AMTEC®

Tecplot® GUI Builder

User's Manual
Version 3.0
For Tecplot 10

Amtec Engineering, Inc.
Bellevue, Washington
December, 2003

Contents

CHAPTER 1 Tecplot GUI Builder

The Tecplot GUI Builder (TGB) is a tool for building a graphical user interface for a Tecplot add-on. It is not necessary to use TGB—you can use other commercial graphical layout tools. However, using TGB will allow you to quickly generate platform-independent user-interface code.

The remainder of this document presumes you have already run **`CreateNewAddOn`** under UNIX or Mac OS X, or the Tecplot Add-On Wizard under Windows, to get your add-on development started. (See the *Tecplot ADK User's Manual*, Chapters 2, for more details on using these utilities.)

1.1. New in Tecplot GUI Builder

`Tabtest`, a sample TGB Add-on, demonstrates how to use TGB's new features. Updated TGB options include:

- **Dynamic option menus:** Dynamically add or remove items for option menus with new application programming interfaces (APIs).
- **Form controls:** Sets of dialog controls which may be dynamically displayed or hidden are now supported.
- **Tab controls:** Create tab pages inside dialogs.
- **Spin controls:** Add up/down arrows to text fields.
- **New dialog APIs:** Allow you to remove action area buttons such as OK and Close from TGB dialogs.

1.2. Using Tecplot GUI Builder

TGB is an add-on which generates the C or FORTRAN source code used to create dialogs and controls (push buttons, text fields, scales, and so on).

Dialogs are laid out by creating frames in Tecplot and adding text and geometries to the frames. TGB distinguishes among different controls based on the style of the text and on keywords that appear in the text. TGB's controls allow you to place new controls in a dialog easily, without requiring you to remember the particular text style for each type of control.

If you have enabled TGB from your **`tecplot.add`** file, it will be accessible via the Tools menu in Tecplot.

1.3. How TGB Works

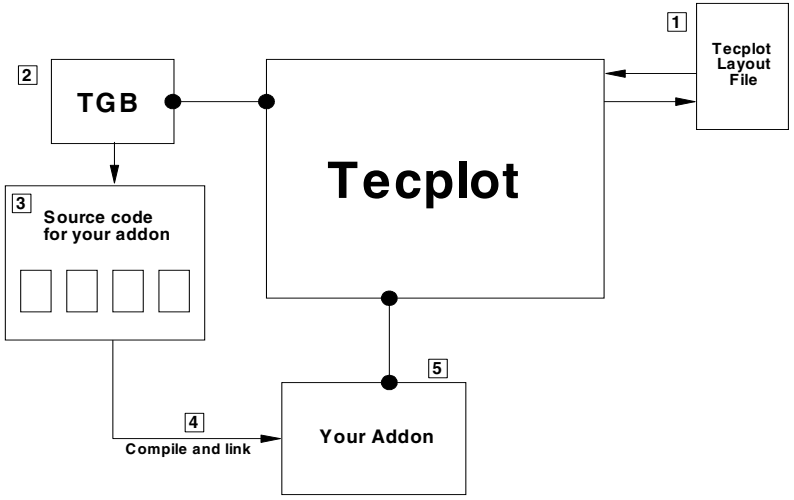

Figure 1-1. Building a graphical user interface using TGB.

Figure 1-1 shows the main steps in building a graphical user interface for your add-on using TGB. These steps are:

1. Using Tecplot and TGB (accessed via Tecplot's Tools menu), create or open an existing layout file that stores all information needed to define dialogs and the controls that go into the dialogs.
2. On the TGB dialog, select a language and the control buttons you need, then click Go Build. TGB generates source code (FORTRAN or C) to operates the controls on your GUI.
3. Modify the source code files as desired.
4. Compile and link the source code to create a shared library add-on.
5. Inform Tecplot of your new add-on.
6. Restart Tecplot—your add-on is attached.

You can repeat steps 1 through 6 as needed to make modifications to the graphical user interface for your add-on. The rest of this chapter describes each step above in detail.

1.3.1. Step 1: Building and Maintaining the GUI

If you used the Tecplot Add-on Wizard or **`CreateNewAddOn`** shell script to create your add-on, there will already be a default **`gui.lay`** file in your add-on directory, along with a number of default source code files. You are now ready to modify and/or extend the default GUI.

The following sections describe how to create and add controls to the dialogs. Before you add dialogs or controls to your GUI you must first start Tecplot and open the layout file that defines your GUI.

1.3.1.1. Adding Dialogs. This section assumes you are running Tecplot, have loaded the layout file defining your GUI, and have the Tecplot GUI Builder dialog up on the screen.

Dialogs are added by choosing either the Modeless Dialog or Modal Dialog buttons from the TGB dialog. This will create a new frame in Tecplot. The frame will have a default size and position. The frame name contains three important pieces of information that determine characteristics of the dialog to be generated. Different keywords are used in the frame name. These keywords are listed below along with a description.

Keyword	Default	Description
ID = *n*	Dynamic (TGB automatically generates a unique number).	The dialog ID is assigned to be *n*. Do not change this once you start generating source code. Each dialog must have a unique dialog ID.

Keyword	Default	Description
TITLE = *string*	Untitled	The title at the top of the dialog.
MODE = *mode*	Dynamic (the mode is set based on the type of dialog selected from the GUI Builder dialog).	The mode of the dialog. Choose between MODAL and MODELESS. If you change the mode of a dialog after generating source code, you will have to move the corresponding OK, Cancel, or Close callback functions from the **`guicb.tmp`** file to **`guicb.c`** or **`guicb.F`**.

In addition, the frame title may also begin with a comment enclosed in square brackets. The text inside the brackets is ignored by TGB. This is useful if you have a lot of dialogs and want to quickly identify and pop them using Tecplot's frame ordering function.

For example, the frame title could be:

```
[This is the main dialog] ID=1 TITLE="Main Dialog" MODE=MODAL
```

The comment shows up first in the frame ordering dialog, so you can quickly see which dialog this frame represents.

You can edit the frame name by double clicking on a frame in Tecplot and editing the frame name text field. (Or choose the Edit Current Frame option from the Frame menu.)

1.3.1.2. Adding Controls to Dialogs. This section assumes you are running Tecplot, have loaded the layout file defining your GUI, and have the Tecplot GUI Builder dialog up on the screen.

Controls are added to dialogs by choosing any one of the Controls buttons in TGB. The control is immediately added to the current frame in Tecplot. You can reposition the control and edit the text of the control.

A unique name must be assigned to controls that are not plain labels, so events generated using the control can be sent to the appropriate callback function. TGB gives you two ways to name a control. The simplest way to name a control is to assign its name by using the Macro Function Command text field assigned to text in Tecplot. Clicking Options on the Text Details dialog takes you to this feature.

If a name is not assigned as the Macro Function Command, TGB assigns a name by looking at the text used for the control. Some controls must begin with a keyword identifying the control

type. The following table lists the controls, the keyword, and text style that TGB uses to identify the control type:

Control (X Motif)	Control (Windows)	Keyword	Tecplot text style
Label.	Static text.	None.	Plain text (no text box).
Form.	Form.	**FM:Group=NN**	Filled text box.
Multi-line text field.	Multi-line edit.	**T:**	Filled text box (multi-line).
Multi-selection list.	Multi-selection list box.	**MLIST:**	Filled text box.
Option menu.	Combo box (drop-down menu).	**OPT:**	Filled text box.
Push button.	Push button.	None.	Filled text box.
Radio box.	Radio box.	**7**	Hollow text box.
Read only multi-line text field.	Multi-line read only.	**TRO:**	Filled text box.
Scale.	Slider.	**SC:**	Filled text box.
Set of tabs.	Property sheet.	**TAB:Group=NN**	Filled text box.
Single-selection list.	Single-selection list.	**SLIST:**	Filled text box.
Text field.	Edit.	**TF:**	Filled text box.
Text field with spin.	Text field with spin.	**TFS:**	Filled text box.
Toggle.	Toggle.	**7**	Plain text.

For all controls except labels and push buttons, TGB can use the text after the keyword to determine a name for the control. This is only used if the Macro Function Command field, mentioned earlier, is not used. In order for TGB to identify the control type correctly, these keywords must be used at the beginning of the actual text used for the control. TGB does not look for these keywords in the Macro Function Control field. The keywords will not show up in the compiled version of the dialog.

For example, if you have a toggle with the text **7 Include Banana**, TGB will name the control “Include Banana.” Note that **7** signifies toggles and the radio box because it resembles a check box when it is displayed on the screen. For labels, TGB just uses the label text. TGB limits the length of the name for controls to 31 characters so that code generated in FORTRAN is valid.

1.3.1.3. Adding Group Boxes and Separators. Group boxes are rectangles that surround groups of controls in a dialog. A group box can be added by simply adding a rectangle

geometry to a frame in Tecplot. In addition, you can add a label to the group box. This is done by adding the text for the label into the Macro Function field for the rectangle geometry.

Horizontal and Vertical separators can also be added by creating a simple two point line segment geometry that is either horizontal or vertical. Note that you can press the “H” or “V” keys on the keyboard while drawing the line segment and Tecplot will force it to be horizontal or vertical, respectively.

1.3.1.4. Adding Form Controls. A form control is a rectangular region of a dialog which can show and hide different sets of controls at different times.

A parent form control is a rectangular region of a dialog. A form page is a set of dialog controls which can be shown or hidden inside the parent form control. Form pages are shown as separate dialogs in TGB. You can add controls to them just as you would a regular TGB dialog. The difference is that the size of a form page dialog is always exactly the same as the size as its parent form control. Child form dialogs in TGB have a cyan background in order to distinguish them from normal dialogs.

