SLgtk is a software package which provides Gtk bindings for the S-Lang scripting language.

Copyright (C) 2003-2005 Massachusetts Institute of Technology
Copyright (C) 2002 Michael S. Noble <mnoble@space.mit.edu>

This software was partially developed at the MIT Center for Space Research, under contract SV1-61010 from the Smithsonian Institution.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in the supporting documentation, and that the name of the Massachusetts Institute of Technology not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. The Massachusetts Institute of Technology makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
# Contents

1 Introduction

1.1 Code Generation .................................................. 2

2 Some Examples ....................................................... 3

3 Programming Interface .............................................. 7

3.1 Memory Management ............................................... 7
3.2 Gtk #define Macros ............................................... 8
3.3 Variable Length Argument Lists ................................. 8
3.4 Widget Field Accessor Functions ............................... 8
3.5 Typing ............................................................... 9
3.5.1 Unique S-Lang Opaque Types ................................. 9
3.5.2 Gtk to S-Lang Type Mappings ............................... 10
3.5.3 GTypes and Properties .................................... 14
3.5.4 GdkColor ....................................................... 14
3.5.5 GdkRectangle ................................................. 15
3.5.6 GtkAllocation ............................................... 15
3.5.7 GdkPoint ..................................................... 15
3.5.8 GdkEvent .................................................... 15
3.5.9 GtkTextIter .................................................. 15
3.5.10 GError ...................................................... 16
3.5.11 GtkMenu ..................................................... 16
3.6 Signal Handlers and Callbacks ................................ 16
3.7 Callback Errors .................................................. 16

4 Data Exploration With vWhere .................................. 19
5 Printing ................................. 23
  5.1 _print_dialog .................... 23
  5.2 _print_context_new() .......... 24

6 Interactive Plotting ................. 25
  6.0.1 _gtk_plot ...................... 25
  6.0.2 _gtk_oplot .................... 25
  6.0.3 _gtk_plot_set_x_range, _gtk_plot_set_y_range .......... 26
  6.0.4 _gtk_plot_redraw ............ 26
  6.0.5 _gtk_plot_remove ............. 26

7 Miscellaneous ....................... 27
  7.1 Default GdkColor Variables .... 27
  7.2 _menu_new ....................... 27
  7.3 _option_menu_new ............... 28
  7.4 _color_button_new .............. 28
  7.5 _info_window ................... 28
  7.6 _is_numeric ..................... 28
  7.7 _is_callable ..................... 28
  7.8 _get_slgtk_doc_string .......... 29
  7.9 _gtk_window_destroy_with(window,parent) .......... 29
  7.10 gdk_pixbuf_get_formats() ...... 29

8 Development and Debugging Utilities 31
  8.1 slgtksh .......................... 31
  8.2 importify ......................... 31
  8.3 slirp_debug_pause ............... 32
  8.4 slgtk-deno ....................... 32
  8.5 checkgtk ......................... 32

9 Versioning .......................... 33
Chapter 1

Introduction

The SLgtk package binds the GIMP Toolkit, also known as Gtk, to the S-Lang scripting language. It provides an importable module which makes most of Gtk and its constituent libraries callable directly from S-Lang scripts. With SLgtk the S-Lang programmer now has access to a powerful, cross-platform widget set for creating sophisticated graphical user interfaces (GUIs). Bindings to a subset of the GtkExtra widget set are also provided, for 2D visualization.

A driving force behind the development of SLgtk has been the notion of the guilet (pronounced "gooey-let"), by which we mean to connote visual interfaces of a small-ish, scriptable nature, that may be easily embedded within – and launched from – other programs, even applications with interactive command lines and no ostensible graphical interface.

As such guilets fill an important software niche, the middle ground between novice users – who often argue that visual interfaces make software easier to use – and power users – who frequently complain that GUIs are more cumbersome and inflexible than command lines. Experience suggests that the code size and development cycle of the typical guilet is considerably smaller than that of traditional, monolithic GUIs (by which we mean to connote applications coded from scratch to have only a visual interface, typically utilizing a compiled language in conjunction with lower-level toolkits such as Xt, Motif, or the Microsoft’s Windows Foundation Classes). Moreover, given their scripting heritage, guilets can be more amenable to changing requirements or ad-hoc experimentation and customization by the end-user.

