The (GRIM)REEPER

Ken Livingston

July 14, 2017

(**G**RIM **R**EEPER **I**mplements **M**acros to) **R**ead **E**PICS **E**vents and **P**lot **E**verything in **R**OOT

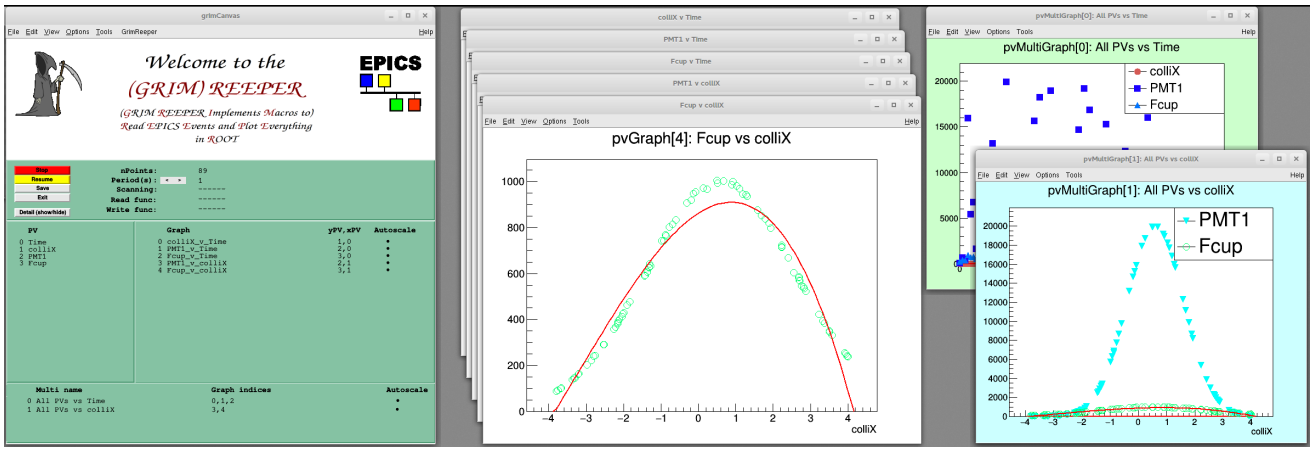

Figure 1: REEPER windows

1 Overview

(GRIM)REEPER, is a package for plotting EPICS PVs (Process Variables) in ROOT. At it's simplest it can be used as a strip chart, and, at it's most complex, as a general tool for modifying one or more EPICS PVs in a scan, reading back many PVs and analysing the result. The display (Figure 1) consists of one or more of the following:

Status and control window (left). This is optional.
Graphs (middle). One or more canvases, each with a graph of a PV vs Time, or PVy vs PVx.
MultiGraphs (Right). Showing one or more graphs on the same canvas.

It runs with a command line utility which has 2 modes:

```
reeper [OPTION]...  [PV1] [PV2] ...                          \\mode1
reeper [ROOT OPTIONS]...[file1.C] ...  fileN.C]              \\mode2
```

mode1 is for specification of PVs, scan parameters, fit fuctions etc to set up periodic reading of PVs and plotting of graphs and multiGraphs. The use of the `-w <start, stop, step>` option writes an updated value to PV1 before each time period (with caput commands).

mode2 is for customization using ROOT macros to call the REEPER functions directly.

Examples are given in Section 2 and full documentation in section 3.

2 Quickstart

Prequisites

You must be running on a system with a working ROOT installation, EPICS channel access and `caget, caput` in your path (and `softIOC`, if you want to try the simulated examples)

Setup

Source the relevant setup in your `.cshrc or .bashrc:`
`source <reeper_installation>/thisreeper.csh or <reeper_installation>/thisreeper.sh`

This sets the environment variable `$REEPER` to point to the installation directory, and adds it to the `$PATH`

Examples 1 - specifying PVs on the command line.

These examples use simulation mode (`-s` flag) which runs a softIOC. The `Exit` button on the control panel will kill the `softIOC` process before quitting ROOT, otherwise it may need to be killed manually, later. The status and control window is not shown, since it is common to all, and is documented in more detail below.The examples get progressively more fancy.

