Introduction to SimTools

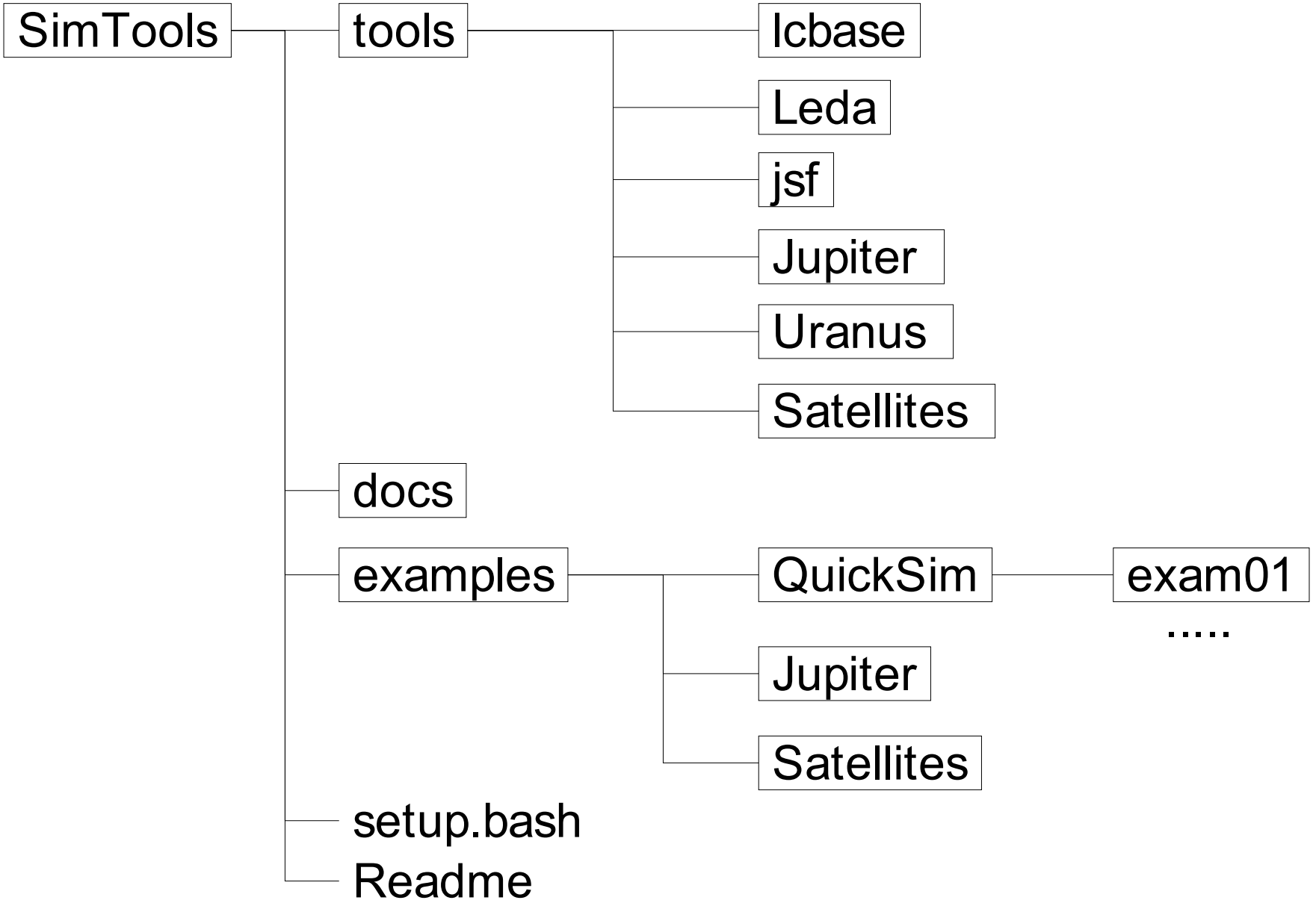
Akiya Miyamoto
KEK

16-17 March 2005

Get started with SimTools

- SimTools is a collection of precompiled binaries of JSF, Jupite r, Satellites and related package.
  - Compiled on Redhat 9 linux, using gcc 2.2.2
  - Requires ROOT-4.00.08
- Web site - http://ilcphys.kek.jp/soft/SimTools
- For installation,
  - Download from the web site
  - Edit setup.bash properly
  - Do "source setup.bash"

SimTools files



setup.bash

```
# .bashrc
# Following two lines are essential
export SIMTOOLS_DIR=/home/sample/SimTools
export ROOTSYS=/home/root/root-4.00.08

# next three lines are required to compile package
export JDK_HOME=/home/soft/JDK/j2sdk1.4.2_06
export G4INSTALL=/home/soft/Geant4/geant4.6.1
export LCIO=/home/soft/lcio/v01-03

.....
```

.rootrc

- ROOT uses .rootrc file to set configuration parameters.
- Following two parameters must be defined to run JSF.

```
Unix.*.Root.DynamicPath:  .:$(ROOTSYS)/lib:$(JSFROOT)/lib:$(LEDAROOT)/lib
Unix.*.Root.MacroPath:    .:$(ROOTSYS)/macros:$(JSFROOT)/macro
```

- .rootrc file is in your current directory or in your home directory. All example directories of SimTools package include .rootrc file.

