Next: GtkTreeSelection, Previous: GtkTextView, Up: Top

---

32 GtkTreeModel

The tree interface used by

32.1 Overview

The <gtk-tree-model> interface defines a generic tree interface for use by the <gtk-tree-view> widget. It is an abstract interface, and is designed to be usable with any appropriate data structure. The programmer just has to implement this interface on their own data type for it to be viewable by a <gtk-tree-view> widget.

The model is represented as a hierarchical tree of strongly-typed, columned data. In other words, the model can be seen as a tree where every node has different values depending on which column is being queried. The type of data found in a column is determined by using the GType system (ie. <g-type-int>, <gtk-type-button>, <g-type-pointer>, etc.). The types are homogeneous per column across all nodes. It is important to note that this interface only provides a way of examining a model and observing changes. The implementation of each individual model decides how and if changes are made.

In order to make life simpler for programmers who do not need to write their own specialized model, two generic models are provided — the <gtk-tree-store> and the <gtk-list-store>. To use these, the developer simply pushes data into these models as necessary. These models provide the data structure as well as all appropriate tree interfaces. As a result, implementing drag and drop, sorting, and storing data is trivial. For the vast majority of trees and lists, these two models are sufficient.

Models are accessed on a node/column level of granularity. One can query for the value of a model at a certain node and a certain column on that node. There are two structures used to reference a particular node in a model. They are the <gtk-tree-path> and the <gtk-tree-iter> Most of the interface consists of operations on a <gtk-tree-iter>.

Here, is short for

A path is essentially a potential node. It is a location on a model that may or may not actually correspond to a node on a specific model. The <gtk-tree-path> struct can be converted into either an array of unsigned integers or a string. The string form is a list of numbers separated by a colon. Each number refers to the offset at that level. Thus, the path refers to the root node and the path refers to the fifth child of the third node.

By contrast, a <gtk-tree-iter> is a reference to a specific node on a specific model. It is a generic struct with an integer and three generic pointers. These are filled in by the model in a model-specific way. One can convert a path to an iterator by calling gtk-tree-model-get-iter. These iterators are the primary way of accessing a model and are similar to the iterators used by <gtk-text-buffer>. They are generally statically allocated on the stack and only used for a short time. The model interface defines a set of operations using them for navigating the model.

It is expected that models fill in the iterator with private data. For example, the <gtk-list-store> model, which is internally a simple linked list, stores a list node in one of the pointers. The <gtk-tree-model-sort> stores an array and an offset in two of the pointers. Additionally, there is an integer field. This field is generally filled with a unique stamp per model. This stamp is for catching errors resulting from using invalid iterators with a model.

The lifecycle of an iterator can be a little confusing at first. Iterators are expected to always be valid for as long as the model is unchanged (and doesn't emit a signal). The model is considered to own all outstanding iterators and nothing needs to be done to free them from the user's point of view. Additionally, some models guarantee that an iterator is valid for as long as the node it refers to is valid (most notably the <gtk-tree-store> and <gtk-list-store>). Although generally uninteresting, as one always has to allow for the case

where iterators do not persist beyond a signal, some very important performance enhancements were made in the sort model. As a result, the <gtk-tree-model-iters-persist> flag was added to indicate this behavior.

To help show some common operation of a model, some examples are provided. The first example shows three ways of getting the iter at the location . While the first method shown is easier, the second is much more common, as you often get paths from callbacks.

```
/* Three ways of getting the iter pointing to the location
 */
{
  GtkTreePath *path;
  GtkTreeIter iter;
  GtkTreeIter parent_iter;

  /* get the iterator from a string */
  gtk_tree_model_get_iter_from_string (model, &iter, "3:2:5");

  /* get the iterator from a path */
  path = gtk_tree_path_new_from_string ("3:2:5");
  gtk_tree_model_get_iter (model, &iter, path);
  gtk_tree_path_free (path);


  /* walk the tree to find the iterator */
  gtk_tree_model_iter_nth_child (model, &iter, NULL, 3);
  parent_iter = iter;
  gtk_tree_model_iter_nth_child (model, &iter, &parent_iter, 2);
  parent_iter = iter;
  gtk_tree_model_iter_nth_child (model, &iter, &parent_iter, 5);
}
```