- Several different examples of specifying a single PV for plotting (Figure 2)

```
>reeper -s colliX                          //plot colliX vs Time
>reeper -s colliX::L                       //plot colliX vs time, join with line
>reeper -s colliX::C                       //plot colliX vs time, join with curve
>reeper -s colliX::pol3                    //plot colliX vs time, fit with pol3
>reeper -s colliX::pol3,10,20              //plot colliX vs time, fit with pol3 in range 10-20s
```

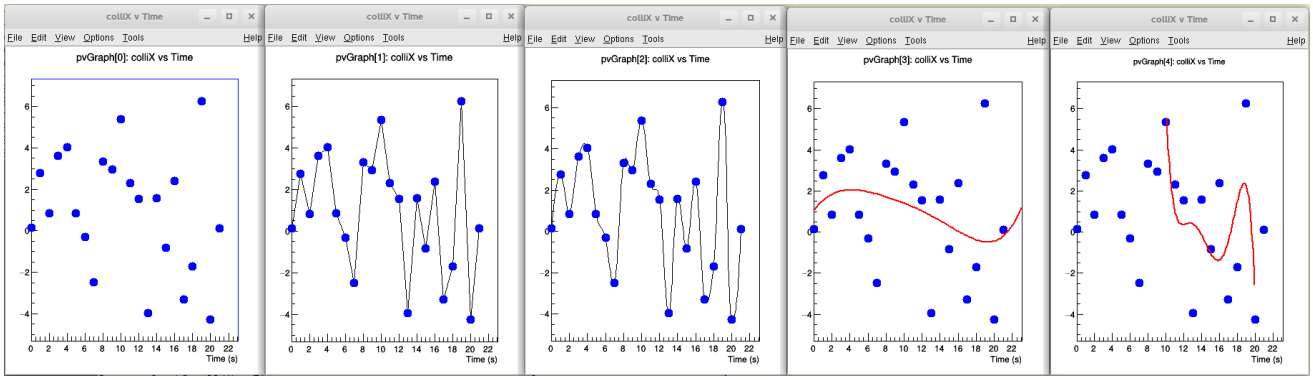

Figure 2: Several ways of plotting the graph of a PV using the command line tool

- Multiple PVs can be specified on the command line, with the option of writing (`caput`) to PV1 to do a scan (Figure 3)

```
>reeper -s -x colliX::pol3 PMT1##gaus,-4,4 Fcup##pol3   //All PVs vs Time and PMT1, Fcup vs colliX, with fits
>reeper -s -w -4,4,0.1 colliX::pol3 PMT1##gaus,-4,4 Fcup##pol3  //As above, but scan from -4 to +4 in colliX
```

  The `-x` option causes the creation of an extra set of graphs where all PVs are plotted vs PV1.
  The `-w [start,stop,step]` option causes the same behaviour, but actively controls PV1 (colliX) using `caput` commands.
  For the PVs where a fit is specified, the `::` delimiter means the fit is applied to the Time graph, and the `##` delimiter means it is applied to the PV1 graph. Two multiGraphs are also produced - one with all the Time graphs and one with all the PV1 graphs. The axes on all graphs will autoscale, by default, each time a new

point is added unless autoscaling it turned off (*toggle autoscaling* on the context menu on the graph's canvas using the RH mouse button). The colours and symbols are changed automatically for every new graph, in a tasteless and unpredictable manner, but can be improved by bringing up the editor from the ROOT toolbar.

