WHITESTEIN Technologies

Living Systems® Process Suite

Standard Library

Living Systems Process Suite Documentation

3.3
Mon Nov 1 2021

Whitestein Technologies AG | Hinterbergstrasse 20 | CH-6330 Cham
Tel +41 44-256-5000 | Fax +41 44-256-5001 | http://www.whitestein.com

Contents

1 Standard Library **1**

2 Module uicommon **3**

2.1 Data Types . . . 3

2.1.1 Uicommon . . . 3

3 Module ui **7**

3.1 Data Types . . . 7

3.1.1 Charts . . . 7

3.1.2 Components . . . 10

3.1.3 Events . . . 22

3.1.4 Listeners . . . 25

3.2 Functions . . . 28

3.2.1 Ui . . . 28

3.2.2 Dynamic . . . 29

3.3 Hints . . . 33

3.3.1 Ui . . . 33

4 Module reports **41**

4.1 Data Types . . . 41

4.1.1 Reports . . . 41

4.2 Functions . . . 41

4.2.1 Reports . . . 41

5 Module human **43**

5.1 Data Types . . . 43

5.1.1 Human . . . 43

5.1.2 Navigation . . . 45

5.2 Functions . . . 46

5.2.1 Todo . . . 46

5.2.2 Organization . . . 48

5.2.3 Documents . . . 52

5.2.4 Utilities . . . 53

5.3 Tasks . . . 54

5.3.1 Human . . . 54

6 Module dmn **55**

6.1 Data Types . . . 55

6.1.1 Dmn . . . 55

6.2 Functions . . . 58

6.2.1 Dmn . . . 58

7 Module forms **59**

7.1 Data Types . . . 59

7.1.1 Forms . . . 59

7.1.2 Chart . . . 133

7.1.3 Datasource . . . 146

7.2 Functions . . . 153

7.2.1 Forms . . . 153

7.2.2 Datasource . . . 154

8 Module core **157**

8.1 Constants . . . 157

8.1.1 Core . . . 157

8.2 Constraint Types . . . 158

8.2.1 Core . . . 158

8.3 Data Types . . . 159

8.3.1 Core . . . 159

8.4 Functions . . . 163

8.4.1 Proxy . . . 163

8.4.2 Collection . . . 165

8.4.3 Binary . . . 197

8.4.4 Enumeration . . . 199

8.4.5 Label . . . 200

8.4.6 Pdf . . . 200

8.4.7 Validation . . . 201

8.4.8 String . . . 202

8.4.9 Xml . . . 207

8.4.10 Reflection . . . 208

8.4.11 Type . . . 209

8.4.12 Number . . . 210

8.4.13 Model . . . 214

8.4.14 Date . . . 218

8.4.15 Restart . . . 234

8.4.16 Record . . . 235

8.4.17 Property . . . 236

8.4.18 Generic . . . 239

8.4.19 Json . . . 241

8.4.20 Ws . . . 244

8.4.21 Auditing . . . 245

8.4.22 Query . . . 246

8.4.23 Utilities . . . 247

8.5 Tasks . . . 249

8.5.1 Core . . . 249

Chapter 1

Standard Library

- Module uicommon
- Module ui
- Module reports
- Module human
- Module dmn
- Module forms
- Module core

Chapter 2

Module uicommon

The *uicommon* module defines a set of data types, functions, and UI presentation hints used by *ui* and *forms* modules.

- Data Types

2.1 Data Types

2.1.1 Uicommon

SelectItem A single option for components that present a list of selectable items (combobox, select list etc.).

value : Object Value itself.

label : String Label for the value.

GeographicCoordinates The geographic coordinate reference system used by the attributes is the World Geodetic System (2d) aka WGS84 aka EPSG:4326. The location on the globe is provided as a pair of two coordinates, latitude and longitude.

latitude : Decimal The latitude of the position, not null.

longitude : Decimal The longitude of the position, not null.

ResourceConverter A converter interface which provides a method to convert a value to a resource.

convert(value : Object) : Resource Converts a value to a resource.

Parameters:

- *value* the value to convert to a resource

StringToExternalResourceConverter A converter which converts a string value to an ExternalResource.

convert(value : Object) : ExternalResource Converts a String value to an external resource.

Parameters:

- *value* the value to convert to a resource

Returns:

- the external resource

StringToThemeResourceConverter A converter which converts a string value to a ThemeResource.

convert(value : Object) : ThemeResource Converts a String value to a theme resource.

Parameters:

- *value* the value to convert to a resource

Returns:

- the theme resource

DateEditor extends Editor Definition of a date editor for the grid cell date values.

formatPattern : String The date format pattern. For the format pattern syntax see http://docs.↩oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

resolution : DateTimeResolution The resolution of the date fields used by the editor. The finer the resolution the more precise date/time can be set through the editor.

EnumerationEditor extends Editor Definition of the enumeration editor for the enumeration values.

NumberEditor extends Editor Definition of a number editor for the grid cell values which represent numbers.

formatPattern : String The number format pattern. For the format pattern syntax see https://docs.↩oracle.com/javase/7/docs/api/java/text/DecimalFormat.html.

Editor[ABSTRACT] This abstract record type defines grid cell editor definition. Records extending this type provide definition of grid cell editors.

FileResource extends DownloadableResource Provides contents of given File.

file : { : File} When asked, the resource runs given closure in database transaction. The file contents are then served back to the client.

FileResource(file : { : File}) [CONSTRUCTOR]

FileResource(file : File) [CONSTRUCTOR]

fromModule(module∗ : String, path∗ : String) : FileResource Loads given file from the module.

Parameters:

- *module* the module name
- *path* the path, e.g. "images/image_15596.png"

FontAwesome extends Resource The Font Awesome resource, renders as a single Font Awesome characters. This type of resource is not downloadable (it has no downloadable byte stream content like PNG does, for example) and it may only be usable for #setIcon().

name : String The name of the icon in the Font Awesome font. Please see https://fortawesome.↩github.io/Font-Awesome/icons/ for a list of available icons. This string must be a lowercase string, separated with dashes. Either name or codepoint must be non-null.

codepoint : Integer The code of the icon in the Font Awesome font. Please see https↩://fortawesome.github.io/Font-Awesome/icons/ for a list of available icons. Either name or codepoint must be non-null.

fontFamily : String Some font awesome icons are found in both regular and solid icon sets. Use font family to specify which set should be used. Regular set is denoted by "far" and solid set by "fas".

FontAwesome(codepoint : Integer) [CONSTRUCTOR]

FontAwesome(name : String) [CONSTRUCTOR]

FontAwesome(fontFamily : String, codepoint : Integer) [CONSTRUCTOR]

FontAwesome(fontFamily : String, name : String) [CONSTRUCTOR]

ExternalResource extends DownloadableResource **url : String** A resource which may be found on another site. Use this resource to navigate in Links.

ExternalResource(url : String) [CONSTRUCTOR]

ThemeResource extends DownloadableResource Loads images and files from your currently selected Vaadin Theme.

resourceId : String The resource ID available in your current Vaadin theme. For example if the resourceId is "img/themeimage.png", the the full class-path name of the icon will be "VAADIN/themes/mytheme/img/themeimage.png"

ThemeResource(resourceId : String) CONSTRUCTOR

Resource ABSTRACT An abstract resource. The hierarchy is closed - please do not add your own custom resources, they will not work.

DownloadableResource ABSTRACT **extends Resource** Resources of this type are able to provide a file with a byte array content. Only resources of this type can be downloaded via the Link component, or displayed via the Image component.

DateTimeResolution Resolution for date fields.

Second

Minute

Hour

Day

Month

Year

DownloadStyle Link

Button

Orientation Horizontal

Vertical

Chapter 3

Module ui

The *ui* module defines a set of data types, functions, and UI presentation hints used to define application user interface.

The specified data types mostly represent UI components, UI-related events and event listeners. All of them are used to define forms in GO-BPMN models.

- Data Types
- Functions
- Hints

3.1 Data Types

3.1.1 Charts

PolarChart extends Chart Polar chart.

series : { : List<DataSeries>} A set of data series displayed by polar chart. It is recalculated on each refresh.

xAxes : List<ChartAxis> A list of x axes determining angles of data points.

yAxes : List<ChartAxis> A list of y axes determining the distance of data points from the chart center.

CartesianChart extends Chart Cartesian chart.

series : { : List<DataSeries>} A set of data series displayed by cartesian chart. It is recalculated on each refresh.

xAxes : List<ChartAxis> A list of x axes.

yAxes : List<ChartAxis> A list of y axes.

rotateAxes : Boolean If true, the x axes are displayed vertically and y axes are displayed horizontally. If false or unspecified, the x axes are displayed horizontally and y axes are displayed vertically.

PieChart extends Chart Pie chart.

slices : { : List<PieSlice>} A set of slices displayed by pie chart. It is recalculated on each refresh.

GaugeChart extends Chart Gauge chart.

value : { : Decimal} A value to be displayed by gauge needle. It is recalculated on each refresh.

valueName : String A value to be displayed as a tooltip when the mouse hovers over the gauge needle.

axis : GaugeAxis Gauge axis. If not specified, the gauge displays default axis.

PlotOptions[ABSTRACT] Options used to determine how the data are displayed in chart.

color : String Main color of the series (hex format, e.g. "#ff0000"). The default value is pulled from the VaadinTheme colors.

showLabels : Boolean Controls visibility of labels.

Chart[ABSTRACT] **extends UIComponent** An abstract UI component representing any type of chart.

title : { : String} The chart's main title. It is recalculated on each refresh.

subtitle : { : String} The chart's subtitle. It is recalculated on each refresh.

showLegend : Boolean If true or unspecified, the chart legend is displayed. If false the chart legend is hidden. The chart legend is a box containing a symbol and name for each data series.

configuration : ChartConfig General chart configuration, allows to configured background color.

ListDataSeries extends DataSeries A series consisting of a list of numerical values. Numerical values (Data↩ Point.value) will be interpreted as Y values, and X values will be automatically calculated

values : List<DataPoint> The list of values.

DataSeries[ABSTRACT] Abstract type for all specific data series types. Data series defines a set of data points that are displayed as values in the chart.

label : String Label of the data serie.

options : PlotOptions PlotOptions specifies how the data serie will be drawn (color, legend, linestyle, etc.).

xAxisIndex : Decimal 0-based index of X axis to use for plotting (CartesianChart.xAxes[?]).

yAxisIndex : Decimal 0-based index of Y axis to use for plotting (CartesianChart.yAxes[?]).

CategoryDataSeries extends DataSeries Values are defined as a map of Strings and DataPoints: the String is used as the value on the x axis and the data point defines the values on the y axis.

values : Map<String, DataPoint> The map of values.

TimedDataSeries extends DataSeries Values are defined as a map of Dates and DataPoints: the date is used as the value on the x axis and the data point defines the values on the y axis.

values : Map<Date, DataPoint> The map of values.

DecimalDataSeries extends DataSeries Values are defined as a map of Decimals and DataPoints: the decimal is used as the value on the x axis and the data point defines the values on the y axis.

values : Map<Decimal, DataPoint> The map of values.

DataPoint Data point for chart.

value : Decimal Usually x coordinate of data point.

value2 : Decimal Value2 is used, e.g., if PlotOptionsBubble is used to determine diameter of bubble.

payload : Object Business object which is sent in ChartClickEvent when user clicks bar\pie representing this DataPoint.

PieSlice Specification of the appearance of a pie chart slice.

label : String Label of the slice.

value : Decimal Determines portion of the slice.

color : String Color of the slice. If null, the default color is used.

payload : Object Business object which is sent in ChartClickEvent when user clicks pie representing this PieSlice.

GaugeAxis extends ChartAxis Record used to specifyan axis for gauge chart.

startAngle : Decimal Start angle of the polar X-axis or gauge axis, given in degrees where 0 is north. Defaults to 0.

endAngle : Decimal End angle of the polar X-axis or gauge value axis, given in degrees where 0 is north. Defaults to startAngle + 360.

centerY : Decimal Center of a polar chart or angular gauge. Positions is given as percentages of the plot area size. Defaults to ['50', '50'].

ChartAxis Record used to specify chart axes.

min : Object The minimum value of the axis.

max : Object The maximum value of the axis.

label : String Label of the axis.

opposite : Boolean Whether the axis is shown on the opposite side of the normal.

bands : List<PlotBand> Bands of the axis. A plot band is a colored band stretching across the plot area marking an interval on the axis.

PlotBand Record used to specify a band. A plot band is a colored band stretching across the plot area marking an interval on the axis.

from : Decimal Start of the band.

to : Decimal End of the band.

color : String Color of the band.

PlotOptionsBubble extends PlotOptions Data series will be displayed as bubbles.

PlotOptionsArea extends PlotOptionsLine The data series will be displayed as area. Used data points should specify also value2.

range : Boolean Render only difference between DataPoint.value and DataPoint.value2.

opacity : Decimal Fill opacity for the area. Defaults to .75.

PlotOptionsBar extends PlotOptions The data series will be displayed as bars.

range : Boolean Render only difference between DataPoint.value and DataPoint.value2.

stacked : Boolean Controls whether to stack the values of each series on top of each other.

PlotOptionsLine extends PlotOptionsScatter The data series will be displayed as line.

lineStyle : LineStyle The line style (LineStyle.solid, LineStyle.dot, ...).

lineWidth : Decimal With of the line in pixels. Defaults to 2.

spline : Boolean Controls whether to render as spline.

PlotOptionsScatter extends PlotOptions The data series will be displayed as a scatter chart - a series of unconnected data points.

stacked : Boolean Controls whether to stack the values of each series on top of each other.

marker : Marker The tipe of the marker (Marker.circle, Marker.square, ...).

ChartConfig chartBorderRadius : Decimal

backgroundColor : String

legendBackgroundColor : String

tooltipBackgroundColor : String

tooltipTextColor : String

tooltipBorderRadius : Decimal

tooltipTextShadow : String Sets the textShadow CSS attribute

tooltipShadow : Boolean

Marker Marker shape used for data points

circle

square

diamond

triangle

triangle-down

LineStyle Line style used for drawing line-based charts

solid

dot

dash

dashDot

dashDotDot

3.1.2 Components

UIComponent[ABSTRACT] **extends UIDefinition** Base type for UI components.

listeners : Set<Listener> Set of listeners registered on this component which listens for events emitted be this component.

hints : { : Map<String, Object>} Hints to be applied to this component. Recomputed on each refresh. Predefined hints are available in ui/ui.hint.

modelingId : String Unique identifier of the modeling element. Used in various log statements. It is possible to search by modeling id. To enable modelingId in generated html pages, use -Dcom.whitestein.lsps.↩ vaadin.ui.debug=true jvm property.

excludeValidationError : {ValidationError : Boolean} Allows to declaratively specify, which validation errors cannot be shown on this component.

includeValidationError : {ValidationError : Boolean} Allows to declaratively specify, which validation errors must be shown on this component.

contextMenuStatic : List<MenuItem> Lists menu items which should be displayed as a context menu when the component is mouse-right-clicked. May be null.

contextMenuDynamic : { : List<MenuItem>} Lists menu items which should be displayed as a context menu when the component is mouse-right-clicked. May be null. Re-computed on each right-click.

visible : { : Boolean} Controls the visibility of the component. The component is visible unless this closure is not null and returns false.

Container extends UIComponent Record type which is used to "wrap" reusable component and define its interface.

registrationPoints : Map<String, Set<Reference<Set<Listener>>>> Map, through which listener from "outside" can be registered on component within this reusable component.

publishedListeners : Map<String, Set<Listener>> Map holding listeners, which are defined withing this reusable component and can be registered on "outside" components.

child : UIComponent Child component, i.e. the reusable component itself.

methods : Map<String, {List<Object> : Object}> Map of "methods" which can be dynamically invoked via ui::invoke(...) method. Key of the maps is the name of the method.

OutputText extends UIComponent **content : { : Object}** The object, to be displayed as a text. See description of format property for more details. It is recalculated on each refresh.

If the closure is null, the value is calculated from the binding property.

format : String Depending on the return type of the content closure, the format value is defined as:

Date: pattern that conforms with java.text.SimpleDateFormat

Decimal: pattern that conforms with java.text.DecimalFormat

Integer: pattern that conforms with java.text.DecimalFormat

String: {html, plaintext, preformatted}

for other types {Boolean, Record, List, Set, Map, Reference, Closure, TypeNull, Object}, the format is ignored and core::toString() is used to transform the object to text

SECURITY NOTE: be caucious about XHTML format!

HTML format supports:

<b> Bold

<i> Italic

<u> Underlined

 Linebreak

item 1
item 2 List of items

plaintext shrinks spaces, etc.; preformatted keeps formating.

binding : Reference<Object> If used, the table will pick this field and allow automatic sorting and filtering. The referenced value is also displayed unless the content closure is specified (the content closure has higher display priority).

label : { : String} Label is the visible name of the component. Recalculated on each refresh.

ActionLink extends UIComponent **disabled : { : Boolean}** If true, the button is disabled, i.e. it is not possible to click it. It is recalculated on each refresh.

text : { : String} A text to be displayed as a link. It is recalculated with each refresh.

helpText : { : String} Text shown in the tooltip. Returned string might contain some tags, for details see com.vaadin.ui.AbstractComponent.getDescription(). It is recalculated on each refresh.

Button extends UIComponent **disabled : { : Boolean}** If true, the button is disabled, i.e. it is not possible to click it. It is recalculated on each refresh.

text : { : String} A text to be displayed within the button. It is recalculated with each refresh.

helpText : { : String} Text shown in the tooltip. Returned string might contain some tags, for details see com.vaadin.ui.AbstractComponent.getDescription(). It is recalculated on each refresh.

Message extends UIComponent Shows the error messages of all failed validators.

Conditional extends UIComponent **show^{DEPRECATED} : { : Boolean}** Determines, whether the child (content) is shown or not. It is mandatory. It is recalculated on each refresh.

child : UIComponent The child of the Conditional.

TableColumn extends UIComponent **header : { : String}** The columns header. It is recalculated on each refresh.

content : UIComponent The content of the TableColumn.

shown^{DEPRECATED} : { : Boolean} Whether the column is shown or not. It is recalculated on each refresh. If the column is not shown, it cannot be made visible through GUI selector.

Optional. If not specified, table column is shown. Deprecated.

ordering : Object The object which specifies by what to order. E.g., property Book.title.

inferOrderingDisabled : Boolean Controls, whether ordering inferring is disabled for this column.

filter : Filter Definition of the filter for this column.

inferFilteringDisabled : Boolean Controls, whether filter inferring is disabled for this column.

groupValue : {Collection<Null> : Object} This closure is used to populate corresponding column of the "aggregated" row (a.k.a. reduction function). Usual toString is applied on the return value of the closure. If groupValue closure is null, then the column is just not populated with any data. Note that it is not possible to "model" cells content for aggregated rows, the cell content is always a label with value of groupValue.

Table extends UIComponent **data : Object** Datasource of the table. Datasource can be specified by↩ : Type<Record>, {:Collection<Object>} or {Integer,Integer:Collection<Object>}. Recalculated on each refresh.

dataCount : { : Integer} Total count of all entries. Mandatory when table is paged and datasource provided as {Integer,Integer:Collection<Object>}. Recalculated on each refresh.

iterator : Reference<Object> Reference, to which object for "current" row is written, when the row is being calculated\refreshed.

idx : Reference<Integer> Reference, to which index of "current" row is written, when the row is being calculated\refreshed. Optional.

showIdx : { : Integer} Optional. Ignored when table type is 'simple'.

If provided, table shows page with the entry with the given index.

type : TableType TableType.simple - no paging

TableType.lazy - paging with scrollbar (no explicit pages)

TableType.paged - paging with explicit pages

This property is optional. If not provided, default TableType.simple is used.

columns : List<TableColumn> Specification of columns of the table.

ordering : Reference<Map<Object, OrderDirection>> Reference to which a "new" Table ordering is written when changed. Can be also used to programatically change the ordering from the model or set initial ordering of the Table. Example: assume Column1 is ordered by property 'Book.Title'. If user clicks this column, new Map will be written to ordering reference. The first map entry of that map will be ['Book.Title' -> OrderDirection, ...].

inferOrderingDisabled : Boolean Controls, whether ordering is automatically inferred. Optional. If true, sorting is automatically inferred.

orderingDisabled : Boolean Allows to completly disable ordering in the table "in one place". Optional. If true, table does not support ordering.

filtering : Reference<Collection<Filter>> Reference, to which is written "new" filtering, when user selects some filter in the table. It can be used to programatically control the table filtering from the model as well as define initial table filtering. Optional.

inferFilteringDisabled : Boolean Controls, whether filtering is automatically inferred. Optional. If false, filtering is automatically inferred.

filteringDisabled : Boolean Allows to completly disable filtering in the table "in one place". Optional. If true, table does not support filtering.

groupSpec : Collection<GroupSpec> Definition of "groupping options" by which it is possible to group data in table.

grouping : Reference<Collection<GroupSpec>> Reference, to which is written "new" Table grouping, when it is changed. Can be also used to programatically change the grouping from the model or set initial grouping of the Table.

inferGroupingDisabled : Boolean Controls, whether grouping is automatically inferred. Optional. If true, grouiping is automatically inferred.

groupingDisabled : Boolean Allows to completly disable grouping in the table "in one place". Optional. If true, table does not support grouping.

Panel extends UIComponent Panel

child : UIComponent Child component of the Panel.

title : { : String} The title of the panel. It is recalculated on each refresh. If left empty, no title is shown in the panel.

collapsed : Reference<Boolean> Specifies, whether the pannel is collapsed. If the panel is collapsed, only the panel title is shown. If user un/collapses the panel, the renderer writes current collapse state into this reference and it is possible to change the collapse state from the model. It is recalculated on each refresh.

TabbedLayout extends UIComponent **tabs : List<Tab>** Tabs of the Tab component.

GridLayout extends UIComponent **cells : Set<GridItem>** Individual cells of the Grid.

ViewModel extends UIComponent **child : UIComponent** The child component.

mergeType : { : MergeType} If the closure returns a non-null value and the ViewModel is merged (by some listener), the merge is performed. See MergeType for more details.

TextArea extends InputComponent **binding : Reference<String>** Reference to a slot containng the value.

placeholder : { : String} A text shown in the component if the binding is null. It is recalculated on each refresh.

isRichText : Boolean If true, component allows 'rich' formatting of the input.

CheckBox extends InputComponent **binding : Reference<Boolean>** Reference to a slot containng the value.

SingleSelectList extends InputComponent **binding : Reference<Object>** Reference to a slot containng the selected value.

options : { : List<Option>} List of value options.

CheckBoxList extends InputComponent **binding : Reference<Set<Object>>** Reference to a slot containing a list of selected values.

options : { : List<Option>} List of value options.

FileUpload extends InputComponent **binding : Reference<Set<File>>** Reference to a slot containng a set of uploaded files. It is not mandatory (newly uploaded data are received in event).

multiple : { : Boolean} If true, it is possible to upload mutiple files, otherwise, only one file can be uploaded.

uploadToMemory : Boolean If true, upload is done to memory, otherwise, upload is done to lsps binary data.

buttonText : { : String} Text displayed in the upload button. It is recalculated on each refresh.

deleteTempData : Boolean If FileUpload.uploadToMemory = false, then data are temporarily stored in L↩ SPS_BINARY_DATA table. FileUpload.deleteTempData is used to control whether the temporary data are automatically deleted from LSPS_BINARY_DATA by LSPS.

The deletion is done when associated http session is invalidated.

MultiSelectList extends InputComponent **binding : Reference<Set<Object>>** Reference to a slot containng a list of selected values.

options : { : List<Option>} List of value options.

TextBox extends InputComponent **binding : Reference<Object>** Reference to a slot containng the value.

format : String Base on binding type, format value should be:

Date: pattern that conforms with java.text.SimpleDateFormat

Decimal: pattern that conforms with java.text.DecimalFormat

Integer: pattern that conforms with java.text.DecimalFormat

String: format is ignored. However, it can contain pattern that conform with java.util.Pattern for client side validation

binding cannot reference any other type {Boolean, Record, List, Set, Map, Reference, Closure, TypeNull, Object}

This component supports client-side validation; i.e. if format is not null, JS on client will allow only to enter valid values according to format.

placeholder : { : String} A text shown in the component if the binding is null. It is recalculated on each refresh.

ComboBox extends InputComponent **binding : Reference<Object>** Reference to a slot containng the value.

options : { : List<Option>} List of value options.

createNewOption : {String : Object} If user enters new option into the combo, this closure is invoked with user entered input and the result is written to the binding.

placeholder : { : String} A text shown in the component if the binding is null. It is recalculated on each refresh.

RadioButtonList extends InputComponent **binding : Reference<Object>** Reference to a slot containng the selected value.

options : { : List<Option>} List of value options.

InputComponent^{ABSTRACT} extends UIComponent **readOnly : { : Boolean}** If true, the component is read only. Otherwise (false or unspecified), the component is read write. Recalculated on each refresh.

triggerProcessingOnChange : Boolean If true, each change of the component causes processing (data are sent to the server and response is displayed to the user). If false or unspecified, the changed data are sent to the server within the next post.

label : { : String} Label is the visible name of the component. Recalculated on each refresh.

required : { : Boolean} If true, visual notation is shown in the UI to indicate, that this component is mandatory. Note that the actual check for the provided value must be done within the listener. This is only a VISUAL INDICATOR. Recalculated on each refresh.

helpText : { : String} Text shown as tooltip. Recalculated on each refresh. If unspecified, no tooltip is displayed.

Repeater extends UIComponent **data : { : Collection<Object>}** Data through which repeater iterates. It is recalculated on each refresh.

iterator : Reference<Object> Reference, to which object for "current" iteration is written.

idx : Reference<Integer> Reference, to which index of "current" iteration is written. Optional.

content : UIComponent Component which defines how the entry will be rendered.

layout : RepeaterLayout The layout to use when laying out children. null defaults to "wrap". Not dynamically recomputed on refresh.

Tab **text : { : String}** The title of the tab. It is recalculated on each refresh.

content : UIComponent Content of the Tab.

shown^{DEPRECATED} : { : Boolean} Controls the visibility of the Tab. Deprecated. Use visible property instead.

visible : { : Boolean} Controls the visibility of the component. The component is visible unless this closure is not null and returns false.

GridItem **content : UIComponent** The content of the GridItem.

row : Integer 0-based. Position of the GridItem within the GridLayout.

column : Integer 0-based. Position of the GridItem within the GridLayout.

spanRows : Integer How many rows should it span.

spanColumns : Integer How many columns should it span.

FileDownload extends UIComponent Refresh is caused if the file is downloaded and there is registered listener which can handle FileDownloadEvent.

content : { : File} A file to be downloaded. It is recalculated on each refresh.

text : { : String} A text of the download link. It is recalculated on each refresh.

helpText : { : String} Tooltip of the download link. It is recalculated on each refresh.

style : FileDownloadStyle Specifies whether the file download component is shown as a hyperlink (default), or as a button.

Image extends UIComponent **content : { : File}** A file to be displayed. All usuall image formats are supported (jpeg, png, gif, ...). It is recalculated on each refresh.

text : { : String} The caption of the component (usually a text displayed above the image). It is recalculated on each refresh. Can be unspecified.

helpText : { : String} A tooltip of the image and altText. It is recalculated on each refresh. Can be unspecified.

NavigationLink extends UIComponent **disabled : { : Boolean}** If true, the button is disabled, i.e. it is not possible to click it. It is recalculated on each refresh.

content : { : Navigation} Returns subclass of Navigation type, which specifies, where to navigate when this link is clicked. If is recalculated on each refresh.

text : { : String} A text to be displayed as a link. It is recalculated with each refresh.

helpText : { : String} Text shown in the tooltip. Returned string might contain some tags, for details see com.vaadin.ui.AbstractComponent.getDescription(). It is recalculated on each refresh.

HorizontalLayout extends UIComponent **children : List<UIComponent>** Content of the Hirozontal Layout.

label : { : String} Label is the visible name of the component. Recalculated on each refresh.

VerticalLayout extends UIComponent **children : List<UIComponent>** Content of the Vertical Layout.

label : { : String} Label is the visible name of the component. Recalculated on each refresh.

Option extends SelectItem A single option of a ∗List UI components.

Popup extends UIComponent **show^{DEPRECATED} : { : Boolean}** Drives the popup visibility. Recalculated on each refresh. If null or false, popup is not shown. Deprecated. User visible property instead.

child : UIComponent The content of the Popup.

title : { : String} The title of the popup window. If unspecified, the title is left empty. It is recalculated on each refresh.

isModal : { : Boolean} If true, the popup is modal. Recalculated on each refresh.

LazyComboBox extends InputComponent **binding : Reference<Object>** Reference to a slot containng the value.

options : {String, Integer, Integer : List<Object>} Options for the given input by the user (first parameter). Second parameter of the closure is the start index and third parameter is the count of the options to be returned.

optionsCount : {String : Integer} Total count of all options for the given String (entered by user into the combo box).

formatter : {Null : String} Closure which returns label for the given object.

createNewOption : {String : Object} If user enters new option into the combo, this closure is invoked with user entered input and the result is written to the binding.

placeholder : { : String} A text shown in the component if the binding is null. It is recalculated on each refresh.

BrowserFrame extends UIComponent **url : { : String}** The URL to display withing the IFrame. Recalculated on each refresh.

Dashboard extends UIComponent **toolbar : UIComponent** Component which defines 'toolbar' of the Dashboard.

widgets : List<DashboardWidget> Invidivual widgets of the Dashboard.

DashboardWidget extends UIComponent **title : { : String}** The title of the widget. It is a mandatory property and is recalculated on each refresh.

content : UIComponent Content of the Widget.

widgetId : String Unique identifier of the widget. It is used to to identify a changed widget in the Widget↩ ChangeEvent.

required : Boolean If true, the widget is always visible in dashboard.

configuration : { : WidgetConfiguration} Configuration of widget specifying its visibility, size and position. If unspecified, the default values are used. It is recalculated on each refresh.

WidgetConfiguration **visible : Boolean** If true, Widget is visible.

width : Integer The width of Widget in px.

height : Integer The height of Widget in px.

top : Integer Top position of the widget in px (0,0 -> upper, left).

left : Integer Left position of the widget in px (0,0 -> upper, left).

zIndex : Integer Widget with higher zIndex is always in front of widget with lower zIndex.

maximized : Boolean If true, widget is maginized.

minimized : Boolean If false, widget is nimimized in the tray.

Dimension **width : Integer** Width in pixels.

height : Integer Height in pixels.

TreeItem TreeItem is essentially a holder of various properties which constitues one item (entry) of the Tree.

data : Object The business data reference. Must not be null.

label : String The label of the item, will be displayed in the tree.

expanded : Boolean If true, the item is initially expanded and shows its children. Defaults to false.

parent : TreeItem Parent TreeItem.

children : Collection<TreeItem> Children of this TreeItem. If null, children will be lazyly fetched when user expands 'this' TreeItem (i.e. Tree2.children closure will be invoked with 'this' TreeItem as parameter). Empty set or list means that 'this' TreeItem is leaf.

FormLayout extends UIComponent **children : List<UIComponent>** Content of the Form Layout.

GeographicCoordinate extends GeographicCoordinates The geographic coordinate reference system used by the attributes is the World Geodetic System (2d) aka WGS84 aka EPSG:4326. The location on the globe is provided as a pair of two coordinates, latitude and longitude.

Geoposition Contains details received from browsers HTML5 geolocation request. Note that on some devices selected fields may be null.

The geographic coordinate reference system used by the attributes is the World Geodetic System (2d) aka WGS84 aka EPSG:4326.

coordinate : GeographicCoordinate The geographic position, not null.

accuracy : Decimal The accuracy of the position information in meters. Not null.

altitude : Decimal The height of the position, specified in meters above the ellipsoid or null if device cannot provide the information.

altitudeAccuracy : Decimal The accuracy of the altitude informations in meters or null.

heading : Decimal Denotes the direction of travel of the hosting device and is specified in degrees, where 0° <= heading < 360°, counting clockwise relative to the true north. Null if device don't support it or it is not moving.

speed : Decimal The magnitude of the horizontal component of the hosting device's current velocity and is specified in meters per second. If the implementation cannot provide speed information, the value of this attribute must be null. Otherwise, the value of the speed attribute must be a non-negative real number.

PositionOptions A configuration for the device providing the location information.

enableHighAccuracy : Boolean The enableHighAccuracy attribute provides a hint that the application would like to receive the best possible results.

This may result in slower response times or increased power consumption. The user might also deny this capability,

or the device might not be able to provide more accurate results than if the flag wasn't specified.

The intended purpose of this attribute is to allow applications to inform the implementation that they

do not require high accuracy geolocation fixes and, therefore, the implementation can avoid using geolocation

providers that consume a significant amount of power (e.g. GPS). This is especially useful for applications running

on battery-powered devices, such as mobile phones.

If the PositionOptions parameter to getCurrentPosition or watchPosition is omitted, the default value used for the enableHighAccuracy attribute is false.

The same default value is used in ECMAScript when the enableHighAccuracy property is omitted.

timeout : Decimal The timeout attribute denotes the maximum length of time (expressed in milliseconds) that is allowed to pass from the call

to getCurrentPosition() or watchPosition() until the corresponding successCallback is invoked. If the implementation is unable to

successfully acquire a new Position before the given timeout elapses, and no other errors have occurred in this interval,

then the corresponding errorCallback must be invoked with a PositionError object whose code attribute is set to TIMEOUT.

Note that the time that is spent obtaining the user permission is not included in the period covered by the timeout attribute.

The timeout attribute only applies to the location acquisition operation.

If the PositionOptions parameter to getCurrentPosition or watchPosition is omitted,

the default value used for the timeout attribute is Infinity. If a negative value is supplied, the timeout value is considered to be 0.

The same default value is used in ECMAScript when the timeout property is omitted.

maximumAge : Decimal The maximumAge attribute indicates that the application is willing to accept a cached position whose age is no greater than the specified time in milliseconds.

If maximumAge is set to 0, the implementation must immediately attempt to acquire a new position object. Setting the maximumAge to Infinity must determine

the implementation to return a cached position regardless of its age. If an implementation does not have a cached

position available whose age is no greater than the specified maximumAge, then it must acquire a new position object.

In case of a watchPosition(), the maximumAge refers to the first position object returned by the implementation.

If the PositionOptions parameter to getCurrentPosition or watchPosition is omitted, the default value used for

the maximumAge attribute is 0. If a negative value is supplied, the maximumAge value is considered to be 0.

The same default value is used in ECMAScript when the maximumAge property is omitted.

Geolocator extends UIComponent The Geolocator extension can be used to detect the client's geographical location, direction, altitude, etc.

detect : { : Boolean} Detects the current geographic location of the client. Set to true and refresh this component to perform the detection. The detection happens asynchronously and the position is reported to GeolocationListener as a GeolocationEvent. Note that this only checks the position once, you need to call this method multiple times if you want to update the location as the client moves. User may reject the position detection, in which case the event is fired, with GeolocationEvent.failure set to GeolocationError.PermissionDenied.

positionOptions : { : PositionOptions} A closure which retrieves position gathering options. The closure may return null.

MapMarker title : String The marker title.

location : GeographicCoordinate The marker location.

popup : String If not null, this HTML code is displayed when the marker is clicked.

draggable : Boolean If true, this marker can be dragged to a different location. When dragged, the MarkerDraggedEvent event is fired.

MapDisplay extends UIComponent Displays an OpenStreetMap map.

center : { : GeographicCoordinate} Upon refreshing, asks for the new coordinates to center on. If null is received, no centering is performed.

zoom : { : Integer} Upon refreshing, asks for the zoom. If null is received, no zoom change is performed.

markers : { : Set<Object>} Upon refreshing, asks for a list of markers to show. Old markers are removed from the map. If null is received, no markers are shown.

toMarker : {Null : MapMarker} Converts business data to marker.

Calendar extends UIComponent **data : {Date, Date : Set<Object>}** A closure which returns the data displayed in calendar. The result depends on start and end dates of the displayed period, both specified as closure parameters.

toItem : {Null : CalendarItem} A closure that transforms an underlying business object to a calendar item.

initialDate : Date A date on which the calendar is initially opened. If not specified, the calendar is opened at the current date.

readOnly : { : Boolean} If true, Calendar is read only. Otherwise (false or unspecified), Calendar is read write. Recalculated on each refresh.

mode : { : CalendarMode} Calculated on refresh. If the closure is null or if it returns a non-null value, the calendar component is switched to given mode.

CalendarItem **caption : String** Caption of the calendar item.

description : String Description of the calendar item.

from : Date Start date of the calendar item.

to : Date Finish date of the calendar item.

allDay : Boolean If true, the calendar item represents the all day event, ie., time of the from and to fields is ignored. If false, the calendar item takes into account also the time of the from and to fields.

style : String The style name of event. In the HTML-based client, the style name will be set as the event's element class name and can be styled by CSS. For example, setting this value to "color1" will attach "v-calendar-event-color1" CSS class name to the HTML element and can be further styled using CSS.

MenuItem **caption : String** The caption of the menu item.

id : Object This value will be present in MenuEvent when the menu item is clicked.

submenu : List<MenuItem> May optionally contain child menu items. MenuEvent is fired for "leaf" menu items only - it is not fired for menu items which contains submenu items.

htmlClass : String String which will be used as html class in generated html.

Tree2 extends InputComponent **root : { : Collection<TreeItem>}** Root elements of the Tree (yes, tree can have multiple root elements).

children : {TreeItem : Collection<TreeItem>} Closure, which returns for given TreeItem its children. If the given TreeItem has no children (i.e. it is a leaf), the closure should return empty set or list.

binding : Reference<Object> Reference to which the data object of TreeItem is written when it is selected.

TreeTable2 extends UIComponent **root : { : Collection<TreeItem>}** Root elements of the TreeTable.

children : {TreeItem : Collection<TreeItem>} Closure, which returns for given TreeItem its children. If the given TreeItem has no children (i.e. it is a leaf), the closure should return empty set or list.

iterator : Reference<Object> Reference, to which object for "current" row is written.

treeItemIterator : Reference<TreeItem> Reference, to which TreeItem for "current" row is written.

columns : List<TableColumn> Definition of columns of the TreeTable.

ordering : Reference<Map<Object, OrderDirection>> Reference to which a "new" TreeTable ordering is written when it is changed. Can be also used to programatically change the ordering from the model or set initial ordering. Example: assume Column1 is ordered by property 'Book.Title'. If user clicks this column, new Map will be written to ordering reference. The first map entry of that map will be ['Book.Title' -> OrderDirection, ...].

inferFilteringDisabled : Boolean Controls, whether filtering is automatically inferred. Optional. If false, filtering is automatically inferred.

filteringDisabled : Boolean Allows to completly disable filtering in the tree table "in one place". Optional. If true, tree table does not support filtering.

CustomFilter extends Filter Used when one wants to use his own FilterUI specified through CustomFilter.ui. The CustomFilter.filterText is displayed in the table header when the filter is active.

ui : UIComponent UI component used to display the custom filter.

filterText : { : String} Displayed text of the filter.

popup : Boolean Whether to display CustomFilter.ui in the popup.

SubstringFilterUI extends FilterUI Used to filter String. The usual string% filtering.

substring : String The substring to filter by.

Filter$^{\text{ABSTRACT}}$ An abstract type used to define filter of table column.

FilterUI$^{\text{ABSTRACT}}$ Abstract supertype for all concrete definitions of the filter UI.

PropertyFilter extends Filter Allows to define filter by property. Only meaningful Property can be entered (i.e. property of simple comparable type). Filtering form will be automatically inferred from the type of the property (if it is not explicitly specified via PropertyFilter.ui). E.g. if getPropertyType(propertyFilter.p) == Integer then NumericFilterUI is used.

p : Property The property to filter by.

OptionsFilterUI extends FilterUI A simple combobox which allows to filter by predefined set of options.

options : List<Option> Options to allow to filter by.

selected : List<Option> Allows to set the pre-selected options to filter by when filter is set from model.

multiselect : Boolean Whether to allow to filter by multiple selected options.

ClosureFilter extends Filter Allows to filter by return value of the closure. Input parameter is a "business object", return value is value to filter upon (some simple comparable type).

c : {Null : Object} The closure which computes value to filter upon.

NumericFilterUI extends FilterUI Used to filter Integers, Decimals. Allows to filter by >, <, ==.

lessThan : Decimal Less than value.

equal : Decimal Equal value.

moreThan : Decimal More than value.

DateFilterUI extends FilterUI Used to filter dates.

lessThan : Date Less than date.

moreThan : Date More than date.

resolution : Resolution

RegExpFilterUI extends FilterUI Used to filter string values by regexp.

regexp : String The regexp (java.util.regex.Pattern).

ClosureGroupSpec extends GroupSpec This record type allows to define group by "computed" value.

groupBy : {Null : Object} The closure which defines the grouping. Return value should be only of comparable types, i.e. Boolean, Date, Decimal, Integer, Enum, String.

GroupSpec[ABSTRACT] An abstract type used to define grouping of table data.

label : String Label of the group user will see in UI.

PropertyGroupSpec extends GroupSpec This record type is used to define group by property.

groupBy : Property The property which defines the grouping. e.g. Book.Title.

OptClosureGroupSpec extends GroupSpec This is the same as ClosureGroupSpec but allows for "better performance".

groupBy : {Collection<Null> : Map<Object, Collection<Object>>} The input is set of objects to be grouped in one pass.

GridColumn extends UIComponent header : { : String}

content : Object May be either a PropertyPath, or a closure {Null:Object}

sortable : Boolean By default, Property content is sortable, while the Closure content is not. This allows overriding the auto-detection mechanism.

filtrable : Boolean

renderer : GridCellRenderer

editable : Boolean

editor : Editor

Grid extends UIComponent data : Object

dataCount : { : Integer}

columns : List<GridColumn>

frozenColumnCount : Integer Number of leftmost columns which are frozen, that is, not scrolled out when scrolling vertically.

editorEnabled : Boolean

editorBuffered : Boolean

GridCellRenderer[ABSTRACT] **extends UIComponent**

ButtonRenderer extends GridCellRenderer Renders the textual content of a cell as a button. This renderer produces "ui::RendererClickEvent" on the button click.

LinkRenderer extends GridCellRenderer Renders the textual content of a cell as a link. This renderer produces "ui::RendererClickEvent" on the link click.

DateRenderer extends GridCellRenderer formatPattern : String The date format pattern. For the format pattern syntax see http://docs.oracle.com/javase/7/docs/api/java/text/↩SimpleDateFormat.html

NumberRenderer extends GridCellRenderer formatPattern : String The number format pattern. For the format pattern syntax see https://docs.oracle.com/javase/7/docs/api/java/text/↩DecimalFormat.html.

ImageRenderer extends GridCellRenderer Renders a cell as an image. The content of the cell is expected to be a string. This renderer produces "ui::RendererClickEvent" on the image click.

converter : ResourceConverter The converter which converts the cell value to the image resource. See the specific type of the converter to learn the expected cell value type.

HtmlRenderer extends GridCellRenderer

EnumerationRenderer extends GridCellRenderer

TokenField extends CheckBoxList

TableColumnState columnId : Object

collapsed : Boolean

width : Integer

GeolocatorError Enumerates all possible causes of a geolocation retrieval failure.

UnknownError An unknown error occurred.

PermissionDenied The user declined access to their position.

PositionUnavailable The browser was unable to locate the user.

Timeout The browser was unable to locate the user in the time specified in the PositionOptions.timeout

UnsupportedInBrowser The browser does not support geolocation retrieval.

MergeType Specifies the merge algorithm used by a particular ViewModel when a ViewModel merge is requested in a Listener.

oneLevel Merges one level down towards the screen context (corresponds to the deprecated setting mergeToTopLevel = false)

screenLevel Evaluation context and any lower execution contexts are merged to the screen context.

