ViUR Vi

Release 3.0

Tilman Oestereich, Andreas H. Kelch

Aug 06, 2021

VIUR VI

Administrativ viusal interface for the ViUR Information System

CHAPTER

ONE

ABOUT

soon...

1.1 Getting started

1.1.1 Systemrequirements

soon...

1.1.2 Setup and Installation

Linux or WSL

soon...

Mac OS

soon...

1.2 Reference Guide

1.2.1 Overview

soon...

1.2.2 Navigation

soon...

1.2.3 Handlers

soon...

1.2.4 Acionbars

soon...

1.3 Tutorials

1.3.1 Create a Actionbar Plugin

soon...

1.3.2 Create a Handler Plugin

soon...

1.3.3 Create a Navigation Plugin

soon...

1.4 API Reference

This page contains auto-generated API reference documentation[1].

1.4.1 vi

Subpackages

vi.actions

Submodules

vi.actions.context

Module Contents

Classes

[1] Created with sphinx-autoapi

ContextAction

class vi.actions.context.**ContextAction**(*module*, *handler*, *actionName*, **args*, ***kwargs*)
Bases: flare.button.Button

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onClick(*self*, *sender=None*)

openModule(*self*, *data*, *title=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

vi.actions.edit

Module Contents

Classes

SaveContinue

SaveSingleton

ExecuteSingleton

SaveClose

Refresh

CancelClose

class vi.actions.edit.**SaveContinue**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.edit.**SaveSingleton**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.edit.**ExecuteSingleton**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.edit.**SaveClose**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.edit.**Refresh**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

performReload(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.edit.**CancelClose**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

vi.actions.file

Module Contents

Classes

FileSelectUploader	Small wrapper around <input type="file">.
AddNodeAction	Adds a new directory to a tree.simple application.
AddLeafAction	Allows uploading of files using the file dialog.
EditAction	Provides editing in a tree.simple application.
DownloadAction	Allows downloading files from the server.

class vi.actions.file.**FileSelectUploader**(**args*, ***kwargs*)
Bases: flare.html5.Input

Small wrapper around <input type="file">. Creates the element; executes the click (=opens the file dialog); runs the callback if a file has been selected and removes itself from its parent.

onChange(*self*, *event*)

class vi.actions.file.**AddNodeAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Adds a new directory to a tree.simple application.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

createDir(*self*, *dialog*, *dirName*)

onMkDir(*self*, *req*)

resetLoadingState(*self*)

class vi.actions.file.**AddLeafAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows uploading of files using the file dialog.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.file.**EditAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Provides editing in a tree.simple application. If a directory is selected, it opens a dialog for renaming that directory, otherwise the full editWidget is used.

onAttach(*self*)

onDetach(*self*)

onSelectionActivated(*self*, *table*, *selection*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

editDir(*self*, *dialog*, *dirName*)

resetLoadingState(*self*)

class vi.actions.file.**DownloadAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows downloading files from the server.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

disableViUnloadingWarning(*self*, **args*, ***kwargs*)

enableViUnloadingWarning(*self*, **args*, ***kwargs*)

doDownload(*self*, *fileData*)

resetLoadingState(*self*)

vi.actions.hierarchy

Module Contents

Classes

AddAction	Adds a new node in a hierarchy application.
EditAction	Edits a node in a hierarchy application.
CloneAction	Allows cloning an entry (including its subentries) in a hierarchy application.
DeleteAction	Deletes a node from a hierarchy application.
ReloadAction	Allows adding an entry in a list-module.
SelectRootNode	Selector for hierarchy root nodes.
ListViewAction	Allows adding an entry in a list-module.

class vi.actions.hierarchy.**AddAction**(**args*, ***kwargs*)

Bases: flare.button.Button

Adds a new node in a hierarchy application.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.hierarchy.**EditAction**(**args*, ***kwargs*)

Bases: flare.button.Button

Edits a node in a hierarchy application.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onSelectionActivated(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

openEditor(*self*, *key*)

resetLoadingState(*self*)

class vi.actions.hierarchy.**CloneAction**(**args*, ***kwargs*)

Bases: flare.button.Button

Allows cloning an entry (including its subentries) in a hierarchy application.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

openEditor(*self*, *key*)

resetLoadingState(*self*)

class vi.actions.hierarchy.**DeleteAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Deletes a node from a hierarchy application.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

doDelete(*self*, *dialog*)

allDeletedSuccess(*self*, *success*)

resetLoadingState(*self*)

class vi.actions.hierarchy.**ReloadAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows adding an entry in a list-module.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.hierarchy.**SelectRootNode**(*module*, *handler*, *actionName*, **args*, ***kwargs*)
Bases: flare.html5.Select

Selector for hierarchy root nodes.

onAttach(*self*)

onDetach(*self*)

update(*self*)

onRootNodeChanged(*self*, *newNode*)

onRootNodesAvailable(*self*, *req*)

onChange(*self*, *event*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.hierarchy.**ListViewAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows adding an entry in a list-module.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

vi.actions.list

Module Contents

Classes

AddAction	Allows adding an entry in a list-module.
EditAction	Allows editing an entry in a list-module.
CloneAction	Allows cloning an entry in a list-module.
DeleteAction	Allows deleting an entry in a list-module.
ListPreviewAction	
ListPreviewInlineAction	
CloseAction	
SelectAction	
SelectFieldsPopup	
SelectFieldsAction	
ReloadAction	Allows Reloading
TableNextPage	
TablePrevPage	
TableItems	
SetPageRowAmountAction	Load a bunch of pages
LoadNextBatchAction	Load a bunch of pages
LoadAllAction	Allows Loading all Entries in a list
PageFindAction	Allows Loading all Entries in a list
ListSelectFilterAction	
CreateRecurrentAction	
ExportCsvAction	
SelectAllAction	
UnSelectAllAction	
SelectInvertAction	

class vi.actions.list.**AddAction**(**args*, ***kwargs*)

Bases: flare.button.Button

Allows adding an entry in a list-module.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.list.EditAction(**args*, ***kwargs*)
Bases: flare.button.Button

Allows editing an entry in a list-module.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onSelectionActivated(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

openEditor(*self*, *key*)

resetLoadingState(*self*)

class vi.actions.list.CloneAction(**args*, ***kwargs*)
Bases: flare.button.Button

Allows cloning an entry in a list-module.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

openEditor(*self*, *key*)

resetLoadingState(*self*)

class vi.actions.list.DeleteAction(**args*, ***kwargs*)
Bases: flare.button.Button

Allows deleting an entry in a list-module.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

doDelete(*self*, *dialog*)

allDeletedSuccess(*self*, *success*)

deletedSuccess(*self*, *req=None*, *code=None*)

deletedFailed(*self*, *req=None*, *code=None*)

resetLoadingState(*self*)

class vi.actions.list.**ListPreviewAction**(*module*, *handler*, *actionName*, **args*, ***kwargs*)
Bases: flare.html5.Span

onChange(*self*, *event*)

rebuildCB(*self*, **args*, ***kwargs*)

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**ListPreviewInlineAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onClick(*self*, *sender=None*)

toggleIntPrev(*self*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**CloseAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**SelectAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**SelectFieldsPopup**(*listWdg*, **args*, ***kwargs*)
Bases: flare.popup.Popup

doApply(*self*, **args*, ***kwargs*)

doSetFields(*self*, **args*, ***kwargs*)

doCancel(*self*, **args*, ***kwargs*)

doSelectAll(*self*, **args*, ***kwargs*)

doUnselectAll(*self*, **args*, ***kwargs*)

doInvertSelection(*self*, **args*, ***kwargs*)

class vi.actions.list.**SelectFieldsAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onClick(*self*, *sender=None*)

onAttach(*self*)

onDetach(*self*)

onTableChanged(*self*, *table*, *count*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**ReloadAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows Reloading

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *event=None*)

resetLoadingState(*self*)

class vi.actions.list.**TableNextPage**(**args*, ***kwargs*)
Bases: flare.button.Button

postInit(*self*, *widget=None*)

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

resetLoadingState(*self*)

class vi.actions.list.**TablePrevPage**(**args*, ***kwargs*)
Bases: flare.button.Button

postInit(*self*, *widget=None*)

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**TableItems**(**args*, ***kwargs*)
Bases: flare.html5.Div

postInit(*self*, *widget=None*)

onTableChanged(*self*, *table*, *rowCount*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**SetPageRowAmountAction**(**args*, ***kwargs*)
Bases: flare.html5.Div

Load a bunch of pages

onClick(*self*, *sender=None*)

onChange(*self*, *sender=None*)

setPageAmount(*self*)

static isSuitableFor(*module*, *handler*, *actionName*)

resetLoadingState(*self*)

class vi.actions.list.**LoadNextBatchAction**(**args*, ***kwargs*)
Bases: flare.html5.Div

Load a bunch of pages

registerScroll(*self*)

onScroll(*self*, *sender=None*)

onClick(*self*, *sender=None*)

onChange(*self*, *sender=None*)

loadnextPages(*self*, **args*, ***kwargs*)

static isSuitableFor(*module*, *handler*, *actionName*)

resetLoadingState(*self*)

class vi.actions.list.**LoadAllAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Allows Loading all Entries in a list

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

loadAllRows(*self*)

resetLoadingState(*self*)

class vi.actions.list.**PageFindAction**(**args*, ***kwargs*)
Bases: flare.html5.Div

Allows Loading all Entries in a list

onKeyPress(*self*, *event*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

startFind(*self*)

findText(*self*)

resetLoadingState(*self*)

class vi.actions.list.**ListSelectFilterAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onAttach(*self*)

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**CreateRecurrentAction**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

class vi.actions.list.**ExportCsvAction**(**args*, ***kwargs*)
Bases: flare.button.Button

onClick(*self*, *sender=None*)

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list.**SelectAllAction**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

onAttach(*self*)

onDetach(*self*)

onTableChanged(*self*, *table*, *count*)

class vi.actions.list.**UnSelectAllAction**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

onAttach(*self*)

onDetach(*self*)

onTableChanged(*self*, *table*, *count*)

class vi.actions.list.**SelectInvertAction**(**args*, ***kwargs*)
Bases: flare.button.Button

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

onAttach(*self*)

onDetach(*self*)

onTableChanged(*self*, *table*, *count*)

vi.actions.list_order

Module Contents

Classes

ShopMarkAction
ShopMarkPayedAction
ShopMarkSentAction
ShopMarkCanceledAction

class vi.actions.list_order.**ShopMarkAction**(*action*, *title*, *cls=''*, *txtQuestion=None*, *txtSuccess=None*, *txtFailure=None*, **args*, ***kwargs*)
Bases: flare.button.Button