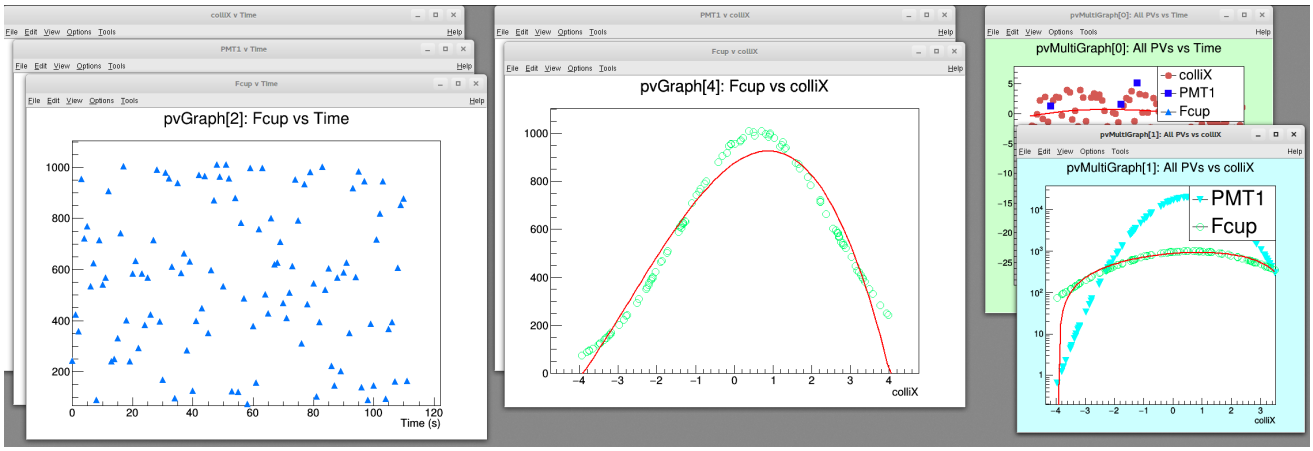

Figure 3: Multiple PVs, some with fits. Graphs and multiGraphs of all PVs vs Time, and vs PV1 are made

Examples 2 - specifying PVs in a ROOT macro

- The inclusion of one or more `.C` files on the command line opens ROOT with the REEPER library loaded and processes arguments in the standard ROOT way. For example:

```
>reeper grBasic.C              //Sets up a simple scan and plots one PV against another
>reeper grScanFancy.C          //Shows how to do a 2d scan, fill 2d histograms and read waveforms
```

  The files `grBasic.C` and `grScanFancy.C` are included in the `$REEPER` directory. The run using the built in simulation, and are well commented. They should be used as templates to help write macros to deal with real PVs.

The ROOT prompt is always available and the global variable nd be used to access the class members functions and data, which are all public. Yes - plenty of scope for screwing this up! This is all described in section 3.

3 Detailed documentation

A brief description with options, arguments and usage, is given by running `>reeper -h` (see 3.5). Some more detail is given here.

REEPER is a collection of functions and variables to do EPICS read/write, and plot the results. All the functions can be accessed from the ROOT prompt, or by using a ROOT macro. The command line utility `reeper` calls a specific set of these commands, then offers the ROOT prompt, where a knowledgeable user could type commands to generate more graphs, save canvases etc. Access to some commands is provided via the status / control panel, which is created by default when `reeper` is called, (or can be brought up by calling `makeControlGUI()` from the ROOT prompt).

The documentation below describes aspects of the functionality in an *inverse need to know* order (the more obscure and specialised information comes towards the end).

3.1 Control / Status Panel and context menu options

The control panel is shown in Figure 4 (left):