Other benefits of embedded guilets are the memory conservation and seamlessness of operation which can result from having visualization capabilities located more closely to the data upon which they will operate. That is, it can be vastly preferable to generate sophisticated graphics from within the same address space of the process holding the data to, say, model plasma emission, rather than having to invoke a distinct process (loading the entire dataset again) to generate, say, a 3D image or surface plot from such. It is increasingly common, within astronomy for instance, for scientists to interactively analyze datasets 500 megabytes and greater in size. As dataset sizes scale even higher even the most powerful desktop workstation will starve for memory if several copies of such behemoths must be simultaneously resident in RAM.
1.1 Code Generation

Most of the over 2000 functions present in the SLgtk module are created automatically by inspection of the header files of Gtk and its constituent libraries, using the bundled SLIRP code generator (documentation available online at the SLIRP website http://space.mit.edu/~mnoble/slirp/). Note that a list of functions omitted by the code generator is created during the SLgtk build process, and that the checkgtk script may also be used to determine whether or a given function has been wrapped.
Chapter 2

Some Examples

At this point you would probably like to see an example of the Hello World variety, so here is our very own howdy.sl:

```sl
import("gtk");

variable win = gtk_window_new(GTK_WINDOW_TOPLEVEL);
() = g_signal_connect(win, "destroy", &gtk_main_quit);

variable button = gtk_button_new_with_label("Howdy: Press Me to Quit!");
gtk_container_add(win,button);

() = g_signal_connect_swapped(button, "clicked", &gtk_widget_destroy, win);
gtk_widget_show_all(win);

gtk_main();
```

This script may be executed in any application which embeds an S-Lang interpreter endowed with the capacity to import() modules. The slsh shell is one such program, which if invoked on Unix/Linux (assuming you’ve installed SLgtk) as follows

```
unix% slsh ./howdy.sl
```

should raise on your display an image which looks something like:

![Image of howdy.sl example output]

The next example shows how to use the powerful Gtk font selector, and is taken directly from the SLgtk examples/fontsel.sl sample code.
static variable window = NULL;

#define display_selection (widget, fs)
{
    variable s = gtk_font_selection_dialog_get_font_name (fs);
    vmessage ("Currently Selected Font: %s\n", s);
    gtk_widget_destroy (fs);
}

#define create_fontsel (test)
{
    if (window == NULL) {

        window = gtk_font_selection_dialog_new ("Font Selection Dialog");
        gtk_window_set_position (window, GTK_WIN_POS_MOUSE);

        () = g_signal_connect (window,"destroy",&gtk_widget_destroyed,&window);
        () = g_signal_connect (gtk_font_selection_dialog_get_ok_button(window),
                                "clicked", &display_selection,window);

        test.lower = gtk_font_selection_dialog_get_cancel_button(window);
        () = g_signal_connect(test.lower,"clicked",&display_selection,window);
    }

    if (gtk_widget_visible (window))
        gtk_widget_destroy (window);
    else
        gtk_widget_show (window);
}
Note that a wealth of sample guilets are packaged within the SLgtk distribution. The **examples** directory alone contains more than 4000 lines of code, in roughly 40 working guilets (largely adaptations from testgtk.c packaged with Gtk), while the **packages** directory contains the vwhere() guilet and supporting scripts.
Chapter 3

Programming Interface

The SLgtk application programming interface (api) aims at providing a faithful mirror of the apis of the Gtk widget set and selected portions of its dependencies (e.g Gdk, GObject, GLib, and Pango). As such we do not attempt to exhaustively document here every function wrapped from these libraries, but rather will document the discrepancies between the two apis and refer the reader to the online GTK documentation http://www.gtk.org/api/.

Knowledgeable Gtk programmers should find the mapping from Gtk to SLgtk rather straightforward, and should be pleased by the absence of casting and memory management concerns, and the shorter, simpler code that results. Developers with no prior Gtk experience are encouraged to peruse the GTK tutorial http://www.gtk.org/tutorial/, as well as the SLgtk examples and packages directories mentioned above.

The remaining sections describe the major discrepancies between the SLgtk and Gtk interfaces. The CHANGELOG file within the distribution also provides valuable supplemental information.

3.1 Memory Management

Some of the most significant benefits of coding Gtk guilets in S-Lang is that the user may remain largely ignorant of explicit memory management and type casting concerns. In contrast with Gtk C programs, which expose substantial use of casting macros, casting is never required in SLgtk guilets. There should likewise be virtually zero need to explicitly unref, sink, or destroy Gtk, Gdk, or GLib objects instantiated in S-Lang scope, as SLgtk will automatically call the appropriate finalizer when the S-Lang variable goes out of scope. In other words, it is not necessary to

- gdk_cursor_destroy() a GdkCursor
- gdk_gc_destroy() a graphics context (GdkGc)
- gdk_region_destroy() a GdkRegion
- gtk_object_sink(), g_object_ref(), g_object_unref() a GtkTooltips
- g_object_unref() bitmaps, pixbufs, or pixmaps (GdkBitmap, GdkPixbuf, GdkPixmap)
• `g_object_get_data()/g_object_unref()` after `g_object_set_data()`

• `gtk_text_iter_free()` a GtkTextView iterator (GtkTextIter)

The obvious exception to this is that if your S-Lang code explicitly adds a reference to an instance then it should likewise explicitly remove the reference later.

