chicken-gtk2

version <unreleased>

Tony Garnock-Jones

chicken-gtk2: version <unreleased>

by Tony Garnock-Jones

Published Mon May 19 21:00:15 BST 2003

Describes installation and use of the chicken-gtk2 Chicken Scheme binding to GTK+ v2.0.

Table of Contents

Chapter 1. User Guide

1.1. General Operation

Chapter 2. Extension modules

2.1. Glib/GObject 2.0 binding

```
(require 'gobject)
```

The gobject extension module provides a wrapping for a subset of the features offered by GLib version 2.0. Currently it exposes a partial API for manipulating GType, GBoxed, GEnum, GFlags, GValue, GClosure, GObject and GSignal types and values.

2.1.1. Initialization and glib miscellany

procedure: `(g-warning` *`args ...`*`)`

Delegates to the C function `g_warning` to produce a warning message using the GLib logging facility.

2.1.2. GType

GType is the GLib Runtime type identification and management system. Most of the datatypes used in GLib (and GDK/GTK+ etc) are registered with the GType system.

A certain amount of introspection over the GType system is possible. GType itself does not provide information about methods on objects, but does allow enumeration of object properties, signals, superclasses and subclasses, and also provides information on the allowable values of enumerations (GEnum) and flags (GFlags).

The combination of the following procedures and variables ought to allow access to much of the available metainformation:

```
(gtype-name t)                    ; query type name
gtype:fundamental-types           ; list of root types
(gtype-parent t)                  ; retrieve supertype
(gtype-children t)                ; retrieve subtypes
(gobject-type o)                  ; extract GType from GObject
(gobject-class-properties t)      ; list properties of class
(gobject:methods-on-class t)      ; list methods on a class
(gobject:methods-in-gf gfname)    ; list methods in a generic function
(gsignal-list t)                  ; list signals in a class
(gsignal-list-complete t)         ; list signals in a class and parents
```

Since GType does not collect the information returned by `gobject:methods-on-class` or `gobject:methods-in-gf` itself, explicit calls to `gobject:register-method!` are required to fill in the associated datastructures.

Much of the introspection API is described in the sections devoted to each major grouping of GType instances.

record: `(make-gtype` *`number`*`)`

Represents a GType instance - a representation of a type known to the GLib system. The *`number`* is the unsigned-long GType value as used in C.

procedure: `(gtype-name` *`t`*`)`

Given a gtype record, returns the name associated with the GType as a string.

procedure: `(wrap-gtype` *`num`*`)`

Wraps a GType number in a gtype record. If `num` is zero (the invalid GType), `#f` is returned.

procedure: `(gtype-from-name` *`name`*`)`

Looks up a GType by name, wrapping it in a gtype record. Returns `#f` if the type name is not found.

procedure: `(raw-gtype->fundamental` *`num`*`)`

Given a GType number (not a record!), returns the GType number of its ultimate parent type - the root of the inheritance tree for the passed-in GType.

procedure: `(gtype->fundamental` *`t`*`)`

As for `raw-gtype->fundamental`, but takes and returns a gtype record instead of a raw GType number.

procedure: `(wrap-gtype-fundamental` *`num`*`)`

Produces a gtype record from a fundamental type number, using the C macro `G_TYPE_MAKE_FUNDAMENTAL`.

procedure: `(raw-unmake-gtype-fundamental` *`num`*`)`

Converts a GType number to its raw fundamental-GType number by shifting right by `G_TYPE_FUNDAMENTAL_SHIFT`.

procedure: `(unwrap-gtype-fundamental` *`t`*`)`

As for `raw-unmake-gtype-fundamental`, but takes a record instead of a GType number.

variable: *`gtype:...`*

The *gtype:...* variables correspond to the fundamental types defined in `gtype.h` as `G_TYPE_....`

```
G_TYPE_INVALID       gtype:invalid
G_TYPE_NONE          gtype:none
G_TYPE_INTERFACE     gtype:interface
G_TYPE_CHAR          gtype:char
G_TYPE_UCHAR         gtype:uchar
G_TYPE_BOOLEAN       gtype:boolean
```

`G_TYPE_INT`	`gtype:int`
`G_TYPE_UINT`	`gtype:uint`
`G_TYPE_LONG`	`gtype:long`
`G_TYPE_ULONG`	`gtype:ulong`
`G_TYPE_INT64`	`gtype:int64`
`G_TYPE_UINT64`	`gtype:uint64`
`G_TYPE_ENUM`	`gtype:enum`
`G_TYPE_FLAGS`	`gtype:flags`
`G_TYPE_FLOAT`	`gtype:float`
`G_TYPE_DOUBLE`	`gtype:double`
`G_TYPE_STRING`	`gtype:string`
`G_TYPE_POINTER`	`gtype:pointer`
`G_TYPE_BOXED`	`gtype:boxed`
`G_TYPE_PARAM`	`gtype:param`
`G_TYPE_OBJECT`	`gtype:object`

variable: *`gtype:fundamental-types`*

Collects all the fundamental (root) types together in a list.

procedure: `(gtype-...? `*`t`*`)`

Predicates for examining attributes of GType records.

