ViSta

The Visual Statistics System

Enhancements to XLisp-Stat for Windows

Fabian Camacho & Forrest W. Young

THE L.L. THURSTONE
PSYCHOMETRIC LABORATORY
UNIVERSITY OF NORTH CAROLINA
CB 3270 DAVIE HALL, CHAPEL HILL N.C., USA 27599-3270

VISUAL STATISTICS PROJECT
WWW.VISUALSTATS.ORG
REPORT NUMBER 2000-1
JANUARY, 2000

Enhancements to XLispStat for Windows

Fabian Camacho & Forrest W. Young

Abstract

This report documents the enhancements made by the Visual Statistics project to XlispStat 3.52.13 for Windows. The major enhancements provide

1. a DDE connection between XLispStat and Microsoft Excel, and between XLispStat and Microsoft Access, giving XLispStat and Access users easy access to XLispStat's dynamic graphics; and
2. windows which can contain other windows, permitting multiple dynamic graphics within a single window. Since the contained windows can have invisible borders, titlebars, etc., the appearance to the user is of a single window containing multiple dynamic graphics. These container windows are the basis for the simplified user interface in ViSta 6.

The enhancements were made by Fabian Camacho under the direction of Forrest W. Young. Great appreciation is expressed to Bibb Latane for the financial support of a project to develop these enhancements.

1 DDE connection with Microsoft Excel and Access

1.1 Excel/ViSta

The Excel/Vista additions allow transfer between Microsoft Excel and Vista via a DDE connection.

There are five components to the system: InstallExcelMenus.exe, VistaComp.dll (which should be located in the window/system directory), ExcelVista.xla (a VBA library), SendtoVista.xls, and ExcelMenus.txt. The latter file consists of the menu names and their associated scripts (these should correspond to the files containing scripts located in the script folder) which will execute when its corresponding menu is selected by the Excel user. All files must be located in the startup subdirectory of the Vista folder, except for VistaComp.dll.

To install the ViSta Menu on the Excel principal menu bar, two methods are available. If the user wishes the ViSta Menu to appear automatically, then the InstallExcelMenu.exe executable should be run and the Install Menus option should be selected. This procedure modifies the registry so that the ViSta menu appears upon startup. The other method consists in simply

opening the SendtoVista worksheet. The ViSta menu is automatically installed when the worksheet is loaded and uninstalled when it is closed. If a ViSta menu is already available no other menus are installed.

To send data to Vista, the user should select a region in the Excel spreadsheet. Once a desired region is selected then the user selects a particular choice in the ViSta Menu. The program then starts Vista if an instance is not already available and waits for 30 seconds or until the VistaOp key in the wxls32.ini file is set to yes. The data selected in Excel should then be observable in the ViSta Desktop and the script indicated in ExcelMenus.txt should be run.

1.2 Access/ViSta

The AccessVista additions work in a similar way to ExcelVista. The corresponding files are: InstallAccessModule.exe, VistaComp.dll (which is also needed for ExcelVista), Access-Vista.mda, and AccessMenus.txt. Selecting the Install Access option which appears when executing InstallAccessModule installs the Access Add-in necessary for transferring data. The user can then start this add-in by selecting the Transfer to Vista option in the Add-Ins menu. A form appears in which the user may select a table of the current database, change the variable types of the table, and choose a desired script before selecting the button which sends all the information to Vista.

2 Containers

Container windows are graph-windows which can contain other graph-windows. The contained windows can have invisible borders and titles, making it possible to construct a window that shows the viewer what appears to be a single window containing several graphics. These are used heavily in WinViSta 6.0. An example of ViSta's desktop container window and of a spreadplot container window are shown in Figures 1 and 2.

