Reference Manual for SLgtk Version 0.7.6

Michael S. Noble, mnoble@space.mit.edu Jul 21, 2010

Preface

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

Contents

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.

1.2 Non-interactive Execution

Given their role as graphical toolkits Gtk and SLgtk will most frequently be used interactively, e.g. from an X windows session. Sometimes, however, one might like to utilize them within a windowless environment, such as a cron job or remotely-executed regression test. Towards that end SLgtk bundles the xvfb-run script, which is described in chapter 9 (Utilities).

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`:

```
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:

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 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

Two of the most significant benefits of coding Gtk guilets in S-Lang are 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* , 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:

```
+  void_ptr
   +  int_ptr
   +  double_ptr
   +  opaque_ptr
   +  file_ptr
   +  float_ptr
   +  long_ptr
   +  string_ptr
   +  uint_ptr
   +  short_ptr
   +  ulong_ptr
   +  ushort_ptr
   +  uchar_ptr
   +  GtkOpaque
      +  GObject
         +  GdkDrawable
            +  GdkPixmap
            +  GdkBitmap
         +  GdkGC
         +  GdkPixbuf
         +  GdkPixbufAnimation
            +  GdkPixbufSimpleAnim
         +  GdkPixbufAnimationIter
         +  GtkObject
            +  GtkCellRenderer
               +  GtkCellRendererPixbuf
               +  GtkCellRendererText
               +  GtkCellRendererToggle
            +  GtkItemFactory
            +  GtkTreeViewColumn
            +  GtkTooltips
            +  GtkAdjustment
            +  GtkWidget
      +  GdkCursor
      +  GdkRegion
      +  GtkIconSource
      +  GtkIconSet
      +  GtkTreeIter
```