To create a new form in a TGB dialog, click Form. This will create a new control on the dialog with the type **`FM:Group=NN`**, where **`NN`** is an automatically generated link group number. *Do not edit this text or the group number*, since it is needed by TGB to identify the control as a form and to identify the form pages associated with this control. When you add a new form to a dialog it is initially empty. In order to create sets of controls, you must add form pages.

To add a form page, click Form Page. *The button is active only if you have selected a form control*. This will create a new dialog in TGB which is actually a form page. This new dialog (a Tecplot frame) will have the same link group number as the **`Group=NN`** text in its parent form control. It will also be exactly the same size as the form control it is linked to. *Do not edit or remove the group linking from this frame*, or it will not be recognized by TGB as a form page. In the frame name will be an additional keyword, **`FORMPAGE=T`**, which identifies this dialog as a form page. See Section 1.3.1.2 for a complete list of keywords for form and tab pages.

You may add any number of form pages to a form control. Controls are added to form pages exactly like dialogs and they are built by TGB exactly like dialogs.

In your source code, TGB will generate a variable representing each form page using the parent form. This variable is similar to the **`Dialog`*`N`*`Manager`** variable that TGB creates for each dialog.

To set the controls for a form control to this set, call:

```
GUI_FormSetCurrentPage(FormN_GManager)
```

Where **G** is the group number associated with the form and **N** is the form page number.

For a sample TGB Add-on which uses forms, tabs and spin controls, see the source code for **Tabtest**.

Note: An add-on can have no more than twenty form or tab controls. This is because forms and tabs make use of frame-linking, which is limited to twenty distinct linking groups.

1.3.1.5. Adding Tab Controls. Tabs are identical to forms except that tabs have an additional set of controls above the parent form area. The form is automatically changed for you when the user clicks a tab. Everything documented about using forms also applies to tab controls. Individual tab pages are added using the Tab, and Tab Page buttons on the TGB dialog.

Tab pages have the additional keyword, **POS=NN**, in the frame name. This specifies the position from left to right. See Section 1.3.1.7 for a complete list of keywords for form and tab pages.

For a sample TGB Add-on which uses forms, tabs and spin controls, see the source code for **Tabtest**.

1.3.1.6. The Resize Button. If you change the size of a parent frame or tab control, you must click Resize on the TGB dialog. This resizes all of the linked forms or tab pages to reflect the new size of the parent form or tab. If you do not resize the tab or form pages to match the new size of the parent control they will not be sized correctly in the final GUI.

1.3.1.7. New Frame Name Keywords. Generally, you will not have to specify the **FORMPAGE**, **TABPAGE**, and **POS** keywords yourself. They are automatically generated when the appropriate TGB buttons are clicked. You should *not* edit these keywords manually. However, if you wish to reorder a set of tab pages, you may edit the **POS=*n*** value. Note that the ordinal *n's* do not have to be consecutive, since tab pages are sorted by TGB in ascending order before generating source code. For example:

```
POS=1, POS=8, POS=10
```

is equivalent to

```
POS=1,POS=2,POS=3
```

You will want to change the **TITLE** association with the tab. This is done using the Edit Current Frame dialog.

Each form and tab page frame must also have a linked group number, allowing TGB to associate a set of pages with a parent control. This is done automatically if you use Add Form, Add Tab, Add Form Page, or Add Tab Page buttons. *This is strongly recommended.*

Keyword	Default	Description
FORMPAGE=boolean	**FALSE**	**TRUE** if this dialog is a form page.
TABPAGE=boolean	**FALSE**	**TRUE** if this dialog is a tab page.
POS=n	None.	If this dialog is a form page, the **NN** is its position, with **NN=1** at the left.
TITLE="string"	Page n.	Title of the tab page.

1.3.1.8. Text Field Spin Controls. In TGB a text field spin control is a text field with two small arrow buttons anchored at the right end of the text field control. Spin controls are interchangeable with text field controls, and may be passed to any **GUI_** function requiring a text field control.

In addition to the text changed callback, spin controls also receive a callback when users click up or down arrows. It is up to the add-on to manage the text inside the control. This typically involves incrementing or decrementing a numeric value in the text control then re-displaying it. However, this is not a requirement. Spin controls may contain any text which may be changed in any way when up or down arrows are clicked.

1.3.2. Step 2: Building the Source Code

This is an easy step. Choose a Language setting on the TGB dialog, then click Go Build at the bottom of the dialog. TGB will generate the source code for your GUI and at the same time update the Tecplot layout file so it reflects the changes you have made. You can now exit Tecplot and compile your add-on.

1.3.2.1. TGB-Generated Text Files. When you finish laying out one or more dialogs, save them as a Tecplot layout file, and click Go Build at the bottom of the TGB dialog, TGB creates the following files:

C language:

- **Guicb.tmp:** Template for the callback module.

- **Guibld.c:** Interface builder module.
- **GUIDEFS.h:** Include file naming all of the controls plus some other stuff.
- **Guidefs.c:** Contains definitions of global variables.

FORTRAN language:

- **Guicb.tmp:** Template for the callback module.
- **Guibld.F:** Interface builder module.
- **GUICB.INC:** Include file naming all of the callback functions.
- **GUIDEFS.INC:** Include file naming all of the controls plus some other items.

The file **guicb.tmp** is the template for the **guicb.c** (or **guicb.F**) module you will be editing to customize all of the callbacks generated by your interface. A callback is a function that is called when the user interacts with one of the controls in your dialogs. The first time you run TGB, just rename **guicb.tmp** to **guicb.c** (or **guicb.F**). Section 1.3.3.1 goes into detail on what the **guicb.c** module is and how to modify it.

The files **guibld.c** and **guibld.F** are the C and FORTRAN interface build modules. You should never have to edit these as they simply reflect any changes made to dialogs or controls in your interface.

Note: You should never edit the file **guidefs.c** or the include files, **GUIDEFS.h** (C language), **GUIDEFS.INC** and **GUICB.INC** (FORTRAN language). If you ever modify any of these files, be aware that they will be overwritten the next time you run Tecplot GUI Builder.

1.3.3. Step 3: Modifying Your Source Code

The file **guicb.c** (or **guicb.F** for FORTRAN) contains the functions called whenever a control (that is, a button or a text field) in your interface is operated by the end user. For example, suppose you have a push button that is labeled "Eject." TGB will then create code for a function called **Eject_BTN_CB_D1** that is called when the button is pressed. TGB names the functions according to some base string that you provide ("Eject" in this case, see Section 1.3.1.2 above) plus some other decoration to uniquely identify the function. Here **BTN_CB** means this is a push button callback and **D1** means the button resides in dialog number 1.

1.3.3.1. Adding or Removing Controls. If you later decide to make changes to the interface, and the changes involve more than just the placement of controls or shape of the dialog, you must make changes to the **guicb.c** or **guicb.F** file.

For example, if you add a new push button to a dialog you would perform the following steps:

1. Look at the **guicb.tmp** template file that is generated. It contains a new callback function for the new button.
2. Cut and paste this new function from **guicb.tmp** to the existing **guicb.c** or **guicb.F** file. You can then add code to carry out the button press action.

If you remove a control from a dialog, it is not necessary to edit **guicb.c** or **guicb.F**. However, if you do not, you will end up with a callback function that is never called.

If you rename a control, you should look at **guicb.tmp** and see how TGB has now named things, then edit **guicb.c** or **guicb.F**. Change the name of the callback function to match.

1.3.3.2. Special Coding For Option Menus.

Option menus require special callback coding.

Dynamic Option Menus:

In addition to specifying a static string for the options, you may also call the new dynamic option menu functions. These dynamically add and remove strings from the option menu at run time.

The new dynamic option menu APIs are similar to the **GUI_List** APIs. For example, **GUI_OptionMenuDeleteAllItems()** removes all items in an option menu. (See the API reference for further information.)

Using C:

When generating interface code in C, TGB creates a static string in the **guicb.tmp** file that is used to store the options for the option menu. For example, if you have an option menu control with the name “Fruit” then **guicb.tmp** will contain the declaration:

```
static char *Fruit_OPT_D1_List = "Option 1,Option 2,Option 3";
```

After transferring this to **guicb.c** you can edit the string and put the items you want to appear in the option menu in the static string. Separate items with a comma. For example, the resulting declaration in **guicb.c** may appear as:

```
static char *Fruit_OPT_D1_List = "apple,banana,orange";
```

Using FORTRAN:

Option menu coding in FORTRAN is different than C, because TGB does not give you any hints as to what to add. The procedure is the following:

1. Find the spelling of the character string created to hold the option menu items. It will be located in the file **`GUIDEFS.INC`**. For example, if an option menu to assign colors is named **`coloropt`** and is in the first dialog, then you will find the variable **`coloropt_OPT_D1_List`** in **`GUIDEFS.INC`**. It is of type **`character*100`** by default.
2. Put all assignments for the character strings that define the option menus into the **`InitTecAddon`** function. It is critical that this assignment is made at the very beginning. If the color choices are “red,” “blue,” and “green,” then the statement to add to the initialization function is as follows:

```
Subroutine IntTecAddOn()
   .
   .
   .
Call TecUtilLockOn()
coloropt_OPT_D1_List = "Red,Blue,Green"
```

1.3.3.3. Adding a Menu Bar to a Dialog. Menu Bars currently must be added by hand. A menu bar is constructed as follows:

1. Call **`GUI_MenuBarAdd`**.
2. For each menu option to add to the menu bar call **`GUI_MenuAdd`** using the ID of the menu bar as the parent.
3. For each menu item added to a menu option call **`GUI_MenuAddItem`**.

