // ヒストグラムの保存
  delete f;                                             // ファイルのクローズ
  return;
}
```

- ラン毎にデータファイルを作る場合
- ヒストグラムが複数ある場合はそれぞれWrite()関数をコールする



GNUmakefile

ソースコード

コマンドライン

GNUmakefile

```
name := exampleXX
G4TARGET := $(name)
G4EXLIB := true

G4WORKDIR = .

.PHONY: all
all: lib bin

include $(G4INSTALL)/config/binmake.gmk

CPPFLAGS += $(shell $(ROOTSYS)/bin/root-config --cflags)
LDFLAGS += $(shell $(ROOTSYS)/bin/root-config --libs)
```

典型的なGeant4のMakefile

ROOTを使用するにはこの2行を追記

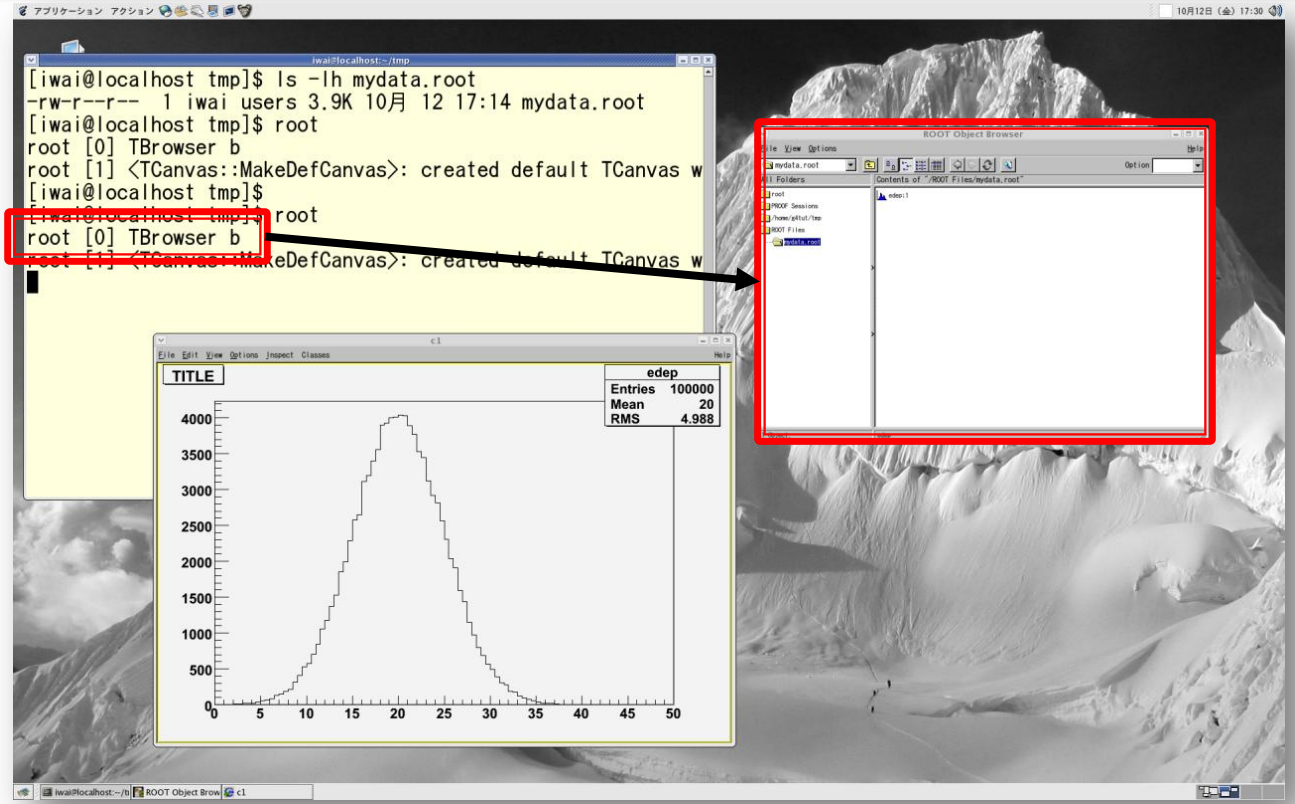
- make -n — 実際には処理を行わず、処理の内容だけを出力します
- make -k — エラーが生じても処理を続行します
- make -f filename — Makefile として使用するMakefileを指定します
- make -p — ビルトインルールを表示します
- make -jN — 同時にN個のコマンドを実行できるようにします

オフラインでの表示

Go Iwai, KEK/CRC

平成19年10月18日(木)

TBrowser


[iwai@localhost tmp]$ ls -lh mydata.root
-rw-r--r-- 1 iwai users 3.9K 10月 12 17:14 mydata.root
[iwai@localhost tmp]$ root
root [0] TBrowser b
root [1] <TCanvas::MakeDefCanvas>: created default TCanvas w
[iwai@localhost tmp]$
root [0] TBrowser b
TITLE
edep
Entries 100000
Mean 20
RMS 4.988
ROOT Object Browser

対話的に

ソースコード

コマンドライン

