THE CERN/SL XDATAVIEWER: AN INTERACTIVE GRAPHICAL TOOL FOR DATA VISUALIZATION AND EDITING

G. Morpurgo, CERN

Abstract

As a result of many years of successive refinements, the CERN/SL Xdataviewer tool has reached its final stage. This graphical tool was especially developed to plot mono- or bi-dimensional arrays of data, and to interact with them, in many different ways. Many pages of graphical information can be handled by the program, in a loose hierarchy of Views, Graphs and Plots objects. Data can be displayed, and interacted with, both in graphical and in text format. Sophisticated built-in Zoom and Data Editing capability is implemented, as well as a flexible Data Output generation facility. A complete C Callable Interface is provided, including a mechanism for feeding back the Application Program with the selections made by the User in the Data Display part. The tool has been written in C language, making use of the standard X Window libraries (Xlib,Xt,Motif). It can be run as a stand alone process, communicating with the Application Program via a shared memory, or it can be embedded in the Application Program itself.

1 INTRODUCTION

The Dataviewer was originally developed at CERN/SPS in the late 80's , during the preparation of the software for the new CERN accelerator, LEP. The intention was to provide the Application Programmers and the LEP Operators with an uniform way to implement and use graphical data representations. Many features specific to the LEP context (ex. function editing) were introduced from the first design stage. The implementation was APOLLO specific, as those were the Workstations used at LEP. X-Window did not exist yet. The tool became rapidly popular, and many Applications were built on top of it. Many slightly different variants were developed by different people, until 1990, when Ann Sweeney cleaned and enhanced the code, merging the best features of this "parallel development", and froze the product[1]. Around 1994 the APOLLO workstations were replaced by HP-UX and Xterminals, running Unix and X-Window. Suddendly the Dataviewer became unusable. As your author had many Applications built on top of it, he decided to port the Dataviewer to the new environment. More and more Applications used it, and many new features were added to the tool, as a consequence of requirements of new Application. This was done always mantaining the backwards compatibility, so that already existing Applications could always use the latest versions without any modification to the source code[2]. One of these new features is the "Embedded Dataviewer". The Dataviewer was originally conceived as a Stand Alone Process, displaying data that an Application Program had put into a shared memory area. We found that in many cases is useful to merge

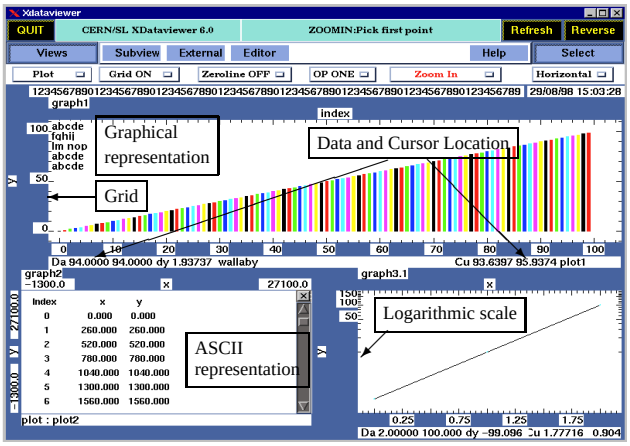

Figure 1: A typical Dataviewer page. A View containing three Graphs is shown

the Application Program with the Dataviewer, and therefore we have implemented a way to embed the Dataviewer inside the Application Program itself. This is done with the auxil of a very few function calls.

2 PHILOSOPHY AND REQUIREMENTS

The original design and implementation of the Dataviewer dates from the early days of workstations. At that time, the few commercial products on the market were primitive and expansive, so that there weren't many alternatives but developing this tool. Nowadays the scenario is quite different; many commercial packages are available, they are powerful and claimed to be grosso modo complete, and probably affordable. Nevertheless, they lack at least three of the most important requirements for such a tool in our environment: specific adaptation to our needs, immediate support, and possibility of developing new features as soon as the users need them. This is why if you have a good piece of home-made software, fulfilling your requirements and supported well, you should think twice before dropping it in favour of software on which you do not have any control.
Let us now have a look to the list of specific requirements satisfied by the Dataviewer.

- Ability of showing graphical pages ("Views"), structured in one or more parts ("Graphs"), each of which can display one or mode data arrays ("Plots").
- Possibility of online selection between different Views.
- Each Graph can be displayed in graphical or text format, independently from the other Graphs in the same View. The switch between graphical and text mode is available online.