Other **`GUI_Menu`** functions are available to add things such as toggled menu items and to modify the menu structure once it is in place. Also note that **`GUI_MenuAdd`** can be used to create walking menus by using another menu as the parent instead of the menu bar.

The menu creation code must only be executed once and should be done so immediately after the creation of the dialog. The best place to put the code is just after the call to **`BuildDialog()`** for the dialog. The example below demonstrates how to do this in a way that guarantees the menu bar code will only be executed once.

Create a menu bar that has the following menu structure:

```
Main Menu Bar
 +-> File
     +-> New Project ..
```

```
     +-> Open Project ...
     +-> Save Project ...
 +-> Setup
     +-> Solver Setup ...
     +-> Reference Values ...
     +-> Define Output
         +-> Print ...
         +-> Integeration ...
         +-> History Plot ...
         +-> Solution Plot ...

... in the callback to launch the dialog....

if (Dialog1Manager == BADDIALOGID)
  {
    BuildDialog1(MAINDIALOGID);

    MenuBar = GUI_MenuBarAdd(Dialog1Manager);

    FileMenu = GUI_MenuAdd(MenuBar,"File");
    NewProject_item = GUI_MenuAddItem(FileMenu,"New Project...",
                                      NewProject_MN1_D1_CB);
    OpenProject_item = GUI_MenuAddItem(FileMenu,"Open Project...",
                          OpenProject_MN1_D1_CB);
    SaveProject_item = GUI_MenuAddItem(FileMenu,"Save Project...",
                                       SaveProject_MN1_D1_CB)

    SetupMenu = GUI_MenuAdd(MenuBar,"Setup");

    SolverSetup_item = GUI_MenuAddItem(SetupMenu,"Solver Setup...",
                                       SolverSetup_MN2_D1_CB);

    ReferenceVal_item = GUI_MenuAddItem(SetupMenu,"Reference Values...",
                                        ReferenceVal_MN2_D1_CB);

    DefineOutput_menu = GUI_MenuAdd(SetupMenu,"Define Output");
    PrintOutput_item = GUI_MenuAddItem(DefineOutput_menu,"Print...",
                                       PrintOutput_MN2_D1_CB);
    IntegrationO_item = GUI_MenuAddItem(DefineOutput_menu,"Integration...",
                                        IntegrationO_MN2_D1_CB);
    HistoryPlotO_item = GUI_MenuAddItem(DefineOutput_menu,"History Plot...",
                                        HistoryPlotO_MN2_D1_CB);
    SolutionPlot_item = GUI_MenuAddItem(DefineOutput_menu,"Solution Plot...",
                                        SolutionPlot_MN2_D1_CB);
  }
...
```

1.3.4. Step 4: Compiling Your Add-On

UNIX and Mac OS X: Compiling the add-on consists of running the **`Runmake`** shell script provided in the distribution. You can run **`Runmake`** with no parameters and you will be prompted for the options, or you can put the options on the command line. For example, if your platform is sgix.62, use:

```
Runmake sgix.62 -debug
```

Note: Always use the **-debug** flag when developing add-ons. Only when you are ready to make a release version use the **-release** flag. Using **-debug** puts the resulting shared library in the appropriate location so that Tecplot will know where to get it when using the **-develop** flag.

Windows: In Developer Studio, click Build.

1.3.5. Step 5: Informing Tecplot of Your New Add-On

This step is only required if you are developing add-ons under UNIX or Mac OS X.

If you have just created this TGB add-on, then you must inform Tecplot of its existence by editing the **tecdev.add** file in the add-on development root directory and adding the entry

```
$!LoadAddon "|TECADDONDIR|/libmyaddon"
```

Where *myaddon* is the base name of your add-on.

1.3.6. Step 6: Running Your New Add-On

UNIX or Mac OS X: To run the debug version of your new add-on you must set the environment variables:

```
TECADDONDEVDIR=myaddondevdir
TECADDONPLATFORM=myplatform
```

where *myaddondevdir* is the path to the directory above your add-on projects. This is the directory from which you run **CreatNewAddOn**, to create your add-on in the first place. We recommend that you add the above environment variable settings to your **.cshrc** or **.profile** files.

myplatform is the same platform name you used with **Runmake**.

After setting up these environment variables, run Tecplot using:

```
tecplot -develop
```

Windows: In Developer Studio, click Go, or press F5.

CHAPTER 2 Function Reference

This chapter contains functions available in TGB's API library, **`libtgb.a`** (UNIX, Linux, and Mac OS X) and **`wingui.dll`** (Windows).

TGB generates code that makes use of most of these functions. All functions related to dialog creation and the addition of controls to dialogs fall into this category. Functions that will be of most interest to developers are ones that query or set the state of existing controls.

Most functions listed in the next two sections refer to a control ID. The control ID is a unique number given to each control that is added to a dialog. For example, if a push button is added to a dialog, it is added using the following function call:

C:

```
Go_BTN_D1 = GUI_AddButton(DialogID,
                          29,90,
                          126,22,
                          "Go",
                          GoFunctionCallback);
```

FORTRAN:

```
 Go_BTN_D1 = GUIF_AddButton(DialogID,
&                           29,90,
&                           126,22,
&                           'Go'//char(0),
&                           GoFunctionCallback)
```

The above code (generated automatically by TGB) adds a button to a dialog. The button is given a position, size, and default text. The function call itself returns a unique number that thereafter will refer to that button. If you later wanted to desensitize the button (that is, turn it gray and make the user unable to push it) you would use **Go_BTN_D1** as follows:

C:

```
IsSensitive = FALSE;
GUI_SetSensitivity(Go_BTN_D1,IsSensitive);
```

FORTRAN:

```
IsSensitive = 0
IErr = GUIF_SetSensitivity(Go_BTN_D1,IsSensitive)
```

All C functions start with **GUI_** and all FORTRAN functions start with **GUIF_**. Do not attempt to call **GUIF_** functions from C or **GUI_** functions from FORTRAN as this will not work. **GUI_** functions assume call by value and **GUIF_** functions assume call by reference.

2.1. Dialog Coordinate System

When you click Go Build on TGB, code is generated to create dialogs and place controls within the dialogs. In order to make the resulting dialog appearance the same on all platforms regardless of what resolution monitor you used to develop the interface, TGB uses a coordinate system based on the height and average width of the text used in the dialogs themselves.

When you click Go Build, TGB first scans all frames in your Tecplot session and checks to make sure all non-text field controls are using the same size font. If this is the case, the font height and an estimate of the average font width are used to size dialogs and controls and to position the controls themselves. The actual values used in the generated code are really the font width or font height times 100.

Throughout the rest of this reference section the terms character width units and character height units refer to the units used in the horizontal and vertical directions in dialogs, where one character width unit is equal to 1/100th of the average width of the characters used in the dialog and one character height unit is equal to 1/100th the height of the characters used in the dialog.

For example, the following code creates a push button 30 character widths from the left edge of the dialog and 10.4 character heights from the top edge of the dialog, with a button width of 16 character widths and a height of 1.5 character heights:

```
PushButton7_BTN_D1 = GUI_ButtonAdd(Dialog1Manager
        3000,    /* i.e. 30x100   */
        1040,    /* i.e. 10.4x100 */
        1600,    /* i.e. 16x100   */
        150,     /* i.e. 1.5x100  */
        "Just Do It",

PushButton7_BTN_D1_CB);
```

2.2. Function Callback Prototypes

Callback functions are called when events occur in the dialogs you create. TGB generates default callback functions for you.

GUIIntCallback_pf

Description: Type definition for a callback function with an **`int *`** parameter. Many of the GUI functions require you to provide a function that has this function prototype.

Syntax: **`void YourCallbackName(const int *`***Data*`);`**

Return Value: None.

Parameters: *Data*

Read-only pointer. Depending on the calling function it could reference a single integer or an entire array of integers, where the end of the list is

identified by a zero. The context of the control issuing the call governs the content. Guaranteed to be non-**NULL**.

Example 1: A callback function that receives an integer reference to a single value. Generally this form of callback is used by TGB when a toggle or radio button is clicked, an option menu selection changes, a scale value changes, and so forth. See Section 2.3 for specific details.

```
void MySingleValueIntFunction(const int *Data)
{
  int Selection = *Data;

  /* do something useful with the integer value */
  switch (Selection)
    {
      case 1:
        {
          /* handle selection 1 */
            .
            .
        } break;
      case 2:
        {
          /* handle selection 2 */
            .
            .
        } break;
        .
        .
      default:
        {
          /* handle unexpected selection */
            .
            .
        } break;
    }
}
```

Example 2: A callback function that receives an integer reference to a zero terminated array of integer values. Generally this form of callback is used by TGB when a list changes, etc. See Section 2.3, “C Function Syntax,” for specific details.

```
void MyMultiValueIntFunction(const int *Data)
{
  int *Selection = NULL;

  /* Traverse the list and do something useful */
```

```
/* with each integer value.  NOTE: The end   */
/* of list marker, 0, identifies there are   */
/* no more items. */
for (Selection = Data;
     *Selection != 0;
     Selection++)
  {
    switch (*Selection)
      {
        case 1:
          {
            /* Handle selection 1 */
              .
              .
          } break;
        case 2:
          {
            /* Handle selection 2 */
              .
              .
          } break;
          .
          .
        default:
          {
            /* Handle unexpected selection */
              .
              .
          } break;
      }
  }
}
```

GUITextCallback_pf

Description: Type definition for a callback function with a **`char*`** parameter. GUI functions related to text fields and multi-line text fields require you to provide a function that has this function prototype.