TableType Defines the table type and the way the table accesses the data.

simple A simple table, which reads all data rows from the underlying data source and holds them in-memory. By default the table wraps and shows all rows. When the height is set to fill-parent, the table is able to scroll its contents.

lazy A lazy table with a scrollbar

Upon scrolling, the table polls the data source for data. The data is internally retrieved in pages or batches of 30 rows. You can use the initial-page-size hint to modify the batch size. As a rule of thumb, the page size should be twice as big as the number of rows shown in the table.

paged A paged table with no scrollbar

The page shows 20 items at most by default; this can be changed by the initial-page-size hint value. Paging controls are displayed below the table, which allows the user to move to the next/last/previous/first page.

OrderDirection The ordering direction of the Table Column; similar to SQL ORDER BY.

Ascending The data is sorted in ascending order.

Descending The data is sorted in descending order.

RepeaterLayout Defines the layout of child UI components in the repeater.

wrap The default layout behavior

Children are positioned horizontally until there is no more space - in such a case a next row is started. To activate this mode, you need to set the child's width to wrap-content; If the child has the width of fill_parent, the children are laid out vertically.

horizontal Lays out children horizontally. Equal to HorizontalLayout.

vertical Lays out children vertically. Equal to VerticalLayout.

FileDownloadStyle The FileDownload component style

Link The FileDownload component is rendered as a hyperlink (default).

Button The FileDownload component is rendered as a button.

CalendarMode The display mode of the Calendar component

Daily The calendar shows a single day, in a single column, with hours displayed as rows.

Weekly The calendar shows seven days in seven columns, with hours displayed as rows.

Monthly The calendar shows all days of a particular month, as a grid of tiles.

NotificationType Predefined notification types

Each notification type has its distinctive UI look, default screen position and behavior (for example, an Info notification fades away automatically while an Error notification is displayed until the user clicks the notification).

Info Information notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaadi for examples.

By default, notification with this type disappears immediately (that is, it slowly fades in and immediately, with no delay, starts fading out) . The nofication is shown in the middle center part of the screen by default.

Warning Warning notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaad for examples.

By default, notification with this type disappears after 1,5 second and is shown in the middle center part of the screen.

Error Error notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaadin for examples.

By default, notification with this type requires user click to disappear and is shown in the middle center part of the screen.

Tray Bottom-right small notification. See https://vaadin.com/blog/-/blogs/user-notifications-with- for examples.

By default, this notification disappears after 3 seconds and is shown in the bottom-right corner.

Position The notification position in the browser tab

TopLeft

TopCenter

TopRight

MiddleLeft

MiddleCenter

MiddleRight

BottomLeft

BottomCenter

BottomRight

Resolution **Second**

Minute

Hour

Day

Month

Year

3.1.3 Events

ValueChangeEvent[SYSTEM] **extends Event** Event fired by input components when their value is changed.

source : UIComponent Component that produced the event.

oldValue : Object Value before it was changed.

newValue : Object Value after it was changed.

ActionEvent[SYSTEM] **extends Event** Event fired by e.g. button, link when it is clicked.

source : UIComponent Component that produced the event.

FileDownloadEvent[SYSTEM] **extends Event** Event fired when file is downloaded through FileDownload component.

source : UIComponent Component that produced the event.

InitEvent[SYSTEM] **extends Event** Event fired by "all" components when they change their visibility.

source : UIComponent Component that produced the event.

isFirstLoad : Boolean True for a component displayed for the first time (the property is true also when a hidden component is displayed for the first time).

isFirstLoadAfterSave : Boolean True for a component displayed for the first time or for the first time after save

FileUploadEvent[SYSTEM] **extends Event** Event fired from FileUpload component when file is uploaded.

source : UIComponent Component that produced the event.

uploadedFiles : Set<File> Set of uploaded files.

errorMessage : String Error message returned if the upload fails.

ApplicationEvent extends Event ApplicationEvent.

eventName : String Custom name of the ApplicationEvent.

payload : Object Custom event data.

Event[ABSTRACTSYSTEM] Abstract supertype of all Events.

ChartClickEvent[SYSTEM] **extends Event** Event fired when bar\pie\... is clicked in chart.

source : UIComponent Component that produced the event.

series : String Label of data series that was clicked.

key : Object Key value for the data point.

value : Decimal First value defining the data point.

value2 : Decimal Second value defining the data point.

payload : Object Payload of the data point.

WidgetChangeEvent[SYSTEM] **extends Event** Event fired when widget of dashboard is moved, resized, etc.

source : UIComponent Component that produced the event.

widgetId : String ID of the widget that produced the event set in the Widget ID parameter.

configuration : WidgetConfiguration Widget configuration with details about the widget position and size.

CalendarCreateEvent[SYSTEM] **extends CalendarItemEvent** The CalendarCreateEvent is fired by a calendar component when the user clicks and drags over a period in a calendar. The event holds the selection data as its payload and the data can be used to create a new calendar entry.

from : Date Start date of the selected period.

to : Date End date of the selected period.

allDay : Boolean If the entry is a whole-day event (if selected across days, the entry is an allDay entry; if the selected area is across hours, the allDay property is false and the exact hours are included).

CalendarEditEvent[SYSTEM] **extends CalendarItemEvent** The CalendarEditEvent is fired by a calendar component when a calendar entry is clicked. Note that the event has as its payload the business object of the calendar entry that was clicked.

data : Object Business object of the calendar entry that was clicked.

CalendarItemEvent[ABSTRACT SYSTEM] **extends Event** **source : Calendar**

CalendarRescheduleEvent[SYSTEM] **extends CalendarItemEvent** The CalendarRescheduleEvent is fired by the calendar component when a calendar entry is dragged-and-dropped to a different date. Note that the event has as its payload the business object of the rescheduled calendar entry.

from : Date Start date of the new period.

to : Date End date of the new period.

data : Object Business object of the calendar entry that was rescheduled.

MapClickedEvent[SYSTEM] **extends Event** The MapClickedEvent is fired by the Map Display component when the user clicks into the map.

source : MapDisplay Component that produced the event.

point : GeographicCoordinate Point that was clicked.

MarkerClickedEvent[SYSTEM] **extends Event** The MarkerClickedEvent is fired by the Map Display component when the user clicks a marker.

source : MapDisplay Component that produced the event.

markerData : Object Underlying marker business object (the respective object of the set defined in the Markers property of the Map Display component).

MarkerDraggedEvent[SYSTEM] **extends Event** The MarkerDraggedEvent is fired by the Map Display component when the user drag-and-drops a marker.

source : MapDisplay Component that produced the event.

markerData : Object Underlying marker business object (the respective object of the set defined in the Markers property of the Map Display component).

newLocation : GeographicCoordinate New marker position after dropped.

GeolocationEvent[SYSTEM] **extends Event** The GeolocationEvent is fired by the Geolocator component after the component has acquired the geographical position of the user or when the request for location times out.

source : Geolocator Component that produced the event.

position : Geoposition When a geolocation request succeeds, this value will contain a non-null position.

failure : GeolocatorError When a geolocation request fails, this field will contain the error cause.

MenuEvent[SYSTEM] **extends Event** Fired by menu item when it is clicked.

source : UIComponent The component which has the menu attached.

id : Object The value of MenuItem.id.

ItemExpandedEvent[SYSTEM DEPRECATED] **extends Event** DEPRECATED.

source : UIComponent The component which has the menu attached

data : Object The value of MenuItem.id

ItemCollapsedEvent[SYSTEM DEPRECATED] **extends Event** DEPRECATED.

source : UIComponent The component which has the menu attached

data : Object The value of MenuItem.id

TreeEvent[SYSTEM] **extends Event** Fired by Tree component when some TreeItem is selected.

source : UIComponent Component that produced the event.

treeItem : TreeItem Event-related TreeItem.

PopupCloseRequestEvent extends Event This event is only fired when the user clicks the X (close) popup button. It is not fired when the popup is closed because its visiblity has been set to false.

source : Popup Component that produced the event.

TablePageSizeChangeEvent extends Event **source : Table** The component which has the menu attached.

pageSize : Integer The value of MenuItem.id.

RendererClickEvent extends Event An event indicating that a click occurred on the grid cell renderer.

source : GridCellRenderer

rowObject : Object the row object of the row in which the click occurred

AsynchronousTextChangeEvent^{SYSTEM} extends Event Event fired by input text components when the text inside them is changed. The event is fired asynchronously, a new one can be fired even if the old one is still being processed.

source : UIComponent Component that produced the event.

text : String Current content of the text field.

3.1.4 Listeners

Listener^{ABSTRACT} Abstract supertype of all listeners.

refresh : {Event : Set<UIComponent>} Set of components to refresh.

process : Set<UIComponent> Set of components to process in request-responce lifecycle. Events from other components will be ignored.

validate : {Event : Set<ValidationError>} Closure, which returns validation errors.

executeOnlyIfVisible : Set<UIComponent> Listener will not execute if specified components are not visible.

executeEvenIfFailedValidations : Boolean If true, listener is executed even in the case some validations failed in 'Validate II' phase (however, all validations on this listener must pass). Otherwise (null, false), should the listener be executed in 'Handle ActionEvents' phase, it is only executed if all validations from all listeners passed in the 'Validate II' phase (i.e. outcome of 'Validation II' phase is success). Default is null/false.

executionContext : UIComponent Execution context of the listener. If not specified, listener is executed in the context of component on which it is registered.

modelingId : String Modeling id which is used in exceptions, etc. to identify the listener.

actions : {Event : List<Action>} Set of actions to be executed.

executeEvenIfInvalidComponents : Boolean If true, listener is executed even in the case some components are invalid (has unparsable values, such as 'a' for integer/date field). Default is null/false.

eventFilter : {Event : Boolean} Listener gets executed only if event passes eventFilter (if specified).

ValueChangeListener extends Listener Listener which listens for ValueChangeEvents.

handle : {ValueChangeEvent : void} Closure that is executed when listener is fired.

InitListener extends Listener Listener which listens for InitEvent.

handle : {InitEvent : void} Closure that is executed when listener is fired.

GenericListener extends Listener Listener which listens for any type of Event.

handle : {Event : void} Closure that is executed when listener is fired.

ActionListener extends Listener Listener which listens for ActionEvents.

handle : {ActionEvent : void} Closure that is executed when listener is fired.

FileDownloadListener extends Listener Listener which listens for FileDownloadEvent.

handle : {FileDownloadEvent : void} Closure that is executed when listener is fired.

FileUploadListener extends Listener Listener which listens for FileUploadEvents.

handle : {FileUploadEvent : void} Closure that is executed when listener is fired.

ApplicationEventListener extends Listener Listener which listens for ApplicationEvents.

handle : {ApplicationEvent : void} Closure that is executed when listener is fired.
eventName : Collection<String> Name of the ApplicationEvent for which this listener listens.

ChartClickListener extends Listener Listener which listens for ChartClickEvents.

handle : {ChartClickEvent : void} Closure that is executed when listener is fired.

WidgetChangeListener extends Listener Listener which listens for WidgetChangeEvents.

handle : {WidgetChangeEvent : void} Closure that is executed when listener is fired.

CalendarRescheduleListener extends Listener Listener which listens for CalendarRescheduleEvents.

handle : {CalendarRescheduleEvent : void} Closure that is executed when listener is fired.

CalendarCreateListener extends Listener Listener which listens for CalendarCreateEvents.

handle : {CalendarCreateEvent : void} Closure that is executed when listener is fired.

CalendarEditListener extends Listener Listener which listens for CalendarEditEvents.

handle : {CalendarEditEvent : void} Closure that is executed when listener is fired.

MapClickedListener extends Listener Listener which listens for MapClickedEvents.

handle : {MapClickedEvent : void} Closure that is executed when listener is fired.

MarkerClickedListener extends Listener Listener which listens for MarkerClickedEvents.

handle : {MarkerClickedEvent : void} Closure that is executed when listener is fired.

MarkerDraggedListener extends Listener Listener which listens for MarkerDraggedEvents.

handle : {MarkerDraggedEvent : void} Closure that is executed when listener is fired.

GeolocationListener extends Listener Listener which listens for GeolocationEvents.

handle : {GeolocationEvent : void} Closure that is executed when listener is fired.

FireApplicationEventAction extends Action **event : { : ApplicationEvent}**

NavigationAction extends Action Action which allows to navigate to other target (e.g. todo, document, ...).

navigation : { : Navigation} Closure which returns navigation target.

Action[ABSTRACT] Abstract supertype for all Actions.

PersistAction extends Action Persist action causes the data to be stored to database.

persistAction : { : void} Closure, which is executed after the persist happened.

ViewModelAction extends Action Action performed on view mode.

clearViewModel : { : Set<UIComponent>} Specified view models will be cleared.

mergeViewModel : { : Set<UIComponent>} Specified view models will be merged.

viewModelInit : { : void} Closure, which is usually used to initialize view model after it was cleared.

SubmitAction extends Action Action which causes submit of document or todo. With submit also persist is performed.

SaveAction extends Action Save action which results in save of current todo or document.

saveAction : {Todo, SavedDocument : void} Closure, which receives saved document or todo object as parameter and is usually used to associate saved document\todo with business data.

ValidationError[ABSTRACT] Abstract supertype of all validation error types.

shownOn : Set<UIComponent> Components, on which the validation error is shown (i.e. it might be automatically placed on some component based on binding).

DataValidationError extends ValidationError Validation errors produced by contraints.

constraintViolation : ConstraintViolation

UIValidationError extends ValidationError Validation errors produced by UI validators.

message : String Error message.

placement : Set<UIComponent> Components, on which validation error should be displayed.

MenuListener extends Listener Listener which listens for MenuEvents.

handle : {MenuEvent : void} Closure that is executed when listener is fired.

ItemExpandedListener[DEPRECATED] **extends Listener handle : {ItemExpandedEvent : Object}**

ItemCollapsedListener[DEPRECATED] **extends Listener handle : {ItemCollapsedEvent : Object}**

TreeListener extends Listener Listener which listens for TreeEvents.

handle : {TreeEvent : void} Closure that is executed when listener is fired.

PopupCloseRequestListener extends Listener Listener which listens for PopupCloseRequestEvents.

handle : {PopupCloseRequestEvent : void} Closure that is executed when listener is fired.

TablePageSizeChangeListener extends Listener handle : {TablePageSizeChangeEvent : void} Closure that is executed when listener is fired.

RendererClickListener extends Listener A listener for handling click events originating in the grid cell renderers.

handle : {RendererClickEvent : void}

AsynchronousTextChangeListener extends Listener Listener which listens for AsynchronousTextChange↩ Events. The events are fired asynchronously, a new one can be fired even if the old one is still being processed.

handle : {AsynchronousTextChangeEvent : Object} Closure that is executed when listener is fired.

3.2 Functions

3.2.1 Ui

addToPublishedListeners(publishedListeners : Map<String, Set<Listener>>, key : String, listeners : Listener...) : Map<St
Utility function which adds listener specified in third parameter to map of publishedListeners under the key key. Deprecated.

Parameters:

- *publishedListeners*
- *key*
- *listeners*

addToRegistrationPoints(registrationPoints : Map<String, Set<Reference<Set<Listener>>>>, key : String, components
Utility function which adds component specified in third parameter to map of registrationPoints under the key key. Deprecated.

Parameters:

- *registrationPoints*
- *key*
- *components*

createValidationError(message : String) : UIValidationError If the given message is not null, this function returns a UIValidationError with a given message. Otherwise it returns null.

Parameters:

- *message*

createValidationError(message : String, placement : UIComponent) : UIValidationError If the given message is not null, this function returns a UIValidationError with a given message and placement. Otherwise it returns null.

Parameters:

- *message*
- *placement*

createValidationError(message : String, placement : Set<UIComponent>) : UIValidationError If the given message is not null, this function returns a UIValidationError with a given message and placement. Otherwise it returns null.

Parameters:

- *message*
- *placement*

getBrowserWindowSize() : Dimension Retrieves the size, in DIPs, of the current browser window.

notify(caption∗ : String, description : String, type : NotificationType, position : Position, delayMillis : Integer, cssStyle : Stri
Shows a simple notification to the user. Must be called from UI listener.

Parameters:

- *caption* The main notification body, required
- *description* Additional notification text, may be null
- *type* Defaults to Info
- *position* Default value depends on the 'type' parameter.

- *delayMillis* if 0 or greater, the notification auto-closes after specified period of milliseconds after any user activity; if -1, the notification never disappears and must be clicked by the user. The default value depends on the 'type' parameter
- *cssStyle*
- *htmlContentAllowed* Defaults to false. If false, all html content in caption/description is escaped.

rgb(red∗ : Integer, green∗ : Integer, blue∗ : Integer) : String Returns a string used to represent a color given by red, green and blue components. Each color component is an integer from interval 0 to 255. The result has form of usual hexadecimal color representation, e.g., "#ff0000" for red.

Parameters:

- *red*
- *green*
- *blue*

Throws:

- *NullParameterError* if mandatory parameter is null.

3.2.2 Dynamic

addColumn(what∗ : TableColumn, where∗ : UIComponent) : TableColumn[SIDE EFFECT][DEPRECATED] Requies Vaadin 7. Creates a Vaadin instance for given table column definition and adds it to given table. Does nothing if there already is table column present for given definition.

Parameters:

- *what* The table column to add
- *where* Table or TreeTable2

Throws:

- *NullParameterError* if a mandatory parameter is null

addTab(tabbedLayout∗ : TabbedLayout, tab∗ : Tab) : void[SIDE EFFECT] Dynamically adds a tab to given tabbed layout. If the tab is already present in the tabbed layout, this function does nothing.

Parameters:

- *tabbedLayout*
- *tab*

Throws:

- *NullParameterError* if a mandatory parameter is null

clear(viewModels∗ : Collection<ViewModel>) : void[SIDE EFFECT] Clears given view models.

Parameters:

- *viewModels*

Throws:

- *NullParameterError* if a mandatory parameter is null

createAndAdd(what∗ : T, where∗ : UIComponent) : T[SIDE EFFECT] Creates a Vaadin instance for given component definition and adds it to given layout.

Parameters:

- *what* What component to create dynamically. Do not use with popups and table columns.

- *where* Where to add the component. Only horizontal layout, vertical layout and form layout are supported

Throws:

- *NullParameterError* if a mandatory parameter is null

createAndShow(def∗ : Popup) : Popup[SIDE EFFECT] Creates a Vaadin instance of popup, bound to given definition record, and shows it. Does nothing if there already is popup bound to this instance of definition record.

Parameters:

- *def* The definition record of popup that is to be shown.

Throws:

- *NullParameterError* if a mandatory parameter is null

findTopmostComponents(type∗ : Type<T>, root∗ : UIComponent) : List<T> Returns top most components.

Parameters:

- *type* The type of the components to find
- *root* Start search from this component

Throws:

- *NullParameterError* if a mandatory parameter is null

findTopmostContainers(root∗ : UIComponent) : List<Container> Returns top most containers.

Parameters:

- *root* Start search from this component

Throws:

- *NullParameterError* if a mandatory parameter is null

getChildren(layout∗ : UIComponent) : List<UIComponent> Returns the current list of children of a given layout component. This may differ to the 'children' property if the child list has been altered dynamically.

Parameters:

- *layout* VerticalLayout, HorizontalLayout or FormLayout only.

getColumnStates(table∗ : UIComponent) : List<TableColumnState>[DEPRECATED] Requies Vaadin 7. Returns the state of columns of a Table or TreeTable2. The state can be restored to the table by calling 'restore↩ ColumnStates' function.

Parameters:

- *table* Table or TreeTable2

Throws:

- *NullParameterError* if a mandatory parameter is null

getColumns(table∗ : UIComponent) : List<TableColumn>[DEPRECATED] Requies Vaadin 7. Returns the current list of table columns of a given Table or TreeTable2. This may differ to the 'columns' property if the column list has been altered dynamically.

Parameters:

- *table* Table or TreeTable2

getTabs(tabbedLayout∗ : TabbedLayout) : List<Tab> Returns the current list of tabs of a given TabbedLayout. This may differ to the 'tabs' property if the tab list has been altered dynamically.

Parameters:

- *tabbedLayout*

hideAndDestroy(def∗ : Popup) : void[SIDE EFFECT] Hides Vaadin popup, bound to given definition record, and destroys it. Does nothing if there is no popup registered to given record.

Parameters:

- *def* The definition record of popup which is to be destroyed.

Throws:

- *NullParameterError* if a mandatory parameter is null

invoke(targets∗ : Collection<Container>, methodName∗ : String, parameters∗ : List<Object>) : void[SIDE EFFECT] Executes "Container methods".

Parameters:

- *targets* Invoke the method on these containers
- *methodName* The method name
- *parameters* Method parameters

Throws:

- *NullParameterError* if a mandatory parameter is null

merge(viewModels∗ : Collection<ViewModel>) : void[SIDE EFFECT] Merges given view models to upper levels (one level up).

Parameters:

- *viewModels*

Throws:

- *NullParameterError* if a mandatory parameter is null

persist() : void[SIDE EFFECT] Persists immediately.

refresh(components : Collection<UIComponent>) : void[SIDE EFFECT] Slates given components for refresh. The components are refreshed when the listener ends.

Parameters:

- *components* A list of components to refresh

removeAll(container∗ : UIComponent) : void[SIDE EFFECT] Removes all children from given container. Only horizontal layout, vertical layout and form layout are supported.

Parameters:

- *container* Container whose children are to be removed. Only horizontal layout, vertical layout and form layout are supported.

Throws:

- *NullParameterError* if a mandatory parameter is null

removeAndDestroy(what∗ : UIComponent, where : UIComponent) : void[SIDE EFFECT] Removes and destroys given component. If called from a listener, the "where" parameter is ignored. When called from form initializer, "where" must point to "what"'s parent.

Parameters:

- *what* Which component to remove and destroy. Must not be a popup nor a table column.
- *where* Where to add the component. Only horizontal layout, vertical layout and form layout are supported. Ignored when called from a listener.

Throws:

- *NullParameterError* if a mandatory parameter is null

removeTab(tabbedLayout∗ : TabbedLayout, tab∗ : Tab) : voidSIDE EFFECT Dynamically removes a tab to given tabbed layout. If the tab is not yet present in the tabbed layout, this function does nothing.

Parameters:

- *tabbedLayout*
- *tab*

Throws:

- *NullParameterError* if a mandatory parameter is null

requestSubmit() : voidSIDE EFFECT Requests submit after all listeners are processed

requestSubmitAndNavigate(navigateTo∗ : Navigation) : voidSIDE EFFECT Requests submit and navigation after all listeners are processed

Parameters:

- *navigateTo*

Throws:

- *NullParameterError* if a mandatory parameter is null

restoreColumnStates(table∗ : UIComponent, columnStates∗ : List<TableColumnState>) : voidSIDE EFFECTDEPRECATED Requies Vaadin 7. Restore the state to the table columns. The state of table columns can be obtained by calling 'getColumnStates' function.

Parameters:

- *table*
- *columnStates*

Throws:

- *NullParameterError* if a mandatory parameter is null

selectTab(tabbedLayout∗ : TabbedLayout, tab∗ : Tab) : voidSIDE EFFECT Selects given tab on given tabbed layout. Does nothing if the tabbed layout does not contain such tab.

Parameters:

- *tabbedLayout*
- *tab*

Throws:

- *NullParameterError* if a mandatory parameter is null

showConstraintViolations(constraintViolations∗ : List<ConstraintViolation>) : voidSIDE EFFECT Maps given constraint violations to Vaadin components, according to the exclude/include rules.

Parameters:

- *constraintViolations*

Throws:

- *NullParameterError* if a mandatory parameter is null

3.3 Hints

3.3.1 Ui

html-class Applicable for components: UIComponent

Hint value type:String

Predefined hint options:

Label	Expression
no-text	#"icon-only"
no-border	#"l-border-none"
border-left	#"l-border-left"
border-top	#"l-border-top"
border-right	#"l-border-right"
border-bottom	#"l-border-bottom"
content-highlight	#"l-highlighted"
content-emphasized	#"l-emphasized"
content-grayed	#"l-grayed"
allow-overflow	#"l-overflow-visible"
border	#"l-border"

Value of this hint ends up as a class attribute of the html element.

selectable Applicable for components: Table

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true

Allows one to select a row in a table.

disable-collapsing Applicable for components: TableColumn

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

By default all table columns are collapsible. To prevent unwanted collapsing of important table columns use this hint.

header-align Applicable for components: TableColumn, GridColumn

Hint value type:String

Predefined hint options:

Label	Expression
Left	#"left"
Center	#"center"
Right	#"right"

By default the table header inherits its alignment from the child component. Using this hint, you can override this mechanism and specify the table header text alignment.

page-length Applicable for components: ComboBox

Hint value type:Decimal

Predefined hint options:

Label	Expression
disable-lazy	0
default	10

Sets the page length for the suggestion popup. Setting the page length to 0 will disable suggestion popup paging (all items visible).

suffix Applicable for components: TextBox

Hint value type:String

Adds suffix to this TextBox preserving existing layout. Used to show currency symbols or custom texts related to the input.

icon Applicable for components: UIComponent

Hint value type:String

Defines the name of an icon from the font Awesome, for example, "wpexplorer", "hand-o-up".

width Applicable for components: UIComponent

Hint value type:String

Predefined hint options:

Label	Expression
100%	#"100%"
10em	#"10em"
Wrap Content	#"wrap-content"
Fill Parent	#"fill-parent"

Width of the component. The value can be expressed in any unit supported by vaadin - com.vaadin.↩ terminal.Sizeable.Unit (px, pt, pc, em, ex, mm, cm, in, %).

Column width can be specified in pixels only. This is true for both the table columns and grid columns.

Wrap-Content makes the component width wide enough to contain all children without overlapping. Equal to setting this value to null. Fill-Parent makes the component as wide as its parent. Equal to "100%".

height Applicable for components: UIComponent

Hint value type:String

Predefined hint options:

Label	Expression
100%	#"100%"
10em	#"10em"
Wrap Content	#"wrap-content"
Fill Parent	#"fill-parent"

Height of the component. The value can be expressed in any unit supported by vaadin - com.vaadin.↩ terminal.Sizeable.Unit (px, pt, pc, em, ex, mm, cm, in, %)

Wrap-Content makes the component width high enough to contain all children without overlapping. Equal to setting this value to null. Fill-Parent makes the component as high as its parent. Equal to "100%".

height-by-rows Applicable for components: Grid

Hint value type:Decimal

Sets the number of rows that should be visible in the Grid's body. This hint overrides the 'height' hint. Value of 0 or negative has the same effect as if the hint wouldn't be set.

expand Applicable for components: TableColumn, UIComponent

Hint value type:Decimal

Predefined hint options:

Label	Expression
Full	1

Sets the expand ratio for a column or component. This hint can only be applied to a table column, a grid column, or a direct child of vertical / horizontal layout (form layout is unsupported).

Expand ratios can be defined to customize the way how excess space is divided among columns in a table or among components in a vertical / horizontal layout. Table can have excess space if it has its width defined and there is horizontally more space than columns consume naturally. Excess space is the space that is not used by columns with explicit width or with natural width (no width nor expand ratio).

When specified for a GridColumn only integer values are accepted.

align Applicable for components: UIComponent

Hint value type:String

Predefined hint options:

Label	Expression
Top left (default)	#"top left"
Top center	#"top center"
Top right	#"top right"
Middle left	#"middle left"
Centered	#"middle center"
Middle right	#"middle right"
Bottom left	#"bottom left"
Bottom center	#"bottom center"
Bottom right	#"bottom right"

Sets the alignment of the component.

This is applicable only for direct children of VerticalLayout, HorizontalLayout, GridLayout, TableColumn and GridColumn. When this hint is applied to a GridColumn or the child of the TableColumn this hint not only controls the child alignment, it also controls the column header alignment. You can override this with the header-align hint.

Aligning children of TableColumn vertically is not supported, vertical aligns are ignored for these components.

size Applicable for components: MultiSelectList, SingleSelectList

Hint value type:Integer

Predefined hint options:

Label	Expression
One row	1
Two rows	2
Three rows	3

Determines number of rows. If not specified (or set to 0), number of rows is determined implicitly.

resizable Applicable for components: Popup

Hint value type:Boolean

Predefined hint options:

Label	Expression
True (default)	true
False	false

Sets the resizability of the popup window. If the hint is not used, popup is resizable.

layout Applicable for components: RadioButtonList,CheckBoxList

Hint value type:String

Predefined hint options:

Label	Expression
Horizontal	#"horizontal"
Vertical (default)	#"vertical"

Determines layout of radio buttons, checkboxes. If not specified, vertical layout is used.

max-text-size Applicable for components: TextBox, TextArea

Hint value type:Integer

Predefined hint options:

Label	Expression
255 characters	255

Sets the maximum number of characters in the field. If not specified or -1, unlimited length is considered.

initial-page-size Applicable for components: Table

Hint value type:Integer

Predefined hint options:

Label	Expression
10 rows per page	10
20 rows per page	20
30 rows per page	30
50 rows per page	50
100 rows per page	100

Number of rows per page of the paged table when it is first displayed. If not specified, 20 rows per page is considered.

tab-order Applicable for components: InputComponent, Button, ActionLink, NavigationLink, FileDownload

Hint value type:Integer

Value of this hint ends up as a tabindex attribute of the html element. Having two components with the same tabindex is not treated in any special way (i.e. tab order is undefined).

initial-focus Applicable for components: InputComponent

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

If the value is true, this element will have initial focus. If multiple components have this property, behaviour is unspecified.

open-in-new-window Applicable for components: NavigationLink

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

Determines whether the link target will be opened in current or new window.

hidden Applicable for components: TableColumn, GridColumn

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

If the value is true, table or grid column is hidden by default.

file-upload-mime Applicable for components: FileUpload

Hint value type:String

Predefined hint options:

Label	Expression
All files (default)	null
Images	#"image/∗"
Videos	#"video/∗"
Sound files	#"audio/∗"

Restricts files which can be uploaded by mime type. Support varies by browser and thus it may be ignored.

spacing Applicable for components: GridLayout, HorizontalLayout, VerticalLayout, TabbedLayout, Popup, Panel

Hint value type:Boolean

Predefined hint options:

Label	Expression
Enabled	true
Disabled	false

Enables/disables spacing of inner content. Default value for GridLayout, HorizontalLayout, VerticalLayout is disabled, for TabbedLayout, Popup, Panel is enabled.

margin Applicable for components: GridLayout, HorizontalLayout, VerticalLayout, TabbedLayout, Popup, Panel

Hint value type:Boolean

Predefined hint options:

Label	Expression
Enabled	true
Disabled	false

Enables/disables margin around this container, in case of TabbedLayout margin is around inner content. Default value for GridLayout, HorizontalLayout, VerticalLayout, Panel is disabled, for TabbedLayout, Popup is enabled.

breadcrumbs Applicable for components: HorizontalLayout

Hint value type:String

Predefined hint options:

Label	Expression
Numbered	#"numbered"
Plain	#"plain"
Disabled	null

Enables breadcrumbs inside this layout. All ActionLinks will act like breadcrumbs.

disable-runtime-performance-warnings Applicable for components: Table

Hint value type:Object

Predefined hint options:

Label	Expression
Yes	null

If specified, it disables writing warnings about in-memory sorting or filtering of more than 500 rows to the server log.

additional-formats Applicable for components: TextBox

Hint value type:List<String>

Adds additional formats accepted by the date text box. Uses the Java SimpleDateFormat formatting.

no-data-message Applicable for components: Table

Hint value type:String

A message which should be displayed in the table body when the table has no data.

simulate-click-on-session-close Applicable for components: Button, ActionLink

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

When the vaadin ui is closing, the component will perform action listener as if the component was clicked. Vaadin ui is closing when:

- session expires due to timout,
- hearthbeat did not arrived two times per row (some undefined time after user closed the tab).

Chapter 4

Module reports

The *reports* module contains functions that allow you to use Jasper reports in LSPS forms and display them in Browser Frames or export them as PDF.

- Data Types
- Functions

4.1 Data Types

4.1.1 Reports

ReportFormat Available report formats

pdf PDF format of the report

xlsx MS Excel format of the report

docx MS Word format of the report

4.2 Functions

4.2.1 Reports

embeddedJasperReportUrl(module∗ : String, reportPath∗ : String, reportParameters : Map<String, Object>) : String

When using internal embedded simple Jasper Server: Produces a JasperReport Server URL which can be passed directly to the Browser Frame. The URL will display given report as HTML.

Parameters:

- *module*
- *reportPath*
- *reportParameters*

jasperReportExport(module∗ : String, path∗ : String, format∗ : reports::ReportFormat, reportParameters : Map<String, Obj
Runs given Jasper Report and produces a single file of the required type.

Parameters:

- *module*
- *path*
- *format*
- *reportParameters* Optional report parameters

jasperReportUrl(jasperServerUrl∗ : String, username : String, password : String, reportPath∗ : String, embedded∗ : Boolean
When using external Jasper Server: Produces a JasperReport Server URL which can be passed directly to the Browser Frame. The URL will display given report as HTML.

Parameters:

- *jasperServerUrl* Jasper Server URL, for example `http://localhost:8080/jasperserver/`
- *username* Optional Jasper Server username. Warning - the username is transmitted as a part of the URL - this is not secure!
- *password* Optional Jasper Server password. Warning - the password is transmitted as a part of the URL - this is not secure!
- *reportPath* The path to report, e.g. /reports/interactive/CustomersReport
- *embedded* If false, a menu bar is displayed along with the report, which allows you to export the report as PDF
- *reportParameters* Optional report parameters

Chapter 5

Module human

The *human* module defines a set of data types, functions and task types used to reflect the organization model, access the user data from a process model, and to generate to-dos.

It imports the *core* module.

- Data Types
- Functions
- Tasks

5.1 Data Types

5.1.1 Human

Performer[ABSTRACTSYSTEM] An abstract record type referring to a process performer. Since it is the supertype of RoleUnit and Person, performers can be defined as either or both of these.

name : String Name of the process performer

Person[SHAREDSYSTEM] **extends Performer** An application user

id : String unique identifier and the primary key of the person

firstName : String first name of the person

lastName : String last name of the person

email : String e-mail address of the person

phone : String phone of the person

isEnabled : Boolean true if the person is enabled; false if the person is disabled.

RoleUnit[ABSTRACTSYSTEM] **extends Performer** An abstract record type that represents both, an organization role or an organization unit.

parameters : Map<String, String> map of organization-unit parameters or organization-role parameters

metadata : Map<String, String> metadata key-value pairs of the role or role unit

Role[SYSTEM] **extends RoleUnit** organization role

OrganizationUnit[SYSTEM] **extends** RoleUnit organization unit

Todo[SHAREDSYSTEM] A to-do

id : Integer unique identifier of the to-do

title : String title of the to-do

start : Date issue date of the to-do

finish : Date date when the to-do finished;
null if the to-do is not finished yet

state : String current execution state of the to-do;
Possible values: "ALIVE", "ACCOMPLISHED", "INTERRUPTED", and "SUSPENDED"

interruptionReason : String Reason for interruption if in the state "INTERRUPTED"

task : String user task that generated the todo

modelInstanceId : Integer parent model instance

allocatedToId : String person who the to-do is allocated to; null if not allocated

TodoEscalation[SHAREDSYSTEM] signal of to-do escalation

id : Integer unique identifier of the to-do escalation

reason : String reason for the escalation

todoStatus : String status of the to-do at escalation;
Possible values: "Unlocked", "Locked", "Accomplished", and "Interrupted"

time : Date time of escalation

DocumentType[SYSTEM] The document type

name : String name of the document type

SavedDocument[SHAREDSYSTEM] Saved document

id : Integer identifier of the saved document

parameters : Map<String, Object> parameters of the saved document

savedDate : Date date when the document was saved

isDeleted : Boolean True if the saved document has been deleted; False if the document is still saved.

UIDefinition[ABSTRACT] An abstract record type used to represent any kind of user interface definition, for instance, UI components or screen flows or custom UI.

TodoListCriteria Criteria that a to-do returned by a query with to-do join must meet.

person : Person person whose to-dos will be included included in the result

includeSubstituted : Boolean includes to-dos of substitutes; If false or null, only person's own to-dos are included

includeAllocatedByOthers : Boolean includes to-dos allocated by other persons are included; If false or null,
only unallocated to-dos or to-dos allocated by the specified person are included

includeRejected : Boolean includes rejected to-dos

includeAllStates : Boolean includes todos in interrupted, accomplished, suspended state

QueryTodo[SHAREDSYSTEM] To-do used in join of queried records with person's to-do list;
Instances of this record are temporary and can be used only in queries.

id : Integer identifier of the to-do

title : String title of the to-do

start : Date issue date of the to-do

finish : Date finish date of the to-do; null if the to-do is not finished

state : String current execution state of the to-do;
Possible values: "ALIVE", "ACCOMPLISHED", "INTERRUPTED", and "SUSPENDED".

interruptionReason : String reason for interruption for to-do in state "Interrupted"

task : String corresponding user task in the process model

allocatedToId : String person who has the to-do allocated; null if the to-do is not allocated

modelInstanceId : Integer identifier of the parent model instance

5.1.2 Navigation

Navigation[ABSTRACT] An abstract record type that represents any navigation.

openNewTab : Boolean Determines whether the link will be opened in new tab. Works in Chrome and Edge. Blocked by popup blocker in Firefox, user have to allow it. After they allow it, Firefox opens new window instead of new tab.

DocumentNavigation extends Navigation Navigation to a document.

documentType : DocumentType type of the document to navigate to

parameters : Map<String, Object> parameters of the document specified as a map of name-value pairs

SavedDocumentNavigation extends Navigation Navigation to a saved document

savedDocument : SavedDocument saved document to navigate to

TodoNavigation extends Navigation Navigation to a to-do. If the current user has no access rights for the to-do, the "no access rights" exception occurs.

todo : Todo To-do to navigate to.

openAsReadOnly : Boolean If true, the to-do is opened in read-only mode. If false, the to-do is opened in read-write mode.

AppNavigation extends Navigation Navigation to an application page.

code : String Code of the application page; the following codes are supported: "todoList", "documents", and "runModel"

UrlNavigation extends Navigation Navigation to a web page given by URL.

url : String URL of the page to navigate to

HistoricalNavigation[ABSTRACT] **extends Navigation** An abstract record type that represents any navigation in history of browsing the application pages. Its sub-types are created by the application and put to the navigation history.

id : String identifier of the entry in the history to navigate to

firstDisplay : Date date when the application page was first displayed

title : String title of the page from navigation history

HistoricalDocumentNavigation[SYSTEM] **extends HistoricalNavigation** Navigation to a document from the navigation history

documentType : DocumentType type of the document to navigate to

parameters : Map<String, Object> parameters of the document specified as a map of name-value pairs.

HistoricalSavedDocumentNavigation[SYSTEM] **extends HistoricalNavigation** Navigation to a saved document from the navigation history

savedDocument : SavedDocument saved document to navigate to

HistoricalTodoNavigation[SYSTEM] **extends HistoricalNavigation** Navigation to a to-do from the navigation history (If the current user has no access rights for the to-do, the "no access rights" exception occurs.)

todo : Todo to-do to navigate to

openAsReadOnly : Boolean opens the to-do as read-only

HistoricalAppNavigation[SYSTEM] **extends HistoricalNavigation** Navigation to an application page from the navigation history

code : String code of the application page; the following codes are supported by default: "todoList", "documents", and "runModel"

5.2 Functions

5.2.1 Todo

allocateTodo(todo∗ : Todo, person∗ : Person) : void[SIDE EFFECT] Allocates the todo to the person. If the to-do is allocated to another person, any form changes are dropped unless the todo is saved before the call.

Parameters:

- *todo* todo to allocate
- *person* person that belongs to the current todo assignees

Throws:

- *NullParameterError* if a mandatory parameter is null
- *PersonIsNotAssigneeError* if the person does not belong to the set of the current todo assignees

getCurrentTodo() : Todo Returns the currently opened to-do. If not called from the context of a to-do, it returns null.

getTodoCurrentAssignees(todo∗ : Todo) : Set<Person> Returns the set of persons who can see the todo. The function takes the effect of delegation, rejection, and substitution into account.

```
getTodoCurrentAssignees(findById(Todo, 13006))
```

Parameters:

- *todo* todo

Throws:

- *NullParameterError* if a mandatory parameter is null

getTodoPerformers(todo∗ : Todo) : Set<Performer> Returns the set of the performers of the todo as defined by the corresponding taskI; that means it returns the set of the RoleUnits and Persons as defined by the performers parameter of the task.

```
getTodoPerformers(findById(Todo, 13004))
//returns
//{
//human::Person(name->"john", id->"john0", firstName->"john", lastName->"john", email->"john@john.com", p
// human::Role(name->"orgModel::MyRole", parameters->Map<String, Null>, metadata->Map<Null, Null>)
//}
```

Parameters:

- *todo* todo

Throws:

- *NullParameterError* if a mandatory parameter is null

getTodoSubmitter(todo∗ : Todo) : Person Returns the person who submitted the todo. The todo must be in the "Accomplished" state, otherwise the call returns null.

```
getTodoSubmitter(Todo.findById(13005))
```

Parameters:

- *todo* accomplished todo

Throws:

- *NullParameterError* if a mandatory parameter is null

getTodosFor(person∗ : Person) : Set<Todo> Returns the set of alive to-dos that are assigned to the person and not allocated to any other person.

```
getTodosFor(getPerson("john"))
```

Parameters:

- *person*

Throws:

- *NullParameterError* if a mandatory parameter is null

getTodosFor(person∗ : Person, includeAllocatedByOthers∗ : Boolean) : Set<Todo> Returns the set of alive to-dos assigned to the person and provides the option to include or exclude to-dos allocated to other persons.

```
getTodosFor(getPerson("john"), true)
```

Parameters:

- *person*
- *includeAllocatedByOthers* whether the output includes the to-dos allocated to another person

Throws:

- *NullParameterError* if a mandatory parameter is null

reassignTodo(todo∗ : Todo, performers∗ : Set<Performer>) : void[SIDE EFFECT] Changes the set of performers of the todo. Delegations and allocation are removed.

Parameters:

- *todo* todo to reassign
- *performers* the set of new perfomers

Throws:

- *NullParameterError* if a mandatory parameter is null

rejectTodo(todo∗ : Todo, persons∗ : Set<Person>, reason : String) : void[SIDE EFFECT] Rejects the todo on behalf of the persons.

```
rejectTodo(
    getCurrentTodo(),
    { getPerson("admin") },
    "I am admin and will not deal with this."
    )
}
```

Parameters:

- *todo* todo to reject
- *persons* persons that reject the todo
- *reason* message with the rejection reason

Throws:

- *NullParameterError* if a mandatory parameter is null

resetTodo(todo∗ : Todo) : void[SIDE EFFECT] Removes any saved states of the todo.

Parameters:

- *todo*

Throws:

- *NullParameterError* if a mandatory parameter is null

unallocateTodo(todo∗ : Todo) : void[SIDE EFFECT] Unallocates a todo (unlocks a to-do locked by a user) and assigns it to the initial performers as defined in the Performers parameter of the task. Any form changes are dropped unless the todo is saved before the call.

Parameters:

- *todo* to-do to unallocate

Throws:

- *NullParameterError* if a mandatory parameter is null

5.2.2 Organization

addPersonToRole(person∗ : Person, roleUnit∗ : RoleUnit) : void[SIDE EFFECT] Adds the person to the role. Note that atempting to add a person to an organization unit results in a runtime exception.

```
addPersonToRole(getPerson("john"), Developer(["project"->"lsps"]))
```

Parameters:

- *person* person
- *roleUnit* role or role unit

Throws:

- *NullParameterError* if a mandatory parameter is null

anyPerformer() : Performer Returns a performer representing any process performer; with this performer, every performer has access to the entity.

children(roleUnit∗ : RoleUnit) : Set<RoleUnit> Returns the set of the direct children of a role unit, that is, an organization role or a unit.

```
children(getRoleUnitByName("org", "Engineering"))
```

Parameters:

- *roleUnit* role or organization unit

Throws:

- *NullParameterError* if a mandatory parameter is not specified

emailAddresses(performers∗ : Set<Performer>) : Set<String> Returns the set of email addresses of the performers. If a performer is of the type Person, its email address is added to the result. If a performer is of type RoleUnit, email addresses of all persons in that role or organization unit are added to the result.