- Online capability of selective displaying of View subsets.
- Possibility of editing a Plot online (modifying the correspondent data array).
- Possibility of zooming on the data.
- Active cursor, indicating online its position, and the position of the closest data point.
- Dynamic online reconfigurability of Views, Graphs and Plots.
- Access to the data via pointers.
- Optional drawing of axes and grids on the Graphs.
- Optional automatic rescaling of the Graph coordinates, driven by the data to be displayed.
- Possibility of assigning different colours and markers to each data point.
- Interaction with the Application Program, to know when to refresh the data and to inform it when a data point has been selected.
- Possibility of saving the screen data onto a file or to a printer.
- Callable Interface written in C, to enable the Application Program to configure the graphical information and to select the wished features.

3 IMPLEMENTATION ISSUES

3.1 Object definition

Four basic entities constitute the hierarchy on which the information visible through the Dataviewer are built :

- *The View.* A View is a graphical page, displayable as a whole on the Dataviewer Window. It can contain many Graphs.
- *The Graph.* A Graph is a window on the x,y space, in which one or more Plots are shown. To be visible, a Graph must be attached to at least a View.
- *The Plot.* A sequence of x,y points. the y values always come from a Data Objects. The x values come either from a Data Object (2-dim Plot), or from the index of the sequence (1-dim Plot). A Plot needs to be attached to at least a Graph.
  Auxiliary Objects can be used to define colours, labels, etc. for each point of the Plot.
- *The Object*. An Object is an array of numeric values, provided by the Application Program. An Object can contain data to be plotted by the Dataviewer, color codes, marker types, error bar values, or labels (in this case it will be an array of character strings). Objects are contained in MOPS structures (see next paragraph), referred to by names and accessed by pointers.

3.2 Object properties

We briefly list here the most important properties for Graphs and Plots

- *A Graph* can be displayed in graphical ot text format.
- Its x,y limits might be fixed or automatically adapted to the data.
- It may use linear or logarithmic scales.
- It may have timestamps as x coordinates.
- It may have a Grid and x,y axes.
- It may display also a few 80-char labels.
- A Graph can be attached to any number of Views.

- *A Plot* may be 1-dim or 2-dim.
- It may be drawn as an histogram, or as a sequence of markers, optionally connected by a line.
- Every point may have a different colour and label, and also a different marker type (32 colours and about 20 marker types are defined).
- A 2-dim Plot may have its x coordinates increasing monotonously, or being randomly distributed.
- Every point may have error bars.
- Also for a Plot listed in text format, every point can have a different colour.
- Output formats for x and y values can also be specified.
- It is also possible to ask the Dataviewer to show only a selected part of a Plot.
- A Plot can be attached to any number of Graphs.

All of these properties can be controlled by the Application Program via the C Callable interface to the Dataviewer. Default settings for many of these properties are also predefined. Again via functions in the Callable Interface, these defaults can be overridden at creation time both for single or for multiple Objects.

3.3 Data format

All the data required by the Dataviewer (plottable Objects, Views, Graphs and Plots definitions) must be contained into a MOPS[3]. The MOPS (Multiple Object Partitioned Structure) is a reserved memory area (Unix shared memory or internal process memory) dynamically configurable in User defined ”Objects”. The access to the MOPS and to the Objects is done by name, through a library of C (or FORTRAN) functions. In particular, some functions return pointers to the beginning of each Object Data Area. Data access can be protected by semaphores, if needed. An Object is an array of elements of the same type (int, float, structures, strings,...), and different Objects may have different types. The User can dynamically create or delete objects, change their number of elements, and retrieve or update their values. The data arrays displayed by the Dataviewer are MOPS Objects, and the MOPS access by name capability is used by the Dataviewer Callable Interface. The MOPS Library provides a easy and flexible way for structuring the data and for sharing it between different processes. For instance, there is no need to tell the Dataviewer the number of elements in a plot; it will find it by themself when looking at the MOPS. This solution mirrors very well the structure of many Applications, where the Application Program provides new data and leaves to the Dataviewer the task of showing them to the User.