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

setPayed(*self*, *order*)

setPayedSucceeded(*self*, *response*)

setPayedFailed(*self*, *response*)

doMarkPayed(*self*, **args*, ***kwargs*)

onClick(*self*, *sender=None*)

class vi.actions.list_order.**ShopMarkPayedAction**(**args*, ***kwargs*)
Bases: *ShopMarkAction*

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list_order.**ShopMarkSentAction**(**args*, ***kwargs*)
Bases: *ShopMarkAction*

static isSuitableFor(*module*, *handler*, *actionName*)

class vi.actions.list_order.**ShopMarkCanceledAction**(**args*, ***kwargs*)
Bases: *ShopMarkAction*

static isSuitableFor(*module*, *handler*, *actionName*)

vi.actions.tree

Module Contents

Classes

AddLeafAction	Creates a new leaf (ie. a file) for a tree application
AddNodeAction	Creates a new node (ie. a directory) for a tree application
EditAction	Edits an entry inside a tree application.
DeleteAction	Allows deleting an entry in a tree-module.
ReloadAction	Allows adding an entry in a list-module.
SelectRootNode	Allows selecting a different rootNode in Tree applications

class vi.actions.tree.**AddLeafAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Creates a new leaf (ie. a file) for a tree application

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.tree.**AddNodeAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Creates a new node (ie. a directory) for a tree application

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.tree.**EditAction**(**args*, ***kwargs*)
Bases: flare.button.Button

Edits an entry inside a tree application. The type (node or leaf) of the entry is determined dynamically

onAttach(*self*)

onDetach(*self*)

onSelectionActivated(*self*, *table*, *selection*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.tree.DeleteAction(**args*, ***kwargs*)
Bases: flare.button.Button

Allows deleting an entry in a tree-module. The type (node or leaf) of the entry is determined dynamically.

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

doDelete(*self*, *dialog*)

allDeletedSuccess(*self*, *success*)

resetLoadingState(*self*)

class vi.actions.tree.ReloadAction(**args*, ***kwargs*)
Bases: flare.button.Button

Allows adding an entry in a list-module.

static isSuitableFor(*module*, *handler*, *actionName*)

onClick(*self*, *sender=None*)

resetLoadingState(*self*)

class vi.actions.tree.SelectRootNode(*module*, *handler*, *actionName*, **args*, ***kwargs*)
Bases: flare.html5.Select

Allows selecting a different rootNode in Tree applications

onAttach(*self*)

onDetach(*self*)

update(*self*)

onRootNodeChanged(*self*, *newNode*)

onRootNodesAvailable(*self*, *req*)

onChange(*self*, *event*)

static isSuitableFor(*module*, *handler*, *actionName*)

vi.framework

Subpackages

vi.framework.components

Submodules

vi.framework.components.actionbar

Module Contents

Classes

ActionBar	Provides the container for actions (add,edit,..) suitable for one module (eg. for lists).

class vi.framework.components.actionbar.**ActionBar**(*module=None*, *appType=None*, *currentAction=None*, **args*, ***kwargs*)

Bases: flare.html5.Div

Provides the container for actions (add,edit,..) suitable for one module (eg. for lists).

setActions(*self*, *actions*, *widget=None*)

Sets the list of valid actions for this module. This function tries to resolve a suitable action-widget for each given action-name and adds them on success. All previous actions are removed. :param actions: List of names of actions which should be available. :type actions: list of str

getActions(*self*)

Returns the list of action-names currently active for this module. May also contain action-names which couldn't be resolved and therefore not displayed. :returns: List of str

resetLoadingState(*self*)

Resets the loading-state of each child. Each child has the ability to provide visual feedback once it has been clicked and started working. This function is called from our parent once that action has finished, so we can tell our children to return to a sane state.

vi.framework.components.datatable

Module Contents

Classes

SelectTable	Provides an Html-Table which allows for row selections.
DataTable	
ViewportDataTable	

class vi.framework.components.datatable.**SelectTable**(*checkboxes=False*, *indexes=False*, **args*, ***kwargs*)

Bases: flare.ignite.Table

Provides an Html-Table which allows for row selections.

Parent widgets can register for certain events:

- **selectionChanged: called if the current _multi_ selection changes. (Ie the user** holds ctrl and clicks a row). The selection might contain no, one or multiple rows. Its also called if the cursor moves. Its called if the user simply double clicks a row. So its possible to receive a selectionActivated event without an selectionChanged Event.
- **selectionActivated: called if a selection is activated, ie. a row is double-clicked or Return** is pressed.
- **cursorMoved: called when the currently active row changes. The user can select the current row** with a single click or by moving the cursor up and down using the arrow keys.

onAttach(*self*)

setHeader(*self*, *headers*)
Sets the table-headers to 'headers' :param headers: list of strings :type headers: list

getTrByIndex(*self*, *idx*)
Retrieves the TR element by the given row number :param idx: Rownumber to retrieve the tr of :type idx: int :returns: HTMLTableRowElement

getIndexByTr(*self*, *tr*)
Returns the rowNumber for the given tr element or None if the given tr element is invalid. :param tr: A HTMLTableRowElement of this table :type tr: HTMLTableRowElement :returns: int or None

_rowForEvent(*self*, *event*)
Determines the row number for the given event

onChange(*self*, *event*)

onMouseDown(*self*, *event*)

onMouseOut(*self*, *event*)

onMouseUp(*self*, *event*)

onKeyDown(*self*, *event*)

onKeyUp(*self*, *event*)

onDblClick(*self*, *event*)

addSelectedRow(*self*, *row*)
Marks a row as selected

removeSelectedRow(*self*, *row*)
Removes 'row' from the current selection (if any) :param row: Number of the row to unselect :type row: int

selectRow(*self*, *newRow*)
Sets the current selection to 'row'. Any previous selection is removed. :param newRow: Number of the row to select :type newRow: int

setCursorRow(*self*, *row*, *removeExistingSelection=True*)
Move the cursor to row 'row'. If removeExistingSelection is True, the current selection (if any) is invalidated.

focusRow(*self*, *row*)

getCurrentSelection(*self*)
Returns a list of currently selected row-numbers :returns: list

clear(*self*)
Hook the clear() method so we can reset some internal states, too

removeRow(*self*, *row*)
Hook the removeRow method so we can reset some internal states, too

_extraCols(*self*)

prepareCol(*self*, *row*, *col*)
Lets hook up the original removeRow function to optionally provide index and checkbox columns.

dropTableContent(*self*)
Drops content from the table, structure remains unchanged

setCell(*self*, *row*, *col*, *val*)
Interface for self["cell"] that directs to the correct cell if extra columns are configured for this SelectTable.

selectAll(*self*)
Selects all entries of the table.

unSelectAll(*self*)
Unselects all entries of the table.

invertSelection(*self*)
Inverts the current selection on the whole table currently displayed.

class vi.framework.components.datatable.**DataTable**(*_loadOnDisplay=False*, **args*, ***kwargs*)
Bases: flare.html5.Div

setDataProvider(*self*, *obj*)
Register's 'obj' as the provider for this table. It must provide a onNextBatchNeeded function, which must fetch and feed new rows using add() or reset the dataProvider to None if no more rows are available. Notice: If the bottom of the table is reached, onNextBatchNeeded will only be called once. No further calls will be made until add() or setDataProvider() has been called afterwards.

onCursorMoved(*self*, *table*, *row*)
Ensure the table scrolls according to the position of its cursor

getRowCount(*self*)
Returns the total amount of rows currently known. :returns: int

add(*self*, *obj*)
Adds an row to the model :param obj: Dictionary of values for this row :type obj: dict

update(*self*, *objList*, *writeToModel=True*)
Adds multiple rows at once. Much faster than calling add() multiple times.

extend(*self*, *objList*, *writeToModel=True*)

remove(*self*, *objOrIndex*)
Removes 'obj' from the table. 'obj' may be an row-index or an object recieved by any eventListener. It _cannot_ be any original object passed to 'add' - it _must_ be recived by an eventListener!

clear(*self*, *keepModel=False*)
Flushes the whole table.

_renderObject(*self*, *obj*, *tableIsPrepared=False*, *recalculate=True*)
Renders the object to into the table. Does nothing if the list of _shownFields is empty. :param obj: Dictionary of values for this row :type obj: dict

rebuildTable(*self*, *recalculate=True*)
Rebuilds the entire table. Useful if something fundamental changed (ie. the cell renderer or the list of visible fields)

setShownFields(*self*, *fields*)
Sets the list of _shownFields. This causes the whole table to be rebuild. Be careful if calling this function often on a large table! :param fields: List of model-keys which will be displayed. :type fields: list

onSelectionChanged(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onSelectionActivated(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onTableChanged(*self*, *table*, *rowCount*)
Re-emit the event.

getCurrentSelection(*self*)
Override the getCurrentSelection method to yield actual models, not row-numbers.

setCellRender(*self*, *field*, *render*)
Sets the render for cells of 'field' to render. A cell render receives the data for a given cell and returns the appropriate widget to display that data for the table.

setCellRenders(*self*, *renders*)
Like setCellRender, but sets multiple renders at one. Much faster than calling setCellRender repeatedly.

activateCurrentSelection(*self*)
Emits the selectionActivated event if there's currently a selection

class vi.framework.components.datatable.**ViewportDataTable**(*_loadOnDisplay=False*, *rows=99*, **args*, ***kwargs*)

Bases: *DataTable*

clear(*self*, *keepModel=False*)
Flushes the whole table. Override explanation - replaced clear with dropTableContent

rebuildTable(*self*, *recalculate=True*)
Rebuilds the entire table. Useful if something fundamental changed (ie. the cell renderer or the list of visible fields)#

Override explanation - uses the predefined _rows to prepare the grid - only load first rows from model

add(*self*, *obj*)
Adds an row to the model :param obj: Dictionary of values for this row :type obj: dict

Override explanation - _renderObject call is always prepared

extend(*self*, *objList*, *writeToModel=True*)

update(*self*, *objList*, *writeToModel=True*)
Adds multiple rows at once. Much faster than calling add() multiple times.