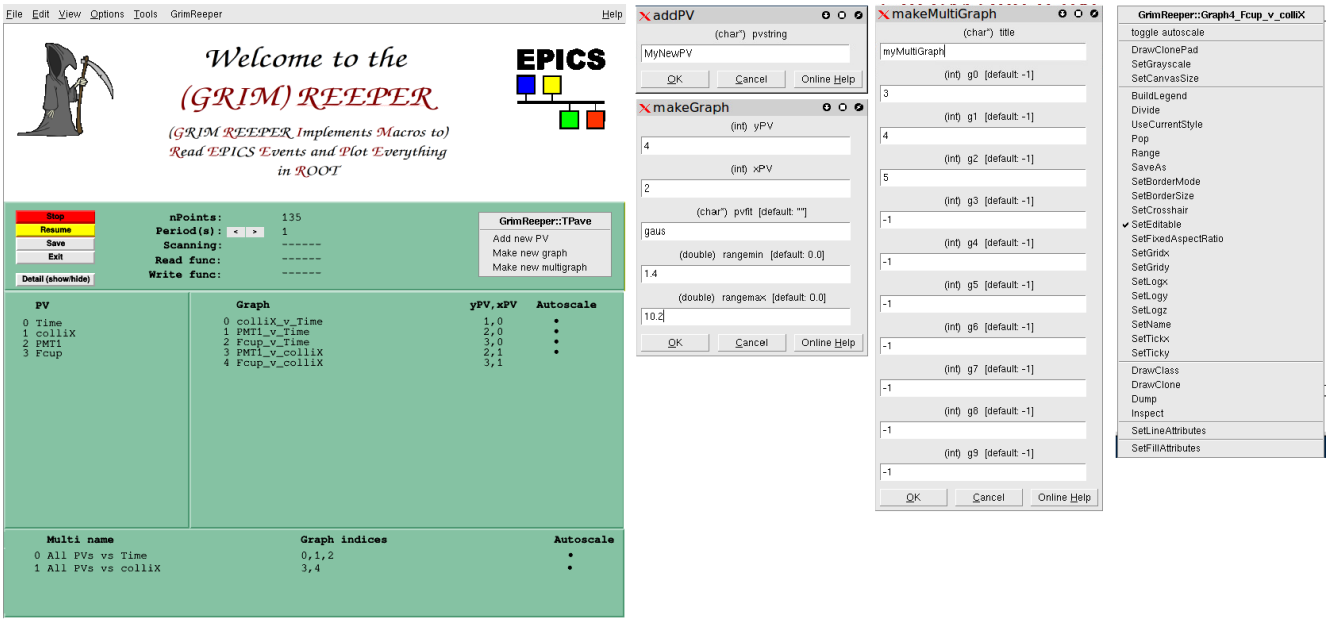

Figure 4: Control panel and context menus for panel and graphs / multiGraph canvases

- **Stop, Start, Pause/Resume, Exit.** Do the obvious.
- **Save.** Saves the data in several ways:
  - PVs are written a text file with columns. Eg.

```
# PV0 = Time
# PV1 = colliX
# PV2 = PMT1
# PV3 = Fcup
#step PV0     PV1         PV2       PV3
0     0.0000 -2.7656     69.3981 243.0320
1     1.0000 -2.0289    631.3830 423.8440
2     2.0000 -2.2597    335.1750 357.4400
3     3.0000 -0.0776 15897.6000 953.7850
4     4.0000  2.2213   5373.3400 722.6930
5     5.0000 -0.8775   6714.0500 768.8300
6     6.0000  2.8360   1642.0800 534.1220
....
....
```

    These can then be easily read back into ROOT as graphs (eg. for PV2, PV3):

```
g = new TGraph("file","%*s%*s%*s%lg%lg")
```

  - Graphs are written as `.root`, `.pdf`, `.gif` and `.C` files
  - By default they are written with timestamps like this into directory `epics_gr`, like this:

```
epics_gr/gr_save_15_05_17-11_07.txt,
epics_gr/Graph2_Fcup_v_Time_15_05_17-11_07.gif ...  etc.
```

  - The output directory can be changed that root prompt:

```
GR->setSaveDir("mydir")
```

  - The list of formats can also be changed: eg.

```
char *mySaveFormats[] = {"C","pdf",NULL}
GR->setSaveFormats(mySaveFormats)
```

- **nPoints**. Number of read events in the scan.
- **Period**. Time (s) between read. Can be changed with the arrows in increments of 1s.
- **Scanning**. If a PV is being controlled with caput commands, start,stop,step and current value are shown.
- **Read / Write func**. Details of user defined read/write functions.

- **Details**. This buttons toggles the visibility of the bottom half of the GUI, which shows the current lists of PVs, graphs and mulitGraphs.

Context Menus

Some useful REEPER functions are available via ROOT context menus (Figure 4 (right)). Context menus are brought up by clicking the right mouse button over an item; the options depend on the context (ie the entity over which the mouse was hovering when clicked).

- **Control panel context menu**. This is the `GrimReeper::TPave` menu, shown on top right of the green zone in Figure 4 (left). The 3 options each pop up a different input dialog box (also shown in the figure). They are simply calling functions that are part of REEPER, and could also be typed at the ROOT prompt:
  - **addPV**: Add a new PV. If this is mid scan, points relating to earlier time steps are set to zero for this PV. The new PV now becomes available for making graphs.
  - **makeGraph**: Create a graph from any 2 PVs specified by index. PV0 is always Time. There is the option to include a fit and range as illustrated in Section 2.
  - **makeMultiGraph**: Create a multiGraph from several graphs, specified by index. The x axis will be taken from the 1st graph.
- **Graph Canvas context menu**. This is the standard Graph canvas with an added option at the top:
  - **toggle autoscale**. This toggles the autoscaling on/off for the relevant graph or multiGraph. When autoscaling is on it will be applied every time the graph is redrawn with new data. The status of autoscaling can be seen in the bottom part of the control GUI.