Parameters:

- *performers*

Throws:

- *NullParameterError* if a mandatory parameter is null

emailAddresses(roleUnits : RoleUnit...) : Set<String> Returns the set of email addresses of persons in the specified roles or organisation units.

Parameters:

- *roleUnits*

getCurrentPerson() : Person Returns the person who has initiated the current model processing request; for example, the person who has submitted a to-do or called a web service.

In the case of receiving time events or signals, the technical person "ProcessAgent" is returned; when called from the management perspective of PDS, the admin user is returned (this applies, for example, when you restart a model instance: the user who started the model instance is set to admin).

getLabel(roleUnit∗ : RoleUnit) : String Returns the roleUnit label.

Parameters:

- *roleUnit*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPerson(name∗ : String) : Person Returns a person with the specified login name.

```
getPerson("john")
```

Parameters:

- *name* name of the person

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *PersonNotFoundError* if there is no such person

getPersonFullName(person∗ : Person) : String Returns the first and last name of the specified person.

```
getPersonFullName("john")
//returns "John Doe"
```

Parameters:

- *person*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPersonPicture(person∗ : Person) : File Returns the profile picture of the person. Returns null if no picture for the person exists.

Parameters:

- *person* person

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getPersonProperties(person∗ : Person) : Map<String, String> Returns the properties of the person.

Parameters:

- *person*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPersonRoles(person∗ : Person) : Set<Role> Returns the set of organization roles assigned to the person.

```
getPersonRoles(getPerson("develWithProject"))[0].parameters
//returns the paramater of the 1st role
```

Parameters:

- *person*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPersonWithId(id∗ : String) : Person Returns a person with the id.

```
getPersonWithId("john0")
```

Parameters:

- *id* id of the person

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *PersonNotFoundError* if there is no such person

getRoleUnitByName(module∗ : String, name∗ : String) : RoleUnit Returns an organization role or an organization unit with the specidi.

```
getRoleUnitByName("org", "Developer")
```

Parameters:

- *module* module with the role unit
- *name* name of the role unit

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *RoleUnitNotFoundError* if there is no such role or a unit

getRoleUnitByName(module∗ : String, name∗ : String, parameters : Map<String, String>) : RoleUnit Returns an existing organization role or an organization unit with the parameters specified in the map and assigns the parameters the specified values.

```
addPersonToRole(
   getPerson("manager"),
   getRoleUnitByName("org", "Developer", ["project" -> "lsps"])
 )
 //gets the manager person and assigns it the Developer role
 //with the parameter project set to lsps
```

Parameters:

- *module* module with the role unit
- *name* name of the role unit
- *parameters* parameters of the role unit with the required values

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *RoleUnitNotFoundError* if no such role or a unit exists

isPersonIn(person∗ : Person, roleUnit∗ : RoleUnit) : Boolean Returns true if the person belongs to the given roleUnit. A person belongs to a given organization role or unit if it belongs to the role directly or if it belongs to any of the role's or unit's descendants (evaluated recursively).

```
isPersonIn(getPerson("john"), getRoleUnitByName("org", "Engineering"))
```

Parameters:

- *person* person
- *roleUnit* role unit the person potentially belongs to

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isPersonStrictlyIn(person∗ : Person, role∗ : Role) : Boolean Returns true if the given person belongs directly to the specified role. The descendant roles are excluded: it is a non-recursive version of the isPersonIn() function applied to organization roles.

```
isPersonStrictlyIn(getPerson("john"), getRoleUnitByName("org", "Developer"))
```

Parameters:

- *person* person
- *role* role the person is potentially in

Throws:

- *NullParameterError* if a mandatory parameter is not specified

personsStrictlyWith(role∗ : Role) : Set<Person> Returns the set of persons that belong directly to the given role. Persons that belong to the descendant roles are not considered. This is a non-recursive version of the personsWith() function applied to organization roles.

Parameters:

- *role*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

personsWith(roleUnit∗ : RoleUnit) : Set<Person> Returns the set of persons that belong to the given role. A person belongs to an organization role if it belongs directly to the specified role or it belongs to any of its descendants (evaluated recursively).

```
personsWith(getRoleUnitByName("org","Developer"))
```

Parameters:

- *roleUnit*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removePersonFromRole(person∗ : Person, roleUnit∗ : RoleUnit) : void[SIDE EFFECT] Remove the person from the role or role unit. Note that atempting to add a person to an organization unit results in a runtime exception.

```
removePersonFromRole(getPerson("john"), MyRole(["parameter" -> "value"]))
```

Parameters:

- *person* person
- *roleUnit* role or role unit

Throws:

- *NullParameterError* if a mandatory parameter is null

setPersonPicture(person∗ : Person, picture : File) : void[SIDE EFFECT] Sets the picture as the profile picture of the person

```
setPersonPicture(getPerson("john"), getResource(module -> "my-module", path -> "john.png");
```

Parameters:

- *person* person
- *picture* picture

Throws:

- *NullParameterError* if a mandatory parameter is not specified

5.2.3 Documents

deleteSavedDocuments(documents∗ : Set<SavedDocument>) : void[SIDE EFFECT] Markes the saved documents as deleted (the documents are retained in the dabase; use isDeleted() to check if a document is marked as deleted).

```
deleteSavedDocuments({findAll(SavedDocument)[0]})
```

Parameters:

- *documents* documents to be deleted

Throws:

- *NullParameterError* if a mandatory parameter is null

5.2.4 Utilities

getUIHistory() : Map<Integer, HistoricalNavigation> Returns UI history. Key 0 contains the "current" page. Positive keys hold older entries (back); negative keys hold newer entries (forward).

sendEmail(subject : String, body* : String, attachments : Set<File>, recipientsTo* : Set<String>, recipientsCc : Set<Strin
Sends an e-mail. Deprecated: use function sendEmail with extended parameters.

Parameters:

- *subject* email subject
- *body* email body paragraphs
- *attachments* attachments
- *recipientsTo* email addresses of recipients TO
- *recipientsCc* email addresses of recipients CC
- *recipientsBcc* email addresses of recipients BCC
- *charset* Name of character encoding used for subject and body of e-mail, for instance, “ISO-8859-1”, “UTF-8”, “windows-1250”, etc. If not speci- fied, the default is “UTF-8”. The supported encodings vary between different implementations of the Java 2 platform, see Java docu- mentation for details.

Throws:

- *NullParameterError* if a mandatory parameter is null

sendEmail(subject : String, body* : String, attachments : Set<File>, from : String, recipientsTo* : Set<String>, recipients
Sends an e-mail. Deprecated: use function sendEmail with extended parameters.

Parameters:

- *subject* email subject
- *body* email body paragraphs
- *attachments* attachments
- *from* email address of sender. If not specified, the default (given by application server settings) is used.
- *recipientsTo* email addresses of recipients TO
- *recipientsCc* email addresses of recipients CC
- *recipientsBcc* email addresses of recipients BCC
- *charset* Name of character encoding used for subject and body of e-mail, for instance, “ISO-8859-1”, “UTF-8”, “windows-1250”, etc. If not speci- fied, the default is “UTF-8”. The supported encodings vary between different implementations of the Java 2 platform, see Java docu- mentation for details.

Throws:

- *NullParameterError* if a mandatory parameter is null

sendEmail(subject : String, body* : String, attachments : Set<File>, from : String, recipientsTo* : Set<String>, recipients
Sends an e-mail.

Parameters:

- *subject* email subject
- *body* email body paragraphs
- *attachments* attachments
- *from* email address of sender. If not specified, the default (given by application server settings) is used.
- *recipientsTo* email addresses of recipients TO
- *recipientsCc* email addresses of recipients CC
- *recipientsBcc* email addresses of recipients BCC
- *mime* Mime subtype (without text/) e.g plain, html, rtf,...

- *charset* character encoding used for subject and body of the e-mail, for instance, “ISO-8859-1”, “UTF-8”, “windows-1250”, etc. If not speci- fied, “UTF-8” is used. Note that supported encodings vary between implementations of Java, see your Java documentation for details.

Throws:

- *NullParameterError* if a mandatory parameter is null

5.3 Tasks

5.3.1 Human

User Generates a to-do that serves to present information to users and collect user input.

title : String title of the generated to-do

performers : Set<Performer> process performers who can see the generated to-do

uiDefinition : UIDefinition content of the generated to-do

escalationTimeout : Duration Deprecated! Use GO-BPMN escalation mechanism (for example throw↩ Escalation() function). Period of time from the start of the task until its escalation. If null, the task never escalates.

issueAction : {Todo : void} closure executed after the to-do is generated with the to-do as its input parameter

navigation : {Set<Todo> : Navigation} closure executed after the task is accomplished. The return value of the closure specifies where the UI is navigated after the to-do is submitted.

Throws:

- *NullParameterError* if a mandatory parameter is null

AddPersonToRole Adds the person to the role. If the role does not exist, it is created.

person : Person person

role : Role role

Throws:

- *NullParameterError* if a mandatory parameter is not specified

RemovePersonFromRole Removes the persom from the role.

person : Person person

role : Role role to remove from the person

Throws:

- *NullParameterError* if a mandatory parameter is not specified

SendEmail Sends an email using the email server of the application server.

subject : String email subject

body : String email body

attachments : Set<File> attachments

from : String email address of sender. If not specified, the default (given by application server settings) is used.

recipientsTo : Set<String> email addresses of recipients TO

recipientsCc : Set<String> email addresses of recipients CC

recipientsBcc : Set<String> email addresses of recipients BCC

mime : String Mime subtype without "text/", e.g plain, html, rtf.

charset : String Name of character encoding used for subject and body of e-mail, for instance, “ISO-8859-1”, “UTF-8”, “windows-1250”, etc. If not speci- fied, the default is “UTF-8”. The supported encodings vary between different implementations of the Java platform, see Java docu- mentation for details.

Chapter 6

Module dmn

The *dmn* module contains resources that allow you to design and work with decision tables as defined in DMN version 1.1.

The documentation on how to use decision tables refer to the `decision tables chapter`.

- Data Types
- Functions

6.1 Data Types

6.1.1 Dmn

DecisionTable Represents a decision table. Contains all necessary information to evaluate a decision table: hit policy, language type, input parameters, output parameters and rules.

label : String An optional label of the decision table.

hitPolicy : HitPolicy Hit policy of the decision table.

language : ExpressionLanguage Language type used in decision table expressions.

module : String An optional module name against which the expressions are validated, parsed and executed.

DecisionTable() CONSTRUCTOR Constructs an empty decision table.

DecisionTable(inputs∗ : List<Parameter>, outputs∗ : List<Parameter>) CONSTRUCTOR Constructs a decision table with given inputs and outputs.

addInput(input∗ : Parameter) : void Adds a single input at the end of the current decision table inputs. Empty rule entries are added as well.

addInputAt(index∗ : Integer, input∗ : Parameter) : void Adds a single input at the specified index position of the current decision table inputs. Empty rule entries are added as well.

addInputs(inputs∗ : List<Parameter>) : void Adds an list of inputs at the end of the current decision table inputs. Empty rule entries are added as well.

addOutput(output∗ : Parameter) : void Adds a single output at the end of the current decision table outputs. Empty rule entries are added as well.

addOutputAt(index∗ : Integer, output∗ : Parameter) : void Adds a single output at the specified index position of the current decision table outputs. Empty rule entries are added as well.

addOutputs(outputs∗ : List<Parameter>) : void Adds an list of outputs at the end of the current decision table outputs. Empty rule entries are added as well.

addRules(rules∗ : Rule...) : void Adds rules.

evaluate(inputValues∗ : Map<String, Object>) : List<Map<String, Object>> Evaluates this decision table. Input values are given in a map, where keys are input parameter names.

evaluate(inputValues∗ : List<Object>) : List<List<Object>> Evaluates this decision table. Input values are given in a list in the same order as the input parameters of this decision table.

getHitPolicy() : HitPolicy Returns a decision table hit policy.

getInputs() : List<Parameter> Returns a decision table inputs.

getLabel() : String Returns a decision table label.

getLanguage() : ExpressionLanguage Returns a decision table language.

getModule() : String Returns a decision table module. May be null.

When specified, the decision table module is used to validate and parsed the expressions in the decision table.

getOutputs() : List<Parameter> Returns a decision table outputs.

getRules() : List<Rule> Returns a decision table rules.

initialize() : void Initializes the decision table. Implementation of this method does nothing. Subclasses typically override this method.

load(id : String) : Boolean Loads the content of this decision table from a previously stored decision table. Returns true in the case of successful loading and false when there is no saved decision table with the specified identifier.

removeInputAt(index∗ : Integer) : void Removes an input at the specified index position. Rule entries at the index column are removed as well.

removeOutputAt(index∗ : Integer) : void Removes an output at the specified index position. Rule entries at the index column are removed as well.

removeRule(rule∗ : Rule) : void Removes the specified rules.

save(id : String) : void Persistently stores the content of the decision table under the selected identifier.

setHitPolicy(hitPolicy∗ : HitPolicy) : void Sets the decision table hit policy.

setLabel(label : String) : void Sets the decision table label.

setLanguage(language∗ : ExpressionLanguage) : void Sets the decision table language.

setModule(module : String) : void Sets a decision table module. May be null.

When specified, the decision table module is used to validate and parsed the expressions in the decision table.

Rule FINAL Represents a rule (a row) of a decision table.

description : String An optional rule description.

Rule() CONSTRUCTOR Creates a rule with no entries.

Rule(inputEntries : List<Entry>, outputEntries : List<Entry>) CONSTRUCTOR Creates a rule with a given list of inputs and outputs.

getDescription() : String Returns a rule description.

getInputEntries() : List<Entry> Returns a list of inputs.

getOutputEntries() : List<Entry> Returns a list of outputs.

setDescription(description : String) : void Sets a rule description.

Entry FINAL Represents a single rule entry. An entry contains

either an expression or a selection of allowed values.

expression : String An entry expression. It is null if the entry contains a selection of allowed values.

Entry() CONSTRUCTOR Creates an empty rule entry.

Entry(allowedValues : Set<AllowedValue>) CONSTRUCTOR Creates a rule entry with a selection of allowed values.

Entry(expression : String) CONSTRUCTOR Creates a rule entry with a given expression.

getAllowedValues() : Set<AllowedValue> Returns an allowed value selection.

getExpression() : String Returns a rule entry expression.

ParameterFINAL Represents both input and output parameter of the decision table.

name : String A name of the parameter. Name of input parameter is optional (such inputs are called anonymous).
Name of output parameter is required.

type : String Type of the parameter. Anonymous parameters should not specify type.

multiAllowedValues : Boolean If the parameter specifies allowed values, this flag specifies whether multi selection of allowed values in a rule is possible.

Parameter() CONSTRUCTOR Constructor for anonymous inputs.

Parameter(allowedValues : List<AllowedValue>, multiAllowedValues : Boolean) CONSTRUCTOR Constructor for anonymous inputs with allowed values.

Parameter(name : String, 'type' : Type<Object>) CONSTRUCTOR Creates a parameter with a given name and type.

Parameter(name : String, 'type' : String) CONSTRUCTOR Creates a parameter with a given name and type.

Parameter(name : String, 'type' : Type<Object>, allowedValues : List<AllowedValue>, multiAllowedValues : Boolean Creates a parameter with a given name, type and allowed values.

Parameter(name : String, 'type' : String, allowedValues : List<AllowedValue>, multiAllowedValues : Boolean) CONSTRU Creates a parameter with a given name, type and allowed values.

getAllowedValues() : List<AllowedValue> Returns a list of allowed values.

getName() : String Returns a parameter name.

getType() : String Returns a parameter type.

isMultiAllowedValues() : Boolean Returns true if the parameter allows multi-selection of allowed values in the rule entries.

AllowedValueFINAL Represents an allowed value.

expression : String Expression of the allowed value.

AllowedValue(expression : String) CONSTRUCTOR Creates an allowed value object with a given expression.

getExpression() : String Returns the expression of the allowed value.

DecisionTableComponent extends FormComponent Renders decision table in a form and allows decision table editing. The table is read-only by default. User will be able to perform only rights specified using <code>setRights</code> method.

refresh() : void

setDecisionTable(dt : DecisionTable) : void Set decision table to be edited.

setRights(rights : dmn::DecisionTableRights...) : void Configure user rights for decision table component. The table is read-only by default. User will be able to perform only rights listed here.

setRights(rights : Collection<dmn::DecisionTableRights>) : void Configure user rights for decision table component. The table is read-only by default. User will be able to perform only rights listed here.

HitPolicy Enumeration of supported decision table hit policies.

FIRST Hit policy that returns the ouput(s) of the first matching rule.

RULE_ORDER Hit policy that returns the ouput(s) of all the matching rules in order.

DecisionTableRights Enumeration of rights controlling what the user can do with the decision table, when edited in Web UI.

CAN_CHANGE_NAME A right specifying that the user can edit the decision table name.

CAN_CHANGE_LANGUAGE

CAN_CHANGE_INPUTS A right specifying that the user can change the decision table inputs.

CAN_CHANGE_OUTPUTS A right specifying that the user can change the decision table outputs.

CAN_CHANGE_RULES A right specifying that the user can change the decision table rules.

CAN_EVERYTHING A right specifying that the user can change all properties of the decision table.

ExpressionLanguage Enumeration of supported languages in decision tables.

LSPS LSPS expression language.

S_FEEL S-FEEL expression language.

6.2 Functions

6.2.1 Dmn

findDecisionTableIds(idPattern : String) : Set<String> Returns a set of ids of the saved decision tables matching the idPattern regexp. If the idPattern parameter is null, it returns ids of all saved decision tables.

Parameters:

- *idPattern*

findDecisionTables(idPattern : String) : Set<DecisionTable> Returns a set of the saved decision tables matching the idPattern regexp. If the idPattern parameter is null, it returns all saved decision tables. A decision table is returned as an initialized instance of the DecisionTable record.

Parameters:

- *idPattern*

Chapter 7

Module forms

The *forms* module defines a set of data types and functions used to define application user interface.

The specified data types mostly represent UI components, UI-related events and event listeners. All of them are used to define forms in GO-BPMN models.

- Data Types
- Functions

7.1 Data Types

7.1.1 Forms

FormComponent[ABSTRACT] The base of all components. All Forms framework components must extend this class.

modelingId[DEPRECATED] **: String** Modeling id of the component. Do not use the field directly, use getModelingId and setModelingId methods instead.

data : Object Arbitrary data which you can use for any purpose. This field is not used by the Forms framework itself.

includeConstraintViolation : {ConstraintViolation : Boolean}

excludeConstraintViolation : {ConstraintViolation : Boolean}

internalState : Object A property used by the framework to store component state when saving document or todo. Do not use this property directly.

FormComponent() [CONSTRUCTOR]

_getAllErrors(custom : Boolean, data : Boolean, validators : Boolean) : List<String>

_getErrorsRecursive(custom : Boolean, data : Boolean, validators : Boolean) : Map<FormComponent, List<String>>

addMessageListener(listener : {MessageEvent : void}) : String

addStyleName(style : String) : void Adds one or more style names to this component. Multiple styles can be specified as a space-separated list of style names. The style name will be rendered as a HTML class name, which can be used in a CSS definition.

```
Label label = new Label("This text has style");
label.addStyleName("mystyle");
```

Each style name will occur in two versions: one as specified and one that is prefixed with the style name of the component. For example, if you have a Button component and give it "mystyle" style, the component will have both "mystyle" and "v-button-mystyle" styles. You could then style the component either with:

```
.mystyle {font-style: italic;}
```

or

```
.v-button-mystyle {font-style: italic;}
```

This method will trigger a RepaintRequestEvent.

Parameters:

- *style* the new style to be added to the component

addStyleSheet(sheet : String) : String Generates new unique id and adds a new stylesheet with that id. The style sheet can be removed by removeStyleSheet(String id) method. It is also automatically removed when the component is detached from the page.

Parameters:

- *sheet* - css sheet

Returns:

- id - uniquely identifies the style sheet

addStyleSheet(id : String, sheet : String) : String Adds a new stylesheet to the page or modifies existing stylesheet. The style sheet can be removed by removeStyleSheet(String id) method. It is removed when the component is detached from the page.

Parameters:

- *id* - uniquely identifies the style sheet
- *sheet* - css sheet

Returns:

- id

call(method∗ : String, args : Object...) : Object Calls a method on the underlying Java UI component. You can use this method e.g. for calling methods of component-produced objects on the component itself. For example, the Tab itself has utility methods which delegates to the TabSheet itself. Default implementation will first try to find given method reflectively on the FormComponent-based Java class; if there is no method with given name and given parameter count (parameter types are not matched), the underlying Vaadin widget is searched. If there is no such method, this method fails with an exception.

Parameters:

- *method* the method name, not null.
- *args* parameters

Returns:

- optional method result, may be null. Note that LSPS will pass Decimal instead of Java's Integer here, so make sure you are not calling Vaadin Component methods directly with Decimals.

call(method∗ : String, args : List<Object>) : Object[DEPRECATED] Calls a method on the underlying Java UI component.

DEPRECATED. Use call(String method, Object... args)

clearCustomErrorMessagesRecursive() : void Clears custom error message in this component and in all its children.

createNative() : void

excludeConstraintViolation(violation : ConstraintViolation) : Boolean Tells whether the 'exclude↩ ConstraintViolation' predicate evaluates the constraint violation as the one which should not be displayed on this component. When this predicate is not set then this method returns false.

Parameters:

- *violation* the constraint violation to evaluate

Returns:

- true if the predicate tells that the constraint violation should not be displayed on this component.

executeAfter(intervalMs : Decimal, closure : { : void}) : String Turns on polling and executes the closure exactly once approximately after intervalMs. Returns unique id that can be used to stop the closure.

The closure is run only while the component is attached to the screen. It stops when component is detached.

Polling causes the browser to send request to server at regular interval. This allows ui to update visible information after each regular request. Note that frequent polling causes higher server load.

executeRepeatedly(intervalMs : Decimal, closure : { : void}) : String Turns on polling and executes the closure approximately every intervalMs. Returns unique id that can be used to stop the closure.

The closure is run only while the component is attached to the screen. It stops when component is detached.

Polling causes the browser to send request to server at regular interval. This allows ui to update visible information after each regular request. Note that frequent polling causes higher server load.

getAllErrorMessages() : List<String> Collects error, data and validation messages from this component.

getAllErrorMessagesRecursive() : Map<FormComponent, List<String>> Collects error, data and validation messages from this component, all its children and their descendants.

getCaption() : String Returns the current component caption. By default the component has no caption.

Returns:

- caption, may be null.

getComponentAlignment() : Alignment Returns the current Alignment of this component, with respect to its parent layout.

Parameters:

- *child* the child component

Returns:

- the alignment, not null.

getContextClickHandler() : {ContextClickEvent : void} Gets the handler which is notified when a context click event occurs.

Returns:

- the context click handler or null if none is installed.

getCustomErrorMessage() : String Gets the component error message. Return value contains only custom component error message set by `setErrorMessage` method, it does NOT contain data constraint error messages nor validation error messages.

Returns:

- the component error message or `null`

getCustomErrorMessageRecursive() : Map<FormComponent, List<String>> Returns all custom error messages set to this component and to its child components. Returned list does NOT contain validation nor data error messages.

getDataErrorMessages() : List<String> Returns all data error messages assigned to this component by data validation. Returned list does NOT contain validation error messages nor custom error message.

Data validation is typically performed via `validate` validation function, its results are distributed to components using `clearConstraintErrorMessages` function.

getDataErrorMessagesRecursive() : Map<FormComponent, List<String>> Returns all data error messages assigned by data validation to this component and to its child components. Returned list does NOT contain validation error messages nor custom error message.

Data validation is typically performed via `validate` validation function, its results are distributed to components using `clearConstraintErrorMessages` function.

getDescription() : String Gets the components description, used in tooltips and can be displayed directly in certain other components such as forms. The description can be used to briefly describe the state of the component to the user.

Returns:

- component's description `String`

getExcludeConstraintViolation() : {ConstraintViolation : Boolean} Gets the predicate which is used by this component to specify which constraint violation **should not** be displayed on this component.

Parameters:

- *the* predicate

getExpandRatio() : Decimal Returns the expand ratio of this component, with respect to its parent VerticalLayout or HorizontalLayout.

Warning: this method will fail if the component is not placed directly in the VerticalLayout or Horizontal↩ Layout.

Returns:

- expand ratio of given component, null by default.

getHeight() : String Returns the component height, in the form of [number][units], such as 100%, 10em, 25px.

Returns:

- the height of the component; null if the component wraps its contents. Most, if not all, components have default height of null.

getIcon() : Resource Gets the icon resource of the component.

See #setIcon(Resource) for a detailed description of the icon.

Returns:

- the icon resource of the component or null if the component has no icon

getIncludeConstraintViolation() : {ConstraintViolation : Boolean} Gets the predicate which is used by this component to specify which constraint violation should be displayed on this component.

Returns:

- the predicate

getModelingId() : String Gets the modeling ID of this component. Note: if the component is rendered multiple times, the rendered id is different. Each copy gets unique suffix.

Returns:

- the modeling ID

getStyleName() : String Gets all user-defined CSS style names of a component. If the component has multiple style names defined, the return string is a space-separated list of style names. Built-in style names defined in Vaadin or GWT are not returned.

The style names are returned only in the basic form in which they were added; each user-defined style name shows as two CSS style class names in the rendered HTML: one as it was given and one prefixed with the component-specific style name. Only the former is returned.

Returns:

- the style name or a space-separated list of user-defined style names of the component

getUniqueModelingId() : String Gets html id of this component as rendered on the page.

Returns:

- the modeling ID

getValidatorsMessages() : List<String> Returns all validation errors produced by the validators attached to this component. If there are no validators or all of them pass, returns an empty list (never `null`). Only components with value can have validator, other components return an empty list. Returned list does NOT contain data error messages nor custom error message.

getValidatorsMessagesRecursive() : Map<FormComponent, List<String>> Returns all errors from validators added to this component and to its children components. Only components with value can have validator attached. Returned list does NOT contain data error messages nor custom error message.

getWidth() : String Returns the component width, in the form of [number][units], such as 100%, 10em, 25px.

Returns:

- the width of the component; null if the component wraps its contents.

includeConstraintViolation(violation : ConstraintViolation) : Boolean Tells whether the 'include↩ ConstraintViolation' predicate evaluates the constraint violation as the one which should be displayed on this component. When this predicate is not set then this method returns false.

Parameters:

- *violation* the constraint violation to evaluate

Returns:

- true if the predicate tells that the constraint violation should be displayed on this component.

isEnabled() : Boolean Tests whether the component is enabled or not. A user can not interact with disabled components. Disabled components are rendered in a style that indicates the status, usually in gray color. Children of a disabled component are also disabled. Components are enabled by default.

As a security feature, all updates for disabled components are blocked on the server-side.

Note that this method only returns the status of the component and does not take parents into account. Even though this method returns true the component can be disabled to the user if a parent is disabled.

Returns:

- `true` if the component and its parent are enabled, `false` otherwise.

isReadOnly() : Boolean Tests whether the component is in the read-only mode. The user can not change the value of a read-only component. As only ComponentWithValue components normally have a value that can be input or changed by the user, this is mostly relevant only to field components, though not restricted to them. Notice that the read-only mode only affects whether the user can change the *value* of the component; it is possible to, for example, scroll a read-only table.

Returns:

- true if the component is read only.

isVisible() : Boolean Returns true if the component is visible. All components are by default visible.

Returns:

- true if the component is visible, false if not.

localize(message : String) : String

markAsDirty() : void Marks that this component's state might have changed.

removeAllPollingClosures(id : String) : void Removes all polling closures added by this component.

removeMessageListener(id : String) : String

removePollingClosure(id : String) : Boolean Removes polling closure. If all the closures were removed, polling stops. Only closures that have been added into this component can be removed.

removeStyleName(style : String) : void Removes one or more style names from component. Multiple styles can be specified as a space-separated list of style names.

The parameter must be a valid CSS style name. Only user-defined style names added with addStyle↩ Name() or setStyleName() can be removed; built-in style names defined in Vaadin or GWT can not be removed.

- This method will trigger a RepaintRequestEvent.

Parameters:

- *style* the style name or style names to be removed

removeStyleSheet(id : String) : void Removes stylesheet identified by id from the page.

Parameters:

- *id* - uniquely identifies the style sheet

setAppCloseListener(onUnload : { : void}) : void Sets callback on "beforeunload" browser event. It is fired when:

- the browser is about to closed,
- the tab is about to closed,
- user changed url in browser and is leaving the application entirely.

setCaption(caption : String) : void Sets the component's caption. The caption itself is rendered by the parent layout.

Parameters:

- *caption* the new caption, null for no caption.

setComponentAlignment(alignment∗ : Alignment) : void Set alignment for this component in its parent layout. Use predefined alignments from Alignment enum.

Parameters:

- *child* the component to align within it's layout cell.
- *alignment* the Alignment value to be set

setContextClickHandler(handler : {ContextClickEvent : void}) : void Sets a handler which is notified when a context click event occurs. Usually the handler configures the context menu items before the context menu is displayed.

setContextMenuItems(items : List<MenuItem>) : void Sets the context menu items to this component.

Parameters:

- *items* a list of context menu items

setCustomErrorMessage(errorMessage : String) : void Sets custom error message to this component. If the error message is not null, the component is marked with a red asterisk. Custom component error message is shown first, above data constraint error messages and validation error messages.

Parameters:

- *errorMessage* errorMessage the new error message or `null`

setDescription(description : String) : void Sets the component's description. See #getDescription() for more information on what the description is. This method will trigger a RepaintRequestEvent.

The description is displayed as HTML in tooltips or directly in certain components so care should be taken to avoid creating the possibility for HTML injection and possibly XSS vulnerabilities.

Parameters:

- *description* the new description string for the component.

setEnabled(enabled∗ : Boolean) : void Enables or disables the component. The user can not interact with disabled components, which are shown with a style that indicates the status, usually shaded in light gray color. Components are enabled by default.

```
Button enabled = new Button("Enabled");
enabled.setEnabled(true); // The default
layout.addComponent(enabled);

Button disabled = new Button("Disabled");
disabled.setEnabled(false);
layout.addComponent(disabled);
```

This method will trigger a RepaintRequestEvent for the component and, if it is a ComponentContainer, for all its children recursively.

Parameters:

- *enabled* a boolean value specifying if the component should be enabled or not

setExcludeConstraintViolation(predicate : {ConstraintViolation : Boolean}) : void Sets a predicate shich allows to specify which constraint violation **should not** be displayed on this component.

Parameters:

- *predicate* the predicate to set

setExpandRatio(ratio : Decimal) : void This method is used to control how excess space in layout is distributed among components in the parent layout. Excess space may exist if layout is sized and contained non relatively sized components don't consume all available space.

Example how to distribute 1:3 (33%) for component1 and 2:3 (67%) for component2 :

```
layout.setExpandRatio(component1, 1);
layout.setExpandRatio(component2, 2);
```

If no ratios have been set, the excess space is distributed evenly among all components.

Note, that width or height (depending on orientation) needs to be defined for this method to have any effect.

Warning: this method will fail if the component is not placed directly in the VerticalLayout or Horizontal↩ Layout.

Parameters:

- *ratio* the ratio. If null or 0, the expanding is disabled for given child component.

setHeight(height : String) : void Sets the component height, in the form of [number][units], such as 100%, 10em, 25px. Use null to direct the component to wrap its contents tightly.

Parameters:

- *height* the new height, may be null.

setHeightFull() : void Makes the component fill the parent vertically. A shorthand for calling setHeight("100%").

setHeightWrap() : void Makes the component wrap its contents vertically. A shorthand for calling setHeight(null).

setIcon(resource : Resource) : void Sets the icon of the component.

An icon is an explanatory graphical label accompanying a user interface component, usually shown above, left of, or inside the component. Icon is closely related to caption (see setCaption()) and is usually displayed horizontally before or after it, depending on the component and the containing layout.

The image is loaded by the browser from a resource, typically a ThemeResource.

The icon of a component is, by default, managed and displayed by the layout component or component container in which the component is placed. For example, the VerticalLayout component shows the icons left-aligned above the contained components, while the FormLayout component shows the icons on the left side of the vertically laid components, with the icons and their associated components left-aligned in their own columns.

An icon will be rendered inside an HTML element that has the v-icon CSS style class. The containing layout may enclose an icon and a caption inside elements related to the caption, such as v-caption .

Parameters:

- *icon* the icon of the component. If null, no icon is shown and it does not normally take any space.

setIncludeConstraintViolation(predicate : {ConstraintViolation : Boolean}) : void Sets a predicate which allows to specify which constraint violation should be displayed on this component.

Parameters:

- *predicate* the predicate to set.

setModelingId(modelingId : String) : void Sets the modeling ID of this component. Note: if the component is rendered multiple times, the rendered id might not exactly match configured id. Each html component copy gets unique suffix.

Parameters:

- *modelingId* - modeling id to be set to component

setReadOnly(readOnly∗ : Boolean) : void Sets the read-only mode of the component to the specified mode. The user can not change the value of a read-only component. As only ComponentWithValue components normally have a value that can be input or changed by the user, this is mostly relevant only to field components, though not restricted to them. Notice that the read-only mode only affects whether the user can change the value of the component; it is possible to, for example, scroll a read-only table.

Parameters:

- *readOnly* true if the component should be read only.

setSizeFull() : void Makes the component fill the parent. A shorthand for calling setWidthFull() and setHeightFull().

setStyleName(style : String) : void Sets one or more user-defined style names of the component, replacing any previous user-defined styles. Multiple styles can be specified as a space-separated list of style names. The style names must be valid CSS class names and should not conflict with any built-in style names in Vaadin or GWT.

```
Label label = new Label("This text has a lot of style");
label.setStyleName("myonestyle myotherstyle");
```

Each style name will occur in two versions: one as specified and one that is prefixed with the style name of the component. For example, if you have a Button component and give it "mystyle" style, the component will have both "mystyle" and "v-button-mystyle" styles. You could then style the component either with:

```
.myonestyle {background: blue;}
```

or

```
.v-button-myonestyle {background: blue;}
```

It is normally a good practice to use addStyleName() rather than this setter, as different software abstraction layers can then add their own styles without accidentally removing those defined in other layers.

This method will trigger a RepaintRequestEvent.

Parameters:

- *style* the new style or styles of the component as a space-separated list

setVisible(visible∗ : Boolean) : void Changes the component visibility. Invisible component's HTML elements are not present in the browser's DOM tree.

Parameters:

- *visible* true if the component should be visible, false if not.

setWidth(width : String) : void Sets the component width, in the form of [number][units], such as 100%, 10em, 25px. Use null to direct the component to wrap its contents tightly.

Parameters:

- *width* the new width, may be null.

setWidthFull() : void Makes the component fill the parent horizontally. A shorthand for calling setWidth("100%").

setWidthWrap() : void Makes the component wrap its contents horizontally. A shorthand for calling setWidth(null).

showContextMenu(x∗ : Integer, y∗ : Integer) : void Opens context menu on this component.

Parameters:

- *x* - screen coordinate where to put the menu on
- *y* - screen coordinate where to put the menu on

toString() : String

Label extends ComponentWithValue Shows the value; does not allow the value to be edited. Accepts Object and calls toString() on the object. If the value is null, nothing (empty string) is displayed. isValid() always returns true for this component. Does not support validators and is always valid.

Label() CONSTRUCTOR Creates an empty label which initially shows nothing (null).

Label(value : Object) CONSTRUCTOR Creates a label which initially shows given value.

Parameters:

- *value* the value to show, may be null.

Label(value : Object, mode∗ : ContentMode) CONSTRUCTOR Creates a label with a given value and content mode.

Parameters:

- *value* the value to show, may be null.

getCaptionMode() : CaptionMode Checks whether captions are rendered as HTML

The default is false, i.e. to render that caption as plain text.

Returns:

- true if the captions are rendered as HTML, false if rendered as plain text

getContentMode() : ContentMode Gets the content mode of the label.

setCaptionMode(captionMode∗ : CaptionMode) : void Sets whether the caption is rendered as HTML or plain text.

When a caption is rendered as HTML, it is up to the developer to ensure that no harmful HTML can be used. If set to plain text, the caption is rendered in the browser as plain text.

The default is plain text.

Parameters:

- *captionMode* caption mode

setContentMode(mode∗ : ContentMode) : void Sets the content mode of the caption. Content mode defines how the label's content is rendered - as html or as text and whether new line is rendered as is or as tag.

Parameters:

- *mode* the content mode to set

toString() : String

Binding The user of this interface binds a component to a value which the component displays or edits.

It provides values either by holding them directly, wrapping them in a closure or a reference, or by other means.

get() : Object Returns the value of the binding.

Returns:

- binding value

getBinding() : RecordAndProperty If the Binding value represents a writable reference to a record and its property, the function returns the *RecordAndProperty* object with the record and property. In other cases, for example, for a simple value or a closure, the return value is `null`. This method is used, for example, to infer validators with *ComponentWithValue.inferValidator()*.

Returns:

- null if the value does not reference a record and property

set(value : Object) : void Sets the Binding to this value. Note that this operation may not be supported and fail.

Parameters:

- *value* target Binding value

ComponentGroup[ABSTRACT] extends FormComponent A group of components. This component is valid if all child groups and child ComponentWithValues are valid.

addComponent(child∗ : FormComponent) : void Adds component to this component group. Does nothing if the component is already a child of this component group.

Parameters:

- *child* the component, not null.

addComponents(children∗ : Collection<FormComponent>) : void Adds all components to this component group.

Parameters:

- *children* the collection of children to add, not null, may be empty.

getComponentAt(i∗ : Integer) : FormComponent Returns the i-th child of this group. Fails if the index is out-of-range.

Parameters:

- *i* the child index, 0-based. Must be 0..getComponentCount()-1

Returns:

- the i-th child component, not null.

getComponentCount() : Integer Returns the number of child components present in this group.

getComponents() : List<FormComponent> Returns the child components as a list.

Returns:

- the list of all child components, not null, may be empty.

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isMargin() : Boolean Returns:

- true if the group is margined, false if not. Always false for tab sheet.

Returns:

isSpacing() : Boolean

- true if spacing between child components within this layout is enabled, false otherwise.

isValid() : Boolean Checks if all child groups and child ComponentWithValues are valid. If any invalid child is found, returns false.

refresh() : void Calls refresh() on all nested components with values.

removeAllComponents() : void Removes all child components from this group, leaving it empty.

removeComponent(child∗ : FormComponent) : void Removes given component from this group. Does nothing if the component is not a child of this component group.

Parameters:

- *child* the component to remove, not null.

removeComponents(children∗ : Collection<FormComponent>) : void Removes all given components from this group.

Parameters:

- *children* the components to remove, not null, may be empty.

setComponentAlignment(children∗ : Collection<FormComponent>, alignment∗ : Alignment) : void Set alignment for all given contained component in this layout. Use predefined alignments from Alignment enum.

Parameters:

- *children* the components to align within it's layout cell.
- *alignment* the Alignment value to be set

setMargin(margin∗ : Boolean) : void Enable layout margins. Affects all four sides of the layout. This will tell the client-side implementation to leave extra space around the layout. The client-side implementation decides the actual amount, and it can vary between themes.

Does nothing for TabSheet.

Parameters:

- *enabled* true if margins should be enabled on all sides, false to disable all margins

setSpacing(spacing∗ : Boolean) : void Enable spacing between child components within this layout.

NOTE: This will only affect the space between components, not the space around all the components in the layout (i.e. do not confuse this with the cellspacing attribute of a HTML Table). Use #set↩Margin(boolean) to add space around the layout.

See the reference manual for more information about CSS rules for defining the amount of spacing to use.

TabSheet has no spacing support, therefore this method does nothing for tab sheet.

Parameters:

- *enabled* true if spacing should be turned on, false if it should be turned off

toString() : String

VerticalLayout extends OrderedComponentGroup **VerticalLayout()** CONSTRUCTOR Creates an empty vertical layout initially containing no components.

VerticalLayout(children∗ : FormComponent...) CONSTRUCTOR Creates a vertical layout initially containing given components.

Parameters:

- *children* the child components, may be empty.

toString() : String

withMarginSpacing() : VerticalLayout Sets both margin and spacing to true.

Returns:

- this

HorizontalLayout extends OrderedComponentGroup **HorizontalLayout()** CONSTRUCTOR Creates an empty horizontal layout containing no components.

HorizontalLayout(children∗ : FormComponent...) CONSTRUCTOR Creates a horizontal layout containing given components.

Parameters:

- *children* the child components, may be empty.

toString() : String

FormLayout extends OrderedComponentGroup **FormLayout()** CONSTRUCTOR Creates an empty form layout initially containing no components.

FormLayout(children∗ : FormComponent...) CONSTRUCTOR Creates a form layout initially containing given components.

Parameters:

- *children* the child components, may be empty.

toString() : String

CheckBox extends InputComponentWithValue Shows boolean values only. Always convertible.

CheckBox() CONSTRUCTOR Creates a checkbox with no label and simple value holder.

CheckBox(caption : String) CONSTRUCTOR Creates a checkbox with given label and simple value holder.

Parameters:

- *caption* the caption

CheckBox(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a checkbox with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Boolean value.

CheckBox(caption : String, ref∗ : Reference<Boolean>) CONSTRUCTOR Creates a checkbox with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getValue() : Boolean

setBinding(ref : Reference<Boolean>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

toString() : String

ComponentWithValueABSTRACT **extends FormComponent** A component showing a value. The value is not yet editable - please use InputComponentWithValue for that. This component is primarily extended by components which only show some data, for example the Label component.

ComponentWithValue() CONSTRUCTOR

clearBinding() : void Convenience method setBinding(null as Reference<Object>). Clears previous reference and field will end up empty.

getBinding() : Binding Every component with value is bound to a data object. Gets the currently displayed Binding object.

Returns:

- currently displayed Binding, never null.

getValidatorsMessages() : List<String>

getValue() : Object Polls the current Binding for its data. Shorthand for getBinding().get()

Returns:

- getBinding().get()

isConvertible() : BooleanDEPRECATED Deprecated, returns the same thing as #isValid.

isValid() : Boolean

refresh() : void Re-sets the property, causing the value to be re-read and re-displayed.

refreshBinding() : void Refresh component binding.

setBinding(binding∗ : Binding) : void Sets the Binding being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed
Parameters:

- *binding* the new value holder.

setBinding(ref : Reference<Object>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.
Parameters:

- *ref* reference to the new value holder.

setValue(value : Object) : void Sets given value to the currently bound Value. Shorthand for getProperty().set(). It also refreshes the binding.

toString() : String

TextField extends AbstractTextArea Shows String values only. Always convertible.

TextField() CONSTRUCTOR Creates a text field with no label and simple value holder.

TextField(caption : String) CONSTRUCTOR Creates a text field with given label and simple value holder.
Parameters:

- *caption* the caption

TextField(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a text field with given label and given binding.
Parameters:

- *caption* the caption
- *binding* the binding, not null.

TextField(caption : String, ref∗ : Reference<String>) CONSTRUCTOR Creates a text field with given label, referencing given variable/field.
Parameters:

- *caption* the caption
- *ref* the reference not null.

toString() : String

DecimalField extends InputComponentWithValue Shows Decimal values only. If the value typed by the user is not convertible to Decimal, isValid() returns false and the field shows error message.

DecimalField() CONSTRUCTOR Creates a decimal field with no label and simple value holder.

DecimalField(caption : String) CONSTRUCTOR Creates a decimal field with given label and simple value holder.
Parameters:

- *caption* the caption

DecimalField(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a decimal field with given label and given property.
Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Boolean value.

DecimalField(caption : String, ref∗ : Reference<Decimal>) CONSTRUCTOR Creates a decimal field with given label, referencing given variable/field.
Parameters:

- *caption* the caption
- *ref* the reference not null.

Returns:

getUserText() : String • field value as written and currently visible. Unlike `getValue` method, this returns whatever is in field including invalid or unparseable values.

getValue() : Decimal

setNumberFormat(numberFormat : String) : void Sets the number format for the decimal field, for example df.setNumberFormat("#,##0.00;\(#,##0.00\)"). See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html for further information on the formatting string. By default the default number format of user's locale is used.