Syntax: `int YourCallbackName(const char *`*Data*`);`

Return Value: This function should return one if the text is valid, zero otherwise. For example, if the text field requires the user to enter a number and he or she enters a letter, you should return zero.

Note: It is the responsibility of the add-on to replace the text field value if the text is invalid.

Parameters:

Data

Read-only pointer. Text string sent to the callback function by the control issuing the call. Guaranteed to be non-**NULL**.

Example:

A callback function that receives a text string. This form of callback is used by TGB when a text field changes. The callback should check that the text is a valid entry. See Section 2.3, "C Function Syntax," for specific details.

```
int MyTextFunction(const char *Text)
{
   int IsOk = TRUE; /* Assume no errors */
   TecUtilLockOn();
   /* User should have entered a number */
   /* in this text field */
   if (!isdigit(*Text))
     IsOk = FALSE; /* Needs to be a number */

   if (IsOK)
   {
     /* Do something useful with the string */
   }
   else
   {
      TecUtilDialogErrMsg("Please enter a number");
   }

   TecUtilLockOff();
   return IsOk;
}
```

GUIVoidCallback_pf

Description:

Type definition for a callback function with no parameters. Many of the GUI functions require you to provide a function that has this function prototype.

Syntax:

```
void YourCallbackName(void);
```

Return Value: None.

Parameters: None.

Example: A callback that does not receive any arguments. Generally this form of callback is used by TGB when a button is clicked, a dialog is initialized, and so on. See Section 2.3, "C Function Syntax," for specific details.

```
void MyVoidFunction(void)
{
  /* check my dialog for correct a */
  /* correctly specified zone      */
  if (IsSpecifiedZoneValid())
    {
      /* perform a specific operation */
        .
        .
    }
  else
    {
      TecUtilDialogErrMsg("Missing or incorrectly "
                          "specified zone number.");
    }
}
```

2.3. C Function Syntax

For functions added in the future, refer to the file **`GUI.h`** in the **`include`** directory below the Tecplot home directory. This file contains all of the function prototypes, sans explanations. Most function uses are self-explanatory.

GUI_BlockForModalDialog

Description: Call this function to wait for a modal dialog to close.

Syntax: `void GUI_BlockForModalDialog(Boolean_t *`*DoneWithModalDialog*`);`

Return Value: None.

Parameters: *DoneWithModalDialog*

Pointer to a boolean variable which the add-on sets to **TRUE** to stop blocking. Typically this is done by the add-on in the OK and Cancel callback functions.

Example: Launch and block a modal dialog.

```
Boolean_t DoneWithModalDialog = FALSE

 {

  BuildDialog1(MAINDIALOGID);
  GUI_DialogLaunch(Dialog1Manager);

  GUI_BlockForModalDialog(&DoneWithModalDialog);
  /* Will not return until DoneWithModalDialog is
TRUE. */

  /* In the OK and Cancel dialog callbacks set
DoneWithModalDialog to TRUE */

  TecUtilDialogMessageBox("Finished
blocking.",MessageBox_Information);
  }
```

Note: Call to **GUI_BlockForModalDialog()** cannot be nested if the modal dialog launches its own modal dialogs. It is only valid for a single modal dialog and will issue an error if it is called when there is more than one modal dialog being displayed.

GUI_ButtonAdd

Description: Adds a button to a dialog. Note that you must call this function before calling **GUI_DialogLaunch()**.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_ButtonAdd(
    int             ParentDialogID,
    int             X,
```

```
int                 Y,
int                 Width,
int                 Height,
const char          *Label,
GUIVoidCallback_pf  ButtonCallback);
```

Return Value: The identifier of the button.

Parameters: *ParentDialogID*

ID of the parent dialog. This must be a valid dialog ID.

X

Left coordinate of the button in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the button in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the button in character width units. Must be greater than or equal to zero.

Height

Height of the button in character height units. Must be greater than or equal to zero.

Label

Label of the button. Must not be **NULL**.

ButtonCallback

Function that performs a user-defined operation when clicked. See **GUIVoidCallback_pf** for a definition example.

GUI_ButtonSetText

Description: Sets the text of a button control.

Syntax: `void GUI_ButtonSetText(int` *ButtonID*`,`
`const char *`*NewText*`);`

Return value: None.

Parameters: *ButtonID*

ID of the button.

NewText

New text for the button. This parameter cannot be **`NULL`**.

GUI_DialogAddApplyButton

Description: Call this function to add an Apply button to the action area of a TGB modeless dialog. This function must be called *before* the dialog is launched.

Syntax: `int GUI_DialogAddApplyButton(int` *DialogID*`,`
`GUIVoidCallback_pf` *ApplyButtonCallback*`);`

Return Value: *ID of the button*

This ID may be passed at any time to **`GUI_SetSensitivity()`**, setting the sensitivity of the Apply button. This function has no effect if the action area has been disabled.

Note: Do not call this function more than once per dialog.

Parameters: *DialogID*

ID of the parent dialog. The dialog must be modeless.

ApplyButtonCallback

Called when the Apply button is clicked.

GUI_DialogCreateModal

Description: Creates a modal dialog and returns the ID of the dialog. A modal dialog is one that restricts the user to acting within the dialog, and locks everything else on the screen, until the user clicks OK or Cancel. The dialog is not displayed until you call **`GUI_DialogLaunch()`**.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_DialogCreateModal(
    int                 DialogParentID,
    int                 Width,
    int                 Height,
    const char          *Title,
    GUIVoidCallback_pf  HelpButtonCallback,
    GUIVoidCallback_pf  OkButtonCallback,
    GUIVoidCallback_pf  CancelButtonCallback,
    GUIVoidCallback_pf  InitCallback);
```

Return Value: Dialog ID.

Parameters:

DialogParentID

ID of the parent dialog. You can also pass **`MAINDIALOGID`** for this parameter.

Width

Width in character width units. Must be greater than or equal to zero.

Height

Height in character height units. Must be greater than or equal to zero.

Title

Caption of the dialog. Must not be **`NULL`**.

HelpButtonCallback

Function that performs a user-defined operation when Help is clicked. See **`GUIVoidCallback_pf`** for a definition example.

OkButtonCallback

Function that performs a user-defined operation when OK is clicked. See **`GUIVoidCallback_pf`** for a definition example.

CancelButtonCallback

Function that performs a user-defined operation when Cancel is clicked. See **`GUIVoidCallback_pf`** for a definition example.

InitCallback

Function that performs a user-defined operation immediately before the dialog is displayed. See **`GUIVoidCallback_pf`** for a definition example.

Note: The dialog will not be displayed until **`GUI_DialogLaunch`** is called.

GUI_DialogCreateModeless

Description: Creates a modeless dialog and returns the ID of the dialog. A modeless dialog is one that will allow the user to interact with Tecplot and the controls within the dialog concurrently. Note that the dialog is not displayed until you call **`GUI_DialogLaunch()`**.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_DialogCreateModeless(
    int                 DialogParentID,
    int                 Width,
    int                 Height,
    const char          *Title,
    GUIVoidCallback_pf  HelpButtonCallback,
    GUIVoidCallback_pf  CloseButtonCallback,
    GUIVoidCallback_pf  InitCallback);
```

Return Value: Dialog ID.

Parameters: *DialogParentID*

ID of the parent dialog. You can also pass **MAINDIALOGID** for this parameter.

Width

Width in character width units. Must be greater than or equal to zero.

Height

Height in character height units. Must be greater than or equal to zero.

Title

Caption of the dialog. Must not be **NULL**.

HelpButtonCallback

Function that performs a user-defined operation when Help is clicked. See **GUIVoidCallback_pf** for a definition example.

CloseButtonCallback

Function that performs a user-defined operation when Close is clicked. See **GUIVoidCallback_pf** for a definition example.

InitCallback

Function that performs a user-defined operation immediately before the dialog is displayed. See **GUIVoidCallback_pf** for a definition example.

Note: The dialog will not be displayed until **GUI_DialogLaunch** is called.

GUI_DialogDismiss

Description: Closes a dialog. Usually this is called from the OK, Close, or Cancel button callbacks.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax: `void GUI_DialogDismiss(int DialogID);`

Return Value: None.

Parameters: *DialogID*

Dialog ID.

GUI_DialogEnableActionArea

Description: Shows or hides the Close and Help buttons at the bottom of TGB modeless dialogs. Use the function if you do not need the standard set of buttons at the bottom of the dialog, or if you wish to use other buttons or menu options for these functions.

This function must be called *before* the dialog is launched. It will assert if called after a dialog is launched. This function does nothing if called with a modal dialog argument.

Note: Do not call this function more than once with different values for the ShowActionArea parameter. Once this function has been called with **`FALSE`**, it cannot be called again on the same dialog with **`TRUE`**.

Syntax:
```
void GUI_DialogEnableActionArea(int DialogID,
                                Boolean_t ShowActionArea);
```

Return Value: None.

Parameters: *DialogID*

ID of the parent dialog.

ShowActionArea

Set to **`TRUE`** to show buttons at the bottom of the dialog, **`FALSE`** otherwise. Since **`TRUE`** is the default, it is unnecessary to call this function unless you wish to hide buttons.

GUI_DialogIsUp

Description: Returns **`TRUE`** if a dialog is currently displayed.

Syntax: `Boolean_t GUI_DialogIsUp(int` *DialogID*`);`

Return Value: **`TRUE`** if the dialog is currently displayed, **`FALSE`** if not.

Parameters: *DialogID*

ID of the dialog.

GUI_DialogLaunch

Description: Displays a dialog created with **`GUI_DialogCreatexxx()`**. After a dialog is launched, you cannot add any new controls to the dialog.

Syntax: `void GUI_DialogLaunch(int` *DialogID*`);`

Return Value: None.

Parameters: *DialogID*

ID of the dialog to display.

GUI_DialogSetLaunchPosition

Description: Sets the initial location of the dialog when it is launched. This is not available in FORTRAN.

Syntax:
```
void GUI_DialogSetLaunchPosition(int              DialogID,
                                 AnchorAlignment_e Placement,
                                 int              OffsetX,
                                 int              OffsetY);
```

Return Value: None.

Parameters: *DialogID*

ID of the dialog for which you are setting the launch position.

Placement

The location on the dialog that is to be considered the anchor. Possible values are:

`AnchorAlignment_TopLeft`

`AnchorAlignment_TopCenter`

`AnchorAlignment_TopRight`

`AnchorAlignment_MiddleLeft`

`AnchorAlignment_MiddleCenter`

`AnchorAlignment_MiddleRight`

`AnchorAlignment_BottomLeft`

`AnchorAlignment_BottomCenter`

`AnchorAlignment_BottomRight`

OffsetX

The X-position of the dialog.

OffsetY

The Y-position of the dialog.

GUI_DialogSetTitle

Description: Sets the title text of a dialog.

Syntax:
```
void GUI_DialogSetTitle(int DialogID,
                        const char *NewTitle);
```

Return Value: None.

Parameters: *DialogID*

ID of the dialog.

NewTitle

New title for the dialog. This parameter cannot be **`NULL`**.

GUI_DialogSetTopmost

Description: Sets a modal or modeless dialog to be the topmost window.

In Windows, calling this function with *MakeTopmost* equal to **TRUE** will add the **WS_EX_TOPMOST** style to the dialog, otherwise the style will be cleared. If set, the dialog will always remain on top of all other windows.

In UNIX, this function does nothing.

Syntax:

```
void GUI_DialogSetTopmost(
     int DialogID,
     int MakeTopmost);
```

Parameters:

DialogID

Dialog ID.

MakeTopmost

Set to one to make the dialog a topmost window, zero to make the dialog a non-topmost window.

Return Value: None.

GUI_FormAdd

Description: Adds a new form control.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_FormAdd(int ParentDialogID,
    int X,
    int Y,
    int Width,
    int Height);
```

Return Value: ID of a form which can be passed to **GUI_FormAddPage**.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

GUI_FormAddPage

Description: Creates a new form page.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax: `int GUI_FormAddPage(int` *FormID*`);`

Return Value: ID which can be passed to any **`GUI_*Add()`** function to add controls to this form page, such as buttons, text fields, and so forth.

Parameters: *FormID*

Parent form ID.

GUI_FormSetCurrentPage

Description: Sets a specific form page to be displayed.

Syntax: `void GUI_FormSetCurrent(int` *FormPageID*`);`

Return Value: None.

Parameters: *FormPageID*

ID of the form page control (the ID returned by **`GUI_FormAddPage`**).

GUI_FrameAdd

Description: Add a frame to the specified parent dialog. A frame is a box used to visually separate one control, or group of controls, from another.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_FrameAdd(
    int ParentDialogID,
    int X,
    int Y,
    int Width,
    int Height,
    const char *Label);
```

Return Value: ID of the frame control.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

Label

Text of the label added to the upper-left hand corner of the frame. If **`NULL`**, no label is added.

GUI_GetVersion

Description: Gets the version of the GUI API. *This is not the version of TGB, but rather the version of the GUI API function library.*

Syntax:
```
int GUI_GetVersion(void);
```

Return Value: The version of TGB API. The return value will be at least 100 (that is, version 1.00).

Parameters: None.

GUI_HorzSeparatorAdd

Description: Adds a horizontal separator to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_HorzSeparatorAdd(
    int ParentDialogID,
    int X,
    int Y,
    int Width);
```

Return Value: The ID of the horizontal separator.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the separator in character width units. Must be greater than or equal to zero.

GUI_LabelAdd

Description: Adds a static text label to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_LabelAdd(
    int         ParentDialogID,
    int         X,
    int         Y,
    const char *Label);
```

Return Value: The ID of the label.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the label in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the label in character height units relative to the dialog. Must be greater than or equal to zero.

Label

Text of the label. Must not be **NULL**.

GUI_LabelSetText

Description: Sets the text of a static label control.

Syntax:

```
void GUI_LabelSetText(
     int          LabelID,
     const char *LabelString);
```

Return Value: None.

Parameters: *LabelID*

ID of the label control.

LabelString

New text for the label.

GUI_ListAdd

Description: Adds a single or multi-selection list control to a dialog. After adding the list control, you can call **`GUI_ListAppendItem()`** to add items to the list control.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_ListAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    int                 IsMultiSelection,
    GUIIntCallback_pf   ValueChangedCallback);
```

Return Value: The ID of the control.

Parameters:

ParentDialogID

ID of the parent dialog. Must be a valid ID.

X

Left coordinate of the control in character width units relative to the dialog box. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog box. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

IsMultiSelection

Set to **TRUE** for a multi-selection list box, **FALSE** for a single-selection list.

ValueChangedCallback

Function that performs a user-defined operation when the list selection changes. The data passed to the callback is a zero terminated array of integers where each integer is the one-based index of a selected item. If no items are selected then the first item of the array is the zero array terminator. See **GUIIntCallback_pf** for a multi-value definition example.

GUI_ListAppendItem

Description: Appends an item to a list control.

Syntax:

```
void GUI_ListAppendItem(
     int          ListID,
     const char*  Item);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

Item

New list item. Must not be **NULL**.

GUI_ListDeallocItemList

Description: Deallocates the memory allocated when calling **GUI_ListGetSelectedItems()**.

Syntax:

```
void GUI_ListDeallocItemList(int **ItemList);
```

Return Value: None.

Parameters: *ItemList*

Address of a pointer to an int. See the example for **`GUI_ListGetSelectedItems()`**.

GUI_ListDeleteAllItems

Description: Removes all the items from a list control.

Syntax:

```
void GUI_ListDeleteAllItems(int ListID);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

GUI_ListDeleteItemAtPos

Description: Deletes an item in a list control.

Syntax:

```
void GUI_ListDeleteItemAtPos(
     int ListID,
     int Position);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

Position

One-based index of the item to delete.

GUI_ListDeselectAllItems

Description: Deselects all items in a list control.

Syntax: `void GUI_ListDeselectAllItems(int ListID);`

Return Value: None.

Parameters: *ListID*

ID of the list control.

GUI_ListGetItemCount

Description: Gets the number of items in a list control.

Syntax: `int GUI_ListGetItemCount(int ListID);`

Return Value: The number of items in the list control.

Parameters: *ListID*

ID of the list control.

GUI_ListGetSelectedItems

Description: Gets the indexes of all selected items in a list control. You can use this function for both single and multi-selected list controls.

Syntax:

```
void GUI_ListGetSelectedItems(
        int ListID,
        int **SelectedItemList,
        int *SelectedItemCount);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

SelectedItemList

Address of a pointer to an **int** (see the example below). Upon return, the pointer will contain a **NULL**-terminated array of integers. Each element of the array is the 1-based index of the selected item. You must call **GUI_ListDeallocItemList()** to free the array.

SelectedItemCount

The number of selected items is returned in this pointer.

Example:

```
int ListId; /* Returned from GUI_ListAdd() */
int count;
int *sel;
int i;
GUI_ListGetSelectedItems(ListId,&sel,&count);
for (i=0;i<count;i++)
 {
  /* Do something useful with sel[] */
 }
GUI_ListDeallocItemList(&sel); /* Clean up when done.
*/
```

GUI_ListGetString

Description: Gets the text of an item in a list box.

Syntax:

```
char *GUI_ListGetString(
     int ListID,
     int Position);
```

Return Value: The text of the item at the specified position in the list control. Note that the position is a 1-based index. You must call **TecUtilStringDealloc()** to free this pointer.

Parameters: *ListID*

ID of the list control.

Position

The one-based index of the item.

GUI_ListReplaceItem

Description: Replaces the text of an item in a list control.

Syntax:

```
void GUI_ListReplaceItem(
     int         List,
     const char *Item,
     int         Position);
```

Return Value: None.

Parameters: *List*

ID of the list control.

Item

New text of the item. Must not be **NULL**.

Position

One-based index of the item to change.

GUI_ListSelectAllItems

Description: Selects all items in a list control (if multi-selection).

Syntax:

```
void GUI_ListSelectAllItems(int ListID);
```

Return Value: None.

Parameters: *ListID*

ID of the multi-selection list control. If called for a single selection list control it will **ASSERT** with an error message.

GUI_ListSetSelectedItem

Description: Selects an item in a list control.

Syntax:
```
void GUI_ListSetSelectedItem(
     int ListID,
     int Position);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

Position

One-based position index of the item to select.

GUI_ListSetSelectedItems

Description: Selects one or more (if the list is multi-selection) items in a list control.

Syntax:
```
void GUI_ListSetSelectedItems(
     int ListID,
     int *SelectedItemList,
     int SelectedItemCount);
```

Return Value: None.

Parameters: *ListID*

ID of the list control.

SelectedItemList

Array of one-based indices. Each element of the array is the index of an item to select in the List control.

SelectedItemCount

Number of elements in the array.

GUI_MenuAdd

Description: Add a menu to a menu bar or a walking menu to a menu list.

Syntax:

```
int GUI_MenuAdd(
    int          ParentMenuID,
    const char *Label);
```

Return Value: Returns the ID of the menu.

Parameters: *ParentMenuID*

ID of the menu bar or menu list that this menu is to be added.

Label

Text to place on the menu. Use an ampersand (**&**) to mark the mnemonic. The character immediately following **&** will be the mnemonic and the **&** is removed from the final text.

Example: Add a menu item to Menu Bar called Options with t as the mnemonic.