```
      + GtkTextIter
      + GtkTreePath
```

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 underlying Gtk 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
   + opaque_ptr
   + file_ptr
   + float_ptr
   + long_ptr
   + string_ptr
   + uint_ptr
   + short_ptr
   + ulong_ptr
   + ushort_ptr
   + uchar_ptr
   + GtkOpaque
      | GList*
      | GClosure*
      | GScanner*
      | GSList*
      | GTypePlugin*
      | GTypeInstance*
      | GTypeClass*
      | GValue*
      | PangoFontDescription*
      | PangoContext*
      | PangoLayout*
      | PangoAttrList*
      | gpointer
      | gconstpointer
      | GtkArg*
      | GtkRequisition*
      | GtkSelectionData*
```

```
|  GdkAtom
|  GdkColormap*
|  GdkDragContext*
|  GdkEvent*
|  GdkFont*
|  GdkDisplay*
|  GdkVisual*
|  GdkScreen*
|  GParamSpec*
|  GtkTreeModel*
|  cairo_t*
|  cairo_surface_t*
|  cairo_content_t*
+  GObject
   |  GObject*
   |  GtkAccessible*
   |  AtkObject*
   |  GtkIconFactory*
   |  GtkRcStyle*
   |  GtkAction*
   |  GtkActionGroup*
   |  GtkSettings*
   |  GtkStyle*
   |  GtkTextMark*
   |  GtkTextTagTable*
   |  GtkTreeSelection*
   |  GtkWindowGroup*
   |  GdkDragContext*
   |  GdkImage*
   |  GdkPixbufFormat*
   |  GdkDevice*
   |  GtkAccelGroup*
   |  GtkListStore*
   |  GtkTreeStore*
   |  GtkTextBuffer*
   |  GtkTextChildAnchor*
   |  GtkTextTag*
   |  GtkUIManager*
   +  GdkDrawable
      |  GdkDrawable*
      |  GdkWindow*
      +  GdkPixmap
         |  GdkPixmap*
      +  GdkBitmap
         |  GdkBitmap*
   +  GdkGC
      |  GdkGC*
   +  GdkPixbuf
      |  GdkPixbuf*
   +  GdkPixbufAnimation
      |  GdkPixbufAnimation*
```

```
   +  GdkPixbufSimpleAnim
      |  GdkPixbufSimpleAnim*
+  GdkPixbufAnimationIter
   |  GdkPixbufAnimationIter*
+  GtkObject
   |  GtkObject*
   +  GtkCellRenderer
      |  GtkCellRenderer*
      +  GtkCellRendererPixbuf
         |  GtkCellRendererPixbuf*
      +  GtkCellRendererText
         |  GtkCellRendererText*
      +  GtkCellRendererToggle
         |  GtkCellRendererToggle*
   +  GtkItemFactory
      |  GtkItemFactory*
   +  GtkTreeViewColumn
      |  GtkTreeViewColumn*
   +  GtkTooltips
      |  GtkTooltips*
   +  GtkAdjustment
      |  GtkAdjustment*
   +  GtkWidget
      |  GtkWidget*
      |  GtkAccelLabel*
      |  GtkAlignment*
      |  GtkArrow*
      |  GtkAspectFrame*
      |  GtkBin*
      |  GtkBox*
      |  GtkButton*
      |  GtkButtonBox*
      |  GtkCalendar*
      |  GtkCheckButton*
      |  GtkCheckMenuItem*
      |  GtkColorSelection*
      |  GtkColorSelectionDialog*
      |  GtkCombo*
      |  GtkComboBox*
      |  GtkContainer*
      |  GtkCurve*
      |  GtkData*
      |  GtkDialog*
      |  GtkDrawingArea*
      |  GtkEditable*
      |  GtkEntry*
      |  GtkEventBox*
      |  GtkExpander*
      |  GtkFileSelection*
      |  GtkFixed*
      |  GtkFontSelection*
```

```
| GtkFontSelectionDialog*
| GtkFrame*
| GtkGammaCurve*
| GtkHandleBox*
| GtkHBox*
| GtkHButtonBox*
| GtkHPaned*
| GtkHRuler*
| GtkHScale*
| GtkHScrollbar*
| GtkHSeparator*
| GtkImage*
| GtkInputDialog*
| GtkInvisible*
| GtkItem*
| GtkLabel*
| GtkLayout*
| GtkMenu*
| GtkMenuBar*
| GtkMenuItem*
| GtkMenuShell*
| GtkMisc*
| GtkNotebook*
| GtkOptionMenu*
| GtkPacker*
| GtkPaned*
| GtkPlug*
| GtkPreview*
| GtkProgress*
| GtkProgressBar*
| GtkRadioButton*
| GtkRadioMenuItem*
| GtkImageMenuItem*
| GtkRange*
| GtkRuler*
| GtkScale*
| GtkScrollbar*
| GtkScrolledWindow*
| GtkSeparator*
| GtkSocket*
| GtkSpinButton*
| GtkStatusbar*
| GtkTable*
| GtkTearoffMenuItem*
| GtkTextView*
| GtkTipsQuery*
| GtkToggleButton*
| GtkToolbar*
| GtkTreeView*
| GtkVBox*
| GtkVButtonBox*
```

```
            |  GtkViewport*
            |  GtkVPaned*
            |  GtkVRuler*
            |  GtkVScale*
            |  GtkVScrollbar*
            |  GtkVSeparator*
            |  GtkWindow*
+  GdkCursor
   |  GdkCursor*
+  GdkRegion
   |  GdkRegion*
+  GtkIconSource
   |  GtkIconSource*
+  GtkIconSet
   |  GtkIconSet*
+  GtkTreeIter
   |  GtkTreeIter*
+  GtkTextIter
   |  GtkTextIter*
+  GtkTreePath
   |  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

```
variable red = gdk_color_parse("red");
```

in S-Lang with the comparable code in 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:

```
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

```
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

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

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

```
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

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

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

```
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

```
variable iter = gtk_text_buffer_get_iter_at_offset(buffer,0);
```

which are cleaner and simpler than their C equivalents:

```
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

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

to

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

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

3.5.12 Screen Depths

`gdk_query_depths(int **depths, int *num)` has been wrapped so that it may be called from S-Lang scope with zero arguments, with the depths array returned on the stack:

```
Integer_Type[]  gdk_query_depths()
```

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 GUIs 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