`gtype-fundamental?`	`G_TYPE_IS_FUNDAMENTAL`
`gtype-derived?`	`G_TYPE_IS_DERIVED`
`gtype-interface?`	`G_TYPE_IS_INTERFACE`
`gtype-classed?`	`G_TYPE_IS_CLASSED`
`gtype-instantiatable?`	`G_TYPE_IS_INSTANTIATABLE`
`gtype-derivable?`	`G_TYPE_IS_DERIVABLE`
`gtype-deep-derivable?`	`G_TYPE_IS_DEEP_DERIVABLE`
`gtype-abstract?`	`G_TYPE_IS_ABSTRACT`
`gtype-value-abstract?`	`G_TYPE_IS_VALUE_ABSTRACT`
`gtype-has-value-table?`	`G_TYPE_IS_HAS_VALUE_TABLE`

procedure: `(gtype-parent `*`t`*`)`

Returns the parent type of the passed-in gtype record.

procedure: `(gtype-depth `*`t`*`)`

Returns the depth in the inheritance tree of the passed-in gtype record. A fundamental (root) type has depth 1, its child types have depth 2, and so forth.

procedure: (gtype-next-base *leaf-t root-t*)

Given a leaf-t and a root-t which is contained in its ancestry, return the type that root-t is the immediate parent of. In other words, this function determines the type that is derived directly from root-t which is also a base class of leaf-t. Given a root type and a leaf type, this function can be used to determine the types and order in which the leaf type is descended from the root type[1].

procedure: (gtype-isa? *t is-a-t*)

Returns #t if t is equal to, or a subtype of, is-a-t; otherwise returns #f.

procedure: (gtype-children *t*)

Returns a list of child types of the passed-in gtype record.

procedure: (gtype-interfaces *t*)

Returns a list of the interfaces supported by the passed-in gtype record.

2.1.3. GBoxed

Boxed types are non-reference-counted, explicitly allocated, copied and freed structures. Each boxed type has a pair of associated copy and free routines, which are called automatically when pointers to GBoxed instances are put under the control of wrap-gboxed.

record: (make-gboxed *type pointer*)

Represents a wrapped instance of a GBoxed type. *type* is the gtype record that is the type of the boxed value; *pointer* is the C pointer pointing to the boxed value. Do not call make-gboxed directly - usually, wrap-gboxed is more appropriate (as it arranges for reference-counting/finalization where make-gboxed does not).

procedure: (gboxed-copy-hook *(#:optional new-value)*)

Gets (or sets, if the optional argument is supplied) the current value of the hook function called when a GBoxed instance is to be copied. The default hook is the C function g_boxed_copy. The hook function should take an unsigned long (GType) and a c-pointer, and should return a c-pointer.

procedure: (gboxed-finalizer-hook *(#:optional new-value)*)

Gets (or sets, if the optional argument is supplied) the current value of the hook function called when a GBoxed instance is to be destroyed. The default hook does nothing. The hook function should accept an unsigned long (GType) and a c-pointer.

procedure: (wrap-gboxed *type ptr (#:optional copy?)*)

If ptr is non-#f and non-NULL, calls g_boxed_copy on it, wraps it in a gboxed record, and arranges for g_boxed_free to be called on the copied pointer when the gboxed record is garbage collected. type is required to decide which copying/freeing procedures to use.

The optional copy? parameter defaults to #t: it controls whether the pointer is to be copied before being wrapped. If #f, the passed-in pointer is wrapped without being copied first. Use this only if you know what you are doing, otherwise you can introduce “double-free” problems to your program.

`copy?` does not control finalization: all records returned by `wrap-gboxed` are finalized with `g_boxed_free` when they are garbage collected, whether they were copied originally or not.

procedure: `(null-gboxed)`

Returns the GBoxed equivalent of the null pointer.

