testmanager Documentation

Release 0.1

Javier Collado

November 03, 2011

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Test Manager is a tool to not only manage test cases, but also run them, review execution results and submit them.

CHAPTER

TWO

REFERENCE

`testmanager` package implements all the functionality required by the `testmanager` application. In particular, it contains packages and modules to drive the GTK interface and handle the testing information needed in a testing project.

2.1 application

`testmanager` main application module

class `testmanager.application.`**Application**
Application implementation

main()
Run GTK main loop

Returns None

open_cb(*widget*)
Load test data

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

file_opened_cb(*file*)
Set widget sensitivity properly

Parameters **file** (`ApplicationFile`) – The file that emitted the `opened` signal

Returns None

close_cb(*widget*)
Close test data

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

file_closed_cb(*file*)
Reset application state

Parameters **file** (`ApplicationFile`) – The file that emitted the `opened` signal

Returns None

save_cb (*widget*)
Save changes to opened file

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

save_as_cb (*widget*)
Save changes to a new file

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

file_saved_cb (*file*)
Set widget sensitivity properly

Parameters **file** (`ApplicationFile`) – The file that emitted the `saved` signal

Returns None

revert_cb (*widget*)
Revert changes

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

quit_cb (*widget*)
Exit from application

Parameters **widget** (*Gtk.ToolButton* | *Gtk.MenuItem*) – Widget that emitted the `clicked` or the `activate` signal

Returns None

window_destroy_cb (*window*)
Quit from application

Parameters **window** (*gkt.Window*) – Main application window

Returns None

name_server_cb (*menuitem*)
Check name server status

Parameters **menuitem** (`Gtk.MenuItem`) – The menuitem that emitted the `activated` signal

Returns None

class `testmanager.application.`**ApplicationFile**
ApplicationFile object takes care of loading/saving data from/to a file

open (*filename*)
Load data from a file encoded in json format

Once the data has been loaded, the `opened` signal is emitted.

Parameters **filename** (*str*) – File to open

Returns None

save (*filename*, *data*)
Save data to a file encoded in json format

Once the data has been loaded, the `saved` signal is emitted.

Parameters

- **filename** (*str*) – File to write
- **data** (*dict*) – Data to save

Returns None

revert ()
Revert changes

Since filename and data are cached, data file isn't reloaded, but just the `opened` signal is emitted so that widgets are updated with the stored data.

Returns None

close ()
Close file and reset internal data

Filename and data are set to None to make clear that no file is opened and the `closed` signal i emitted so that widgets listening to it can be updated.

Returns None

2.2 dialog

Dialogs used in the application

class `testmanager.dialog.`**Dialog** (*parent*, *ui_filename*)
Dialog with an interface defined in a .ui file and that can be run as a context manager

destroy_cb (*dialog*)
Disconnect all signal handlers

Parameters **dialog** (`Gtk.Dialog`) – The dialog that emitted the `destroy` signal

Returns None

class `testmanager.dialog.`**NameServerDialog** (*parent*, *name_server*)
Name server dialog to check name server status

response_cb (*dialog*, *response_id*)
Handle dialog buttons activity

ResponseType.APPLY is the response id assigned to the refresh button so when that happens, a request to refresh the name server information will be made.

For all other responses, the dialog will be destroyed.

Parameters

- **dialog** (`Gtk.Dialog`) – The dialog that emitted the `response` signal.
- **response_id** – Id for the button that was clicked

Returns None

name_server_cb (*name_server*, **args*)
Update widget contents for every signal emitted by the `NameServer` object.