```
root [0] f = new TFile("mydata.root")                  // ファイルのオープン
root [1] cv = new TCanvas("cv", "canvas title")        // キャンバスの作成
root [2] edep->Draw()                                  // ヒストグラムの描画
root [3] cv->Print("mydata.png")                       // キャンバスの保存
```

- ヒストグラム作成時に指定したオブジェクト識別名"edep"がそのままヒストグラムオブジェクトとして使用可能
- キャンバスは自分で作らなくともDraw()をコールした段階で作成される
- キャンバスに描かれたものはPNG、GIF、PS形式で保存できる
  - Print()をコールしたときの拡張子で判別される

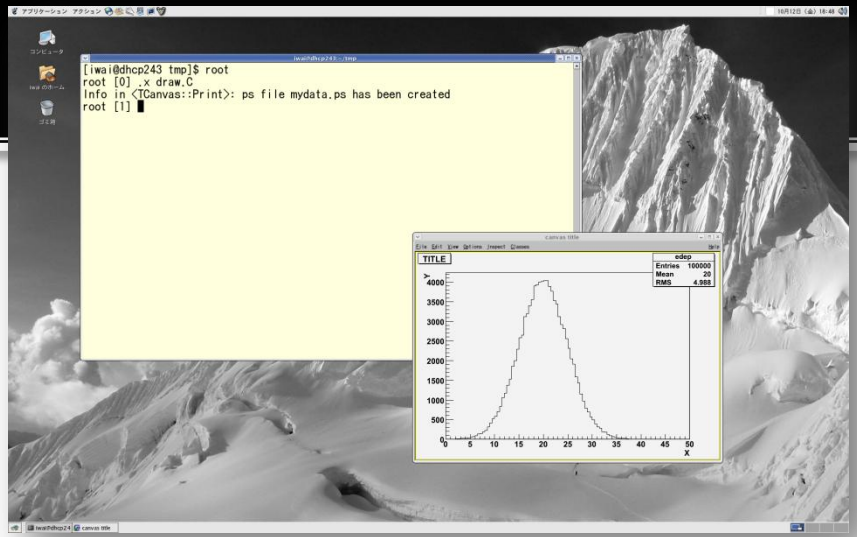
マクロの使用

ソースコード

コマンドライン

draw.cc