2.1.4. GEnum and GFlags

The GType API provides information about enumeration and flags types registered with the system. The associated wrappers provide convenience functions for introspection and translation between enumeration/flag nicknames and numbers.

procedure: `(genum-info` *`t`*`)`

Retrieves a list of information about the values in the enumeration GType record passed in.

procedure: `(make-genum-number->nick` *`t`*`)`

Returns a procedure that when given a number returns the associated nickname from the enumeration GType record passed in.

```
((make-genum-number->nick (gtype-from-name "GtkJustification"))
 3)
==> fill
```

procedure: `(make-genum-nick->number` *`t`*`)`

Returns a procedure that when given a symbol returns the associated number from the enumeration GType record passed in.

```
((make-genum-nick->number (gtype-from-name "GtkJustification"))
 'fill)
==> 3
```

procedure: `(gflags-info` *`t`*`)`

Retrieves a list of information about the available values in the flags GType record passed in.

procedure: `(make-gflags->number` *`t`*`)`

Returns a procedure that when given a list of symbols returns the bitwise or of the associated numbers from the flags GType record passed in.

```
((make-gflags->number (gtype-from-name "GdkWindowState"))
 '(iconified sticky))
==> 10
```

procedure: `(make-number->gflags t)`

Returns a procedure that when given a number returns the list of symbols making up that number, from the flags GType record passed in.

```
((make-number->gflags (gtype-from-name "GdkWindowState"))
 10)
==> (sticky iconified)
```

2.1.5. GValue

GValue is a subtype of GBoxed which is a polymorphic value cell - it can hold any of the fundamental types and their subclasses. The wrapper provides conversion routines between Scheme objects and GValue instances.

procedure: `(raw-gvalue-type gvalue-pointer)`

Given a C pointer to a GValue object, returns the GType number associated with the GValue.

procedure: `(gvalue-type gv)`

Given a properly boxed GValue, returns the gtype record associated with the GValue.

procedure: `(gvalue->object gv)`

Extracts a Scheme object from the passed-in boxed GValue. (Also accepts a raw pointer to a GValue object, instead of a properly boxed GValue, for internal implementation use.)

procedure: `(gvalue-empty! gv)`

Empties a boxed GValue, without altering the type associated with it.

procedure: `(make-gvalue (#:optional gtype-record))`

Returns a newly-allocated, boxed GValue, with its type set to the passed in GType record. If `gtype-record` is omitted, returns a completely blank GValue object, ready for filling in with any type (by, for instance, `gtk-tree-model-get-value`).

procedure: `(raw-gvalue-fill! gvalue-ptr scheme-object)`

Fills a pointer to a GValue object with a value taken from the passed-in Scheme object. If the type of `scheme-object` is not compatible with the type of `gvalue-ptr`, returns `#f`; if the fill operation was otherwise successful, returns `#t`.

procedure: (gvalue-fill! *gv o*)

Fills a properly boxed GValue object with the value of the passed-in scheme object, as for raw-gvalue-fill!.

procedure: (object->gvalue *t o*)

Allocates a new boxed GValue of type t using make-gvalue, fills it using gvalue-fill!, and returns it. If the fill operation failed, an error is signalled.

2.1.6. GClosure

Only basic support for GClosures is implemented, using a custom marshalling function (cg_gclosure_marshaller). Scheme functions wrapped in GClosure instances are properly collected - when the GClosure object is destroyed, a finalizer function (cg_gclosure_finalizer) causes the handle on the scheme function to be released.

GClosures are not usually manipulated explicitly in Scheme code. Usually a function like gsignal-connect (a.k.a. gtk-signal-connect) is used, which transparently manages GClosure instances.

procedure: (make-gclosure *fn*)

Wrap a scheme function in a GClosure, and return a C pointer to the new GClosure structure. See gsignal-connect.

2.1.7. GObject

GObject is the base type for all reference-counted objects in the GType hierarchy.

record: (make-gobject *pointer*)

Represents a GObject instance. *pointer* is the C pointer to the GObject instance. Do not call make-gobject directly - use wrap-gobject instead.

procedure: (gobject-type *o*)

Returns the gtype record representing the type of the passed-in GObject.

procedure: (gobject-ref-hook *(#:optional new-value)*)

Gets (or sets, if the optional argument is supplied) the current value of the hook function called when a GObject instance is to be referenced. The default hook is the C function g_object_ref. The hook function should take a c-pointer and return a c-pointer.

procedure: (gobject-finalizer-hook *(#:optional new-value)*)

Gets (or sets, if the optional argument is supplied) the current value of the hook function called when a GObject instance is to be unreferenced. The default hook does nothing. The hook function should accept a c-pointer.

procedure: (`wrap-gobject` *`p`*)

Given a C pointer to a GObject instance, calls `g_object_ref` on it, constructs a gobject record for it, and registers `g_object_unref` as the finalizer for the new record. If `p` is `#f` or the null pointer, `#f` is returned; otherwise the newly-allocated gobject record is returned.