2.1 Constructor Function

Container windows are instances of `container-proto`, the new container window prototype. This prototype inherits from the graph-window-proto prototype with the specific additional capability that windows pertaining to other objects derived from window-proto prototype can be inserted into its client area. The constructor function which constructs container-windows is:

```
(defun container
   (&key (in nil set) (style 5) (toolwindow nil) (localmenu nil)
  (enabled t) (size '(250 250)) (location '(54 24))
         (title "Container Window") (show t) (go-away t)
         (menu nil) (black-on-white t)
         (has-v-scroll nil) (has-h-scroll nil)
         )
```

The constructor function creates a new instance of a container window and returns the container object. The new container window can be inside the main XLisp client window, inside another container window, or outside of all other windows. Where it goes depends on the way the `:in` argument is used. Specifically:

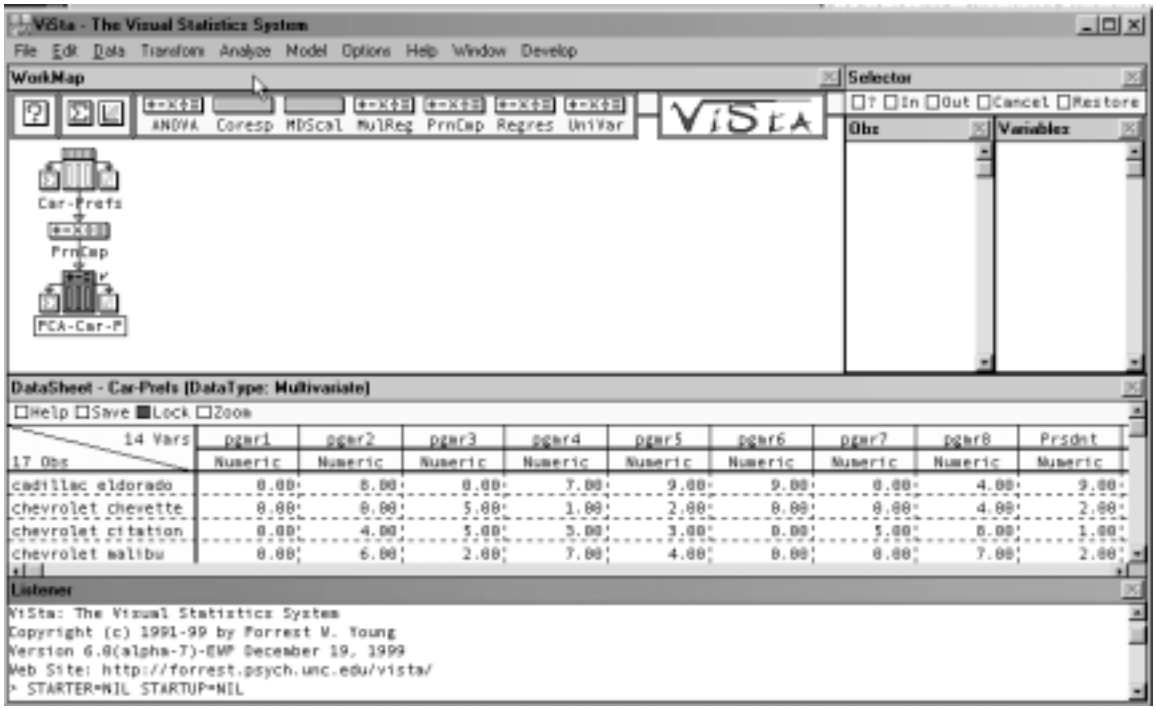

Figure 1: ViSta's Desktop involves 6 container windows and 5 standard windows. The main container has style 7. It's windows have title bars without controls, and medium borders.