Parameters

- **name_server** (NameServer) – Name server that emitted the signal
- **args** (*list*) – Additional arguments that depend on the signal emitted

Returns None

class testmanager.dialog.**ServerDialog**(*parent*, *application*)
Server dialog to select on which hardware run a test case

selected_server
Get selected server name

Returns Selected server name

Return type str

selection_changed_cb(*selection*)
Set ok button sensitivity

Parameters **selection** (Gtk.TreeSelection) – Treeview selection that emitted the selection-changed signal

Returns None

row_activated_cb(*treeview*, *path*, *view_column*)
Run test case on selected server if available

Parameters

- **treeview** (Gtk.TreeView) – The treeview that emitted the row-activated signal
- **path** (*tuple*) – Path to the activated row
- **view_column** – Column that was activated in the row

2.3 history

History package contains all the modules needed to handle the history of the actions taken by the user to modify test object data

class testmanager.history.**History**(*application*)
History object captures signals with information to undo/redo user actions gracefully

all_rows_added_to_plan(*tab*, *paths_added*)
Keep action information regarding all rows added to a test plan

Parameters

- **tab** (PlanTab) – Tab with the test plan information
- **paths_added** (list(tuple(tuple(int), KeyPath))) – Paths to the test rows added

Returns None

all_rows_removed_from_plan(*tab*, *paths_removed*)
Keep action information regarding rows removed from a test plan

Parameters

- **tab** (PlanTab) – Tab with the test plan information
- **paths_removed** (list(tuple(tuple(int), KeyPath))) – Paths to the test rows removed

Returns None

block_handlers(**args*, ***kwds*)
Block signal handlers in a block of code

This is used to prevent new actions from being added to history when they happen as a result of clicking on undo/redo buttons

Returns None

redo_cb(*widget*, *how_many=1*)
Redo last undone action

Parameters

- **button** (`gtk.MenuToolButton` | `gtk.MenuItem`) – The widget that emitted the `clicked` or `activate` signal
- **how_many** (*int*) – Number of actions to be redone

Returns None

row_added_cb(*model*, *row*)
Keep a record about the added row action

Parameters

- **model** (`TreeStore` (any of its subclasses)) – The model that emitted the `row-added` signal
- **row** (*tuple*) – The contents of the row that was removed

Returns None

row_removed_cb(*model*, *path*, *key_path*, *row_tuple*, *child_row_tuples*)
Keep a record about the removed row action

Parameters

- **model** (`TreeStore` (any of its subclasses)) – The model that emitted the `row-removed` signal
- **path** (*tuple*) – The path to the row that was removed
- **key_path** (`KeyPath`) – The key path to the row that was removed
- **row_tuple** (*tuple*) – The contents of the row that was removed
- **child_row_tuples** (*list*) – Contents of the removed children rows

Returns None

rows_added_to_plan(*tab*, *paths_added*)
Keep action information regarding rows added to a test plan

Parameters

- **tab** (`PlanTab`) – Tab with the test plan information
- **paths_added** (list(tuple(tuple(int), `KeyPath`))) – Paths to the test rows added

Returns None

rows_removed_from_plan(*tab*, *paths_removed*)
Keep action information regarding all rows removed from a test plan

Parameters

- **tab** (PlanTab) – Tab with the test plan information
- **paths_removed** (list(tuple(tuple(int), KeyPath))) – Paths to the test rows removed

Returns None

tab_attribute_changed(*tab*, *attribute_name*, *old_value*, *new_value*)
Keep action information regarding attributes changed in a tab

Parameters

- **tab** (Tab (any of its subclasses)) – Tab with the test plan information
- **attribute_name** (*str*) – Name of the changed attribute
- **old_value** (*object*) – Old attribute value
- **new_value** (*object*) – New attribute value

Returns None

undo_cb(*widget*, *how_many=1*)
Undo last action

Parameters

- **button** (gtk.MenuToolButton | gtk.MenuItem) – The widget that emitted the clicked or activate signal
- **how_many** (*int*) – Number of actions to be undone

Returns None

2.3.1 action

Action package contains all the action class definitions needed to undo/redo specific actions on test object data

class testmanager.history.action.**Actions**(*application*, *undo_cb*, *redo_cb*)
Actions object keeps a record of all the actions that can be undone/redone

append(*action*)
Append action to the list

Parameters **action** (Action (any of its subclasses)) – Action captured to be recorded

Returns None

clear()
Clear all actions

Returns None

file_closed_cb(*file*)
Clear history and filename information

Parameters **file** (ApplicationFile) – File that emitted the closed signal

Returns None

file_opened_cb(*file*)
Clear history information and keep filename

Parameters **file** (ApplicationFile) – File that emitted the opened signal

Returns None

file_saved_cb(*file*)
Set last saved action index

Parameters **file** (ApplicationFile) – File that emitted the saved signal

Returns None

redo()
Redo last action

When the action is redone, no other actions can be added. This means that any event that happens when the action is redone is ignored and a new action is not added.