procedure: (`null-gobject`)

Returns the GObject equivalent of the null pointer. Useful with functions like `gtk-scrolled-window-new`.

procedure: (`gobject-class-properties` *`t`*)

Returns a list of the properties supported by instances of the GObject GType record passed in.

procedure: (`gobject-class-find-property` *`t pname`*)

Returns a property specification for the named property on instances of the GObject GType record passed in, or `#f` if no property by that name is found on that class.

procedure: (`make-gobject-property-getter` *`t pname-symbol-or-string`*)

Produces a getter function for the passed-in GType and property name.

procedure: (`gobject-get-property` *`o pname`*)

Retrieves the value of the named property on the GObject instance passed in.

procedure: (`make-gobject-property-setter` *`t pname-symbol-or-string`*)

Produces a setter function for the passed-in GType and property name.

procedure: (`gobject-set-property!` *`o pname newval`*)

Updates the value of the named property on the GObject instance passed in.

record: (`make-gobject-method` *`name gf class function`*)

Represents a method associated with a GObject class. *`name`* is the name of the method; *`gf`* is the name of the generic function; *`class`* is the GType record for the class; and *`function`* is the method function itself.

procedure: (`gobject:methods-on-class` *`g`*)

Retrieve a list of all methods supported by the GObject GType passed in.

procedure: (`gobject:methods-in-gf` *`gfname`*)

Retrieve a list of all methods in the named generic function.

procedure: (`gobject:register-method!` *`classname gfname methodname function`*)

Registers a method on a particular class with the system. This procedure is called by the generated code for the GTK+ wrapper.

2.1.8. GSignal

Only a partial interface to the GSignal system is supported. In particular, there is no support for signal emission.

procedure: `(gsignal-connect` *`o sigdetail fn (#:optional after)`*`)`

(also known as `gtk-signal-connect` within the gtk module) Connects `fn` to the signal (string or symbol) `sigdetail` on GObject instance `o`. When the signal is emitted, `fn` will be called with an argument list appropriate to the particular signal. Returns a number representing the connection which can then be passed into `gsignal-handler-disconnect`.

procedure: `(gsignal-disconnect` *`o handlerid`*`)`

Given an object and a handler connection number as returned by `gsignal-connect`, disconnects the handler so it will no longer fire when the signal is emitted.

procedure: `(gsignal-lookup` *`name t`*`)`

Look up a signal in a class by name; returns zero if the signal is not found for some reason.

procedure: `(gsignal-query` *`sigid`*`)`

Returns a list containing information about the signal identified by the signal identifier number passed in.

procedure: `(gsignal-list` *`t`*`)`

Returns a list of information about the signals that can be emitted by objects of the passed-in GType record, but not signals that can be emitted by its supertypes.

procedure: `(gsignal-list-complete` *`t`*`)`

Returns a list of information about the signals that can be emitted by objects of the passed-in GType record, including the signals that can be emitted by its supertypes.

2.2. GTK+ 2.0 binding

```
(require 'gtk)
```

The gtk extension module provides a wrapping for the GTK+ GUI toolkit library, version 2.0. It depends upon the gobject extension.

2.2.1. General

Most of the functions supported by the GTK+ binding extension are automatically generated from `*.defs` files, taken from James Henstridge's pygtk GTK+ binding for Python.

The generated code is contained in internal modules which don't need to be `required` separately - they're automatically included when the gtk module is loaded. Some of the generated code is not a good fit for Chicken, so it has been overridden by hand-written code[2] in the gtk module itself.

Generated procedures usually have a name derived from the name of the C function they are wrapping: case is folded to lowercase, and underscores are replaced with hyphens, so for instance `gtk_main_quit` becomes `gtk-main-quit`.

Methods on wrapped GtkObject subclasses are registered with the introspection facilities of the gobject module with calls to `gobject:register-method!`.

procedure: `(gtk-signal-connect` *`object signal-name handler-fn`*`)`

An alias for `gsignal-connect`.

procedure: `(gtk-main)`

Pass control to the GTK+ main loop. This call does not return until the application indicates it is ready to terminate by calling `gtk-main-quit`.

procedure: `(gtk-main-iteration)`

Delegates directly to the C function `gtk_main_iteration`.

