## Contents

1 The Tutorials and Tests 3
   1.1 $ROOTSYS/tutorials 4
   1.2 $ROOTSYS/test 5
Chapter 1

The Tutorials and Tests

This chapter is a guide to the examples that come with the installation of ROOT. They are located in two directories: $ROOTSYS/tutorials and $ROOTSYS/test.
The tutorials directory contains many example scripts. *To have all examples working you must have write permission and you will need to execute* `hsimple.C` *first*. If you do not have write permission in the directory `$ROOTSYS/tutorials`, copy the entire directory to your area. The script `hsimple.C` displays a histogram as it is being filled, and creates a ROOT file used by the other examples.

To execute it type:

```
$ cd $ROOTSYS/tutorials
$ root
```

1.2 $ROOTSYS/TEST

The test directory contains a set of examples that represent all areas of the framework. When a new release is cut, the examples in this directory are compiled and run to test the new release’s backward compatibility.

We see these source files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Makefile to build all test programs.</td>
</tr>
<tr>
<td>hsimple.cxx</td>
<td>Simple test program that creates and saves some histograms</td>
</tr>
<tr>
<td>MainEvent.cxx</td>
<td>Simple test program that creates a ROOT Tree object and fills it with some</td>
</tr>
<tr>
<td></td>
<td>simple structures but also with complete histograms.</td>
</tr>
<tr>
<td></td>
<td>This program uses the files Event.cxx, EventCint.cxx and Event.h. An example</td>
</tr>
<tr>
<td></td>
<td>of a procedure to link this program is in bind_Event. Note that the Makefile</td>
</tr>
<tr>
<td></td>
<td>invokes the rootcling utility to generate the Cling interface EventCint.cxx</td>
</tr>
<tr>
<td>Event.cxx</td>
<td>Implementation for classes Event and Track</td>
</tr>
<tr>
<td>minexam.cxx</td>
<td>Simple test program for data fitting</td>
</tr>
<tr>
<td>tcollex.cxx</td>
<td>Example usage of the ROOT collection classes</td>
</tr>
<tr>
<td>tcollbm.cxx</td>
<td>Benchmarks of ROOT collection classes</td>
</tr>
<tr>
<td>cttwist.cxx</td>
<td>Test program for the class TComplex</td>
</tr>
<tr>
<td>tstring.cxx</td>
<td>Example usage of the ROOT string class</td>
</tr>
<tr>
<td>vmatrix.cxx</td>
<td>Verification program for the TMatrix class</td>
</tr>
<tr>
<td>vvector.cxx</td>
<td>Verification program for the TVector class</td>
</tr>
<tr>
<td>vlazy.cxx</td>
<td>Verification program for lazy matrices</td>
</tr>
<tr>
<td>hworld.cxx</td>
<td>Small program showing basic graphics</td>
</tr>
<tr>
<td>gui_test.cxx</td>
<td>Example usage of the ROOT GUI classes</td>
</tr>
<tr>
<td>gui viewer.cxx</td>
<td>Another ROOT GUI example program</td>
</tr>
<tr>
<td>Hello.cxx</td>
<td>Dancing text example</td>
</tr>
<tr>
<td>Aclock.cxx</td>
<td>Analog clock (a la X11 xclock)</td>
</tr>
<tr>
<td>Tetris.cxx</td>
<td>The known Tetris game based on the ROOT graphics</td>
</tr>
<tr>
<td>stress.cxx</td>
<td>Important ROOT stress testing program</td>
</tr>
<tr>
<td>stress*.cxx</td>
<td>Stress testing of different ROOT classes</td>
</tr>
<tr>
<td>bench.cxx</td>
<td>STL and ROOT container test and benchmarking program</td>
</tr>
<tr>
<td>QpRandomDriver.cxx</td>
<td>Verification program for Quadratic programming classes in Quadp library</td>
</tr>
<tr>
<td>DrawTest.sh</td>
<td>Entry script to extensive TTree query test suite</td>
</tr>
<tr>
<td>dt_*</td>
<td>Scripts used by DrawTest.sh</td>
</tr>
</tbody>
</table>

The $ROOTSYS/test directory is a gold mine of root-wisdom nuggets, and we encourage you to explore and exploit it. These instructions will compile all programs in $ROOTSYS/test:

If you do not have write permission in the $ROOTSYS/test directory, copy the entire $ROOTSYS/test directory to your area. The Makefile is a useful example of how ROOT applications are linked and built. Edit the Makefile to specify your architecture by changing the ARCH variable, for example, on an SGI machine type: ARCH = sgikcc.

