GUI Tools

Graphic User Interface for GAUSS

ECONOTRON SOFTWARE, INC.

Version 5.0

Jon Breslaw

April, 2005

Support:

Econotron Software
447 Grosvenor Ave.
Westmount, P.Q. Canada
H3Y-2S5

Tel: (514) 939-3092
Fax: (514) 938-4994
Eml: support@econotron.com
Web: http://www.econotron.com

Contents

Chapter 1

Concept

The underlying idea behind the development of GUI Tools was the concept of providing an interactive graphic user interfaces for GAUSS for Windows. GAUSS provides two intrinsic functions for interactive input - cons and con. These are used for the interactive input of text and matrices respectively. However, these two functions are text based, and are thus inadequate in a Windows environment. What is missing is the ability to have an end user respond to a graphic based dialog, with mouse control, and where the required inputs ar clearly shown, along with prompts and default values.

GUI Tools provides exactly this functionality. It provides a range of standard, pre-programmed interfaces for common tasks, including both controls and dialogs. It also allows the programmer to create his/her own interface, using a graphic builder technique - the same technique that is used to build dialogs in Visual Basic and Visual C. The GUI is created using a form and a toolbox of controls. The programmer clicks the required control to copy that control to the form, and then, using the mouse, drags the control to the desired location and sizes it. A list of properties is provided for each control, which the programmer sets as desired. Each control is given a name, and the characteristic of that control is saved in GAUSS under that name. Thus a text box called text1 containing the phrase "Hello World" will be available in GAUSS as if the user had typed in GAUSS:

```
text1 = "Hello World";
```

A fair degree of extensibility has been built into GUI Tools. At the simplest level, GUI Tools allows for interactive input using standard controls or dialogs. At the

more advanced level, custom interfaces can call specified procedures, and, after completion of a procedure, control is returned back to the GUI.

The remainder of this manual describes how to install GUI Tools and run the examples. A number of demonstration files are included, and are discussed in depth as a way of introduction to the GUI Tools functions. A detailed description of each function is given in the reference section.

Chapter 2

Installation and Testing

This chapter describes the hardware and software configuration required to run GUI Tools on your computer, as well as instructions on using the GUI Tools commands.

2.1 Installation Requirements

The GUI Tools for GAUSS system requirements are:

- Windows 9x, NT4, ME, 2000, or XP.
- GAUSS 4.0 or higher

2.2 Installing GUI Tools

Before you open the product package, please read the license agreement that accompanies GUI Tools. By installing and using the product, you accept the terms of this agreement.

The program files on the CD are compressed, so you cannot simply copy them to your computer. Rather, you must run the installation program which decompresses the files and copies them to your hard disk in the appropriate directories.

1. Insert the GUI Tools CD into the appropriate drive.
2. From the Windows Start menu, chose Run.
3. Type d:\setup.exe (where d: is the letter for your CD drive).
4. Choose OK.
5. Follow the instructions on the screen.

The installation routines creates a folder called gauss\guitools which has the following structure:

`Doc`	This folder contains the GUI Tools help file and the GUI Tools manual in pdf format.
`Examples`	This folder contains example files for controls, dialogs and user defined GUIs .
`Gui`	This folder contains the GUI description files.

2.3 Testing GUI Tools

From the GAUSS prompt, type

```
run guitest.e;
```

This should display a welcome message in a message box.

2.4 GUI Tools Demo

From the GAUSS prompt, type

```
run guidemo.e;
```

This will consecutively execute the 18 files in gauss\guitools\examples. The example files - gui1.e to gui18.e can serve as templates for your own code.

Chapter 3

Program Examples

This section shows how one programs the GUI Tools interface. This is accomplished by discussing three of the example files.

3.1 Structure

GUI Tools uses a structure to handle the various options. This structure is defined as follows:

```
struct gstruct {
          string    command;          ("")
          string    text;             ("")
          string    title;            ("")
          string    prompt;           ("")
          scalar    style;            (0)
          scalar    splash;           (1)
          string    mode;             ("run")
          string    gaussState;       ("hidden")
          string    guiState;         ("hidden")
          string    procName;         ("")
          string    fileName;         ("")
    string array    list;             ("")
              };

                    (defaults in parenthesis)
```

3.2 Example 1 - A TextBox Control

This is an example that uses the built in controls and dialogs. In this example (gui3.e), the text entered by the user is returned.

```
1     library guitools;
2     #include guitools.sdf;
3     struct gstruct g0;
4     g0 = guiSet;
5
6     g0.command =  textbox;
7     g0.title   = "Text Box Example";
8     g0.prompt  = "Enter your name";
9     name =  guiRun(g0);
10
11    "Your name is: " name;
```

The first four lines are standard - the GUI Tools library is specified, the GUI Tools structure `gstruct` is loaded from guitools.sdf, and an instance of this structure g0 is created and initialized in `guiSet`.