2.2.2. Timeouts, idle-handlers, and input-handlers

Input handlers are not currently supported.

procedure: `(gtk-timeout-add` *`interval thunk`*`)`

Installs a timeout-handling procedure. After `interval` milliseconds, and every `interval` thereafter, `thunk` will be called with no arguments. If `thunk` returns `#f`, the timeout-handler will not run again (it will be removed). The semantics are derived from the underlying C procedure, `gtk_timeout_add`. This function returns a gtk:timeout-handle record, which can be passed in to `gtk-timeout-remove`.

procedure: `(gtk-timeout-remove` *`handle`*`)`

Removes a previously-registered timeout handler, using a gtk:timeout-handle record returned by `gtk-timeout-add`.

procedure: `(gtk-idle-add` *`thunk`*`)`

Installs `thunk` as a GTK+ idle handler, as per the C function `gtk_idle_add`. Returns a gtk:idle-handle record, which may be used with `gtk-idle-remove`.

procedure: `(gtk-idle-remove` *`handle`*`)`

Removes a previously installed GTK+ idle handler, using the gtk:idle-handle record returned from `gtk-idle-add`.

2.2.3. GDK

procedure: `(gdk-color->list` *`c`*`)`

Return a list (R G B) of the three colour components contained in a GdkColor structure.

procedure: `(list->gdk-color` *`l`*`)`

Convert a list (R G B) into a GdkColor boxed object.

procedure: `(gdk-color-pixel` *`c`*`)`

Extract the pixel value from a GdkColor structure.

procedure: `(gdk-color-pixel-set!` *`color newpixel`*`)`

Update the pixel value within a GdkColor structure.

procedure: `(gdk-rectangle->list` *`r`*`)`

Convert a GdkRectangle into a list (x y width height).

procedure: `(list->gdk-rectangle` *`l`*`)`

Convert a list (x y width height) into a GdkRectangle boxed object.

procedure: `(gdk-window-get-pointer` *`w`*`)`

Returns multiple values: (x y state), where x and y make up the current pointer coordinate, and state is a list of GdkModifierType symbols.

2.2.4. Miscellaneous and overridden procedures

procedure: `(gtk:gc-idle-timeout` *`(#:optional value)`*`)`

If `value` is omitted, returns the current setting for the number of milliseconds of GTK idleness before a GC is forced; otherwise, sets the setting to the passed-in number of milliseconds. Only used when `gtk:gc-when-idle` has been enabled. Defaults to 1000 milliseconds.

procedure: `(gtk:gc-when-idle` *`(#:optional value)`*`)`

If `value` is omitted, returns `#t` if the GTK-idle-garbage-collector is enabled, or `#f` otherwise. If `value` is specified, enables the idle-garbage-collector unless `value` is `#f`. Defaults to being switched off.

procedure: `(gtk-calendar-get-date` *`cal`*`)`

Retrieve the date selected by a GtkCalendar widget, in the form of a list of three numbers, year, month, day: `(2002 10 13)`.

procedure: `(gtk-stock-list-ids)`

Returns a list of all current GTK+ “stock ID” strings.

procedure: `(gtk-tree-iter-new)`

Allocates a new instance of GtkTreeIter, for use with various GTK+ tree model and view functions.

procedure: `(gtk-list-store-new` *`coltypes ...`*`)`

Creates and returns a new instance of GtkListStore with the same number of columns as parameters to the function call. Each parameter should be a gtype record (as returned by `gtype-from-name`, for example, or as stored in variables such as *gtype:string* or *gtype:boolean*).

procedure: `(gtk-tree-store-new` *`coltypes ...`*`)`

Creates and returns a new instance of GtkTreeStore with the same number of columns as parameters to the function call. Each parameter should be a gtype record, as for `gtk-list-store-new`.

procedure: `(gtk-list-store-set-column-types` *`l coltypes ...`*`)`

Sets the number and type of columns associated with the GtkListStore `l`. `coltypes` are as for `gtk-list-store-new`.

procedure: `(gtk-tree-store-set-column-types` *`t coltypes ...`*`)`

Sets the number and type of columns associated with the GtkTreeStore `t`. `coltypes` are as for `gtk-tree-store-new`.

procedure: `(gtk-tree-selection-get-selected` *`sel iter`*`)`

Stores the currently-selected row of the GtkTreeSelection `sel` (single-row-selection mode only) into the GtkTreeIter `iter`. If there is no current selection, `#f` is returned; otherwise, the associated GtkTreeModel is returned.

procedure: `(gtk-widget-window` *`w`*`)`

Extracts the *`window`* field of the GtkWidget struct associated with the passed-in object.

procedure: `(gtk-widget-allocation` *`w`*`)`

Extracts the *`allocation`* field of the GtkWidget struct associated with the passed-in object.

procedure: `(gtk-widget-get-state` *`w`*`)`

Extracts the *`state`* field of the GtkWidget struct associated with the passed-in object, and returns it in symbolic form.

procedure: `(gtk-style-black-gc` *`style`*`)`

Retrieves the black GC from the passed-in style.

procedure: `(gtk-style-white-gc` *`style`*`)`

Retrieves the white GC from the passed-in style.

procedure: `(gtk-style-fg-gc` *`style state`*`)`

Retrieves the foreground GC from the passed-in style that is appropriate to the passed-in GtkStateType symbol.

procedure: (gtk-editable-insert-text *editable string position*)

Inserts text `string` at the `position` passed in. Returns the new insertion position after the insert operation.