### 3.2 Gtk `#define` Macros

Several `#define` macros for querying or manipulating the state of GtkWidget instances have been wrapped for use in S-Lang scope. Currently these, and their corresponding S-Lang name, are:

- `GTK_WIDGET_VISIBLE(widget)`  `gtk_widget_visible(widget)`
- `GTK_WIDGET_MAPPED(widget)`  `gtk_widget_mapped(widget)`
- `GTK_WIDGET_REALIZED(widget)`  `gtk_widget_realized(widget)`
- `GTK_WIDGET_TOPLEVEL(widget)`  `gtk_widget_toplevel(widget)`
- `GTK_WIDGET_STATE(widget)`  `gtk_widget_state(widget)`
- `GTK_WIDGET_IS_SENSITIVE(widget)`  `gtk_widget_is_sensitive(widget)`
- `GTK_WIDGET_SET_FLAGS(widget,flag)`  `gtk_widget_set_flags(widget,flag)`
- `GTK_WIDGET_UNSET_FLAGS(widget,flag)`  `gtk_widget_unset_flags(widget,flag)`

### 3.3 Variable Length Argument Lists

Because the S-Lang interpreter records how many arguments have been passed to a function, those SLgtk functions which wrap C functions evincing a variable number of arguments generally do not require any special termination parameter. For example, it is not necessary to:

- terminate calls to `gtk_list_store_set()` with -1
- terminate calls to `gtk_tree_view_column_new_with_attributes()` with NULL
- terminate calls to `gtk_text_buffer_create_tag()` with NULL

### 3.4 Widget Field Accessor Functions

In most cases it is good that S-Lang code is not permitted to peer within Gtk structure internals at runtime, since it enforces the encapsulation provided by the object abstraction. For those instances, though, where Gtk itself does not enforce object encapsulation (by allowing widget internals to be exposed in user code through direct structure field references, rather than object accessor functions), SLgtk provides a corresponding `_get()` wrapper function to access the field.

One example of such a function is `gtk_dialog_get_vbox()`. Another is `gtk_color_selection_dialog_get_ok_button()`, while the complete list can be found in the file `src/faccessors_ftable.c` generated during the SLgtk build.
3.5 Typing

As discussed below all Gtk object, widget, and structured types map to either structure or opaque S-Lang types. The latter should be viewed as handles to internal data of an unspecified type, created by the S-Lang memory managed type (MMT) mechanism. The reader is encouraged to read the documentation, available at the SLIRP website http://space.mit.edu/~mnoble/slirp/, to better understand how types are mapped between S-Lang and Gtk.

3.5.1 Unique S-Lang Opaque Types

At import time SLgtk introduces the following unique S-Lang types:

```plaintext
+ void_ptr
+ int_ptr
+ double_ptr
+ float_ptr
+ long_ptr
+ string_ptr
+ uint_ptr
+ short_ptr
+ ulong_ptr
+ ushort_ptr
+ uchar_ptr
+ opaque_ptr
+ GtkOpaque
  + GObject
    + GdkDrawable
    + GdkGC
  + GdkPixbuf
  + GtkWidget
    + GdkCursor
    + GdkRegion
    + GdkIconSource
    + GdkIconSet
    + GdkGC
    + GdkPixbuf
    + GtkWidget
    + GdkAdjustment
    + GdkWidget
```

Upcasts and downcasts may be safely performed between a parent type and any of its ancestors, and SLgtk will automatically cast a given opaque instance variable to the type most appropriate for a given call. Attempting a cast between siblings, or other incompatible types, will signal an error.
Note that the function `gtk_object_type()`, a wrapper for the GTK_OBJECT_TYPE() C macro, may be used to query the underlyingGtk type of the C variable encapsulated by an opaque S-Lang variable. Likewise, `gtk_object_type_name()`, which wraps the GTK_OBJECT_TYPE_NAME() C macro, may be used to query the name of underlying Gtk type.

### 3.5.2 Gtk to S-Lang Type Mappings

These new opaque types map most of the GObject and Gtk class hierarchy to S-Lang as follows, where S-Lang types are prefixed with a plus and C types are suffixed with an asterisk:

```
+ void_ptr
+ int_ptr
+ double_ptr
+ float_ptr
+ long_ptr
+ string_ptr
+ uint_ptr
+ short_ptr
+ ulong_ptr
+ ushort_ptr
+ uchar_ptr
+ opaque_ptr
+ GtkWidget
  |  GList*
  |  GClosure*
  |  GScanner*
  |  GSList*
  |  GTypePlugin*
  |  GTypeInstance*
  |  GTypeClass*
  |  GValue*
  |  PangoFontDescription*
  |  PangoContext*
  |  PangoLayout*
  |  PangoAttrList*
  |  gpointer
  |  gconstpointer
  |  GtkWidget
  |  GdkAtom
  |  GdkColormap*
  |  GdkDragContext*
  |  GdkEvent*
  |  GdkFont*
  |  GdkVisual*
  |  GParamSpec*
```
3.5. Typing