Returns None

undo()
Undo last action

When the action is undone, no other actions can be added. This means that any event that happens when the action is undone is ignored and a new action is not added.

Returns None

class testmanager.history.action.**Action**
Action objects keep information about a single user action and are able to undo/redo it

model

TreeStore undo/redo for row addition/removal actions

class testmanager.history.action.model.**RowAdded**(*model*, *row*)
Action objects that are able remove (undo) or add (redo) the added row as required

undo()
Undo row add action

Returns None

redo()
Redo row add action

Returns None

class testmanager.history.action.model.**RowRemoved**(*model*, *path*, *key_path*, *row_tuple*, *child_row_tuples*)
Action objects that are able to add (undo) or remove (redo) the removed row as required

undo()
Undo row remove action

Returns None

redo()
Redo row remove action

Returns None

tab

Plan tab undo/redo for selection actions

class testmanager.history.action.tab.**TabAction**(*tab*)
Action objects that are able to undo/redo changes to tabs

open_tab()
Open tab with the test plan information

Returns Plan tab

Return type PlanTab

class testmanager.history.action.tab.**TabAttributeChanged**(*tab*, *attribute_name*, *old_value*, *new_value*)
Action objects that are able to undo/redo changes to tab attributes

undo()
Undo tab attribute change

Returns None

redo()
redo tab attribute change

Returns None

class testmanager.history.action.tab.**PlanRowAction**(*tab*, *test_paths*)
Action objects that are able to add/remove the selected rows to/from the desired test plan

add()
Select test rows in the tab and add them to the plan

Returns None

remove()
Select test row in the tab and remove them from the plan

Returns None

class testmanager.history.action.tab.**RowsAddedToPlan**(*tab*, *test_paths*)
Action objects that are able to remove (undo) or add (redo) the added rows to the desired plan as required

undo()
Undo row(s) addition to test plan

Returns None

redo()
Redo row(s) addition to test plan

Returns None

class testmanager.history.action.tab.**AllRowsAddedToPlan**(*tab*, *test_paths*)
Action objects that are able to remove (undo) or add (redo) all the added rows to the desired plan as required

class testmanager.history.action.tab.**RowsRemovedFromPlan**(*tab*, *test_paths*)
Action objects that are able to add (undo) or removed (redo) the added rows to the desired plan as required

undo()
Undo row(s) removal from test plan

Returns None

redo()
Redo row(s) removal from test plan

Returns None

class testmanager.history.action.tab.**AllRowsRemovedFromPlan**(*tab*, *test_paths*)
Action objects that are able to add (undo) or remove (redo) all the removed rows to the desired plan as required

2.4 networking

Networking modules to interact with Pyro4 objects

class testmanager.networking.**NameServer**
Name server proxy wrapper

start()
Get name_server connection data and start refreshing it periodically

Returns None

refresh()
Refresh connection information if no other update is already running

This method can be used to update data immediately by calling it directly or periodically by calling it through GLib.timeout_add* functions. In such a case, it will be executed forever since it always returns True.

Returns True

Return type bool

class testmanager.networking.**Server**
A server that takes care of running test cases and returning results back to the client

get_status()
Get server status

Returns Server status

Return type str

run(*key_path*, *data*)
Run test case

Parameters

- **key_path** (KeyPath) – Key path to the test row containing the test case. In practice, this is the full unique name for the test.
- **data** (*dict*) – Test case description data

Returns Test case execution results

Return type dict

2.5 notebook

Notebook package contains the class definitions needed to work with a notebook and their tabs

class testmanager.notebook.**Notebook**(*application*)
Gtk.Notebook wrapper that takes care of opening/closing tabs that display the contents of a row in any of the application models.