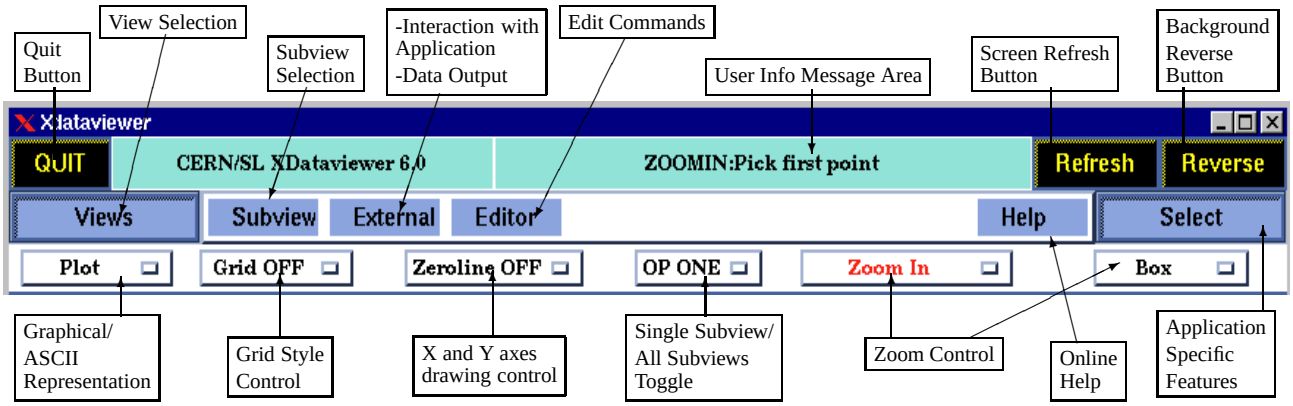

Figure 2: The Dataviewer Control Panel.

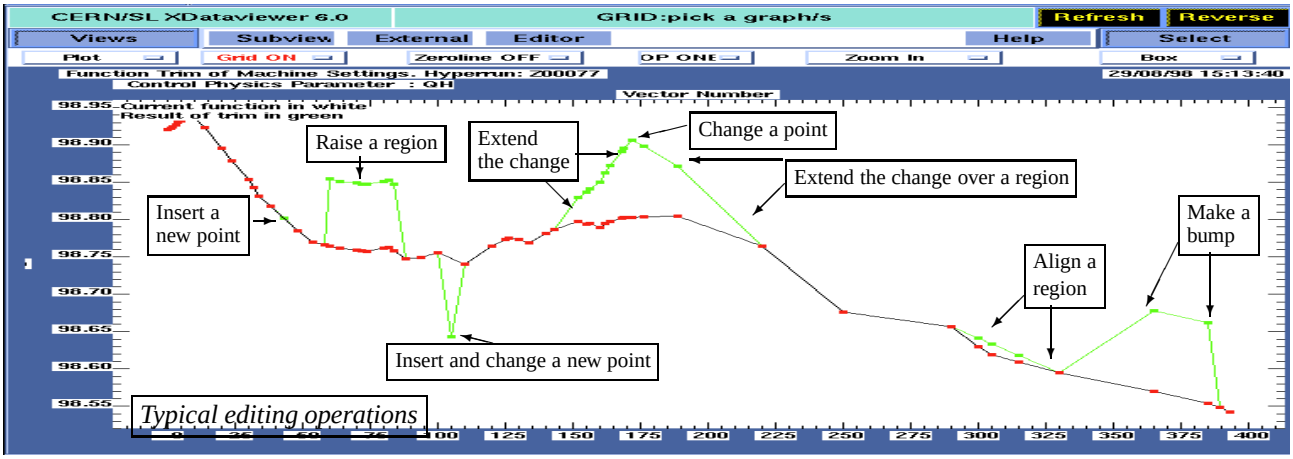

Figure 3: Edit facilities in graphic mode.

3.4 *The X Window Implementation of the Dataviewer Human Interface and Data Display parts*

The Control Panel of the Dataviewer (see Fig. 2) is written using the Motif toolkit, on top of X Window. The actual code is generated by a home made C code generator for X Window applications[4]. The code needed to display the data heavily relies on the Xlib package. As no machine-dependent code is used, the product should be easy to port onto other platforms.

3.5 *Graphic and Text Data Display*

Each Graph can be displayed in two different modes : the graphic mode (default), where each Plot in the Graph is drawn on the Dataviewer part allocated to that Graph in the way specified by the Application Program (Markers, Histograms, Lines..), and the text mode, where the values of the points of one or all of the plots contained by the Graph are written in columns using a default format or a specified one. In text mode, a scrollbar is also displayed, to help the User when moving through a long Plot. If a Graph contains more than one Plot, and the X coordinates of the different Plots are not homogeneous, all these coordinates are merged so that every Plot will find an entry for each of its points.

3.6 *Editing Facility*

The need for being able to edit a plot derives from the usage of the Dataviewer as a Function Editor, when defining and modifying the behavior of the different LEP equipment during the energy ramp. Both in graphical (via the cursor) and in text (via an input shell) modes it is possible to add, delete, move single points, to align a range of points, to interpolate modifications through a range of points (see Fig. 3).