2.3. G+, a higher-level GTK+ interface

```
(require 'g+)
```

G+ is based on the ideas in JLib, a library for building GUI widget trees which comes with Jscheme.

2.3.1. Core macros and functions

macro: (g+predicate-case *(varname ...) ((predicate ...) body ...) ...*)

Expands into a `cond` expression which tests each `varname` against the corresponding `predicate`, executing the `body` of the first clause for which all the `predicate`s return true. (A clause may also have the keyword `else` instead of a list of predicates, with effect similar to `cond` and `case`.)

macro: (g+define-ctor *name (base-ctor args ...)*)

Expands into a definition of `name`, a function which accepts `args ...` and passes them to `base-ctor`, keeping the result, passing the result to `g+:configure` with any extra arguments supplied, and then returning the result of the call to `base-ctor`.

For example:

```
(g+define-ctor X (A B C D))
```

expands into:

```
(define (X B C D . g+args)
  (let ((x (A B C D)))
    (g+:configure x g+args)
    x))
```

2.3.2. Constructors and modifiers

procedure: `(g+:configure x items)`

Given an object `x`, and a list of `items`, takes different actions depending on the types of `x` and each `item` in turn. In general, if `x` is some kind of container, and an `item` is some kind of widget or object appropriate for containment within that container, it will be placed inside it. If an `item` is a string, and `x` has some kind of intuitively-default text-string property on it, the property will be set. If an `item` is a procedure, the procedure will be called with `x` as its single argument.

This function is the core of the G+ library, and is the main idea taken from JLib: the heavy use of lambdas makes for a fairly clean way of building an extensible optional-argument and -property system.

procedure: `(g+property name value)`

Returns a function that when applied to a GObject, sets a property on its argument. For use with `g+:configure` (and by extension constructors defined with `g+define-ctor`).

procedure: `(g+signal name handler)`

Returns a function that when applied to a GObject, installs a signal-handler on it using `gsignal-connect`.

procedure: `(g+pack-start expand fill padding widgets ...)`

Returns a function that when applied to a GtkBox, packs all the `widgets` into it using `gtk-box-pack-start`.

procedure: `(g+pack-end expand fill padding widgets ...)`

Returns a function that when applied to a GtkBox, packs all the `widgets` into it using `gtk-box-pack-end`.

procedure: `(g+tip tooltips text)`

Returns a function that when applied to a GtkWidget, sets the tooltip on that widget in the `tooltips` set to be `text`.

procedure: `(g+label-markup markup-mnemonic)`

Returns a function that when applied to a GtkLabel, sets its markup and mnemonic keysequence according to `markup-mnemonic`.

procedure: `(g+label-markup* markup)`

Returns a function that when applied to a GtkLabel, sets its markup according to `markup`.

procedure: `(g+button mnemonic ...)`

procedure: `(g+button* ...)`

procedure: `(g+stock-button stock-id ...)`

These three constructors use `g+:configure` to build variants on GtkButton.

procedure: `(g+label mnemonic ...)`

procedure: `(g+label` *`text ...`*`)`

These constructors use `g+:configure` to build variants on GtkLabel.

procedure: `(g+entry ...)`

procedure: `(g+entry/max-length` *`max-length ...`*`)`

These constructors use `g+:configure` to build variants on GtkEntry.

procedure: `(g+window` *`type ...`*`)`

Builds a GtkWindow using `gtk-window-new` and `g+:configure`.

procedure: `(g+dialog ...)`

Builds a GtkDialog using `gtk-dialog-new` and `g+:configure`.

procedure: `(g+vbox` *`homogeneous spacing ...`*`)`

procedure: `(g+hbox` *`homogeneous spacing ...`*`)`

These constructors use `g+:configure` to build variants on GtkBox.

procedure: `(g+vbutton-box ...)`

procedure: `(g+hbutton-box ...)`

These constructors use `g+:configure` to build variants on GtkButtonBox.

procedure: `(g+vpaned ...)`

procedure: `(g+hpaned ...)`

These constructors use `g+:configure` to build variants on GtkPaned.