| GtkTreeModel*  
+ GObject       
| GObject*      
| GtkAccessible* 
| AtkObject*    
| GtkIconFactory* 
|GtkRcStyle*    
| GtkSettings*  
| GtkStyle*     
| GtkTextMark*  
| GTKTextTagTable* 
| GtkTreeSelection* 
| GtkWindowGroup* 
| GdkDragContext* 
| GdkImage*     
| GdkPixbufAnimation* 
| GdkPixbufFormat* 
| GdkDevice*    
| GtkAccelGroup* 
| GtkListStore* 
| GtkTreeStore* 
| GtkTextBuffer* 
| GTKTextChildAnchor* 
| GTKTextTag* 
+ GdkDrawable 
| GdkDrawable* 
| GdkWindow*    
| GdkPixmap*    
| GdkBitmap*    
+ GdkGC 
| GdkGC*        
+ GdkPixbuf 
| GdkPixbuf*    
+ GObject 
| GObject*      
+ GtkCellRenderer 
| GtkCellRenderer* 
+ GtkCellRendererPixbuf 
| GtkCellRendererPixbuf* 
+ GtkCellRendererText 
| GtkCellRendererText* 
+ GtkCellRendererToggle 
| GtkCellRendererToggle* 
+ GtkItemFactory 
| GtkItemFactory* 
+ GtkTreeViewColumn 
| GtkTreeViewColumn* 
+ GtkTooltips 
| GtkTooltips* 
+ GtkAdjustment 
| GtkAdjustment*
Chapter 3. Programming Interface

+ GtkWidget
  | GtkWidget*
  | GtkAccelLabel*
  | GtkAlignment*
  | GtkArrow*
  | GtkAspectFrame*
  | GtkBin*
  | GtkBox*
  | GtkWidget*
  | GtkButtonBox*
  | GtkCalendar*
  | GtkCheckButton*
  | GtkCheckMenuItem*
  | GtkColorSelection*
  | GtkColorSelectionDialog*
  | GtkCombo*
  | GtkWidget*
  | GtkContainer*
  | GtkWidget*
  | GtkCurve*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  | GtkWidget*
  |GtkWidget
| GtkPacker* |
| GtkPaned* |
| GtkPlug* |
| GtkPreview* |
| GtkProgress* |
| GtkProgressBar* |
| GtkRadioButton* |
| GtkToggleButton* |
| GtkIconSource* |
| GtkIconSet* |
| GtkTreeIter* |
| GdkCursor* |
| GdkCursor* |
| GdkRegion* |
| GdkRegion* |
| GtkIconSource* |
| GtkIconSet* |
| GtkTreeIter* |
| GtkTreeIter* |
| GtkTreePath* |
3.5.3 GTypes and Properties

Most of dynamic typing system provided by GLib is not wrapped, so users cannot presently register S-Lang-scoped types as GTypes. There should be very little call for this in S-Lang scripts, but should the need arise the developer would be better off defining such types in C scope and upwardly promoting the registered GType instances to S-Lang scope, rather than trying to map a pure S-Lang type downward to a C-scoped GType.

The batch widget construction and property setting mechanisms, which accept variable length, variable-typed argument lists (e.g., `gtk_widget_new()` or `g_object_set()`), are likewise unsupported.

3.5.4 GdkColor

The S-Lang version of `gdk_color_parse()` is cleaner, simpler to use, and more powerful than its C counterpart. Color parsing and allocation are achieved in one step, with no variable references needed. Compare

```s-lang
variable red = gdk_color_parse("red");
```

in S-Lang with the comparable code in C

```c
GdkColor red;
gdk_color_parse("red", &color);
gdk_color_alloc(gdk_colormap_get_system(), &color);
```

Also, since it returns a GdkColor structure instead of a gboolean, the SLgtk wrapper for `gdk_color_parse()` will return NULL when it cannot either parse or allocate the color requested.

The `gtk_color_selection_get_current_color()` routine has been likewise modified to take only 1 argument and return its GdkColor result on the stack (or NULL on failure).

The `gdk_color_copy()` and `gdk_color_free()` routines are intentionally not wrapped. The need for the latter is questionable, while the functionality of the former can be achieved with the S-Lang `@` operator:

```s-lang
variable red = gdk_color_parse("red");
variable red_copy = @red;
```

Other SLgtk wrappers of C routines which accept a GdkColor* likewise do not require that a GdkColor reference be passed. For example, consider the C implementation of `gtk_widget_modify_fg()`, called idiomatically as

```c
GdkColor color;
...
gdk_color_parse("red", &color);
...
gtk_widget_modify_fg(..., &color);
```

The matching SLgtk idiom is
variable color = gdk_color_parse("red");
...
gtk_widget_modify_fg(...,color);

### 3.5.5 GdkRectangle

GdkRectangle structures may be instantiated in S-Lang scope by either calling the function

```c
rect1 = gdk_rectangle_new(x,y,width,height);
```

or using the S-Lang dereference operator on an existing GdkRectangle instance:

```c
rect2 = @rect1;
```

Note that `gdk_rectangle_new()` does not exist in Gdk proper, but rather is provided by SLgtk as a convenience.