Parameters:

- *numberFormat* the number format to use; null to use the default number format for user's locale.

setNumberFormats(numberFormats∗ : List<String>) : void Sets the number formats for this decimal field. By default a default number format for user's locale is selected. The field will accept user string if any of the formats is able to convert it to number. If multiple formats match the number, first one is used. When formatting numbers, first format is used.

Please see https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html for information on the formatting string.

Parameters:

- *numberFormats* the number formats to use, empty list to use default number format for user's locale.

setParseErrorMessage(message : String) : void Customize error message that shows up when the string inside the field can not be converted to a number.

setProperty(ref∗ : Reference<Decimal>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed.

Parameters:

- *value* the new value holder.

toString() : String

DateField extends InputComponentWithValue Shows Date values only. If the value typed by the user is not convertible to Date, isValid() returns false.

DateField() CONSTRUCTOR Creates a date field with no label and simple value holder.

DateField(caption : String) CONSTRUCTOR Creates a date field with given label and simple value holder.

Parameters:

- *caption* the caption

DateField(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a date field with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Date value.

DateField(caption : String, ref∗ : Reference<Date>) CONSTRUCTOR Creates a date field with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

Returns:

getUserText() : String • field value as written and currently visible. Unlike `getValue` method, this returns whatever is in field including invalid or unparseable values.

getValue() : Date

setBinding(ref : Reference<Date>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setDateFormat(dateFormat : String) : void

setDateFormats(dateFormats∗ : List<String>) : void Sets the formats accepted by this date field. The date field resolution (or precision - second, minute, hour, day, month, year) is updated automatically. The list may contain a special String constant "date" or "dateTime" - this constant is automatically replaced by application messages named "app.dateFormat" or "app.dateTimeFormat" respectively, via the LspsAppConnector.getApplicationMessage().

For formatting string specification please follow http://docs.oracle.com/javase/6/docs/api/java/tex SimpleDateFormat.html

Parameters:

- *dateFormats* the list of accepted date formats. Must not be null. Must not be empty. First list item format is used to format the value in the field itself. User can however delete the text in the date field and enter a new date in any of these formats.

setParseErrorMessage(parsingErrorMessage : String) : void Sets error message to be used when text in the field can not be parsed into date. If null then default unlocalized error message is used.

toString() : String

RecordAndPropertyREAD-ONLY **record : Record**

property : Property

toString() : String

BrowserFrame extends FormComponent **BrowserFrame()** CONSTRUCTOR Creates a frame which initally shows nothing. Call #setUrl(String) to force the frame to show a web page.

BrowserFrame(url∗ : String) CONSTRUCTOR Creates a frame which shows given URL.

Parameters:

- *url* the URL, not null. Should start with http:// or https://

getUrl() : String Returns the URL set via the #setUrl(String) method. Does not return the current frame URL, which may have been changed for example as a result of redirects and/or user clicking on links.

Returns:

- the URL, may be null if #setUrl(String) has not yet been called.

postMessage(message : String) : void Posts message to target frame.

Parameters:

- *message* - the message

postMessage(message : String, targetOrigin : String) : void Posts message to target frame.

Parameters:

- *message* - the message
- *targetOrigin* - the message is sent only if frame origin matches this URI. Use "∗" to specify any origin.

refresh() : void

setMessageListener(listener : {MessageEvent : void}) : void Installs listener that will get all messages sent by post message methods.

setUrl(url : String) : void Forces the frame to show given URL. The frame may follow redirects and show a completely different URL.

toString() : String

Grid extends FormComponent **addButtonColumn(buttonCaption∗ : String, onClick∗ : {Null : void}) : GridColumn** Adds a column containing just a single button. Utility method which ultimately calls addColumn().

Parameters:

- *buttonCaption* the text rendered on every button.
- *onClick* Called when the button is clicked. Receives the business data for given row as input. Closure result is ignored.

Returns:

- the column, not null.

addColumn(valueProvider∗ : ValueProvider) : GridColumn Adds a sortable column to this Grid. The column is by default filtrable and has no renderer.

Parameters:

- *valueProvider* Provides values to show in this column's cells. Not null.

Returns:

- the newly added column, not null.

addColumn(valueProvider∗ : ValueProvider, header∗ : String) : GridColumn Adds a sortable column to this Grid. The column is by default filtrable and has no renderer.

Parameters:

- *valueProvider* Provides values to show in this column's cells. Not null.
- *header* the column header, not null.

Returns:

- the newly added column, not null.

addColumn(valueProvider∗ : ValueProvider, renderer : Renderer, sortable : Boolean) : GridColumn Adds a column to this Grid. You can call this before or after the data source has been set. The column is by default filtrable and has no header.

Parameters:

- *valueProvider* Provides values to show in this column's cells. Not null.
- *renderer* Renders every cell in this Grid column - generates HTML code and updates the code according to the row-extracted data. The HTML code generator is implemented in Java - for details please see Vaadin documentation on Grid Renderers.
- *sortable* If false, the column will not be sortable.

Returns:

- the newly added column, not null.

addColumn(valueProvider∗ : ValueProvider, renderer : Renderer, sortable : Boolean, editable : Boolean, editor : Edito Adds a column to this Grid. You can call this before or after the data source has been set. The column is by default filtrable and has no header.

Parameters:

- *valueProvider* Provides values to show in this column's cells. Not null.
- *renderer* Renders every cell in this Grid column - generates HTML code and updates the code according to the row-extracted data. The HTML code generator is implemented in Java - for details please see Vaadin documentation on Grid Renderers.
- *sortable* If false, the column will not be sortable.
- *editable* If false, the column will not be editable.
- *editor* Editor definition if the column is editable and the value requires special editor.

Returns:

- the newly added column, not null.

addColumn(valueProvider∗ : ValueProvider, filterValueProvider : ValueProvider, sortValueProvider : ValueProvider, re

addLinkColumn(linkCaption∗ : String, onClick∗ : {Null : void}) : GridColumn Adds a column containing just a single link. Utility method which ultimately calls addColumn().

Parameters:

- *linkCaption* the text rendered on every link.
- *onClick* Called when the button is clicked. Receives the business data for given row as input. Closure result is ignored.

Returns:

- the column, not null.

getColumns() : List<GridColumn> Returns the current list of columns in the current order.

Returns:

- the current columns, not null, may be empty.

getComponents() : List<FormComponent> Returns the list of the components currently rendered in the grid.

The components in the grid are transient: The grid detaches and forgets its components, and creates and attaches new ones as the user scrolls. The method returns the components currently attached to the page, that is, the displayed rows plus a few rows above and a few rows below. The components that are not loaded or have been detached are not returned. Therefore, do not store the returned components for later. The returned components are not ordered.

Returns:

- a list of contained components

getDataSource() : DataSource Returns the current Grid data source.

Returns:

- the currently set data source. If none has been set yet, this returns null.

getFrozenColumnCount() : Integer Gets the number of frozen columns in this grid. 0 means that no data columns will be frozen, but the built-in selection checkbox column will still be frozen if it's in use. -1 means that not even the selection column is frozen.

NOTE: this count includes hidden columns in the count.

Returns:

- the number of frozen columns

getHeightByRows() : Decimal Returns the current row-height. Please see setHeightByRows() for more information.

Returns:

- the current row-height, not null, 0 or greater.

getSelection() : List<Object> Returns list of currently selected row objects or an empty list. The rows were selected either by user clicking on the grid rows, or by calling select() programatically. Grid in single select mode returns a list with one element. Initially, there is no row selected.

Returns:

- currently selected row object, null if there is no selection.

getSelectionChangeListener() : {ValueChangeEvent : void} Gets the current value change listener

Returns:

- the current value change listener or null if none is installed.

getSelectionMode() : SelectionMode Returns current selection mode.

isColumnReorderingAllowed() : Boolean Returns whether column reordering is allowed or not - that is, whether the user can drag a column to the new location or not.

Returns:

- whether column reordering is allowed or not. Never null.

isDetailsVisible(item : Object) : Boolean Returns the visibility of details component for given item.

isSelectAllCheckBoxVisible() : Boolean Returns select all checkbox visibility when in multiselect mode.

isSelectable() : Boolean Returns true if the table is either in single row selection mode or multiple rows selection mode. Returns false otherwise.

recalculateColumnWidths() : void Requests that the column widths should be recalculated.

In most cases Grid will know when column widths need to be recalculated but this method can be used to force recalculation in situations when grid does not recalculate automatically.

Note: if the user manually resized any of the columns, the method does nothing. It wont modify users configuration.

refresh() : void Refreshes the contents of this Grid - polls the data source for fresh data.

select(rowObject : Object) : void Marks an item as selected. Unselects all other items. The row object must have been provided by the DataSource. Calling select(null) clears the selection. Fails if the grid is not selectable.

Parameters:

- *rowObject* select the row for this row object. If null, the selection is cleared.

select(rowObjects : Collection<Object>) : void Changes grid selection. Previous selection is cleared and new set of rows is selected. Exact behavior depends on selection mode:

- NONE - call fails.
- SINGLE - previous selection is cleared. If the `rowObjects` argument is non-empty, only one row is selected.
- SINGLE - previous selection is cleared. If the `rowObjects` argument is non-empty, all objects in it are selected.

All row objects must have been provided by the DataSource.

setColumnOrder(expectedColumnsOrder : List<GridColumn>) : void Sets a new column order for the grid. All columns which are not ordered here will remain in the order they were before as the last columns of grid.

Parameters:

- *columnsInOrders* the columns in the order they should be

setColumnOrder(expectedColumnsOrder* : GridColumn...) : void Sets a new column order for the grid. All columns which are not ordered here will remain in the order they were before as the last columns of grid.

Parameters:

- *columnsInOrders* the columns in the order they should be

setColumnReorderListener(listener : {ColumnReorderEvent : void}) : void Sets callback for columns reorder event.

Parameters:

- *listener* the listener to be notified

setColumnReorderingAllowed(columnReorderingAllowed* : Boolean) : void Sets whether column reordering is allowed or not - that is, whether the user can drag a column to the new location or not. By default this value is set to true.

Parameters:

- *columnReorderingAllowed* specifies whether column reordering is allowed.

setColumnResizeListener(listener : {GridColumn : void}) : void Sets a column resize listener to this grid. Replaces any previously set column resize listener. Setting null will remove any previously set column resize listener.

Parameters:

- *listener* the listener which will be notified when a grid column resize event occrus

setDataSource(dataSource : DataSource) : void Sets a new data source to this Grid. The grid automatically refreshes itself.

Parameters:

- *dataSource* the new data source to be shown. May be null - in such case an empty Grid will be shown.

Returns:

- this

setDescription(tooltipGenerator* : {Null : String}) : void Sets common tooltip on all grid columns. The closure obtains row object and generates tooltip string. If it returns `null`, nothing is shown. Uses "Preformatted" content mode.

Note: Column tooltip (description) set directly on column takes precedence over row tooltip set here.

Parameters:

- *tooltips* - closure obtains row object and generates tooltip

setDescription(tooltipGenerator∗ : {Null : String}, contentMode∗ : ContentMode) : void Sets common tooltip on all grid columns. The closure obtains row object and generates tooltip string. If it returns `null`, nothing is shown.

Note: Column tooltip (description) set directly on column takes precedence over row tooltip set here.

Parameters:

- *tooltips* - closure obtains row object and generates tooltip
- *contentMode* - defines tooltip formatting

setDetailsGenerator(detailsGenerator : {Null : FormComponent}) : void Sets the details component generator. Detail component is shown between grid rows and can be large compared to grid data.

setDetailsVisible(item : Object, visible : Boolean) : void Sets the visibility of details component for given item.

setEditorBuffered(buffered∗ : Boolean) : void Sets the buffered flag for the editor row. If the editor is buffered the changes are not propagated to the model directly but are buffered first and are propagated to the modle only after the user finishes editing by pressing a save button. If user finishes editing by pressing a cancel button the edited changes are lost.

By default the editor is buffered.

setEditorEnabled(enabled∗ : Boolean) : void Sets the enabled state of the value editor. If the editor is enabled the editor is displayed on row double click.

Parameters:

- *enabled* if true the editor row is enabled

setFilterChangedListener(listener : {FilterChangedEvent : void}) : void Sets callback for grid filter event.

Parameters:

- *listener* the listener to be notified

setFiltrable(filtrable∗ : Boolean) : void Setting false here will hide the filter component for all columns. Equivalent of calling GridColumn.setFiltrable for every column.

Parameters:

- *filtrable* if false, the filter components will be hidden for all columns; if true, the filter components will be generated.

setFrozenColumnCount(numberOfColumns∗ : Integer) : void Sets the number of frozen columns in this grid. Setting the count to 0 means that no data columns will be frozen, but the built-in selection checkbox column will still be frozen if it's in use. Setting the count to -1 will also disable the selection column.

The default value is 0.

Parameters:

- *numberOfColumns* the number of columns that should be frozen

Throws:

- *IllegalArgumentException* if the column count is $<$ 0 or $>$ the number of visible columns

setHeightByRows(heightByRows∗ : Decimal) : void Sets the Grid row-height. Setting this to a non-zero value will make the Grid show exactly given number of rows, with the header and a possible scrollbar. In this mode the Grid will ignore its current component height as set by setHeight()/getHeight().

Setting row-height to zero will cause the Grid to again respect the component height.

Defaults to zero. You can use fraction numbers.

As opposed to AbstractTable.setPageLength(), it is not possible to configure Grid's server-client fetch page size.

Parameters:

- *heightByRows* the new row-height, 0 or greater.

setItemClickListener(listener : {ItemClickEvent : void}) : void Sets the listener for row item click events.

Parameters:

- *listener* the listener to set

setRowStyleGenerator(rowStyleGenerator : {Null : String}) : void Sets the style generator that is used for generating styles for rows. The generator obtains a row objects and returns css class that will be added to that row div.

Parameters:

- *styleGenerator* the row style generator to set, or `null` to remove a previously set generator

setSelectAllCheckBoxVisible(visible : Boolean) : void Sets the select all checkbox visibility when in multiselect (MULTI) selection mode. The setting is ignored if the grid is in SINGLE or NONE selection mode.

WARNING: checking select all checkbox causes all items in the datasource to be loaded into memory. Do not use this unless you are sure the datasource is small.

Parameters:

- *visible* - whether select all checkbox is visible

setSelectable(selectable∗ : Boolean) : void If the arguments is true, sets the selection mode to SINGLE. Otherwise sets it to NONE.

setSelectionChangeListener(listener : {ValueChangeEvent : void}) : void Sets or clears the selection change listener. Use getSelection() to query for the current selection.

setSelectionMode(selectionMode∗ : SelectionMode) : void Sets selection mode for the grid. Grid supports three kinds of selection: single row selection, multiple rows selection and no selection. The grid is not selectable by default.

setSortListener(listener : {SortEvent : void}) : void Sets callback for grid sort event.

Parameters:

- *listener* the listener to be notified

toString() : String

GridColumn Do not create directly - instead, use the Grid.addColumn functions.

grid : Grid The owner grid, not null.

valueProvider : ValueProvider Provides values to show in this column's cells.

sortValueProvider : ValueProvider

filterValueProvider : ValueProvider

renderer : Renderer Renders every cell in this Grid column - generates HTML code and updates the code according to the row-extracted data. The HTML code generator is implemented in Java - for details please see Vaadin documentation on Grid Renderers.

If null, the objects provided by the value provider are converted to string (using the toString() function) and displayed as strings. In case of the built-in Button or Link Renderers, the string value obtained from the value provider will be shown as a Button's/Link's caption.

editable : Boolean Enables editing support in this column.

editor : Editor

sortable : Boolean If false, the column will not be sortable.

modelingId[DEPRECATED] : String Modeling id of the component. Do not use the field directly, use getModelingId and setModelingId methods instead.

data : Object Arbitrary data which you can use for any purpose. This field is not used by the Forms framework itself.

getAlignment() : Alignment Returns the current column's child alignment, with respect to the column itself. Defaults to left alignment.

Returns:

- the alignment, never null.

getComponentInRow(rowObject : Object) : FormComponent If the column contains a component renderer, returns the component generated from rowObject. Otherwise returns null.

Returns:

- component or null

getExpandRatio() : Integer Returns the column's expand ratio.

Returns:

- the column's expand ratio

getFilterConfig() : FilterConfig Gets the current filter configuration for this column.

Returns:

- filter configuration or null if this column does not have a filter configuration.

getHeader() : String Returns the caption of the header. By default the header caption is an empty string.

Returns:

- the text in the default row of header.

Throws:

- *IllegalStateException* if the column no longer is attached to the grid

getHeaderAlignment() : Alignment Returns the current column's header content alignment. Defaults to left alignment.

Returns:

- the alignment, never null.

getHeaderStyleName() : String Returns the style name of this column's header.

Returns:

- the column's header style name

getHidingToggleCaption() : String Gets the caption of the hiding toggle for this column.

Returns:

- the caption for the hiding toggle for this column

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

getMaximumWidth() : Integer Returns the maximum width for this column.

Returns:

- the maximum width for this column

getMinimumWidth() : Integer Return the minimum width for this column.

Returns:

- the minimum width for this column

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

getWidth() : Integer Returns the width (in pixels). By default a column is 100px wide.

Returns:

- the width in pixels of the column. null if the width is undefined.

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

isFiltrable() : Boolean Checks whether the Grid should show the filtering component for this column.

Returns:

- true if the column is user-filtrable, false if not.

isHidable() : Boolean Returns whether this column can be hidden by the user. Default is false.

Note: the column can be programmatically hidden using #setHidden(boolean) regardless of the returned value.

Returns:

- `true` if the user can hide the column, `false` if not

isHidden() : Boolean Returns whether this column is hidden. Default is false.

Returns:

- `true` if the column is currently hidden, `false` otherwise

isResizable() : Boolean Returns whether this column can be resized by the user. Default is true.

Note: the column can be programmatically resized using #setWidth(double) and #setWidthUndefined() regardless of the returned value.

Returns:

- true if this column is resizable, false otherwise

isSortAscending() : Boolean Returns true if the grid is sorted in ascending order and false if it is sorted in descending order. Can return anything for unsorted columns.

Returns:

- whether the grid is sorted by the specified column

isSorted() : Boolean Returns whether the grid is sorted by the specified column

Returns:

- whether the grid is sorted by the specified column

remove() : void Removes the column from the grid. This record instance becomes invalid and must not be used anymore.

setAlignment(alignment∗ : Alignment) : void Sets the specified column's child component alignment inside of the column. In order for this to work, the div element produced by renderer must wrap its contents, otherwise the child will just fill the column and there will be no space for the alignment flag to move the child.

Ignores the vertical aspect of the alignment - only the horizontal aspect is applied.

Parameters:

- *alignment* the desired alignment, not null. Defaults to left align.

setCellStyleGenerator(rowStyleGenerator : {Null : String}) : void

setDescription(tooltipGenerator∗ : {Null : String}) : GridColumn Sets the tooltip on the grid column. Column tooltip takes precedence over row tooltip (description) set on grid. If both are set, only column tooltip is shown. The closure obtains row object and generates tooltip string. If it returns `null`, the row tooltip common for all columns is show instead.

Uses "Preformatted" content mode.

Parameters:

- *tooltips* - closure obtains row object and generates tooltip

setDescription(tooltipGenerator∗ : {Null : String}, contentMode∗ : ContentMode) : GridColumn Sets the tooltip on the grid column. Column tooltip takes precedence over row tooltip (description) set on grid. If both are set, only column tooltip is shown. The closure obtains row object and generates tooltip string. If it returns `null`, the row tooltip common for all columns is show instead.

Parameters:

- *tooltips* - closure obtains row object and generates tooltip
- *contentMode* - defines tooltip formatting

setExpandRatio(expandRatio : Integer) : GridColumn Sets the ratio with which the column expands.

By default, all columns expand equally (treated as if all of them had an expand ratio of 1). Once at least one column gets a defined expand ratio, the implicit expand ratio is removed, and only the defined expand ratios are taken into account.

If a column has a defined width (#setWidth(double)), it overrides this method's effects.

Example: A grid with three columns, with expand ratios 0, 1 and 2, respectively. The column with a **ratio of 0 is exactly as wide as its contents requires**. The column with a ratio of 1 is as wide as it needs, **plus a third of any excess space**, because we have 3 parts total, and this column reserves only one of those. The column with a ratio of 2, is as wide as it needs to be, **plus two thirds** of the excess width.

Parameters:

- *expandRatio* the expand ratio of this column. 0 to not have it expand at all. A null to clear the expand value.

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

setFilterConfig(config : FilterConfig) : void Sets the new filter configuration to the grid.

Parameters:

- *config* filter configuration must be a subtype of forms::FilterConfig.

setFiltrable(filtrable∗ : Boolean) : GridColumn Setting false here will hide the filter component for this column.

Parameters:

- *filtrable* if false, the filter component will be hidden; if true, the filter component will be generated.

setHeader(header : String) : GridColumn Sets the caption of the header. This caption is also used as the hiding toggle caption, unless it is explicitly set via #setHidingToggleCaption(String).

Parameters:

- *caption* the text to show in the caption

Returns:

- the column itself

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

setHeaderAlignment(alignment∗ : Alignment) : void Sets the specified column's header content alignment.

Ignores the vertical aspect of the alignment - only the horizontal aspect is applied.

Parameters:

- *alignment* the desired alignment, not null. Defaults to left align.

setHeaderDescription(tooltip : String) : GridColumn Sets the tooltip on the grid column header. Uses "Preformatted" content mode.

Parameters:

- *tooltip* - tooltip

setHeaderDescription(tooltip∗ : String, contentMode∗ : ContentMode) : GridColumn Sets the tooltip on the grid column header.

Parameters:

- *tooltip* - tooltip
- *contentMode* - defines tooltip formatting

setHeaderStyleName(styleName : String) : void Sets the style name to this column's header.

Parameters:

- *styleName* the style name to add to the column's header

setHidable(hidable∗ : Boolean) : GridColumn Sets whether this column can be hidden by the user. Hidable columns can be hidden and shown via the sidebar menu.

Parameters:

- *hidable* `true` iff the column may be hidable by the user via UI interaction

Returns:

- this column

setHidden(hidden∗ : Boolean) : GridColumn Hides or shows the column. By default columns are visible before explicitly hiding them.

Parameters:

- *hidden* `true` to hide the column, `false` to show

Returns:

- this column

setHidingToggleCaption(hidingToggleCaption : String) : GridColumn Sets the caption of the hiding toggle for this column. Shown in the toggle for this column in the grid's sidebar when the column is hidable.

The default value is `null`, and in that case the column's header caption is used.

NOTE: setting this to empty string might cause the hiding toggle to not render correctly.

Parameters:

- *hidingToggleCaption* the text to show in the column hiding toggle

Returns:

- the column itself

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

setMaximumWidth(pixels : Integer) : GridColumn Sets the maximum width for this column.

This defines the maximum allowed pixel width of the column *when it is set to expand*.

Parameters:

- *pixels* the maximum width. null clears the maximum width.

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

setMinimumWidth(pixels : Integer) : GridColumn Sets the minimum width for this column.

This defines the minimum guaranteed pixel width of the column *when it is set to expand*.

Parameters:

- *pixels* the minimum width in pixels. null clears the minimum width.

Throws:

- *IllegalStateException* if the column is no longer attached to any grid

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

setResizable(resizable∗ : Boolean) : GridColumn Sets whether this column can be resized by the user.

Parameters:

- *resizable* true if this column should be resizable, false otherwise

setSorted(sorted∗ : Boolean) : void Sets whether the specified column is used for grid sort or not. The column will be in ascending order.

Parameters:

- *sorted* true if the grid should be sorted by this column

setSorted(sorted∗ : Boolean, ascending∗ : Boolean) : void Sets whether the specified column is used for grid sort or not and whether it should be ascending or descending.

Parameters:

- *sorted* true if the grid should be sorted by this column
- *ascending* true if the grid should be in ascending order, false otherwise

setWidth(pixelWidth : Integer) : GridColumn Sets the width (in pixels).

This overrides any configuration set by any of #setExpandRatio(int), #setMinimumWidth(double) or #setMaximumWidth(double).

Provide null to set the column width to undefined. An undefined width means the grid is free to resize the column based on the cell contents and available space in the grid.

Parameters:

- *pixelWidth* the new pixel width of the column. null sets the width to undefined.

Returns:

- the column itself

Throws:

- *IllegalStateException* if the column is no longer attached to any grid
- *IllegalArgumentException* thrown if pixel width is less than zero

toString() : String

Forms[READ-ONLY] Utility methods for your UI - you can e.g. Call ui() to obtain instance of this class. You can control the UI with this class.

addBrowserWindowResizeListener(listener∗ : { : void}) : void

detectLocation(opts : PositionOptions, resultCallback∗ : {GeolocationEvent : void}) : void Asks the browser to detect the geoposition location. When the location is detected (or the detection is rejected by the user or any other error happens), the callback is invoked.

Parameters:

- *opts* optional geolocation detection options. If null, defaults are used.
- *resultCallback* invoked when the desired GPS position with desired accuracy is available, or an error occurred.

executeJavascript(javascript : String) : void

getBrowserWindowSize() : Dimension Retrieves the size, in DIPs, of the current browser window.

handleSubmitNavigation(todos : Set<Todo>, navigation : Navigation) : void Handles the submit navigation logic.

Parameters:

- *todos* the newly created todos
- *navigation* the navigation which should be applied if the document/todo navigation rule does not yield a navigation

navigateTo(navigation : Navigation) : void Navigates to given target.

Parameters:

- *navigation* target, may be null - in such case it simply navigates to the root (or default) page, which by default is the Todo List page. To override this, open the LspsUI Java class, the openHomePage() method and implement accordingly.

navigateToDocument(documentType∗ : DocumentType) : void Navigates to given document.

Parameters:

- *documentType* the document type, call the document function to obtain the type. Not null.
- *parameters* optional document parameters

navigateToDocument(documentType∗ : DocumentType, parameters : Map<String, Object>) : void Navigates to given document.

Parameters:

- *documentType* the document type, call the document function to obtain the type. Not null.
- *parameters* optional document parameters

navigateToUrl(url∗ : String) : void Redirects the browser to `http://` link.

Parameters:

- *url* the link

save() : Record Saves the form. The result of this action is either a saved Todo (human::Todo) or a saved Document (human::SavedDocument).

Returns:

- Todo or SavedDocument depending on the context in which the method was invoked.

submit() : void Submits this todo and navigates further.

submit(navigation : Navigation) : void Submits a form (document/todo) and navigates further.

Parameters:

- *navigation* the navigation which will be used if the document/todo does not yield a navigation

submitInternal() : List<Todo>

LinkRenderer extends Renderer Renders a simple link, with the cell value set at its caption.

onClick : {Null : void} Called when the button is clicked. Receives the business data for given row as input. Closure result is ignored.

LinkRenderer() CONSTRUCTOR Creates a new link renderer.

LinkRenderer(listener : {Null : void}) CONSTRUCTOR Creates a new link renderer with the click listener.

Parameters:

- *listener* the listener which will be notified about the click events.

getClickListener() : {Null : void} Gets the listener handling the click event.

setClickListener(listener : {Null : void}) : void Sets the click listener which is called when the link is clicked. The listener receives the business data for given row as input. The closure result is ignored.

Parameters:

- *listener* the closure handling the click event

RendererABSTRACT

ButtonRenderer extends Renderer Renders a simple single button, with the cell value set at its caption.

onClick : {Null : void} Called when the button is clicked. Receives the business data for given row as input. Closure result is ignored.

ButtonRenderer() CONSTRUCTOR Creates a new button renderer.

ButtonRenderer(listener : {Null : void}) CONSTRUCTOR Creates a new button renderer with click listener.

Parameters:

- *listener* the listener which will be notified about the click events.

getClickListener() : {Null : void} Gets the listener handling the click event.

setClickListener(listener : {Null : void}) : void Sets the click listener which is called when the button is clicked. The listener receives the business data for given row as input. The closure result is ignored.

Parameters:

- *listener* the closure handling the click event

ComboBox extends AbstractComboBox Combo box form component

ComboBox() CONSTRUCTOR Creates a combobox with no label and a simple value holder.

ComboBox(caption : String) CONSTRUCTOR Creates a combobox with the given label and a simple value holder.

Parameters:

- *caption* caption of the combo box

ComboBox(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a combobox with the given binding.

Parameters:

- *caption* caption
- *binding* binding which holds the selected value

ComboBox(caption : String, ref∗ : Reference<Object>) CONSTRUCTOR Creates a combobox that references the given variable/field and has the given caption.

Parameters:

- *caption* caption
- *ref* reference; must not be null.

getHint() : String Gets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input.

Returns:

- hint text, initially null.

getPageLength() : Integer Gets the page length (the number of options shown in the suggestion popup). Zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

setHint(hint : String) : void Sets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input. For example "A value in USD, omitting the dollar sign". The hint text is initially null.

Parameters:

- *hint* the new hint text, may be null.

setPageLength(pageLength∗ : Integer) : ComboBox Sets the page length (the number of options shown in the suggestion popup). Setting this to zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

toString() : String

ReferenceBinding A reference binding with the reference to the value for the component.

ref : Reference<Object> reference to the value

ReferenceBinding(reference∗ : Reference<Object>) CONSTRUCTOR Creates a binding which reads and writes the value for the component into given reference.

Parameters:

- *reference* reference value; must not be null.

get() : Object

getBinding() : RecordAndProperty Returns:

- If the reference points to record's property, returns the record and the property. If the reference points to a variable, returns null.

getRecordAndProperty(ref∗ : Reference<Object>) : RecordAndProperty If the reference points to record's property, returns the record and the property. If the reference points to a variable, returns null.

set(value : Object) : void

toString() : String

ObjectBinding Binding with the value for the component. It is mutable.

value : Object The value provided, may be null. Mutated by call to set(Object).

ObjectBinding() CONSTRUCTOR Creates the binding with the initial value set to null.

ObjectBinding(value : Object) CONSTRUCTOR Creates the object binding and sets the value to the given value.

Parameters:

- *value* the initial value; may be null.

get() : Object

getBinding() : RecordAndProperty Returns:

- always null

set(newVal : Object) : void

toString() : String Returns:

- not localizable value of the binding

FormABSTRACT **extends UIDefinition** A default superclass for all forms.

To define an API for your form, just add methods to your form's object.

To add support for listeners, just introduce a method which takes a closure, and call the closure as needed.

To insert the reusable form programatically into layout components, just create the form via the new keyword, then call getWidget() to obtain the FormComponent, and finally insert the FormComponent itself into the layout.

widget : FormComponent

modelingId : String Modeling id of the component. Do not use the field directly, use getModelingId and setModelingId methods instead.

createWidget() : FormComponent Called exactly once, to create the widget contents.

Returns:

- widget the widget, must not be null.

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

getWidget() : FormComponent This method will be called by LSPS, to obtain the component which will be shown on-screen.

onLoad() : void This method is called by framework after the form was restored from saved todo or document.

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

Validatable A component which may contain invalid value.

Does not have support for adding validators, as this interface is also implemented by layout components which have no support for validators. You can only attach validators to subclasses of ComponentWithValue.

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isValid() : Boolean Checks if this component is valid - that is:

1. it has a non-empty value if it is required;
2. the value can be parsed and compatible with target data type.
3. all validators succeed

A component will not call Value.set() if the value is not valid. This causes bindings to be not updated and binding-based validations to work on old values.

Not all Validatables allow validators to be attached. For example component groups do not allow custom validators -they only validates its child components.

Only validators attached to component using `addValidator` method influence the result. Error message set by FormComponent.setErrorMessage method does not influence the result. Data error messages distributed using by `showConstraintViolations` visible from FormComponent.get↩ DataErrorMessages dont influence the result either.

Returns:

- `true` if the component and all its child are valid, i.e. all validators attached to this component and to its descendants succeeded.

Rect x : Integer

y : Integer

width : Integer

height : Integer

AbstractSelect[ABSTRACT] **extends InputComponentWithValue isNullSelectionAllowed() : Boolean**

setNullSelectionAllowed(nullSelectionAllowed∗ : Boolean) : void Allow or disallow empty selection by the user. By default the null selection is allowed.

Parameters:

- *nullSelectionAllowed* whether or not to allow empty selection

setOptions(options∗ : Collection<SelectItem>) : void Sets a constant list of options to be shown in this select component. Choosing an item from options will write SelectItem.value to the underlying Value. By default the select has zero options.

Parameters:

- *options* a collection of options, not null.

setOptions(options∗ : Collection<Object>) : void Sets a constant list of options to be shown in this select component. The caption is computed by calling toString on the object. By default the select has zero options.

Parameters:

- *options* a collection of options, not null.

setOptions(options∗ : Map<Object, String>) : void Sets a constant list of options to be shown in this select component. Choosing an item from the map will write the map's key to the underlying Value. By default the select has zero options.

Parameters:

- *options* a map of options, not null. Maps possible values to captions.

setOptionsDataSource(optionValues∗ : DataSource, caption∗ : ValueProvider) : void Sets the available options for this selector to display. The DataSource provides business objects listed in the selector; choosing an item from options will write it to the underlying Value. By default the selector has zero options.

Parameters:

- *optionValues* provides available options for the selector. The DataSource is not asked to sort items - use DataSource.withSort() for that.
- *caption* provides captions for the options items.

SingleSelectList extends AbstractSelect **SingleSelectList()** CONSTRUCTOR Creates a single-select-list with no label and simple value holder.

SingleSelectList(caption : String) CONSTRUCTOR Creates a single-select-list with given label and simple value holder.

Parameters:

- *caption* the caption

SingleSelectList(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a single-select-list with given label and given binding.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Boolean value.

SingleSelectList(caption : String, ref∗ : Reference<Object>) CONSTRUCTOR Creates a single-select-list with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getRowsCount() : Integer Returns the number of rows to show in the list. If the number of rows is set 0, the actual number of displayed rows is determined implicitly by the adapter.

Returns:

- rows the number of rows to show, 0 or more.

setRowsCount(rows∗ : Integer) : void Sets the number of rows in the editor. If the number of rows is set 0, the actual number of displayed rows is determined implicitly by the adapter.

Parameters:

- *rows* the number of rows to set, 0 or more.

toString() : String

GSDashboard extends FormComponent onWidgetUpdate : {DashboardUpdateEvent : void}

addWidget(component∗ : FormComponent, position : Rect, autoposition∗ : Boolean) : GSDashboardWidget
Creates a new widget and adds it to the dashboard.

Parameters:

- *component* the content component of the widget, not null.
- *position* if not null, the widget will be positioned at exactly this position. Must not be null if autoposition is false. The position is not given in pixels, but in dashboard cells instead. The dashboard has 12 columns and an unlimited number of rows.
- *autoposition* if true, the widget will be positioned automatically at the nearest topmost-leftmost free position in the dashboard. The widget will by default take 1x1 cell, unless position is given - in that case the widget's width and height is taken from the 'position' parameter. If false, the position parameter is required and the widget is positioned as specified by the parameter.

getComponentCount() : Integer Returns the number of child components present in this component. The returned number includes the widgets components and their toolbars

getComponents() : List<FormComponent> Returns the child components as a list. The list contains widgets content and toolbar components as well. If interested in widgets use #getWidgets() instead.

Returns:

- the list of all child components, not null, may be empty.

getDashboardUpdateListener() : {DashboardUpdateEvent : void} Gets the listener handling the DashboardUpdateEvents.

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

getWidgets() : List<GSDashboardWidget> Returns all widgets currently added to this dashboard.

Returns:

- a list of widgets, not null, may be empty.

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isValid() : Boolean Checks if all child groups and child ComponentWithValues are valid. If any invalid child is found, returns false.

refresh() : void Calls refresh() on all nested components with values.

removeWidget(widget∗ : GSDashboardWidget) : void Removes the widget from the dashboard. The widget must no longer be used.

Parameters:

- *widget* the widget to remove. Must be registered to this dashboard. Not null.

setDashboardUpdateListener(listener : {DashboardUpdateEvent : void}) : void Sets the listener handling the DashboardUpdateEvents.

Parameters:

- *listener* the listener to set

setMaximizeWindowTooltip(maximizeTooltip : String, restoreTooltip : String) : void Tooltip for maximize widget button.

Parameters:

- *maximizeTooltip* tooltip when the widget has normal size
- *restoreTooltip* tooltip when the widget is maximized

setMinimizeWindowTooltip(minimizeTooltip∗ : String, restoreTooltip∗ : String) : void Tooltip for minimize widget button.

Parameters:

- *minimizeTooltip* tooltip when the widget has normal size
- *restoreTooltip* tooltip when the widget is minimized

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

toString() : String

GSDashboardWidget owner : GSDashboard

component : FormComponent The component, displayed by the widget.

data : Object Arbitrary data which you can use for any purpose. This field is not used by the Forms framework itself.

modelingId : String

checkNotClosed() : void

close() : void

getCaption() : String

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getOwner() : GSDashboard

getPosition() : Rect

getState() : WidgetState

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

Returns:

isCloseable() : Boolean • whether close button is visible

isClosed() : Boolean

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isMaximized() : Boolean

isMinimized() : Boolean

isRestored() : Boolean

isValid() : Boolean Checks whether the child is valid. Returns false only if the child is not valid. If the component does not have a child or the child does not implement Validatable interface, retuns true.

maximize() : void

minimize() : void

restore() : void

setCaption(caption : String) : void

setCloseable(showCloseButton∗ : Boolean) : void Hides and shows close widget button.

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

setPosition(position∗ : Rect) : void

setState(state∗ : WidgetState) : void

setToolbar(component : FormComponent) : void

toString() : String

Popup extends SingleComponentContainer A popup window. By default the popup is not visible, you need to call setVisible(true), show() or showModal() to show it on screen. By default the popup is not modal, it is resizable, draggable and closable.

You can model a Grid- or Table-based popup, which reveals more data as the popup is resized. To achieve this, simply call the setSizeFull() method on the Popup's content. This will however make the popup to be initially zero-sized; to remedy this, simply call popup's setWidth("50%"); setHeight("50%") before it is shown. This will make the popup to take 50% of browser window's width and height and will center the popup in the browser window.

The Popup descends from a FormComponent, yet it is an error to add the popup to any ComponentGroup: the Popup is not part of the UI component tree, but rather it floats over the components and "starts" its own component hierarchy.

onClose : {PopupCloseEvent : void}

Popup() CONSTRUCTOR Creates a popup with no contents. The popup is initially invisible.

Popup(content : FormComponent) CONSTRUCTOR Creates a popup with given component as its contents. The popup is initially invisible.

Parameters:

- *content* the popup contents, may be null.

getPopupCloseListener() : {PopupCloseEvent : void}

Returns:

- the popup close listener

hide() : Popup Shortcut for setVisible(false)

Returns:

- this

isClosable() : Boolean Returns the closable status of the popup. If a popup is closable, it typically shows an X in the upper right corner. Clicking on the X sends a close event to the server. Setting closable to false will remove the X from the popup and prevent the user from closing the popup.

By default the popup is closable.

Returns:

- true if the popup can be closed by the user.

isDraggable() : Boolean Indicates whether a window can be dragged or not. By default a window is draggable.

Returns:

- true if the window can be dragged by the user

isModal() : Boolean Gets window modality. When a modal window is open, components outside that window cannot be accessed. Keyboard navigation is restricted by blocking the tab key at the top and bottom of the window by activating the tab stop function internally.

By default the popup is not modal.

Returns:

- true if this window is modal.

isResizable() : Boolean true if window is resizable by the end-user, otherwise false. Defaults to true.

Returns:

- true if window is resizable by the end-user, otherwise false.

setClosable(closable∗ : Boolean) : void Sets the closable status for the popup. If a popup is closable it typically shows an X in the upper right corner. Clicking on the X sends a close event to the server. Setting closable to false will remove the X from the popup and prevent the user from closing the popup.

By default the popup is closable.

Parameters:

- *closable* determines if the popup can be closed by the user.

setDraggable(draggable∗ : Boolean) : void Indicates whether a window can be dragged or not. By default a window is draggable.

Parameters:

- *draggable* true if the window can be dragged by the user

setModal(modal∗ : Boolean) : void Sets window modality. When a modal window is open, components outside that window cannot be accessed. Keyboard navigation is restricted by blocking the tab key at the top and bottom of the window by activating the tab stop function internally.

By default the popup is not modal.

Parameters:

- *modal* true if modality is to be turned on

setPopupCloseListener(listener : {PopupCloseEvent : void}) : void Sets the popup close listener which is notified on the popup close event.

Parameters:

- *listener* the listener to set

setResizable(resizable∗ : Boolean) : void true if window is resizable by the end-user, otherwise false. Defaults to true.

Parameters:

- *resizable* true if window is resizable by the end-user, otherwise false.

show() : Popup Shortcut for setVisible(true)

Returns:

- this

showModal() : Popup Shortcut for setModal(true); setVisible(true)

Returns:

- this

toString() : String

TabSheet extends ComponentGroup Adding components to the tab sheet will create tab for every individual component. You can then use the getTab() methods to return the Tab object which further controls the tab - you can set the tab's caption etc.

addTab(contents∗ : FormComponent, caption : String) : Tab Helper method for quickly creating a tab with given caption. If the tab sheet already contains contents, existing tab caption is updated.

Parameters:

- *contents* the tab contents, not null.
- *caption* the new tab caption, may be null.

Returns:

- the tab object which may be further used to control the tab itself.

getSelectedTab() : Tab Returns selected tab. Returns null if the tab sheet has no tabs.

Returns:

- selected tab, may be null.

getTab(child∗ : FormComponent) : Tab Returns a tab for given child component. Fails if there is no such child component.

Parameters:

- *child* the child component of this tab sheet

Returns:

- the tab object which may be further used to control the tab itself. Never null.

getTab(tabIndex∗ : Integer) : Tab Returns a tab for given child component. Fails if there is no such child component.

Parameters:

- *tabIndex* the tab index. Tabs are created in the same order as the child components are added to this tab sheet.

Returns:

- the tab object which may be further used to control the tab itself. Never null.

getTabChangedListener() : {TabChangedEvent : void} Gets the listener invoked when the current tab is changed

Parameters:

- *the* listener or null if none is installed

getTabs() : List<Tab>

removeTab(tab : Tab) : void Removes tab from tab sheet. Fails if there is no such child component.

setSelectedTab(tab∗ : Tab) : void Selects given tab. The tab must belong to this TabSheet. Cannot be null - if the TabSheet has tabs, one must be selected at all times.

Parameters:

- *tab* the tab to select, not null.

setSelectedTab(tabIndex∗ : Integer) : void Selects tab by its position in tab sheet. Indexing starts at 0. Such tab must exist, the method fails if the argument is either negative or too big.

Parameters:

- *tabIndex* index to the tab to select, not null.

setTabChangedListener(listener : {TabChangedEvent : void}) : void Sets the listener invoked when the current tab is changed.

Parameters:

- *listener* the listener, may be null.

toString() : String

Tab owner : TabSheet

modelingId[DEPRECATED] **: String** Modeling id of the component. Do not use the field directly, use get↩ ModelingId and setModelingId methods instead.

data : Object Arbitrary data which you can use for any purpose. This field is not used by the Forms framework itself.

icon : Resource

Tab() [CONSTRUCTOR] Do not construct tabs directly, use TabSheet.addTab instead.

getCaption() : String Returns the caption of this tab.

Returns:

- the caption, may be null for no caption.

getContents() : FormComponent Returns the content component of this tab. Never null.

Returns:

- the content component, not null.

getCustomErrorMessage() : String Gets the tab error message.

Returns:

- the component error message or `null`

getDescription() : String Gets the description for the tab. The description can be used to briefly describe the state of the tab to the user, and is typically shown as a tooltip when hovering over the tab.

Returns:

- the description for the tab

getIndex() : Integer

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getStyleName() : String Gets the user-defined CSS style name of the tab. Built-in style names defined in Vaadin or GWT are not returned.

Returns:

- the style name or of the tab

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

isEnabled() : Boolean Returns the enabled status for the tab. A disabled tab is shown as such in the tab bar and cannot be selected.