JSF

- JSF provides common framework for studies using
  - Quick Simulator
  - Jupiter/Satellites
- To start interactive session of jsf, do

  $ jsf gui.C

  JSF control panel popes up

Using JSF Control Panel

- Controls menu
  - run mode
  - generator type
  - generator parameters
    - pythia
      - event type
        - zh
  - save parameters
- Next Event button

UserAnalysis.C

- Example in $JSFROOT/macro/UserAnalysis.C
- Three functions:
  - UserInitialize() : Called at Job initialization
    define Histgrams, etc.
  - UserAnalysis() : Called at each event
    for event analysis
  - DrawHist() : Called to draw histogram

jsf/ex1/UserAnalysis_1.C

Batch run

$ jsf -b -q --maxevt=100 gui.C

- root option:
  - -b : run without X
  - -q : quit at the end
- jsf option
  - --maxevt=N : N is number of events

Set parameters

- In a file, jsf.conf
- Command line arguments
  $ jsf -conf=conf_file --optionN=valueN .... gui.C
  - conf_file : a parameter file name
  - optionN=valueN: parameter name and its value

Format of jsf.conf

```
Parameter.name : value
#!option_name
# comment-1
# comment-2
```

JSF Features – 1

1. JSF is based on ROOT

   User needs to lean just one language, C+

2. JSF provides a framework for modular analyses

   Common framework for event generation, detector simulation, and analyses.

   Same framework for beam test data analysis

3. Unified framework for interactive and batch jobs

   GUI for control of an interactive run

   Histogram and event display packages included

   A file similar to .rootrc is used to set parameters

   Default values an be overidden by command line argument at run time.

JSF Features – 2

1. Object I/O

   Each modules can save/read their event data
   as branches of a root tree.
   Job parameters, histograms, ntuples and private
   analysis tree can be saved in the same file

2. Packages

   1) Included in the release

      Pythia6.2, Bases/Spring++, ZVTOP, JETNET, BSGEN

   2) Provided as separated packages

      Physsim (Event generators and analysis utilities)
      LCLIB (QuickSim, Helas)
      JIM (Geant3)
      Jupiter (Geant4)

Parameter file

All parameters are managed by JSFEnv class

In the user program, they are obtained by a class

JSFEnv::GetEnv(*"Parameter.Name"*, *default*)

At run time, parameter can be changed by three method

1. In a file, jsf.conf

   Parameter.Name : value
   #!*argname*
   # Comments
   .....

2. As a command line argument, like

   [%] jsf *--argname=value* gui.C

3. By popup menus of JSF Control panel

   PythiaGenerator: Type of process, CM energy, etc
   DebugGenerator: Particle ID, momentum, etc...

Each user can add their own menu by a function, *UserMenu()*

argname is an alias of *Parameter.Name*
used to parse command line argument

Concept of JSF run control

General feature of HEP data analysis:

1. Event-by-event analysis
2. Event data consists of several sub-components, analizer of them needs initialization and termination when job or run begins

Standard flow of JSF job

Create *modules*
Job Initialize
Begin run
Event Analysis
End Run
Job Termination

- Execution flow are controled by a class JSFSteer.
- One Modules are created, calls of their function are controled by JSFSteer
- Thus, inclusion/exclusion of analysis module is easy.

All analysis classes must be inherited from JSFModule and JSFEventBuf

JSFModule : provide functions such as

Initialize(), BeginRun(), Process(), EndRun(), Terminate()

JSFEventBuf : A class to save event data in a ROOT file as a tree

Access JSFModule and JSFEventBuf

- In scipt
  - JSFSteer *jsf (defined in gui.C)

```
jsf->GetEventNumber();
JSFXXX *mod=(JSFXXX*)jsf->FindModule("JSFXXX");
JSFXXXBuf *buf=(JSFXXXBuf*)mod->EventBuf();
```

- In compiled code,
  - JSFSteer *gJSF (defined in JSFSteer.h)

Build compiled library

- buildjsf command

jsf/ex2/buildjsf

JSF Components

1. Libraries

Pre-compiled C++ classes to build JSF application
Such as libJSF.a, libJSFQuickSim.a, ...

2. Executables (= jsf)

Root libraries + JSF libraries + Fortran libraries ⟹ jsf application

Fortran libaries = lclib, jlcsim, cernlib, ...