New tabs are opened when a row is activated in the treeview and closed when the tab close button is clicked or when the row that it displays is removed from the model

close_tab(*tab*)
Close an existing tab

Parameters **tab** (Tab (any of its subclasses)) – The tab to be closed

Returns None

find_tab(*row*)
Look for an already opened tab displaying a row

Parameters **row** (`TreeModelRowWrapper`) – Row displayed by the tab

Returns tab displaying row contents

Return type `Tab` (any of its subclasses)

open_tab(*row*)
Display row contents in a tab. A new tab will be created if the row is not being displayed in any existing tab. Otherwise, the tab will be just focused.

Parameters **row** (`TreeModelRowWrapper`) – Row to be displayed

Returns The tab created/focused

Return type `Tab` (any of its subclasses)

pre_row_removed_cb(*store*, *row*)
Close tab showing data for a row that no longer exists

Parameters

- **store** (`TreeStore` (any of its subclasses)) – Store where the row was deleted
- **row** (`TreeModelRowWrapper`) – Row to be removed

row_activated_cb(*treeview*, *path*, *view_column*)
Open tab for activated row so that its contents can be edited

Parameters

- **treeview** (`ToolbarTreeView` (any of its subclasses)) – The treeview for which the row was activated (= self)
- **path** (*str* | *int* | *tuple*) – The path in the model for the activated row
- **view_column** (`Gtk.TreeViewColumn`) – The column that was activated in the row (unused)

Returns None

2.5.1 tab

Tab module contains the class definitions needed to work with all kind of test object related tabs in testmanager

class `testmanager.notebook.tab.`**Tab**(*row_reference*, *application*)
Notebook tab with a close button

Tab shows the contents of a TreeView row

close_cb(*button*)
Close tab and save changes to row

Parameters **button** (`Gtk.Button`) – Button that emitted the `clicked` signal

static create(*tab_type*, *row_reference*, *application*)
Factory method to create tabs using the most suitable class

Parameters

- **tab_type** (*str*) – The type of tab to be created

- **row_reference** (`TreeRowReference`) – Reference to the row to be displayed
- **treeview** (`ToolbarTreeView`) – Treeview displaying the name of the row

Returns New tab object

Return type `Tab` (any of its subclasses)

Raises `ValueError` when `tab_type` is not valid

destroy()
Save changes in tab to the model that contains the displayed row, disconnect all signal handlers and destroy widgets

Once the tab is destroyed, an attempt call them will still be made by `Gtk` when their associated signals are emitted. To avoid this, the handlers are cleanly disconnected.

Returns None

save()
Save contents of the widgets in the tab to the row in the store

Each subclass must implement this method to save any additional information not displayed in other tabs

title
Get tab title

Returns Tab title

Return type str

class testmanager.notebook.tab.**LeftTab**(*row_reference*, *application*)
`Tab` that shows the contents of a `LeftTreeStore` row

description
Get description field

Returns Description field

Return type str

description_changed_cb(*textbuffer*)
Validate description, update it and save change to history

Parameters **textbuffer** (`Gtk.TextBuffer`) – The textbuffer that emitted the `changed` signal

Returns None

name
Get name field

Returns None

name_changed_cb(*name_entry*)
Validate name, update it and save changes to history

Parameters **name_entry** (`Gtk.Entry`) – The entry widget that emitted the `changed` signal

Returns None

remove_cb(*button*)
Remove row displayed by this tab. The tab itself will be automatically closed after the row is removed

Parameters **button** (`Gtk.Button`) – Button that emitted the `clicked` signal

Returns None

save()
Save contents of the widgets in the tab to the row in the store

Each subclass must implement this method to save any additional information not displayed in other tabs

class testmanager.notebook.tab.**SuiteTab**(*row_reference*, *application*)
LeftTab that displays test suite data

add_case_cb(*button*)
Add a test case under the suite displayed by this tab

Parameters **button** (*Gtk.Button*) – Button that emitted the clicked signal

Returns None

add_suite_cb(*button*)
Add a test suite under the one displayed by this tab

Parameters **button** (*Gtk.Button*) – Button that emitted the clicked signal

Returns None

class testmanager.notebook.tab.**CaseTab**(*row_reference*, *application*)
Tab that displays test case data

destroy()
Destroy tab widgets and all cached widgets from runners UI

Returns None

runner
Get runner field

Returns Runner field

Return type str

runner_changed_cb(*combobox*)
Validate runner, update it and save change to history