VWhere: N-Dimensional Data Mining

4.1 vwhere

Synopsis

A graphical, interactive version of the S-Lang where function

Usage

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

Description

VWhere provides an easy to understand yet capable mechanism for exploring and filtering multidimensional datasets. Using a visual, interactive approach to the construction of complex, multi-dimensional filters, VWhere offers a fluid and intuitive alternative to the classic approach of file-based filtering with command line tools (as used, e.g., within astronomy data analysis), and can be considerably faster, cleaner, and more powerful. Filtering is performed upon data vectors generated either in-memory or from disk files, with no filter syntax required and the result instantly visualized for inspection. In contrast, file-based filtering tools require explicit syntax (often conflicting with the syntax employed by other tools or systems) and that the resulting file(s) be re-loaded into separate programs for verification (e.g. a plot or file dump). In contrast with the all-at- once style mandated by file-based filtering, VWhere filters may be applied incrementally (or not) to arbitrary axes of your input dataset. This avoids the creation of numerous "file litter" products while one experiments with filter ranges or axis combinations, as well as the performance penalties of multiple I/O iterations over files. Moreover, most file-based filtering tools are static in function: they cannot be augmented at runtime by dynamic loading of modules. VWhere filters, however, may employ not only any built-in S-Lang arithmetic operator or function, but also essentially arbitrary C, C++, or FORTRAN codes loaded from external modules.

Input to vwhere should contain at least 2 numeric vectors. Vectors may be passed in the form of a comma-separated list of 1-D arrays or as a single structure containing two or more fields. All vectors must have the same length. Vectors will be ignored if they do not match the length

of the first vector, or are non-numeric in type, or have names prefixed with an underscore. When using the comma-separated list form it can be useful to prefix the name of each vector with "&" (the S-Lang reference operator). This lets VWhere determine the vector name and reflect it in the Axis Expression window, instead of assigning it a less-meaningful name such as "array1" (see examples below).

Upon invocation VWhere launches its Axis Expression Window, which provides a means of generating plots, fabricating new data vectors, and issuing arbitrary commands through an interactive S-Lang prompt.

PLOTTING

Filtering in VWhere amounts to manipulating regions of interest on plots. The number of plots that may be created or overplotted, and the number of region filters applied to each, is effectively unlimited. Plots may also be deleted, panned, and zoomed – providing a rapid means of data exploration – as well as customized through a number of graphical user interface preferences.

Plots are specified in the Axis Expression Window via two editable text fields, one for each of the X and Y axes. The content of each field defaults to the name of the first and second input vectors, respectively, and may be changed either by typing new expressions for each axis or by selecting from the Choose dropdown menu. In general each axis expression may contain any valid S-Lang statement, even calls to C, C++, or FORTRAN functions imported from external modules. The chief constraints upon an axis expression are that it be less than 256 characters long and that it generate a numeric vector.

Two kinds of plots may be visualized, filter plots and overplots, by pressing either the Plot or OPlot buttons. The main distinction between the two is that overplotted X/Y vector pairs may beof arbitrary length, while filter plots require vectors exactly equal in length to those within the input dataset. The latter constraint stems from the fact that, logically, array expressions given to the underlying S-Lang where command [e.g. where(A $<$ 5 and B $>$ 11)] can operate only upon isomorphic vectors. In addition, because overplotted vectors are not subject to where filtering they are always drawn in their entirety; thus they provide additional means for qualitative, visual comparison, but have no quantitative effect on the result returned by VWhere. Finally, when a filter plot is created each unique axis expression – and the resulting vector that it generates – is "remembered" in the Choose dropdown menu. This provides for easy re-selection later, and is a fast and simple mechanism for fabricating new data on the fly, of essentially arbitrary complexity, thanks to the extensibility and generality of axis expressions – on the fly.

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. By default, regions perform INCLUDE filtering: points within a region are kept and points outside it will be discarded. Alternatively, a region may be used for EXCLUDE filtering by pressing the 'e' key while it is initially being laid; the region will be marked with a diagonal slash, and points within it will be discarded during subsequent filtering while points outside it will be kept. A region may be toggled between the INCLUDE/EXCLUDE state by pressing 'e' while it is selected.

Points included during filtering are considered "selected," and will be drawn in the foreground line style and symbol color; excluded points are considered "filtered" 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 useful 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.