The control type is specified in line 6, and the title and prompt in lines 7 and 8. The GUI is run in line 9, and the contents of the text box is returned and stored in `name`.

3.3 Example 2 - A Simple GUI

This example (gui13.e) demonstrates how a simple user defined interface is used.

```
1     library guitools;
2     #include guitools.sdf;
3     struct gstruct g0;
4     g0 = guiSet;
5
6     g0.gaussstate = normal;
7     g0.filename = guipath $+ "gt13.gui";
8     call guiRun(g0);
9
```

```
10    cls;
11    format /rd 2,0;
12    " Values returned";
13    " ";
14    "     Control Name  Control Value  ";
15    " ";
16    "      ListBox        ";; varget("listbox");
17    "";
18    "      Beer_c         ";; varget("beer_c");
19    "      Milk_c         ";; varget("milk_c");
20    "      Wine_c         ";; varget("wine_c");
21    "      Whisky_c       ";; varget("whisky_c");
22    "";
23    "      Beer_o         ";; varget("beer_o");
24    "      Milk_o         ";; varget("milk_o");
25    "      Wine_o         ";; varget("wine_o");
26    "      Whisky_o       ";; varget("whisky_o");
27    "";
28    "      Slider         ";; varget("slider");
29    "";
30    "      OK_b           ";; varget("ok_b");
31    "      Cancel_b       ";; varget("cancel_b");
32    "";
33    "      Comments:\n    ";  varget("comments");
```

This demonstrates a user defined interface. It was initailly created using the `guiNew` command, creating each of the controls on the form. The GUI was then saved as a GUI description file (gt31.gui). Each control on the form has a name, and this name is used to store the control specific property in GAUSS. Thus, for example, the first check box is called `beer_c`, and a GAUSS variable with the same name is created, and which is set to unity if the `beer_c` checkbox is checked, else zero.

Lines 1 to 4 are standard, as before. Line 6 specifies that the GAUSS window is to stay shown while the GUI is displayed - the default is `hidden`. Line 7 gives the filename of the GUI description file (gt13.gui), and the GUI is displayed in line 8. On return, lines 10 to 33 are executed - in each case, a characteristic property of each control is retrieved as a GAUSS variable with the name of the control, and displayed. Thus, checkboxes, option buttons, and command buttons return a zero or unity, depending whether the respective control is checked (or clicked), a slider returns its value, and a listbox and textbox return a string.

Full details for each control is provided in the reference for `guiRun`.

3.4 Example 3 - A programmed GUI

This example (gui16.e) demonstrates how to program a GUI which calls a proc.

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;
clear _sqp_Start, _sqp_Title, _sqp_FnProc;

g0.procname   = "sqp_prog";
g0.filename   =  guipath $+ "gt16.gui";
call guiRun(g0);

proc sqp_prog;
   local ok,txt, guivar;
   local x, f, lagr, ret;
   cls;
   ok = varget("OK_b");
   if ok;
       sqpSolveSet;
       guiPut("Bounds"  , "_sqp_Bounds");
       guiPut("Start"   , "_sqp_Start");
       guiPut("FnProc"  , "_sqp_FnProc")
       guiPut("EqProc"  , "_sqp_EqProc");
       guiPut("IneqProc", "_sqp_IneqProc");

   .
   .
   .