Parameters **combobox** (Gtk.ComboBox) – Combobox that emitted the changed signal

Returns None

save()
Save contents of the widgets in the tab to the row in the store

Returns None

class testmanager.notebook.tab.**HardwareTab**(*row_reference*, *application*)
Tab that displays hardware data

add_inventory_cb(*button*)
Add an inventory under the hardware displayed by this tab

Parameters **button** (gtk.Button) – Button that emitted the clicked signal

Returns None

name_changed_cb(*name_entry*)
Validate name, update it and save changes to history

Parameters **name_entry** (gtk.Entry) – The entry widget that emitted the changed signal

Returns None

name_server_cb(*name_server*, **args*)
Update status lavel for every signal emitted by the NameServer object.

Parameters

- **name_server** (NameServer) – Name server that emitted the signal
- **args** (*list*) – Additional arguments that depend on the signal emitted

Returns None

class testmanager.notebook.tab.**InventoryTab**(*row_reference*, *application*)
LeftTab that displays hardware inventory data

class testmanager.notebook.tab.**PlanTab**(*row_reference*, *application*)
Tab that displays test plan data

add_all_button_cb(*button*)
Move all tests from unselected_tests to selected_tests

Parameters **button** (Gtk.Button) – The button that emitted the clicked signal

Returns None

add_button_cb(*button*)
Move selected rows from unselected_tests to selected_tests and add action to application history

Parameters **button** (Gtk.Button) – The button that emitted the clicked signal

Returns None

add_run_cb(*button*)
Add a test suite under the one displayed by this tab

Parameters **button** (Gtk.Button) – Button that emitted the clicked signal

Returns None

add_selected_rows()
Move selected rows from unselected_tests to selected_tests

Returns Path information of the moved rows (both path and key path)

Return type list(tuple(tuple, KeyPath))

remove_all_button_cb(*button*)
Move all tests from selected_tests to unselected_tests

Parameters **button** (Gtk.Button) – The button that emitted the clicked signal

Returns None

remove_button_cb(*button*)
Move selected tests from selected_tests to unselected_tests

Parameters **button** (Gtk.Button) – The button that emitted the clicked signal

Returns None

remove_selected_rows()
Move selected rows from selected_tests to unselected_tests

Returns Key path of the rows that were moved

Return type list(KeyPath)

row_activated_cb(*treeview*, *path*, *view_column*)
Open test tab for row being activated

Parameters

- **treeview** (PlanTestSelectionTreeView) – Treeview that emitted the row-activated signal
- **path** (*tuple*) – Path to the activated row
- **view_column** – Column that was activated in the row

save()
Save contents of the widgets in the tab to the row in the store

Returns None

test_pre_row_removed_cb(*store*, *row*)
Rows removed from all_tests are also removed from unselected/selected_test

Parameters

- **store** (TestsTreeStore) – Store where the row was removed
- **row** (TreeModelRowWrapper) – Row to be removed

Returns None

test_row_added_cb(*store*, *row*)
Rows added to all_tests are also added to unselected_tests

Parameters

- **store** (TestsTreeStore) – Store where the row was added
- **row** (TreeModelRowWrapper) – Row object just added

Returns None

whitelist
Get a list of selected tests as displayed in the tab

Returns Whitelist containing all selected tests

Return type list

class testmanager.notebook.tab.**RunTab**(*row_reference*, *application*)
Tab that displays test run results

execute_cb(*button*)
Execute selected test

Parameters **button** (Gtk.Button) – Button that emitted the clicked signal

Returns None

results
Get results as displayed in the tab

Returns Results for all test cases in the test run

Return type list

row_activated_cb(*treeview*, *path*, *view_column*)
Open tab for selected object

Parameters

- **treeview** (Gtk.TreeView) – Treeview that emitted the row-activated signal
- **path** (*tuple*) – Path to the activated row