procedure: `(g+menu ...)`

procedure: `(g+menu-bar ...)`

procedure: `(g+menu-item` *`mnemonic ...`*`)`

procedure: `(g+menu-item* ...)`

These constructors use `g+:configure` to build variants on GtkMenu and GtkOptionMenu.

procedure: `(g+option-menu ...)`

Builds a GtkOptionMenu using `gtk-option-menu-new` and `g+:configure`.

procedure: `(g+tooltips ...)`

Builds a GtkTooltips object using `gtk-tooltips-new` and `g+:configure`.

procedure: `(g+toolbar ...)`

Builds a GtkToolbar object using `gtk-toolbar-new` and `g+:configure`.

procedure: `(g+calendar ...)`

Builds a GtkCalendar object using `gtk-calendar-new` and `g+:configure`.

procedure: `(g+check-button` *`mnemonic ...`*`)`

procedure: `(g+check-button* ...)`

These constructors use `g+:configure` to build variants on GtkCheckButton.

procedure: `(g+radio-button` *`group-or-null-gobject mnemonic ...`*`)`

procedure: `(g+radio-button*` *`group-or-null-gobject ...`*`)`

These constructors use `g+:configure` to build variants on GtkRadioButton.

procedure: `(g+adjustment` *`current min max stepincr pageincr pagesize ...`*`)`

procedure: `(g+hscrollbar` *`adjustment ...`*`)`

procedure: `(g+vscrollbar` *`adjustment ...`*`)`

procedure: `(g+hscale` *`adjustment ...`*`)`

procedure: `(g+vscale` *`adjustment ...`*`)`

procedure: `(g+spin-button` *`adjustment climbrate numdigits ...`*`)`

procedure: `(g+spin-button/range` *`min max step ...`*`)`

These constructors use `g+:configure` to build variants on GtkAdjustment, GtkScrollbar, GtkScale and GtkSpinButton.

procedure: `(g+arrow` *`arrow-type shadow-type ...`*`)`

Builds a GtkArrow object using `gtk-arrow-new` and `g+:configure`.

procedure: `(g+scrolled-window` *`hscrollbar vscrollbar ...`*`)`

Builds a GtkScrolledWindow object using `gtk-scrolled-window-new` and `g+:configure`.

procedure: `(g+table` *`rows columns homogeneous`*`)`

Builds a GtkTable using `gtk-table-new` and `g+:configure`.

procedure: `(g+table-cell` *`left right top bottom widget`*`)`

Uses `gtk-table-attach-defaults` to place a widget within a GtkTable.

procedure:
`(g+table-cell*` *`left right top bottom xoptions yoptions xpadding ypadding widget`*`)`

Uses `gtk-table-attach` to place a widget within a GtkTable.

procedure: `(g+notebook ...)`

Builds a GtkNotebook object using `gtk-notebook-new` and `g+:configure`.

procedure: (g+notebook-page *label-widget page-widget*)

Returns a function that when applied to a GtkNotebook, appends a page to it using `gtk-notebook-append-page`.

procedure: (g+notebook-page* *label-widget menu-widget page-widget*)

Returns a function that when applied to a GtkNotebook, appends a page to it using `gtk-notebook-append-page-menu`.

procedure: (g+list-store *typelist rows ...*)

Builds a GtkListStore object using `g+:make-list-store` and `g+:configure`.

procedure: (g+:make-list-store *typelist rows*)

Creates a new GtkListStore, and creates (`length typelist`) columns. Each element of `typelist` should be a GType record. The `rows` should contain zero or more lists of entries to put in the list store. Each row must contain items that correspond to the GTypes passed in `typelist`.

procedure: (g+:list-store-append! *ls typelist rows*)

Appends `rows` to `ls`, using the list of GType records in `typelist` to build the intermediate GValues.

procedure: (g+tree-store *typelist rows ...*)

Builds a GtkTreeStore object using `g+:make-tree-store` and `g+:configure`.

procedure: (g+:make-tree-store *typelist rows*)

Creates a new GtkTreeStore, and creates (`length typelist`) columns. Each element of `typelist` should be a GType record. The `rows` should contain zero or more lists of entries to put in at the root of the tree. Each row must contain items that correspond to the GTypes passed in `typelist`, followed by child rows (that follow the same definition).