       print _sqp_title;
       { x,f,lagr,ret } = sqpSolve( _sqp_FnProc,_sqp_start );
       call guiWait;
   else;
     call guiEnd;
   endif;
   retp("");
endp;
```

This demonstrates a GUI front end for estimation - in this case we have used sqp, but the same concept can be used for ml or cml. As before, the GUI description file (gt16.gui) was created using guiNew or guiEdit. The GUI requests all the

inputs that are used by sqp; and these are stored in GAUSS using the name of the control.

Lines 1 to 4 are standard. Line 5 is needed for compilation. Line 7 specifies the proc that is to be called when the user clicks a command button in the GUI - in this case, `sqp_prog`. The GUI description file is specified in line 8, and the gui is displayed in line 9.

After the user has clicked one of the command buttons, `sqp_prog` is run. The first job is to see if the user clicked the OK button, which is called `OK_b`. If this button was clicked, then unity would be stored under its name. This is tested in line 16. `sqpSolveSet` is called at line 17, and then each of the values from the GUI are stored in the appropriate `sqp` global, using `guiPut`. For example, if the textbox called `EqProc` had a function specified - say `&eqp`, then line 21 will assign that text to the variable `_sqp_EqProc`.

Optimization takes place in line 32, and a keyboard input is requested from the user in line 33. After a key has been entered, the proc is completed, and control is returned to the GUI, so that one can do another run.

If the OK button had not been clicked, `guiEnd` at line 35 would have been executed. This command terminates the GUI, and returns control to the GAUSS prompt.

Chapter 4

GUI Tools Command Summary

4.1 Summary

The commands are arranged alphabetically. For easy reference, a summary of commands arranged by type is given below.

4.1.1 GUI execution

guiEdit — Modifies an existing GUI.
guiNew — Creates a new GUI.
guiRun — Executes a control, dialog or GUI.

4.1.2 Programming

guiClear — Clears an active GUI.
guiEnd — Terminates a GUI.
guiEval — Executes a string.
guiHelp — Displays the GUI Tools help file.
guiPut — Assigns control value to a symbol.
guiSet — Initializes a GUI structure.
guiWait — Wait for keyboard input.

Chapter 5

GUI Tools Reference

- **Purpose**

  Clears an active GUI.

- **Format**

  guiClear;

- **Remarks**

  When a GUI has a proc specified in `g0.procname`, that proc is called when a command button on the GUI is clicked by the user. At this stage, the GUI is not closed (though it is usually hidden), since after the proc has been completed, control is returned to the GUI.

  However, if there is a GAUSS compile or execution error, the user is instead returned to the GAUSS prompt with an error message, while the GUI remains hidden. Before any further work is done in GAUSS, However, the GUI should be terminated. This can be done by executing the `guiClear` command.

  A GUI can also be terminated by ending the gui.exe process from the Windows task manager.

- **Example**

```
guiClear;
```

Purpose

Creates or modifies a user defined GUI .

Format

guiEdit (*g0*);

Inputs

g0 GUI structure.

Remarks

This command is used to create or modify a user specified GUI. An existing GUI is specified in `g0.filename`. A new GUI is created by setting `g0.filename = ""`, or by using the `guiNew` command.

This command opens controls toolbox, and either a blank design form (for a new GUI) or an existing design form. Controls are added by clicking the appropriate control in the toolbox. The control can then be sized and/or moved using the mouse, or by setting the `top, left, height` or `width` properties in the property window. Other properties are set by clicking the respective property name in the property window. Each control returns a property to GAUSS, which is stored under the `name` of the control. For example, setting the name of a textbox to *algmth*, and entering the string `"Hello World"` into the `text` property would result in GAUSS storing `"Hello World"` in the global variable *algmth* when the GUI is executed.

The GUI tools toolbox contains the following controls

Control	Content returned to GAUSS
Button	value, 1 if clicked, else 0.
Check Box	value, 1 if checked, else 0.
Directory List	string, selected folder.
Drive List	string, selected drive.
File Browser	string, selected file.
Frame	none.
Label	none.
List Box	string, selected item.
Option Button	value, 1 if checked, else 0.
Slider	value, slider value.
Text Box	string, text content.

A GUI is saved as an ASCII file with a .gui extension.

See the `guiRun` command for details on the `g0` structure. An example is given in gui18.e.

- **Example**

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;

g0.filename = guipath $+ "gt13.gui";
guiEdit(g0);
```

In this example, an existing file (`gt13.gui`) is opened for modification.

- **See also**

`guiNew, guiRun`

- **Purpose**

  Terminates a GUI.

- **Format**

  guiEnd;

- **Remarks**

  After a proc, that has been called by a GUI, has exited, control is returned back to the GUI.Clearly, it is necessary to have a method of terminating this process - when, for example, the user has clicked a "Cancel" button. The `guiEnd` command terminates this process.

- **Example**

```
g0.procname   = "marshal";
g0.filename   = guipath $+ "gt15.gui";
call guiRun(g0);

proc marshal;
    local ok;
      ok = varget("OK_b");
    if ok;
      txt = varget("Gauss_Text");
      call guiEval(txt);
      call guiWait;
    else;
      call guiEnd;
    endif;
    retp("");
endp;
```

  In this example, the GUI calls the proc `marshal` when a command button is clicked. The proc checks to see if the `OK_b` button was clicked - in which case it will have a value of unity. In this case, the proc carries out the process of evaluating the text in the text box called `Gauss_Text`, and then control returns to the GUI on return from the proc. If the `OK_b` button was not clicked, then the `guiEnd` will terminate the GUI, and return control to the GAUSS prompt.