3.7 *The Zoom Facility*

Using the cursor and the mouse, the User can zoom in and out the data. The axes (X and Y, only X, only Y) to be used for the zoom operation are controlled either by the Interface or by the Application Program. The Dataviewer maintains a linked list of zoom operations, so that it is possible to reverse these operations or to come back to the original configuration.

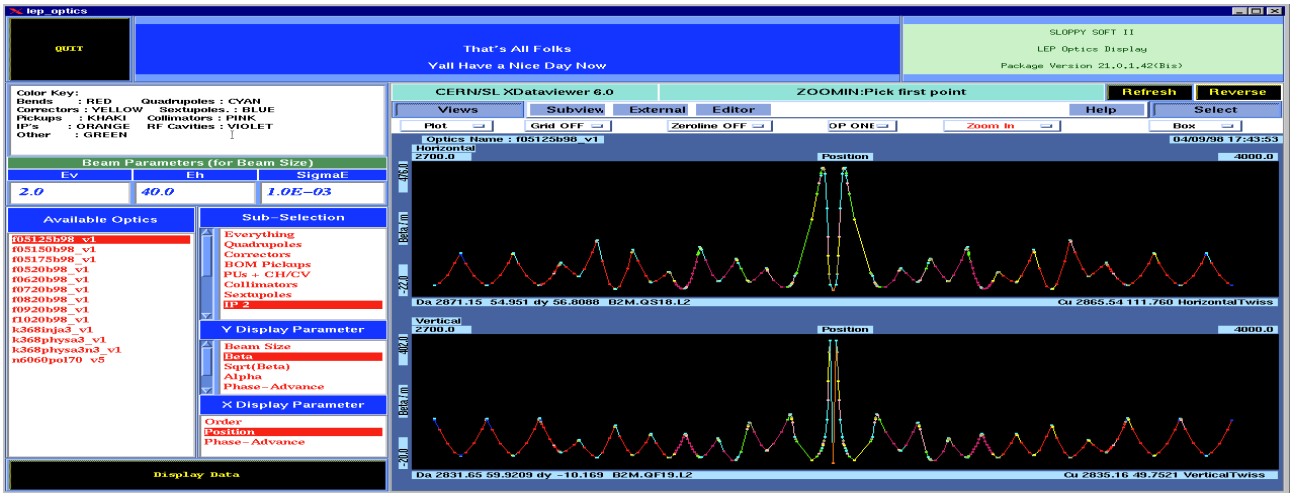

Figure 4: An Application using the Embedded Dataviewer.

3.8 The Grid Facility

If specified, a self adaptive Grid will be plot around or across a graph. The spacing of the Grid tics will depend on the graph limits and on the number of screen pixels available for the graph. The format will also be automatically adapted to the data, case by case.

3.9 The Help Facility

The Application Programmer can provide, in a text file containing special delimiter lines, explanations on the meaning of the different Plots, Graphs, Views. This information will be displayed on request with the help of a popup window.

3.10 Output Facility

The Dataviewer provides utilities to plot the contents of the screen on a printer (either colour or black and white). It may also produce .ps or .eps files containing the same information. Finally, it may print or save onto file the values of all the points in the different Plots in a selected Graph.

3.11 Communication with the Application Program

The two typical situations requiring communication between the Dataviewer and the Application are a) the production by the Application of new data to be displayed, and b) the need for the Application to be informed about some User selection on the Dataviewer data display part. In the former case the Application sends a Unix signal to the Dataviewer, which will refresh the data displayed. In the latter case the Dataviewer sends a signal to the Application, which, by means of a C function , may retrieve the relevant information. The Application may also decide to disable this mechanism.

3.12 The Embedded Dataviewer

Originally the Dataviewer was always running as a Stand Alone process. In some cases, however, it is desirable to integrate the graphical part within the same window as the rest of the Application. Due to some features of our code generator, this is actually straightforward. All the widgets composing the Dataviewer part are created within a single C function. A few other functions specific to the Embedded Dataviewer (command line arguments equivalent, or communication between the Application part and the Dataviewer part) have been made available to the Programmers.

3.13 The Callable Interface : (Keep It Simple, Stupid !)

The Golden Rule for insuring success to a tool is its easiness of use. Application Programmers like to concentrate on how to solve their problems, and not on how a tool works. Although about 100 C functions are available to tailor everything to the user's need, less than 10 are enough to start :

- dv_init : to allocate space for the Dataviewer data structures.
- dv_vwcreate : to create a view.
- dv_grcreate : to create a graph.
- dv_grattach : to attach a graph to a view.
- dv_plcreate1(2) : to create a mono(bi)dimensional plot,to display data objects.
- dv_plattach : to attach a plot to a graph.
- dv_kick : to inform the Dataviewer that new data are ready.