Override explanation - removed grid preparation

_renderObject(*self*, *obj*, *tableIsPrepared=True*, *recalculate=True*)
Renders the object to into the table. Does nothing if the list of _shownFields is empty. :param obj: Dictionary of values for this row :type obj: dict

Override explanation - removed Table preperation - rowIndex modulo shownrows

vi.sidebarwidgets

Submodules

vi.sidebarwidgets.filterselector

Module Contents

Classes

CompoundFilter
FilterSelector

class vi.sidebarwidgets.filterselector.**CompoundFilter**(*view*, *module*, *embed=False*, **args*, ***kwargs*)

Bases: flare.html5.Div

onFilterChanged(*self*, **args*, ***kwargs*)

reevaluate(*self*, **args*, ***kwargs*)

focus(*self*)

class vi.sidebarwidgets.filterselector.**FilterSelector**(*module*, **args*, ***kwargs*)

Bases: flare.html5.Div

onClick(*self*, *event*)

Handle event on filter selection (fold current active filter, expand selected filter and execute, if possible) :param event: :return:

onAttach(*self*)

onDetach(*self*)

onStartSearch(*self*, *searchTxt=None*)

setView(*self*, *btn*)

applyFilter(*self*, *filter*, *filterID*, *filterName*)

vi.sidebarwidgets.internalpreview

Module Contents

Classes

InternalPreview

class vi.sidebarwidgets.internalpreview.**InternalPreview**(*module*, *structure*, *item*, **args*, ***kwargs*)

Bases: flare.html5.Ul

onCopyKey(*self*, *btn*)

vi.translations

Submodules

vi.translations.de

Module Contents

vi.translations.de.lngDe

vi.translations.en

Module Contents

vi.translations.en.lngEn

Package Contents

vi.translations.lngDe

vi.translations.lngEn

vi.views

Submodules

vi.views.edit

Module Contents

Classes

editHandler
editHandlerWidget

class vi.views.edit.editHandler
Bases: flare.views.view.View

class vi.views.edit.editHandlerWidget
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

vi.views.hierarchy

Module Contents

Classes

hierarchyHandler
hierarchyHandlerWidget

class vi.views.hierarchy.**hierarchyHandler**
Bases: flare.views.view.View

static canHandle(*moduleName*, *moduleInfo*)

class vi.views.hierarchy.**hierarchyHandlerWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

vi.views.list

Module Contents

Classes

listHandler
listHandlerWidget

class vi.views.list.**listHandler**
Bases: flare.views.view.View

static canHandle(*moduleName*, *moduleInfo*)

class vi.views.list.**listHandlerWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

onViewfocusedChanged(*self*, *viewname*, **args*, ***kwargs*)

vi.views.log

Module Contents

Classes

logHandler
logHandlerWidget

class vi.views.log.**logHandler**
Bases: flare.views.view.View

class vi.views.log.**logHandlerWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

vi.views.notfound

Module Contents

Classes

NotFound
NotFoundWidget

class vi.views.notfound.**NotFound**
Bases: flare.views.view.View

class vi.views.notfound.**NotFoundWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

vi.views.overview

Module Contents

Classes

Overview

OverviewWidget

class vi.views.overview.**Overview**
Bases: flare.views.view.View

class vi.views.overview.**OverviewWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

vi.views.singleton

Module Contents

Classes

singletonHandler

singletonHandlerWidget

class vi.views.singleton.**singletonHandler**
Bases: flare.views.view.View

static canHandle(*moduleName*, *moduleInfo*)

class vi.views.singleton.**singletonHandlerWidget**
Bases: flare.views.view.ViewWidget

initWidget(*self*)
Here we start!

onViewfocusedChanged(*self*, *viewname*, **args*, ***kwargs*)

vi.views.tree

Module Contents

Classes

treeHandler

treeHandlerWidget

class vi.views.tree.**treeHandler**

Bases: flare.views.view.View

static canHandle(*moduleName*, *moduleInfo*)

class vi.views.tree.**treeHandlerWidget**

Bases: flare.views.view.ViewWidget

initWidget(*self*)

Here we start!

vi.widgets

Submodules

vi.widgets.accordion

Module Contents

Classes

AccordionSegment
Accordion

class vi.widgets.accordion.**AccordionSegment**(*ident*, *title=None*)

Bases: flare.html5.Fieldset

checkVisibility(*self*)

activate(*self*)

deactivate(*self*)

isActive(*self*)

toggle(*self*)

onClick(*self*, *event*)

addWidget(*self*, *widget*)

class vi.widgets.accordion.**Accordion**

Bases: flare.html5.Form

_segments = []

addSegment(*self*, *ident*, *title=None*, *directAdd=False*, **args*)

clearSegments(*self*)

buildAccordion(*self*, *order=None*)

Parameters **sort** – None: sorted by Bones, "asc":ascending, "desc":descending, dict: {"category":index,...}

Returns

vi.widgets.appnavigation

Module Contents

Classes

NavigationElement
NavigationSeperator
Navigationblock
AppNavigation

class vi.widgets.appnavigation.**NavigationElement**(*name*, *icon=None*, *view=None*, *nav=None*, *closeable=False*)

Bases: flare.html5.Div

tpl = Multiline-String

```
            <div [name]="item" class="item has-hover">
                    <a class="item-link" @click="navigationAction">
                            <div class="item-image">
                                    <flare-icon value="{{icon}}" title="
↪{{name}}"></flare-icon>
                            </div>

                            <div class="item-content">
                                    <div class="item-headline">{{name}}
↪</div>
                            </div>
                    </a>

                    <span [name]="itemArrow" class="item-open is-hidden
↪" @click="ArrowAction">
                            <flare-svg-icon value="icon-arrow-left"></
↪flare-svg-icon>
                    </span>
                    <span [name]="itemRemove" class="is-hidden" @click=
↪"RemoveAction">
                            <flare-svg-icon value="icon-cross"></flare-
↪svg-icon>
                    </span>

            </div>
            <div [name]="subItem" class="list list--sub">
            </div>
```

onActiveViewChanged(*self*, *e*, *wdg*)

navigationAction(*self*, *e=None*, *wdg=None*)

Handle Click on Navigation Button

RemoveAction(*self*, *e=None*)
remove this Nav Element

ArrowAction(*self*, *e*, *wdg=None*)

onActiveNavigationChanged(*self*, *e*, *wdg*)
What should happen if the State from the surrounding Navigation gets an update

onHasSubItemsChanged(*self*, *e*, *wdg*)
If subChild is added, show itemArrow, hide if no subitem present

appendSubChild(*self*, *element*)

class vi.widgets.appnavigation.**NavigationSeperator**(*name=None*)
Bases: flare.html5.Div

buildSeperator(*self*)

_setValue(*self*, *value*)

class vi.widgets.appnavigation.**Navigationblock**(*name*)
Bases: flare.html5.Div

addSeperator(*self*)

seperatorAction(*self*, *e*, *wdg=None*)

class vi.widgets.appnavigation.**AppNavigation**
Bases: flare.html5.Nav

getPreviousNavigationPoint(*self*, *view*)

getNavigationPoint(*self*, *view*)

addNavigationBlock(*self*, *name*)

addNavigationPoint(*self*, *name*, *icon*, *view=None*, *parent=None*, *closeable=False*)

addNavigationPointAfter(*self*, *name*, *icon*, *view=None*, *beforeElement=None*, *closeable=False*)

removeNavigationPoint(*self*, *view*)

vi.widgets.csvexport

Module Contents

Classes

ExportCsv
ExportCsvStarter

class vi.widgets.csvexport.**ExportCsv**(*widget*, *selection*, *encoding=None*, *language=None*, *separator=None*, *lineSeparator=None*, **args*, ***kwargs*)
Bases: flare.html5.Progress

nextChunk(*self*, *cursor=None*)

nextChunkComplete(*self*, *req*)

exportToFile(*self*)

nextChunkFailure(*self*, *req*, *code*)

replaceWithMessage(*self*, *message*, *logClass='success'*)

class vi.widgets.csvexport.**ExportCsvStarter**(*widget*, **args*, ***kwargs*)
Bases: flare.popup.Popup

onExportBtnClick(*self*, **args*, ***kwargs*)

vi.widgets.edit

Module Contents

Classes

ParsedErrorItem	
PassiveErrorItem	
EditWidget	

Functions

parseHashParameters(src, prefix='')	Converts a flat dictionary containing dotted properties into a multi-dimensional one.

class vi.widgets.edit.**ParsedErrorItem**(*error*)
Bases: flare.html5.Li

style = []

class vi.widgets.edit.**PassiveErrorItem**(*error*)
Bases: flare.html5.Li

style = []

vi.widgets.edit.**parseHashParameters**(*src*, *prefix=''*)
Converts a flat dictionary containing dotted properties into a multi-dimensional one.

Example: { “a”:”a”, “b.a”:”ba”,”b.b”:”bb” } -> { “a”:”a”, “b”:{“a”:”ba”,”b”:”bb”} }

If a dictionary contains only numeric indexes, it will be converted to a list: { “a.0.a”:”a0a”, “a.0.b”:”a0b”,a.1.a”:”a1a” } -> { “a”:[{“a”:”a0a”,”b”:”a0b”},{“a”:”a1a”}] }

class vi.widgets.edit.**EditWidget**(*module*, *applicationType*, *key=0*, *node=None*, *skelType=None*, *clone=False*, *hashArgs=None*, *context=None*, *logAction='Entry saved!'*, **args*, ***kwargs*)
Bases: flare.html5.Div

appList = list

appHierarchy = hierarchy

appTree = tree

appSingleton = singleton

__editIdx_ = 0

onDetach(*self*)

onAttach(*self*)

performLogics(*self*)

onChange(*self*, *event*)

onBoneChange(*self*, *bone*)

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displays an error message

reloadData(*self*)

save(*self*, *data*)
Creates the actual NetworkService request used to transmit our data. If data is None, it fetches a clean add/edit form.

