**GTK Development Blog**

All things GTK

# On list models

In the previous post, I promised to take a deeper look at list models and what GTK 4 offers in this area. Lets start be taking a look at the GListModel interface:

```
struct _GListModelInterface
{
  GTypeInterface g_iface;

  GType    (* get_item_type) (GListModel *list);
  guint    (* get_n_items)   (GListModel *list);
  gpointer (* get_item)      (GListModel *list,
                               guint       position);
};
```

An important part of implementing the interface is that you need to emit the ::items-changed signal when required, using the helper function that GLib has for this purpose:

```
void g_list_model_items_changed (GListModel *list,
                                 guint       position,
                                 guint       removed,
                                        guint        added)
```

A few things to note about this interface:

- It is *very* minimal; which makes it easy to implement
- The API is in terms of positions and only deals with changes in list membership —keeping track of changes to the items themselves is up to you

## A list model zoo

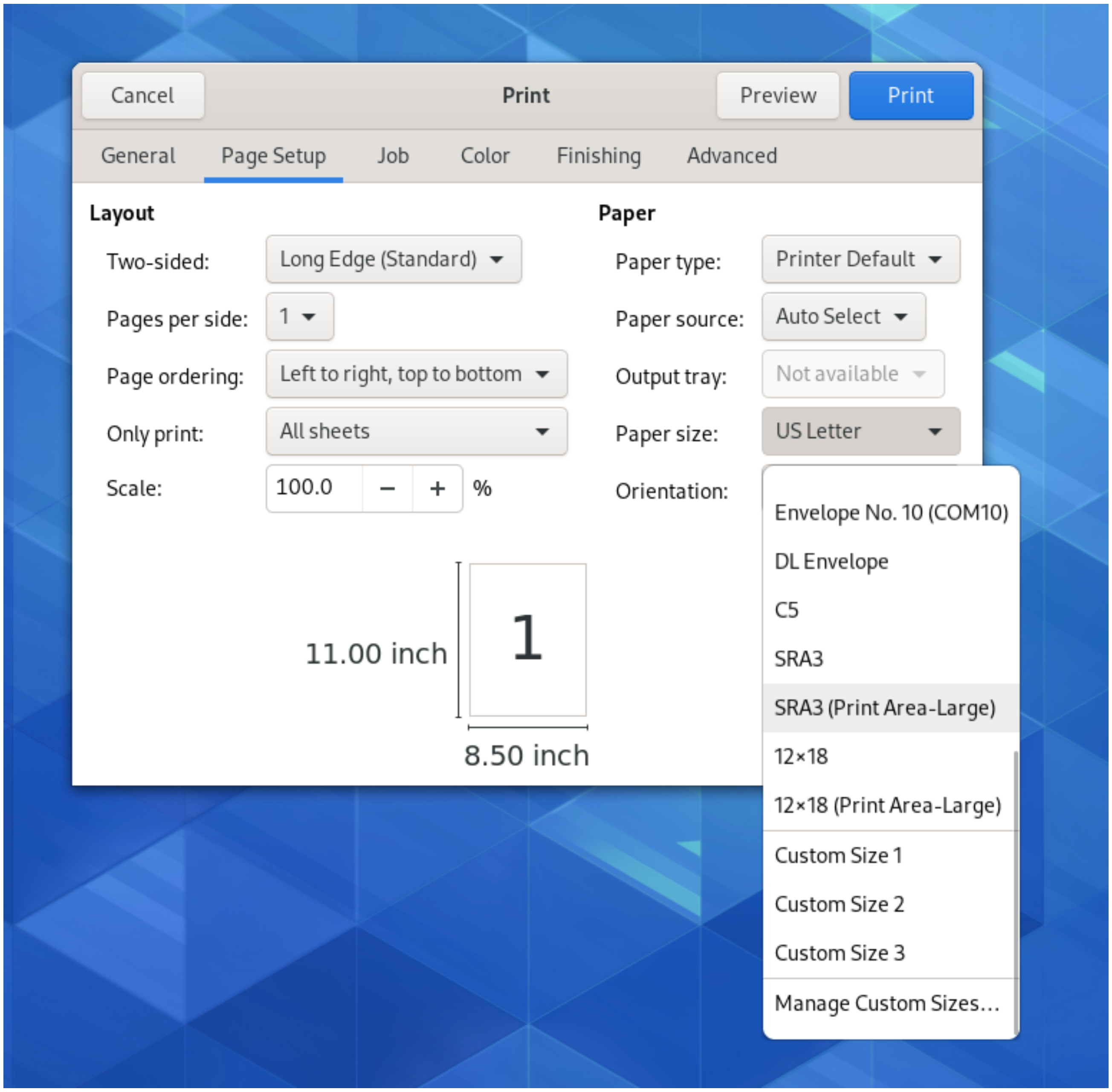
GTK ships a sizable collection of list model implementations. Under closer inspection, they fall into several distinct groups.