### 3.5.6 GtkAllocation

The GtkAllocation structure is simply a GdkRectangle typedef. Thus the GtkAllocation type is mirrored in S-Lang scope as a proper structure.

### 3.5.7 GdkPoint

Likewise, GdkPoint structures may be instantiated in S-Lang scope either by calling the convenience function

```c
p1 = gdk_point_new(x,y);
```

or using the S-Lang dereference operator on an existing GdkPoint instance:

```c
p2 = @p1;
```

### 3.5.8 GdkEvent

Each S-Lang GdkEvent structure will reflect the three core GdkEventAny fields: `type`, `window`, and `send_event`. Additionally, motion notify and button events contain integral `x` and `y` coordinate fields, while keypress events also contain a `keyval` field, and expose events will reflect the `area` field.

### 3.5.9 GtkTextIter

The family of GtkTextBuffer routines which return a GtkTextIter do so by putting the result on the stack, instead of using a reference parameter as in the C api. This fosters natural S-Lang usages, such as

```c
variable iter = gtk_text_buffer_get_iter_at_offset(buffer,0);
```
which are cleaner and simpler than their C equivalents:

```c
GtkTextIter iter;
gtk_text_buffer_get_iter_at_offset(buffer,&iter,0);
```

### 3.5.10 GError

Unlike in C, it is not necessary to explicitly NULL out GError variables intended to receive a GError reference upon return from a function invocation. Such variables are guaranteed to have a legal S-Lang value (possibly NULL) upon return.

### 3.5.11 GtkMenu

Note that the Gtk menu popup function has been simplified in SLgtk from

```c
gtk_menu_popup(menu, parent_menu_shell, parent_menu_item, 
    menu_position_func, func_data, mouse_button_num, 
    activate_time);
```

to

```c
gtk_menu_popup(menu, mouse_button_num, activate_time);
```

Per the Gtk documentation, the parent menu and position function parameters are typically unused.

### 3.6 Signal Handlers and Callbacks

Gtk and GObject functions which install timeout or signal handlers – such as `g_signal_connect()`, `gtk_timeout_add()`, or `gtk_idle_add()` – return a positive integer uniquely identifying the registered callback or, though this does not appear to be stated explicitly in the Gtk reference, zero upon error.

The core signal connection mechanisms, `g_signal_connect()` and `g_signal_connect_swapped()` are considerably more flexible than their C counterparts, in that both transparently construct GObject closures when called. When registering signal handlers the caller may thus specify as many callback data parameters as desired, including zero. The same holds true for callback functions registered via `gtk_idle_add`, `gtk_quit_add`, `gtk_timeout_add`, and `gtk_container_foreach`.

### 3.7 Callback Errors

If an S-Lang callback function – or other S-Lang function(s) called from within an S-Lang callback – signals a non-recoverable error (as indicated by the C-scoped SLang_Error variable having a value less than zero), then the top level window will be destroyed and the outermost main loop will be terminated via `gtk_main_quit()`. In the typical case, since many scripted GUls will not nest `gtk_main()` loops, this will cause the GUI to terminate entirely.
If such functions signal a recoverable error, then the SLang interpreter will be restarted and execution resumed normally, possibly with detritus left on the stack.
Chapter 4

Data Exploration With VWhere

A graphical version of the S-Lang where command, providing a simple yet powerful mechanism for visually exploring and filtering datasets.

USAGE

Array_Type = vwhere( structure | 1Darray, 1Darray, ...)

Input to vwhere should contain at least 2 numeric vectors, all of equal length. These vectors should be passed in the form of either a comma-separated list of 1-D arrays or a single structure containing two or more fields. Structure fields will be ignored if they are non-numeric in type, or of unequal length, or have names prefixed with an underscore.

PLOTTING

Filtering in vwhere amounts to manipulating regions of interest on plots generated from the input dataset. The number of plots that may be created, and the number of region filters applied to each, is effectively unlimited.

Plots are generated from the axis expression window, which contains two editable text fields (one for each of the X and Y axes) whose contents respectively default to the names of the first and second input vectors. In general each axis expression may contain any valid S-Lang statement, even calls to C or FORTRAN functions imported from external modules. The only constraints upon an axis expression are that it be less than 256 characters long, and (when evaluated) generates a numeric vector equal in length to that of the input dataset.

Plots may also be deleted, panned, and zoomed -- providing a rapid means of data exploration -- as well as customized through a number of preferences.