Parameters **data** (*dict or None*) – The values to transmit or None to fetch a new, clean add/edit form.

clear(*self*)
Removes all visible bones/forms/fieldsets.

closeOrContinue(*self*, *sender=None*)

doCloneHierarchy(*self*, *sender=None*)

cloneComplete(*self*, *request*)

formatReadFromClientErrorSeverity(*self*, *error*)

setData(*self*, *request=None*, *data=None*, *ignoreMissing=False*, *askHierarchyCloning=True*)
Rebuilds the UI according to the skeleton received from server

Parameters

- **request** (*NetworkService*) – A finished NetworkService request
- **data** (*dict*) – The data received

unserialize(*self*, *data=None*, *errors=()*)
Applies the actual data to the bones.

serializeForPost(*self*, *validityCheck=False*)

serializeForDocument(*self*)

doSave(*self*, *closeOnSuccess=False*, **args*, ***kwargs*)
Starts serializing and transmitting our values to the server.

vi.widgets.file

Module Contents

Classes

FileImagePopup	
FilePreviewImage	
Uploader	Uploads a file to the server while providing visual feedback of the progress.
MultiUploader	
FileLeafWidget	
FileNodeWidget	
FileWidget	Base Widget that renders a tree.

class vi.widgets.file.**FileImagePopup**(*preview*, **args*, ***kwargs*)
Bases: flare.popup.Popup

onClick(*self*, *event*)

onDownloadBtnClick(*self*, *sender=None*)

class vi.widgets.file.**FilePreviewImage**(*file=None*, *size=150*, **args*, ***kwargs*)
Bases: flare.html5.Div

setFile(*self*, *file*)

download(*self*)

onClick(*self*, *sender=None*)

class vi.widgets.file.**Uploader**(*file*, *node*, *context=None*, *module='file'*, **args*, ***kwargs*)
Bases: flare.html5.Div

Uploads a file to the server while providing visual feedback of the progress.

onUploadUrlAvailable(*self*, *req*)
Internal callback - the actual upload url (retrieved by calling /file/getUploadURL) is known.

onSkeyAvailable(*self*, *req*)
Internal callback - the Security-Key is known. Only for core 2.x needed

onLoad(*self*, **args*, ***kwargs*)
Internal callback - The state of our upload changed.

onUploadAdded(*self*, *req*)

onProgress(*self*, *event*)
Internal callback - further bytes have been transmitted

onSuccess(*self*, **args*, ***kwargs*)
Internal callback - The upload succeeded.

onFailed(*self*, *errorCode*, **args*, ***kwargs*)

replaceWithMessage(*self*, *message*, *isSuccess*)

class vi.widgets.file.**MultiUploader**(*files*, *node*, *context=None*, *module='file'*, **args*, ***kwargs*)
Bases: flare.html5.Div

handleFile(*self*, *file*)

onUploadUrlAvailable(*self*, *req*)
Internal callback - the actual upload url (retrieved by calling /file/getUploadURL) is known.

onLoad(*self*, **args*, ***kwargs*)
Internal callback - The state of our upload changed.

onUploadAdded(*self*, *req*)

onSuccess(*self*, **args*, ***kwargs*)
Internal callback - The upload succeeded.

onFailed(*self*, *errorCode*, **args*, ***kwargs*)

replaceWithMessage(*self*, *message*, *isSuccess*)

closeMessage(*self*)

class vi.widgets.file.**FileLeafWidget**
Bases: vi.widgets.tree.TreeLeafWidget

EntryIcon(*self*)

setStyle(*self*)

class vi.widgets.file.**FileNodeWidget**
Bases: vi.widgets.tree.TreeNodeWidget

setStyle(*self*)

class vi.widgets.file.**FileWidget**(*module*, *rootNode=None*, *selectMode=None*, *node=None*, *context=None*, **args*, ***kwargs*)
Bases: *vi.widgets.tree.TreeBrowserWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

searchWidget(*self*)

onStartSearch(*self*, *searchStr*, **args*, ***kwargs*)

getChildKey(*self*, *widget*)
Derives a string used to sort the entries on each level

onDrop(*self*, *event*)
We got a drop event. Make that item a direct child of our rootNode

static canHandle(*module*, *moduleInfo*)

vi.widgets.hierarchy

Module Contents

Classes

HierarchyWidget	A Hierarchy is a Tree without leaf distiction!

class vi.widgets.hierarchy.**HierarchyWidget**(**args*, ***kwargs*)

Bases: *vi.widgets.tree.TreeWidget*

A Hierarchy is a Tree without leaf distiction!

leafWidget

reloadData(*self*)

Reload the data were displaying.

reloadListWidget(*self*)

toggleListView(*self*)

setListView(*self*, *visible=False*)

showListView(*self*)

hideListView(*self*)

onSelectionChanged(*self*, *widget*, *selection*)

static canHandle(*moduleName*, *moduleInfo*)

vi.widgets.htmleditor

Module Contents

Classes

TextInsertImageAction	
HtmlEditor	

class vi.widgets.htmleditor.**TextInsertImageAction**(*summernote=None*, *boneName=''*, **args*, ***kwargs*)

Bases: flare.button.Button

onClick(*self*, *sender=None*)

onSelectionActivated(*self*, *selectWdg*, *selection*)

static isSuitableFor(*modul*, *handler*, *actionName*)

resetLoadingState(*self*)

class vi.widgets.htmleditor.**HtmlEditor**(**args*, ***kwargs*)

Bases: flare.html5.Textarea

initSources = False

_attachSummernote(*self*, *retry=0*)

onAttach(*self*)

onDetach(*self*)

onEditorChange(*self*, *e*, **args*, ***kwargs*)

_getValue(*self*)

_setValue(*self*, *val*)

enable(*self*)

disable(*self*)

vi.widgets.internaledit

Module Contents

Classes

ParsedErrorItem
PassiveErrorItem
InternalEdit

Functions

checkErrors(bone) → Tuple[bool, List[str]]

class vi.widgets.internaledit.**ParsedErrorItem**(*error*)

Bases: flare.html5.Li

style = []

class vi.widgets.internaledit.**PassiveErrorItem**(*error*)

Bases: flare.html5.Li

style = []

vi.widgets.internaledit.**checkErrors**(*bone*) → Tuple[bool, List[str]]

class vi.widgets.internaledit.**InternalEdit**(*skelStructure*, *values=None*, *errorInformation=None*, *readOnly=False*, *context=None*, *defaultCat=''*, *module=None*, *boneparams=None*, *errorQueue=None*, *prefix=None*)

Bases: flare.html5.Div

renderStructure(*self*, *readOnly=False*)

serializeForPost(*self*, *validityCheck=False*)

serializeForDocument(*self*)

doSave(*self*, *closeOnSuccess=False*, **args*, ***kwargs*)
Starts serializing and transmitting our values to the server.

unserialize(*self*, *data=None*)
Applies the actual data to the bones.

onChange(*self*, *event*)

onKeyDown(*self*, *event*)

performLogics(*self*)

vi.widgets.list

Module Contents

Classes

ListWidget	Provides the interface to list-applications.
ViewportListWidget	Provides the interface to list-applications.

class vi.widgets.list.**ListWidget**(*module*, *filter=None*, *columns=None*, *filterID=None*, *filterDescr=None*, *batchSize=None*, *context=None*, *autoload=True*, **args*, ***kwargs*)

Bases: flare.html5.Div

Provides the interface to list-applications. It acts as a data-provider for a DataTable and binds an action-bar to this table.

setSelector(*self*, *callback*, *multi=True*, *allow=None*)
Configures the widget as selector for a relationalBone and shows it.

selectorReturn(*self*)
Returns the current selection to the callback configured with *setSelector*.

tableInitialization(*self*, **args*, ***kwargs*)
Instantiates the table :param args: ListWidget Parameter :param kwargs: ListWidget Parameter :return:

setAmount(*self*, *amount*)

setPage(*self*, *page=0*)
sets targetpage. if not enougth loadedpages this pages will be requested :param page: sets targetpage :return:

onRequestingFinished(*self*, **args*, ***kwargs*)

onClick(*self*, *event*)

setTableActionBar(*self*)

getDefaultEntryActions(*self*)
Returns the list of actions available in our actionBar

getActions(*self*)
Returns the list of actions available in our actionBar

getAllActions(*self*, *view=None*)
Returns the list of actions available in the action bar.

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displayes an error message

onNextBatchNeeded(*self*)
Requests the next rows from the server and feed them to the table.

onAttach(*self*)

onDetach(*self*)

onDataChanged(*self*, *module*, ***kwargs*)
Refresh our view if element(s) in this module have changed

requestStructure(*self*)

receivedStructure(*self*, *resp*)

reloadData(*self*)
Removes all currently displayed data and refetches the first batch from the server.

setFilter(*self*, *filter*, *filterID=None*, *filterDescr=None*)
Applies a new filter.

setContext(*self*, *context*)
Applies a new context.

getFilter(*self*)

updateEmptyNotification(*self*)

onCompletion(*self*, *req*)
Pass the rows received to the datatable. :param req: The network request that succeed.

setFields(*self*, *fields*)

getFields(*self*)

onSelectionActivated(*self*, *table*, *selection*)

activateSelection(*self*)

static canHandle(*moduleName*, *moduleInfo*)

class vi.widgets.list.**ViewportListWidget**(*module*, *filter=None*, *columns=None*, *filterID=None*, *filterDescr=None*, *batchSize=None*, *context=None*, *autoload=True*, **args*, ***kwargs*)

Bases: *ListWidget*

Provides the interface to list-applications. It acts as a data-provider for a DataTable and binds an action-bar to this table.

tableInitialization(*self*, **args*, ***kwargs*)
Instantiates the table :param args: ListWidget Parameter :param kwargs: ListWidget Parameter

Override explanation

- use ViewPort DataTable with rows parameter

setAmount(*self*, *amount*)

setPage(*self*, *page=0*)
sets targetpage. if not enougth loadedpages this pages will be requested else

Parameters **page** – sets targetpage

Returns

_setPage(*self*, *page=0*)
render page to table :param page: :return:

onRequestingFinished(*self*, **args*, ***kwargs*)

setTableActionBar(*self*)

static canHandle(*moduleName*, *moduleInfo*)

vi.widgets.preview

Module Contents

Classes

Preview

class vi.widgets.preview.**Preview**(*urls*, *entry*, *modul*, **args*, ***kwargs*)
Bases: flare.html5.Div

onChange(*self*, *event*)

setUrl(*self*, *url*)

doClose(*self*, **args*, ***kwargs*)

vi.widgets.repeatdate

Module Contents

Classes

RepeatDatePopup

class vi.widgets.repeatdate.**RepeatDatePopup**(*module*, *key*)
Bases: flare.html5.Div

__editIdx_ = 0

reloadData(*self*)

save(*self*, *data*)

setData(*self*, *request=None*, *data=None*, *ignoreMissing=False*)
Rebuilds the UI according to the skeleton received from server

Parameters

- **request** (*NetworkService*) – A finished NetworkService request
- **data** (*dict*) – The data received

clear(*self*)
Removes all visible bones/forms/fieldsets.

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displayes an error message

doSave(*self*, *closeOnSuccess=False*)

vi.widgets.search

Module Contents

Classes

Search

class vi.widgets.search.**Search**(**args*, ***kwargs*)
Bases: flare.html5.Div

doSearch(*self*, **args*, ***kwargs*)

resetSearch(*self*)

onKeyDown(*self*, *event*)

resetLoadingState(*self*)

reevaluate(*self*)

focus(*self*)

vi.widgets.sidebar

Module Contents

Classes

SideBar

class vi.widgets.sidebar.**SideBar**(**args*, ***kwargs*)
Bases: flare.html5.Div

onAttach(*self*)

onDetach(*self*)

setWidget(*self*, *widget*)

getWidget(*self*)

close(*self*, **args*, ***kwargs*)

vi.widgets.table

Module Contents

Classes

SelectTable	Provides an Html-Table which allows for row selections.
DataTable	Provides kind of MVC on top of SelectTable.

class vi.widgets.table.**SelectTable**(*checkboxes=False*, *indexes=False*, **args*, ***kwargs*)

Bases: flare.ignite.Table

Provides an Html-Table which allows for row selections.

