Next: GtkTreeStore, Previous: GtkCellRendererToggle, Up: Top

---

52 GtkListStore

A list-like data structure that can be used with the

52.1 Overview

The <gtk-list-store> object is a list model for use with a <gtk-tree-view> widget. It implements the <gtk-tree-model> interface, and consequentialy, can use all of the methods available there. It also implements the <gtk-tree-sortable> interface so it can be sorted by the view. Finally, it also implements the tree drag and drop interfaces.

The <gtk-list-store> can accept most GObject types as a column type, though it can't accept all custom types. Internally, it will keep a copy of data passed in (such as a string or a boxed pointer). Columns that accept <gobject>s are handled a little differently. The <gtk-list-store> will keep a reference to the object instead of copying the value. As a result, if the object is modified, it is up to the application writer to call *gtk-tree-model-row-changed* to emit the "row_changed" signal. This most commonly affects lists with <gdk-pixbuf>s stored.

```
enum {
  COLUMN_STRING,
  COLUMN_INT,
  COLUMN_BOOLEAN,
  N_COLUMNS
};

{
  GtkListStore *list_store;
  GtkTreePath *path;
  GtkTreeIter iter;
  gint i;

  list_store = gtk_list_store_new (N_COLUMNS,
                                   G_TYPE_STRING,
                                   G_TYPE_INT,
                                   G_TYPE_BOOLEAN);

  for (i = 0; i < 10; i++)
    {
      gchar *some_data;

      some_data = get_some_data (i);

      /* Add a new row to the model */
      gtk_list_store_append (list_store, &iter);
      gtk_list_store_set (list_store, &iter,
                          COLUMN_STRING, some_data,
                          COLUMN_INT, i,
                          COLUMN_BOOLEAN,  FALSE,
                          -1);

      /* As the store will keep a copy of the string internally, we
       * free some_data.
       */
      g_free (some_data);
    }

  /* Modify a particular row */
  path = gtk_tree_path_new_from_string ("4");
  gtk_tree_model_get_iter (GTK_TREE_MODEL (list_store),
                           &iter,
```

```
                                path);
      gtk_tree_path_free (path);
      gtk_list_store_set (list_store, &iter,
                          COLUMN_BOOLEAN, TRUE,
                          -1);
    }
```

52.2 Performance Considerations

Internally, the <gtk-list-store> was implemented with a linked list with a tail pointer prior to GTK+ 2.6. As a result, it was fast at data insertion and deletion, and not fast at random data access. The <gtk-list-store> sets the <gtk-tree-model-iters-persist> flag, which means that <gtk-tree-iter>s can be cached while the row exists. Thus, if access to a particular row is needed often and your code is expected to run on older versions of GTK+, it is worth keeping the iter around.

It is important to note that only the methods *gtk-list-store-insert-with-values* and *gtk-list-store-insert-with-valuesv* are atomic, in the sense that the row is being appended to the store and the values filled in in a single operation with regard to <gtk-tree-model> signaling. In contrast, using e.g. *gtk-list-store-append* and then *gtk-list-store-set* will first create a row, which triggers the "row_inserted" <gtk-tree-model> signal on <gtk-list-store>. The row, however, is still empty, and any signal handler connecting to "row_inserted" on this particular store should be prepared for the situation that the row might be empty. This is especially important if you are wrapping the <gtk-list-store> inside a <gtk-tree-model-filter> and are using a <gtk-tree-model-filter-visible-func>. Using any of the non-atomic operations to append rows to the <gtk-list-store> will cause the <gtk-tree-model-filter-visible-func> to be visited with an empty row first; the function must be prepared for that.

52.3 Usage

— Class: **<gtk-list-store>**

Derives from <gtk-tree-model>, <gtk-tree-sortable>, <gtk-buildable>, <gtk-tree-drag-dest>, <gtk-tree-drag-source>, <gobject>.

This class defines no direct slots.

— Function: **gtk-list-store-new** (*types* `scm`) ⇒ (*ret* <gtk-list-store>)

Creates a new list store as with *n-columns* columns each of the types passed in. Note that only types derived from standard GObject fundamental types are supported.

As an example, ‘`gtk_tree_store_new (3, G_TYPE_INT, G_TYPE_STRING, GDK_TYPE_PIXBUF);`’ will create a new <gtk-list-store> with three columns, of type int, string and <gdk-pixbuf> respectively.

n-columns
: number of columns in the list store

...
: all <g-type> types for the columns, from first to last

ret
: a new <gtk-list-store>

— Function: **gtk-list-store-set-value** (*self* <gtk-list-store>) (*iter* <gtk-tree-iter>) (*column* `int`) (*value* `scm`)
— Method: **set-value**

Sets the data in the cell specified by *iter* and *column*. The type of *value* must be convertible to the type of the column.

list-store

A <gtk-list-store>

iter
A valid <gtk-tree-iter> for the row being modified

column
column number to modify

value
new value for the cell

— Function: **gtk-list-store-remove** (*self* <gtk-list-store>) (*iter* <gtk-tree-iter>) ⇒ (*ret* <gtk-tree-iter>)
— Method: **remove**

Removes the given row from the list store. After being removed, *iter* is set to be the next valid row, or invalidated if it pointed to the last row in *list-store*.

list-store
A <gtk-list-store>

iter
A valid <gtk-tree-iter>

ret
'#t' if *iter* is valid, '#f' if not.