Returns:

- true for enabled, false for disabled

isVisible() : Boolean Returns the visible status for the tab. An invisible tab is not shown in the tab bar and cannot be selected. By default the tab is visible.

Returns:

- true for visible, false for hidden

refresh() : void

setCaption(caption : String) : Tab Sets the caption of this tab. The caption is shown in the TabSheet itself.

Parameters:

- *caption* the new caption, may be null.

setCustomErrorMessage(errorMessage : String) : void Sets custom error message to this tab. If the error message is not null, the tab is marked with a red exclamation mark. The error message is visible on mouse over.

Parameters:

- *errorMessage* errorMessage the new error message or `null`

setDescription(description : String) : Tab Sets the description for the tab. The description can be used to briefly describe the state of the tab to the user, and is typically shown as a tooltip when hovering over the tab.

Parameters:

- *description* the new description string for the tab.

setEnabled(enabled∗ : Boolean) : Tab Sets the enabled status for the tab. A disabled tab is shown as such in the tab bar and cannot be selected.

Parameters:

- *enabled* true for enabled, false for disabled

setIcon(icon : Resource) : Tab Sets the icon for the tab.

Parameters:

- *icon* the icon to set

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

setStyleName(styleName : String) : Tab Sets a style name for the tab. The style name will be rendered as a HTML class name, which can be used in a CSS definition.

```
Tab tab = tabsheet.addTab(tabContent, "Tab text");
tab.setStyleName("mystyle");
```

The used style name will be prefixed with " v-tabsheet-tabitemcell-". For example, if you give a tab the style "mystyle", the tab will get a " v-tabsheet-tabitemcell-mystyle" style. You could then style the component with:

```
.v-tabsheet-tabitemcell-mystyle {font-style: italic;}
```

This method will trigger a RepaintRequestEvent on the TabSheet to which the Tab belongs.

Parameters:

- *styleName* the new style to be set for tab

setVisible(visible∗ : Boolean) : Tab Sets the visible status for the tab. An invisible tab is not shown in the tab bar and cannot be selected, selection is changed automatically when there is an attempt to select an invisible tab.

Parameters:

- *visible* true for visible, false for hidden. By default the tab is visible.

toString() : String

MultiSelectList extends AbstractSelect **MultiSelectList()** CONSTRUCTOR Creates a multi select list with no label and simple value holder.

MultiSelectList(caption : String) CONSTRUCTOR Creates a multi select list with given label and simple value holder.

Parameters:

- *caption* the caption

MultiSelectList(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a multi select list with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

MultiSelectList(caption : String, ref∗ : Reference<Set<Object>>) CONSTRUCTOR Creates a multi select list with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getRows() : Integer Returns the number of rows to show in the list. If the number of rows is set 0, the actual number of displayed rows is determined implicitly by the adapter.

Returns:

- rows the number of rows to show, 0 or more.

getValue() : Set<Object>

setBinding(ref : Reference<Set<Object>>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setRows(rows∗ : Integer) : void Sets the number of rows in the editor. If the number of rows is set 0, the actual number of displayed rows is determined implicitly by the adapter.

Parameters:

- *rows* the number of rows to set, 0 or more.

toString() : String

GridLayout extends ComponentGroup **GridLayout()** CONSTRUCTOR Creates a simple 1x1 grid layout with initially no child components.

GridLayout(columns∗ : Integer, rows∗ : Integer) CONSTRUCTOR Creates a empty grid layout of given size.

addComponentToGrid(child∗ : FormComponent, column∗ : Integer, row∗ : Integer) : void Adds a component to the grid, at given column and row, occupying one cell. Fails if the cell is already occupied by some other component. The grid is automatically enlarged, to accomodate the new component.

Parameters:

- *column* zero-based column index, 0 or greater.
- *row* zero-based row index, 0 or greater.

addComponentToGrid(child∗ : FormComponent, column∗ : Integer, row∗ : Integer, columnSpan∗ : Integer, rowSpan∗ Adds a component to the grid, at given column and row. Fails if the cells to be taken are already occupied by some other component. The grid is automatically enlarged, to accomodate the new component.

Parameters:

- *column* zero-based column index, 0 or greater.
- *row* zero-based row index, 0 or greater.
- *columnSpan* number of columns to span over, 1 or greater.
- *rowSpan* number of rows to span over, 1 or greater.

getColumnExpandRatio(columnIndex∗ : Integer) : Decimal Returns the current column expand ratio. See setColumnExpandRatio() for details.

Parameters:

- *columnIndex* zero-based column index, 0 or greater.

Returns:

- ratio the ratio, 0 or greater. Use null or 0 to disable expand for this column. Defaults to null

getColumns() : Integer

getComponent(column∗ : Integer, row∗ : Integer) : FormComponent Gets the Component at given index.

Parameters:

- *column* The column index, starting from 0 for the leftmost column.
- *row* The row index, starting from 0 for the topmost row.

Returns:

- Component in given cell or null if empty

getRowExpandRatio(columnIndex∗ : Integer) : Decimal Returns the current row expand ratio. See setRowExpandRatio() for details.

Parameters:

- *columnIndex* zero-based row index, 0 or greater.

Returns:

- ratio the ratio, 0 or greater. Use null or 0 to disable expand for this column. Defaults to null

getRows() : Integer

setColumnExpandRatio(columnIndex∗ : Integer, ratio : Decimal) : void Sets the expand ratio of given column. Defaults to null.

The expand ratio defines how excess space is distributed among columns. Excess space means space that is left over from components that are not sized relatively. By default, the excess space is distributed evenly.

Note that the component width of the GridLayout must be defined (fixed or relative, as opposed to undefined) for this method to have any effect.

Parameters:

- *columnIndex* zero-based column index, 0 or greater.
- *ratio* the ratio, 0 or greater. Use null or 0 to disable expand for this column.

setColumns(columns∗ : Integer) : void

setRowExpandRatio(columnIndex∗ : Integer, ratio : Decimal) : void Sets the expand ratio of given row. Defaults to null.

Expand ratio defines how excess space is distributed among rows. Excess space means the space left over from components that are not sized relatively. By default, the excess space is distributed evenly.

Note, that height of this GridLayout needs to be defined (fixed or relative, as opposed to undefined height) for this method to have any effect.

Note that checking for relative height for the child components is done on the server so you cannot set a child component to have undefined height on the server and set it to `100%` in CSS. You must set it to `100%` on the server.

Parameters:

- *rowIndex* The row index, starting from 0 for the topmost row.
- *ratio* the ratio, 0 or greater. Use null or 0 to disable expand for this column. Defaults to null

setRows(rows∗ : Integer) : void

toString() : String

Table extends AbstractTable **getDataSource() : DataSource** Returns the current table data source.

Returns:

- the currently set data source. If none has been set yet, this returns null.

Returns:

getNoDataMessage() : String • message displayed when no data are available

setColumnOrder(expectedColumnsOrder : List<TableColumn>) : void Sets a new column order for the table. All columns which are not ordered here will remain in the order they were before as the last columns of table.

Parameters:

- *columnsInOrders* the columns in the order they should be

setColumnOrder(expectedColumnsOrder* : TableColumn...) : void Sets a new column order for the table. All columns which are not ordered here will remain in the order they were before as the last columns of table.

Parameters:

- *columnsInOrders* the columns in the order they should be

setDataSource(dataSource : DataSource) : void Sets a new data source to this table. The table automatically refreshes itself.

Parameters:

- *dataSource* the new data source to be shown. May be null - in such case an empty table will be shown.

Returns:

- this

setNoDataMessage(noDataMessage : String) : void Sets message displayed inside table body when no data are available.

setRowStyleGenerator(rowStyleGenerator : {Null : String}) : void Sets the style generator that is used for generating styles for rows. The generator obtains a row objects and returns style name. The CSS class name that will be added to the row is v-table-row-[style name].

Parameters:

- *styleGenerator* the row style generator to set, or `null` to remove a previously set generator

toString() : String

TableColumn Defines a table column.

To model a column with the "Delete" or "Edit" action link just use the IdentityValueProvider - this way, the link will gain access to the entire Record and can delete it easily.

table : AbstractTable Filled when the column is assigned to a table. The column may be assigned to at most one table.

valueProvider : ValueProvider Extracts column values from rows. Not null.

sortable : Boolean If null or true, allows sorting by this column. Expects the value provider to return Comparable types.

sortValueProvider : ValueProvider Takes row as a parameter and returns value to be used when sorting by column. If it is null, sort will use value provider.

filtrable : Boolean If null or true, allows filtering for this column. The filter is auto-generated based on the value type. The auto-generated filter can be configured by the filter configuration provided by the filterConfig attribute.

filterConfig : FilterConfig Configuration for the auto-generated column filter. If this attribute contains instance of OptionsFilterConfig the filter component is rendered as a combo box or in case of multi-select mode as a multi select list.

filterValueProvider : ValueProvider Takes row as a parameter and returns value to be used when by column filter. If it is null, filter will use value provider.

generator : {Null : FormComponent} Creates a component for given row-extracted data. For every table cell a fresh component will be created. This approach is very flexible, but also quite heavyweight, both on the server and on the browser. If possible, use Grid and Renderers - they are way more lightweight.

If the closure is itself null, the value will simply be rendered as a String. If the closure returns null, nothing will be rendered in that particular cell.

modelingId[DEPRECATED] : String Modeling id of the component. Do not use the field directly, use getModelingId and setModelingId methods instead.

data : Object Arbitrary data which you can use for any purpose. This field is not used by the Forms framework itself.

getAlignment() : Alignment Returns the current column's child alignment, with respect to the column itself. Defaults to left alignment.

Returns:

- the alignment, never null.

getComponentInRow(rowItem : Object) : FormComponent Returns form component rendered in this column in row generated from rowItem. May return null if the row is scrolled out of view or if the column does not have component configured.

Returns:

- cell form component

getExpandRatio() : Decimal Returns the current expand ratio.

Returns:

- the expand ratio, may be null or 0 or higher.

getHeader() : String Returns the column header for this column. The column must be assigned to a table.

Returns:

- header the current header, defaults to null.

getHeaderAlignment() : Alignment Returns the current column's header alignment. Defaults to left alignment.

Returns:

- the header alignment, never null.

getHeaderStyleName() : String Gets the style name applied to the header content of this column.

getModelingId() : String Gets the modeling ID of this component.

Returns:

- the modeling ID

getUniqueModelingId() : String Gets html id of this component as rendered on the page or null if there is no direct representation.

Returns:

- the modeling ID

getWidth() : Decimal Returns the column width in pixels.

Returns:

- the column width, may be null.

isCollapsed() : Boolean Returns whether the specified column is collapsed or not.

Returns:

- the current column collapsedness. By default the column is not collapsed. Not null.

isCollapsible() : Boolean Checks if the given column is collapsible. Note that even if this method returns `true`, the column can only be actually collapsed (via UI or with setColumnCollapsed()) if isColumnCollapsingAllowed() is also true.

Returns:

- true if the column can be collapsed; false otherwise. Never null.

isSortAscending() : Boolean Returns true if the table is sorted in ascending order and false if it is sorted in descending order. Can return anything for unsorted columns.

Returns:

- whether the table is sorted by the specified column

isSorted() : Boolean Returns whether the table is sorted by the specified column

Returns:

- whether the table is sorted by the specified column

setAlignment(alignment∗ : Alignment) : void Sets the specified column's child component alignment inside of the column. In order for this to work, set the child's width to wrap contents (using FormComponent.setWidthWrap() for example), otherwise the child will just fill the column and there will be no space for the alignment flag to move the child.

Ignores the vertical aspect of the alignment - only the horizontal aspect is applied.

Parameters:

- *alignment* the desired alignment, not null. Defaults to left align.

setCollapsed(collapsed∗ : Boolean) : void Sets whether the specified column is collapsed or not. Fails if column collapsing is not allowed. By default the column is not collapsed.

Parameters:

- *collapsed* the desired collapsedness.

setCollapsible(collapsible∗ : Boolean) : void Sets whether the given column is collapsible. Note that collapsible columns can only be actually collapsed (via UI or with setColumnCollapsed()) if isColumnCollapsingAllowed() is true. By default all columns are collapsible.

Parameters:

- *collapsible* true if the column should be collapsible, false otherwise.

setExpandRatio(expandRatio : Decimal) : void Sets the column expand ratio for given column.

Expand ratios can be defined to customize the way how excess space is divided among columns. CustomTable can have excess space if it has its width defined and there is horizontally more space than columns consume naturally. Excess space is the space that is not used by columns with explicit width (see setColumnWidth(Decimal)) or with natural width (no width nor expand ratio).

By default (without expand ratios) the excess space is divided proportionally to columns natural widths.

Only expand ratios of visible columns are used in final calculations.

Column can either have a fixed width or expand ratio. The latter one set is used.

A column with expand ratio is considered to be minimum width by default (if no excess space exists). The minimum width is defined by terminal implementation.

If terminal implementation supports re-sizable columns the column becomes fixed width column if users resizes the column.

Parameters:

- *expandRatio* the expandRatio used to divide excess space for this column. May be null or -1.

setHeader(header : String) : void Sets the column header for this column. The column must be assigned to a table.

Parameters:

- *header* the header to set, may be null.

Sets the specified column's header alignment.

Ignores the vertical aspect of the alignment - only the horizontal aspect is applied.

Parameters:

setHeaderAlignment(alignment∗ : Alignment) : void

- *alignment* the desired alignment, not null, Defaults to left align.

setHeaderStyleName(styleName : String) : void Sets a style name which should be applied to the header content of this column.

Parameters:

- *styleName* the name of the style to apply

setModelingId(modelingId : String) : void Sets the modeling ID of this component.

Parameters:

- *modelingId* - modeling id to be set to component

setSorted(sorted∗ : Boolean) : void Sets whether the specified column is used for table sort or not. The column will be in ascending order.

Parameters:

- *sorted* true if the table should be sorted by this column

setSorted(sorted∗ : Boolean, ascending∗ : Boolean) : void Sets whether the specified column is used for table sort or not and whether it should be ascending or descending.

Parameters:

- *sorted* true if the table should be sorted by this column
- *ascending* true if the table should be in ascending order, false otherwise

setWidth(width : Integer) : void Sets columns width (in pixels). Theme may not necessary respect very small or very big values. Setting width to -1 or null (default) means that theme will make decision of width.

Column can either have a fixed width or expand ratio. The latter one set is used. See setExpandRatio().

Parameters:

- *width* width to be reserved for colunmns content. May be null.

CheckBoxList extends AbstractSelect **CheckBoxList()** CONSTRUCTOR Creates a multi-select checkbox list with no label and simple value holder.

CheckBoxList(caption : String) CONSTRUCTOR Creates a multi-select checkbox list with given label and simple value holder.

Parameters:

- *caption* the caption

CheckBoxList(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a multi-select checkbox list with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

CheckBoxList(caption : String, ref∗ : Reference<Set<Object>>) CONSTRUCTOR Creates a multi-select checkbox list with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getOrientation() : Orientation Returns the current orientation.

Returns:

- the current checkbox list orientation, not null. Default value is vertical.

getValue() : Set<Object>

setBinding(ref∗ : Reference<Set<Object>>) : void Sets the ReferenceBinding being displayed by this component. By default an ObjectBinding with the initial value of null is displayed. The component is automatically refreshed.

Parameters:

- *value* the new value holder.

setOrientation(orientation∗ : Orientation) : void Sets the checkbox list orientation. Default orientation is vertical.

Parameters:

- *orientation* whether to lay out checkboxes horizontally or vertically.

toString() : String

RadioButtonList extends AbstractSelect **RadioButtonList()** CONSTRUCTOR Creates a single-select radiobutton list with no label and simple value holder.

RadioButtonList(caption : String) CONSTRUCTOR Creates a single-select radiobutton list with given label and simple value holder.

Parameters:

- *caption* the caption

RadioButtonList(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a single-select radiobutton list with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

RadioButtonList(caption : String, ref∗ : Reference<Set<Object>>) CONSTRUCTOR Creates a single-select radiobutton list with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getOrientation() : Orientation Returns the current orientation.

Returns:

- the current checkbox list orientation, not null. Default value is vertical.

getValue() : Object

setBinding(ref : Reference<Set<Object>>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setOrientation(orientation∗ : Orientation) : void Sets the radiobutton list orientation. Default orientation is vertical.

Parameters:

- *orientation* whether to lay out checkboxes horizontally or vertically.

toString() : String

TextArea extends AbstractTextArea Shows String values only. Always convertible.

TextArea() CONSTRUCTOR Creates a text area with no label and simple value holder.

TextArea(caption : String) CONSTRUCTOR Creates a text area with given label and simple value holder.

Parameters:

- *caption* the caption

TextArea(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a text area with given label and given binding.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

TextArea(caption : String, ref∗ : Reference<String>) CONSTRUCTOR Creates a text area with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getMaxRows() : Integer Returns the maximum allowed number of rows that the textarea in expandable mode will grow to. Default is null that means that there is no limit for growing.

Returns:

- maximum allowed number of rows for expandable mode

getMinRows() : Integer Returns the minimum allowed number of rows that the textarea in expandable mode will shrink to.Default is null that means that the limit is 1.

Note: the value is ignored if the textarea is not in expandable mode

Returns:

- minimum allowed number of rows for expandable mode

getRowsCount() : Integer Returns the number of rows to shown in the text area when it is in non expandable mode.

Returns:

- rows the number of rows to show, 0 or more.

isAppendExtraRow() : Boolean Should last row always be an empty row?

Returns:

- true if text are in expandable mode adds an extra empty row to the end

Returns:

isExpandable() : Boolean • true if the textarea is in expandable mode, false otherwise

setAppendExtraRow(appendExtraRow∗ : Boolean) : void Should last row always be an empty row? The value is ignored if the textarea is not in expandable mode. The empty row is added only while scroll bars are not visible, e.g. while text area is exanading. There is no additional empty row if text area already reached max rows size.

Parameters:

- *appendExtraRow* true to append extra row, false otherwise

setExpandable(isExpandable∗ : Boolean) : void Switch between expandable and normal mode. Textarea in expandable mode grows/shrinks with content within #getMaxRows() and #getMinRows() boundaries.

Parameters:

- *isExpandable*

setMaxRows(maxRows∗ : Integer) : void Sets the maximum allowed number of rows that the textarea in expandable mode will grow to. Default is null that means that there is no limit for growing.

Note: the value is ignored if the textarea is not in expandable mode

Parameters:

- *maxRows* null or number

setMinRows(minRows∗ : Integer) : void Sets the minimum allowed number of rows that the textarea in expandable mode will shrink to.

Parameters:

- *minRows* null or number

setRowsCount(rows∗ : Integer) : void Sets the number of rows in the text area. If the number of rows is 0, the default number determined by the component is used.

Note: the value is ignored if the textarea is in expandable mode

Parameters:

- *rows* the number of rows to set, 0 or more.

toString() : String

AbstractTextArea[ABSTRACT] **extends InputComponentWithValue** Shows String values only. Always convertible.

getHint() : String Gets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input.

Returns:

- hint text, initially null.

getMaxLength() : Integer Returns the maximum number of characters in the field. null is considered unlimited. Terminal may however have some technical limits.

Returns:

- the maxLength, null if unlimited. null, 0 or greater, never a negative number.

Returns:

getUserText() : String • field value as written and currently visible. Unlike `getValue` method, this returns whatever is in field including invalid values.

getValue() : String Returns:

- field value as stored in the binding. Note that only valid values are stored into binding. If any validator returns error, binding is unchanged and this method retuns value in binding. Otherwise said, if the field shows an error after user input, this method returns old correct value.

Use getUserText() method to acquire text as written by user.

setBinding(ref : Reference<String>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setHint(hint : String) : void Sets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input. For example "A value in USD, omitting the dollar sign". The hint text is initially null.

Parameters:

- *hint* the new hint text, may be null.

setMaxLength(maxLength : Integer) : void Sets the maximum number of characters in the field. Value -1 is considered unlimited. Terminal may however have some technical limits.

Parameters:

- *maxLength* the maxLength to set; setting null or values less than zero will remove any limitation.

CustomDelegateField[ABSTRACT] **extends CustomField** A custom field where there is a single primary field holding the value, and a couple of other components decorating around the component. Think a date field with a date picker button, a decimal field with a label showing the currency, etc. This class automatically delegates all relevant ComponentWithValue's methods to the primary field, you just need to implement the getDelegate() method which returns the primary field itself.

cdfinitialized : Boolean

CustomDelegateField() [CONSTRUCTOR]

addValidator(validator* : {Object : String}) : void

focus() : void

getBinding() : Binding

getDelegate() : InputComponentWithValue Returns the primary field where all data-related calls are delegated to. This method may be called multiple times - it must not create new fields with every call; instead, it must return a single instance. It is recommended to create all UI components in the constructor, store them in private fields and return them here.

Returns:

- the primary field, not null.

inferValidator(tags : Collection<Tag>) : void

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isRequired() : Boolean

isValid() : Boolean

refresh() : void

removeAllValidators() : void

setBinding(value : Binding) : void

setOnChangeListener(listener : {ValueChangeEvent : void}) : void

setRequired(required* : Boolean) : void

CustomField[ABSTRACT] **extends InputComponentWithValue** A ComponentWithValue whose UI content can be constructed by the user, enabling the creation of e.g. form fields by composing Vaadin components. Customization of both the visual presentation and the logic of the field is possible.

It is recommended to create all UI components in the constructor, store them in private fields and return them here.

Subclasses must implement the getContent() method.

It is recommended to override the valueChanged() method to detect the new value and update the inner components accordingly. To set a new value, simply call setValue() on yourself.

initialized : Boolean

CustomField() CONSTRUCTOR

getContents() : FormComponent

refresh() : void

setBinding(value : Binding) : void

setValue(value : Object) : void

valueChanged() : void Invoked when the Value has been changed programatically, either by calling setValue(), setProperty() or refresh(). Does not detect cases when the property value is changed spuriously. The default implementation does nothing.

InputComponentWithValueABSTRACT **extends ComponentWithValue** An editable (and focusable) version of ComponentWithValue. All data-modifying fields inherit from this class.

InputComponentWithValue() CONSTRUCTOR

addValidator(validator∗ : {Null : String}) : void Registers given validator to run on the component value; the validator must return a non-null error message when the value is invalid.

Parameters:

- *validator* the validator in the form of a closure, not null. This closure will receive the new value on input. If the value is invalid, a non-null String containing the error message must be returned. If the value is valid, null must be returned.

focus() : void

getOnChangeListener() : {ValueChangeEvent : void} Gets the currently installed value change listener

Returns:

- the value change event handling closure or null if none is installed

getOnValidValue() : {ValueChangeEvent : void} Gets the currently installed valid value change listener

Returns:

- the valid value change event handling closure or null if none is installed

getTabIndex() : Integer

inferValidator(tags : Collection<Tag>) : void Infers the validator from the binding provided by the Value. Fails if Value.getBinding() returns null.

Parameters:

- *tags* may be null. If not null, given collection of tags are applied

isImmediate() : Boolean Returns the immediate mode of the component.

Certain operations such as adding a value change listener will set the component into immediate mode if #setImmediate(boolean) has not been explicitly called with false.

Returns:

- true if the component is in immediate mode (explicitly or implicitly set), false if the component if not in immediate mode

isRequired() : Boolean Checks if the component is visually marked as required to have a value.

Returns:

- true if true, a non-null value is required to be provided by the user

removeAllValidators() : void Removes all validators from this component, including the inferred ones.

setImmediate(immediate∗ : Boolean) : void Sets the component's immediate mode to the specified status.

Parameters:

- *immediate* the boolean value specifying if the component should be in the immediate mode after the call.

setOnChangeListener(listener : {ValueChangeEvent : void}) : void Sets the value change listener for this component. Only one value change listener may be attached to a component - previous value change listener is automatically unregistered. The component implicitly becomes immediate, unless it was explicitly marked as non-immediate, by calling setImmediate(false).

Parameters:

- *listener* the new listener to set. If null, there will be no value change listener registered.

setOnValidValue(listener : {ValueChangeEvent : void}) : void Sets the valid value change listener for this component. Valid value listener is value was changed and new value is valid (passes all the validators).

Only one value change listener may be attached to a component - previous value change listener is automatically unregistered. The component implicitly becomes immediate, unless it was explicitly marked as non-immediate, by calling setImmediate(false).

Parameters:

- *listener* the new listener to set. If null, there will be no value change listener registered.

setRequired(required∗ : Boolean) : void If set to true, this component will be visually marked as required but no additional validator is appended to the field.

Parameters:

- *required* if true, a non-null value is required to be provided by the user

setTabIndex(tabIndex∗ : Integer) : void

toString() : String

Repeater extends Form The Repeater is basically a horizontal or vertical layout.

vertical : Boolean

dataSource : DataSource

childGenerator : {Object : FormComponent}

startIndex : Integer

pageSize : Integer

container : ComponentGroup

childrenWereCreated : Boolean Prevents multiple initialization when the component is first time rendered. See #refresh method for details

Repeater(childGenerator : {Object : FormComponent}, vertical∗ : Boolean) CONSTRUCTOR Creates a repeater which lays out components for every row object vertically or horizontally. Do not forget to set the data source (which will produce row objects for the Repeater), via the #setDataSource() function.

Parameters:

- *childGenerator* generates form component for every row object. Accepts row object produced by the data source.
- *vertical* if true, the components are nested in a VerticalLayout, if false, the components are nested in HorizontalLayout

createChildren(layout : ComponentGroup) : void

createContainer() : ComponentGroup

createWidget() : FormComponent

getComponentCount() : Integer

getComponents() : List<FormComponent>

getContainer() : ComponentGroup

getDataSource() : DataSource Returns current data source.

getExpandRatio() : Decimal Gets the expand ratio which set to the container this repeater generates.

Returns:

- the expand ratio of the generated container

getModelingId() : String Gets the modeling ID of this component. This modeling id is set to layout container component upon its creation. Note: if the rendered itself apppears on page multiple times, the rendered id is different. Each copy gets unique suffix.

Returns:

- the modeling ID

getPageSize() : Integer Returns page size - how many items are rendered.

getStartIndex() : Integer Returns index of item the repeater starts rendering from.

getUniqueModelingId() : String Gets html id of this component as rendered on the page.

Returns:

- the modeling ID

getWidget() : ComponentGroup

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isValid() : Boolean

refresh() : void Refreshes the repeater - re-polls the data source and re-creates all child components.

setDataSource(ds∗ : DataSource) : Repeater Sets the data source for this repeater. All child components are automatically re-generated.

setExpandRatio(expandRatio : Decimal) : void Sets the expand ratio to the container generated by this repeater.

Parameters:

- *expandRatio* the expand ratio to set

setModelingId(modelingId : String) : void Sets the modeling ID of this component. This modeling id is set to layout container component upon its creation. Note: if the rendered itself apppears on page multiple times, the rendered id might not exactly match configured id. Each html component copy gets unique suffix.

Parameters:

- *modelingId* - modeling id to be set to component

setPageSize(pageSize : Integer) : void If set, the repeater renders pageSize items instead of rendering whole datasource.

setStartIndex(startIndex : Integer) : void If set, the repeater starts rendering from the startIndex-th child. Children at index lower then startIndex are not rendered.

HtmlRenderer extends Renderer Renders given string as a HTML.

Panel extends SingleComponentContainer Panel - a simple single component container. Able to scroll its children component.

Panel() [CONSTRUCTOR]

Panel(caption : String) [CONSTRUCTOR]

Panel(child : FormComponent) [CONSTRUCTOR]

Panel(caption : String, child : FormComponent) [CONSTRUCTOR]

scrollToBottom() : void

scrollToTop() : void

toString() : String

HasChildren All components marked by this interface may have child components.

getComponentCount() : Integer Returns the number of child components present in this component.

getComponents() : List<FormComponent> Returns the child components as a list.

Returns:

- the list of all child components, not null, may be empty.

refresh() : void Calls refresh() on all nested components with values.

SingleComponentContainer[ABSTRACT] **extends FormComponent** This kind of component may have at most one child.

createWidget() : FormComponent If there is no content set to this container by the call to #setContent() and this component instance is attached to the screen for the first time, this method is called once to produce the container contents.

This method is particularly helpful when modeling a form which extends a container component: the PDS will override this method, which will now produce the component designed by the user.

Called exactly once, to create the widget contents.

Returns:

- widget the widget, may be null if you wish the container to have no content. The default implementation just returns null.

getComponentCount() : Integer

getComponents() : List<FormComponent>

getContent() : FormComponent Returns the current content component.

Returns:

- current content component or null if none.

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isValid() : Boolean Checks whether the child is valid. Returns false only if the child is not valid. If the component does not have a child or the child does not implement Validatable interface, retuns true.

refresh() : void Calls refresh() on all nested components with values.

setContent(child : FormComponent) : void Sets the content component of this container. By default the container has no content.

Parameters:

- *child* the new content, may be null if the container should have no content.

Collapsible extends SingleComponentContainer Allows its child to collapse, rendering itself as a thin bar. Upon clicking, the bar is expanded back (uncollapsed) to show the child.

Collapsible() [CONSTRUCTOR]

Collapsible(caption : String) [CONSTRUCTOR]

Collapsible(child : FormComponent) [CONSTRUCTOR]

Collapsible(caption : String, child : FormComponent) [CONSTRUCTOR]

isCollapsed() : Boolean Returns the collapsed state of the component. When the component is collapsed, its content is not shown - instead, only a thin bar with the caption is shown. By default the component is not collapsed.

Returns:

- true if the component is collapsed, false if not.

setCollapsed(collapsed∗ : Boolean) : void Sets the collapsed state of the component. When the component is collapsed, its content is not shown - instead, only a thin bar with the caption is shown. By default the component is not collapsed.

Parameters:

- *collapsed* true if the component is collapsed, false if not.

toString() : String

Image extends FormComponent **Image()** [CONSTRUCTOR] Creates a new Image with initially no image shown. Call setSource() to actually show an image.

Image(caption : String) [CONSTRUCTOR] Creates a new Image with a caption. Initially, no image is shown - call setSource() to actually show an image.

Parameters:

- *caption* the caption to set

Image(resource : DownloadableResource) CONSTRUCTOR

Image(caption : String, resource : DownloadableResource) CONSTRUCTOR

refresh() : void

setSource(resource : DownloadableResource) : void Sets the image resource. The dimensions are assumed if possible.

Parameters:

- *source* the source to set, may be null.

Upload extends FormComponent Allows the user to upload a file.

onUploadResult : {File, String : void} Invoked when a file has been uploaded. In case of success, the File will not be null. If the upload failed, the file will be null and the second parameter will be a non null error message.

Upload() CONSTRUCTOR

getAcceptedMimeTypes() : String Returns the currently accepted mime types. Please see http://www.w3schools.com/tags/att_input_accept.asp for details.

Returns:

- current mime types, may be null.

Returns:

getButtonCaption() : String • String to be rendered into button that fires uploading

getUploadResultListener() : {File, String : void} Gets the listener which listens to the event of uploaded file.

isImmediate() : Boolean Returns the immediate mode of the component.

Certain operations such as adding a value change listener will set the component into immediate mode if #setImmediate(Boolean) has not been explicitly called with false.

Returns:

- true if the component is in immediate mode (explicitly or implicitly set), false if the component if not in immediate mode

isRequired() : Boolean Indicates whether this component was marked as required.

Returns:

- true if the component was marked as required.

refresh() : void

setAcceptedMimeTypes(mimeTypes : String) : void Updates the input accept attribute. This determines which files are visible in Open file dialog by default. Most common mime types are "text/plain" for text files and "application/octet-stream" for binary files or unknown files. Note: this influences the default file selection, but user still can change the Open file dialog filter and submit different kind of file. Server side validation needs to be implemented by modeler.

A list of most common mime-types: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Complete_list_of_MIME_types

Parameters:

- *mimeTypes* the new mime types, for example `audio/*,video/*,image/*`. Pass null to remove the attribute.

setButtonCaption(caption∗ : String) : void In addition to the actual file chooser, upload components have button that starts actual upload progress. This method is used to set text in that button.

The button caption must not be null.

In case the Upload is used in immediate mode using #setImmediate(boolean), the file choose (html input with type "file") is hidden and only the button with this text is shown.

Note the string given is localized by this method and set to the button. HTML formatting is not stripped. Be sure to properly validate your value (including localized version) to your needs.

Parameters:

- *buttonCaption* text for upload components button.

setFieldStyle(style∗ : UploadFieldStyle) : void Sets how token field looks like: combo box or text field. Convenience method, does the same thing as calling setImmediate(true) and setImmediate(false).

setImmediate(immediate∗ : Boolean) : void Sets the component's immediate mode to the specified status. Setting upload component immediate initiates the upload as soon as a file is selected, instead of the common pattern of file selection field and upload button.

Parameters:

- *immediate* the Boolean value specifying if the component should be in the immediate mode after the call.

setRequired(required∗ : Boolean) : void Sets the required flag to the upload component.

Parameters:

- *required* the value of the required flag

setUploadResultListener(listener : {File, String : void}) : void Sets the listener which is notified when a file has been uploaded. In case of success, the parameter File will not be null. If the upload failed, the file will be null and the second parameter will be a non null error message.

Parameters:

- *listener* the listener

ImageRenderer extends Renderer Renders an image. The cell value is expected to be a ThemeResource or an ExternalResource. If the cell value is non of the mentioned resource types then it needs to be converted first to one of the mentioned types. The conversion can be done by providing an ImageValueConverter to the 'converter' field.

onClick : {Null : void} Called when the button is clicked. Receives the business data for given row as input. Closure result is ignored.

converter : ResourceConverter The converter which converts the cell value to an image resource.

ImageRenderer() CONSTRUCTOR

ImageRenderer(converter : ResourceConverter) CONSTRUCTOR

ImageRenderer(listener : {Null : void}) CONSTRUCTOR

ImageRenderer(converter : ResourceConverter, listener : {Null : void}) CONSTRUCTOR

getClickListener() : {Null : void} Gets the listener handling the click event.

getConverter() : ResourceConverter Returns:

- the renderer's converter

setClickListener(listener : {Null : void}) : void Sets the click listener which is called when the button is clicked. The listener receives the business data for given row as input. The closure result is ignored.

Parameters:

- *listener* the closure handling the click event

setConverter(converter : ResourceConverter) : void Sets the converter which converts the cell value to an image resource.

Parameters:

- *converter* the converter to set

Tree extends InputComponentWithValue **Tree()** CONSTRUCTOR Creates a tree with no label and simple value holder.

Tree(caption : String) CONSTRUCTOR Creates a single-select-list with given label and simple value holder.

Parameters:

- *caption* the caption

Tree(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a single-select-list with given label and given binding.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Boolean value.

Tree(caption : String, ref∗ : Reference<Object>) CONSTRUCTOR Creates a single-select-list with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

setExpand(expand : {Object, Integer : Boolean}) : void Sets a closure which is used by the tree to decide whether to expand a tree item. The closure receives item's data and the item's depth, and returns true if the item should be expanded or false otherwise. Root items have the depth of 0, their children have the depth of 1, etc.

Parameters:

- *expand* the expand predicate closure

setOptionsDataSource(optionValues∗ : TreeDataSource, caption : ValueProvider) : void Sets the available options for this Tree to display. The DataSource provides business objects listed in the tree; choosing an item from options will write it to the underlying Value. By default the tree has zero options.

Parameters:

- *optionValues* provides options for the tree.
- *caption* provides captions for the options items.

toString() : String

MapDisplay extends FormComponent **onClick : {MapClickedEvent : void}** a closure handling the map click

onMarkerClick : {MarkerClickedEvent : void}

onMarkerDrag : {MarkerDraggedEvent : void}

addMarker(marker : MapMarker) : void Adds a marker to the map.

Parameters:

- *marker*

clearMarkers() : void Removes all markers from the map.

getClickListener() : {MapClickedEvent : void} Returns:

- the listener handling map clicks

Returns:

getMarkerClickListener() : {MarkerClickedEvent : void} • the listener handling clicks on map markers

getMarkerDragListener() : {MarkerDraggedEvent : void} Returns:

- the listener handling drags of map markers

getZoomLevel() : Integer

refresh() : void

removeMarker(marker : MapMarker) : void Removes the marker from the map.

Parameters:

- *marker* the marker to remove

setCenter(coordinates : GeographicCoordinates) : void Positions the location specified by the coordinates to the map center.

Parameters:

- *coordinates*

setCenter(latitude : Decimal, longitude : Decimal) : void Positions the location specified by [latitude, longitude] to the map center.

Parameters:

- *latitude*
- *longitude*

setClickListener(listener : {MapClickedEvent : void}) : void Sets the listener handling map clicks.

Parameters:

- *listener* the listener to set

setMarkerClickListener(listener : {MarkerClickedEvent : void}) : void Sets the listener handling clicks on map markers.

Parameters:

- *listener* the listener to set

setMarkerDragListener(listener : {MarkerDraggedEvent : void}) : void Sets the listener handling drags of map markers.

setZoomLevel(zoomLevel : Integer) : void Sets the map zoom level.

Parameters:

- *zoomLevel*

MapMarker title : String

popup : String

location : GeographicCoordinates

draggable : Boolean

data : Object business data

MapClickedEvent extends Event location : GeographicCoordinates

MarkerClickedEvent extends Event marker : MapMarker the marker business data

MarkerDraggedEvent extends Event This event indicates that a marker has been dragged. The marker's coordinates is updated to the new location. Its co

marker : MapMarker The marker which has been dragged. Consulting it's location will return the updated value.

TreeTable extends AbstractTable getDataSource() : TreeDataSource Returns the current table data source.

Returns:

- the currently set data source. If none has been set yet, this returns null.

setDataSource(dataSource : TreeDataSource) : void Sets a new data source to this table. The table automatically refreshes itself.

Parameters:

- *dataSource* the new data source to be shown. May be null - in such case an empty table will be shown.

Returns:

- this

setExpand(expand : {Object, Integer : Boolean}) : void Sets a closure which is used by the tree table to decide whether to expand a tree item. The closure receives item's data and the item's depth, and returns true if the item should be expanded or false otherwise. Root items have the depth of 0, their children have the depth of 1, etc.

Parameters:

- *expand* the expand predicate closure

toString() : String

AbstractTable[ABSTRACT] extends FormComponent Table is more heavyweight than the Grid, but also more flexible. Instead of a limited Renderer, you can place a full-blown FormComponent into the cells.

addColumn(column∗ : TableColumn) : TableColumn Adds a column to this table. You can call this before or after the data source has been set.

Parameters:

- *column* the column to add, not null.

Returns:

- the column, for chaining methods.

addColumn(column∗ : TableColumn, header∗ : String) : TableColumn Adds a column to this table. You can call this before or after the data source has been set.

Parameters:

- *column* the column to add, not null.
- *header* the column header, not null.

Returns:

- the column, for chaining methods.

getColumns() : List<TableColumn>

getComponentCount() : Integer Returns the number of components currently rendered by this table.

Returns:

- number of contained components.

getComponents() : List<FormComponent> Returns a list of all components rendered in this table. The order of the returned components is unspecified.

1.) If the datasource contains more rows/pages then is currently visible on page, not yet loaded rows are ignored. 2.) Rows hidden under scroll bar, but still loaded are count as 'rendered'.

Example: if the table contains two columns with modelled component in them, has page size 10 and shows 3 pages, the method will return 2∗3∗10 = 60 components.

Returns:

- a list of contained components

getPageLength() : Integer Returns the current page length. Please see setPageLength() for more information.

Returns:

- the current page length, not null, 0 or greater.

getSelection() : Object Returns the currently selected row object. The row was selected either by user clicking on the grid row, or by calling select() programatically. Initially, there is no object selected.

Returns:

- currently selected row object, null if there is no selection.

isColumnCollapsingAllowed() : Boolean Checks if column collapsing is allowed.

Returns:

- true if columns can be collapsed; false otherwise. Defaults to true. Never null.

isColumnReorderingAllowed() : Boolean Sets whether column reordering is allowed or not - that is, whether the user can drag a column to the new location or not. Defaults to true.

Returns:

- whether column reordering is allowed or not. Never null.

isFilteringEnabled() : Boolean Indicates whether the filtering on the table is enabled or not.

Returns:

- true if the filtering is enabled; otherwise false.

isSelectable() : Boolean

refresh() : void Refreshes the contents of this table - polls the data source for fresh data.

select(rowObject : Object) : void Marks an item as selected. Unselects all other items. The row object must have been provided by the DataSource. Only single-selections are supported at the moment. Calling select(null) clears the selection. Fails if the table is not selectable.

Parameters:

- *rowObject* select the row for this row object. If null, the selection is cleared.

setColumnCollapsingAllowed(collapsingAllowed∗ : Boolean) : void Sets whether column collapsing is allowed or not. By default this value is set to true.

Parameters:

- *collapsingAllowed* specifies whether column collapsing is allowed.

setColumnReorderingAllowed(columnReorderingAllowed∗ : Boolean) : void Sets whether column reordering is allowed or not - that is, whether the user can drag a column to the new location or not. By default this value is set to true.

Parameters:

- *columnReorderingAllowed* specifies whether column reordering is allowed.

setColumnResizeListener(listener : {TableColumn : void}) : void Sets a column resize listener to this table. Replaces any previously set column resize listener. Setting null will remove any previously set column resize listener.

Parameters:

- *listener* the listener which will be notified when a table column resize event occrus

setFilteringEnabled(enabled∗ : Boolean) : void Enables/disables the filtering on this table. Disabling the filtering also clears any currently set filter.

Parameters:

- *enabled* true to enable the filtering; false to disable the filtering.

setPageLength(pageLength∗ : Integer) : void Sets the table page length. This setting influences both the internal row batch fetching size, and the table height:

- the batch size of items being fetched from server to client by the Table JavaScript component. If the page length is set to 0, all data is fetched from the server upfront.
- the client-side height of the table, but only when the height is set to null (wrap content). In such case the table shows exactly page-length rows; other items are accessible via the table scrollbar. If the page length is set to 0, the table polls all data and shows them all, with no scrollbar. In this case, the table parent component (or parent's parent) is responsible for correct scrolling.

It is not possible to limit the table height (that is, wrap at most x rows, shrink horizontally when there is less data).

Use-cases:

- the "slow data source" case - the data source is slow to produce items, and calling it repeatedly would kill the performance. Just set the page length to 0 - this forces the table to load all data, but only once, thus avoiding repeated calls to the data source. In order for the user to be able to scroll the table, you can either set table height explicitly (for example to 300px or 100%) - this way the table itself will show the scrollbar, or you can set the table height to wrap contents and place it inside a panel, or a popup - this way the panel/popup will scroll the table itself.

Parameters:

- *pageLength* the new page length, 0 or greater.

setSelectable(selectable∗ : Boolean) : void Enables or disables single-row selection for table. The table is not selectable by default.

setSelectionChangeListener(listener : {ValueChangeEvent : void}) : void Sets or clears the selection change listener. Use getSelection() to query for the current selection.

PositionOptions Instructs the browser to return only geoposition with e.g. given accuracy or age.

enableHighAccuracy : Boolean The enableHighAccuracy attribute provides a hint that the application would like to receive the best possible results.

This may result in slower response times or increased power consumption. The user might also deny this capability,

or the device might not be able to provide more accurate results than if the flag wasn't specified.

The intended purpose of this attribute is to allow applications to inform the implementation that they

do not require high accuracy geolocation fixes and, therefore, the implementation can avoid using geolocation

providers that consume a significant amount of power (e.g. GPS). This is especially useful for applications running

on battery-powered devices, such as mobile phones.

If the PositionOptions parameter to getCurrentPosition or watchPosition is omitted, the default value used for the enableHighAccuracy attribute is false.

The same default value is used in ECMAScript when the enableHighAccuracy property is omitted.

timeout : Decimal The timeout attribute denotes the maximum length of time (expressed in milliseconds) that is allowed to pass from the call

to Forms.detectLocation() until the corresponding resultCallback is invoked. If the implementation is unable to

successfully acquire a new Geoposition before the given timeout elapses, and no other errors have occurred in this interval,

then the corresponding GeolocationEvent.failure must be set with a GeolocationError object whose code attribute is set to GeolocationError.Timeout.