Parent widgets can register for certain events:

- **selectionChanged: called if the current _multi_ selection changes. (Ie the user** holds ctrl and clicks a row). The selection might contain no, one or multiple rows. Its also called if the cursor moves. Its called if the user simply double clicks a row. So its possible to receive a selectionActivated event without an selectionChanged Event.
- **selectionActivated: called if a selection is activated, ie. a row is double-clicked or Return** is pressed.
- **cursorMoved: called when the currently active row changes. The user can select the current row** with a single click or by moving the cursor up and down using the arrow keys.

onAttach(*self*)

setHeader(*self*, *headers*)

Sets the table-headers to 'headers' :param headers: list of strings :type headers: list

getTrByIndex(*self*, *idx*)

Retrieves the TR element by the given row number :param idx: Rownumber to retrieve the tr of :type idx: int :returns: HTMLTableRowElement

getIndexByTr(*self*, *tr*)

Returns the rowNumber for the given tr element or None if the given tr element is invalid. :param tr: A HTMLTableRowElement of this table :type tr: HTMLTableRowElement :returns: int or None

_rowForEvent(*self*, *event*)

Determines the row number for the given event

onChange(*self*, *event*)

onMouseDown(*self*, *event*)

onMouseOut(*self*, *event*)

onMouseUp(*self*, *event*)

onKeyDown(*self*, *event*)

onKeyUp(*self*, *event*)

onDblClick(*self*, *event*)

addSelectedRow(*self*, *row*)

Marks a row as selected

removeSelectedRow(*self*, *row*)
Removes 'row' from the current selection (if any) :param row: Number of the row to unselect :type row: int

selectRow(*self*, *newRow*)
Sets the current selection to 'row'. Any previous selection is removed. :param newRow: Number of the row to select :type newRow: int

setCursorRow(*self*, *row*, *removeExistingSelection=True*)
Move the cursor to row 'row'. If removeExistingSelection is True, the current selection (if any) is invalidated.

focusRow(*self*, *row*)

getCurrentSelection(*self*)
Returns a list of currently selected row-numbers :returns: list

clear(*self*)
Hook the clear() method so we can reset some internal states, too

removeRow(*self*, *row*)
Hook the removeRow method so we can reset some internal states, too

_extraCols(*self*)

prepareCol(*self*, *row*, *col*)
Lets hook up the original removeRow function to optionally provide index and checkbox columns.

setCell(*self*, *row*, *col*, *val*)
Interface for self["cell"] that directs to the correct cell if extra columns are configured for this SelectTable.

selectAll(*self*)
Selects all entries of the table.

unSelectAll(*self*)
Unselects all entries of the table.

invertSelection(*self*)
Inverts the current selection on the whole table currently displayed.

class vi.widgets.table.**DataTable**(*_loadOnDisplay=False*, **args*, ***kwargs*)
Bases: flare.html5.Div

Provides kind of MVC on top of SelectTable.

recalcHeight(*self*, **args*, ***kwargs*)

setDataProvider(*self*, *obj*)
Register's 'obj' as the provider for this table. It must provide a onNextBatchNeeded function, which must fetch and feed new rows using add() or reset the dataProvider to None if no more rows are available. Notice: If the bottom of the table is reached, onNextBatchNeeded will only be called once. No further calls will be made until add() or setDataProvider() has been called afterwards.

onCursorMoved(*self*, *table*, *row*)
Ensure the table scrolls according to the position of its cursor

getRowCount(*self*)
Returns the total amount of rows currently known. :returns: int

add(*self*, *obj*)
Adds an row to the model :param obj: Dictionary of values for this row :type obj: dict

extend(*self*, *objList*)
Adds multiple rows at once. Much faster than calling add() multiple times.

testIfNextBatchNeededImmediately(*self*)
Test if we display enough entries so that our contents are scrollable. Otherwise, we'll never request a second batch

remove(*self*, *objOrIndex*)
Removes 'obj' from the table. 'obj' may be an row-index or an object recieved by any eventListener. It _cannot_ be any original object passed to 'add' - it _must_ be recived by an eventListener!

clear(*self*, *keepModel=False*)
Flushes the whole table.

_renderObject(*self*, *obj*, *tableIsPrepared=False*)
Renders the object to into the table. Does nothing if the list of _shownFields is empty. :param obj: Dictionary of values for this row :type obj: dict

rebuildTable(*self*)
Rebuilds the entire table. Useful if something fundamental changed (ie. the cell renderer or the list of visible fields)

setShownFields(*self*, *fields*)
Sets the list of _shownFields. This causes the whole table to be rebuild. Be careful if calling this function often on a large table! :param fields: List of model-keys which will be displayed. :type fields: list

onScroll(*self*, *event*)
Check if we got a scroll event and need to fetch another set of rows from our dataProvider

onSelectionChanged(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onSelectionActivated(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onTableChanged(*self*, *table*, *rowCount*)
Re-emit the event.

getCurrentSelection(*self*)
Override the getCurrentSelection method to yield actual models, not row-numbers.

setCellRender(*self*, *field*, *render*)
Sets the render for cells of 'field' to render. A cell render receives the data for a given cell and returns the appropriate widget to display that data for the table.

setCellRenders(*self*, *renders*)
Like setCellRender, but sets multiple renders at one. Much faster than calling setCellRender repeatedly.

activateSelection(*self*)
Emits the selectionActivated event if there's currently a selection

vi.widgets.task

Module Contents

Classes

TaskWidget

continues on next page

Table 36 – continued from previous page

ServerTaskWidget	
TaskSelectWidget	

class vi.widgets.task.**TaskWidget**(*title*)
Bases: flare.popup.Popup

class vi.widgets.task.**ServerTaskWidget**(*title*, *key*)
Bases: *TaskWidget*

class vi.widgets.task.**TaskSelectWidget**
Bases: *TaskWidget*

getSelectedTask(*self*)

setActiveTask(*self*)

onChange(*self*, *event*)

invokeTask(*self*, **args*, ***kwargs*)

vi.widgets.tooltip

Module Contents

Classes

ToolTip	Small utility class for providing tooltips

class vi.widgets.tooltip.**ToolTip**(*shortText=''*, *longText=''*, **args*, ***kwargs*)
Bases: flare.html5.Div

Small utility class for providing tooltips

onClick(*self*, *event*)

_setDisabled(*self*, *disabled*)

_getDisabled(*self*)

vi.widgets.topbar

Module Contents

Classes

TopBarWidget	Provides the top-bar of VI
UserState	
Tasks	

continues on next page

Table 38 – continued from previous page

Logout	

class vi.widgets.topbar.**TopBarWidget**
Bases: flare.html5.Header

Provides the top-bar of VI

invoke(*self*)

setTitle(*self*, *title=None*)

onClick(*self*, *event*)

setCurrentModulDescr(*self*, *descr=''*, *iconURL=None*, *iconClasses=None*, *path=None*)

class vi.widgets.topbar.**UserState**(**args*, ***kwargs*)
Bases: flare.html5.Div

onCurrentUserAvailable(*self*, *req*)

update(*self*)

static canHandle(*action*)

onClick(*self*, *sender=None*)

openEdit(*self*, *key*)

class vi.widgets.topbar.**Tasks**(**args*, ***kwargs*)
Bases: flare.button.Button

onTaskListAvailable(*self*, *req*)

onTaskListFailure(*self*)

onCurrentUserAvailable(*self*, *req*)

update(*self*)

onClick(*self*, *event*)

static canHandle(*action*)

class vi.widgets.topbar.**Logout**(**args*, ***kwargs*)
Bases: flare.button.Button

onClick(*self*, *event*)

logout(*self*)

static canHandle(*action*)

vi.widgets.tree

Module Contents

Classes

TreeWidget	Base Widget that renders a tree.

continues on next page

Table 39 – continued from previous page

BrowserLeafWidget	
BrowserNodeWidget	
BreadcrumbNodeWidget	
TreeBrowserWidget	Base Widget that renders a tree.

class vi.widgets.tree.**TreeWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: flare.html5.Div

Base Widget that renders a tree.

nodeWidget

leafWidget

requestStructure(*self*)

receivedStructure(*self*, *resp*)

setSelector(*self*, *callback*, *multi=True*, *allow=None*)
Configures the widget as selector for a relationalBone and shows it.

selectorReturn(*self*)
Returns the current selection to the callback configured with *setSelector*.

onKeyDown(*self*, *event*)

onKeyUp(*self*, *event*)

getActions(*self*)
Returns a list of actions that are being set for the ActionBar. Override this to provide additional actions.

clearSelection(*self*)
Empties the current selection.

extendSelection(*self*, *element*)
Extends the current selection to element.

This is normally done by clicking or tabbing on an element.

activateSelection(*self*, *element*)
Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

requestChildren(*self*, *element*)

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displayes an error message

onDataChanged(*self*, *module*, ***kwargs*)

onAttach(*self*)

onDetach(*self*)

itemForKey(*self*, *key*, *elem=None*)
Returns the HierarchyWidget displaying the entry with the given key. :param key: The key (id) of the item. :type key: str :returns: HierarchyItem

onSetDefaultRootNode(*self*, *req*)
We requested the list of rootNodes for that module and that request just finished. Parse the respone and set our rootNode to the first rootNode received.

setRootNode(*self*, *rootNode*, *node=None*)
Set the currently displayed hierarchy to 'rootNode'. :param rootNode: Key of the rootNode which children we shall display :type rootNode: str

reloadData(*self*)
Reload the data were displaying.

loadNode(*self*, *node*, *cursor=None*, *reqType=None*, *overrideParams=None*)
Fetch the (direct) children of the given node. Once the list is received, append them to their parent node. :param node: Key of the node to fetch :type node: str

onRequestSucceded(*self*, *req*)
The NetworkRequest for a (sub)node finished. Create a new HierarchyItem for each entry received and add them to our view

onDrop(*self*, *event*)
We got a drop event. Make that item a direct child of our rootNode

onDragOver(*self*, *event*)
Allow dropping children on the rootNode

getChildKey(*self*, *widget*)
Order by sortindex

static canHandle(*moduleName*, *moduleInfo*)

class vi.widgets.tree.**BrowserLeafWidget**
Bases: flare.forms.widgets.tree.TreeLeafWidget

setStyle(*self*)

class vi.widgets.tree.**BrowserNodeWidget**
Bases: flare.forms.widgets.tree.TreeNodeWidget

setStyle(*self*)

class vi.widgets.tree.**BreadcrumbNodeWidget**
Bases: flare.forms.widgets.tree.TreeNodeWidget

setStyle(*self*)

class vi.widgets.tree.**TreeBrowserWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)
Bases: *TreeWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

reloadData(*self*)
Reload the data were displaying.