3.2 Concepts, coding approach etc

In a counting house, EPICS experts might do quick PV scans using a combination of unix shell tools (sh, awk etc), piping the output to text files and opening ROOT to make any required graphs and fit the data. The grimReeper is an attempt to make that easier - particularly for users with limited ROOT and unix experience. The main design features are:

- **PV0 = Time**: The fundamental PV for the grimReeper is Time. PV0 is always Time(s), and scanning is done in time steps.
- **Command line use**: Basic behaviour is accessible with a simple shell command in the standard unix style. `reeper` is a shell(sh) script which parses args and opts and opens ROOT as required.
- **ROOT**: Once ROOT is opened, all standard functionality is available at the ROOT prompt; the global variable `grimReeper *GR`, allows control of scanning, visuals, etc and gives access to the raw PV data. GUI features are an *add-on*.
- **GUI**: Fancy ROOT GUIs are too complicated to produce and need lots of code. The simple GUI functionality here is using `TButton` and customization of `TContextMenu` items.
- **Compilation**: This is done *on the fly*, using ROOT's ACLiC features. No makefiles. The details can be seen in `reeper` and `grLoad.C`. Debugging can be done with gdb if required
  For example, see https://wiki.physik.uni-muenchen.de/etp/index.php/Debugging_root_with_gdb.

3.3 `Class grimReeper`

The grimReeper class definition is in `gr.C`, which also defines a global variable `grimReeper *GR` pointing to the (single) grimReeper object. This allows the member functions to be accessed from the commend line and from `TButtons`. For details look into the code. Everything is public, so lots of opportunity to mess things up! Some functions are worth a special mention, since they are useful for macros:

```
void setMAXPV(int maxpv)                \\set max no of PVs that can be added (default = 50)
void setMAXDATA(int data)               \\set max no of read points per PV (default = 10000)
Note:  The above functions must be called before init()
void intit();                           \\set up initial params, arrays etc.  This is called implicitly ny addPV()
void addPV(const char *pvstring);       \\add PV to the list being read (each call adds 8*MAXDATA bytes to memory used)
void makeGraph(int yPV, int xPV, ...);  \\add new graph (each will require memory dynamically up to 16*MAXDATA bytes)
int getPoints()                         \\number of PV scan points read so far
double **getPVData()                    \\2d array of PV data:  array[PVindex][point]
void setUserWrite(const char *write)    \\user function to be called to write PVs for a step
void setUserRead(const char *read)      \\user function to be called after scan step period elapsed
void waveToArray(const char *pv, ...);  \\read waveform into arrray
void waveToTH1D(const char *pv,...);    \\read waveform into 1D histogram bins 1D
void waveToTH2DRow(const char *pv,...); \\read waveform into a row of a 2D histogram
```

The use is discussed in the next section.

3.4 Custom scans from ROOT macro.

The basic (mode1) command line use of reeper described above can cover most situations where PVs are to be read and plotted, and can control a single PV to do a 1D scan. However, sometimes more complicated behaviour is required. For example, filling / fitting 2D hstograms, reading writing waveforms, modifying more than 1 PV during a scan. This is done by calling grimReeper class functions from within a macro (or compiled code). The required features are illustrated in example code `$REEPER/grScanFancy.C`. The macro must do the following:

1. Create an instance of grimReeper, select PVs to be read, make graphs, make canvases, histograms, graphs to be filled by user defined function(s).
2. Define the function(s) to be called to be called. My own preference is to have a single function:
   `myFunc(int mode)` where mode is `INIT(=0), READ(=1) or WRITE(=2)`
   The read and write modes are called before and after each step period in a scan