In contrast with command line tools, users may find visually interactive construction of complex, multi-dimensional filters to be more intuitive,
as well as faster, cleaner, and more powerful. Filtering may be performed
upon either in-memory arrays or disk files, with no filter syntax required,
multiple iterations through disk files are avoided, no file litter is
created while one experiments with filter ranges, filters may be applied
incrementally (instead of all at once), as well as combined with arbitrary
S-Lang, C, or FORTRAN numerical functions.

REGION FILTERS

The following region filters may be applied after visualizing a plot:

- **rectangle**: click MouseButton1, then drag mouse to
define bounding box

- **ellipse**: same as rectangle

- **polygon**: click MouseButton1 to add vertices
click MouseButton2 to close polygon
click MouseButton3 to cancel

Filters may be deleted (by hitting the BACKSPACE or DELETE key),
moved, or resized after initial placement.

Points on or within the boundary of at least one region filter are
considered "selected," and will be drawn in the foreground line
style and symbol color. All other points are considered "filtered"
or "excluded," and will be drawn, when requested, in the background
line style and color. Line styles and symbol colors may be adjusted
from within the preferences dialogs.

INCREMENTAL FILTERING

One of the more powerful features of vwhere is the incremental manner in
which the dataset may be filtered. In contrast with file in / file out
filtering method offered by command line tools, which applies the entire
set of filters to the entire input dataset -- conceptually in just one
pass -- vwhere provides the option of filtering some axes of the dataset,
by applying region filters to currently displayed plots, prior to
filtering other axes.

This provides a powerful mechanism for exploring relationships within
your data, and can also speed up subsequent plotting and filtering.
When incremental filtering is on (the default) only points selected by
the current filters will be colored in subsequent plots. Filtered points
will either be drawn grayed out on subsequent plots (the default) or not
drawn at all (a faster option for large datasets), per the current
preferences. The next section describes how filters are incrementally
combined.

RETURN VALUE
The `vwhere` guilet return value matches that of the native S-Lang `where` function: an array of numbers, each representing an index into the vector(s) given to the comparison operator(s) of the `where` expression. These indices may then be applied to related datasets, or used to create filtered output files, etcetera.

Filters applied to a single plot are unioned to form the set of points selected by that plot. If only one plot is visualized then this set completely specifies the indices returned by `vwhere`. When multiple plots are visualized the incremental selections from each are either intersected (the default) or unioned (when chosen in the preferences dialog) to generate the aggregate set of selected points.

If zero region filters have been applied the entire input dataset will be returned. Dismissing the guilet by any means other than pressing "Done" in the plots window will return the empty dataset.

EXAMPLES

The following explores the curves $y = x^2$ and $z = x^3$ over $[1,100]$:

```python
result = vwhere([1:100], [1:100]^2, [1:100]^3);
```

The following explores a hypothetical binary table read from disk:

```python
tab = your_favorite_FITS_file_reader("table.fits");
result = vwhere(tab);
```

If the `tab` structure contained `CCD_ID`, `PHA`, and `TIME` fields, then valid expressions by which two plots could generated from this table might be:

PLOT 1:

```text
X : ccd_id
Y : pha
```

PLOT 2:

```text
X : time
Y : log10(pha)
```

SEE ALSO

`where`
Chapter 5

Printing

5.1 _print_dialog

Void_Type _print_dialog(print_context, callback [, cbarg1, ...])

This function will create a modal dialog which your guilet can use to query the user for common printing options. The print context argument can either be NULL (in which case a default context will be generated) or an instance of the GtkPrintContext structure, with fields

- **title**: text to display in dialog window title bar (default: "Print")
- **dest**: destination of generated output: 0 = printer (default), 1 = file
- **cmd**: operating system command used to generate a print job (default: "lpr")
- **file**: an output file name (default: "slgtk.ps")
- **orient**: page orientation: 0 = portrait (default), 1 = landscape
- **size**: page size: 0 = letter (default), 1 = legal, 2 = A4

The second argument is a reference to a function that will be called when the user presses the Print dialog button. This function should be defined to receive at least one argument (the print context, updated to reflect any input given within the dialog), and optionally any callback arguments specified when the dialog was invoked. It is important to understand that this callback, rather than the print dialog itself, is responsible for output generation.