- **See also**

  `guiClear`

- **Purpose**

  Executes a string consisting of a set of GAUSS expressions.

- **Format**

  guiEval (*str*);

- **Inputs**

  str string, GAUSS expression.

- **Remarks**

  The `guiEval` command evaluates the string *str* as if the contents of *str* had been typed in at the GAUSS prompt. This permits GAUSS to evaluate text from a `textbox` control.

- **Example**

```
txt = varget("Gauss_Text");
call guiEval(txt);
call guiWait;
```

  This example shows how GAUSS statements that had been entered in the textbox `Gauss_Text` can be evaluated.

- **See also**

  `guiPut`

- **Purpose**

  Displays the GUI Tools help file.

- **Format**

  guiHelp;

- **Remarks**

  This command is used to display the GUI Tools help file from GAUSS.

- **Example**

```
library guitools;
guiHelp;
```

- **Purpose**

  Creates a new user defined GUI .

- **Format**

  guiNew (*g0*);

- **Inputs**

  g0 GUI structure.

- **Remarks**

  This command is used to create a new user specified GUI.

  See the `guiEdit` command for details on the use of the controls toolbox. An example is given in gui17.e.

- **Example**

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;

guiNew (g0);
```

- **See also**

  `guiEdit`

Purpose

Assigns the contents of a GUI control to a GAUSS global symbol.

Format

guiPut (*guistr, varname*)**;**

Inputs

guistr	string, name of a GUI control.
varname	string, name of a GAUSS global symbol.

Remarks

`guiPut` takes the contents of the control named in *guistr*, and assigns it to the GAUSS variable specified in *varname*. The contents of *guistr* can be a string - such as the content of a `textbox` - or a value. The only requirement is that the contents of the control evaluate to a valid GAUSS expression.

Example

```
guiPut("Bounds", "_sqp_Bounds");
```

In this example, the GUI contains a `textbox` named `Bounds`. Assume that this textbox contained the string { `2 3 4, 1 0 1`}. The `guiPut` command is equivalent to the GAUSS command:

```
_sqp_Bounds = { 2 3 4, 1 0 1};
```

See also

`guiEval`

- **Purpose**

  Executes a control or a user defined GUI .

- **Format**

  guiRun (*g0*);

- **Inputs**

  g0 GUI structure.

- **Outputs**

  rslt Return value or string.

- **Remarks**

  This command is used to execute a control, a dialog, or a user specified GUI. Controls and dialogs are prespecifed, and can be called directly, while user specified GUIs must first be created using `guiEdit` or `guiNew` before being executed. The input arguments of the `guiRun` command are specified within the structure *g0*. This structure has the following elements:

g0.command	string.
g0.filename	string.
g0.gaussstate	string.
g0.guistate	string.
g0.list	string array.
g0.mode	string.
g0.procname	string.
g0.prompt	string.
g0.splash	scalar.
g0.style	scalar.
g0.text	string.
g0.title	string.

  An instance of the GUI structure is specified at the beginning of the GAUSS command file:

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;
```

This code creates a structure `g0` of type `gstruct`, and initializes it in `guiSet`. Before calling `guiRun`, the elements of `g0` that are required should be specified.

g0.command This string specifies the type of control that is to be displayed. The following controls are supported:

`checkbox` A check box control. The list of entries is specified in `g0.list`. Returns a vector with unity elements for checked items, and zero for non-checked items.

`colordlg` A color selector dialog. Returns the numeric expression for the selected color.

`combobox` A combo box control. The list of entries is specified in `g0.list`. Returns the selected element as a string.

`filedlg` A file browser dialog. Returns a string containing the selected file.

`fontdlg` A font selector dialog. Returns a string array containing information on the selected font:

Row	Value.
1	Font name
2	Font color
3	Font size
4	Bold
5	Italic
6	Strikeout
7	Underline

`gaussbox` A predefined `GUI` for `GAUSS`.

`loginbox` A login control. The user name is specified in `g0.text`, and the password in `g0.prompt`. Returns the string `OK` if successful, and `Cancel` otherwise.

`msgbox` A message box control. The message is specified in `g0.text` or `g0.prompt`. The button style is specified in `g0.style` - see below. Returns a string containing the caption of the button that was clicked.

`optionbox` An option box (or radio box) control. The list of entries is specified in `g0.list`. Only one option can be selected. Returns the element number of the selected item.