```
{
  int OptionMenu;
  OptionMenu = GUI_MenuAddItem(MenuBar,Op&tions);
}
```

In the Options menu of the menu bar, the ampersand (**&**) will not show, but t will be underlined. The Options menu may be selected by pressing T on your keyboard.

GUI_MenuAddItem

Description: Add a menu item to a menu list.

Syntax:

```
int GUI_MenuAddItem(
    int                  ParentMenuID,
    const char           *Label,
    GUIVoidCallback_pf   Callback);
```

Return Value: Returns the ID of the menu item.

Parameters: *ParentMenuID*

ID of the parent menu.

Label

Text to put on the menu item.

Callback

Name of the function to call when this menu item is selected. See Section 2.2, "Function Callback Prototypes."

GUI_MenuAddSeparator

Description: Add a separator to a menu list.

Syntax:

```
void GUI_MenuAddSeparator(
     int ParentMenuID);
```

Return Value: None.

Parameters: *ParentMenuID*

ID of the menu list.

GUI_MenuAddToggle

Description: Add a menu item with a toggle to a menu list.

Syntax:

```
int GUI_MenuAddToggle(
    int                 ParentMenuID,
    const char          *Label,
    GUIIntCallback_pf   Callback);
```

Return Value: ID of the menu toggle.

Parameters: *ParentMenuID*

ID of the parent menu list.

Label

Text to place on the menu item. Use an ampersand (**&**) to mark the mnemonic. The character immediately following **&** will be the mnemonic and the **&** is removed from the final text. See **GUI_MenuAdd** for an example of using a mnemonic.

Callback

Name of the function called when the menu item toggle is selected. See Section 2.2, "Function Callback Prototypes."

GUI_MenuBarAdd

Description: Add a menu bar to an existing dialog.

Syntax:
```
int GUI_MenuBarAdd(
    int ParentDialogID);
```

Return Value: ID of the menu bar added.

Parameters: *ParentDialogID*

ID of the parent dialog.

GUI_MenuDeleteItem

Description: Delete a menu item from a menu list.

Syntax:
```
void GUI_MenuDeleteItem(
     int MenuItemID);
```

Return Value: None.

Parameters: *MenuItemID*

ID of the menu item to delete.

GUI_MenuItemSetText

Description: Set the text for a menu item.

Syntax:

```
void GUI_MenuItemSetText(
     int          MenuItemID,
     const char   *NewText);
```

Return Value: None.

Parameters: *MenuItemID*

ID of the menu item.

NewText

Text to assign to the menu item.

GUI_MenuSetToggle

Description: Set the state of a menu item toggle.

Syntax:

```
void GUI_MenuSetToggle(
     int MenuItemID,
     int SetOn);
```

Return Value: None.

Parameters: *MenuItemID*

ID of the menu item.

SetOn

Value to assign to the toggle. One = On, zero = Off.

GUI_OptionMenuAdd

Description: Adds an option menu control to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_OptionMenuAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    const char*         OptionList,
    GUIIntCallback_pf   ValueChangedCallback);
```

Return Value: ID of the control.

Parameters:

ParentDialogID

ID of the parent dialog. Must be a valid dialog ID.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

OptionList

Options in the control. Separate each option with a comma. For example, "Bart, Lisa, Homer, Marge." Options are not sorted; you can assume the order of the items will not change. Must not be **NULL**.

ValueChangedCallback

Function that performs a user-defined operation when the option menu selection changes. The data passed to the callback is a reference to the one-based index of the selected item. See **GUIIntCallback_pf** for a single-value definition example.

GUI_OptionMenuAppendItem

Description: Appends an item to an option menu control.

Syntax:
```
void GUI_OptionMenuAppendItem(int OptionMenuID,
                              const char  *Item);
```

Return Value: None.

Parameters: *OptionMenuID*

ID of the option menu control.

Item

New option menu item. Must be a valid string with length greater than zero.

GUI_OptionMenuDeleteAllItems

Description: Remove all items from an option menu.

Syntax:
```
void GUI_OptionMenuDeleteAllItems(int OptionMenuID)
```

Return Value: None.

Parameters: *OptionMenuID*

ID of the option menu control.

GUI_OptionMenuDeleteItemAtPos

Description: Delete an item in an option menu control.

Syntax:
```
void GUI_OptionMenuDeleteItemAtPos(
                                    int OptionMenuID,
                                    int Position);
```

Return Value: None.

Parameters: *OptionMenuID*

ID of the option menu.

Position

One-based index of the item to delete.

GUI_OptionMenuGet

Description: Get the current selection in an option menu.

Syntax: `int GUI_OptionMenuGet(int OptionMenuID)`

Return Value: Number of the currently selected option. Options are numbered starting at one. This function will **ASSERT** if there are no items in the option menu.

Parameters: *OptionMenuID*

ID of the option menu control.

GUI_OptionMenuGetItemCount

Description: Get the number of items in an option menu.

Syntax: `int GUI_OptionMenuGetItemCount(int` *OptionMenuID*`)`

Return Value: Number of items in the option menu.

Parameters: *OptionMenuID*

ID of the option menu control.

GUI_OptionMenuGetString

Description: Get the text of an item in an option menu.

Syntax:
```
char *GUI_OptionMenuGetString(int OptionMenuID,
                              int Position);
```

Return Value: The text of the item at the specified position in the option menu control. The position is a one-based index. You must call `TecUtilStringDealloc()` to free this string.

Parameters: *OptionMenuID*

ID of the option menu.

Position

The one-based index of the item.

GUI_OptionMenuReplaceItem

Description: Replace the text of an item in an option menu control.

Syntax:
```
void GUI_OptionMenuReplaceItem(
                                    int OptionMenuID,
                                    const char *NewText,
```

```
                                                    int Position);
```

Return Value: None.

Parameters: *OptionMenuID*

ID of the option menu.

NewText

New text of the item. Must be a valid string with a length greater than zero.

Position

The one-based index of the item.

GUI_OptionMenuSet

Description: Set the current option in an option menu.

Syntax:

```
void GUI_OptionMenuSet(
        int         OptionMenuID,
        int         Selection)
```

Parameters: *OptionMenuID*

ID of the option menu control.

Selection

The number of the option to set as the default. Options are numbered starting at one.

GUI_RadioBoxAdd

Description: Adds a set of radio box controls to a dialog. You must call this function before calling **`GUI_DialogLaunch()`**. *Radio boxes are limited to five toggles.*

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_RadioBoxAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    const char          *Label1,
    const char          *Label2,
    const char          *Label3,
    const char          *Label4,
    const char          *Label5,
    GUIIntCallback_pf   ValueChangedCallback);
```

Return Value:

The ID of the radio box control.

Parameters:

ParentDialogID

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

Label1

Label of the first option button. Must not be **NULL**.

Label2

Label of the second radio button Must not be **NULL**.

Label3

Label of the third radio button. Can be **NULL** if *Label4* and *Label5* are **NULL**.

Label4

Label of the fourth radio button. Can be **NULL** if *Label5* is **NULL**.

Label5

Label of the fifth radio button. Can be **NULL**.

ValueChangedCallback

Function that performs a user-defined operation when the selected button within the radio box changes. The data passed to the callback is a reference to the one-based index of the selected radio button. See **GUIIntCallback_pf** for a single-value definition example.

GUI_RadioBoxGetToggle

Description: Get the current radio box selection.

Syntax: `int GUI_RadioBoxGetToggle(int` *RadioBoxID*`);`

Return Value: Number of the radio box control that is active. Toggles are numbered starting at one.

Parameters: *RadioBoxID*

ID of the radio box.

GUI_RadioBoxSetToggle

Description: Sets a radio button in radio box control.

Syntax:

```
void GUI_RadioBoxSetToggle(
     int RadioBoxID,
     int ToggleNumber);
```

Parameters: *RadioBoxID*

ID of the radio box.

ToggleNumber

One-based index of the radio button to select.

GUI_ScaleAdd

Description: Adds a scale control to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_ScaleAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    int                 ScaleMin,
    int                 ScaleMax,
    int                 DecimalPrecision,
    GUIIntCallback_pf   ValueChangedCallback,
    GUIIntCallback_pf   DragValueChangedCallback);
```

Return Value: The ID of the scale control.

Parameters:

ParentDialogID

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog.

Y

Top coordinate of the control in character height units relative to the dialog.

Width

Width of the control in character width units.

Height

Height of the control in character height units.

ScaleMin

The minimum position of the scale. Usually zero.

ScaleMax

The maximum position of the scale.

DecimalPrecision

Specifies the number of decimal points to shift the slider value when displaying it. For example, a slider value of 2,350 and a *DecimalPrecision* value of 2 results in a display value of 23.50.

ValueChangedCallback

Function that performs a user-defined operation when the scale's value changes (for drag notification use the **`DragValueChangedCallback`**). The data passed to the callback is a reference to the scale value. See **`GUIIntCallback_pf`** for a single-value definition example.

DragValueChangedCallback

Function that performs a user-defined operation when the scale's value changes while dragging the scale slider. The data passed to the callback is a reference to the current scale value. See **`GUIIntCallback_pf`** for a single-value definition example.

GUI_ScaleGetValue

Description: Sets the current position of a scale control.

Syntax: `int GUI_ScaleGetValue(int` *ScaleID*`);`

Return Value: Current value of the scale.

Parameters: *ScaleID*

ID of the scale.

GUI_ScaleSetLimits

Description: Set the limits (that is, minimum and maximum values) and decimal precision of a scale control.