The following S-Lang snippet gives a mock usage of the print dialog, and intentionally avoids initializing several print context fields in order to show that the dialog gives them suitable default values.

```s-lang
define print_cb(context, callback_message)
{

  vmmessage(callback_message); % sample callback arg usage

  variable file;
  if (context.dest == 0) % generate output to printer
    file = "temp.ps"; % mock temp file name
  else

  ...
```

23
file = context.file; % else dump output to user-
% specified filename

% the print dialog uses a callback function to provide the flexibility
% of calling an application-specific output rendering method, such as
% the following mocked-up "export_postscript" function

%() = export_postscript(file);

if (context.dest == 0) {
    () = system( sprintf("%s %s",context.cmd, file));
    () = remove(file);
}

define launch_print_dialog()
{
    variable context = @GtkPrintContext;
    context.title = "MyApp Print Dialog";
    context.file = "MyApp.ps";
    _print_dialog(context, &print_cb, "Hello, from MyApp print callback!");
}

5.2 _print_context_new()

GtkPrintContext = _print_context_new()

This function will generate a GtkPrintContext structure with default values for all fields.
Chapter 6

Interactive Plotting

SLGTK also wraps much of the functionality of the

- GtkPlot
- GtkPlotCanvas
- GtkPlotData

widgets from the GtkExtra widget set, with the aim of providing a lightweight mechanism for interactive 2D plotting and data exploration, with support for PostScript hardcopy. To simplify use of these widgets SLGTK introduces the GtkPlotDescriptor S-Lang type, a structured container, which can be created and manipulated by the functions described below. Examples of their use may be gleaned from the vwhere guilet described above.

6.0.1 _gtk_plot

    GtkPlotDescriptor = _gtk_plot(x, y [,color [, style]])

This function will create a plot canvas, draw upon it an 2D scatterplot from the given vectors (optionally in the given line color and style), and return an S-Lang structure containing both. To visualize the result, add the canvas field of the returned structure to a suitable container.

6.0.2 _gtk_oplot

    _gtk_oplot(GtkPlotDescriptor, x, y, [, color [, style]])

This function will overplot the given vectors (optionally in the given line color and style) upon the canvas contained within the given plot descriptor.
6.0.3 \_gtk\_plot\_set\_x\_range, \_gtk\_plot\_set\_y\_range

\_gtk\_plot\_set\_xrange(GtkPlotDescriptor, xmin, xmax)
\_gtk\_plot\_set\_yrange(GtkPlotDescriptor, ymin, ymax)

These functions customize the tick marks of the bounding box in which the plots are displayed.

6.0.4 \_gtk\_plot\_redraw

\_gtk\_plot\_redraw(GtkPlotDescriptor)

This is a convenience function which repaints the internal canvas pixmap and then refreshes the current display with the new pixmap. Note that plots containing many points may require several seconds to perform this combined action. If multiple canvas operations are to be performed – each of which logically induce a repaint – a better choice may be to avoid this function altogether, and instead freeze the canvas, perform the necessary operations, thaw the canvas, and then paint the canvas or refresh the display explicitly.

6.0.5 \_gtk\_plot\_remove

\_gtk\_plot\_remove(GtkPlotDescriptor, i)

Eliminate the \textit{i}th plot from the given descriptor, causing it to be erased from the display upon the next redraw. The index \textit{i} may take the values 1 through \textit{N}, where \textit{N} is the number of plots that have been drawn to the descriptor.
Chapter 7

Miscellaneous

SLgtk provides additional S-Lang functions and variables to simplify common case use of selected
Gtk widgets. The names of some of these are prefixed with underscores, usually to distinguish them
from wrappers for Gtk functions proper or denote a their higher susceptibility to revision in a future
release.

7.1 Default GdkColor Variables

As a convenience SLgtk instantiates several GdkColor variables, for colors expected to be commonly
used. These are

- gdk_red
- gdk_green
- gdk_blue
- gdk_black
- gdk_grey

and will be available in the global namespace after gtk.sl has been loaded.

7.2 _menu_new

    GtkWidget = _menu_new(String_Type labels[], callback [,cbdata1, cbdata2, ...] )

A convenience function to instantiate a GtkMenu, whose entries are labels created from the given
array of strings. The callback specification may either be NULL or a signature (a function reference,
optionally followed by one or more callback data parameters) suitable for responding to the activate
signal via g_signal_connect(). Recall that such callback functions should accept at least one
argument, even when no callback data is registered, to receive the activated menu item.
7.3 \_option\_menu\_new

\texttt{GtkWidget = \_option\_menu\_new(String\_Type labels[], Integer\_type default)}

A convenience function to instantiate a GtkOptionMenu, whose entries are labels created from the given array of strings. The active menu item will be set to the given default.

7.4 \_color\_button\_new

\texttt{GtkWidget = \_color\_button\_new(GdkColor\_structure)}

This function returns an instantiated GtkButton, which when realized will display a swatch of the given color. When the button is pressed a color selection dialog will be launched through which the color of the swatch (as well as the RGB field values within the given GdkColor structure, since S-Lang structures are passed as references) may be modified.

7.5 \_info\_window

\texttt{Void\_Type \_info\_window(title, text)}

This function will create a non-modal dialog to display the given text in a window with the given title. The dialog will be raised immediately, provided there is at least one active \texttt{gtk\_main()} loop context.