Now compile all programs:

```
% gmake
```

This will build several applications and shared libraries. We are especially interested in Event, stress, and gui_test.
1.2.1 Event - An Example of a ROOT Application

Event is created by compiling `MainEvent.cxx` and `Event.cxx`. It creates a ROOT file with a tree and two histograms. When running Event we have four optional arguments with defaults:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Events (1 ... n)</td>
</tr>
<tr>
<td>2</td>
<td>Compression level:</td>
</tr>
<tr>
<td>3</td>
<td>Split or not Split</td>
</tr>
<tr>
<td>4</td>
<td>Fill</td>
</tr>
</tbody>
</table>

1.2.1.1 Effect of Compression on File Size and Write Times

You may have noticed that a ROOT file has up to nine compression level, but here only levels 0, 1, and 2 are described. Compression levels above 2 are not competitive. They take up to much write time compared to the gain in file space. Below are three runs of Event on a Pentium III 650 MHz and the resulting file size and write and read times.

No Compression:

```
> Event 400 0 1 1
400 events and 19153182 bytes processed.
RealTime=6.840000 seconds, CpuTime=3.560000 seconds
compression level=0, split=1, arg4=1
You write 2.800173 Mbytes/Realtime seconds
You write 5.380107 Mbytes/Cputime seconds

> ls -l Event.root
... 19752171 Feb 23 18:26 Event.root
```

We see the file size without compression is 19.75 MB, the write time is 6.84 seconds and the read time is 0.79 seconds.

Compression = 1: event is compressed:

```
> Event 400 1 1 1
400 events and 19153182 bytes processed.
RealTime=6.440000 seconds, CpuTime=4.020000 seconds
compression level=1, split=1, arg4=1
You read 24.244533 Mbytes/Realtime seconds
You read 24.244533 Mbytes/Cputime seconds

We see the file size is 19.75 MB, the write time is 6.84 seconds and the read time is 0.79 seconds.
```

Compression = 2: event is compressed:

```
> Event 400 2 1 1
400 events and 19153182 bytes processed.
RealTime=0.790000 seconds, CpuTime=0.790000 seconds
You read 24.244533 Mbytes/Realtime seconds
You read 24.244533 Mbytes/Cputime seconds

We see the file size is 19.75 MB, the write time is 6.84 seconds and the read time is 0.79 seconds.
```
You write 2.974096 Mbytes/Realtime seconds
You write 4.764473 Mbytes/Cputime seconds

> ls -l Event.root
... 17728188 Feb 23 18:28 Event.root

> Event 400 1 1 20
400 events and 19153182 bytes processed.
RealTime=0.900000 seconds, CpuTime=0.900000 seconds
You read 21.281312 Mbytes/Realtime seconds
You read 21.281312 Mbytes/Cputime seconds

We see the file size 17.73, the write time was 6.44 seconds and the read time was 0.9 seconds.
Compression = 2: Floating point numbers are compressed with level 1:

> Event 400 2 1 1
400 events and 19153182 bytes processed.
RealTime=11.340000 seconds, CpuTime=9.510000 seconds
compression level=2, split=1, arg4=1
You write 1.688993 Mbytes/Realtime seconds
You write 2.014004 Mbytes/Cputime seconds

> ls -l Event.root
... 13783799 Feb 23 18:29 Event.root

> Event 400 2 1 20
400 events and 19153182 bytes processed.
RealTime=2.170000 seconds, CpuTime=2.170000 seconds
You read 8.826351 Mbytes/Realtime seconds
You read 8.826351 Mbytes/Cputime seconds

The file size is 13.78 MB, the write time is 11.34 seconds and the read time is 2.17 seconds.

This table summarizes the findings on the impact of compressions:

<table>
<thead>
<tr>
<th>Compression</th>
<th>File Size</th>
<th>Write Times</th>
<th>Read Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19.75 MB</td>
<td>6.84 sec.</td>
<td>0.79 sec.</td>
</tr>
<tr>
<td>1</td>
<td>17.73 MB</td>
<td>6.44 sec.</td>
<td>0.90 sec.</td>
</tr>
<tr>
<td>2</td>
<td>13.78 MB</td>
<td>11.34 sec.</td>
<td>2.17 sec.</td>
</tr>
</tbody>
</table>
1.2.1.2 Setting the Split Level

Split Level = 0:
Now we execute Event with the split parameter set to 0:

```c
> Event 400 1 0 1
> root
root[] TFile f("Event.root")
root[] TBrowser T
```

We notice that only one branch is visible (event). The individual data members of the Event object are no longer visible in the browser. They are contained in the event object on the event branch, because we specified no splitting.

Split Level = 1:

Setting the split level to 1 will create a branch for each data member in the Event object. First we execute Event and set the split level to 1 and start the browser to examine the split tree:

```c
> Event 400 1 1 1
```
1.2. $ROOTSYS/TEST

```
> root
root[] TFile f("Event.root")
root[] TBrowse browser
```

1.2.2 stress - Test and Benchmark

The executable stress is created by compiling `stress.cxx`. It completes sixteen tests covering the following capabilities of the ROOT framework.