Example

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

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

The next call

```
result = vwhere(&x, &y, &z);
```

is identical, but because the arrays are passed as references VWhere can determine the name of each vector and reflect them in the Axis Expression window as "x," "y," and "z," instead of fabricating the names "array1," "array2," and "array3".

The following explores a hypothetical binary table read from disk:

```
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:
                X :       ccd_id
                Y :       pha
PLOT 2:
                X :       time
                Y :       log10(pha)
```

The log10(pha) expression for the Y axis of the second plot creates a new data vector, which will also be selectable from the Choose dropdown menu for use in subsequent plots.

Notes

The GtkPlot widget atop which VWhere is built is not robust in the face of Inf/Nan values. VWhere attempts to compensate for this, but for performance reasons does not execute both isnan() and isinf() on all X/Y plot vectors. To avoid undefined behavior and potential data loss, Inf/Nan values should thus be culled first.

See Also

http://arxiv.org/abs/astro-ph/0412003 (ADASS XIV Proceedings)

Chapter 5

Image Display

5.1 imdisplay

Synopsis

Render a stack of still images or display an animation

Usage

```
imdisplay( FileName_or_ImageArray_or_Option [, ...])
```

Description

This function accepts an arbitrary number of images as input, and by default renders them stacked upon one another, respecting transparency. The composite image may be easily tiled or scaled, as well as auto- matically scrolled for images too large to fit reasonably within your display. Pressing any button while the mouse is positioned over the image will show the color value for that pixel, in either RGB or RGBA format (for images with an alpha channel); an alpha value of 1.0 indicates that the pixel is fully opaque, while 0.0 indicates full transparency.

Images may also be displayed in one of several alternate modes: First, the panes option causes each input image to be displayed within its own window, tiled either vertically or horizontally; second, the 'anim' option may be used to animate a series of 2D still images, with the ability to pause and step forward/backward between frames; finally, 'anim' can also animate 3D volumes (as series of 2D slices), although this is often not necessary because; imdisplay() will automatically animate 3D arrays of dimension [N, M, Z>4] (as they are clearly not RGBA images), as well as images specified via Array_Type[N] and List_Type[N].

By default image windows are chained to each other, and the control window, so that all may be moved onscreen, at once, simply by moving the master.

The images passed in may be names of files, GDK pixbufs, or S-Lang arrays (2D/greyscale, 3D/RGB, or 3D/RGBA, or arbitrary 3D volumes). If any input image in a composite contains an alpha channel (transparency) then the rendered result will as well.

A wide variety of input file formats are supported, including JPEG, PNG, GIF, XPM, TIFF, animations and (optionally) the FITS file format popular in astronomy. The rendered result may also be saved to a variety of formats, including JPEG, PNG, GIF (still and animated) and FITS. The range of supported formats depends upon how your Gtk distribution was compiled.

Comma-delimited options to imdisplay() may be specified either as strings within the argument list or via qualifiers separated from the argument list by a semicolon. Most options affect the composite image just prior to display, and include:

```
anim[=delay]            Animate input images into a movie.  The
                        optional frame refresh delay should be a
                        single scalar value in milliseconds, and
                        defaults to 500 (0.5 sec) if omitted.

fill=<rule>             how to fill new space created in image
                        display window when it is enlarged; one of

                           none    no fill; keep original image
                           tile    fill with consecutive image copies
                           scale   fill by enlarging image to fit

flip                    mirror the image vertically
flop                    mirror the image horizontally

save=<name>[/type]      Instead of rendering image(s) onscreen, save
                        to <filename>; a file type may be specified
                        via a /gif, /png, etc. optional qualifier
                        [see _gdk_pixbuf_get_formats()] or inferred
                        from the file extension; if no file type can
                        be determined then still images will be saved
                        in PNG format; animations are always saved
                        in GIF format.

size=<geometry>         resize the image; geometry may either
                        be a scaling percentage or an absolute
                        pixel size, such as 150x200% or 100x300;
                        when only one value is provided it will
                        be applied to both axes of the image

scale=<geometry>        synonym for size= option.

panes=<layout>          how to display multiple images; layout may be

                        one | single    composite all images into
                                        one window (the default)
                        horiz[ontal]    tile images horizontally
                        vert[ical]      tile images vertically

                        No compositing is performed for horizontal
                        or vertical tiling.

horiz[ontal]            shorthand for panes=horiz[ontal]
vert[ical]              shorthand for panes=vert[ical]
```

The default display options may be changed with imdisplay_defaults(); the new defaults will persist between subsequent invocations of imdisplay() (in the same process) or until imdis-

play_defaults() is called with no arguments.