7.6 \_is\_numeric

\texttt{Integer\_Type = \_is\_numeric(Any\_Type datum)}

This function will return a one if the given datum can be interpreted as a real- or integral-valued numerical type, and zero otherwise. This function will be phased out of SLGTK proper, as it will be made available in the next S-Lang release.

7.7 \_is\_callable

\texttt{Integer\_Type = \_is\_callable(Any\_Type datum)}

This function returns either 1 or 0, to indicate whether the given datum can be dereferenced to invoke a function. This function will be phased out of SLGTK proper, as it will be made available in the next S-Lang release.
7.8  _get_slgtk_doc_string

String_Type = _get_slgtk_doc_string(topic)

This function will search the SLgtk documentation for the given topic. If the topic is found then its entire entry will be returned as a single string, otherwise an error string will be returned.

7.9  _gtk_window_destroy_with(window,parent)

Provides a one-step way to set the transient parent of a window AND have it be destroyed with that parent. It is equivalent to calling

    gtk_window_set_transient_for(window,parent);
    gtk_window_destroy_with_parent(window,TRUE);

7.10 _gdk_pixbuf_get_formats()

Array_Type = _gdk_pixbuf_get_formats()

Similar to the Gdk function gdk_pixbuf_get_formats(), in that its return value indicates the number and kind of file formats supported by the Gdk pixbuf interface, only this version returns an array of String_Type instead of a GList*.
Chapter 8

Development and Debugging Utilities

8.1 slgtksh

Included within src/Makefile of the SLGTK distribution is a target which builds slgtksh, a version of the S-Lang shell, slsh, with SLGTK statically linked. See the importify utility for more details.

8.2 importify

This script provides a simple mechanism for building the S-Lang shell, slsh with one or more portable modules statically linked.

unix% <slgtk>/admin/importify
importify, version 0.6 (mnoble@space.mit.edu)
Usage: importify [-h|--help] [-slsh path_to_slsh.c] module_name [ module_name ...]

Using modules in this manner simplifies debugging in two ways. First, since the module entry points are known at invocation, breakpoints can be set in module code, or within the library wrapped by the module, prior to launching the process. In contrast, when modules are imported into an S-Lang-enabled application at runtime (i.e., after launching the process) its entry points are made available to the debugger only after the module has been fully loaded (on UNIX systems, for example, after a successful dlopen() call), which makes setting breakpoints in the module code (or its wrapped library) much more difficult.

A second benefit of using a statically loaded module is that breakpoints set within the module persist when the parent application is restarted from within the debugger. With code imported at runtime such breakpoints are lost each time the parent application is restarted.
8.3 slirp_debug_pause

In situations where it is not desirable or possible to use slgtksh to debug your guilet, you may instead use the `slirp_debug_pause()` stub routine.

To activate this stub set the SLIRP_DEBUG_PAUSE environment variable before importing the SLgtk module. This will cause the parent process to wait a specified amount of time prior to entering the Gtk main loop. During this time you may set breakpoints within the codebase of the SLgtk module (since its symbol table will have been loaded prior to the invocation of the debugging stub), including Gtk and its constituent libraries, or within your own compiled code.

If SLIRP_DEBUG_PAUSE is unset in the environment the debugging stub will do nothing. Otherwise, if the variable evaluates to a negative integer \( N \) `slirp_debug_pause()` will sleep for \( \text{abs}(N) \) seconds. If SLIRP_DEBUG_PAUSE is set to any other value the stub will pause indefinitely, awaiting a keypress in the window from which the parent process was launched.

More information is available on the SLIRP website [http://space.mit.edu/~mnoble/slirp/](http://space.mit.edu/~mnoble/slirp/).

8.4 slgtk-demo

This script provides a convenient way to execute the demonstration code in the examples directory. The usage synopsis is:

```
slgtk-demo [--auto | guilet_name ]
```

The `--auto` option will cause the script to cycle through raising and lowering the entire suite of example guilets, and serves as a useful way to exercise the module at large. When invoked with the name of a specific guilet the script will launch the parent guilet and additionally raise the named example. When invoked with no options only the parent guilet will be raised.

8.5 checkgtk

This script allows the developer to quickly identify whether or not a given Gtk/Gdk/Pango/etc C function is available within the SLgtk module.

```
unix% <slgtk>/src/checkgtk
Usage: checkgtk gtk_function_name [ gtk_function_name ...]
```
Chapter 9

Versioning

SLgtk indicates version information in the following variables:

- _slgtk_major_version
- _slgtk_minor_version
- _slgtk_micro_version

To conform with the nascent S-Lang modules development standard SLgtk also provides:

- _slgtk_version An integer containing the value 
  (10000*major_ver) + (100*minor_ver) + micro_ver
- _slgtk_version_string A string containing the value 
  major_ver.minor_ver.micro_ver

Finally, as a user convenience SLgtk also indicates the version of Gtk against which it was built:

- _gtk_major_version
- _gtk_micro_version
- _gtk_minor_version