This second example shows a quick way of iterating through a list and getting a string and an integer from each row. The populate-model function used below is not shown, as it is specific to the <gtk-list-store>. For information on how to write such a function, see the <gtk-list-store> documentation.

```
enum
{
  STRING_COLUMN,
  INT_COLUMN,
  N_COLUMNS
};

{
  GtkTreeModel *list_store;
  GtkTreeIter iter;
  gboolean valid;
  gint row_count = 0;

  /* make a new list_store */
  list_store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, G_TYPE_INT);

  /* Fill the list store with data */
  populate_model (list_store);

  /* Get the first iter in the list */
  valid = gtk_tree_model_get_iter_first (list_store, &iter);

  while (valid)
    {
      /* Walk through the list, reading each row */
      gchar *str_data;
      gint   int_data;

      /* Make sure you terminate calls to gtk_tree_model_get()
       * with a '-1' value
```

```
          */
         gtk_tree_model_get (list_store, &iter,
                             STRING_COLUMN, &str_data,
                             INT_COLUMN, &int_data,
                             -1);

         /* Do something with the data */
         g_print ("Row %d: (%s,%d)\n", row_count, str_data, int_data);
         g_free (str_data);

         row_count ++;
         valid = gtk_tree_model_iter_next (list_store, &iter);
       }
   }
```

32.2 Usage

— Class: **<gtk-tree-model>**

Derives from `<ginterface>`.

This class defines no direct slots.

— Signal on <gtk-tree-model>: **row-changed** (*arg0* `<gtk-tree-path>`) (*arg1* `<gtk-tree-iter>`)

This signal is emitted when a row in the model has changed.

— Signal on <gtk-tree-model>: **row-inserted** (*arg0* `<gtk-tree-path>`) (*arg1* `<gtk-tree-iter>`)

This signal is emitted when a new row has been inserted in the model.

Note that the row may still be empty at this point, since it is a common pattern to first insert an empty row, and then fill it with the desired values.

— Signal on <gtk-tree-model>: **row-has-child-toggled** (*arg0* `<gtk-tree-path>`) (*arg1* `<gtk-tree-iter>`)

This signal is emitted when a row has gotten the first child row or lost its last child row.

— Signal on <gtk-tree-model>: **row-deleted** (*arg0* `<gtk-tree-path>`)

This signal is emitted when a row has been deleted.

Note that no iterator is passed to the signal handler, since the row is already deleted.

Implementations of GtkTreeModel must emit row-deleted *before* removing the node from its internal data structures. This is because models and views which access and monitor this model might have references on the node which need to be released in the row-deleted handler.

— Signal on <gtk-tree-model>: **rows-reordered** (*arg0* `<gtk-tree-path>`) (*arg1* `<gtk-tree-iter>`) (*arg2* `<gpointer>`)

This signal is emitted when the children of a node in the `<gtk-tree-model>` have been reordered.

Note that this signal is *not* emitted when rows are reordered by DND, since this is implemented by removing and then reinserting the row.

— Class: **<gtk-tree-iter>**

Derives from `<gboxed>`.

This class defines no direct slots.

— Class: **<gtk-tree-path>**

Derives from <gboxed>.

This class defines no direct slots.

— Class: **<gtk-tree-row-reference>**

Derives from <gboxed>.

This class defines no direct slots.

— Function: **gtk-tree-path-new-from-string** (*path* mchars) (*path* mchars) ⇒ (*ret* <gtk-tree-path>)

Creates a new <gtk-tree-path> initialized to *path*. *path* is expected to be a colon separated list of numbers. For example, the string "10:4:0" would create a path of depth 3 pointing to the 11th child of the root node, the 5th child of that 11th child, and the 1st child of that 5th child. If an invalid path string is passed in, '#f' is returned.

path
The string representation of a path.
ret
A newly-created <gtk-tree-path>, or '#f'

— Function: **gtk-tree-path-append-index** (*self* <gtk-tree-path>) (*index* int)

Appends a new index to a path. As a result, the depth of the path is increased.

path
A <gtk-tree-path>.
index
The index.

— Function: **gtk-tree-path-prepend-index** (*self* <gtk-tree-path>) (*index* int)

Prepends a new index to a path. As a result, the depth of the path is increased.

path
A <gtk-tree-path>.
index
The index.

— Function: **gtk-tree-path-copy** (*self* <gtk-tree-path>) ⇒ (*ret* <gtk-tree-path>)

Creates a new <gtk-tree-path> as a copy of *path*.

path
A <gtk-tree-path>.
ret
A new <gtk-tree-path>.