- **view_column** – Column that was activated in the row

save()
Save contents of the widgets in the tab to the row in the store

Returns None

selection_changed_cb(*selection*)
Set execute button sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the `selection-changed` signal

Returns None

test_pre_row_removed_cb(*store*, *test_row*)
Make sure that missing rows are marked as such

Parameters

- **store** (`TestsTreeStore`) – Store where the row was added
- **row** (`TreeModelRowWrapper`) – Row to be removed

Returns None

test_row_added_cb(*store*, *test_row*)
Make sure that existing rows aren't marked as missing

Parameters

- **store** (`TestsTreeStore`) – Store where the row was added
- **row** (`TreeModelRowWrapper`) – Row object just added

Returns None

class `testmanager.notebook.tab.`**ResultTab**(*row_reference*, *application*)
`Tab` that displays test case execution results

save()
This method doesn't save any change, because the widgets in this tab don't support any modification yet

Returns None

2.6 tree

`tree` package contains all the tree related modules needed to deal with tree-like data in a `gtk` application

2.6.1 view

`Gtk.TreeView` specialized classes

class `testmanager.tree.view.`**TreeView**(*treeview_name*, *store*)
`Gtk.TreeView` wrapper

model
Get model used by this view

Returns model

Return type `TreeStore`

class testmanager.tree.view.**ToolbarTreeView**(*treeview_type*, *store*, *application*)
TreeView wrapper that takes care of the buttons in the toolbar next to the treeview

static create(*treeview_type*, *store*, *application*)
Factory method used to create treeviews using the most suitable class

Parameters

- **treeview_type** (*str (tests|resources|plans)*) – The type of treeview to be created
- **store** (TestsTreeStore | ResourcesTreeStore | PlansTreeStore) – Store that contains the data to display
- **application** (Application) – The testmanager application

Returns New treeview object

Return type TestsTreeView | ResourcesTreeView | PlansTreeView

Raises ValueError when treeview_type is not valid

selection_changed_cb(*selection*)
Set treeview toolbar buttons sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the selection-changed signal

Returns None

add_cb(*toolbutton*)
Add a new row with default values

Parameters **toolbutton** (*Gtk.ToolButton*) – Toolbutton that emitted the clicked signal

Returns None

edit_cb(*toolbutton*)
Edit selected row contents in a tab

Parameters **toolbutton** (*Gtk.ToolButton*) – Toolbutton that emitted the clicked signal

Returns None

remove_cb(*toolbutton*)
Remove selected row and any related open tab

Parameters **toolbutton** (*Gtk.ToolButton*) – Toolbutton that emitted the clicked signal

Returns None

row_inserted_cb(*store*, *path*, *iterator*)
Expand treeview to display rows being added

Parameters

- **store** (TestsTreeStore | ResourcesTreeStore | PlansTreeStore) – Store where the row was inserted
- **path** (*str* | *int* | *tuple*) – Path in the store to the inserted row
- **iterator** (Gtk.TreeIter) – Iterator pointing to the inserted row

Returns None

row_deleted_cb(*store*, *path*)
Make sure toolbar buttons sensitivity is correctly set when rows are deleted

When rows are deleted, selection does not always fire the 'changed' event, so here the callback is explicitly called to avoid problems with the toolbar buttons sensitivity

Parameters

- **store** (`TreeStore` (any of its subclasses)) – Store where the row was deleted
- **path** (*str* | *int* | *tuple*) – Path in the store to the deleted row

class `testmanager.tree.view.`**`TestsTreeView`**(*name*, *store*, *buttons*)
`ToolbarTreeView` that displays test cases hierarchy

`selection_changed_cb`(*selection*)
Set treeview toolbar buttons sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the `selection-changed` signal

Returns None

Raises `ValueError` when selected row type is not valid

class `testmanager.tree.view.`**`ResourcesTreeView`**(*name*, *store*, *buttons*)
`ToolbarTreeView` that displays resources hierarchy

`selection_changed_cb`(*selection*)
Set treeview toolbar buttons sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the `selection-changed` signal

