The ROOT Event Recorder/Player

B. Bellenot [1)], R. Brun [1)], Kateřina Opočenská [2)], F. Rademakers [1)]
1) CERN – European Organization for Nuclear Research, Geneva, Switzerland
2) Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

The ROOT framework offers a very complete set of GUI (Graphical User interface) widgets. In order to perform Quality Assurance (QA) and validation of the GUI, an event recorder has been developed, to automate the process on all the supported platforms (e.g. Linux/X11, Win32, MacOS X, …). The event recorder can also be used for tutorial purposes, or even as a bug report tool (e.g. to illustrate a weird behavior or a crash).

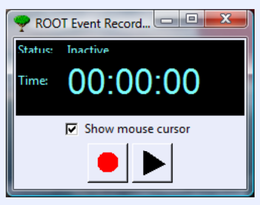
Features:
- Fully cross-platform: Sessions recorded on one specific platform can be replayed on any other supported platform.
- Small: Recorded sessions are saved in ROOT files, allowing to have long and complex GUI tests in a small file, in regard of e.g. capture video

Types of events recorded:
- Command line events
  - Commands typed by user (e.g. "new TBrowser();")
- GUI events
  - Window creation, resize, movement
  - Mouse movement, (double) click, drag & drop
  - …

RECORDING SESSION

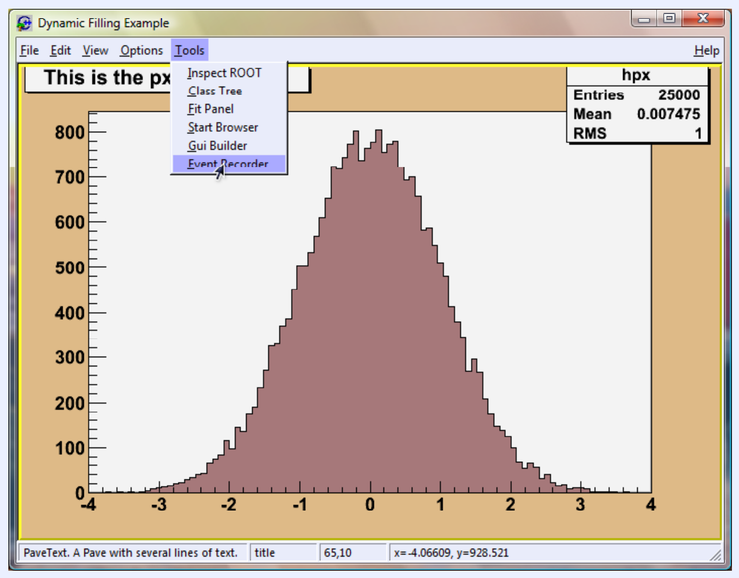
There are several way of using the recorder. Here are a few examples.

1. Start ROOT
2. Start recorder GUI: new TGRecorder(); this creates and display the recorder panel:

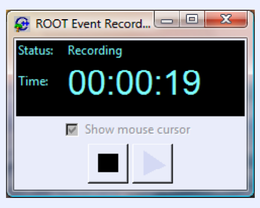


3. Push the ● button to start recording
4. In "Save As" dialog enter name for log file
5. Work with ROOT freely
   - Type commands in the ROOT prompt
   - Work with GUI
6. Press stop button on recorder to stop recording

1. Start ROOT
2. Execute a macro creating a canvas (e.g. hsimple.C from $ROOTSYS/tutorials)
3. From the “Tools” canvas menu, select “Event Recorder” Then, as previously:
4. Push the ● button to start recording
5. In "Save As" dialog enter name for log file
6. Work with ROOT freely
7. Press stop button on recorder to stop recording



When recording, the user interface of the recorder shows its current status (recording), and the recording time. The record button is also replaced by the stop button.