rebuildPath(*self*)
Rebuild the displayed path-list.

onPathRequestSucceded(*self*, *req*)
Rebuild the displayed path-list according to request data

activateSelection(*self*, *element*)
Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

static canHandle(*module*, *moduleInfo*)

vi.widgets.userlogoutmsg

Module Contents

Classes

UserLogoutMsg

class vi.widgets.userlogoutmsg.**UserLogoutMsg**(**args*, ***kwargs*)
Bases: flare.popup.Popup

pollInterval = 120

checkIntervall

visibilityChanged(*self*, *e*)

stopInterval(*self*)

hideMessage(*self*)
Make this popup invisible

showMessage(*self*)
Show this popup

showLoginWindow(*self*, **args*, ***kwargs*)
Return to the login window.

checkForSuspendResume(*self*, **args*, ***kwargs*)
Test if at least self.pollIntervall seconds have passed and query the server if

startPolling(*self*, **args*, ***kwargs*)
Start querying the server

onUserTestSuccess(*self*, *req*)
We received a response from the server

onUserTestFail(*self*, *text*, *ns*)
Error retrieving the current user response from the server

Package Contents

Classes

Accordion	
TopBarWidget	Provides the top-bar of VI
ListWidget	Provides the interface to list-applications.
EditWidget	
ToolTip	Small utility class for providing tooltips
DataTable	Provides kind of MVC on top of SelectTable.
Search	
SideBar	
UserLogoutMsg	
TaskWidget	
TaskSelectWidget	
InternalEdit	
HtmlEditor	
ExportCsv	
ExportCsvStarter	
TreeWidget	Base Widget that renders a tree.
TreeBrowserWidget	Base Widget that renders a tree.
HierarchyWidget	A Hierarchy is a Tree without leaf distiction!
FileWidget	Base Widget that renders a tree.

class vi.widgets.**Accordion**

Bases: flare.html5.Form

_segments = []

addSegment(*self*, *ident*, *title=None*, *directAdd=False*, **args*)

clearSegments(*self*)

buildAccordion(*self*, *order=None*)

Parameters **sort** – None: sorted by Bones, "asc":ascending, "desc":descending, dict: {"category":index,...}

Returns

class vi.widgets.**TopBarWidget**

Bases: flare.html5.Header

Provides the top-bar of VI

invoke(*self*)

setTitle(*self*, *title=None*)

onClick(*self*, *event*)

setCurrentModulDescr(*self*, *descr=''*, *iconURL=None*, *iconClasses=None*, *path=None*)

class vi.widgets.**ListWidget**(*module*, *filter=None*, *columns=None*, *filterID=None*, *filterDescr=None*, *batchSize=None*, *context=None*, *autoload=True*, **args*, ***kwargs*)

Bases: flare.html5.Div

Provides the interface to list-applications. It acts as a data-provider for a DataTable and binds an action-bar to this table.

setSelector(*self*, *callback*, *multi=True*, *allow=None*)
Configures the widget as selector for a relationalBone and shows it.

selectorReturn(*self*)
Returns the current selection to the callback configured with *setSelector*.

tableInitialization(*self*, **args*, ***kwargs*)
Instantiates the table :param args: ListWidget Parameter :param kwargs: ListWidget Parameter :return:

setAmount(*self*, *amount*)

setPage(*self*, *page=0*)
sets targetpage. if not enougth loadedpages this pages will be requested :param page: sets targetpage :return:

onRequestingFinished(*self*, **args*, ***kwargs*)

onClick(*self*, *event*)

setTableActionBar(*self*)

getDefaultEntryActions(*self*)
Returns the list of actions available in our actionBar

getActions(*self*)
Returns the list of actions available in our actionBar

getAllActions(*self*, *view=None*)
Returns the list of actions available in the action bar.

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displayes an error message

onNextBatchNeeded(*self*)
Requests the next rows from the server and feed them to the table.

onAttach(*self*)

onDetach(*self*)

onDataChanged(*self*, *module*, ***kwargs*)
Refresh our view if element(s) in this module have changed

requestStructure(*self*)

receivedStructure(*self*, *resp*)

reloadData(*self*)
Removes all currently displayed data and refetches the first batch from the server.

setFilter(*self*, *filter*, *filterID=None*, *filterDescr=None*)
Applies a new filter.

setContext(*self*, *context*)
Applies a new context.

getFilter(*self*)

updateEmptyNotification(*self*)

onCompletion(*self*, *req*)
Pass the rows received to the datatable. :param req: The network request that succeed.

setFields(*self*, *fields*)

getFields(*self*)

onSelectionActivated(*self*, *table*, *selection*)

activateSelection(*self*)

static canHandle(*moduleName*, *moduleInfo*)

class vi.widgets.EditWidget(*module*, *applicationType*, *key=0*, *node=None*, *skelType=None*, *clone=False*, *hashArgs=None*, *context=None*, *logAction='Entry saved!'*, **args*, ***kwargs*)
Bases: flare.html5.Div

appList = list

appHierarchy = hierarchy

appTree = tree

appSingleton = singleton

__editIdx_ = 0

onDetach(*self*)

onAttach(*self*)

performLogics(*self*)

onChange(*self*, *event*)

onBoneChange(*self*, *bone*)

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displays an error message

reloadData(*self*)

save(*self*, *data*)
Creates the actual NetworkService request used to transmit our data. If data is None, it fetches a clean add/edit form.

Parameters **data** (*dict or None*) – The values to transmit or None to fetch a new, clean add/edit form.

clear(*self*)
Removes all visible bones/forms/fieldsets.

closeOrContinue(*self*, *sender=None*)

doCloneHierarchy(*self*, *sender=None*)

cloneComplete(*self*, *request*)

formatReadFromClientErrorSeverity(*self*, *error*)

setData(*self*, *request=None*, *data=None*, *ignoreMissing=False*, *askHierarchyCloning=True*)
Rebuilds the UI according to the skeleton received from server

Parameters

- **request** (*NetworkService*) – A finished NetworkService request
- **data** (*dict*) – The data received

unserialize(*self*, *data=None*, *errors=()*)
Applies the actual data to the bones.

serializeForPost(*self*, *validityCheck=False*)

serializeForDocument(*self*)

doSave(*self*, *closeOnSuccess=False*, **args*, ***kwargs*)
Starts serializing and transmitting our values to the server.

class vi.widgets.**ToolTip**(*shortText=''*, *longText=''*, **args*, ***kwargs*)
Bases: flare.html5.Div

Small utility class for providing tooltips

onClick(*self*, *event*)

_setDisabled(*self*, *disabled*)

_getDisabled(*self*)

class vi.widgets.**DataTable**(*_loadOnDisplay=False*, **args*, ***kwargs*)
Bases: flare.html5.Div

Provides kind of MVC on top of SelectTable.

recalcHeight(*self*, **args*, ***kwargs*)

setDataProvider(*self*, *obj*)
Register's 'obj' as the provider for this table. It must provide a onNextBatchNeeded function, which must fetch and feed new rows using add() or reset the dataProvider to None if no more rows are available. Notice: If the bottom of the table is reached, onNextBatchNeeded will only be called once. No further calls will be made until add() or setDataProvider() has been called afterwards.

onCursorMoved(*self*, *table*, *row*)
Ensure the table scrolls according to the position of its cursor

getRowCount(*self*)
Returns the total amount of rows currently known. :returns: int

add(*self*, *obj*)
Adds an row to the model :param obj: Dictionary of values for this row :type obj: dict

extend(*self*, *objList*)
Adds multiple rows at once. Much faster than calling add() multiple times.

testIfNextBatchNeededImmediately(*self*)
Test if we display enough entries so that our contents are scrollable. Otherwise, we'll never request a second batch

remove(*self*, *objOrIndex*)
Removes 'obj' from the table. 'obj' may be an row-index or an object recieved by any eventListener. It _cannot_ be any original object passed to 'add' - it _must_ be recived by an eventListener!

clear(*self*, *keepModel=False*)
Flushes the whole table.

_renderObject(*self*, *obj*, *tableIsPrepared=False*)
Renders the object to into the table. Does nothing if the list of _shownFields is empty. :param obj: Dictionary of values for this row :type obj: dict

rebuildTable(*self*)
Rebuilds the entire table. Useful if something fundamental changed (ie. the cell renderer or the list of visible fields)

setShownFields(*self*, *fields*)
Sets the list of _shownFields. This causes the whole table to be rebuild. Be careful if calling this function often on a large table! :param fields: List of model-keys which will be displayed. :type fields: list

onScroll(*self*, *event*)
Check if we got a scroll event and need to fetch another set of rows from our dataProvider

onSelectionChanged(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onSelectionActivated(*self*, *table*, *rows*)
Re-emit the event. Maps row-numbers to actual models.

onTableChanged(*self*, *table*, *rowCount*)
Re-emit the event.

getCurrentSelection(*self*)
Override the getCurrentSelection method to yield actual models, not row-numbers.

setCellRender(*self*, *field*, *render*)
Sets the render for cells of 'field' to render. A cell render receives the data for a given cell and returns the appropriate widget to display that data for the table.

setCellRenders(*self*, *renders*)
Like setCellRender, but sets multiple renders at one. Much faster than calling setCellRender repeatedly.

activateSelection(*self*)
Emits the selectionActivated event if there's currently a selection

class vi.widgets.Search(**args*, ***kwargs*)
Bases: flare.html5.Div

doSearch(*self*, **args*, ***kwargs*)

resetSearch(*self*)

onKeyDown(*self*, *event*)

resetLoadingState(*self*)

reevaluate(*self*)

focus(*self*)

class vi.widgets.SideBar(**args*, ***kwargs*)
Bases: flare.html5.Div

onAttach(*self*)

onDetach(*self*)

setWidget(*self*, *widget*)

getWidget(*self*)

close(*self*, **args*, ***kwargs*)

class vi.widgets.**UserLogoutMsg**(**args*, ***kwargs*)
Bases: flare.popup.Popup

pollInterval = 120

checkIntervall

visibilityChanged(*self*, *e*)

stopInterval(*self*)

hideMessage(*self*)
Make this popup invisible

showMessage(*self*)
Show this popup

showLoginWindow(*self*, **args*, ***kwargs*)
Return to the login window.