— Function: **gtk-tree-row-reference-new** (*model* <gtk-tree-model>) (*path* <gtk-tree-path>) ⇒ (*ret* <gtk-tree-row-reference>)

Creates a row reference based on *path*. This reference will keep pointing to the node pointed to by *path*, so long as it exists. It listens to all signals emitted by *model*, and updates its path appropriately. If *path* isn't a valid path in *model*, then '#f' is returned.

model
A <gtk-tree-model>

path
: A valid `<gtk-tree-path>` to monitor

ret
: A newly allocated `<gtk-tree-row-reference>`, or '`#f`'

— Function: **gtk-tree-row-reference-new-proxy** (*proxy* `<gobject>`) (*model* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`) ⇒ (*ret* `<gtk-tree-row-reference>`)

You do not need to use this function. Creates a row reference based on *path*. This reference will keep pointing to the node pointed to by *path*, so long as it exists. If *path* isn't a valid path in *model*, then '`#f`' is returned. However, unlike references created with `gtk-tree-row-reference-new`, it does not listen to the model for changes. The creator of the row reference must do this explicitly using `gtk-tree-row-reference-inserted`, `gtk-tree-row-reference-deleted`, `gtk-tree-row-reference-reordered`.

These functions must be called exactly once per proxy when the corresponding signal on the model is emitted. This single call updates all row references for that proxy. Since built-in GTK+ objects like `<gtk-tree-view>` already use this mechanism internally, using them as the proxy object will produce unpredictable results. Further more, passing the same object as *model* and *proxy* doesn't work for reasons of internal implementation.

This type of row reference is primarily meant by structures that need to carefully monitor exactly when a row reference updates itself, and is not generally needed by most applications.

proxy
: A proxy `<gobject>`

model
: A `<gtk-tree-model>`

path
: A valid `<gtk-tree-path>` to monitor

ret
: A newly allocated `<gtk-tree-row-reference>`, or '`#f`'

— Function: **gtk-tree-row-reference-get-model** (*self* `<gtk-tree-row-reference>`) ⇒ (*ret* `<gtk-tree-model>`)

Returns the model that the row reference is monitoring.

reference
: A `<gtk-tree-row-reference>`

ret
: the model

Since 2.8

— Function: **gtk-tree-row-reference-get-path** (*self* `<gtk-tree-row-reference>`) ⇒ (*ret* `<gtk-tree-path>`)

Returns a path that the row reference currently points to, or '`#f`' if the path pointed to is no longer valid.

reference
: A `<gtk-tree-row-reference>`

ret
: A current path, or '`#f`'.

— Function: **gtk-tree-row-reference-valid** (*self* `<gtk-tree-row-reference>`) ⇒ (*ret* `bool`)

Returns '`#t`' if the *reference* is non-'`#f`' and refers to a current valid path.

reference
: A `<gtk-tree-row-reference>`, or '`#f`'

ret
: '`#t`' if *reference* points to a valid path.

— Function: **gtk-tree-row-reference-inserted** (*proxy* `<gobject>`) (*path* `<gtk-tree-path>`)

Lets a set of row reference created by `gtk-tree-row-reference-new-proxy` know that the model emitted the "row_inserted" signal.

proxy
: A `<gobject>`

path
: The row position that was inserted

— Function: **gtk-tree-row-reference-deleted** (*proxy* `<gobject>`) (*path* `<gtk-tree-path>`)

Lets a set of row reference created by `gtk-tree-row-reference-new-proxy` know that the model emitted the "row_deleted" signal.

proxy
: A `<gobject>`

path
: The path position that was deleted

— Function: **gtk-tree-row-reference-reordered** (*proxy* `<gobject>`) (*path* `<gtk-tree-path>`) (*iter* `<gtk-tree-iter>`) ⇒ (*new_order* `int`)

Lets a set of row reference created by `gtk-tree-row-reference-new-proxy` know that the model emitted the "rows_reordered" signal.

proxy
: A `<gobject>`

path
: The parent path of the reordered signal

iter
: The iter pointing to the parent of the reordered

new-order
: The new order of rows

— Function: **gtk-tree-iter-copy** (*self* `<gtk-tree-iter>`) ⇒ (*ret* `<gtk-tree-iter>`)

Creates a dynamically allocated tree iterator as a copy of *iter*. This function is not intended for use in applications, because you can just copy the structs by value ('`GtkTreeIter new_iter = iter;`'). You must free this iter with `gtk-tree-iter-free`.

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

ret
: a newly-allocated copy of *iter*.