To these functions, one should add less then 10 C functions from the MOPS library (to create, initialize and access a MOPS, and to create and acces objects inside it)
These 15 functions are enough to start, and to produce perfectly well working Applications; most of the others are used to modify the properties of individual objects (Views, Graph or Plots). Some functions modify the default properties to be assigned to a newly created object, and other will modify the way the Dataviewer works.

4 CONCLUSIONS

The Dataviewer has reached the stage of maturity, and the rate of new user requirements is now very low. It is used by most of the Application Programs in the LEP and SPS Control rooms, and it has contributed in saving many man-years of software efforts. It also makes life easier for the Operators, by presenting them different graphical information in an uniform and coherent style.

5 ACKNOWLEDGEMENTS

So many persons have provided input to the original Dataviewer design and to all the successive refinements, that is impossible to mention here all of them. Thanks to them, the tool has become very versatile. Ann Sweeney deserves, above everyone else, the most merit for having synthetized all the different ideas and for having implemented them in a clean and well structured piece of code, easy to port, to modify, to extend.

6 REFERENCES

[1] The Dataviewer Programmer's Guide, Ann Sweeney, CERN/SL/CO Note/90-13

[2] New features for the HP_UX Dataviewer. Unpublished internal note available from the author (giulio.morpurgo@cern.ch).

[3] M.O.P.S. User Guide for "C" programs, Werner Herr, CERN-SPS/88-43 (AMS) Revised January 1993.

[4] Xcreator: a C code generator for AppliXation Programs. Unpublished internal note available from the author (giulio.morpurgo@cern.ch).

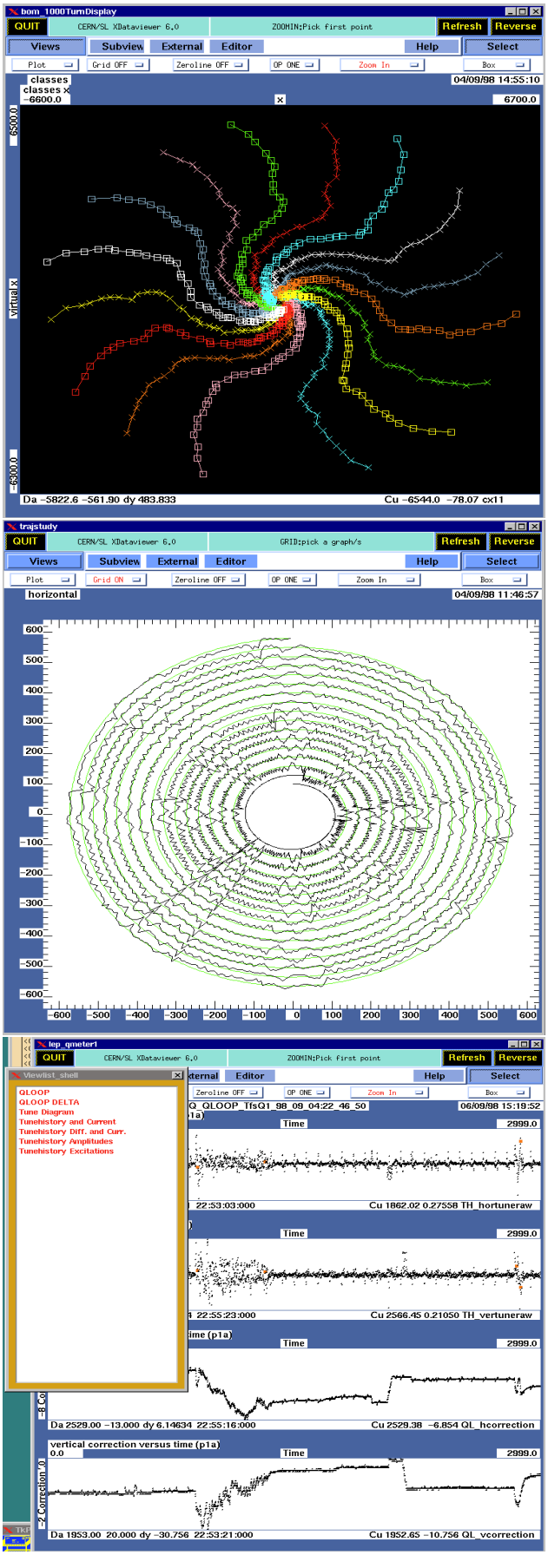

Figure 5: Some Dataviewer applications at LEP