checkForSuspendResume(*self*, **args*, ***kwargs*)
Test if at least self.pollIntervall seconds have passed and query the server if

startPolling(*self*, **args*, ***kwargs*)
Start querying the server

onUserTestSuccess(*self*, *req*)
We received a response from the server

onUserTestFail(*self*, *text*, *ns*)
Error retrieving the current user response from the server

class vi.widgets.**TaskWidget**(*title*)
Bases: flare.popup.Popup

class vi.widgets.**TaskSelectWidget**
Bases: *TaskWidget*

getSelectedTask(*self*)

setActiveTask(*self*)

onChange(*self*, *event*)

invokeTask(*self*, **args*, ***kwargs*)

class vi.widgets.**InternalEdit**(*skelStructure*, *values=None*, *errorInformation=None*, *readOnly=False*, *context=None*, *defaultCat=''*, *module=None*, *boneparams=None*, *errorQueue=None*, *prefix=None*)
Bases: flare.html5.Div

renderStructure(*self*, *readOnly=False*)

serializeForPost(*self*, *validityCheck=False*)

serializeForDocument(*self*)

doSave(*self*, *closeOnSuccess=False*, **args*, ***kwargs*)
Starts serializing and transmitting our values to the server.

unserialize(*self*, *data=None*)
Applies the actual data to the bones.

onChange(*self*, *event*)

onKeyDown(*self*, *event*)

performLogics(*self*)

class vi.widgets.**HtmlEditor**(**args*, ***kwargs*)

Bases: flare.html5.Textarea

initSources = False

_attachSummernote(*self*, *retry=0*)

onAttach(*self*)

onDetach(*self*)

onEditorChange(*self*, *e*, **args*, ***kwargs*)

_getValue(*self*)

_setValue(*self*, *val*)

enable(*self*)

disable(*self*)

class vi.widgets.**ExportCsv**(*widget*, *selection*, *encoding=None*, *language=None*, *separator=None*, *lineSeparator=None*, **args*, ***kwargs*)

Bases: flare.html5.Progress

nextChunk(*self*, *cursor=None*)

nextChunkComplete(*self*, *req*)

exportToFile(*self*)

nextChunkFailure(*self*, *req*, *code*)

replaceWithMessage(*self*, *message*, *logClass='success'*)

class vi.widgets.**ExportCsvStarter**(*widget*, **args*, ***kwargs*)

Bases: flare.popup.Popup

onExportBtnClick(*self*, **args*, ***kwargs*)

class vi.widgets.**TreeWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: flare.html5.Div

Base Widget that renders a tree.

nodeWidget

leafWidget

requestStructure(*self*)

receivedStructure(*self*, *resp*)

setSelector(*self*, *callback*, *multi=True*, *allow=None*)

Configures the widget as selector for a relationalBone and shows it.

selectorReturn(*self*)

Returns the current selection to the callback configured with *setSelector*.

onKeyDown(*self*, *event*)

onKeyUp(*self*, *event*)

getActions(*self*)

Returns a list of actions that are being set for the ActionBar. Override this to provide additional actions.

clearSelection(*self*)
Empties the current selection.

extendSelection(*self*, *element*)
Extends the current selection to element.

This is normally done by clicking or tabbing on an element.

activateSelection(*self*, *element*)
Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

requestChildren(*self*, *element*)

showErrorMsg(*self*, *req=None*, *code=None*)
Removes all currently visible elements and displayes an error message

onDataChanged(*self*, *module*, ***kwargs*)

onAttach(*self*)

onDetach(*self*)

itemForKey(*self*, *key*, *elem=None*)
Returns the HierarchyWidget displaying the entry with the given key. :param key: The key (id) of the item. :type key: str :returns: HierarchyItem

onSetDefaultRootNode(*self*, *req*)
We requested the list of rootNodes for that module and that request just finished. Parse the respone and set our rootNode to the first rootNode received.

setRootNode(*self*, *rootNode*, *node=None*)
Set the currently displayed hierarchy to 'rootNode'. :param rootNode: Key of the rootNode which children we shall display :type rootNode: str

reloadData(*self*)
Reload the data were displaying.

loadNode(*self*, *node*, *cursor=None*, *reqType=None*, *overrideParams=None*)
Fetch the (direct) children of the given node. Once the list is received, append them to their parent node. :param node: Key of the node to fetch :type node: str

onRequestSucceeded(*self*, *req*)
The NetworkRequest for a (sub)node finished. Create a new HierarchyItem for each entry received and add them to our view

onDrop(*self*, *event*)
We got a drop event. Make that item a direct child of our rootNode

onDragOver(*self*, *event*)
Allow dropping children on the rootNode

getChildKey(*self*, *widget*)
Order by sortindex

static canHandle(*moduleName*, *moduleInfo*)

class vi.widgets.**TreeBrowserWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: *TreeWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

reloadData(*self*)
Reload the data were displaying.

rebuildPath(*self*)
Rebuild the displayed path-list.

onPathRequestSucceded(*self*, *req*)
Rebuild the displayed path-list according to request data

activateSelection(*self*, *element*)
Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

static canHandle(*module*, *moduleInfo*)

class vi.widgets.**HierarchyWidget**(**args*, ***kwargs*)
Bases: *vi.widgets.tree.TreeWidget*

A Hierarchy is a Tree without leaf distiction!

leafWidget

reloadData(*self*)
Reload the data were displaying.

reloadListWidget(*self*)

toggleListView(*self*)

setListView(*self*, *visible=False*)

showListView(*self*)

hideListView(*self*)

onSelectionChanged(*self*, *widget*, *selection*)

static canHandle(*moduleName*, *moduleInfo*)

class vi.widgets.**FileWidget**(*module*, *rootNode=None*, *selectMode=None*, *node=None*, *context=None*, **args*, ***kwargs*)
Bases: *vi.widgets.tree.TreeBrowserWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

searchWidget(*self*)

onStartSearch(*self*, *searchStr*, **args*, ***kwargs*)

getChildKey(*self*, *widget*)
Derives a string used to sort the entries on each level

onDrop(*self*, *event*)
We got a drop event. Make that item a direct child of our rootNode

static canHandle(*module*, *moduleInfo*)

Submodules

vi.admin

Module Contents

Classes

AdminScreen	This is the screen superclass.

Attributes

viInitializedEvent

class vi.admin.**AdminScreen**(**args*, ***kwargs*)

Bases: vi.screen.Screen

This is the screen superclass.

It represents a basic screen and its functionality.

onClick(*self*, *event*)

reset(*self*)

invoke(*self*)

Is called to show the screen

getCurrentUser(*self*)

getCurrentUserSuccess(*self*, *req*)

getCurrentUserFailure(*self*, *req*, *code*)

startup(*self*)

initializeViews(*self*)

initializeConfig(*self*)

appendNavList(*self*, *NavList*, *target*, *parentInfo=()*)

openView(*self*, *name*, *icon*, *viewName*, *moduleName*, *actionName*, *data*, *focusView=True*, *append=False*, *target='mainNav'*)

openNewMainView(*self*, *name*, *icon*, *viewName*, *moduleName*, *actionName*, *data*, *focusView=True*, *append=False*)

openNewPopup(*self*, *name*, *icon*, *viewName*, *moduleName*, *actionName*, *data*, *focusView=True*, *append=False*)

log(*self*, *type*, *msg*, *icon=None*, *modul=None*, *action=None*, *key=None*, *data=None*)

checkInitialHash(*self*, **args*, ***kwargs*)

execCall(*self*, *path*, *params=None*)

Performs an execution call.

Parameters

- **path** – Path to the module and action
- **params** – Parameters passed to the module

stackWidget(*self*, *widget*, *disableOtherWidgets=True*)
We dont stack widgets anymore. We use now Popups.

removeWidget(*self*, *widget*)

switchFullscreen(*self*, *fullscreen=True*)

isFullscreen(*self*)

onError(*self*, *req*, *code*)

vi.admin.**viInitializedEvent**

vi.config

Module Contents

Functions

updateConf(_conf)
getConf()

Attributes

vi_conf
conf

vi.config.**vi_conf**

vi.config.**updateConf**(*_conf*)

vi.config.**getConf**()

vi.config.**conf**

vi.exception

Module Contents

exception vi.exception.**InvalidBoneValueException**
Bases: ValueError

Inappropriate argument value (of correct type).

vi.log

Module Contents

Classes

logEntry	PopOut Elements
logA	click handler for loglist
logWidget	
LogButton	
Log	Provides the "messaging" center displayed at the bottom of VI

Attributes

iddbTableName

vi.log.**iddbTableName = vi_log3**

class vi.log.**logEntry**(*logObj=None*)
Bases: flare.html5.Span

PopOut Elements

class vi.log.**logA**(*logObj=None*)
Bases: flare.html5.A

click handler for loglist

onClick(*self*, *sender=None*)

openEditor(*self*, *key*)

class vi.log.**logWidget**(*logList*)
Bases: flare.html5.Div

builDataTable(*self*)

class vi.log.**LogButton**
Bases: flare.html5.Div

idbdata(*self*, *event*)

cleanLog(*self*)

cleanLogAction(*self*, *event*)

renderPopOut(*self*)

onClick(*self*, *sender=None*)

openLog(*self*)

apane = Pane(translate("Log"), closeable=True, iconClasses=["apptype_list"], collapseable=True

)

wg = logWidget(self.logsList)

apane.addWidget(wg)

conf["mainWindow"].addPane(apane) conf["mainWindow"].focusPane(apane)

log(*self*, *type*, *msg*, *icon=None*, *modul=None*, *action=None*, *key=None*, *data=None*, *date=None*, *onlyLoad=False*)

msgOverlay(*self*, *logObj*)

removeInfo(*self*, *wrap*)

reset(*self*)

static canHandle(*action*)

class vi.log.Log

Bases: flare.html5.Div

Provides the "messaging" center displayed at the bottom of VI

toggleMsgCenter(*self*, **args*, ***kwargs*)

log(*self*, *type*, *msg*, *icon=None*, *date=None*)

Adds a message to the log :param type: The type of the message. :type type: "success", "error", "warning", "info", "progress" :param msg: The message to append :type msg: str

removeNewCls(*self*, *span*)

reset(*self*)

vi.login

Module Contents

Classes

LoginInputField
BaseLoginHandler
UserPasswordLoginHandler

continues on next page

Table 48 – continued from previous page

GoogleAccountLoginHandler	
LoginScreen	This is the screen superclass.

class vi.login.**LoginInputField**(*notifier*, **args*, ***kwargs*)
Bases: flare.html5.Input

onKeyPress(*self*, *event*)

class vi.login.**BaseLoginHandler**(*loginScreen*, **args*, ***kwargs*)
Bases: flare.html5.Li

onClick(*self*, *event*)

enable(*self*)

disable(*self*)

lock(*self*)

unlock(*self*)

login(*self*)

reset(*self*)

parseAnswer(*self*, *req*)

class vi.login.**UserPasswordLoginHandler**(*loginScreen*, **args*, ***kwargs*)
Bases: *BaseLoginHandler*

cssname = userpassword

onKeyPress(*self*, *event*)

onLoginClick(*self*, *sender=None*)

doLoginSuccess(*self*, *req*)

doLoginFailure(*self*, *req*, *code*, **args*, ***kwargs*)

onVerifyClick(*self*, *sender=None*)

doVerifySuccess(*self*, *req*)

doVerifyFailure(*self*, **args*, ***kwargs*)

onSendClick(*self*, *sender=None*)

reset(*self*)

enable(*self*)

focusLaterIdiot(*self*)

static canHandle(*method*, *secondFactor*)

class vi.login.**GoogleAccountLoginHandler**(*loginScreen*, **args*, ***kwargs*)
Bases: *BaseLoginHandler*

cssname = googleaccount

onLoginClick(*self*, *sender=None*)

static canHandle(*method*, *secondFactor*)

class vi.login.**LoginScreen**(**args*, ***kwargs*)

Bases: *vi.screen.Screen*

This is the screen superclass.