Returns None

Raises `ValueError` when selected row type is not valid

class `testmanager.tree.view.`**`PlansTreeView`**(*treeview_name*, *store*, *application*)
`ToolbarTreeView` that displays test plans hierarchy

`selection_changed_cb`(*selection*)
Set treeview toolbar buttons sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the `selection-changed` signal

Returns None

Raises `ValueError` when selected row type is not valid

class `testmanager.tree.view.`**`PlanTestSelectionTreeView`**(*treeview_name*, *store*, *tab*)
`TreeView` for selecting tests in a plan tab

`selection_changed_cb`(*selection*)
Set test plan selection buttons sensitivity

Parameters **selection** (*Gtk.TreeSelection*) – Treeview selection that emitted the `selection-changed` signal

Returns None

`row_inserted_cb`(*model*, *path*, *iterator*)
Expand treeview to display rows being added and set test plan selection buttons sensitivity

Parameters

- **store** (`TreeStore`) – Store where the row was inserted
- **path** (*str* | *int* | *tuple*) – Path in the store to the inserted row

- **iterator** (`Gtk.TreeIter`) – Iterator pointing to the inserted row

Returns None

row_deleted_cb(*model*, *path*)
Set test plan selection buttons sensitivity

When rows are deleted, selection does not always fire the 'changed' event, so here the callback is explicitly called to avoid problems with the buttons sensitivity

Parameters

- **store** (`TreeStore`) – Store where the row was deleted
- **path** (*str* | *int* | *tuple*) – Path in the store to the deleted row

2.6.2 row

Tree model row-related classes

class testmanager.tree.row.**TreeModelRowWrapper**(*row*)
`Gtk.TreeModelRow` wrapper with some additional funcionality:

•Rows are compared by their contents

•Attribute access using column name (defined in `TreeStore` object)

next
Get next row in the tree

Returns Next row

Return type `TreeModelRowWrapper`

parent
Get parent row

Returns Parent row or None if there isn't a parent row

Return type `TreeModelRowWrapper` | None

iterchildren()
Iterate through row children

Returns Children rows

Return type iterable(`TreeModelRowWrapper`)

iterancestors()
Iterate through row ancestors

Returns ancestor rows

Return type generator(`TreeModelRowWrapper`)

bfiter(*root=None*)
Breadth first iteration

Returns Rows using breadth first iteration

Return type iterable(`TreeModelRowWrapper`)

dfiter()
Depth first iteration

Returns Rows using depth first iteration

Return type iterable(TreeModelRowWrapper)

reference
Get a reference object for this row

Returns Refernce to the row

Return type TreeRowReferenceWrapper

key_path
Get path created from the key column value of each row in the tree hierarchy needed to get to this row

Returns Path of names to the row

Return type KeyPath

n_children
Get the number of children under this row (not recursive)

Returns The number of child rows

Return type int

n_rows
Get the number of rows under this row (recursive)

Returns The number of rows under this one

Return type int

remove()
Remove row from model

This is just a shortcut to avoid using the following less readable code:

```
row.model.remove(row.iter)
```

Returns None

class testmanager.tree.row.**KeyPath**
A list of keys used to access a tree element by name

To be able to use a key path, a key column index has be defined in the model so that which column should be used to extract the path to a row

All python lists methods apply

parent
Get parent row's KeyPath

Returns Parent row key path

Return type KeyPath | None

class testmanager.tree.row.**TreeRowReferenceWrapper**(*model*, *path*)
A Gtk.TreeRowReference wrapper that compares to another one based on model and path

valid()
Check if the reference still points to a valid row

Returns True | False

Return type bool

model
Get model

Returns Model pointed by this reference

Return type TreeStore

path
Get path

Returns Path to the row pointed by this reference

Return type tuple

row
Get row

Returns Row pointed by this reference

Return type TreeModelRowWrapper

key_path
Get path created from the key column value of each row in the tree hierarchy needed to get to this row