— Function: **gtk-tree-model-get-flags** (*self* `<gtk-tree-model>`) ⇒ (*ret* `<gtk-tree-model-flags>`)
— Method: **get-flags**

Returns a set of flags supported by this interface. The flags are a bitwise combination of `<gtk-tree-model-flags>`. The flags supported should not change during the lifecycle of the *tree-model*.

tree-model

A `<gtk-tree-model>`.

ret
The flags supported by this interface.

— Function: **gtk-tree-model-get-n-columns** (*self* `<gtk-tree-model>`) ⇒ (*ret* `int`)
— Method: **get-n-columns**

Returns the number of columns supported by *tree-model*.

tree-model
A `<gtk-tree-model>`.
ret
The number of columns.

— Function: **gtk-tree-model-get-column-type** (*self* `<gtk-tree-model>`) (*index* `int`) ⇒ (*ret* `<gtype>`)
— Method: **get-column-type**

Returns the type of the column.

tree-model
A `<gtk-tree-model>`.
index
The column index.
ret
The type of the column.

— Function: **gtk-tree-model-get-iter** (*self* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **get-iter**

Sets *iter* to a valid iterator pointing to *path*.

tree-model
A `<gtk-tree-model>`.
iter
The uninitialized `<gtk-tree-iter>`.
path
The `<gtk-tree-path>`.
ret
‘`#t`’, if *iter* was set.

— Function: **gtk-tree-model-get-iter-first** (*self* `<gtk-tree-model>`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **get-iter-first**

Initializes *iter* with the first iterator in the tree (the one at the path "0") and returns ‘`#t`’. Returns ‘`#f`’ if the tree is empty.

tree-model
A `<gtk-tree-model>`.
iter
The uninitialized `<gtk-tree-iter>`.
ret
‘`#t`’, if *iter* was set.

— Function: **gtk-tree-model-get-path** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`) ⇒ (*ret* `<gtk-tree-path>`)
— Method: **get-path**

Returns a newly-created <gtk-tree-path> referenced by *iter*. This path should be freed with gtk-tree-path-free.

tree-model
: A <gtk-tree-model>.

iter
: The <gtk-tree-iter>.

ret
: a newly-created <gtk-tree-path>.

— Function: **gtk-tree-model-get-value** (*self* <gtk-tree-model>) (*iter* <gtk-tree-iter>) (*column* int) ⇒ (*ret* scm)
— Method: **get-value**

Sets initializes and sets *value* to that at *column*. When done with *value*, g-value-unset needs to be called to free any allocated memory.

tree-model
: A <gtk-tree-model>.

iter
: The <gtk-tree-iter>.

column
: The column to lookup the value at.

value
: An empty <gvalue> to set.

— Function: **gtk-tree-model-iter-next** (*self* <gtk-tree-model>) (*iter* <gtk-tree-iter>) ⇒ (*ret* <gtk-tree-iter>)
— Method: **iter-next**

Sets *iter* to point to the node following it at the current level. If there is no next *iter*, '#f' is returned and *iter* is set to be invalid.

tree-model
: A <gtk-tree-model>.

iter
: The <gtk-tree-iter>.

ret
: '#t' if *iter* has been changed to the next node.

— Function: **gtk-tree-model-iter-children** (*self* <gtk-tree-model>) (*parent* <gtk-tree-iter>) ⇒ (*ret* glist-of)
— Method: **iter-children**

Sets *iter* to point to the first child of *parent*. If *parent* has no children, '#f' is returned and *iter* is set to be invalid. *parent* will remain a valid node after this function has been called.

If *parent* is '#f' returns the first node, equivalent to 'gtk_tree_model_get_iter_first (tree_model, iter);'

tree-model
: A <gtk-tree-model>.

iter
: The new <gtk-tree-iter> to be set to the child.

parent
: The <gtk-tree-iter>, or '#f'

ret
: '#t', if *child* has been set to the first child.

— Function: **gtk-tree-model-iter-has-child** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`) ⇒ (*ret* `bool`)
— Method: **iter-has-child**

Returns '`#t`' if *iter* has children, '`#f`' otherwise.

tree-model
A `<gtk-tree-model>`.
iter
The `<gtk-tree-iter>` to test for children.
ret
'`#t`' if *iter* has children.

— Function: **gtk-tree-model-iter-n-children** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`) ⇒ (*ret* `int`)
— Method: **iter-n-children**

Returns the number of children that *iter* has. As a special case, if *iter* is '`#f`', then the number of toplevel nodes is returned.

tree-model
A `<gtk-tree-model>`.
iter
The `<gtk-tree-iter>`, or '`#f`'.
ret
The number of children of *iter*.