Here's the skeleton of `grScanFancy.C`:

```
//Some essential globals
GrimReeper* gr = NULL;                          //The GrimReeper class
int p = 0;                                      //number of points
double **pvdata = NULL;                         //All the acquired data
void myFunc(int mode = 0);                      //custom user function defined below

//Main function
void grScanFancy(){
   gr = new GrimReeper();                       //make the main class object, add PVs make graphs etc
   ....
   ....
   myFunc(GR_INIT);                             //Call myFunc(GR_INIT) to do init the specialised part
}

//Function called by GrimReeper
void myFunc(int mode){
   ....
   ....
   p = gr->getPoints();              //get the number of points read
   switch (mode){
   case GR_INIT:
     ....
     ....
     pvdata = gr->getPVData();       //get the address of the array with all the data points
     gr->setUserRead("myFunc(1)");   //These 2 lines set up REEPER to call myFunc() for Read and Write modes
     gr->setUserWrite("myFunc(2)");
     break;
   case GR_READ:                     //Called during a scan, after reading the PVs
     ....
     ....
     break;
   case GR_WRITE:                    //called during a scan after reading pvs (and before waiting Period time)
     ....
     ....
```

```
    break;
  default:
    break;
  }
}
```

3.5 Usage

```
reeper -h

NAME
        reeper - make ROOT graphs using data read from EPICS PVs


SYNOPSIS
        reeper [OPTION]...  [PV]...(mode 1)                                          //mode1
        reeper [ROOT OPTIONS]...[file1.C ...  fileN.C]                               //mode2

DESCRIPTION
        reeper [OPTION]...  [PV]...                                                  //mode1
        In this mode, make ROOT graphs by reading EPICS values periodically and plots graphs of all PVs against time.

        If -x or -w option is specified, additional graphs of all PVs against the 1st PV in the list are made
        Multigraphs with all graphs using the same x-axis are also created.
        See EXAMPLES 1, below

        -t period
             set the period (ie time(s)) between reads.  Default = 2s

        -x
             make additional graphs of all PVs against the 1st in the list

        -w start,stop,step
             As -x, but additionally, write to the 1st of the listed PVs to do a scan.  Must be 3 comma separated values.
             After each time period the next value will be written to the PV

        -q
             Quiet (and quick).  Won't pop up a GUI and won't ask for confrmation if scanning (with -w option).

        -s simulation mode.  For testing
             The following PVs are available:  PV23004,colliX,PMT1,Fcup,Xsetting,Ysetting,BigScaler,BigScaler,AMO_SCALERS
             These are updated by starting a softIOC and running a ROOT script.

        -h
             print this help message

        [PV]...
              EPICS PV. Can include an optional fit and fit range.  For example:
              MyScalerRate
              MyScalerRate:pol4               will fit 4th deg poly to the graph of this PV
              MyScalerRate:pol4,1.2,10.4      will fit 4th deg poly to the graph in the x-axis range 1.2 - 10.4
              If -x or -w option is selected, any specified fits will be applied to the graph of PV vs PV0
              Otherwise, the fit will be applied to the graph of PV vs Time
              For PV0, any specified fit will always be fitted vs Time
              Choosing a fit of L or C (no range needed) will join the points with a line(L) or curve(C)


        reeper [ROOT OPTIONS]...[file1.C ...  fileN.C] (mode2)

        In this mode call root with the reeper library loaded then set up customised behaviour in .C macros.
        See EXAMPLES 2, below.


EXAMPLES 1
            Note.  These examples use the -s flag, for simulated data.  Try them
            reeper -s colliX PMT1 Fcup             //makes graphs of each of the above PVs vs time
            reeper -s -x colliX PMT1 Fcup          //as above, but additionally makes graphs of PMT1 and Fcup vs colliX
            reeper -s -w -2.0,2.0,0.1 colliX PMT1 Fcup //as above but scans on colliX from -2.0 -> 2.0 in steps on 0.1
            reeper -s -w -2.0,2.0,0.1 colliX PMT1::gaus Fcup::gaus,-1.0,1.0 //as above but fits gaus to PMT1 and Fcup
                                                                              vs colliX (restricted range for the latter)
```

EXAMPLES 2

Note. These examples use root macros to do EPICS PV plotting. All use simulated PVs in a softIOC.
The macros are templates to show how to use the GrimReeper features. Copy and hack.
reeper /home/kl/Dropbox/epics/reeper/grBasic.C //simple scan plotting one PV against another
reeper /home/kl/Dropbox/epics/reeper/grScanFancy.C //2D scan, filling 2d hists and reading waveforms.

AUTHOR

Written by Ken Livingston

SEE ALSO

reeper is part of the (GRIM)REEPER package (GRIM REEPER Implements Macros to) Read EPICS Events and Plot Everything in ROOT) Full documentation at: <http:slowcontrols wiki>