```
void draw()
{
  f = new TFile("mydata.root");
  cv = new TCanvas("cv", "canvas title");
  edep->GetXaxis()->SetTitle("X");          // X軸のタイトルを設定
  edep->GetYaxis()->SetTitle("Y");          // Y軸のタイトルを設定
  edep->Draw();
  cv->Print("mydata.ps");
  return;
}
```



まとめ～このセッションを振り返って

- CINT
  - マクロの実行　　*.x macro.C*
  - マクロのロード　　*.L macro.C*
- フックの利用
  - ヒストグラムのクリア　　*BeginOfRunAction()*
  - ファイルの保存　　*EndOfRunAction()*
  - ヒストグラムへのフィル　　*EndOfEventAction()*
- GNUmakefileの修正
  - *CPPFLAGS*, *LDFLAGS*を追記
- ヒストグラムの表示
  - ファイルを開く　　*f = new TFile("filename.root")*
  - ヒストグラムの識別名を指定して*Draw()*

おまけ

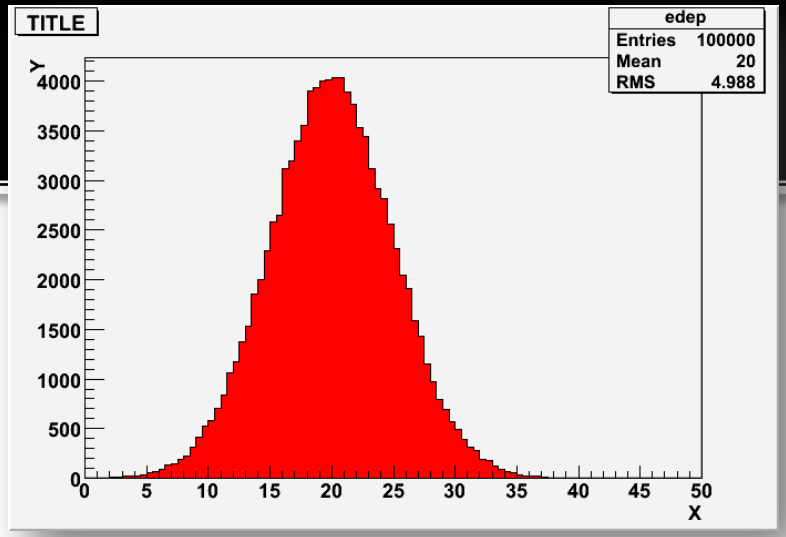
ヒストグラムの装飾

ソースコード

コマンドライン

draw2.C

```
void draw2()
{
  f = new TFile("mydata.root");
  cv = new TCanvas("cv", "canvas title");
  edep->GetXaxis()->SetTitle("X");
  edep->GetYaxis()->SetTitle("Y");
  edep->SetFillColor(kRed);
  edep->Draw();
  return;
}
```



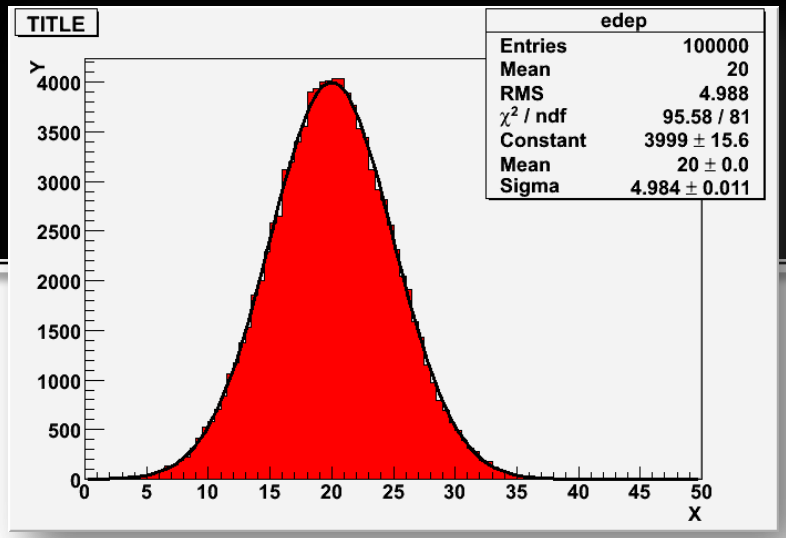
フィッティング

ソースコード

コマンドライン

draw3.C

```
void draw3()
{
  gStyle->SetOptFit(1);
  f = new TFile("mydata.root");
  cv = new TCanvas("cv", "canvas title");
  edep->GetXaxis()->SetTitle("X");
  edep->GetYaxis()->SetTitle("Y");
  edep->SetFillColor(kRed);
  edep->Draw();
  edep->Fit("gaus");
  return;
}
```



スケールの変更

ソースコード

コマンドライン

draw4.C

```
void draw4()
{
  gStyle->SetOptFit(1);
  f = new TFile("mydata.root");
  cv = new TCanvas("cv", "canvas title");
  cv->SetLogy(1);
  edep->GetXaxis()->SetTitle("X");
  edep->GetYaxis()->SetTitle("Y");
  edep->SetFillColor(kRed);
  edep->Draw();
  edep->Fit("gaus");
  return;
}
```