Note that the time that is spent obtaining the user permission is not included in the period covered by the timeout attribute.

The timeout attribute only applies to the location acquisition operation.

If the PositionOptions parameter to Forms.detectLocation() is omitted,

the default value used for the timeout attribute is infinity. If a negative value is supplied, the timeout value is considered to be 0.

The same default value is used in ECMAScript when the timeout property is omitted.

maximumAge : Decimal The maximumAge attribute indicates that the application is willing to accept a cached position whose age is no greater than the specified time in milliseconds.

If maximumAge is set to 0, the implementation must immediately attempt to acquire a new position object. If an implementation does not have a cached

position available whose age is no greater than the specified maximumAge, then it must acquire a new position object.

If the PositionOptions parameter to Forms.detectLocation() is omitted, the default value used for

the maximumAge attribute is 0. If a negative value is supplied, the maximumAge value is considered to be 0.

The same default value is used in ECMAScript when the maximumAge property is omitted.

Geoposition Contains details received from browsers HTML5 geolocation request. Note that on some devices selected fields may be null.

The geographic coordinate reference system used by the attributes is the World Geodetic System (2d) aka WGS84 aka EPSG:4326.

Use Forms.detectLocation() to obtain instance of this class.

coordinate : GeographicCoordinates The geographic position, not null.

accuracy : Decimal The accuracy of the position information in meters. Not null.

altitude : Decimal The height of the position, specified in meters above the ellipsoid or null if device cannot provide the information.

altitudeAccuracy : Decimal The accuracy of the altitude informations in meters or null.

heading : Decimal Denotes the direction of travel of the hosting device and is specified in degrees, where $0° <= \text{heading} < 360°$, counting clockwise relative to the true north. Null if device don't support it or it is not moving.

speed : Decimal The magnitude of the horizontal component of the hosting device's current velocity and is specified in meters per second. If the implementation cannot provide speed information, the value of this attribute must be null. Otherwise, the value of the speed attribute must be a non-negative real number.

GeolocationEvent[SYSTEM] The GeolocationEvent is fired by the Forms.detectLocation() method after the geographical position of the user has been acquired or when the request for location times out.

position : Geoposition When a geolocation request succeeds, this value will contain a non-null position.

failure : GeolocatorError When a geolocation request fails, this field will contain the error cause.

CalendarItem **caption : String** Caption of the calendar item.

startDate : Date Start date of the calendar item. This attribute is mandatory.

endDate : Date End date of the calendar item. This attribute is mandatory.

description : String

allDay : Boolean If true, the calendar item should be rendered as all-day event.

style : String

data : Object A business data related to this calendar event.

CalendarItemProvider **getItems(startDate : Date, endDate : Date) : List<CalendarItem>** Loads calendar items for the specified date range

Parameters:

- *startDate* the start date of the date range
- *endDate* the end date of the date range

Returns:

- a list of calendar items in the target date range

ListCalendarItemProvider **items : List<CalendarItem>**

ListCalendarItemProvider(items∗ : List<CalendarItem>) CONSTRUCTOR Creates a list based calendar item provider.

Parameters:

- *items* a list of items this provider can provide

getItems(startDate : Date, endDate : Date) : List<CalendarItem>

ClosureCalendarItemProvider **closure : {Date, Date : List<CalendarItem>}**

ClosureCalendarItemProvider(closure∗ : {Date, Date : List<CalendarItem>}) CONSTRUCTOR Creates a new calendar event provider which internally uses the provided closure to obtain calendar entries for a target range

Parameters:

- *closure* the closure providing the calendar events

getItems(startDate : Date, endDate : Date) : List<CalendarItem>

Calendar extends FormComponent **onCreateEvent : {CalendarCreateEvent : void}**

onEditEvent : {CalendarEditEvent : void}

onRescheduleEvent : {CalendarRescheduleEvent : void}

getCreateListener() : {CalendarCreateEvent : void} Gets the listener handling calendar create events.

getEditListener() : {CalendarEditEvent : void} Gets the listener handling calendar edit events.

getRescheduleListener() : {CalendarRescheduleEvent : void} Gets the listener handling calendar reschedule events.

refresh() : void Refreshes the content of the calendar.

setCalendarItemProvider(eventProvider : CalendarItemProvider) : void Sets the calendar item provider to this calendar. The item provider is used to load the calendar items.

Parameters:

- *eventProvider* the event provider to set

setCalendarMode(mode : CalendarMode) : void Sets the calendar mode.

Parameters:

- *mode* the calendar mode

setCreateListener(listener : {CalendarCreateEvent : void}) : void Sets the create listener. The listener is notified when the calendar produces a create event which signals that a new calendar event should be created.

Parameters:

- *listener* the listener to set

setEditListener(listener : {CalendarEditEvent : void}) : void Sets the edit listener. The listener is notified when the calendar produces an edit event which signals that a exiting calendar event should be edited.

Parameters:

- *listener* the listener to set

setInitialDate(initialDate : Date) : void Sets the initial date of the calendar. The initial date is used to determine the date range displayed by the calendar.

Parameters:

- *initialDate* the initial date

setRescheduleListener(listener : {CalendarRescheduleEvent : void}) : void Sets the reschedule listener. The listener is notified when the calendar produces a reschedule event which signals that an existing calendar event should be rescheduled.

Parameters:

- *the* listener to set

CalendarCreateEvent extends Event from : Date

to : Date

allDay : Boolean

CalendarEditEvent extends Event data : Object

CalendarRescheduleEvent extends Event from : Date

to : Date

data : Object

Dimension width : Integer Width in DIPs.

height : Integer Height in DIPs.

OrderedComponentGroup$^{\text{ABSTRACT}}$ **extends ComponentGroup** A group of components which explicitly maintains ordering of the child components.

addComponentAt(child$*$: FormComponent, index$*$: Integer) : void Adds a component into indexed position in this container.

Parameters:

- *child* the component to be added.
- *index* the index of the component position. The components currently in and after the position are shifted forwards.

PasswordField extends AbstractTextArea PasswordField() $^{\text{CONSTRUCTOR}}$ Creates a text field with no label and simple value holder.

PasswordField(caption : String) $^{\text{CONSTRUCTOR}}$ Creates a text field with given label and simple value holder.

Parameters:

- *caption* the caption

PasswordField(caption : String, binding$*$: Binding) $^{\text{CONSTRUCTOR}}$ Creates a text field with given label and given binding.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

PasswordField(caption : String, ref$*$: Reference$<$String$>$) $^{\text{CONSTRUCTOR}}$ Creates a text field with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

toString() : String

ClosureBinding A read-only binding with the value wrapped in a closure that is evaluated every time the get() method is called.

closure : { : Object} Called when get() is called. Never null.

ClosureBinding(closure∗ : { : Object}) CONSTRUCTOR Creates the binding with the value se to the closure.

get() : Object Returns the value of the ClosureBinding.

Returns:

- Binding value

getBinding() : RecordAndProperty Returns null since the binding is not a record and property.

Returns:

- null null return value for ClosureBinding

set(value : Object) : void Returns an error since the value of the ClosureBinding is read-only.

Returns:

- error

DateRenderer extends Renderer **formatPattern : String** The date format pattern. For the format pattern syntax see http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

DateRenderer() CONSTRUCTOR

DateRenderer(formatPattern : String) CONSTRUCTOR

getFormatPattern() : String Returns:

- the format pattern used by this renderer.

setFormatPattern(formatPattern : String) : void Sets the date format pattern. For the format pattern syntax see http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

Parameters:

- *formatPattern* the date format pattern for this renderer

NumberRenderer extends Renderer **formatPattern : String** The number format pattern. For the format pattern syntax see https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html.

NumberRenderer() CONSTRUCTOR

NumberRenderer(formatPattern : String) CONSTRUCTOR

getFormatPattern() : String Returns:

- the format pattern used by this renderer.

setFormatPattern(formatPattern : String) : void Sets the number format pattern. For the format pattern syntax see https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html.

Parameters:

- *formatPattern* the number format pattern for this renderer

EnumerationRenderer extends Renderer Renderes given enumeration value as a string.

Download extends FormComponent Allows to download a resource. It can be rendered as a Button or a Link depending on the style which is set.

Download() CONSTRUCTOR A constructor used for custom components.

Download(caption∗ : String, resource∗ : DownloadableResource, style : DownloadStyle) CONSTRUCTOR Creates a new download componet with the given caption, resource and style.

Parameters:

- *caption* the caption of the download component
- *resource* the downloadable resource
- *style* specifies whether the download component is rendered as a hyperlink (default), or as a button.

refresh() : void

setResource(resource∗ : DownloadableResource) : void

setStyle(style : DownloadStyle) : void

DashboardUpdateEvent extends Event **widget : GSDashboardWidget**

newPosition : Rect

newState : WidgetState

TabChangedEvent extends Event Fired when the selected tab is changed in the TabSheet. This event is fired both by user (when the user clicks through the tabs) and programatically (for example when adding tab to an empty tab sheet).

Event **source : FormComponent**

ClickEvent extends Event **x : Integer**

y : Integer

PopupCloseEvent extends Event **closeButton : Boolean** True if the popup was closed via close button on the popup. False if the popup was closed programmatically or if the app navigated away.

ValueChangeEvent extends Event **oldValue : Object**

ActionLink extends FormComponent A simple action link which is similar to a Button - that is, it has a click listener.

onClick : {ClickEvent : void} Called when the button is clicked. Only one click listener may be attached.

ActionLink() CONSTRUCTOR Creates an action link with an empty caption.

ActionLink(caption∗ : String) CONSTRUCTOR Creates an action link with given caption. Clicking on the link does nothing - change the #onClick closure to register the click listener.

Parameters:

- *caption* the caption shown on the link itself, not null.

ActionLink(caption∗ : String, clickListener∗ : {ClickEvent : void}) CONSTRUCTOR Creates an action link with given caption and given click listener.

Parameters:

- *caption* the caption shown on the action link itself, not null.
- *clickListener* called when the action link is clicked.

focus() : void

getCaptionMode() : CaptionMode Checks whether captions are rendered as HTML

The default is false, i.e. to render that caption as plain text.

Returns:

- true if the captions are rendered as HTML, false if rendered as plain text

Returns:

getClickListener() : {ClickEvent : void}

- the listener handling clicks

getTabIndex() : Integer

refresh() : void

setCaptionMode(captionAsHtml∗ : CaptionMode) : void Sets whether the caption is rendered as HTML.

If set to true, the captions are rendered in the browser as HTML and the developer is responsible for ensuring no harmful HTML is used. If set to false, the caption is rendered in the browser as plain text.

The default is false, i.e. to render that caption as plain text.

Parameters:

- *captionAsHtml* true if the captions are rendered as HTML, false if rendered as plain text

setClickListener(listener : {ClickEvent : void}) : void Sets the listener for the click events.

Parameters:

- *listener* the listener to set

setTabIndex(tabIndex∗ : Integer) : void

toString() : String

Focusable A component which may receive keyboard focus.

focus() : void Sets the focus to this component.

getTabIndex() : Integer Gets the *tabulator index* of the Focusable component.

Returns:

- tab index set for the Focusable component

setTabIndex(tabIndex∗ : Integer) : void Sets the *tabulator index* of the Focusable component. The tab index property is used to specify the order in which the fields are focused when the user presses the Tab key. Components with a defined tab index are focused sequentially first, and then the components with no tab index.

After all focusable user interface components are done, the browser can begin again from the component with the smallest tab index, or it can take the focus out of the page, for example, to the location bar.

If the tab index is not set (is set to zero), the default tab order is used. The order is somewhat browser-dependent, but generally follows the HTML structure of the page.

A negative value means that the component is completely removed from the tabulation order and can not be reached by pressing the Tab key at all.

Parameters:

- *tabIndex* the tab order of this component. Indexes usually start from 1. Zero means that default tab order should be used. A negative value means that the field should not be included in the tabbing sequence.

Button extends FormComponent A simple button.

onClick : {ClickEvent : void} Called when the button is clicked. Only one click listener may be attached.

Button() CONSTRUCTOR Creates a button with no caption. Clicking on the button does nothing - change the #onClick closure to register the click listener.

Button(caption∗ : String) CONSTRUCTOR Creates a button with given caption. Clicking on the button does nothing - change the #onClick closure to register the click listener.

Parameters:

- *caption* the caption shown on the button itself, not null.

Button(caption∗ : String, clickListener∗ : {ClickEvent : void}) CONSTRUCTOR Creates a button with given caption and given click listener.

Parameters:

- *caption* the caption shown on the button itself, not null.
- *clickListener* called when the button is clicked.

focus() : void

getCaptionMode() : CaptionMode Checks whether captions are rendered as HTML

The default is false, i.e. to render that caption as plain text.

Returns:

- true if the captions are rendered as HTML, false if rendered as plain text

Returns:

getClickListener() : {ClickEvent : void}

- the listener handling clicks

getTabIndex() : Integer

refresh() : void

removeClickShortcut() : void Removes the keyboard shortcut previously set with #setClickShortcut(KeyCode, ModifierKey...).

setCaptionMode(captionAsHtml∗ : CaptionMode) : void Sets whether the caption is rendered as HTML. If set to true, the captions are rendered in the browser as HTML and the developer is responsible for ensuring no harmful HTML is used. If set to false, the caption is rendered in the browser as plain text.

The default is false, i.e. to render that caption as plain text.

Parameters:

- *captionAsHtml* true if the captions are rendered as HTML, false if rendered as plain text

setClickListener(listener : {ClickEvent : void}) : void Sets the listener for the click events.

Parameters:

- *listener* the listener to set

setClickShortcut(keyCode : KeyCode, modifiers : ModifierKey...) : void Makes it possible to invoke a click on this button by pressing the given KeyCode and (optional) ModifierKeys. The shortcut is global (bound to the containing Window).

Parameters:

- *keyCode* the keycode for invoking the shortcut
- *modifiers* the (optional) modifiers for invoking the shortcut, null for none

setTabIndex(tabIndex∗ : Integer) : void

toString() : String

Link extends FormComponent The Link component differs to ActionLink in that it does not have a click listener - instead, it can only navigate the user to a particular URL. However, it is able to open the link in a new window.

getCaptionMode() : CaptionMode Checks whether captions are rendered as HTML

The default is false, i.e. to render that caption as plain text.

Returns:

- true if the captions are rendered as HTML, false if rendered as plain text

Returns:

isOpensNewWindow() : Boolean • whether the link will be opened in this tab, new tab/window or whether it depends on navigation object

refresh() : void

setCaptionMode(captionAsHtml∗ : CaptionMode) : void Sets whether the caption is rendered as HTML. If set to true, the captions are rendered in the browser as HTML and the developer is responsible for ensuring no harmful HTML is used. If set to false, the caption is rendered in the browser as plain text.

The default is false, i.e. to render that caption as plain text.

Parameters:

- *captionAsHtml* true if the captions are rendered as HTML, false if rendered as plain text

setNavigation(navigation : Navigation) : void Configures this link to perform given navigation upon clicking.

Parameters:

- *navigation* perform this navigation upon clicking.

setOpensNewWindow(opensNewWindow : Boolean) : void Sets whether the link should open in new tab/window or not. Overrides the `openNewTab` property in navigation object. If the value is:

- `null` - (default) the behavior depends on the value of `openNewTab` property of navigation object
- `true` - the link is opened in tab/window regardless of `openNewTab` property
- `false` - the link is opened in this tab regardless of `openNewTab` property

Parameters:

- *opensNewWindow* specifies whether the link should open in new tab or this window

setUrl(url : String) : void Configures this link to navigate to given URL.

Parameters:

- *url* the URL, should be a full `http://` or `https://` based.

toString() : String

ContextClickEvent extends Event

MenuItem An item in the context menu.

onClick : { : void} The menu item click handler.

caption : String The caption of the menu item.

icon : Resource

htmlClass : String

subMenu : List<MenuItem>

TokenField extends InputComponentWithValue **TokenField()** CONSTRUCTOR Creates a token field with no label and simple value holder.

TokenField(caption : String) CONSTRUCTOR Creates a token field with given label and simple value holder.

Parameters:

- *caption* the caption

TokenField(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a token field with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

TokenField(caption : String, ref∗ : Reference<Collection<Object>>) CONSTRUCTOR Creates a token field with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getValue() : Collection<Object>

isExcludeSelectedTokens() : Boolean Returns:

- `true` if the field shows selected tokens among its suggestions, `false` otherwise.

setBinding(ref : Reference<Collection<Object>>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setExcludeSelectedTokens(excludeSelectedTokens∗ : Boolean) : void Configures whether token field should show selected tokens from the user.

Parameters:

- *excludeSelectedTokens* - if it is `true`, token field will NOT show selected tokens among suggestions. If it is `false`, selected token fields will be suggested.

setFieldStyle(fieldStyle : TokenFieldStyle) : void Sets how token field looks like: combo box or text field.

setOptions(options∗ : Collection<SelectItem>) : void Sets a constant list of options to be shown in this select component. Choosing an item from options will write SelectItem.value to the underlying Value. By default the select has zero options.

Parameters:

- *options* a collection of options, not null.

setOptions(options∗ : Collection<Object>) : void Sets a constant list of options to be shown in this select component. The caption is computed by calling toString on the object. By default the select has zero options.

Parameters:

- *options* a collection of options, not null.

setOptions(options∗ : Map<Object, String>) : void Sets a constant list of options to be shown in this select component. Choosing an item from the map will write the map's key to the underlying Value. By default the select has zero options.

Parameters:

- *options* a map of options, not null. Maps possible values to captions.

setOptionsDataSource(optionValues∗ : DataSource, caption∗ : ValueProvider) : void Sets the available options for this selector to display. The DataSource provides business objects listed in the selector; choosing an item from options will write it to the underlying Value. By default the selector has zero options.

Parameters:

- *optionValues* provides available options for the selector. The DataSource is not asked to sort items - use DataSource.withSort() for that.
- *caption* provides captions for the options items.

toString() : String

RichTextArea extends InputComponentWithValue **RichTextArea()** CONSTRUCTOR Creates a rich text area with no label and simple value holder.

RichTextArea(caption : String) CONSTRUCTOR Creates a rich text area with given label and simple value holder.

Parameters:

- *caption* the caption

RichTextArea(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a rich text area with given label and given binding.

Parameters:

- *caption* the caption
- *binding* the binding, not null.

RichTextArea(caption : String, ref∗ : Reference<String>) CONSTRUCTOR Creates a rich text area with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

Returns:

getUserText() : String • field value as written and currently visible. Unlike `getValue` method, this returns whatever is in field including invalid values.

toString() : String

FilterConfig **filterId : Object** Filters generated from this config will be assigned this id.

OptionsFilterConfig extends FilterConfig A configuration of a filter providing a set of options.

options : List<SelectItem> Options to allow to filter by.

selected : List<SelectItem> Allows to set the pre-selected options to filter by when filter is set from model.

multiselect : Boolean Whether to allow to filter by multiple selected options.

popupRows : Integer Number of visible rows in drop down list. If it is 0 or null, the framework will determine number of rows automatically.

DateFilterConfig extends FilterConfig A configuration for the date filter.

lessThan : Date Less than date.

moreThan : Date More than date.

resolution : DateTimeResolution

formatPattern : String The format pattern as specified by the java.text.SimpleDateFormat. This value is optional.

RegExpFilterConfig extends FilterConfig A configuration for the regular expression filter.

pattern : String The pattern (java.util.regex.Pattern).

SubstringFilterConfig extends FilterConfig A configuration for the substring filter.

substring : String The substring to filter by.

NumericFilterConfig extends FilterConfig A configuration for the numeric filter.

lessThan : Decimal Less than value.

equal : Decimal Equal value.

moreThan : Decimal More than value.

CssLayout extends OrderedComponentGroup

PdfViewer extends FormComponent **PdfViewer()** CONSTRUCTOR Creates an empty viewer which initially shows nothing (null).

refresh() : void

setRestorePdfStateOnLoad(restoreLastPdfState∗ : Boolean) : void By default, pdf viewer remembers last opened page, zoom, scroll position and sidebar position for each opened pdf. When the same pdf is opened second time in the same browser, it is opened at last read page. This method is able to turn on/off this functionality.

Parameters:

- *restoreLastPdfState* - if `false`, pdf will be opened on the first page with default settings regardless of where it was opened last time.

setSource(resource : FileResource) : void Sets the pdf resource.

Parameters:

- *source* the source to set, may be null.

DropZone extends SingleComponentContainer **onUploadResult : {List<File>, List<String> : void}** Invoked when a file has been uploaded. In case of success, the File will not be null. If the upload failed, the file will be null and the second parameter will be a non null error message.

getUploadResultListener() : {List<File>, List<String> : void} Gets the listener which listens to the event of uploaded file.

isDropEnabled() : Boolean

setDropEnabled(enable∗ : Boolean) : void Enables and disables dropzone. If the parameter is `false`, dropzone wont accept any files.

Parameters:

- *enable* `true` it the dropzone should accept files, `false` otherwise

setMaxFiles(maxFiles : Integer) : void Maximum amount of files that have to be dropped into dropzone at once.

Parameters:

- `∗maxFiles∗` dropzone will accept less then this amount of files

setMinFiles(minFiles : Integer) : void Minimum amount of files that have to be dropped into dropzone at once.

Parameters:

- `∗minFiles∗` dropzone will accept less then this amount of files

setUploadResultListener(listener : {List<File>, List<String> : void}) : void Sets the listener which is notified when all dropped files have been uploaded. The parameter `List\<File\>` contains list of all succesfully uploaded files. Failures are collected in the `List\<String\>` parameter.

Parameters:

- *listener* the listener

SearchComboBox extends AbstractComboBox **SearchComboBox()** CONSTRUCTOR Creates a search combobox with no label and simple value holder.

SearchComboBox(caption : String) CONSTRUCTOR Creates a search combobox with given label and simple value holder.

Parameters:

- *caption* the caption

SearchComboBox(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a search combobox with given label and given property.

Parameters:

- *caption* the caption
- *property* the property, not null. Must hold Boolean value.

SearchComboBox(caption : String, ref∗ : Reference<Object>) CONSTRUCTOR Creates a search combobox with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getFilterTimeout() : Integer Search combo-box does not refresh drop down content until user stopped typing for `timeout` milliseconds.

Returns:

- expected pause in typing before the combo-box search for data

getHint() : String Gets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input.

Returns:

- hint text, initially null.

getMinimalFilterLength() : Integer The minimal length the text typed into the combo box must have in order to be used for filtering. Text longer then minFilterLength is used as a filter for drop down options - only options containing typed substring are available.

Shorter text typed into the combo-box is ignored and all options are shown.

Returns:

- minimal filter length

getPageLength() : Integer Gets the page length (the number of options shown in the suggestion popup). Zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

Returns:

isCaseSensitive() : Boolean

- whether the field is in case sensitive or case insensitive mode.

setCaseSensitive(caseSensitive : Boolean) : SearchComboBox Set the filter into either case sensitive or case insensitive mode.

Parameters:

- *caseSensitive*

setFilterTimeout(timeout : Integer) : SearchComboBox Search combo-box does not refresh drop down content until user stopped typing for `timeout` milliseconds.

Parameters:

- *timeout*

setHint(hint : String) : void Sets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input. For example "A value in USD, omitting the dollar sign". The hint text is initially null.

Parameters:

- *hint* the new hint text, may be null.

setMinimalFilterLength(minFilterLength : Integer) : SearchComboBox The minimal length the text typed into the combo box must have in order to be used for filtering. Text longer then minFilterLength is used as a filter for drop down options - only options containing typed substring are available.

Shorter text typed into the combo-box is ignored and all options are shown.

Parameters:

- *minFilterLength* - minimal length of filter

setPageLength(pageLength∗ : Integer) : SearchComboBox Sets the page length (the number of options shown in the suggestion popup). Setting this to zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

toString() : String

MultiselectComboBox extends AbstractSelect **MultiselectComboBox()** CONSTRUCTOR Creates a combobox with no label and simple value holder.

MultiselectComboBox(caption : String) CONSTRUCTOR Creates a combobox with given label and simple value holder.

Parameters:

- *caption* the caption

MultiselectComboBox(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a combobox with given label and given property.

Parameters:

- *caption* the caption
- *property* the property, not null. Must hold Boolean value.

MultiselectComboBox(caption : String, ref∗ : Reference<Object>) CONSTRUCTOR Creates a combobox with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

getHint() : String Gets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input.

Returns:

- hint text, initially null.

getPageLength() : Integer Gets the page length (the number of options shown in the suggestion popup). Zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

getValue() : Set<Object>

setHint(hint : String) : void Sets the hint text - a textual prompt that is displayed when the field would otherwise be empty, to prompt the user for input. For example "A value in USD, omitting the dollar sign". The hint text is initially null.

Parameters:

- *hint* the new hint text, may be null.

setPageLength(pageLength∗ : Integer) : MultiselectComboBox Sets the page length (the number of options shown in the suggestion popup). Setting this to zero disables paging and shows all items. Default value is 10.

Parameters:

- *pageLength* the new page length, 0 or greater.

toString() : String

SplitPanel[ABSTRACT] **extends FormComponent** SplitPanel is a two component container that divides the available space into two areas. The split can be made either vertically where one component is above another or horizontally where they are side-by-side. The components are divided by a splitter bar which can be dragged to adjust their relative sizes.

onSplitPositionChanged : {SplitPositionChanged : void}

getComponentCount() : Integer

getComponents() : List<FormComponent>

getFirstComponent() : FormComponent Returns first component of this split panel. Depending on the direction the first component is either at the top one or on the left.

getMaxSplitPosition() : Decimal Returns the current maximum position of the splitter, in #getMaxSplit↩PositionUnit() units.

Returns:

- the maximum position of the splitter

getMaxSplitPositionUnit() : SizeUnit Returns the unit of the maximum position of the splitter

Returns:

- the unit of the maximum position of the splitter

getMinSplitPosition() : Decimal Returns the current minimum position of the splitter, in #getMinSplit↩PositionUnit() units.

Returns:

- the minimum position of the splitter

getMinSplitPositionUnit() : SizeUnit Returns the unit of the minimum position of the splitter.

Returns:

- the unit of the minimum position of the splitter

getSecondComponent() : FormComponent Returns second component of this split panel. Depending on the direction the second component is either at the bottom one or on the right.

getSplitPosition() : Decimal Returns the current position of the splitter, in #getSplitPositionUnit() units.

Returns:

- position of the splitter

getSplitPositionUnit() : SizeUnit Returns the unit of position of the splitter.

isConvertible() : Boolean[DEPRECATED] Deprecated, returns the same thing as #isValid.

isLocked() : Boolean Is the SplitPanel handle locked (user not allowed to change split position by dragging).

Returns:

- `true` if locked, `false` otherwise.

isSplitPositionReversed() : Boolean Is the split position reversed. By default the split position is measured by the first region, but if split position is reversed the measuring is done by the second region instead.

Returns:

- true if reversed, false otherwise.

isValid() : Boolean Checks whether both child values are valid. Returns false only if one of them is validatable and not valid. If the child does not exist (one side of the panel is empty) or if one child does not implement Validatable interface, retuns true.

isValidChild(child : FormComponent) : Boolean

refresh() : void Calls refresh() on all nested components with values.

setFirstComponent(child : FormComponent) : void Sets the first component of this split panel. Depending on the direction the first component is shown at the top or to the left.

Parameters:

- *c* The component to use as first component

setLocked(locked : Boolean) : void Lock the SplitPanels position, disabling the user from dragging the split handle.

Parameters:

- *locked* Set `true` if locked, `false` otherwise.

setMaxSplitPosition(position : Decimal, unit : SizeUnit) : void Sets the maximum split position to the given position and unit. If the split position is reversed, maximum and minimum are also reversed.

Parameters:

- *pos* the maximum position of the split
- *unit* the unit (from Sizeable) in which the size is given. Allowed units are UNITS_PERCENTAGE and UNITS_PIXELS

setMinSplitPosition(position : Decimal, unit : SizeUnit) : void Sets the minimum split position to the given position and unit. If the split position is reversed, maximum and minimum are also reversed.

Parameters:

- *pos* the minimum position of the split
- *unit* the unit (from Sizeable) in which the size is given. Allowed units are UNITS_PERCENTAGE and UNITS_PIXELS

setSecondComponent(child : FormComponent) : void Sets the second component of this split panel. Depending on the direction the second component is shown at the bottom or to the right.

Parameters:

- *c* The component to use as second component

setSplitPosition(position : Decimal, unit : SizeUnit) : void Moves the position of the splitter with given position and unit.

Parameters:

- *pos* the new size of the first region. Fractions are only allowed when unit is percentage.
- *unit* the unit (from Sizeable) in which the size is given.

setSplitPosition(position : Decimal, unit : SizeUnit, reverse : Boolean) : void Moves the position of the splitter with given position and unit and sets reverse property of the component.

Parameters:

- *pos* the new size of the first region. Fractions are only allowed when unit is percentage.
- *unit* the unit (from Sizeable) in which the size is given.
- *reverse* Puts the component into reverse mode. If set to true the split splitter position is measured by the second region else it is measured by the first region.
  Note: this affects also previously set minSplitPosition and maxSplitPosition properties.

setSplitPositionChangedListener(listener : {SplitPositionChanged : void}) : void Sets the listener for the split position changes. The listener is invoked for both user actions and programmatic actions.

Parameters:

- *listener* the listener to set

HorizontalSplitPanel extends SplitPanel HorizontalSplitPanel is a two component container that divides the available space into two areas. The components are side-by-side - first component is on the left and second component is on the right. They are divided by a splitter bar which can be dragged to adjust their relative sizes.

VerticalSplitPanel extends SplitPanel VerticalSplitPanel is a two component container that divides the available space into two areas. The components are above each other - first component is on the top and second component is on the bottom. They are divided by a splitter bar which can be dragged to adjust their relative sizes.

Accordion extends TabSheet Accordion is vertical tab sheet.

AbstractComboBox^{ABSTRACT} extends AbstractSelect An abstract record that represents a combo-box forms component.

setNewItemHandler(itemProvider : {String : Object}) : void Enables or disables whether the user can enter an undefined option.

The `itemProvider` converts the user input from the text field to an option value. The returned object is set as the combo-box value and can be retrieved with the `getValue` method. It is selected in the combo box but does not appear in the drop-down list with options. If the user unselects the value, the value dissappears from the combo.

The label for the new item is generated by the caption provider supplied via the setOptions(Collection<SelectItem> options*, ValueProvider caption) method. If not supplied, the result item's `toString` method is used.

If the `itemProvider` parameter is `null`, the user can use only a defined option: the value written into the field is used to filter the options.

setOptions(options* : Collection<Object>, caption* : ValueProvider) : void Convenience method that calls #setOptions(Collection<SelectItem> options*, ValueProvider caption).

Parameters:

- *options* values of the `SelectItem` objects
- *caption* generator of labels for new select items

setOptions(options* : Collection<SelectItem>, caption : ValueProvider) : void Sets the options of the select component.

When the user selects an item from the options, the option value is written to the underlying Value.

The caption provider generates captions for those items that are not in the `options` collection. Such items are created either with the `setValue` method from the model or when the user enters a new option value to the select component (this is allowed when the new-item handler was previously defined with the `setNewItemHandler` method).

Parameters:

- *options* collection with the options; must not be null.
- *caption* generator of labels for the select items that are not in the options.

setOptions(options* : Map<Object, String>, captionProvider : ValueProvider) : void Sets the options of the select component.

When the user selects an item from the options, the option value is written to the underlying Value.

The caption provider generates captions for those items that are not in the `options` collection. Such items are created either with the `setValue` method from the model or when the user enters a new option value to the select component (this is allowed when the new-item handler was previously defined with the `setNewItemHandler` method).

Parameters:

- *options* collection of options; must not be null.
- *captionProvider* generator of labels for select items that are not in the options.

SplitPositionChanged extends Event **position : Decimal** New split position.

unit : SizeUnit Unit in which split position is reported.

isReversed : Boolean Returns true if originating split panel is in reverse mode. Reverse mode true means that the split splitter position is measured by the second region, false means it is measured by the first region.

getPosition() : Decimal

getUnit() : SizeUnit

isReversed() : Boolean

BooleanFilterConfig extends FilterConfig A configuration for the boolean filter.

value : Boolean Default value set to filter field.

disableIndeterminateState : Boolean If <code>true</code>, user wont be able to set filter to indeterminate state (when both <code>true</code> and <code>false</code> value passes the filter). It is still possible to set the default filter value to indeterminate.

caption : String The checkbox filter field caption.

Refresheable refresh() : void

Slider extends InputComponentWithValue A component for selecting a numerical value within a range. User moves a slider on a scale by mouse instead of writing textual values.

Slider() CONSTRUCTOR Creates a slider with no label and simple value holder.

Slider(caption : String) CONSTRUCTOR Creates a slider with given label and simple value holder.

Parameters:

- *caption* the caption

Slider(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a slider with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold Boolean value.

Slider(caption : String, ref∗ : Reference<Decimal>) CONSTRUCTOR Creates a slider with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

Returns:

getDecimalPlaces() : Integer • The number of digits after the decimal point.

getMax() : Decimal Returns:

- maximal value on the slider

Returns:

getMin() : Decimal • minimal value on slider

getValue() : Decimal

setDecimalPlaces(resolution∗ : Integer) : void Sets number of digits after the decimal point.

setMax(max∗ : Decimal) : void Set maximal value of the slider.

setMin(min∗ : Decimal) : void Set minimal value of the slider.

setOrientation(orientation∗ : Orientation) : void Sets sliders orientation - either horizontal or vertical.

setProperty(ref∗ : Reference<Decimal>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of ????? is displayed. The component is automatically refreshed.

Parameters:

- *value* the new value holder.

toString() : String

LocalDateField extends InputComponentWithValue **LocalDateField()** CONSTRUCTOR Creates a local date field with no label and simple value holder.

LocalDateField(caption : String) CONSTRUCTOR Creates a local date field with given label and simple value holder.

Parameters:

- *caption* the caption

LocalDateField(caption : String, binding∗ : Binding) CONSTRUCTOR Creates a local date field with given label and given property.

Parameters:

- *caption* the caption
- *binding* the binding, not null. Must hold LocalDate value.

LocalDateField(caption : String, ref∗ : Reference<LocalDate>) CONSTRUCTOR Creates a local date field with given label, referencing given variable/field.

Parameters:

- *caption* the caption
- *ref* the reference not null.

Returns:

getUserText() : String • field value as written and currently visible. Unlike `getValue` method, this returns whatever is in field including invalid or unparseable values.

getValue() : LocalDate

setBinding(ref : Reference<LocalDate>) : void Sets the ReferenceValue being displayed by this component. By default a simple value with the initial value of null is displayed. The component is automatically refreshed. If the reference is null, previous reference is cleared and field is empty.

Parameters:

- *ref* reference to the new value holder.

setDateFormat(dateFormat : String) : void

setDateFormats(dateFormats∗ : List<String>) : void Sets the formats accepted by this local date field. The list may contain a special String constant "date" - this constant is automatically replaced by the application message named "app.dateFormat", via the LspsAppConnector.getApplicationMessage().

For formatting string specification please follow http://docs.oracle.com/javase/6/docs/api/java/tex SimpleDateFormat.html

Parameters:

- *dateFormats* the list of accepted date formats. Must not be null. Must not be empty. First list item format is used to format the value in the field itself. User can however delete the text in the date field and enter a new date in any of these formats.

setParseErrorMessage(parsingErrorMessage : String) : void Sets error message to be used when text in the field can not be parsed into date. If null then default unlocalized error message is used.

toString() : String

LocalDateRenderer extends Renderer **formatPattern : String** The date format pattern. For the format pattern syntax see http://docs.oracle.com/javase/7/docs/api/java/text/Simple↩ DateFormat.html

LocalDateRenderer() CONSTRUCTOR

LocalDateRenderer(formatPattern : String) CONSTRUCTOR

getFormatPattern() : String Returns:

- the format pattern used by this renderer.

setFormatPattern(formatPattern : String) : void Sets the date format pattern. For the format pattern syntax see https://docs.oracle.com/javase/8/docs/api/java/time/format/↩ DateTimeFormatter.html

Only date symbols are allowed. If the format pattern contains time related symbols, e.g. 'a', 'H', 'm' etc, the formatting will cause runtime error.

Parameters:

- *formatPattern* the date format pattern for this renderer

FontAwesome5 Free font awesome 5 icons and styles https://fontawesome.com/icons?d=gallery&m=free.

codepoint : Integer

iconName : String

iconStyle : String

size : String

fixedWidth : Boolean

rotate : String

flipHorizontal : Boolean

flipVertical : Boolean

spin : Boolean

pulse : Boolean

border : Boolean

pull : String

cssClasses : String

inlineStyle : String

FontAwesome5(codepoint : Integer) CONSTRUCTOR

FontAwesome5(iconName : String) CONSTRUCTOR

border() : FontAwesome5 Convenience method for border(true);

border(shouldHaveBorder : Boolean) : FontAwesome5 Adds and removes visual border around icon. Adds and removes "fa-border" css class to html.

flipHorizontal() : FontAwesome5 Convenience method for flipHorizontal(true);

flipHorizontal(shouldFlip : Boolean) : FontAwesome5 Adds and removes horizontal flip. Adds and removes "fa-flip-horizontal" css class to html.

flipVertical() : FontAwesome5 Convenience method for flipVertical(true);

flipVertical(shouldFlip : Boolean) : FontAwesome5 Adds and removes vertical flip. Adds and removes "fa-flip-vertical" css class to html.

pullLeft() : FontAwesome5 The icon will float the to left and text will wrap around it.

pullNone() : FontAwesome5 Removes pull.

pullRight() : FontAwesome5 The icon will float the to right and text will wrap around it.

pulse() : FontAwesome5 Convenience method for pulse(true);

pulse(shouldPulse : Boolean) : FontAwesome5 Turns on and off icon pulse - rotation in 8 steps. Adds and removes "fa-pulse" css class to html.

regular() : FontAwesome5 Regular font awesome icon style. Adds "far" css class to html.

rotate180() : FontAwesome5 Icon will be rotated by 180 degrees clockwise. Adds "fa-rotate-180" css class to html.

rotate270() : FontAwesome5 Icon will be rotated by 270 degrees clockwise. Adds "fa-rotate-270" css class to html.

rotate90() : FontAwesome5 Icon will be rotated by 90 degrees clockwise. Adds "fa-rotate-90" css class to html.

rotateNone() : FontAwesome5 Removes rotation.

setClasses(cssClasses : String) : FontAwesome5 Add custom css classes to the icon.

setIconStyle(customIconStyle : String) : FontAwesome5 Css class for custom font awesome icon style. Lsps contains only regular ("far") and solid ("fas") icon sets. Use this method to supply css class for any other icon set added to project.

setInlineStyle(inlineStyle : String) : FontAwesome5 Add inline styling to the icon.

size10x() : FontAwesome5 Ten times as large font awesome icon. Adds "fa-10x" css class to html.

size2x() : FontAwesome5 Two times as large font awesome icon. Adds "fa-2x" css class to html.

size3x() : FontAwesome5 Three times as large font awesome icon. Adds "fa-3x" css class to html.

size5x() : FontAwesome5 Five times as large font awesome icon. Adds "fa-5x" css class to html.

size7x() : FontAwesome5 Seven times as large font awesome icon. Adds "fa-7x" css class to html.

sizeExtraSmall() : FontAwesome5 Extra small font awesome icon. Adds "fa-xs" css class to html.

sizeLarge() : FontAwesome5 Large font awesome icon. Adds "fa-lg" css class to html.

sizeNone() : FontAwesome5 Removes icon sizing. The icon will have default size.

sizeSmall() : FontAwesome5 Small font awesome icon. Adds "fa-xm" css class to html.

solid() : FontAwesome5 Solid font awesome icon style. Adds "fas" css class to html.

spin() : FontAwesome5 Convenience method for spin(true);

spin(shouldSpin : Boolean) : FontAwesome5 Turns on and off icon spinning. Adds and removes "fa-spin" css class to html.

stackIconsHtml(topIcon : FontAwesome5, innerIcon : FontAwesome5) : String Creates html to render two icons at the same place.

toCsClasses(icon : FontAwesome5) : String

toHtml() : String Creates html to render the icon.

toIconResource() : FontAwesome Creates icon resource to be used in setIcon methods. Due to vaadin limitations,the generated icon will be just a picture. Spinning, inline styles, custom classes etc are not supported. If you need animation, colors or other similar cutomizations, add style to the component and update style sheets to make so.

widthFixed() : FontAwesome5 Convenience method for widthFixed(true);

widthFixed(hasFixedWidth : Boolean) : FontAwesome5 Turn on and off fixed width icon style, useful when icons needs to be vertically aligned. By default, icon width is not fixed.

Adds and removes "fa-fx" css class to html.

ColumnReorderEvent extends Event

SortEvent extends Event

FilterChangedEvent extends Event

MultiFileUpload extends FormComponent refresh() : void

setAllFilesUploadedListener(listener : {AllFileUploadsDone : void}) : void Sets the file upload listener. The listener is notified when all files upload was finished.

Parameters:

- *listener* the listener to set

setFileUploadFailedListener(listener : {FileUploadFailed : void}) : void Sets the file upload failure listener. The listener is notified when a file upload failed.

Parameters:

- *listener* the listener to set

setFileUploadedListener(listener : {FileUploaded : void}) : void Sets the file upload listener. The listener is notified when a single file was uploaded.

Parameters:

- *listener* the listener to set

FileUploaded extends Event file : File

AllFileUploadsDone extends Event files : List<File>

FileUploadFailed extends Event fileName : String

errorMessage : String

ItemClickEvent extends ClickEvent item : Object

ComponentRenderer extends Renderer

MessageEvent extends Event message : String

origin : String

WidgetState Restored

Maximized

Minimized

Closed

Alignment **TopRight**

TopLeft

TopCenter

MiddleRight

MiddleLeft

MiddleCenter

BottomRight

BottomLeft

BottomCenter

GeolocatorError Enumerates all possible causes of a geolocation retrieval failure.

UnknownError An unknown error occurred.

PermissionDenied The user declined access to their position.

PositionUnavailable The browser was unable to locate the user.

Timeout The browser was unable to locate the user in the time specified in the PositionOptions.timeout

UnsupportedInBrowser The browser does not support geolocation retrieval.

CalendarMode **Monthly**

Weekly

Daily

NotificationType Predefined notification types. Each notification type has its distinctive UI look, default screen position and behavior (for example, an Info notification fades away automatically while an Error notification is displayed until the user clicks the notification).

Info Information notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaadi for examples.

By default, notification with this type disappears immediately (that is, it slowly fades in and immediately, with no delay, starts fading out) . The nofication is shown in the middle center part of the screen by default.

Warning Warning notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaac for examples.

By default, notification with this type disappears after 1,5 second and is shown in the middle center part of the screen.

Error Error notification. See https://vaadin.com/blog/-/blogs/user-notifications-with-vaadin for examples.

By default, notification with this type requires user click to disappear and is shown in the middle center part of the screen.

Tray Bottom-right small notification. See https://vaadin.com/blog/-/blogs/user-notifications-with- for examples.

By default, this notification disappears after 3 seconds and is shown in the bottom-right corner.

ContentMode Content modes defining how the label's content should be rendered.

Text Content is only a plain text.

Preformatted Content is a preformatted text. New lines are preserved.

Html Content is rendered as HTML.

SelectionMode **NONE**

SINGLE

MULTI

TokenFieldStyle ComboBox

TextField

SizeUnit Pixels

Percentage

CaptionMode Content modes defining how the component caption should be rendered.

Text Content is only a plain text.

Html Content is rendered as HTML.