For example:

```
(g+:make-tree-store (list gtype:string gtype:int)
                    '(("A" 100
                           ("AA" 110
                                 ("AAA" 111))
                           ("AB" 120))
                      ("B" 200
                           ("BA" 210)
                           ("BB" 220))))
```

procedure: (g+:tree-store-append! *ts typelist parent-iter rows*)

Appends `rows` to `ts`, under the parent element at `parent-iter` (pass in `(null-gboxed)` to refer to the root element), using the list of GType records in `typelist` to build the intermediate GValues.

procedure: (g+tree-view *tree-model ...*)

Wraps `gtk-tree-view-new-with-model` with a `g+:configure` step.

procedure:
`(g+tree-view-column` *`title renderer column-id updater editable-column g+args ...`*`)`

Creates and returns a configured instance of GtkTreeViewColumn.

`title` should be the text used as the column heading. `renderer` should be either one of the symbols `(text toggle pixbuf)`, or an instance of GtkCellRenderer. `column-id` should be the column from the GtkTreeModel to fetch data to render from. (To render the data in the first column on the GtkTreeModel, pass in 0; the third column, pass 2; etc.)

`updater` may supply a function which will be called when the content of the cell renderer is edited by the user. Set it to `#f` if you don't want to install a handler for edited cells. `editable-column` may supply a GtkTreeModel column number which contains GBoolean information specifying whether the cell rendered by this column at a particular row should be user-editable or not. Supply `#f` if you want the cell to be left in its default state with regard to editability.

Both `updater` and `editable-column` are only relevant if `renderer` is a symbol - if it's a GtkCellRenderer instance, this function has no way of working out how to set updater or edit-column properties, so it leaves it up to its caller.

2.4. GdkEvent binding

```
(require 'gdkevent)
```

This extension is automatically included when the gtk extension is `require`d. It provides accessors for fields in GdkEvent boxed structures.

procedure: `(gdk-event-type` *`e`*`)`

Retrieves the (symbolic) GdkEventType from a GdkEvent.

procedure: `(gdk-event-window` *`e`*`)`

Retrieves the GdkWindow associated with a GdkEvent.

procedure: `(gdk-event-string` *`e`*`)`

Retrieves the string associated with a GdkEvent, or `#f` if there is no associated string. (Currently supports `key-press` and `key-release` events.)

procedure: `(gdk-event-area` *`e`*`)`

Retrieves the area rectangle of an expose event, or `#f` if the passed-in event is of the wrong type.

procedure: `(gdk-event-button` *`e`*`)`

Retrieves the button number of a button event, or `#f` if the passed-in event is of the wrong type.

procedure: `(gdk-event-xy` *`e`*`)`

Returns two values, the X and Y coordinates associated with a GdkEvent. Returns `(values #f #f)` if there is no associated coordinate pair.

procedure: `(gdk-event-xy-root` *`e`*`)`

As for `gdk-event-xy`, except returns coordinates in the root window coordinate system rather than the window-local coordinate system.

2.5. Libglade 2.0 binding

```
(require 'libglade)
```

The libglade extension module provides a wrapping for James Henstridge's Libglade library, version 2.0. It depends upon the gobject and gtk extensions.

procedure: `(glade-xml-new` *`filename (#:domain domain) (#:root root)`*`)`

Reads the Glade XML file `filename`, constructing the widget tree. The optional keyword arguments `domain` and `root` are passed through to the underlying C function, `glade_xml_new`; if they are omitted, `NULL` is passed in their place.

procedure:
`(glade-xml-new-from-memory` *`bv-or-string (#:domain domain) (#:root root)`*`)`

As for `glade-xml-new`, except instead of reading XML from a file, reads XML from a byte-vector or string (`bv-or-string`). Delegates to the C function `glade_xml_new_from_memory`.

procedure: `(glade-xml-construct` *`xml filename (#:domain domain) (#:root root)`*`)`

Fills in a newly-created GladeXML widget, `xml`, with information from the Glade XML file `filename`, as for `glade-xml-new`. Delegates to the C function `glade_xml_construct`.

procedure: `(glade-xml-signal-autoconnect` *`xml handlers-alist`*`)`

Connects handlers named in the GladeXML widget `xml` to the Scheme functions passed in in `handlers-alist`. `handlers-alist` should be an association list, suitable for use with `assoc`, which maps strings (the names of the handlers as specified in the original XML) to Scheme functions of appropriate arity. Delegates to the C function `glade_xml_signal_autoconnect_full`.

procedure: `(glade-xml-get-widget` *`xml name`*`)`

Retrieve a named subwidget from a GladeXML widget by name. Delegates to the C function `glade_xml_get_widget`.

procedure: `(glade-xml-get-widget-by-long-name` *`xml name`*`)`

Retrieve a named subwidget from a GladeXML widget by long name. Delegates to the C function `glade_xml_get_widget_by_long_name`.

Notes

1. Documentation nicked outright from the GLib GType documentation.
2. Isn’t it nice having procedures in mutable global variables?

Index