At launch all of the windows created by imdisplay will be chained: e.g. the control window will be chained to the window of the last image loaded, meaning that the controller will follow the image window around onscreen when the latter is moved; when the controller is moved its new gravity (i.e. placement relative to the image) will be remembered. An entire vertical (or horizontal) tiling of windows may be moved simply by moving its top- (or left-) most window.

To disable chaining, ensure that the slave window you wish to unchain has focus, then hold down the SHIFT key while moving the slave. This will disconnect the slave from its own master, but leave intact any chains in which the slave is itself a master.

Example

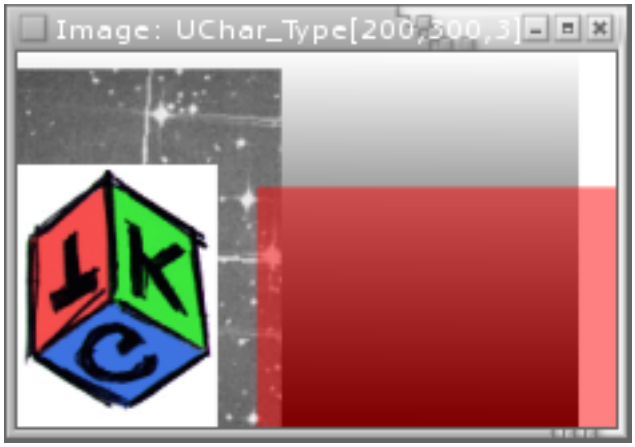
The sequence of commands

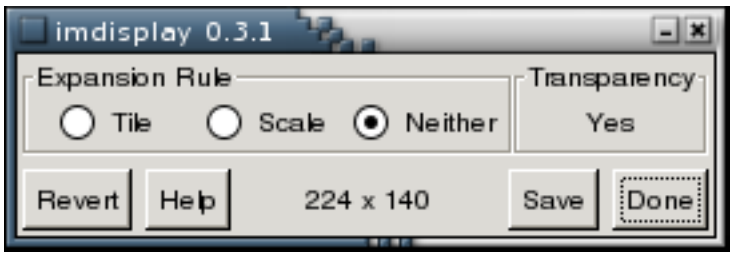
```
unix% cd <slgtk_distribution_on_your_system>/images

unix% slsh
slsh> require("imdisplay")

slsh> pb1 = gdk_pixbuf_new_from_file("stars.fits")
slsh> im1 = _reshape( [1:200*300], [200,300])
slsh>  imdisplay(im1, pb1, "gtk-logo-rgb.gif", "flip,size=70%", "red.png")
```

should yield the visual

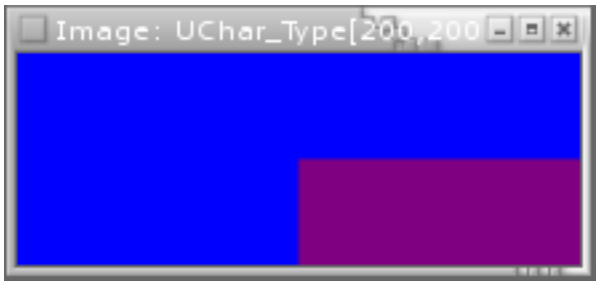


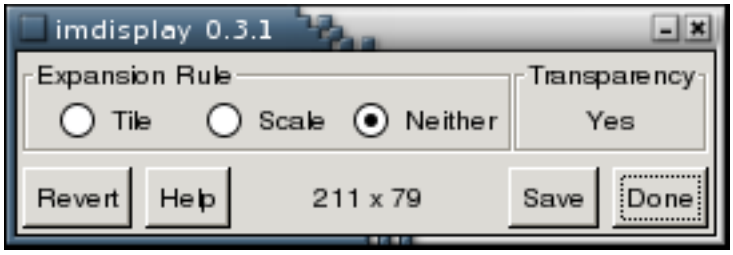


Similarly,

```
slsh> red = UChar_Type[100,100,4]
slsh> red[*,*,0] = 255
slsh> red[*,*,3] = 127                    % 50% transparency
slsh> blue = UChar_Type[200,200,3]
slsh> blue[*,*,2] = 255
slsh> imdisplay("size=211x79", blue, "flip", red, "flop")
```