KeyCode ENTER

ESCAPE

PAGE_UP

PAGE_DOWN

TAB

ARROW_LEFT

ARROW_UP

ARROW_RIGHT

ARROW_DOWN

BACKSPACE

DELETE

INSERT

END

HOME

F1

F2

F3

F4

F5

F6

F7

F8

F9

F10

F11

F12

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

NUM0

NUM1

NUM2

NUM3

NUM4

NUM5

NUM6

NUM7

NUM8

NUM9

SPACEBAR

DOES_NOT_EXISTS

ModifierKey **SHIFT**

CTRL

ALT

META

UploadFieldStyle **Button**

FileFieldAndButton

7.1.2 Chart

Chart[ABSTRACT] **extends** **FormComponent** Abstract record for charts

onPointClick : {ChartPointClickEvent : void}

getOnPointClick() : {ChartPointClickEvent : void} Returns on-point-click listener closure attached to this record type.

refresh() : void Redraws the chart.

setOnPointClick(onPointClick : {ChartPointClickEvent : void}) : void Attach on-point-click listener closure to this record type.

setPlotOptions(plotOptions∗ : PlotOptions...) : void Sets the default plot options.

Parameters:

- *plotOptions* default plot options

setSubTitle(subTitle : String) : void Sets the chart sub-title

Parameters:

- *subTitle* chart sub-title

setTitle(title : String) : void Sets the chart title

Parameters:

- *title* chart title

PlotOptions[ABSTRACT] **color : Color**

dataLabels : DataLabels

stacked : Boolean

showInLegend : Boolean shows the graph in the graph legend

shadow : Boolean drops the shadow to the graph line

CartesianChart extends Chart Cartesian chart that can render CategoryDataSeries, DecimalDataSeries, List↩ DataSeries, TimeDataSeries as line, bar, bubble, scatter, and line area graphs.

addDataSeries(dataSeries : DataSeries) : void Adds a data series to the Cartesian chart. Once added, the data series cannot be changed with the exception of the addItem(..) method.

Parameters:

- *dataSeries* data series to add

addXAxis(axis : Axis) : void Use the defined X axis in the chart.

addYAxis(axis : Axis) : void Use the define Y axis in the chart.

Parameters:

- *axis* Y-axis configuration

setDataSeries(dataSeriesList : List<DataSeries>) : void Sets the series of the chart: the current series are removed. Once added, the data series cannot be changed with the exception of calling the add↩ Item(..) method.

Parameters:

- *dataSeriesList* list of data series for the chart

Axis chart axis used by the cartesian and gauge charts

Note that ticks define the properties of the units on the axis, such as tick interval (not only the tick marks).

```
new Axis\(

 min -\> 0,

 max -\> 10,

 tickLength -\> 5,

 tickInterval -\> 1,

 tickWidth -\> 2,

 tickColor -\> new Color\(0, 100, 100, 1\),

 tickPosition -\> TickPosition.outside,
```

```
  plotBands -\> \[
    new forms::PlotBand(from -> 180, to -> 240, color -> new Color(255, 51, 102, 0.8)),
    new forms::PlotBand(from -> 130, to -> 180, color -> new Color(255, 255, 102, 1))
  \],
  gridLineWidth -\> 3,
  gridLineColor -\> new Color\(100, 100, 10\)
\);
```

type : AxisType axis type defined as an AxisType literal:

- AxisType.linear: points on the axis are considered part of a numeric axis
- AxisType.category: points on the axis are considered categories
- AxisType.datetime: points on the axis are considered dates

min : Decimal The minimum value of the axis.

max : Decimal The maximum value of the axis.

title : String The title of the axis.

lineColor : Color

opposite : Boolean A flag indicating whether to display the axis on the opposite side of the normal. This is typically used with dual or multiple axes.

reversed : Boolean A flag indicating whether to reverse the axis so that the highest number is closest to the origin.

offset : Integer Distance in pixels from the plot area to the axis line. This is typically used when two or more axes are displayed on the same side of the plot. Defaults to 0.

categories : List<String>

plotBands : List<PlotBand>

tickColor : Color

tickPosition : TickPosition The position of the tick marks on the axis relative to the axis line. Default is 'outside'.

tickInterval : Decimal

tickLength : Integer

tickWidth : Integer

gridLineColor : Color Hexadecimal color value in the form: #RRGGBB, where RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and FF.

gridLineDashStyle : LineStyle The style of the grid line.

gridLineWidth : Integer The width in pixels of the grid line.

labels : ChartLabels The configuration of the axis labels.

PieChart extends Chart Pie chart with one or mulitiple pies; one pie for each PieSliceSeries.

addPieSliceSeries(series∗ : PieSliceSeries) : void Adds a pie slice series to the pie chart. Once add, the series can not be changed: any changes to the series are ignored.

setPieSliceSeries(dataSeriesList∗ : List<PieSliceSeries>) : void Sets a pie slice series to the graph. The previously set series are dropped. Once set, the series can not be changed: any changes to the series are ignored.

GaugeChart extends Chart Gauge chart with a Decimal value

addYAxis(axis : Axis) : void Uses the defined Y axis in the gauge chart.

setEndAngle(angle : Decimal) : void Sets the end angle of the gauge in degrees. 0 is the top, 90 degrees on the right, etc.

Parameters:

- *angle* angle in degrees

setFormat(format : String) : void A format string for the data label. See forms::ChartLabels for more details.

Parameters:

- *format* format string

setFormatter(formatter : String) : void Defines a callback JavaScript function that formats the data label. See forms::ChartLabels for more details.

c.setFormatter("'Value: ' + this.y + ' Series: ' + this.series.name ")

Parameters:

- *formatter* JavaScript function

setStartAngle(angle : Decimal) : void Sets the start angle of the gauge in degrees. 0 is at the top, 90 degrees on the right, etc.

Parameters:

- *angle* angle in degrees

setValue(value : Decimal) : void Sets the value on the gauge chart.

setValueName(name : String) : void Sets the name of the value; For example: "Speed", "Pressure", etc.

PlotOptionsLine extends PlotOptionsScatter **lineWidth : Integer** Width of the graph line in pixels.

lineStyle : LineStyle

spline : Boolean

CategoryDataSeries extends DataSeries Data series with data arranged in categories; it can be plotted in the CartesianChart and its subtypes.

The categories represent the values of the x axis. The x axis value is the index of the item.

When defining the items for the series, you need to define at least the amount for the y axis, or the range values.

On the cartesian chart, the categories are on the x axis and have integer labels by default.

```
//data on population by country:

new CategoryDataSeries \(

  \[

    new CategoryDataSeriesItem("Greenland", 56196, 56239),

    new CategoryDataSeriesItem("China", 1388232693, 194202093),

    new CategoryDataSeriesItem("India", 1342512706, 1358354355),

    new CategoryDataSeriesItem("USA", 326474013, 328857273)

  \]

\);
```

CategoryDataSeries(items∗ : List<CategoryDataSeriesItem>) CONSTRUCTOR Creates a category data series.

Parameters:

- *items* list of items of the data series

CategoryDataSeries(items : CategoryDataSeriesItem...) CONSTRUCTOR Creates a new category data series.

Parameters:

- *items* a list of items this data series should contain

addItem(item : CategoryDataSeriesItem) : void Adds a data series item to the chart.

Parameters:

- *item* item to be added to the series

addItem(item : CategoryDataSeriesItem, updateChart : Boolean, shift : Boolean) : void Adds a data series item to the data series.

Parameters:

- *item* item added to the data series
- *updateChart* whether to update the chart immediately
- *shift* whether to shift the series by removing one item form the beginning

DecimalDataSeries extends DataSeries Data series with data arranged defined by two numbers: the x and y values. Y value can be defined as a range of low and high.

If one of the numbers is time, consider using TimeDataSeries.

```
//Income and percentage with bachelor's degree or higher:

def DecimalDataSeries dds := new DecimalDataSeries\(\[

   new DecimalDataSeriesItem\(8,10000\),

   new DecimalDataSeriesItem\(28, 50000\),

    new DecimalDataSeriesItem(30, 55000),

    new DecimalDataSeriesItem(40, 80000),

    new DecimalDataSeriesItem(50, 120000),

    new DecimalDataSeriesItem(60, 148000),

   new DecimalDataSeriesItem\(70, 185000\),

   new DecimalDataSeriesItem\(78, 200000\)

   \]

\);
```

DecimalDataSeries(items∗ : List<DecimalDataSeriesItem>) CONSTRUCTOR Creates a decimal-data-series item.

Parameters:

- *items* decimal-series items

DecimalDataSeries(items : DecimalDataSeriesItem...) CONSTRUCTOR Creates a new decimal data series item.

Parameters:

- *items* a list of items this data series should contain

addItem(item : DecimalDataSeriesItem) : void Adds a data series item to the chart.

Parameters:

- *item* item to add to the series

addItem(item : DecimalDataSeriesItem, updateChart : Boolean, shift : Boolean) : void Adds a decimal-data-series item to the chart.

Parameters:

- *item* item to add to the series
- *updateChart* whether to update the chart immediately
- *shift* whether to shift the series by removing one item form the beginning

CategoryDataSeriesItem extends DataSeriesItem item of a CategoryDataSeries

CategoryDataSeriesItem(value : Decimal) CONSTRUCTOR Creates a category-data-series item with the y value.

Parameters:

- *value* the y value

CategoryDataSeriesItem(name : String, value : Decimal) CONSTRUCTOR Creates a category-data-series item with the name and the y value.

Parameters:

- *name* category name
- *value* item value

CategoryDataSeriesItem(value : Decimal, payload : Object) CONSTRUCTOR Creates a category-data-series item with the y value and payload.

Parameters:

- *value* item value
- *payload* item payload

CategoryDataSeriesItem(name : String, low : Decimal, high : Decimal) CONSTRUCTOR Creates a category-data-series item with a name, and a range with low and high.

Parameters:

- *name* category name
- *low* low boundary of the range
- *high* high boundary of the range

CategoryDataSeriesItem(name : String, value : Decimal, payload : Object) CONSTRUCTOR Creates a category-data-series item with the name, y value, and payload.

Parameters:

- *name* category name
- *value* item value
- *payload* item's payload

CategoryDataSeriesItem(name : String, low : Decimal, high : Decimal, payload : Object) CONSTRUCTOR Creates a category-data-series item with the name, range with low and high, and payload.

Parameters:

- *name* category name
- *low* low boundary of the range
- *high* high boundary of the range
- *payload* item's payload

DecimalDataSeriesItem extends DataSeriesItem item of a DecimalDataSeries

DecimalDataSeriesItem(x : Decimal, y : Decimal) CONSTRUCTOR Creates a data series item with the x and y values.

Parameters:

- *x* x value
- *y* y value

DecimalDataSeriesItem(x : Decimal, low : Decimal, high : Decimal) CONSTRUCTOR Creates a data series item with the x values, and range with the high and low.

Parameters:

- *x* x value
- *low* low point of the range
- *high* high point of the range

DecimalDataSeriesItem(x : Decimal, y : Decimal, payload : Object) CONSTRUCTOR Creates a data series item with the x and y values, and payload.

Parameters:

- *x* x value
- *y* y value
- *payload* payload

DecimalDataSeriesItem(x : Decimal, low : Decimal, high : Decimal, payload : Object) CONSTRUCTOR Creates a data series item with the x values, payload, and range with the high and low.

Parameters:

- *x* x value
- *low* low point of the range
- *high* high point of the range
- *payload* payload

DataSeriesItemABSTRACT abstract record for items of different data series

Depending on the type of data the item holds, not all properties must define a value.

name : String name of the data series item

x : Decimal x coordinate of the data series item

y : Decimal y coordinate of the data series item

high : Decimal maximum of the range for y axis

low : Decimal minimum of the range for y axis

payload : Object payload with data about the item

colorDEPRECATED **: String** Unused property kept only for backward compatibility. Use itemColor property instead.

dataLabels : DataLabels formating of the data label for the data series item

itemColor : Color

DataSeriesABSTRACT Set of related data items; they are plotted together with common options, for example, as one connected line graph.

chart : CartesianChart

name : String data-series name

items : List<DataSeriesItem> data items comprising the data series

plotOptions : PlotOptions properties for plotting of the data series

range : Boolean whether the data series holds range values

xAxis : Integer Zero-based index of the X-axis this series is connected; this is applicable when using dual or multiple X-axes.

yAxis : Integer Zero based index of the Y-axis this series is connected to when using dual or multiple Y-axes.

addItem(item : DataSeriesItem, updateChart : Boolean, shift : Boolean) : void Adds a data-series item to the chart.

Parameters:

- *item* item to add to the series

- *updateChart* whether to update the chart immediately
- *shift* whether to shift the series by removing one item from the beginning

isRangeItems(items : List<DataSeriesItem>) : Boolean Checks if the data series contains only data-series items with range.

Parameters:

- *items* items to check

Returns:

- true if all items define a range, that is, for each item the methods getLow() and getHigh() return a non-null value.

setPlotOptions(plotOptions : PlotOptions) : void Sets plot options for the data series.

Parameters:

- *plotOptions* plot options

PlotOptionsLineArea extends PlotOptionsLine **fillColor : Color** Hexadecimal color value in the form: #RR↩GGBB, where RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and FF.

fillOpacity : Decimal Fill opacity as a decimal number from interval [0.0, 1.0].

range : Boolean Flag indicating whether the data series should be rendered as range series.

PlotOptionsColumn extends PlotOptions **range : Boolean**

PlotOptionsScatter extends PlotOptions **marker : Marker**

PlotBand A plot band is a colored band stretching across the plot area marking an interval on the axis.

from : Decimal

to : Decimal

label : String The text label for the plot band.

color : Color Hexadecimal color value in the form: #RRGGBB, where RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and FF.

TimeDataSeriesItem extends DataSeriesItem item of a TimeDataSeries

date : Date

TimeDataSeriesItem(date∗ : Date, value : Decimal) CONSTRUCTOR Creates a time data series item.

Parameters:

- *date* the item's date
- *value* the item's value

TimeDataSeriesItem(date∗ : Date, low : Decimal, high : Decimal) CONSTRUCTOR Creates a range time data series item.

Parameters:

- *date* the item's date
- *low* the low value
- *high* the high value

TimeDataSeriesItem(date∗ : Date, value : Decimal, payload : Object) CONSTRUCTOR Creates a time data series item.

Parameters:

- *date* the item's date
- *value* the item's value
- *payload* the item's payload

TimeDataSeriesItem(date∗ : Date, low : Decimal, high : Decimal, payload : Object) CONSTRUCTOR Creates a range time data series item.

Parameters:

- *date* the item's date
- *low* the low value
- *high* the high value
- *payload* the item's payload

TimeDataSeries extends DataSeries Data series with data arranged defined by a point in time and a number.

When using the Date type, make sure to set the x axis to the required type;

('new Axis('type' -> AxisType.datetime)').

```
new TimeDataSeries\(

   \[

     new TimeDataSeriesItem(date(1960), 667.1),

     new TimeDataSeriesItem(date(2016), 1379)

   \]

 \)
```

TimeDataSeries(items∗ : List<TimeDataSeriesItem>) CONSTRUCTOR Creates a new time data series item.

Parameters:

- *items* a list of items this data series should contain

TimeDataSeries(items : TimeDataSeriesItem...) CONSTRUCTOR Creates a new time data series item.

Parameters:

- *items* a list of items this data series should contain

addItem(item : TimeDataSeriesItem) : void Adds a data series item to the chart.

Parameters:

- *item* the item which should be added to the series.

addItem(item : TimeDataSeriesItem, updateChart : Boolean, shift : Boolean) : void Adds a data series item to the chart.

Parameters:

- *item* the item which should be added to the series.
- *updateChart* indicates whether to update the chart immediately
- *shift* indicates whether to shift the series by removing one item form the beginning

PieSlice pie slice of pie slice series

```
new PieSliceSeries\(\"My only series\", \[

         new forms::PieSlice("Big slice", 50, "Business data", new Color("#162955")),

         new forms::PieSlice("Middle slice", 30, "Business data", new Color("#4F628E")),

         new forms::PieSlice("Small slice", 20, "Business data", new Color("#7887ab"))

\]\);
```

name : String name of the pie slice

value : Decimal value of the pie slice

payload : Object payload of the pie slice

color : Color color of the slice defined as #RRGGBB, where RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and FF.

PieSlice(name∗ : String, value∗ : Decimal) CONSTRUCTOR Creates a new instance of the pie slice.

Parameters:

- *name* the name of the slice
- *value* the value of the slice - it's size

PieSlice(name∗ : String, value∗ : Decimal, color : Color) CONSTRUCTOR Creates a new instance of the pie slice.

Parameters:

- *name* the name of the slice
- *value* the value of the slice - it's size
- *color* the color of this pie slice

PieSlice(name∗ : String, value∗ : Decimal, payload : Object) CONSTRUCTOR Creates a new instance of the pie slice.

Parameters:

- *name* the name of the slice
- *value* the value of the slice - it's size
- *payload*

PieSlice(name∗ : String, value∗ : Decimal, payload : Object, color : Color) CONSTRUCTOR Creates a new instance of the pie slice.

Parameters:

- *name* the name of the slice
- *value* the value of the slice - it's size
- *payload*
- *color* the color of this pie slice

PolarChart extends CartesianChart Polar chart that can render CategoryDataSeries, DecimalDataSeries, List↩ DataSeries, TimeDataSeries as line, bar, bubble, scatter, and line area graphs.

PlotOptionsPie **size : AttributeSize** The diameter of the pie. If expressed as a percentage value then the size is relative to the plot area.

innerSize : AttributeSize The size of the inner diameter of the pie. If expressed as a percentage value then the size is relative to the plot area.

borderWidth : Integer The width of the border surrounding each slice.

center : AttributeCenter center of the pie relative to the corner of the plotting area (0,0 is the upper left corner)

```
new PlotOptionsPie\(center -\> new AttributeCenter\(\"20%\", \"50\"\)\);
```

startAngle : Decimal Start angle of the pie slices in degrees where 0 is top and 90 right. Should be smaller than end angle.

endAngle : Decimal The end angle of the pie in degrees where 0 is top and 90 is right. Defaults to start↩ Angle plus 360. Should be larger than start angle.

showInLegend : Boolean A flag indicating whether to show the data series in the graph legend.

dataLabels : PieDataLabels Configurations of the labels rendered for the pie slices.

shadow : Boolean drops the shadow to the graph line

AttributeSize Size attribute in a pixel or percentage value.

```
new PlotOptionsPie\(

  innerSize -\> new AttributeSize\(\"2\"\),

  size -\> new AttributeSize\(\"60%\"\)

\);
```

size : Object a size as a pixel value or a percentage value

AttributeSize(size∗ : Integer) CONSTRUCTOR Creates a size attribute with the value in pixels.

Parameters:

- *size* the size in pixels

AttributeSize(size∗ : String) CONSTRUCTOR Creates a relative size attribute with the value in percent.

Parameters:

- *size* size in percent

AttributeCenter Center coordinates in pixel or percentage values relative to the upper left.

left : Object

top : Object

AttributeCenter(left∗ : Integer, top∗ : Integer) CONSTRUCTOR Creates center coordinates with the defined position.

Parameters:

- *left* distance from the left border in pixels
- *top* distance form the top border in pixels

AttributeCenter(left∗ : String, top∗ : String) CONSTRUCTOR Creates relative center coordinates expressed in percentage.

Parameters:

- *left* the distance from the left border
- *top* the distance form the top border

PieSliceSeries Data series for the PieChart

name : String name of the series

slices : List<PieSlice> list of pie slices

plotOptions : PlotOptionsPie plot options for the pie chart

PieSliceSeries(slices∗ : List<PieSlice>) CONSTRUCTOR Creates a new series for the pie chart.

Parameters:

- *slices* a list of pie slices

PieSliceSeries(slices : PieSlice...) CONSTRUCTOR Creates a new series for the pie chart.

Parameters:

- *slices* a list of pie slices

PieSliceSeries(name : String, slices∗ : List<PieSlice>) CONSTRUCTOR Creates a new series for the pie chart.

Parameters:

- *name* the name of this series
- *slices* a list of pie slices

PieSliceSeries(name : String, slices : PieSlice...) CONSTRUCTOR Creates a new series for the pie chart.

Parameters:

- *name* the name of this series
- *slices* a list of pie slices

Labels A configuration record for labels.

enabled : Boolean enables or disabled the data labels

formatter : String Callback JavaScript function to format the data label. If a format is defined, the format takes precedence and the formatter is ignored.

Available data are:

- this.percentage: Stacked series and pies only. The point's percentage of the total.
- this.point: The point object. The point name, if defined, is available through this.point.name.
- this.series: The series object. The series name is available through this.series.name.
- this.total: Stacked series only. The total value at this point's x value.
- this.x: The x value.
- this.y: The y value.

format : String A format string for the data label. Available variables are the same as for formatter. Defaults to {y}.

rotation : Integer Rotation of labels in degrees.

PieDataLabels extends DataLabels **distance : Integer** The distance of the data label from the pie's edge. Negative numbers put the data label on top of the pie slices. Connectors are only shown for data labels outside the pie.

ListDataSeries extends DataSeries Data series with data with one decimal value.

```
new ListDataSeries\(

  \[

     new ListDataSeriesItem( 1 ),

     new ListDataSeriesItem( 5 )

  \]

\)
```

ListDataSeries(items : List<Decimal>) CONSTRUCTOR A convenience constructor to create a list series directly from values.

ListDataSeries(items : Decimal...) CONSTRUCTOR A utility constructor to create list series directly from the values.

ListDataSeries(items : List<ListDataSeriesItem>) CONSTRUCTOR A constructor accepting the list of items.

ListDataSeries(items : ListDataSeriesItem...) CONSTRUCTOR A constructor accepting the list of items.

addItem(item : ListDataSeriesItem) : void Adds a data series item to the chart.

Parameters:

- *item* item to add to the the list data series

addItem(item : ListDataSeriesItem, updateChart : Boolean, shift : Boolean) : void Adds a list-data-series item to the chart.

Parameters:

- *item* item to add to the series
- *updateChart* whether to update the chart immediately
- *shift* whether to remove one item form the beginning and shift the chart

ListDataSeriesItem extends DataSeriesItem item of a ListDataSeries

ListDataSeriesItem(value : Decimal) CONSTRUCTOR Creates a new list series item for the given value.

Parameters:

- *value* the item value

ListDataSeriesItem(value : Decimal, payload : Object) CONSTRUCTOR Creates a new list series item for the given value and payload.

Parameters:

- *value* the item value
- *payload* the item payload

Color **red : Integer** red component: 0 - 255

green : Integer green component: 0 - 255

blue : Integer blue component: 0 - 255

alpha : Decimal The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Color(color : String) CONSTRUCTOR Creates a new color expressed in the format #RRGGBB

Parameters:

- *color* Hexadecimal color value in the form: #RRGGBB, where RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and FF.

Color(red : Integer, green : Integer, blue : Integer) CONSTRUCTOR Creates a new color expressed in RGB

Parameters:

- *red* the red component
- *green* the green component
- *blue* the blue component

Color(red : Integer, green : Integer, blue : Integer, alpha : Decimal) CONSTRUCTOR Creates a new color expressed in RGBa

Parameters:

- *red* the red component
- *green* the green component
- *blue* the blue component
- *alpha* the alpha component

validateColorComponent(name : String, value : Integer) : void

ChartPointClickEvent extends ClickEvent Click event on a point in a chart

series : String name of the clicked series

key : Object key of the clicked point

value : Object value of the point or, for a range item, the lower value

highValue : Object high value if the clicked item has a range; otherwise null

payload : Object payload of the clicked point

DataLabels extends Labels **color : Color**

ChartLabels extends Labels

PlotOptionsBubble extends PlotOptionsScatter Plot options which causes a data series items to be printed as bubbles. The data series must contain range data series items where the 'low' value is interpreted as the item value and the 'high' value is used to calculate the bubble diameter.

lineWidth : Integer

lineStyle : LineStyle

LineStyle Line style used for drawing line-based charts

solid

dot

dash

dashDot

dashDotDot

Marker Marker shape used for data points

circle

square

diamond

triangle

triangle-down

none

TickPosition position of the tick marks on the axis relative to the axis line

outside

inside

AxisType enumeration used by the Axis.type property

linear

datetime value on the axis are plotted as points in time (if an axis is set to another type, for example, by default,
and your axis plots Date objects, the dates are rendered as integers).

category

7.1.3 Datasource

CompareFilter[READ-ONLY] **extends Filter** Comparable filter. Rejects nulls. Expects Decimal and Date only.

op : Op Filtered value must be "op" than the value present in this filter, in order for the filtered value to pass this filter.

value : Object Only supports comparable types: Date, Decimal, String, Enumeration and Boolean. Not null.

CompareFilter(valueProvider∗ : ValueProvider, operation∗ : Op, value∗ : Object) [CONSTRUCTOR] Compares the row-extracted value against this value.

Parameters:

- *operation* the operation
- *value* the value to compare the row-extracted value against. Not null.

compare(o1 : Object, o2 : Object) : Integer

matches(object : Object) : Boolean

toString() : String

Sort[READ-ONLY] **valueProvider : ValueProvider** Provides values from data rows. The sorting of the data rows will be performed upon the provided values.

ascending : Boolean True if rows with greatest values will be placed last (bottom-most).

Sort(property∗ : Property, ascending∗ : Boolean) [CONSTRUCTOR] Creates the sorting criteria.

Parameters:

- *property* The sorting of the data rows will be performed upon the value of this property.
- *ascending* True if rows with greatest values will be placed last (bottom-most).

Sort(valueProvider∗ : ValueProvider, ascending∗ : Boolean) [CONSTRUCTOR] Creates the sorting criteria.

Parameters:

- *valueProvider* Provides values from data rows. The sorting of the data rows will be performed upon the provided values.
- *ascending* True if rows with greatest values will be placed last (bottom-most).

asc(property∗ : Property) : Sort

asc(valueProvider∗ : ValueProvider) : Sort

compare(row1 : Object, row2 : Object) : Integer Compares two row objects according to this sort specification.

Parameters:

- *row1* first row object
- *row2* second row object

Returns:

- negative if row1 should appear before row2, positive if row1 should appear after row2, 0 if row1 and row2 seems equal (at least from the limited perspective of this sort spec)

compareObjects(o1 : Object, o2 : Object) : Integer

desc(property∗ : Property) : Sort

desc(valueProvider∗ : ValueProvider) : Sort

sort(set∗ : Set<E>) : Set<E> Returns a new set, containing all data rows from the old set but sorted.

Parameters:

- *set* the set to sort, not null.

Returns:

- a new set with items sorted according to this sort specification.

sort(set∗ : List<E>) : List<E> Returns a new set, containing all data rows from the old set but sorted.

Parameters:

- *set* the set to sort, not null.

Returns:

- a new set with items sorted according to this sort specification.

toString() : String

MultiValueFilter[READ-ONLY] **extends Filter** Similar to ValueFilter but accepts multiple values. Expects Date, Decimal, EnumerationImpl and Boolean only.

values : Set<Object>

MultiValueFilter(valueProvider∗ : ValueProvider, vals∗ : Set<Object>) [CONSTRUCTOR] Creates a filter which only matches rows having row-extracted value equal to one of provided values. Think of this filter as a more generic version of ValueFilter. This filter accepts any row-extracted values: not only comparables, but also records, closures, anything.

Parameters:

- *vals* only accept rows with row-extracted value equal to one of these values. Must not be null but may contain null. Empty set rejects all values.

matches(object : Object) : Boolean

toString() : String

RegexpFilter[READ-ONLY] **extends Filter** Only works for String objects - throws exception for any other object type. Matches given string against given regular expression.

regexp : String

RegexpFilter(valueProvider∗ : ValueProvider, regex∗ : String) [CONSTRUCTOR] Compares the row-extracted value against given regexp. The value must be a String otherwise the filter fails at runtime.

Parameters:

- *regex* the Java regex, not null.

matches(object : Object) : Boolean

matches(str : String, regexp : String) : Boolean

toString() : String

ValueFilter[READ-ONLY] **extends Filter** Allows only one value. Expects Date, Decimal, EnumerationImpl and Boolean only.

value : Object The expected value, may be null.

ValueFilter(valueProvider∗ : ValueProvider, value : Object) CONSTRUCTOR Creates a filter which only matches rows having row-extracted value equal to this value. This filter accepts any row-extracted values: not only comparables, but also records, closures, anything.

Parameters:

- *value* only accept rows with this row-extracted value. May be null.

matches(object : Object) : Boolean

toString() : String

DataSource Provides paged rows from a storage, for example from a database.

getCount(filters : Collection<Filter>) : Integer Returns the total number of lines, matching given filters.

Parameters:

- *filters* a collection of filters to apply to the data itself. May be null if no filters are to be applied.

Returns:

- the total number of records which match given criteria. Not null, must be 0 or greater.

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object> Retrieves data from the data source. The data is first filtered and sorted, then paged. An exception will be thrown if an unsupported Filter or Sort is passed.

Parameters:

- *startIndex* paging: starting index, or offset, of the (sorted and filtered) data.
- *count* paging: number of (sorted and filtered) rows to return
- *filters* a collection of filters to apply to the data itself. May be null if no filters are to be applied.
- *sortSpecs* an (ordered) set of sort specs to apply. May be null.

Returns:

- a list of matched objects, most often Records. Must be 0 or greater, should not be more than stated by the count parameter. Must not be null. Should be ordered and filtered as specified by filters and/or sortSpecs.

supportsFilter(filter∗ : Filter) : Boolean Checks if this filter is supported by this datasource.

Parameters:

- *filter* the filter in question, not null.

Returns:

- true if this data source supports given filter, false if not. Must not return null.

supportsSort(sort∗ : Sort) : Boolean Checks if this sorting criteria is supported by this datasource.

Parameters:

- *sort* the sort criteria, not null.

Returns:

- true if this data source supports given sorting criteria, false if not. Must not return null.

SubstringFilterREAD-ONLY **extends Filter** Checks that the value is a String which contains given substring. The filter only works for String objects - throws exception for any other object type.

value : String All passed values must contain given string as its substring. Case-insensitive.

SubstringFilter(valueProvider∗ : ValueProvider, value∗ : String) CONSTRUCTOR Creates a filter which only matches rows having row-extracted string value containing given string. This filter accepts any row-extracted values and converts them to String.

Parameters:

- *value* only accept rows which row-extracted string value contains this string, case-insensitive. Not null.

matches(object : Object) : Boolean

toString() : String

FilterABSTRACTREAD-ONLY Filters Grid data row object. First, a value is obtained from the row (using the valueProvider), then it is checked if the value matches this filter. Only a small subset of filters is supported - the objective here is that these filters will be able to be run on the database, thus avoiding in-memory filtering.

In-memory filtering is expensive because it requires us to pull all rows from the database.

valueProvider : ValueProvider Provides the value from the data row. The filter further operates on the value and either rejects it, or accepts it.

id : Object Filter id assigned in <code>FilterConfig</code>. Used to identify all filters configured with the same <code>FilterConfig</code>.

Filter(valueProvider∗ : ValueProvider) CONSTRUCTOR

matches(object : Object) : Boolean Tests if this filter matches given object.

Parameters:

- *object* the object, may be null. This is not a row object per se - instead, this object is a product of #valueProvider run over given row object.

ValueProvider Used to provide a particular value from the data row, for example provides Person.name out of a Person.

get(row : Object) : Object Receives a row data object on input and provides a value from it (a value of a particular field for example), which will later be sorted and/or filtered.

getPropertyPath() : PropertyPath If this provider provides a value of a property path, return this path. Otherwise, return null. This may be for example used by the database, to efficiently perform filtering and/or sorting (using the native SQL WHERE). If this is null, the sorting/filtering can not be performed in the database and will be performed in-memory, massively decreasing performance.

getValueType() : Type<Object> Returns the type of the value produced by the get(Object) function. If the value type cannot be detected properly, just return Type(Object).

ClosureValueProvider A closure value provider - simply polls provided closure for data, in the get() method. When used in Grid, this provider forces the Grid to sort/filter in-memory, massively degrading performance for large data sets.

closure : {Null : Object} This closure is simply called by the get() method.

ClosureValueProvider(closure∗ : {Null : Object}) CONSTRUCTOR Runs the table row through given closure.

Parameters:

- *closure* the closure

callClosure(c∗ : {Null : Object}, input : Object) : Object

get(input : Object) : Object

getPropertyPath() : PropertyPath Closure has no property path, always returns null.

Returns:

- always null

getReturnType(closure∗ : {Null : Object}) : Type<Object>

getValueType() : Type<Object>

toString() : String

ConstantValueProvider Always provides given value. When used in Grid, this provider forces the Grid to sort/filter in-memory, massively degrading performance for large data sets.

value : Object

ConstantValueProvider(value : Object) CONSTRUCTOR Always provide given value.

Parameters:

- *val* the value to provide.

Returns:

get(input : Object) : Object • Always #value

getPropertyPath() : PropertyPath Returns:

- always null.

getValueType() : Type<Object>

toString() : String

PropertyPathValueProvider Runs given PropertyPath over data row records and provides the result. When used in Grid, this provider is able to issue native SQL where/order, massively improving performance.

path : PropertyPath

PropertyPathValueProvider(path∗ : PropertyPath) CONSTRUCTOR De-references given property path on a record and returns its value. Only accepts Record rows. See forms::PropertyPath for more information.

Parameters:

- *path* the record property, not null.

get(row : Object) : Object

getPropertyPath() : PropertyPath Returns:

- the property path given to the constructor, never null

getValueType() : Type<Object>

toString() : String

ForcedSortDataSource extends DataSourceDelegate Delegates calls to the underlying DS but always prepends given set of sort criteria, thus sorting primarily by this criteria. Do not instantiate directly - use the DataSource.withSort() method instead.

additionalSort : Set<Sort>

ForcedSortDataSource(delegate∗ : DataSource, sort∗ : Set<Sort>) CONSTRUCTOR Delegates calls to given delegate, but always applies given set of sort criteria.

Parameters:

- *delegate* delegate data retrieval calls here, not null.
- *sort* always apply this set of sort criteria after any additional sort criteria provided to the getData() function. Not null; may be empty but it makes no sense.

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

toString() : String

DataSourceDelegateABSTRACT Simply delegates all calls to the delegate.

delegate : DataSource

DataSourceDelegate(delegate∗ : DataSource) CONSTRUCTOR Simply delegates all calls to given data-source. Extend this class to implement more interesting implementations.

Parameters:

- *delegate* delegates all calls here, not null.

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

supportsFilter(filter∗ : Filter) : Boolean

supportsSort(sort∗ : Sort) : Boolean

toString() : String

ForcedFilterDataSource extends DataSourceDelegate Delegates calls to the underlying DS but always applies given set of filter criteria, along with any other filters provided in getCount()/getData(). Do not instantiate directly - use the DataSource.withFilter() method instead.

additionalFilters : Collection<Filter> Always applies this set of filter criteria, along with any other filters provided in getCount()/getData().

ForcedFilterDataSource(delegate∗ : DataSource, filters∗ : Collection<Filter>) CONSTRUCTOR Delegates calls to given delegate, but always applies given set of filter.

Parameters:

- *delegate* delegate data retrieval calls here, not null.
- *filters* always apply this set of filters on top of any additional filters provided to getCount()/getData() functions. Not null; may be empty but it makes no sense.

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

toString() : String

TypeDataSource A DataSource which provides all instances of given Record.

recordType : Type<Record>

TypeDataSource(t∗ : Type<Record>) CONSTRUCTOR

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

supportsFilter(filter∗ : Filter) : Boolean

supportsSort(sort∗ : Sort) : Boolean

toString() : String

CollectionDataSource In-memory sorts and filters given collection.

Has very low performance when compared to TypeDataSource; when using Shared Records, please consider using other data sources which perform native in-DB data sorting and filtering.

list : List<Object> Provides contents of this list, filtered and sorted as necessary.

CollectionDataSource(values∗ : Collection<Object>) CONSTRUCTOR In-memory sorts/filters out objects in this collection.

Parameters:

- *values* the collection of values, not null.

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

supportsFilter(filter∗ : Filter) : Boolean

supportsSort(sort∗ : Sort) : Boolean

toString() : String

IdentityValueProvider Simply provides the row itself as the row-extracted value. Useful in Combo-boxes or Tables, for example when modeling a column with the "Delete" or "Edit" action link - this way, the link will gain access to the entire Record and can delete it easily.

valueType : Type<Object>

IdentityValueProvider(valueType : Type<Object>) CONSTRUCTOR Creates an identity provider with the declared value type. If the value type is null then the type Object is used.

Parameters:

- *valueType* the value type

get(row : Object) : Object

getPropertyPath() : PropertyPath

getValueType() : Type<Object>

TreeDataSource **getChildren(parent : Object) : List<Object>** Retrieves child objects for given parent object. Initially, this method is called with the null parameter, to retrieve all possible root nodes; as the root nodes are expanded by the user, this method is gradually called for the node objects.

Parameters:

- *parent* retrieve children for this parent.

Returns:

- the child nodes. If empty or null, the parent node is marked as a leaf node (it has no children).

ClosureTreeDataSource **closure : {Object : List<Object>}** Called to poll the list of children for given parent. See TreeDataSource.getChildren() for details. Not null.

ClosureTreeDataSource(closure∗ : {Object : List<Object>}) CONSTRUCTOR

getChildren(parent : Object) : List<Object>

SimpleTwoLevelTreeDS Constantly provides given map as a simple two-level tree: map keys are simply returned as root objects, map values are returned as respective root children.

data : Map<Object, List<Object>>

SimpleTwoLevelTreeDS(data∗ : Map<Object, List<Object>>) CONSTRUCTOR

getChildren(parent : Object) : List<Object>

EmptyTreeDataSource **getChildren(parent : Object) : List<Object>**

Compatibility31DataSource DEPRECATED A special data source used to convert ui-based forms.

data : {Integer, Integer : Collection<Object>}

dataCount : { : Integer}

Compatibility31DataSource(data∗ : {Integer, Integer : Collection<Object>}, dataCount∗ : { : Integer}) CONSTRUCTOR

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

supportsFilter(filter∗ : Filter) : Boolean

supportsSort(sort∗ : Sort) : Boolean

ToStringValueProvider **get(object : Object) : Object** Converts object to string

getPropertyPath() : PropertyPath Not applicable - returns null.

getValueType() : Type<Object> Returns string type

QueryDataSource **query : { : List<Object>}**

QueryDataSource(query∗ : { : List<Object>}) CONSTRUCTOR

getCount(filters : Collection<Filter>) : Integer

getData(startIndex∗ : Integer, count∗ : Integer, filters : Collection<Filter>, sortSpecs : Set<Sort>) : List<Object>

supportsFilter(filter∗ : Filter) : Boolean

supportsSort(sort∗ : Sort) : Boolean

toString() : String

Op Equal

Less

Greater

LessOrEqual

GreaterOrEqual

7.2 Functions

7.2.1 Forms

allVaadinIconNames() : List<String>

callNative(c∗ : FormComponent, method∗ : String, args : Object...) : ObjectSIDE EFFECT Calls native method on given component. Please see the documentation of the FormComponent.call() method for more information.

callNative(c∗ : FormComponent, method∗ : String, args : List<Object>) : ObjectSIDE EFFECTDEPRECATED Calls native method on given component. Please see the documentation of the FormComponent.call() method for more information.

DEPRECATED. Use callNative(FormComponent c, String method, Object... args)

clearDataErrorMessages(component : FormComponent) : voidSIDE EFFECT Clear data error messages on the component and all it's descendant components. If no component is provided then this function starts with the root component and process all components in the hierarchy.

As a result, the `getDataErrorMessages` method of each form component in the hierarchy returns an empty list and `getDataErrorMessagesRecursive` returns an empty map.

Parameters:

- *component* the root of the component sub-tree to walk through

downloadFile(resource∗ : FileResource) : String The file will be downloaded by browser to desktop.

fontAwesome5Icon(iconName : String, codepoint : Integer, faClasses : String, inline : String) : String

getTimersDiagnostics() : List<String> Returns information about all currently running timers. Format is not guaranteed and may change in the future.

localize(message : String, c : FormComponent) : String Replaces all localizable substrings in the message parameter by the corresponding localized strings from the user's language.

Warning! Use this method only when need to localize messages which are not part of a document's/todo's saved state. If doing so the reopened saved document/todo with a user using different language will see the text in the original language which is not correct.

Parameters:

- *message* the message to localize
- *c*

notify(caption∗ : String, description : String, ntype : NotificationType, alignment : Alignment, delayMillis : Integer, cssStyle
Shows a simple notification to the user. Must be called from the UI listener.

Parameters:

- *caption* The main notification body, required
- *description* Additional notification text, may be null
- *ntype* Defaults to Info

- *alignment* Default value depends on the 'type' parameter.
- *delayMillis* if 0 or greater, the notification auto-closes after specified period of milliseconds after any user activity; if -1, the notification never disappears and must be clicked by the user. The default value depends on the 'type' parameter
- *cssStyle*
- *htmlContentAllowed* Defaults to false. If false, all html content in caption/description is escaped.

showDataErrorMessages(constraintViolations∗ : List<ConstraintViolation>, component : FormComponent) : void SIDE EFFEC
Shows constraint violations on the component and it's descendant components. This function distributes the data error messages to the components which are bound to the record properties which are source of the constraint violations. If no component is specified then all components in the form are evaluated as targets for the error message.

Distributed constraint violations can be read from `getDataErrorMessages` and `getDataError↩MessagesRecursive` of form component.

Parameters:

- *constraintViolations*
- *component* the component defining the root of the component sub-tree to to walk through. If null then this function starts with the form's root component.

showDataErrorMessages(constraintViolations∗ : List<ConstraintViolation>, components : List<FormComponent>, fallbac
Shows constraint violations on the component and it's descendant components. This function distributes the data error messages to the components which are bound to the record properties which are source of the constraint violations.

Distributed constraint violations can be read from `getDataErrorMessages` and `getDataError↩MessagesRecursive` of form component.

Parameters:

- *constraintViolations*
- *components* the components defining the roots of the component sub-trees to to walk through. Can not be null.
- *fallback* constraint violations that were not mapped to any components are shown here. Can be null.

vaadinIcon(iconName∗ : String) : String

vaadinIcon(iconName∗ : String, className∗ : String) : String

vaadinIcon(iconName∗ : String, className∗ : String, inlineStyle∗ : String) : String

7.2.2 Datasource

<(path∗ : PropertyPath, value∗ : Object) : Filter

<=(path∗ : PropertyPath, value∗ : Object) : Filter

=(path∗ : PropertyPath, value : Object) : Filter

==(path∗ : PropertyPath, value : Object) : Filter

>(path∗ : PropertyPath, value∗ : Object) : Filter

>=(path∗ : PropertyPath, value∗ : Object) : Filter

orderBy(ds∗ : DataSource, sortSpecs : Set<Sort>) : DataSource Creates new DataSource which prepends given set of sort criteria when calling this datasource's getData().

Parameters:

- *ds* the data source

- *sortSpecs* prepend this set of sort criteria, thus sorting primarily by this criteria. Null and empty sets are allowed, in such case "this" is simply returned.

Returns:

- data source which always applies given sort criteria first. Never null.

where(ds∗ : DataSource, filters : Collection<Filter>) : DataSource Creates new DataSource which prepends given set of filters when calling this datasource's getData() and getCount()

Parameters:

- *filters* always apply this set of filter criteria, along with any other filters provided in getCount()/getData(). Null and empty sets are allowed, in such case "this" is simply returned.

Returns:

- data source which always applies given filters first. Never null.

Chapter 8

Module core

The *core* module contains data types, functions, and task types used for model reflection, for accessing the model instance execution context, and for fundamental manipulation with primitive data types in GO-BPMN.

It is necessary for proper functioning of GO-BPMN models; therefore, it is recommended to include it in each module.

- Constants
- Constraint Types
- Data Types
- Functions
- Tasks

8.1 Constants

8.1.1 Core

ALIVE: String = #"ALIVE" A constant representing the Alive execution state.

NOT_FINISHED: String = #"NOT FINISHED" A constant representing the Not finished execution state.

INACTIVE: String = #"INACTIVE" A constant representing the Inactive execution state.

ACTIVE: String = #"ACTIVE" A constant representing the Active execution state.

READY: String = #"READY" A constant representing the Ready execution state.

RUNNING: String = #"RUNNING" A constant representing the Running execution state.

FINISHED: String = #"FINISHED" A constant representing the Finished execution state.

ACHIEVED: String = #"ACHIEVED" A constant representing the Achieved execution state.