It represents a basic screen and its functionality.

invoke(*self*, *logout=False*)

Is called to show the screen

onLogoutSuccess(*self*, **args*, ***kwargs*)

doShowLogin(*self*, *req*, *code*, **args*, ***kwargs*)

insufficientRights(*self*)

doSkipLogin(*self*, *req*)

onGetAuthMethodsSuccess(*self*, *req*)

selectHandler(*self*, *handler=None*)

onGetAuthMethodsFailure(*self*, **args*, ***kwargs*)

redirectNoAdmin(*self*)

vi.pane

Module Contents

Classes

Pane	Base class for Panes.
GroupPane	This pane groups subpanes; it cannot have direct childrens

class vi.pane.**Pane**(*descr=None*, *iconURL=None*, *iconClasses=None*, *closeable=False*, *collapseable=True*, *focusable=True*, *path=None*)

Bases: `flare.html5.Div`

Base class for Panes.

A pane represents a entry in the module list as well as a list of widgets associated with this pane.

It is possible to stack panes on-top of each other. If a pane is active, _all_ its child widgets are visible (through they might overlap).

__setattr__(*self*, *key*, *value*)

setImage(*self*, *loading=False*)

lock(*self*)

unlock(*self*)

setText(*self*, *descr=None*, *iconURL=None*, *loading=False*)

onBtnCloseReleased(*self*, **args*, ***kwargs*)

addChildPane(*self*, *pane*)

Stack a pane under this one. It gets displayed as a subpane. :param pane: Another pane :type pane: pane

removeChildPane(*self*, *pane*)
Removes a subpane. :param pane: The pane to remove. Must be a direct child of this pane :type pane: Pane

onDetach(*self*)

addWidget(*self*, *widget*, *disableOtherWidgets=True*)
Adds a widget to this pane. Note: all widgets of a pane are visible at the same time! :param widget: The widget to add :type widget: Widget

rebuildChildrenClassInfo(*self*)

removeWidget(*self*, *widget*)
Removes a widget. :param widget: The widget to remove. Must be a direct child of this pane. :type widget: Widget

containsWidget(*self*, *widget*)
Tests wherever widget is a direct child of this pane. :returns: bool

onClick(*self*, *event=None*, **args*, ***kwargs*)

expand(*self*)

collapse(*self*)

focus(*self*)

class vi.pane.**GroupPane**(**args*, ***kwargs*)
Bases: *Pane*

This pane groups subpanes; it cannot have direct childrens

loadChildren(*self*)

DeferredLoadChildren(*self*, *delay=1000*)

onClick(*self*, *event=None*, **args*, ***kwargs*)

expand(*self*)

collapse(*self*)

onFocus(*self*, *event*)

vi.priorityqueue

Module Contents

Classes

StartupQueue

Attributes

startupQueue
HandlerClassSelector
actionDelegateSelector
initialHashHandler
extendedSearchWidgetSelector
toplevelActionSelector
loginHandlerSelector
protocolWrapperClassSelector
protocolWrapperInstanceSelector

class vi.priorityqueue.**StartupQueue**

Bases: object

reset(*self*)

setFinalElem(*self*, *elem*)

insertElem(*self*, *priority*, *elem*)

run(*self*)

next(*self*)

vi.priorityqueue.**startupQueue**

vi.priorityqueue.**HandlerClassSelector**

vi.priorityqueue.**actionDelegateSelector**

vi.priorityqueue.**initialHashHandler**

vi.priorityqueue.**extendedSearchWidgetSelector**

vi.priorityqueue.**toplevelActionSelector**

vi.priorityqueue.**loginHandlerSelector**

vi.priorityqueue.**protocolWrapperClassSelector**

vi.priorityqueue.**protocolWrapperInstanceSelector**

vi.screen

Module Contents

Classes

Screen	This is the screen superclass.

class vi.screen.**Screen**(**args*, ***kwargs*)
Bases: flare.html5.Div

This is the screen superclass.

It represents a basic screen and its functionality.

lock(*self*)

unlock(*self*)

invoke(*self*)
Is called to show the screen

remove(*self*)
Remove the screen from its parent

setTitle(*self*, *title=None*)

vi.serversideaction

Module Contents

Classes

ServerSideActionWdg

class vi.serversideaction.**ServerSideActionWdg**(*module*, *handler*, *actionName*, *actionData*)
Bases: flare.button.Button

switchDisabledState(*self*, *disabled*)

onAttach(*self*)

onDetach(*self*)

onSelectionChanged(*self*, *table*, *selection*)

onClick(*self*, *sender=None*)

fetchNext(*self*)

fetchSucceeded(*self*, *req*)

fetchFailed(*self*, *req*, *code*)

resetLoadingState(*self*)

vi.utils

Module Contents

Classes

indexeddbConnector	
indexeddb	

Functions

formatString(format, data, structure=None, language=None)	Parses a string given by format and substitutes placeholders using values specified by data.
getImagePreview(data, cropped=False, size=150)	
setPreventUnloading(mode=True)	
mergeDict(original, target)	

vi.utils.**formatString**(*format*, *data*, *structure=None*, *language=None*)
Parses a string given by format and substitutes placeholders using values specified by data.

vi.utils.**getImagePreview**(*data*, *cropped=False*, *size=150*)

vi.utils.**setPreventUnloading**(*mode=True*)

class vi.utils.**indexeddbConnector**(*dbName*, *version=None*)

dbResult

dbTransaction

connect(*self*)

db_error(*self*, *event*)

db_blocked(*self*, *events*)

db_version(*self*, *event*)

db_onupgradeneeded(*self*, *event*)

db_success(*self*, *event*)

class vi.utils.**indexeddb**(*dbName*, *dbVersion=None*)

queue = []

dbqueue = []

connect(*self*)

getList(*self*, *name*)

_getList(*self*, *event*)

getListKeys(*self*, *name*)

_getListKey(*self*, *event*)

db_success(*self*, *event*)

dbAction(*self*, *action*, *name*, *key=None*, *obj=None*)

_processDbUpdate(*self*, *event*)

_processQueue(*self*, *event*)

_writeToStore(*self*, *item*, *dbResult*, *dbTransaction*)

_deleteFromStore(*self*, *item*, *dbResult*, *dbTransaction*)

_updateToStore(*self*, *item*, *dbResult*, *dbTransaction*)

_deleteObjectStore(*self*, *item*, *dbResult*, *dbTransaction*)

_registerObjectStore(*self*, *item*, *dbResult*, *dbTransaction*)

vi.utils.**mergeDict**(*original*, *target*)

Package Contents

Classes

LoginScreen	This is the screen superclass.
AdminScreen	This is the screen superclass.
Application	

Functions

start()

Attributes

vi_conf
conf
s
a
d

continues on next page

Table 58 – continued from previous page

sc
scinv
s

vi.vi_conf

class vi.LoginScreen(*args, **kwargs)

Bases: vi.screen.Screen

This is the screen superclass.

It represents a basic screen and its functionality.

invoke(self, logout=False)

Is called to show the screen

onLogoutSuccess(self, *args, **kwargs)

doShowLogin(self, req, code, *args, **kwargs)

insufficientRights(self)

doSkipLogin(self, req)

onGetAuthMethodsSuccess(self, req)

selectHandler(self, handler=None)

onGetAuthMethodsFailure(self, *args, **kwargs)

redirectNoAdmin(self)

class vi.AdminScreen(*args, **kwargs)

Bases: vi.screen.Screen

This is the screen superclass.

It represents a basic screen and its functionality.

onClick(self, event)

reset(self)

invoke(self)

Is called to show the screen

getCurrentUser(self)

getCurrentUserSuccess(self, req)

getCurrentUserFailure(self, req, code)

startup(self)

initializeViews(self)

initializeConfig(self)

appendNavList(self, NavList, target, parentInfo=())

openView(self, name, icon, viewName, moduleName, actionName, data, focusView=True, append=False, target='mainNav')

openNewMainView(*self*, *name*, *icon*, *viewName*, *moduleName*, *actionName*, *data*, *focusView=True*, *append=False*)

openNewPopup(*self*, *name*, *icon*, *viewName*, *moduleName*, *actionName*, *data*, *focusView=True*, *append=False*)

log(*self*, *type*, *msg*, *icon=None*, *modul=None*, *action=None*, *key=None*, *data=None*)

checkInitialHash(*self*, **args*, ***kwargs*)

execCall(*self*, *path*, *params=None*)
Performs an execution call.

Parameters

- **path** – Path to the module and action
- **params** – Parameters passed to the module

stackWidget(*self*, *widget*, *disableOtherWidgets=True*)
We dont stack widgets anymore. We use now Popups.

removeWidget(*self*, *widget*)

switchFullscreen(*self*, *fullscreen=True*)

isFullscreen(*self*)

onError(*self*, *req*, *code*)

vi.**conf**

class vi.**Application**
Bases: flare.html5.Div

startup(*self*, **args*, ***kwargs*)

getVersionSuccess(*self*, *req*)

getConfigSuccess(*self*, *req*)

startupFailure(*self*, *req*, *err*)

login(*self*, *logout=False*)

admin(*self*)

logout(*self*)

setTitle(*self*, *title=None*)

setPath(*self*, *path=''*)

vi.**start**()

vi.**s**

vi.**a**

vi.**d**

vi.**sc**

vi.**scinv**

vi.**s**

PYTHON MODULE INDEX

V

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V