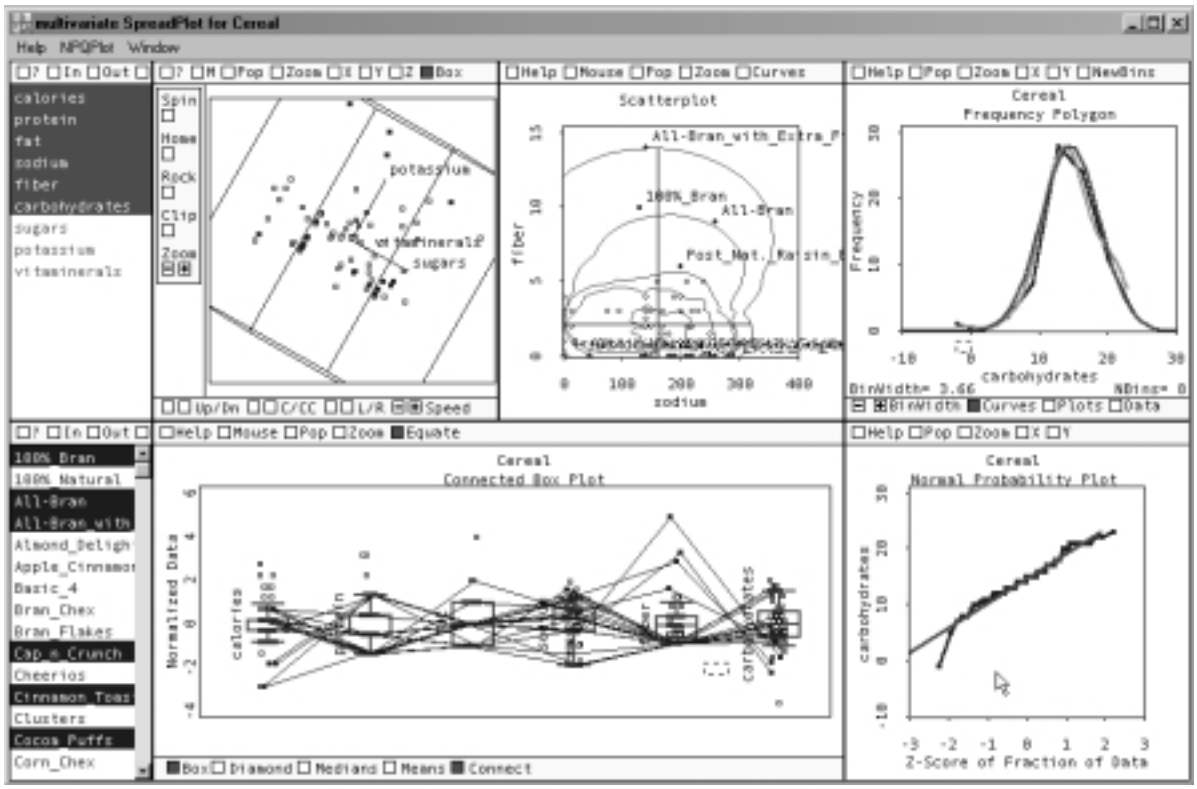

Figure 2: ViSta's Multivariate SpreadPlot is a single container window with 7 graphics windows. This container has style 1. The windows lack title bars and controls and they have very thin borders.

`(container)`
creates a new container in the Xlisp main client window.

`(container :in container-object)`
creates a new container inside `container-object`. An error is signaled if `container-object` does not exist or is not a container object.

`(container :in nil)`
creates a new container window on the desktop (that is, inside no other window). This kind of container is called a desktop window.

`(container :in t)`
creates a new container that is inside the currently enabled container, if there is one, or inside the clicent window, if no container is enabled.

Container windows can be enabled to receive any successively created graph windows (or windows with graph-window-proto as an ancestor) with the :enabled message. By default, a newly created container window is enabled. Only one container at a time can be enabled to receive windows. Thus, by default, a newly created container disables a previously enabled container. The function `(enable-container container-object)` can be used to change which container is enabled. The function `(disable-container)` can be used to disable containers. In this case, new graph windows go into the main client window, the initial state. The global variable `*active-container*` is bound to the enabled container window. Note that all new windows go into the enabled container object, or into the main client window if no container is enabled. Windows inside a container window are said to be child windows of the container window. The relationship between a child window and its container cannot be changed.

The argument `[style number]` is an integer varying from 0 to 8 which determines the style of the child windows: 0: no border; 1: thin border; 2: sunken border; 3,4,6: thick border; 5: standard window (movable, sizable window with title and all controls); 7: movable window with title; 8: title and border like standard window. The default is 5, standard windows. By default, a desktop window has the standard application window style. To change this style to the style of an tool window, specify `:toolwindow t`, which creates a desktop window without minimize, restore, and maximize buttons. Menus for windows inside a container will be placed in the container window's menubar if `:localmenu` is true, provided the container is a desktop window. If `:localmenu` is false menus will placed in the main xlisp window menu bar.

Although the features corresponding to graph-window-proto work as should be expected for the container, as of this date (December 1999) several conflicts have arisen between key and mouse events and buffering animations whenever child windows are present. To this end, it is recommended that events and animations be primarily handled in the child windows.

2.2 :new message

The syntax for the :new message which creates a container object is:

```
(send container-proto :new [style number]
   :outofclientwindow [nil t]
   :localmenu [nil t]
   :mainwindow [nil t]
   :putincontainer [nil t] )
```

All additional graph-window-proto arguments inherited from the graph window :new method should work. We have used :show, :title, :location and :size .