3. Macro

C++ program is used as Macro thanks to CINT, no need to compile and link

Macro can be used to set run parameters without compile/link

In the jsf distribution, gui.C, GUIMainMacro.C, and UserAnalysis.C are included as an analysis example

JSFGenerator

- JSFGenerator
- PythiaGenerator
- JSFBases - JSFSpring - JSFHadronizer
- JSFMEGenerator - JSFSHGenerator
  JSFReadMEGenerator - JSFPythiaHadronizer

PythiaGenerator

- Parameters
  - Process : ZH, ZZ, WW, enW, eeZ, gammaZ
  - BeamStrahlung
  - Decay: Z, W, H
- InitPythia.C

JSFGeneratorParticle

- Particle information
  ID, Mass, Charge, **P**, **X**, DL
  Pointers to Mother, 1st_Daughter, NDaughter
- Example
  - jsf/generator
    - using JSFGeneratorParticle
    - EventShape

Quick Simulator

Detectr components:

VTX, IMT, CDC, CAL are included.

Detector parameters (resolution,geometry, etc) can be changed be a parameter file

Signal generation:

Particles are swimmed through VTX, IMT, CDC, and CAL.

Particles are smeared by multiple scattering by matterials such as VTX, IMT, etc.

VTX and CDC

Equally spaced N sampling with given $\sigma_{r\phi}$ and σ_z in solenoid field

5 dimensional error matric of the track parameter are smeared including the effect of the multiple scattering due to chamber gas.

VTX and CDC parameters are then averaged to get combined helix parameter

IMT Just create smeared hit points

CAL: Particle energy is spread laterally by $f(x) = a_1 \exp(-|x|/\lambda_1) + a_2 \exp(-|x|/\lambda_2)$

Generated energy is distributed to each countes after smearing according to the resolution.

e and γ : Deposite energy only in EM calorimter

hadrons : Deposite energy only in HD calorimter

μ : No energy deposite in calorimeters

JSFQuickSim

- Quick Simulator module
  - Detector parameter file
    - $(LCLIBROOT)/simjlc/param/detect7.com
      -- "JLC-I" Green Book Detector (2 Tesla) , default
    - $(LCLIBROOT)/simjlc/param/jlc3T.com
      -- "ACFA Report" (3 Tesla)
  - JSFQuickSimParam : parameter class
  - JLCQuickSim.ParameterFile: env. param.
- Simulator Output data
  - JSFQuickSimBuf
    VTX (+IT), CDC, EMC, HDC, LTKCLTrack

JSFSIMDST(Buf)

- The format agreed among ACFA group.
- JSFQuickSIM + JSFGenerator
- Same information can be written to a file accesible by FORTRAN program.

Classes for QuickSim Output
JSFSIMDSTBuf

Important Member functions:

```
Int_t GetNLTKCLTracks();
Int_t GetNCDCTracks();
Int_t GetNVTXHits();
Int_t GetNEMCHits();
Int_t GetNHDCHits();
Int_t GetNSMHits();
Int_t GetNGeneratorParticles();

TObjArray *GetLTKCLTracks();   // Pointers to LTKCLTracks objects array
TClonesArray *GetCDCTracks();   // Pointers to CDCTracks object array
TClonesArray *GetVTXHits();      // Pointers to VTXhits object array
TClonesArray *GetEMCHits();      // Pointers to EMhits object array
TClonesArray *GetHDCHits();      // Pointers to HDhits object array
TClonesArray *GetSMHits();       // Pointers to SMhits object array
TClonesArray *GetGeneratorParticles();  // Pointers to GeneratorParticle objects array
```

JSFLTKCLTrack

- Information based on "Combined Track Bank"
  - http://www-jlc.kek.jp/subg/offl/lib/docs/cmbtrk/main.html
- Data in class
  - **P** at closest approach to IP
  - Particle type:

    1=Pure gamma, 2=Gamma in mixed EMC, 3=Pure neutral Hadron,
    4=Hadron in mixed HDC, 5=Pure charged hadron, 6=Unmached Track
    11=Electron candidate, 13=muon candidate
  - Source of information : 100*IHDC + 10*IEMC + ICDC
  - Nsig
  - Pointer to CDC Tracks

Anlib

- ANL4DVector: TLorentz , Lockable
- ANLEventSahpe
  - Using TObjArray of ANL4DVector
  - Calculate Thrust, Oblateness, Major/Minor Axis
- ANLJetFinder
  - base class for Jade, JadeE, Durham jet finder
- ANLJet : ANL4DVector

See examples in $(LEDAROOT)/Anlib/examples

JLCCVS

- Latest packages are available at jlccvs.kek.jp.
- How to get:

  $ cvs -d :pserver:anonymous@jlccvs.kek.jp/home/cvs/soft login <RETURN>
  Password: <RETURN>
  $ cvs -d :pserver:anonymous@jlccvs.kek.jp:/home/cvs/soft co jsf <RETURN>
- Update

  $ cvs update -P
- Web interface to see a code history

  http://jlccvs.kek.jp/cgi-bin/cvsweb.cgi/jsf/

Information on Web

- Home page of ACFA-Sim group
  http://acfahep.kek.jp/subg/sim
- Mailing list: acfa-sim@acfahep.kek.jp
  - JSF update information
- SimTools
  - http://acfahep.kek.jp/subg/sim/simtools