Syntax:

```
void GUI_ScaleSetLimits(
     int ScaleID,
     int ScaleMin,
     int ScaleMax,
     int DecimalPrecision);
```

Return Value: None.

Parameters: *ScaleID*

ID of the scale control.

ScaleMin

Minimum value of the scale.

ScaleMax

Maximum value of the scale.

DecimalPrecision

Decimal precision of the scale. See **`GUI_ScaleAdd()`** for a description.

GUI_ScaleSetValue

Description: Sets the current position of a scale control.

Syntax:
```
void GUI_ScaleSetValue(
     int ScaleID,
     int NewValue);
```

Return Value: None.

Parameters: *ScaleID*

ID of the scale.

NewValue

New value of the scale.

GUI_ScaleShowNumericDisplay

Description: Turns numeric display of a scale on or off. This function may be called at any time.

Syntax:
```
void GUI_ScaleShowNumericDisplay(int ScaleID,
                                 int ShowDisplay);
```

Return Value: None.

Parameters: *ScaleID*

ID of the scale control.

ShowDisplay

Use **TRUE** to show the numeric display, **FALSE** to hide it.

GUI_SetSensitivity

Description: Sets the sensitivity (in Windows, the enabled state) of a control.

Syntax:
```
void GUI_SetSensitivity(
     int ControlID,
     int IsSensitive);
```

Return Value: None.

Parameters: *ControlID*

ID of the control.

IsSensitive

TRUE to set the state of the control to sensitive, **FALSE** otherwise.

GUI_SetVisibility

Description: Sets the visibility of a control.

Syntax:
```
void GUI_SetVisibility(int ControlID,
                       int MakeVisible);
```

Return Value: None.

Parameters: *ControlID*

ID of the control.

MakeVisible

TRUE to make the control visible, **FALSE** to make the control invisible.

GUI_SpinTextFieldAdd

Description: Adds a spin text field to a dialog.

Note: TGB automatically generates code that uses the function. Only in rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_SpinTextFieldAdd(int ParentDialogID,
                         int X,
                         int Y,
                         int Width,
                         int Height,
                         GUITextCallback_pf ValueChangedCallback
                         GUIVoidCallback_pf ButtonUpCallback,
                         GUIVoidCallback_pf ButtonDownCallback);
```

Return Value:

ID of the control. A spin control is a kind of text field control. Thus, the ID can be passed to any **`GUI_*`** requiring a text field control ID.

Parameters:

ParentDialogID

Parent dialog ID.

X

Left coordinate of the control in character width units relative to the dialog.

Y

Top coordinate of the control in character height units relative to the dialog.

Width

Width of the control in character width units.

Height

Height of the control in character width units.

ValueChangedCallback

Function that performs a user-defined operation when the text values changes. The data passed to the callback is the text's new value. See **`GUITextCallback_pf`** for an example.

ButtonUpCallback

Called when the up button is clicked. Typically you will increment and redisplay the numeric value in the text control, however, this is not a requirement.

ButtonDownCallback

Called when the down button is clicked. Typically you will decrement and redisplay the number value in the text field, however, this is not a requirement.

GUI_TabAdd

Description: Adds a tab control to a dialog.

Note: TGB automatically generates code that uses the function. Only in rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_TabAdd(int ParentDialogID,
               int X,
                  int Y,
                  int Width,
                  int Height,
                  GUIIntCallback ActivateCallback,
                  GUIIntCallback DeactivateCallback);
```

Return Value: ID of the tab control. This ID is used only to identify the tab control when adding tab pages to the control. To add controls to a tab page, you must call **`GUI_TabAddPage()`** with the ID returned from this function.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog.

Y

Top coordinate of the control in character height units relative to the dialog.

Width

Width of the control in character width units.

Height

Height of the control in character width units.

Activate Callback

Called when a tab page is activated. The data passed is the ID of the activated tab page.

Deactivate Callback

Called when a tab page is deactivated. The data passed is the ID of the deactivated tab page.

GUI_TabAddPage

Description: Adds a page to a tab control. The ID returned from this function may be passed to any **`GUI_*Add`** function to add controls such as buttons, text fields, and so forth, to this tab page.

Note: TGB automatically generates code that uses the function. Only in rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_TabAddPage(int TabID,
                   const char *Caption);
```

Return Value: ID of the tab page. This ID is returned from **`GUI_TabAdd()`**. It may be passed to any **`GUI_*Add`** function to add controls such as buttons, text fields, and so forth, to this tab page.

Parameters: *TabID*

Parent tab control ID.

Caption

Caption of this tab control. Must be a valid string of length greater than zero.

GUI_TabSetCurrentPage

Description: Sets a specific tab page of a tab control as the current tab page.

Syntax:
```
void GUI_TabSetCurrentPage(int TabID,
                           int PageID);
```

Return Value: The ID of the text control.

Parameters: *TabID*

ID of the parent tab control.

PageID

ID of the page to set as the current tab page.

Note: Calling this function does not generate Activate and Deactivate callback events for the tab page argument. These callbacks are generated only when the user clicks a tab page with their mouse.

GUI_TextAdd

Description: Adds a multi-line text control to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:
```
int GUI_TextAdd(int ParentDialogID,
                int X,
                int Y,
                int Width,
```

```
int          Height,
int          IsReadOnly,
GUITextCallback_pf  ValueChangedCallback);
```

Return Value: The ID of the text control.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

IsReadOnly

Set to **TRUE** to make the control read-only, otherwise set to **FALSE**.

ValueChangedCallback

Function that performs a user-defined operation when the text value changes. The data passed to the callback is the text's new value. See **GUITextCallback_pf** for a definition and example. This parameter may be **NULL** if *IsReadOnly* is **TRUE**.

GUI_TextAppendString

Description: Appends a string to the end of a multi-line text control.

Syntax:

```
void GUI_TextAppendString(
     int          TextID,
     const char *TextString);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

TextString

Text to insert. Must not be **NULL**.

GUI_TextFieldAdd

Description: Adds a text field control to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_TextFieldAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    GUITextCallback_pf  ValueChangedCallback);
```

Return Value: The ID of the text control.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

ValueChangedCallback

Function that performs a user-defined operation when the text value changes. The data passed to the callback is the text's new value. See **`GUITextCallback_pf`** for a definition and example.

GUI_TextFieldGetString

Description: Gets the text in a text field control.

Syntax: `char *GUI_TextFieldGetString(int` *TextID*`);`

Return Value: The text current in the control. You must call **`TecUtilStringDealloc()`** to free the returned pointer.

Parameters: *TextID*

ID of the text field control.

GUI_TextFieldSetString

Description: Sets the text in a text field control. The previous contents of the text field control are erased.

Syntax: `void GUI_TextFieldSetString(`

```
int         TextID,
const char *TextString);
```

Return Value: None.

Parameters: *TextID*

ID of the text field control.

TextString

New string to place in the text control. Must not be **NULL**.

GUI_TextGetString

Description: Gets the text in a multi-line text control. Lines are separated by new line characters ('**\n**') only.

Syntax: `char *GUI_TextGetString(int TextID);`

Return Value: The text current in the control. You must call **TecUtilStringDealloc()** to free the returned pointer.

Parameters: *TextID*

ID of the text control.

GUI_TextInsertString

Description: Inserts text into a multi-line text control. The next text is inserted to the right of the current text insert position. Use **GUI_TextSetInsertPos** to set the text insert position. Individual lines of the text are delimited by the '**\n**' character.

Syntax:

```
void GUI_TextInsertString(
     int         TextID,
     const char *TextString);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

TextString

Text to insert. Must not be **NULL**.

GUI_TextSetInsertPos

Description: Set the text insert position at the specified position in the text string. Text is inserted to the right of the specified position. To insert text at the beginning, set the insert position to zero. To insert text at the end, set the insert position to the length of the string currently maintained by the multi-line text control.

See also **GUI_TextSetMinInsertPos** and **GUI_TextSetMaxInsertPos**.

In Windows, the insert position is the position of the caret.

Syntax:

```
void GUI_TextSetInsertPos(
     int TextID,
     int Position);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

Position

Insert position within the text limits: greater than or equal to zero, and less than or equal to the length of the text string maintained by the multi-line text control.

GUI_TextSetMaxInsertPos

Description: Set the text insert position at the maximum position in the text string. Text inserted at the maximum position places the text at the end of the text string maintained by the multi-line text control.

Syntax:
```
void GUI_TextSetMaxInsertPos(int TextID);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

GUI_TextSetMinInsertPosition

Description: Set the insert position to before the first character in text string maintained by the multi-line text control. This is equivalent to calling `GUI_TextSetInsertPos(id,0)`.

Syntax:
```
void GUI_TextSetMinInsertPos(int TextID);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

GUI_TextSetString

Description: Sets the text in a multi-line text control. The previous contents of the multi-line text control are erased.

Syntax:
```
void GUI_TextSetString(
     int         TextID,
     const char *TextString);
```

Return Value: None.

Parameters: *TextID*

ID of the text control.

TextString

New string to copy into the text control. Must not be **NULL**.

GUI_ToggleAdd

Description: Adds a toggle control to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_ToggleAdd(
    int                 ParentDialogID,
    int                 X,
    int                 Y,
    int                 Width,
    int                 Height,
    const char          *Label,
    GUIIntCallback_pf   ValueChangedCallback);
```

Return Value: ID of the toggle.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Width

Width of the control in character width units. Must be greater than or equal to zero.

Height

Height of the control in character height units. Must be greater than or equal to zero.

Label

Text of the control. Must not be **`NULL`**.