And here is an example of a command line recording session:

```
rec = new TRecorder();
rec->Start("record.root", "RECREATE");
Info in <TRecorderRecording::StartRecording>: Recording started. Log file: record.root
...
rec->Stop();
Info in <TRecorderRecording::Stop>: Recording finished.
```

When starting the recorder from a bare Root session, everything will be recorded, e.g. all commands and all graphic objects created during the session. When starting from an existing Root session, only the already opened canvases with their content will be saved.

REPLAYING SESSION

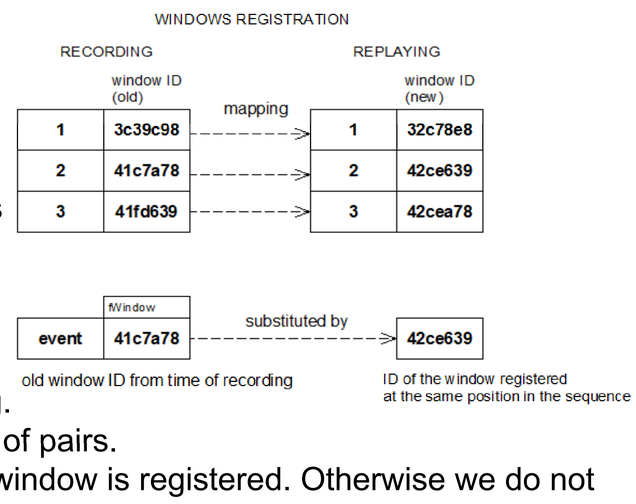
As for recording, replaying can be done via the command line or with the user interface, like for recording.

```
rec = new TRecorder();
rec->Replay("record.root");
Info in <TRecorderReplaying::Initialize>: Replaying of file bbb.root started
...
Info in <TRecorderReplaying::ReplayRealtime>: Replaying finished
```

ISSUES

- Each GUI event has an fWindow attribute with the ID of the window that it belongs to.
- When recording, fWindow is filled with the ID of a window that will not exist anymore at replaying time.

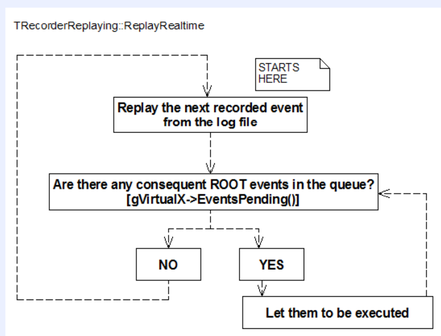
Then, how to determine which window to send an event to at replaying time?
The algorithm is simple and relies on the determinism of windows creation. When replaying, the windows state is gradually recreated exactly the same way as when recording. That means that the windows are always registered in the same order. During recording each window is registered at creation time. So the ID of each registered window is known. This ordered list of windows is stored in the log file together with events. When replaying, pairs of window IDs registered at the same position are created. Each window registered during recording is paired with each same ordered window registered during replaying.
When an event should be replayed, its original fWindow ID is replaced by the one of its partner in the list of pairs.
If an appropriate window is not found (has not been registered yet), replaying must be stopped until this window is registered. Otherwise we do not know which window the event belongs to. In this case, the status field of the player interface is displaying “Waiting...” in red.



How recorded and system events are ordered when replaying

If there is any system event in the queue, it should always be executed before the next recorded event is replayed.

So, at the time of recording, every system event should also be executed before the next user generated event is recorded.



Events have to be recorded at usual user speed to ensure that user caused and automatically generated events are recorded in the right order.

Usually, it is not hard to satisfy that because mouse movements and user clicks are "slow" in comparison to generating & processing events inside ROOT.

Anyway, if the user clicks a button and a new complex GUI (with many components) is open, then it is possible to make some action even before all the windows are registered, properly mapped, exposed etc.
When this kind of event happens, the status field of the player interface is displaying “Waiting...” in red.

For more information see: http://root.cern.ch
For any questions please use following address: rootdev@root.cern.ch