Returns Path of names to the row

Return type KeyPath

2.7 runner

Runner package contains test runners that can be used to run a particular test case

class testmanager.runner.**Runner**(*data*)
Runner objects run test cases, collect results and gather additional information such as log files

static get_all()
Get all classes that are subclasses of Runner

Returns Classes that inherit from Runner

Return type list(type)

classmethod get_name()
Return runner name based on class name

Runner name is just the class name with spaces added for each uppercase letter

Returns Runner name

Return type str

class testmanager.runner.**RunnerCaseUI**(*parent_tab*)
RunnerCaseUI objects take care of handling additional widgets to collect the information required to run the case

static create(*runner_name*, *parent_tab*)
Create RunnerCaseUI object based on runner name

Parameters **runner_name** (*str*) – Runner name used to execute case

Returns Runner UI

Return type RunnerCaseUI (any of its subclasses)

static get_all()
Get all classes that are subclasses of RunnerCaseUI

Returns Classes that inherit from RunnerCaseUI

Return type list(type)

classmethod get_name()
Return runner name based on class name

Runner name is just the class name with spaces added for each uppercase letter

Returns Runner name

Return type str

classmethod get_ui_filename()
Return ui filename based on class name

Returns UI filename

Return type str

class testmanager.runner.**RunnerResultUI**(*data*)
RunnerResultUI objects take car of handling additional widgets to display the additional result information

static create(*runner_name*, *row_data*)
Create RunnerResultUI object based on runner name

Parameters

- **runner_name** (*str*) – Runner name used to execute case
- **row_data** – Result row data with the information to display by the runner

Returns Runner UI

Return type RunnerResultUI (any of its subclasses)

static get_all()
Get all classes that are subclasses of RunnerResultUI

Returns Classes that inherit from RunnerResultUI

Return type list(type)

classmethod get_name()
Return runner name based on class name

Runner name is just the class name with spaces added for each uppercase letter

Returns Runner name

Return type str

classmethod get_ui_filename()
Return ui filename based on class name

Returns UI filename

Return type str

class testmanager.runner.**GtkManualRunner**(*data*)
GtkManualRunner object guides user in manual tests

class testmanager.runner.**GtkManualRunnerCaseUI**(*parent_tab*)
GTK manual runner requires a procedure field

procedure
Get procedure field

Returns Procedure field

Return type str

procedure_changed_cb(*textbuffer*)
Validate procedure, update it and save change to history

Parameters **textbuffer** (`Gtk.TextBuffer`) – The textbuffer that emitted the `changed` signal

Returns None

save()
Save contents of the widgets in the UI to the row in the store

Returns None

class testmanager.runner.**GtkManualRunnerResultUI**(*data*)
Display case result and tester comments

class testmanager.runner.**AutomaticRunner**(*data*)
AutomaticRunner executes a given command

class testmanager.runner.**AutomaticRunnerCaseUI**(*parent_tab*)
Automatic runner requires a command field

command
Get command field

Returns Command field

Return type str

command_changed_cb(*entry*)
Validate command, update it and save change to history

Parameters **entry** (`Gtk.Entry`) – The entry that emitted the `changed` signal

Returns None

save()
Save contents of the widgets in the UI to the row in the store

Returns None

class testmanager.runner.**AutomaticRunnerResultUI**(*data*)
Display command execution results

2.8 uifile

User interface file handling classes

class testmanager.uifile.**UIFile**(*ui_filename*)
Load user interface file and provide the following features:

- Connect callbacks methods from ui file
- Attribute access to widgets in the ui file through `Gtk.Builder.get_object()`
- Helper methods to connect/disconnect all signal handlers to other objects. This is useful for dialogs that are destroyed, but connect to signals of other objects that remain alive. This is important because otherwise a reference to the callback methods is kept and, in fact, those callbacks are executed even after the dialog has been destroyed.

connect (*obj*, *signal*, *handler*, **args*)
Connect signal handler and keep a record of its id to disconnect it on destroy

Parameters

- **obj** (`gobject.GObject`) – Object that will emit the signal
- **signal** (*str*) – Signal name
- **handler** (*function | bound method*) – Callback to be executed when the signal is emitted
- **args** (*iterable*) – Additional arguments required by the signal

Returns None

disconnect_all ()
Disconnect all handlers

Returns None

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

t