- Functions, Random Numbers, Histogram Fits
- Size & compression factor of a ROOT file
- Purge, Reuse of gaps in `TFile`
- 2D Histograms, Functions, 2D Fits
- Graphics & PostScript
- Subdirectories in a ROOT file
- `TNtuple`, Selections, `TCutG`, `TEventList`
- Split and Compression modes for Trees
- Analyze `Event.root` file of stress 8
- Create 10 files starting from `Event.root`
- Test chains of Trees using the 10 files
- Compare histograms of test 9 and 11
- Merging files of a chain
- Check correct rebuilt of Event.root in test 13
- Divert Tree branches to separate files
- Cling test (3 nested loops) with LHCb trigger

The program stress takes one argument, the number of events to process. The default is 1000 events. Be aware that executing stress with 1000 events will create several files consuming about 100 MB of disk space; running stress with 30 events will consume about 20 MB. The disk space is released once stress is done.

There are two ways to run stress:

From the system prompt or from the ROOT prompt using the interpreter.

```sh
> cd $ROOTSYS/test
> stress // default 1000 events
> stress 30 // test with 30 events
```

Start ROOT with the batch mode option (-b) to suppress the graphic output.

```sh
> root -b
root[1] stress(1000) // test with 1000 events
root[1] stress(30) // test with 30 events
```

The output of stress includes a pass/fail conclusion for each test, the total number of bytes read and written, and the elapsed real and CPU time. It also calculates a performance index for your machine relative to a reference machine a DELL Inspiron 7500 (Pentium III 600 MHz) with 256 MB of memory and 18GB IDE disk in ROOTMARKS. Higher ROOTMARKS means better performance. The reference machine has 200 ROOTMARKS, so the sample run below with 53.7 ROOTMARKS is about four times slower than the reference machine.

Here is a sample run:

```sh
% root -b
root[1] .x stress.cxx(30)
```

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Functions, Random Numbers, Histogram Fits</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Check size &amp; compression factor of a Root file</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>Purge, Reuse of gaps in TFile</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>Test of 2-d histograms, functions, 2-d fits</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>Test graphics &amp; PostScript</td>
<td>OK</td>
</tr>
<tr>
<td>6</td>
<td>Test subdirectories in a Root file</td>
<td>OK</td>
</tr>
<tr>
<td>7</td>
<td>TNtuple, selections, TCutG, TEventList</td>
<td>OK</td>
</tr>
<tr>
<td>8</td>
<td>Trees split and compression modes</td>
<td>OK</td>
</tr>
<tr>
<td>9</td>
<td>Analyze Event.root file of stress 8</td>
<td>OK</td>
</tr>
<tr>
<td>10</td>
<td>Create 10 files starting from Event.root</td>
<td>OK</td>
</tr>
<tr>
<td>11</td>
<td>Test chains of Trees using the 10 files</td>
<td>OK</td>
</tr>
<tr>
<td>12</td>
<td>Compare histograms of test 9 and 11</td>
<td>OK</td>
</tr>
<tr>
<td>13</td>
<td>Test merging files of a chain</td>
<td>OK</td>
</tr>
<tr>
<td>14</td>
<td>Check correct rebuilt of Event.root in test 13</td>
<td>OK</td>
</tr>
<tr>
<td>15</td>
<td>Divert Tree branches to separate files</td>
<td>OK</td>
</tr>
<tr>
<td>16</td>
<td>Cling test (3 nested loops) with LHCb trigger</td>
<td>OK</td>
</tr>
</tbody>
</table>

******************************************************************
* IRIX64 fnpat1 6.5 01221553 IP27
******************************************************************

stress : Total I/O = 75.3 Mbytes, I = 59.2, O = 16.1
stress : Compr I/O = 75.7 Mbytes, I = 60.0, O = 15.7
stress : Real Time = 307.61 seconds Cpu Time = 292.82 seconds

******************************************************************
* ROOTMARKS = 53.7 * Root2.25/00 20000710/1022

******************************************************************
1.2.3 guitest - A Graphical User Interface

The guitest example, created by compiling guitest.cxx, tests and illustrates the use of the native GUI widgets such as cascading menus, dialog boxes, sliders and tab panels. It is a very useful example to study when designing a GUI. Some examples of the output of guitest are shown next. To run it type guitest at the system prompt in the $ROOTSYS/test directory. We have included an entire chapter on this subject where we explore guitest in detail and use it to explain how to build our own ROOT application with a GUI. See “Writing a Graphical User Interface”.

![Native GUI widgets](image)

Figure 1.1: Native GUI widgets