— Function: **gtk-tree-model-iter-nth-child** (*self* `<gtk-tree-model>`) (*parent* `<gtk-tree-iter>`) (*n* `int`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **iter-nth-child**

Sets *iter* to be the child of *parent*, using the given index. The first index is 0. If *n* is too big, or *parent* has no children, *iter* is set to an invalid iterator and '`#f`' is returned. *parent* will remain a valid node after this function has been called. As a special case, if *parent* is '`#f`', then the *n*th root node is set.

tree-model
A `<gtk-tree-model>`.
iter
The `<gtk-tree-iter>` to set to the nth child.
parent
The `<gtk-tree-iter>` to get the child from, or '`#f`'.
n
Then index of the desired child.
ret
'`#t`', if *parent* has an *n*th child.

— Function: **gtk-tree-model-iter-parent** (*self* `<gtk-tree-model>`) (*child* `<gtk-tree-iter>`) ⇒ (*ret* `<gtk-tree-iter>`)
— Method: **iter-parent**

Sets *iter* to be the parent of *child*. If *child* is at the toplevel, and doesn't have a parent, then *iter* is set to an invalid iterator and '`#f`' is returned. *child* will remain a valid node after this function has been called.

tree-model
A `<gtk-tree-model>`
iter
The new `<gtk-tree-iter>` to set to the parent.
child
The `<gtk-tree-iter>`.

ret
: '`#t`', if *iter* is set to the parent of *child*.

— Function: **gtk-tree-model-get-string-from-iter** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`) ⇒ (*ret* `mchars`)
— Method: **get-string-from-iter**

Generates a string representation of the iter. This string is a ':' separated list of numbers. For example, "4:10:0:3" would be an acceptable return value for this string.

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

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

ret
: A newly-allocated string. Must be freed with `g-free`.

Since 2.2

— Function: **gtk-tree-model-ref-node** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`)
— Method: **ref-node**

Lets the tree ref the node. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons.

This function is primarily meant as a way for views to let caching model know when nodes are being displayed (and hence, whether or not to cache that node.) For example, a file-system based model would not want to keep the entire file-hierarchy in memory, just the sections that are currently being displayed by every current view.

A model should be expected to be able to get an iter independent of its reffed state.

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

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

— Function: **gtk-tree-model-unref-node** (*self* `<gtk-tree-model>`) (*iter* `<gtk-tree-iter>`)
— Method: **unref-node**

Lets the tree unref the node. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons.

For more information on what this means, see `gtk-tree-model-ref-node`. Please note that nodes that are deleted are not unreffed.

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

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

— Function: **gtk-tree-model-row-changed** (*self* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`) (*iter* `<gtk-tree-iter>`)
— Method: **row-changed**

Emits the "row_changed" signal on *tree-model*.

tree-model
: A `<gtk-tree-model>`

path
: A `<gtk-tree-path>` pointing to the changed row

iter
: A valid `<gtk-tree-iter>` pointing to the changed row

— Function: **gtk-tree-model-row-inserted** (*self* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`) (*iter* `<gtk-tree-iter>`)
— Method: **row-inserted**

Emits the "row_inserted" signal on *tree-model*

tree-model
: A `<gtk-tree-model>`

path
: A `<gtk-tree-path>` pointing to the inserted row

iter
: A valid `<gtk-tree-iter>` pointing to the inserted row

— Function: **gtk-tree-model-row-deleted** (*self* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`)
— Method: **row-deleted**

Emits the "row_deleted" signal on *tree-model*. This should be called by models after a row has been removed. The location pointed to by *path* should be the location that the row previously was at. It may not be a valid location anymore.

tree-model
: A `<gtk-tree-model>`

path
: A `<gtk-tree-path>` pointing to the previous location of the deleted row.

— Function: **gtk-tree-model-rows-reordered** (*self* `<gtk-tree-model>`) (*path* `<gtk-tree-path>`) (*iter* `<gtk-tree-iter>`) ⇒ (*new_order* `int`)
— Method: **rows-reordered**

Emits the "rows_reordered" signal on *tree-model*. This should be called by models when their rows have been reordered.

tree-model
: A `<gtk-tree-model>`

path
: A `<gtk-tree-path>` pointing to the tree node whose children have been reordered

iter
: A valid `<gtk-tree-iter>` pointing to the node whose children have been reordered, or ‘`#f`’ if the depth of *path* is 0.

new-order
: an array of integers mapping the current position of each child to its old position before the re-ordering, i.e. *new-order*‘`[newpos] = oldpos`’.