### LIST MODEL CONSTRUCTION KIT

The first group is what could be called the *list model construction kit:* models that let you build new models by modifying or combining models that you already have.

The first model in this group, GtkSliceListModel, take a slice of an existing model, given by an offset and a size, and makes a new model containing just those items. This is useful if you want to present a big list in a paged view—the forward and back buttons will simply increase or decrease the offset by the size. A slice model can also be used to incrementally populate a list, by making the slice bigger over time. GTK is using this technique in some places.

The next model in this group, GtkFlattenListModel, takes several list models and combines them into one. Since this is all about list models, the models to combine are handed to the flatten model in the form of a list model of list models. This is useful whenever you need to combine data from multiple sources, as for example GTK does for the paper sizes in the print dialog.



Layout

Two-sided: Long Edge (Standard)
Pages per side: 1
Page ordering: Left to right, top to bottom
Only print: All sheets
Scale: 100.0 %

Paper

Paper type: Printer Default
Paper source: Auto Select
Output tray: Not available
Paper size: US Letter
Orientation:

*A flattened list*

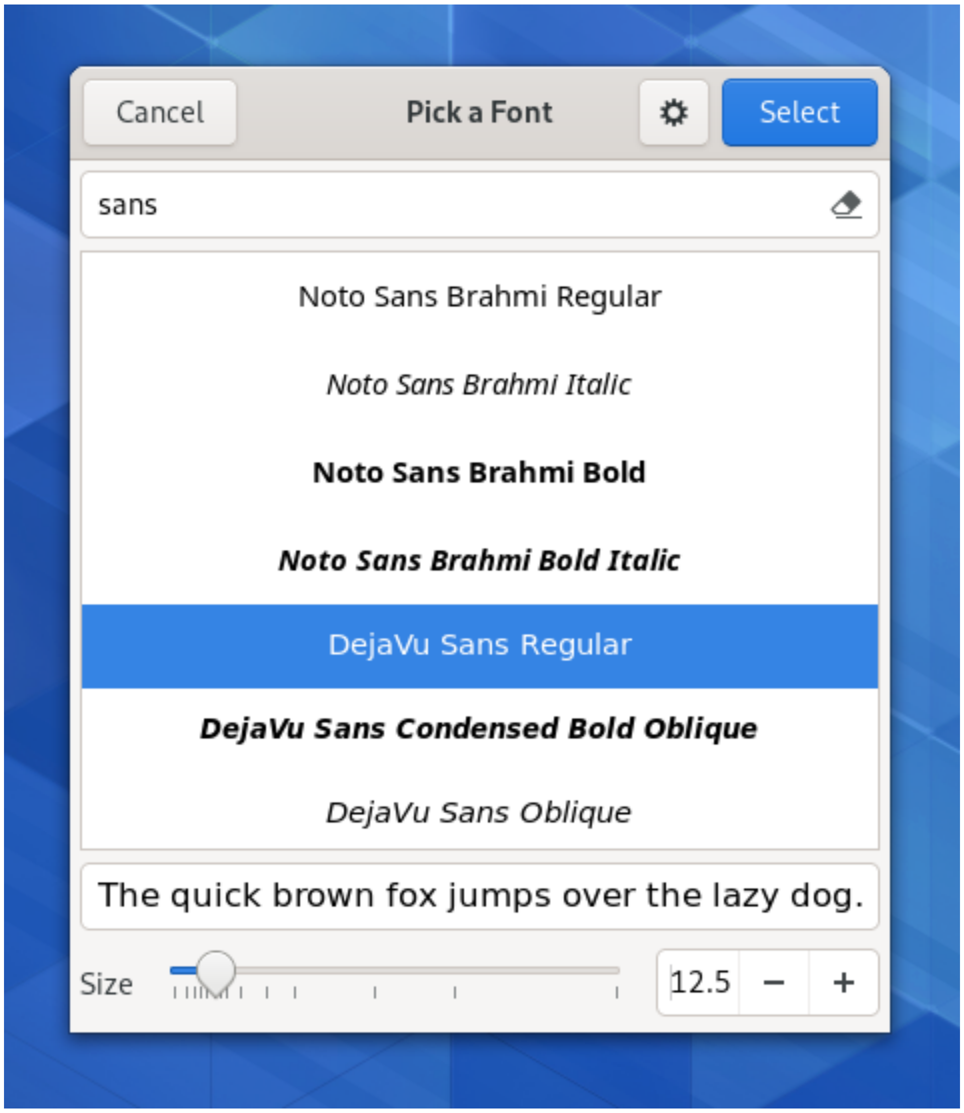
Note that the original models continue to exist behind the flatten model, and their updates will be propagated by the flatten list model, as expected.