`printdlg` A print dialog control. The text to be printed is specified in `g0.text`, or a filename in `g0.filename`. Returns unity if successful, else zero if canceled.

`textbox` A text control. Returns the string entered by the user.

g0.filename This string specifies the filename for user specified GUI description files. These files are created using `guiEdit`, and read using `guiRun`. A global, `guipath`, specifies the path of the gauss\guitools\gui folder.

g0.gaussstate When the GUI is displayed, the user can specify how GAUSS should be displayed. The options are `normal, hidden, minimize,` and `maximize`. The default is `hidden`.

g0.guistate When a procedure is specified in `g0.procname`, the GUI persists while GAUSS executes the proc. The user can specify how the GUI should be displayed. The options are `normal, hidden, minimize,` and `maximize`. The default is `hidden`.

g0.list This consists of a list of items specified in a string array, and is used by the `checkbox, combobox,` and `optionbox` controls.

g0.procname This string specifies the procname that will be called by the user specified GUI when a command button is clicked. This proc acts to control the flow of events - the events called from the proc will depend on the settings returned from the GUI. When the proc is completed, control is returned to the GUI, unless `guiEnd` is called. Thus, typically, if an OK button is clicked, the procedure executes the desired events, while if a `Cancel` button is clicked, the procedure executes `guiEnd` and returns to the GAUSS prompt. If *procname* is an empty string, no procedure will be called.

g0.prompt This consists of a string that is used in a number of controls.

g0.splash When a GUI with many controls is called, a splash screen is displayed while the GUI is loaded. The default is `g0.splash = on`.

g0.style This scalar specifies the button style for `msgbox` controls.

Value	Button Style.
0	OK
1	OK and Cancel
2	Abort, Retry, and Ignore
3	Yes, No, and Cancel
4	Yes and No
5	Retry and Cancel

g0.text This consists of a string that is used in a number of controls.

g0.title This consists of a string that is used as the caption for each control.

Examples of calls to controls are given in gui1.e - gui7.e, and calls to dialogs are in gui8.e - gui11.e. Examples of executing a user defined GUI are provided in gui13.e - gui15.e.

- **Example**

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;

g0.filename = guipath $+ "gt13.gui";
call guiRun(g0);
```

In this example, an existing file (`gt13.gui`) is displayed.

- **See also**

`guiEdit, guiNew`

- **Purpose**

  Initialize an instance of a GUI structure.

- **Format**

  g0 = **guiSet;**

- **Outputs**

  g0 a GUI structure.

- **Remarks**

  This command inializes the entries of a `gstruct` structure at the default values. This command should be called prior to issuing other GUI Tools commands.

- **Example**

```
library guitools;
#include guitools.sdf;
struct gstruct g0;
g0 = guiSet;
```

  This example shows the standard code for a GUI Tools command file. The #include statement provides the gstruct definition. `g0` is an instance of `gstruct`, and is initialized using the `guiSet` statement.

- **Purpose**

  Prompts for a key input.

- **Format**

```
guiWait;
```

- **Remarks**

  This command prompts the user for a keyboard input. It provides an escape facility, and so should be used (instead of `call keyw`) within a proc that is called by a user defined `GUI`.

- **Example**

```
 proc browser;
    local ok;
    ok = varget("ok_b");
    if ok;
       "     FileName       ";;  varget("Filename");
       call guiWait;
    else;
        call guiEnd;
    endif;
    retp("");
endp;
```

  This example shows the use of `guiWait` within a proc called by a `GUI` .

Chapter 6

Trouble Shooting

- `mercurysm not found.`

  The file `mercurysm.dll` should exist on the gauss\guitools folder. Optionally, it can also exist on the gauss\dlib folder.

- `Application hangs during GAUSS execution`

  This will usually occur under the following circumstances:

  1. GAUSS is waiting for a keyboard entry. Restore the GAUSS window, and enter the required key.
  2. GAUSS is looping. Close the GAUSS window. The GUI will still hang, and must be closed using the Windows task manager. The GUI application is called Gui for GAUSS, and the process is called gui.exe. After exiting GAUSS, check that the gauss.exe process has terminated.
  3. A GAUSS error has occurred - for example matrices are not conformable. Restore the GAUSS window, and from the GAUSS prompt type:
     guiClear; *enter*
  4. GAUSS can be closed while waiting for the GUI by typing F12.

- `Cannot size slider control.`

  This is a known bug. Use the properties `Height` and `Width` to size the control.

Index