The container will be put inside the main xlisp window unless :outofclientwindow is true, in which case it becomes a desktop window with the tool window style. To change this style to an application window's, specify :mainwindow t, which creates a window with minimize, restore, and maximize buttons. The :putincontainer key allows a container to be placed inside the current container if true, otherwise a separate container is created and all successive objects will be placed inside it. The constructor function's :in argument effects the :outofclientwindow, :putincontainer and :mainwindow arguments, and uses the :getobjects method.

Note that the arguments in the constructor and the :new method do not map onto each other in a simple one-on-one fashion. This reflects our learning how to deal with container windows, whose presence can change program structure quite markedly. The constructor's arguments are designed to be as easy to use as possible, whereas the :new method reflects our thinking as we developed this project. We would recommend that in a future implementation, the :new method be redesigned to correspond with the construtor's argument structure.

2.3 Other messages and functions

Objects can be redirected to other containers or the main window by `:getobjects [t nil]`. So if object b is a container, and the message (send b :getobjects nil) is sent, successively created objects will appear in the main xlisp window instead of within the container. The enabled container is the one for which :getobjects is t.

The xlisp-stat functions which allow for copying and printing will execute only for the current active child window of a container or not at all. However, pressing ALT-C will copy the entire container's client area to the clipboard.

Window objects placed inside a container can become desktop windows by the method `:pop-out [t nil]`. So if object c is a histogram inside a container, issuing `(send c :pop-out t)` will make object c pop out into the desktop with a tool window style together with any menus corresponding to it. The message `(send c :pop-out nil)` places the object back to its original size and position inside the container.

The `:no-move [t nil]` method immobilizes an object window so that it cannot be resized or moved by user action. Resize, locate and size messages still have their normal effect.

3 listener-proto

A new object prototype: listener-proto, has been added. An instance of this prototype will redirect the standard input and output to its corresponding listener window whenever this window is activated. To create a new instance type (send listener-proto :new).

4 Mainwindow and Listener Window Functions

The following new functions are provided for the main window and the listener.

`(showmainwindow)`
: insures that the main xlisp application window is visible. If the RecallMainFrame key in the wxls32.ini file is "yes" then the window is resized to its former window placement size. If RecallMainFrame is "no" then the window is maximized.

`(hidemainwindow)`
: makes the main xlisp application window invisible.

`(listener x y w h)`
: is a function which changes the size and location of the listener according to coordinates x y and dimensions w h. The coordinates correspond to the upper left corner of the listener's white area relative to the main window's upper left corner's client area. The sizes correspond to the dimensions of the listener window including the caption and border edges. `(listener)` will return the coordinates of the listener when in the normal state.

`(mainwindow x y w h)`
: works exactly like (listener x y w h) except that the location coordinates are relative to the desktop and the menu bar is included as part of the client area.

`(client-size)`
: returns the dimension of the client area of the main window.

`(effective-screen-size)`
: returns the dimensions of the whole screen excluding the task bar.

5 Other Changes:

1. The wxls32.ini file is now searched locally. The search order is as follows: 1st, the local directory where the executable is located is searched. 2nd,the current-working directory is searched. 3rd, the windows directory is searched. 4th: the windows/system directory is searched. If the ini file is not found in any of these directories an error is output.
2. Dialogs have been modified to conform to windows 95 GUI standards. They have also been made topmost windows, so that they will be above all non-topmost windows.
3. A bug in the win32 xlispstat version which rendered pop-up menus inoperative was fixed.
4. Two command line flags have been added: -showmainwindow makes the main window visible on startup. –verbose sets the *****xlispverbose***** variable to true.
5. The name-list prototype has been modified to include margins by sending its instance the :margin method. The scrolling speed for large lists has also been reduced significantly by only drawing the rows which are in view.
6. window-proto objects have been provided with two new methods, `:top-most [t nil]` and `:bottom-most [t nil]`. If a window pertaining to an object has been made into a desktop window by :pop-out t, the :top-most method makes this window remain on top of other windows unless they themselves are top-most and the :bottom-most method makes it be placed first in the window order so that the other windows will be on top of it unless it is activated.