— Function: **gtk-list-store-insert** (*self* <gtk-list-store>) (*position* int) ⇒ (*ret* <gtk-tree-iter>)
— Method: **insert**

Creates a new row at *position*. *iter* will be changed to point to this new row. If *position* is larger than the number of rows on the list, then the new row will be appended to the list. The row will be empty after this function is called. To fill in values, you need to call gtk-list-store-set or gtk-list-store-set-value.

list-store
A <gtk-list-store>

iter
An unset <gtk-tree-iter> to set to the new row

position
position to insert the new row

— Function: **gtk-list-store-insert-before** (*self* <gtk-list-store>) (*sibling* <gtk-tree-iter>) ⇒ (*ret* <gtk-tree-iter>)
— Method: **insert-before**

Inserts a new row before *sibling*. If *sibling* is '#f', then the row will be appended to the end of the list. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk-list-store-set or gtk-list-store-set-value.

list-store
A <gtk-list-store>

iter
An unset <gtk-tree-iter> to set to the new row

sibling
A valid <gtk-tree-iter>, or '#f'

— Function: **gtk-list-store-insert-after** (*self* <gtk-list-store>) (*sibling* <gtk-tree-iter>) ⇒ (*ret* <gtk-tree-iter>)
— Method: **insert-after**

Inserts a new row after *sibling*. If *sibling* is '#f', then the row will be prepended to the beginning of the list. *iter* will be changed to point to this new row. The row will be empty after this

function is called. To fill in values, you need to call `gtk-list-store-set` or `gtk-list-store-set-value`.

list-store
: A `<gtk-list-store>`

iter
: An unset `<gtk-tree-iter>` to set to the new row

sibling
: A valid `<gtk-tree-iter>`, or '`#f`'

— Function: **gtk-list-store-prepend** (*self* `<gtk-list-store>`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **prepend**

Prepends a new row to *list-store*. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call `gtk-list-store-set` or `gtk-list-store-set-value`.

list-store
: A `<gtk-list-store>`

iter
: An unset `<gtk-tree-iter>` to set to the prepend row

— Function: **gtk-list-store-append** (*self* `<gtk-list-store>`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **append**

Appends a new row to *list-store*. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call `gtk-list-store-set` or `gtk-list-store-set-value`.

list-store
: A `<gtk-list-store>`

iter
: An unset `<gtk-tree-iter>` to set to the appended row

— Function: **gtk-list-store-clear** (*self* `<gtk-list-store>`)
— Method: **clear**

Removes all rows from the list store.

list-store
: a `<gtk-list-store>`.

— Function: **gtk-list-store-iter-is-valid** (*self* `<gtk-list-store>`) (*iter* `<gtk-tree-iter>`) ⇒ (*ret* `bool`)
— Method: **iter-is-valid**

purposes.")

Checks if the given iter is a valid iter for this `<gtk-list-store>`.

list-store
: A `<gtk-list-store>`.

iter
: A `<gtk-tree-iter>`.

ret
: '`#t`' if the iter is valid, '`#f`' if the iter is invalid.

Since 2.2

— Function: **gtk-list-store-reorder** (*self* `<gtk-list-store>`) ⇒ (*new_order* `int`)
— Method: **reorder**

Reorders *store* to follow the order indicated by *new-order*. Note that this function only works with unsorted stores.

store
A `<gtk-list-store>`.
new-order
an array of integers mapping the new position of each child to its old position before the re-ordering, i.e. *new-order*‘`[newpos] = oldpos`’.

Since 2.2

— Function: **gtk-list-store-swap** (*self* `<gtk-list-store>`) (*a* `<gtk-tree-iter>`) (*b* `<gtk-tree-iter>`)
— Method: **swap**

Swaps *a* and *b* in *store*. Note that this function only works with unsorted stores.

store
A `<gtk-list-store>`.
a
A `<gtk-tree-iter>`.
b
Another `<gtk-tree-iter>`.

Since 2.2

— Function: **gtk-list-store-move-before** (*self* `<gtk-list-store>`) (*iter* `<gtk-tree-iter>`) (*position* `<gtk-tree-iter>`)
— Method: **move-before**

Moves *iter* in *store* to the position before *position*. Note that this function only works with unsorted stores. If *position* is ‘`#f`’, *iter* will be moved to the end of the list.

store
A `<gtk-list-store>`.
iter
A `<gtk-tree-iter>`.
position
A `<gtk-tree-iter>`, or ‘`#f`’.

Since 2.2

— Function: **gtk-list-store-move-after** (*self* `<gtk-list-store>`) (*iter* `<gtk-tree-iter>`) (*position* `<gtk-tree-iter>`)
— Method: **move-after**

Moves *iter* in *store* to the position after *position*. Note that this function only works with unsorted stores. If *position* is ‘`#f`’, *iter* will be moved to the start of the list.

store
A `<gtk-list-store>`.
iter
A `<gtk-tree-iter>`.
position
A `<gtk-tree-iter>` or ‘`#f`’.

Since 2.2