should yield





Notes

imdisplay may also be used from the OS prompt as an executable script. In this form only files may be loaded, but all other options are supported, including the "help" switch which provides detailed usage information.

Chapter 6

Printing

6.1 _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 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.

```
define print_cb(context, callback_message)
{

   vmessage(callback_message);                % sample callback arg usage

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

```
        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!");
}
```

6.2 _print_context_new()

```
        GtkPrintContext =  _print_context_new()
```

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

Chapter 7

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 2D plotting and data exploration, with support for PostScript hardcopy. While these functions may be used interactively, as is, they are probably more suitable as building blocks for higher-level plotting functions. Towards that end 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.

7.1 _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.

A custom canvas size may be specified via the `width=` and `height=` qualifiers, in units of pixels.

7.2 _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. A custom canvas size may be specified via the `width=` and `height=` qualifiers, in units of pixels.

7.3 _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.

7.4 _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.

7.5 _gtk_plot_remove

```
_gtk_plot_remove(GtkPlotDescriptor, i)
```

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

Chapter 8

Miscellaneous

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

8.1 GdkColor Variables

As a convenience SLgtk pre-allocates a number of GdkColor structures, for colors expected to be commonly used. They are named

- `gdk_red`
- `gdk_green`
- `gdk_blue`
- `gdk_black`
- `gdk_grey`
- `gdk_white`

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

8.2 _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. Any label which, after being lowercased, matches "<separator>" will cause a horizontal separator to be drawn in the menu.

8.3 _option_menu_new

```
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.

8.4 _color_button_new

```
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.

8.5 _info_window

```
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 `gtk_main()` loop context.

8.6 _is_numeric

```
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 is now available in S-Lang 2.x.

8.7 _is_callable

```
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 is now available in S-Lang 2.x.

8.8 _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.

8.9 _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);
```

8.10 _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 `GSList*`.

Chapter 9

Utilities

9.1 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 SLIRP `importify` utility for details.

9.2 imdisplay

This is a wrapper around the imdisplay() function described above, suitable for use from the operating system command line. Input images are specified by file name, and all of the same transform options are supported.

```
unix% imdisplay
Usage: imdisplay file_or_option [ file_or_option ...]
```

9.3 importify

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

```
unix% <slgtk>/slirp/importify -h

importify, version 0.9 (mnoble@space.mit.edu)
Emits to stdout a copy of slsh.c, modified to facilitate static
loading of one or more embedded S-Lang modules.

Usage: importify [options] module_name [ module_name ...]

Options:
-h                  generate this message
-m symbol           allow main() to be overridden, by renaming it to symbol()
-p                  do not generate prototype for module init func
```

```
-q                  generate quiet code (no startup messages)
-s path_to_slsh.c   useful when slsh.c is not in $PWD
```

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.

9.4 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 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* .

9.5 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.

9.6 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 ...]
```

9.7 xvfb-run

The `xvfb-run` script supports the use of SLgtk in a display-less environment, such as a cron job or remotely-executed regression test. Intended for Unix-compatible systems running X11, the script was taken from Debian Linux and modified slightly to enhance its portability to non-Linux systems. It, along with the `mcookie` utility and `md5` algorithm from the `util-linux` package, may be found within the `src/batch` directory of the distribution.

Chapter 10

Versioning

SLgtk indicates version information in the variables:

```
_gtk_module_version          An integer containing the value
                             (10000*major_ver) + (100*minor_ver) + micro_ver

_gtk_module_version_string   A string containing the value
                             major_ver.minor_ver.micro_ver
```

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
```