ValueChangedCallback

Function that performs a user-defined operation when the toggle value changes. The data passed to the callback is a reference to the toggle state: one if the toggle is set, otherwise zero. See **`GUIIntCallback_pf`** for a definition example.

GUI_ToggleGet

Description: Get the current value of a toggle.

Syntax:
```
int GUI_ToggleGet(int ToggleID);
```

Return Value: The current value of a toggle. Returns one if the toggle is set and zero if unset.

Parameters: *ToggleID*

ID of the toggle control.

GUI_ToggleSet

Description: Sets or clears a toggle control.

Syntax:
```
void GUI_ToggleSet(
     int ToggleID,
```

```
    int SetOn);
```

Return Value: None.

Parameters: *ToggleID*

ID of the toggle control.

SetOn

Pass **TRUE** to set the toggle, **FALSE** to clear it.

GUI_VertSeparatorAdd

Description: Adds a vertical separator to a dialog.

Note: TGB automatically generates code that uses this function. Only under rare circumstances will you need to call this function directly yourself.

Syntax:

```
int GUI_VertSeparatorAdd(
    int ParentDialogID,
    int X,
    int Y,
    int Height);
```

Return Value: The ID of the vertical separator.

Parameters: *ParentDialogID*

ID of the parent dialog.

X

Left coordinate of the control in character width units relative to the dialog. Must be greater than or equal to zero.

Y

Top coordinate of the control in character height units relative to the dialog. Must be greater than or equal to zero.

Height

Height of the separator in character height units. Must be greater than or equal to zero.

2.4. FORTRAN Function Syntax

This section shows the syntax for the FORTRAN functions equivalent to the **GUI_** functions of the previous section. For a complete discussion of how to use each function see the corresponding **GUI_** function in the previous section.

This section only documents the FORTRAN syntax and any additional notes related to differences in how the function is used in FORTRAN as compared to the c function equivalent.

Functions related the creation of dialogs and controls are not listed in this section. Those functions are automatically created and maintained for you by TGB itself.

```
 SUBROUTINE GUIF_DialogDismiss(DialogID)
 INTEGER*4 DialogID


 INTEGER*4 FUNCTION GUIF_DialogIsUp(DialogID)
 INTEGER*4 DialogID


 SUBROUTINE GUIF_DialogLaunch(DialogID)
 INTEGER*4 DialogID


 SUBROUTINE GUIF_LabelSetText(Label,
&                             LabelString)
 INTEGER*4 Label
 CHARACTER*(*) LabelString


 SUBROUTINE GUIF_ListAppendItem(List,
&                               Item)
 INTEGER*4 List
 CHARACTER*(*) Item
```

```
 SUBROUTINE GUIF_ListDeleteAllItems(List)
 INTEGER*4 List


 SUBROUTINE GUIF_ListDeleteItemAtPos(List,
&                                    Position)
 INTEGER*4 List
 INTEGER*4 Position


 SUBROUTINE GUIF_ListDeselectAllItems(List)
 INTEGER*4 List


 INTEGER*4 FUNCTION GUIF_ListGetItemCount(List)
 INTEGER*4 List


 SUBROUTINE GUIF_ListGetSelectedItems(List,
&                                     MaxSelectedItemCount,
&                                     SelectedItemList,
&                                     SelectedItemCount)
 INTEGER*4 List
 INTEGER*4 MaxSelectedItemCount
 INTEGER*4 SelectedItemList(1)
 INTEGER*4 SelectedItemCount


 SUBROUTINE GUIF_ListGetString(List,
&                              Position,
&                              MaxCharacters,
&                              ItemString)
 INTEGER*4     List
 INTEGER*4     Position
 INTEGER*4     MaxCharacters)
 CHARACTER*(*) ItemString


 SUBROUTINE GUIF_ListReplaceItem(List,
```

```
&                                  Item,
&                                  Position)
 INTEGER*4     List
 CHARACTER*(*) Item
 INTEGER*4     Position


 SUBROUTINE GUIF_ListSetSelectedItem(List,
&                                    Position)
 INTEGER*4 List
 INTEGER*4 Position


 SUBROUTINE GUIF_ListSetSelectedItems(List,
&                                     SelectedItemList,
&                                     SelectedItemCount)
 INTEGER*4 List
 INTEGER*4 SelectedItemList(1)
 INTEGER*4 SelectedItemCount


 INTEGER*4 FUNCTION GUIF_OptionMenuGet(OptionMenu)
 INTEGER*4 OptionMenu


 SUBROUTINE GUIF_OptionMenuSet(OptionMenu,
&                              Selection)
 INTEGER*4 OptionMenu
 INTEGER*4 Selection


 INTEGER*4 FUNCTION GUIF_RadioBoxGetToggle(RadioBox)
 INTEGER*4 RadioBox


 SUBROUTINE GUIF_RadioBoxSetToggle(RadioBox,
&                                  ToggleNumber)
 INTEGER*4 RadioBox
 INTEGER*4 ToggleNumber
```

```
 INTEGER*4 FUNCTION GUIF_ScaleGetValue(Scale)
 INTEGER*4 Scale


 SUBROUTINE GUIF_ScaleSetLimits(Scale,
&                               ScaleMin,
&                               ScaleMax,
&                               DecimalPrecision)
 INTEGER*4 Scale
 INTEGER*4 ScaleMin
 INTEGER*4 ScaleMax
 INTEGER*4 DecimalPrecision


 SUBROUTINE GUIF_ScaleSetValue(Scale,
&                              NewValue)
 INTEGER*4 Scale
 INTEGER*4 NewValue


 SUBROUTINE GUIF_SetSensitivity(Control,
&                               IsSensitive)
 INTEGER*4 Control
 INTEGER*4 IsSensitive


 SUBROUTINE GUIF_SetVisibility(Control,
&                              IsVisible)
 INTEGER*4 Control
 INTEGER*4 IsVisible


 SUBROUTINE GUIF_TextAppendString(TextControl,
&                                 TextString)
 INTEGER*4     TextControl
 CHARACTER*(*) TextString


 SUBROUTINE GUIF_TextFieldGetString(TextField,
&                                   MaxCharacters,
&                                   TextString)
```

```
 INTEGER*4     TextField
 INTEGER*4     MaxCharacters
 CHARACTER*(*) TextString


 SUBROUTINE GUIF_TextFieldSetString(TextField,
&                                   TextString)
 INTEGER*4     TextField
 CHARACTER*(*) TextString


 SUBROUTINE GUIF_TextGetString(TextControl,
&                              MaxCharacters,
&                              TextString)
 INTEGER*4     TextControl
 INTEGER*4     MaxCharacters
 CHARACTER*(*) TextString


 SUBROUTINE GUIF_TextInsertString(TextControl,
&                                 TextString)
 INTEGER*4     TextControl
 CHARACTER*(*) TextString


 SUBROUTINE GUIF_TextSetInsertPos(TextControl,
&                                 Position)
 INTEGER*4 TextControl
 INTEGER*4 Position


 SUBROUTINE GUIF_TextSetMaxInsertPos(TextControl)
 INTEGER*4 TextControl


 SUBROUTINE GUIF_TextSetMinInsertPos(TextControl)
 INTEGER*4 TextControl


 SUBROUTINE GUIF_TextSetString(TextControl,
&                              TextString)
```

```
 INTEGER*4     Text
 CHARACTER*(*) TextString


 INTEGER*4 FUNCTION GUIF_ToggleGet(Toggle)
 INTEGER*4 Toggle


 SUBROUTINE GUIF_ToggleSet(Toggle,
&                          SetOn)
 INTEGER*4 Toggle
 INTEGER*4 SetOn


 SUBROUTINE GUIF_OptionMenuDeleteItemAtPos(OptionMenu,
&                                          Position)
 INTEGER*4 OptionMenu
 INTEGER*4 Position


 SUBROUTINE GUIF_OptionMenuAppendItem(OptionMenu,
&                                     Item,
&                                     Item_MAXLEN)
 INTEGER*4 OptionMenu
 CHARACTER*(*) Item
 INTEGER*4 Item_MAXLEN


 INTEGER*4 FUNCTION_GUIF_OptionMenuGetItemCount(OptionMenu)
 INTEGER*4 OptionMenu


 SUBROUTINE GUIF_OptionMenuDeleteAllItems(OptionMenu)
 INTEGER*4 OptionMenu


 SUBROUTINE STDCALL GUIF_OptionMenuGetString(OptionMenu,
&                                            Position,
&                                            MaxCharacters,
&                                            ItemString,
&                                            ItemString_MAXLEN)
```

```
 INTEGER*4 OptionMenu
 INTEGER*4 Position
 INTEGER*4 MaxCharacters
 CHARACTER*(*) ItemString
 INTEGER*4 ItemString_MAXLEN


 SUBROUTINE GUIF_OptionMenuReplaceItem(OptionMenu,
&                                      NewText,
&                                      NewText_MAXLEN,
&                                      Position)
 INTEGER*4 OptionMenu
 CHARACTER*(*) NewText
 INTEGER*4 NewText_MAXLEN
 INTEGER*4 Position


 SUBROUTINE GUIF_FormSetCurrentPage(FormID)
 INTEGER*4 FormID


 SUBROUTINE GUIF_ScaleShowNumericDisplay(ScaleID,
&                                        ShowDisplay)
 INTEGER*4 ScaleID
 INTEGER*4 ShowDisplay


 SUBROUTINE GUIF_DialogEnableActionArea(DialogID,
&                                       ShowActionArea)
 INTEGER*4 DialogID
 INTEGER*4 ShowActionArea


 SUBROUTINE GUIF_ListSelectAllItems(List)
 INTEGER*4 List
```

Index

T