Sometimes, you have your data in a list model, but it is not quite in the right form. In this case, you can use a GtkMapListModel replace every item in the original model with different one.

## CONCRETE MODELS

GTK and its dependencies include a number of concrete models for the types of data that we deal with ourselves.

The first example here are Pango objects that are implementing the list model interface for their data: **PangoFontMap** is a list model of PangoFontFamily objects, and **PangoFontFamily** is a list model of PangoFontFace objects. The font chooser is using these models.



*A Pango list model*

The next example are the GtkDirectoryList and GtkBookmarkList objects that will be used in the file chooser to represent directory contents and bookmarks. An interesting detail about these is that they both need to do IO to populate their content, and they do it asynchronously to avoid blocking the UI for extended times.

The last model in this group is a little less concrete: GtkStringList is a simple list model wrapper around the all-too-common string arrays. An example where this kind of list model will be frequently used is with GtkDropDown. This is so common that GtkDropDown has a convenience constructor that takes a string array and creates the GtkStringList for you:

```
GtkWidget *
    gtk_drop_down_new_from_strings (const char * const * strings)
```

## SELECTION

The next group of models extends GListModel with a new interface: GtkSelectionModel. For each item in the underlying model, a GtkSelectionModel maintains the information whether it is *selected* or not.

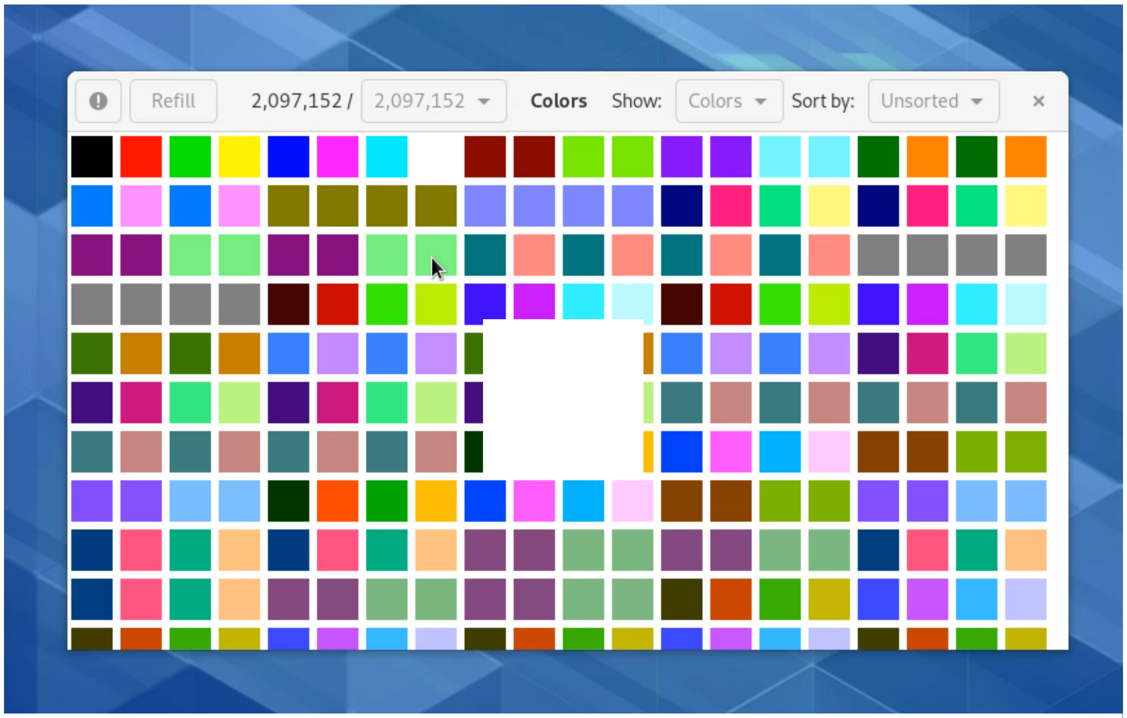
We won't discuss the interface in detail, since  it is unlikely that you need to implement it yourself, but the most important points are:

```
gboolean gtk_selection_model_is_selected (GtkSelectionModel *model)
                                          guint              pos)
GtkBitset *
       gtk_selection_model_get_selection (GtkSelectionModel *model)
```

So you can get the selection information for an individual item, or as a whole, in the form of a bitset. Of course, there is also a ::selection-changed signal that works in a very similar way to the ::items-changed signal of GListModel.

GTK has three GtkSelectionModel implementations: GtkSingleSelection, GtkMultiSelection and GtkNoSelection, which differ in the number of items that can be simultaneously selected (1, many, or 0).

The GtkGridView colors demo shows a multi-selection in action, with rubberbanding:



00:00 00:07

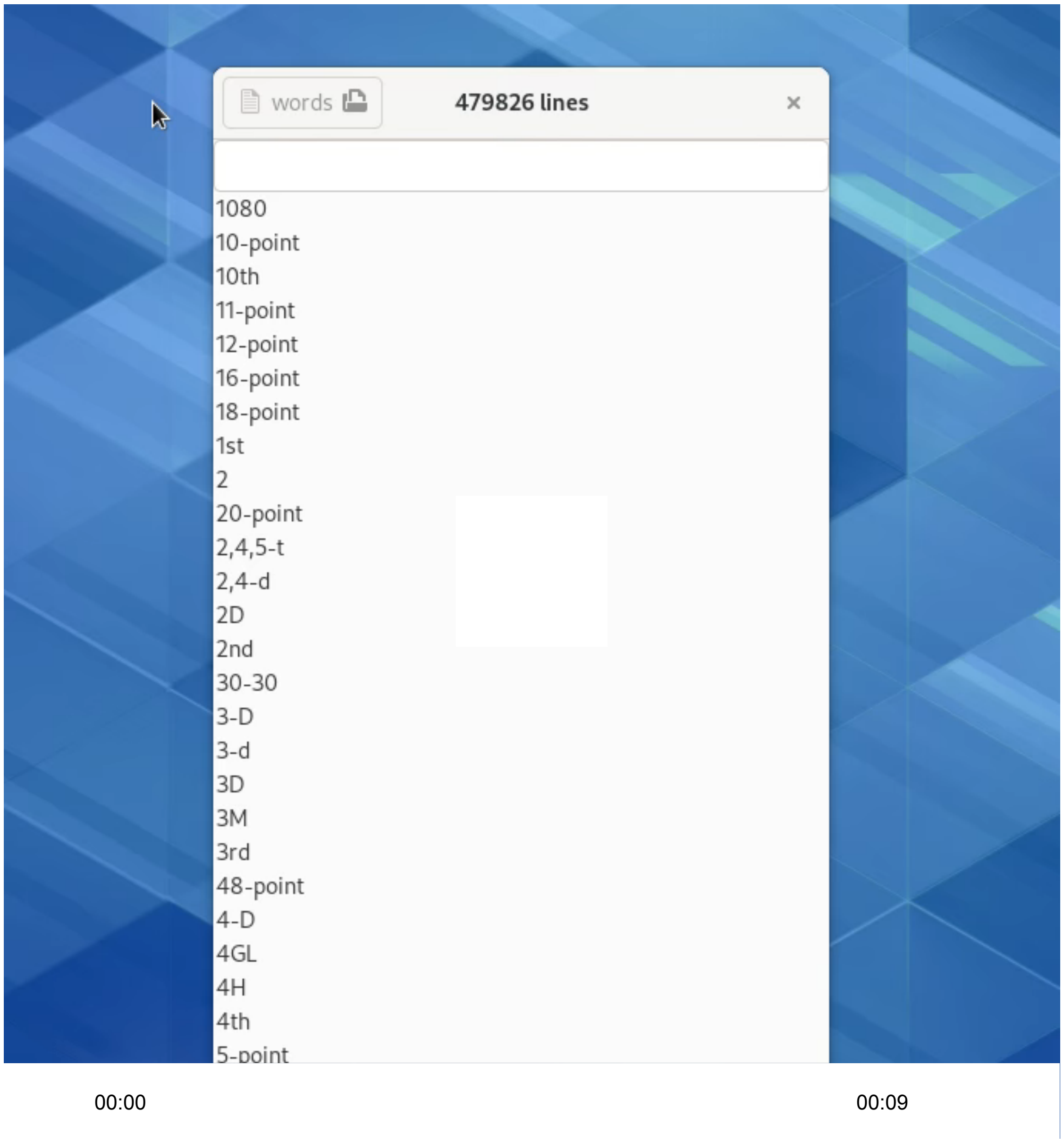
You are very likely to encounter selection models when working with GTK's new list widgets, since they all expect their models to be selection models.

## THE BIG ONES

The last group of models I want to mention are the ones doing the typical operations you expect in lists: filtering and sorting. The models are GtkFilterListModel and GtkSortListModel. The both use auxiliary objects to implement their operations: GtkFilter and GtkSorter. Both of these have subclasses to handle common cases: sorting and filtering strings or numbers, or using callbacks.

We have spent considerable effort on these two models in the run-up to GTK 3.99, and made them do their work incrementally, to avoid blocking the UI for extended times when working with big models.

The GtkListView words demo show interactive filtering of a list of 500.000 words:



1080
10-point
10th
11-point
12-point
16-point
18-point
1st
2
20-point
2,4,5-t
2,4-d
2D
2nd
30-30
3-D
3-d
3D
3M
3rd
48-point
4-D
4GL
4H
4th
5-point

## THE LEFTOVERS

There are some more list model implementations in GTK that do not fit neatly in any of the above groups, such as GtkTreeListModel, GtkSelectionFilterModel or GtkShortcutController. I'll skip these today.

## Models everywhere

I'll finish with a brief list of GTK APIs that return list models:

- gdk_display_get_monitors
- gtk_widget_observe_children
- gtk_widget_observe_controllers
- gtk_constraint_layout_observe_constraints
- gtk_constraint_layout_observe_guides
- gtk_file_chooser_get_files
- gtk_drop_down_get_model
- gtk_list_view_get_model
- gtk_grid_view_get_model
- gtk_column_view_get_model
- gtk_column_view_get_columns
- gtk_window_get_toplevels
- gtk_assistant_get_pages
- gtk_stack_get_pages
- gtk_notebook_get_pages

In summary, list models are everywhere in GTK 4. They are flexible and fun, you should use them!

mclasen / September 8, 2020 / Uncategorized / gtk4, guides, in depth, lists