FAILED: String = #"FAILED" A constant representing the Failed execution state.

DEACTIVATED: String = #"DEACTIVATED" A constant representing the Deactivated execution state.

INFO_LEVEL: Integer = 200 A constant representing the Info log level.

DEBUG_LEVEL: Integer = 100 A constant representing the Debug log level.

WARNING_LEVEL: Integer = 300 A constant representing the Warning log level.

ERROR_LEVEL: Integer = 400 A constant representing the Error log level.

EMAIL_REGEXP: String = #"^\[_A-Za-z0-9-\+\]+\(\.\[_A-Za-z0-9-\]+\)*\[A-Za-z0-9-\]+\(\.\[A-Za-z0-9\]+\)*\(\.\[A-Za-z\]{2,}\)$"
Regular expression for the e-mail address format.

8.2 Constraint Types

8.2.1 Core

AssertFalse(message : String) Applied to: Boolean

The Boolean value must be false.

AssertTrue(message : String) Applied to: Boolean

The Boolean value must be true.

NotNull(message : String) Applied to: Object

The value must not be null.

Null(message : String) Applied to: Object

The value must be null.

NotEmpty(message : String) Applied to: String

The String value must not be empty.

Empty(message : String) Applied to: String

The String value must be empty.

Min(lowerBound : Decimal, message : String) Applied to: Decimal

The Decimal value must be equal to or greater than the specified lowerBound.

Max(upperBound : Decimal, message : String) Applied to: Decimal

The Decimal value must be equal to or less than the specified upperBound.

Range(lowerBound : Decimal, upperBound : Decimal, message : String) Applied to: Decimal

The Decimal value must be within the range of the interval [lowerBound, upperBound] (inclusive).

MinLength(min : Integer, message : String) Applied to: String

The String value must have a length equal to or greater than the specified min value.

MaxLength(max : Integer, message : String) Applied to: String

The String value must have a length equal to or less than the specified max value.

IsNumber(message : String) Applied to: String

The String value must represent a decimal number.

Pattern(format : String, message : String) Applied to: String

The String value must be in the specified format.

format : String The Java regular expression syntax (as defined in the java.Functions.regex.Pattern class) is used.

Email(message : String) Applied to: String

The String value must match the e-mail address format.

Past(before : Date, message : String) Applied to: Date

The Date value must be equal to or before the date specified by the before parameter.

Future(after : Date, message : String) Applied to: Date

The Date value must be equal to or later than the specified after parameter.

CardinalityMin(min : Integer, condition : {T : Boolean}, message : String) Applied to: Collection<T>

The cardinality of the collection elements that meet the specified condition must be at least the specified min.

The condition is a closure with an element of the collection as its input parameter.

CardinalityMax(max : Integer, condition : {T : Boolean}, message : String) Applied to: Collection<T>

The cardinality of the collection elements that meet the specified condition must not be greater than the specified max.

The condition is a closure with an element of the collection as its input parameter.

CardinalityRange(min : Integer, max : Integer, condition : {T : Boolean}, message : String) Applied to↩ : Collection<T>

The cardinality of the collection elements that meet the specified condition must be in the interval [min, max] (inclusive).

The condition is a closure with an element of the collection as its input parameter.

ExpressionConstraint(expression : {T, Map<String, Object> : String}) Applied to: T

The value must satisfy the constraint given by the specified expression.

ComplexExpressionConstraint(expression : {T, Map<String, Object>, Collection<Tag> : List<ConstraintViolation>}) Applied to: T

The value must satisfy the constraint given by the specified expression.

RecordValidity() Applied to: Record

All properties of the Record value must be valid.

RecordCollectionValidity() Applied to: Collection<Record>

All records from the collection must be valid.

8.3 Data Types

8.3.1 Core

GoalPlan[ABSTRACTSYSTEM] An abstract record type used to represent a goal or a plan.

name : String Name of the goal or plan.

state : String Current state of execution of the goal or plan.

parent : Goal Parent goal of the GoalPlan in the current module instance.

metadata : Map<String, String> Set of metadata (key-value pairs) of the goal or plan.

Goal[ABSTRACTSYSTEM] **extends GoalPlan** A goal of any kind.

children : Set<GoalPlan> Set of children (goals or plans).

isAnyPlanRunning : Boolean If true, at least one of the goal's sub-plans (taken recursively, i.e., not only direct sub-plans) is running. If false, none of the goal's sub-plans are running.

runningPlans : Set<Plan> Set of the direct sub-plans that are running. It is always an empty set for a goal decomposed into goals.

AchieveGoal[SYSTEMFINAL] **extends Goal** A single achieve goal.

preCondition : { : Boolean} Precondition of the achieve goal.

deactivateCondition : { : Boolean} Deactivate condition of the achieve goal.

MaintainGoal[SYSTEMFINAL] **extends Goal** A single maintain goal.

maintainCondition : { : Boolean} Maintain condition of the maintain goal.

Plan[SYSTEMFINAL] **extends GoalPlan** Represents a single plan.

preCondition : { : Boolean} Precondition of the plan.

Model[SHAREDSYSTEM] A model

id : Integer Unique identifier of the model.

name : String Name of the model.

version : String Version of the model.

uploadDate : Date Time when the model was uploaded.

ModelInstance[SHAREDSYSTEM] A model instance.

id : Integer Unique identifier of the model instance.

isRunning : Boolean If true, the model instance is running. If false, the model instance has finished.

start : Date Start date of the model instance.

finish : Date Finish date of the model instance. It is null if the model instance is not finished yet.

ProcessInstance[SYSTEM] A process instance.

id : Integer Identifier of the process instance.

process : String Name of the underlying process.

modelInstance : ModelInstance The owning model instance.

isRunning : Boolean If true, the process instance is running. If false, the process instance has finished.

metadata : Map<String, String> Set of metadata (key-value pairs) of the process instance.

Duration Duration expressed in various time units. Its value is given as the sum of all values of the units. Each field value can be positive or negative.

years : Integer Number of years of the duration.

months : Integer Number of months of the duration.

weeks : Integer Number of weeks of the duration.

days : Integer Number of days of the duration.

hours : Integer Number of hours of the duration.

minutes : Integer Number of minutes of the duration.

seconds : Integer Number of seconds of the duration.

millis : Integer Number of milliseconds of the duration.

RepeatedGoal A goal that is to be repeated and a set of settings of slots related to the goal repetition. This type is used in the RepeatGoals task type.

goal : AchieveGoal Achieve goal to be repeated.

slotSettings : Map<Reference<Object>, Object> Map of references to slots and the values to be used for their setting. This allows you to set variable values on goal repeat.

ConstraintViolation Violation of a single data constraint on a record or property. Note that a validation results in a list of constraint violations.

id : String ID of the violated constraint. It is assigned during validation to all created ConstraintViolation objects that have a null id.

guid : String GUID of the violated constraint. It is assigned automatically after the constraint execution during validation to all created ConstraintViolation objects that have a null guid.

message : String Message returned by the constraint expression defined in the corresponding ConstraintType.

record : Record Record with the value that did not pass the validation.

property : Property Property of the record which did not pass the validation.

payload : Object Application data.

Tag Tag that is used to classify data constraints and filter them on runtime.

name : String Full name of the validation tag, for example, "mymodule::mytag".

Activity Action performed by an Execution task.

It is used also as the supertype for activity-reflection types for task types or processes.

Such an Action subtype is created when the "Create activity reflection type" option of a process or task type is selected.

UserTrack[SHAREDSYSTEM] Record for the entry of the user-activity tracking.

id : Integer Unique ID of the user tracking.

userLogin : String Login name of the tracked user.

startDate : Date Time of the first activity of the track entry.

endDate : Date Time of the last activity of the track entry.

File File with binary content.

content : Binary Binary content.

filename : String Filename of the file.

mime : String MIME type of the data.

size : Integer Actual size of the binary data in bytes.

BinaryHandle[SHAREDDEPRECATED] **extends File** Descriptor of binary data. The record is used internally by LSPS server to store data. Any modification or deletion of BinaryHandle records might render the system unusable.

id : Integer Unique identifier of the binary data.

description : String Description of the data.

Time Represents a time in a day, such as 10:15:30.

A time can optionally define a time zone offset.

Time is represented to millisecond precision.

hour : Integer The hour.

minute : Integer The minute.

second : Integer The second.

millis : Integer The millisecond.

timeZoneOffset : ZoneOffset An optional time-zone offset.

ZoneOffset[READ-ONLY] A time-zone offset from Greenwich/UTC, such as +02:00.

hours : Integer The time-zone offset hours, from -18 to +18.

minutes : Integer The time-zone offset minutes, from -59 to +59, sign matches hours and seconds.

seconds : Integer The time-zone offset seconds, from -59 to +59, sign matches hours and minutes.

RecordProxySet[SYSTEM] **lazySpreadingProxies(record∗ : T...) : List<T>** Creates proxies for the record holders. If the record already has proxy in this proxy set it returns the existing proxy. If a new proxy is created it is added to this proxy set. There must be always continuous chain of proxies from level 0 to level n, therefore missing proxies on parent proxy sets are created automatically. On access a property of the proxy, a new proxy is created automaticaly. The new proxy created for a propery is again lazy spreading proxy.

Parameters:

- *record* for which proxy is created

lazySpreadingProxy(record∗ : T) : T Creates a proxy for the record holder. If the record already has a proxy in this proxy set, it returns the existing proxy. If a new proxy is created, it is added to this proxy set. There always must be a continuous chain of proxies from level 0 to level n;; therefore any proxies missing on parent proxy sets are created automatically. When a proxy property is accessed, a new proxy is created automatically. The new proxy created for a property is a lazy-spreading proxy as well.

Parameters:

- *record* for which the proxy is created

merge(checkConflicts∗ : Boolean) : List<T> Merges all proxies for the proxy set to their proxied records.

Parameters:

- *checkConflicts* If the value is true optimistic lock conflicts are checked. In other case the merge overwrites previous changes.

Returns:

- list of merged records

proxies(records∗ : Collection<T>, properties : Property...) : Collection<T> Creates a proxy on each record holder in the collection. If a record already has its proxy in the proxy set, it uses the existing proxy. If a new proxy is created, it is added to the proxy set. There always must be a continuous chain of proxies from level 0 to level n; therefore proxies missing on the parent proxy sets are created automatically. Proxies are created for all listed property values. The created proxies are added to the proxy set.

Parameters:

- *records* collection of records
- *properties* list of record's properties

Returns:

- collection of created proxies

proxies(records∗ : Collection<T>, properties : List<Property>) : Collection<T> Creates a proxy on each record holder in the collection. If a record already has its proxy in the proxy set, it uses the existing proxy. If a new proxy is created, it is added to the proxy set. There always must be a continuous chain of proxies from level 0 to level n; therefore any proxies missing on parent proxy sets are created automatically. Proxies are created for all listed property values. The created proxies are added to the proxy set.

Parameters:

- *records* collection of records
- *properties* list of record's properties

Returns:

- collection of created proxies

proxy(recordType∗ : Type<T>) : T Creates a new proxy of the record type. The new proxy is added to the proxy set.

Parameters:

- *recordType* record type for which a new record proxy is created

Returns:

- created proxy

proxy(record∗ : T, properties : List<Property>) : T Creates a proxy for the record holder. If the record already has a proxy in this proxy set, it returns the existing proxy. If a new proxy is created, it is added to this proxy set. There always must be a continuous chain of proxies from level 0 to level n; therefore any proxies missing on parent proxy sets are created automatically. Proxies are created for all listed property values. The created proxies are added to the proxy set.

Parameters:

- *record* for which the proxy is created
- *properties* properties of the record for which proxies are created

proxy(record∗ : T, property : Property...) : T Creates a proxy for the record holder. If the record already has proxy in this proxy set, it returns the existing proxy. If a new proxy is created, it is added to this proxy set. There always must be a continuous chain of proxies from level 0 to level n; therefore any proxies missing on parent proxy sets are created automatically. Proxies are created for all listed property values. The created proxies are added to the proxy set.

Parameters:

- *record* for which the proxy is created
- *property* properties of the record for which proxies are created

proxyOfProperties(record∗ : Record, properties : Property...) : void Creates proxies of the listed property values of the record. The created proxies are added to the proxy set.

If there are no listed properties the action is executed for all properties of the record type.

Parameters:

- *record* record for which the proxies of the property values are created
- *properties* list of record's properties

SslConfig Ssl configuration for http calls and webservices. Th e field sslContextProvider provides an instance of com.whitestein.lsps.wsclient.SslContextProvider . Other fields are ignored.

sslContextProvider : Object An instance of com.whitestein.lsps.wsclient.SslContextProvider.

keyStoreType : String Type of key store.

keyStoreDataFile : String Name of the key store data file. It must be accesible as a resource in the custom LSPS Application.

keyStorePassword : String Key-store password.

keyPassword : String Key password.

ModelInstanceStartStatus Start status of model instance

IN_PROGRESS Model instance is starting

IN_PROGRESS_EXPLICIT_FINISH Model instance is starting and expects explicit finish

FINISHED Model instance starting succeeded

FAILED Starting of model instance failed

8.4 Functions

8.4.1 Proxy

createProxySet(parentProxySet : RecordProxySet) : RecordProxySet Creates new record proxy set with parent proxy set.

Parameters:

- *parentProxySet*

getProxiedRecord(proxy∗ : T) : T Returns proxied record of the proxy.

Parameters:

- *proxy*

getProxyLevel(record : Record) : Integer

getProxySet(record : Record) : RecordProxySet

isProxy(record∗ : Record) : Boolean Returns true if the record is proxy, false otherwise.

Parameters:

- *record*

Throws:

- *NullParameterError* if a mandatory parameter is null

mergeAllProxies(checkConflicts∗ : Boolean, records∗ : Collection<T>) : List<T> SIDE EFFECT Merges changes from the proxies to the proxied records. It returns proxied records with applied changes.

Parameters:

- *checkConflicts* If the value is true optimistic lock conflicts are checked. In other case the merge overwrites previous changes.
- *records*

Throws:

- *NullParameterError* if a mandatory parameter is null

mergeProxies(checkConflicts∗ : Boolean, records : T...) : List<T> SIDE EFFECT Merges changes from the proxies to the proxied records. It returns proxied records with applied changes.

Parameters:

- *checkConflicts* If the value is true optimistic lock conflicts are checked. In other case the merge overwrites previous changes.
- *records*

proxies(collection∗ : Collection<T>, properties : Property...) : Collection<T> Creates change proxies on records from the collection and change proxies for values referenced by named properties. The proxy objects keeps changes that should be executed on the record. The proxy objects are added to the proxy set if specified.

Parameters:

- *collection*
- *properties*

Throws:

- *NullParameterError* if a mandatory parameter is null

proxy(recordType∗ : Type<T>) : T Creates a proxy without proxied record. It can be used only for shared records. When merged a shared record is created.

Parameters:

- *recordType*

Throws:

- *NullParameterError* if a mandatory parameter is null

proxy(record∗ : T, properties : Property...) : T Creates proxy on the record and proxies for values referenced by named properties.

Parameters:

- *record*

- *properties*

Throws:

- *NullParameterError* if a mandatory parameter is null

proxyOfProperties(record∗ : Record, properties : Property...) : void[SIDE EFFECT] Creates change proxy for all object referenced by the property and sets the proxy values to the property. If there are no listed properties the action is executed for all properties of the record type.

Parameters:

- *record*
- *properties*

Throws:

- *NullParameterError* if a mandatory parameter is null

removeFromProxySet(record∗ : Record) : void

8.4.2 Collection

+(collection∗ : Collection<E>, elements∗ : Collection<E>) : Collection<E> Returns a collection created by appending the specified elements to the end of the collection. The original order of the added elements is preserved. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

[1,2] + {2,3} //returns [1,2,2,3]; {2,3} + [1,2] //returns {2,3,1}

Parameters:

- *collection* input collection
- *elements* element to appended to the input collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

+(left∗ : Map<K, V>, right∗ : Map<K, V>) : Map<K, V> Returns a map created as the union of the keys and their values of the two input maps. If a key is present in both maps, its value is set to the value of the key specified by the right map.

```
[1 -> "a"] + [1 -> "b", 2 -> "c"] //returns [1->"b",2->"c"]
```

Parameters:

- *left* left map
- *right* right map

Throws:

- *NullParameterError* if a mandatory parameter is not specified

+(list∗ : List<E>, collection∗ : Collection<E>) : List<E> Returns a list created by appending the specified elements to the end of the list. The original order of the added elements is preserved in the returned list.

Parameters:

- *list* input list
- *collection* elements to be appended to the input list

Throws:

- *NullParameterError* if a mandatory parameter is not specified

+(set∗ : Set<E>, collection∗ : Collection<E>) : Set<E> Returns a set created by appending the specified elements to the end of the input set. The original order of the added elements is preserved.

```
{"s", "e", "a", "t"} + ["e", "d"] // returns{"s","e","a","t","d"}
```

Parameters:

- *set* input set
- *collection* elements to be appended to the input set

Throws:

- *NullParameterError* if a mandatory parameter is not specified

-(collection∗ : Collection<E>, elements∗ : Collection<E>) : Collection<E> Removes all occurrences of the elements from the collection and returns the resulting collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
[1,1,2,3] - [1];
// returns: [2,3]
{ "hello", 1,2,3} - ["hello"]
// returns = {1,2,3}
```

Parameters:

- *collection* input collection
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

-(list∗ : List<E>, elements∗ : Collection<E>) : List<E> Removes all occurrences of the elements from the input list and returns the resulting list.

```
[ "hello", "hello", 1,2,3] - ["hello"]
// returns = [1,2,3]
```

Parameters:

- *list* input list
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

-(set∗ : Set<E>, elements∗ : Collection<E>) : Set<E> Removes the elements from the input set and returns the resulting set.

```
{1,2,3} - [1];
// returns: [2,3]
```

Parameters:

- *set* input set
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

add(collection∗ : Collection<E>, elements : E...) : Collection<E> Returns a collection with the input-collection items and the specified elements added to the end of the collection. If the input collection is a set, the returned collection is Set. If the input collection is a list, the returned collectionc is List.

Parameters:

- *collection* input collection
- *elements* elements to append to the returned collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

add(list∗ : List<E>, elements : E...) : List<E> Returns a list with the input-list items and the specified elements added to the end of the list.

```
[1,2,3].add(4, 4, 5)//returns [1,2,3,4,4,5]
```

Parameters:

- *list* input list
- *elements* elements to append to the returned list

Throws:

- *NullParameterError* if a mandatory parameter is not specified

add(set∗ : Set<E>, elements : E...) : Set<E> Returns a set with the input-set items and the specified elements added to the end of the set.

```
{1,2,3}.add(4, 4, 5)//returns {1,2,3,4,5}
```

Parameters:

- *set* input set
- *elements* elements to append to the returned set

Throws:

- *NullParameterError* if a mandatory parameter is not specified

add(map∗ : Map<K, V>, key : K, value : V) : Map<K, V> Returns a map with the input-map items and the element with the specified key-value pair added to the end of the map. If the key already exists in the map, the specified value overwrites the value of the existing key.

Parameters:

- *map* input map
- *key* key to add
- *value* key value to add

Throws:

- *NullParameterError* if a mandatory parameter is not specified

addAll(collections∗ : Collection<E>...) : Collection<E> Returns a collection created as the concatenation of the specified collections. If the first input collection is a set, the returned type is Set. If the first input collection is a list, the returned type is List.

```
addAll([1,2,3], {"a", "b"}, {true, false, null})
 //returns [1,2,3,"a","b",true,false,null]
```

Parameters:

- *collections* input collections

Throws:

- *NullParameterError* if a mandatory parameter is not specified

addAll(maps∗ : Map<K, V>...) : Map<K, V> Returns a map created as the union of keys and their respective values of all input maps. If more than one map contains the same key, its value is set to the value of the key specified by the most recent map.

```
addAll([1 -> "a"], [1 -> "b", 2 -> "a"], [1 -> "c"] ) //returns [1->"c",2->"a"]
```

Parameters:

- *maps* input maps

Throws:

- *NullParameterError* if a mandatory parameter is not specified

addAll(list∗ : List<E>, collections∗ : Collection<E>...) : List<E> Returns a list created as concatenation of the input list and collections.

```
addAll([1,2],{3,4}, [4,5]) // returns [1,2,3,4,4,5]
```

Parameters:

- *list* input list
- *collections* collections to add to the list

Throws:

- *NullParameterError* if a mandatory parameter is not specified

addAll(set∗ : Set<E>, collections∗ : Collection<E>...) : Set<E> Returns a set created by concatenating the set and collections.

```
addAll({1,2},[3,4], [4,5]) //returns {1,2,3,4,5}
```

Parameters:

- *set* input set
- *collections* collections to add to the set

Throws:

- *NullParameterError* if a mandatory parameter is not specified

addAll(collection∗ : Collection<E>, index∗ : Integer, elements∗ : Collection<E>...) : Collection<E> Returns a collection with the input-collection items and the specified elements inserted at the position defined by the index parameter. The element originally located at the index position and any subsequent elements to the right are shifted to the indices incremented by the number of the elements added to the collection. The returned collection preserves the original order of the added elements. If the input collection is a set, the returned collection is Set. If the input collection is a list, the returned collection is List.

```
addAll({1,2}, 1, {42, 73}, {13})//returns {1,42,73,13,2}
```

Parameters:

- *collection* input collection
- *index* index position of the adding
- *elements* elements to add

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index $<$ 0 or index $>$ size(collection))

addAll(list∗ : List<E>, index∗ : Integer, elements∗ : Collection<E>...) : List<E> Returns a list with the input-list items and the specified elements inserted at the position defined by the index parameter. The element originally located at the index position and any subsequent elements to the right are shifted to the indices incremented by the number of the elements added to the list. The returned list preserves the original order of the added elements.

```
addAll([1,2], 1, [42, 73], [13]) //returns [1,42,73,13,2]
```

Parameters:

- *list* input list
- *index* index position of the adding
- *elements* elements to add

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index $<$ 0 or index $>$ size(list))

addAll(set∗ : Set<E>, index∗ : Integer, elements∗ : Collection<E>...) : Set<E> Returns a set with the input-set items and the specified elements inserted at the position defined by the index parameter. The element originally located at the index position and any subsequent elements to the right are shifted to the indices incremented by the number of the elements added to the set. The returned set preserves the original order of the added elements.

```
addAll({1,2}, 1, {42, 73}, {13})//returns {1,42,73,13,2}
```

Parameters:

- *set* input set
- *index* index position of the adding
- *elements* elements to add

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index $<$ 0 or index $>$ size(set))

addAt(collection∗ : Collection<E>, index∗ : Integer, elements : E...) : Collection<E> Returns a collection created by adding the input elements to the input collection at the position specified by the index. The element at the index position and the subsequent elements are shifted to the index positions incremented by the number of the elements added to the set. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

Parameters:

- *collection* input collection
- *index* target index position
- *elements* elements to insert at the target index position

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the index is out of range (index $<$ 0 or index $>$ size(collection))

addAt(list∗ : List<E>, index∗ : Integer, elements : E...) : List<E> Returns a list created by adding the input elements to the input list at the position specified by the index. The element at the index position and the subsequent elements are shifted to the index positions incremented by the number of the elements added to the list.

```
addAt([1,1,2,3], 1, "a", 4.2) //returns [1,"a",4.2,1,2,3]
```

Parameters:

- *list* input list
- *index* target index position
- *elements* element to insert at the target index position

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the index is out of range (index $<$ 0 or index $>$ size(list))

addAt(set∗ : Set<E>, index∗ : Integer, elements : E...) : Set<E> Returns a set created by adding the input elements to the input set at the position specified by the index. The element at the index position and the subsequent elements are shifted to the index positions incremented by the number of the elements added to the set.

```
addAt({1,1,2,3}, 1, "a", 4.2) //returns [1,"a",4.2,1,2,3]
```

Parameters:

- *set* input set
- *index* target index position
- *elements* element to insert to the target index position

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the index is out of range (index $<$ 0 or index $>$ size(set))

collect(collection∗ : Collection<E>, function∗ : {E : T}) : Collection<T> Runs the function on each collection element and returns a collection with the output values as its elements. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
collect({1,2}, {i:Integer -> i + 1}) //returns {2,3}
```

Parameters:

- *collection* input collection
- *function* function to run on each collection element

Throws:

- *NullParameterError* if a mandatory parameter is not specified

collect(list∗ : List<E>, function∗ : {E : T}) : List<T> Runs the function on each list element and returns a list with the output values as its elements.

```
collect([1,2], {i:Integer -> i + 1}) //returns [2,3]
```

Parameters:

- *list* input list
- *function* function to run on each list element

Throws:

- *NullParameterError* if a mandatory parameter is not specified

collect(map∗ : Map<K, V>, function∗ : {K, V : E}) : List<E> Runs the function on each map key-value pair and returns a list with the output values as its elements.

```
collect(["key" -> "value"], { key:String, value:String -> key + " " + value}) // returns ["key value"]
```

Parameters:

- *map* input map
- *function* function to run on each key-value pair of the map

Throws:

- *NullParameterError* if a mandatory parameter is not specified

collect(set∗ : Set<E>, function∗ : {E : T}) : Set<T> Runs the function on each set element and returns a set with the output values as its elements.

```
collect({1,2}, {i:Integer -> i + 1}) //returns {2,3}
```

Parameters:

- *set* input set
- *function* function to run on each set element

Throws:

- *NullParameterError* if a mandatory parameter is not specified

collectWithIndex(collection∗ : Collection<E>, function∗ : {Integer, E : T}) : Collection<T> Runs the function on each element of the input collection and returns a collection with the output values as its elements. The function takes 2 parameters: the index of the element and the element itself. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
collectWithIndex(
   collection -> {"a", "b"},
   function -> { index:Integer, item:String -> if index == 0 then item := "c" else item :="d" end }
 ) //returns {"c","d"}
```

Parameters:

- *collection* input collection
- *function* function to run on each collection element

Throws:

- *NullParameterError* if a mandatory parameter is null

collectWithIndex(list∗ : List<E>, function∗ : {Integer, E : T}) : List<T> Runs the function on each list element and returns a list with the output values as its elements. The function takes 2 parameters: the index of the element and the element itself.

```
collectWithIndex(
   ["a", "b"],
   { index:Integer, item:String -> if index == 0 then item := "c" else item :="d" end }
 ) //returns ["c","d"]
```

Parameters:

- *list* input list
- *function* function to run on each list element

Throws:

- *NullParameterError* if a mandatory parameter is null

collectWithIndex(set∗ : Set<E>, function∗ : {Integer, E : T}) : Set<T> Runs the function on each set element and returns a set with the output values as its elements. The function takes 2 parameters: the index of the element and the item element.

```
collectWithIndex(
   {"a", "b"},
   { index:Integer, item:String -> if index == 0 then item := "c" else item :="d" end }
 ) //returns {"c","d"}
```

Parameters:

- *set* input set
- *function* function to run on each set element

Throws:

- *NullParameterError* if a mandatory parameter is null

compact(collection∗ : Collection<E>) : Collection<E> Removes the elements with the `null` value from the input collection and returns a collection with the non-null elements. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
compact({"a", "b", null}) // returns {"a","b"}
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

compact(list∗ : List<E>) : List<E> Removes the elements with the `null` value from the input list and returns a list with the non-null elements.

```
compact(["a", "b", null]) // returns ["a","b"]
```

Parameters:

- *list* input list

Throws:

- *NullParameterError* if a mandatory parameter is null

compact(set∗ : Set<E>) : Set<E> Removes the elements with the `null` value from the input set and returns a set with the non-null elements.

```
compact({"a", "b", null}) // returns {"a","b"}
```

Parameters:

- *set* input set

Throws:

- *NullParameterError* if a mandatory parameter is null

contains(collection∗ : Collection<Object>, elements : Object...) : Boolean Returns `true` if the collection contains all specified elements.

```
contains(["a","b"], "a", "b") //returns true
```

Parameters:

- *collection* input collection
- *elements* elements

Throws:

- *NullParameterError* if a mandatory parameter is not specified

containsAll(collection∗ : Collection<Object>, elements∗ : Collection<Object>) : Boolean Returns `true` if the collection contains all elements from the specified collection.

```
containsAll({1,2,3}, [1, 1, 2]) //returns true
```

Parameters:

- *collection* input collection
- *elements* collection of elements to check

Throws:

- *NullParameterError* if a mandatory parameter is not specified

containsAny(collection∗ : Collection<Object>, elements∗ : Collection<Object>) : Boolean Returns `true` if the collection contains any of the specified elements.

```
containsAny([1,1,11], {1, 3}) // returns true
```

Parameters:

- *collection* input collection
- *elements* collection of elements checked

Throws:

- *NullParameterError* if a mandatory parameter is not specified

containsKeys(map∗ : Map<Object, Object>, keys : Object...) : Boolean Returns `true` if the map contains all specified keys.

```
containsKeys(["a"-> 1, "b" -> 2], "a", "c") //returns false
```

Parameters:

- *map* input map
- *keys* keys to check

Throws:

- *NullParameterError* if a mandatory parameter is not specified

containsValues(map∗ : Map<Object, Object>, values : Object...) : Boolean Returns `true` if the map contains all specified values.

```
containsKeys(["a"-> 1, "b" -> 2], 1, 2) //returns true
```

Parameters:

- *map* input map
- *values* values to check

Throws:

- *NullParameterError* if a mandatory parameter is not specified

count(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the number of elements in the given collection which satisfy the given condition.

```
count([1,2,3,4,55], {i:Integer -> i > 3}) //returns 2
```

Parameters:

- *collection* input collection
- *condition* condition the elements must meet

Throws:

- *NullParameterError* if a mandatory parameter is not specified

exists(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Boolean Returns `true` if the condition is `true` for at least one element in the collection.

```
exists([1,2,3,4,55], {i:Integer -> i > 54}) //returns trues
```

Parameters:

- *collection* input collection
- *condition* condition to check on collection elements

Throws:

- *NullParameterError* if a mandatory parameter is not specified

findIndex(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the index of the first collection element which satisfies the specified condition. If the condition is `false` for all elements of the collection, it returns `-1`.

```
findIndex(
 collection -> [1,22,3, 44],
 condition -> { i:Integer -> i > 21 }
) //returns 1
```

Parameters:

- *collection* input collection
- *condition* condition to check against each element

Throws:

- *NullParameterError* if a mandatory parameter is null

findIndex(collection∗ : Collection<E>, fromIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the first collection element starting at fromIndex inclusive, which satisfies the specified condition. If the condition is `false` for all elements of the collection starting at *fromIndex*, it returns `-1`. If the *fromIndex* parameter is `null` or is less than 0, index 0 is used as the *fromIndex*.

```
findIndex(
 collection -> [1,22,3, 44],
 fromIndex -> 2,
 condition -> { i:Integer -> i > 21 }
) //returns 3
```

Parameters:

- *collection* input collection
- *fromIndex* start condition check from this index position
- *condition* closure with the condition to check

Throws:

- *NullParameterError* if a mandatory parameter is null

findIndex(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the first collection element starting at fromIndex inclusive and ending with the *toIndex* exclusive which satisfies the specified condition. If the condition is `false` for all elements of the collection in the specified range, it returns `-1`. If the *fromIndex* parameter is `null` or is less than 0, index 0 is used as the *fromIndex*. If the *toIndex* parameter is `null` or greater or equal to the size of the collection, it is set to the last index of the collection.

```
findIndex(
 collection -> [1,22,3, 44],
 fromIndex -> 2,
 toIndex -> 3,
 condition -> { i:Integer -> i > 21 }
) //returns -1
```

Parameters:

- *collection* input collection
- *fromIndex* start condition check from this index position
- *toIndex* finish condition check at this position
- *condition* closure with the condition to check

Throws:

- *NullParameterError* if a mandatory parameter is null

findLastIndex(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the index of the last collection element which satisfies the specified condition. If the condition is `false` for all elements of the collection, it returns `-1`.

```
findLastIndex(
 collection -> [1,22,3, 44],
 condition -> { i:Integer -> i > 21 }
) // returns 3
```

Parameters:

- *collection* input collection
- *condition* condition to check

Throws:

- *NullParameterError* if a mandatory parameter is null

findLastIndex(collection∗ : Collection<E>, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the *last* collection element which meets the condition. The check takes the elements starting from the *toIndex* position and continues toward index 0. If no element meets the condition, it returns -1. If *toIndex* is `null`, or greater or equal to the size of the collection, it is set to the last index of the collection.

findLastIndex(collection -> [1, 33, 4, 22,3, 44], toIndex -> 5, condition -> { i:Integer -> i > 21 })) //returns 3 (checked 3 and 22; //returned the position of 22)

Parameters:

- *collection* input collection
- *toIndex* finish condition check at this index
- *condition* closure condition to check on collection elements

Throws:

- *NullParameterError* if a mandatory parameter is null

findLastIndex(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the *last* collection element that meets the condition. The function searches starting from the *toIndex* and ending at the element preceeding the *fromIndex* element. If no element that meets the condition is found in the specified element range, it returns -1. If the *fromIndex* parameter is `null` or less than 0, the search finishes at the beginning of the collection (index 0). If the *toIndex* parameter is `null` or greater or equal to the size of the collection, it is considered to be the end of the collection.

```
findLastIndex(
 collection -> [1, 33, 4, 22,3, 44],
 fromIndex -> 1,
 toIndex -> 4,
 condition -> { i:Integer -> i > 21 })
 //returns 3 (the index of 22)
```

Parameters:

- *collection* input collection
- *fromIndex* start condition check at this index
- *toIndex* finish condition check at this index
- *condition* closure condition to check on collection elements

Throws:

- *NullParameterError* if a mandatory parameter is null

fold(collection∗ : Collection<E>, initialValue : T, function∗ : {T, E : T}) : T Combines the elements of the collection using the binary function, an operation upon two operands of the same type. The binary function executes over the input collection from left to right. The first input T for the function is defined by *initialValue* and the first element in the collection. In each next iteration, the function returns the resulting computed value which is used as the input value for the next iteration with the next element of the collection. The last computed value is the result of the fold function.

```
fold([1,2,3], 4, { i:Integer, n:Integer -> i + n});
// returns 10: 4 + 1 = 5; 5 + 2 = 7; 7 + 3 = 10
fold([ "b", "c", "d"] , "a;", { param1:String, param2:String -> param1 + " " + param2+ ";"})
//returns  "a; b; c; d;"
```

Parameters:

- *collection* input collection

- *initialValue* first T parameter value for the binary function
- *function* binary function

Throws:

- *NullParameterError* if a mandatory parameter is not specified

foldWithIndex(collection∗ : Collection<E>, initialValue : T, function∗ : {Integer, T, E : T}) : T Combines the elements of the collection using the binary function, an operation upon two operands of the same type. The binary function executes over the input collection from left to right. The first input T for the function is defined by *initialValue* and the first element in the collection. In each next iteration, the function returns the resulting computed value which is used as the input value for the next iteration with the next element of the collection. Input of the function includes the index number of the E element. The last computed value is the result of the fold function.

```
foldWithIndex(["a","b","c","d"],"d",{i:Integer, t:String, e:String -> t + e + " index: " + i.toString() +
// returns "da index: 0; b index: 1; c index: 2; d index: 3; "
```

Parameters:

- *collection* input collection
- *initialValue* first T parameter value for the binary function
- *function* binary function with the E index as it Integer input parameter

Throws:

- *NullParameterError* if a mandatory parameter is not specified

forAll(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Boolean Returns true if the condition is true for all elements in the collection.

```
forAll({4,3,2,1}, {e:Integer -> e > 0});
 //returns true
 forAll({4,3,2,1}, {e:Integer -> e > 1})
 //returns false
```

Parameters:

- *collection* input collection
- *condition* condition to check on all elements

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDuplicates(collection∗ : Collection<E>) : Set<E> Returns the set of duplicate elements in the given collection.

```
getDuplicates([1,2,3,1, "a","a", "b"])
 //result {1,"a"}
```

Parameters:

- *collection*

Throws:

- *NullParameterError* if a mandatory parameter is null

getFirst(collection∗ : Collection<E>) : E Returns the first element of the collection, or null if the collection is empty.

```
getFirst({4,3,2,1})
//returns 4
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

getFirst(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : E Returns the first element for which the condition is true. Returns <code>null</cod> if the collection is empty or the condition is not true for any of the collection elements.

```
getFirst({4,3,2,1},{e:Integer -> e<4})
//returns 3
```

Parameters:

- *collection* input collection
- *condition* condition to check

Throws:

- *NullParameterError* if a mandatory parameter is null

getLast(collection∗ : Collection<E>) : E Returns the last item of the collection, or null if the collection is empty.

```
getLast({4,3,2,1})
 //returns 1
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

getLast(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : E Returns the last element for which the condition is true. Returns <code>null</cod> if the collection is empty or the condition is not true for any of the collection elements.

```
getLast([1,2,4,3,1],{e:Integer -> e>1});
 //returns 3
 getLast({4,3,2,1},{e:Integer -> e>1});
 //returns 2
```

Parameters:

- *collection* input collection
- *condition* condition to check

Throws:

- *NullParameterError* if a mandatory parameter is null

groupBy(collection∗ : Collection<E>, classifier∗ : {E : K}) : Map<K, List<E>> The function performs a group-by operation on the collection elements according to the value returned by the *classifier* function, and returns the result in a map.

```
groupBy([1,2,3,4,5], {e:Integer -> e < 3});
//returns [false->[3,4,5],true->[1,2]]
groupBy(["a", "b", "c", 1, 2, now()], {e:Object -> e.getType()})
 //returns [Integer->[1,2],String->["a","b","c"],Date->[d'2018-06-05 16:19:29.106']]
```

Parameters:

- *collection* input collection
- *classifier* classifier function that returns the groups

Throws:

- *NullParameterError* if a mandatory parameter is null

hasDuplicates(collection∗ : Collection<Object>) : Boolean Returns `true` if the collection contains duplicate elements, `false` otherwise.

```
hasDuplicates([2,2])
 //returns true
```

Parameters:

- *collection*

Throws:

- *NullParameterError* if a mandatory parameter is null

indexOf(collection∗ : Collection<Object>, element : Object) : Integer Returns the index of the first occurrence of the specified element in the collection; returns -1 if the collection does not contain the specified element.

```
indexOf([0,1,2,3,4, 2], 2)
//returns 2
```

Parameters:

- *collection* input collection
- *element* element to find the index of

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isEmpty(collection∗ : Collection<Object>) : Boolean Returns `true` if the collection contains no elements.

```
isEmpty([])
// returns true
```

Parameters:

- *collection* collection to check

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isEmpty(map∗ : Map<Object, Object>) : Boolean Returns true if the specified map contains no elements.

```
isEmpty([null -> null]);
 //returns false
isEmpty(["" -> ""]);
 //returns false
isEmpty([->]);
//returns true
```

Parameters:

- *map*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isSubset(set1∗ : Set<Object>, set2∗ : Set<Object>) : Boolean Returns `true` if set1 is a subset of set2.

```
isSubset({1,2,3},{1,2,3,4});
 //returns true
isSubset({null,1,2,3},{1,2,3,4})
//returns false
```

Parameters:

- *set1* potential subset

- *set2* superset

Throws:

- *NullParameterError* if a mandatory parameter is not specified

join(collection∗ : Collection<Object>) : String Concatenates all non-empty string representations of the collection elements separating them with a comma and space (", ").

```
join(["a", "b", "c", null, "", "d"])
 //returns "a, b, c, d"
```

Parameters:

- *collection* collection of elements to join

Throws:

- *NullParameterError* if a mandatory parameter is not specified

join(collection∗ : Collection<Object>, joinString : String) : String Concatenates all non-empty string representations of the collection elements separating them with *joinString*.

```
join(["a", "b", "c", null, "", "d"], ", ")
 //returns "a, b, c, d"
```

Parameters:

- *collection* collection of elements to join
- *joinString* element-string separator

Throws:

- *NullParameterError* if a mandatory parameter is not specified

join(collection∗ : Collection<Object>, joinString : String, includeEmpty : Boolean) : String Concatenates all string representations of the collection elements. The element are separated with *joinString*. If *include↩ Empty* is `true`, `null` and empty elements are included in the resulting string.

```
join(["a", "b", "c", null, "", "d"], ", ", false)
 // returns "a, b, c, d"
```

join(["a", "b", "c", null, "", "d"], ", ", true) //returns "a, b, c, , , d"</pre>

Parameters:

- *collection* collection with elements to join
- *joinString* element-string separator
- *includeEmpty* whether to include empty elements in the result

Throws:

- *NullParameterError* if a mandatory parameter is not specified

keys(map∗ : Map<K, V>) : Set<K> Returns a set of keys contained in the specified map.

```
keys([1->"a", 2->"b", 3 -> "a"])
 //returns  {1,2,3}
```

Parameters:

- *map* input map

Throws:

- *NullParameterError* if a mandatory parameter is not specified

keysOf(map∗ : Map<K, V>, element : V) : Set<K> Returns the keys for all occurrences of the specified value included in the map.

```
keysOf([1->"a", 2->"b", 3 -> "a"], "a")
 //returns {1,3}
```

Parameters:

- *map* input map
- *element* value for which to return the map keys

Throws:

- *NullParameterError* if a mandatory parameter is not specified

lastIndexOf(collection∗ : Collection<Object>, element : Object) : Integer Returns the index of the last occurrence of the specified element in the collection; returns -1 if the collection does not contain the specified element.

```
lastIndexOf([1,2,3,1,5,6], 1)
 //returns 3
```

Parameters:

- *collection* input collection
- *element* element to find the last occurence of

Throws:

- *NullParameterError* if a mandatory parameter is not specified

listUnion(collections : Collection<Collection<E>>...) : List<E> Returns a list containing all elements from all the collections. Null values of collection are treated as empty collections.

```
listUnion([[null, 1,2,3, "hello"], [1,2,55]])
 = [null,1,2,3,"hello",1,2,55]
```

Parameters:

- *collections* input collection with collections

map(keys∗ : Collection<K>, valueFunction∗ : {K : V}) : Map<K, V> Returns a map with the collection elements as its keys and values computed from the collection elements by the *valueFunction*.

```
map({1,2,3}, {e:Integer -> e + 10} )
 //returns [1->11,2->12,3->13]
```

Parameters:

- *keys* collection with key values
- *valueFunction* function to compute the values for a key

Throws:

- *NullParameterError* if a mandatory parameter is not specified

map(keys∗ : Collection<K>, values∗ : Collection<V>) : Map<K, V> Returns a map by combining collections of keys and values. The number of keys must be equal to the number of values. At least one key and one value must be specified in order to determine the type of the map. If the keys parameter contains the same key several times, the most recently used key-value pair is included in the resulting map.

```
map([1,1,3, null],["a","b","c", null])
//result  [1->"b",3->"c", null->null]
```

Parameters:

- *keys*
- *values*

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *WrongSizeError* if either the number of keys is not equal to the number of values, or one of the input collection is empty

map(collection∗ : Collection<E>, keyFunction∗ : {E : K}, valueFunction∗ : {E : V}) : Map<K, V> Returns a map with keys and values both computed based on the collection elements. For each map key-value pair, the key is calculated with the keyFunction and the value with the valueFunction. Both have the collection element as their input.

```
map({1,2,3}, {e:Integer -> e + 1 }, {e:Integer -> e + 10} )
 //returns [2->11,3->12,4->13]
```

Parameters:

- *collection* input collection
- *keyFunction* function that returns the key
- *valueFunction* function that returns the value

Throws:

- *NullParameterError* if a mandatory parameter is not specified

max(collection∗ : Collection<Date>) : Date Returns the minimum element of the collection or `null` if the collection is empty.

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

max(collection∗ : Collection<LocalDate>) : LocalDate Returns maximal value from collection; or null, if the collection is empty.

Parameters:

- *collection*

Throws:

- *NullParameterError* if a mandatory parameter is null

max(collection∗ : Collection<Decimal>) : Decimal Returns the maximum element of the collection or <code>null</ccode> if the collection is empty.

```
max([0.1,1.4,2.2,3.9,-2.8]) //returns 3.9
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

max(collection∗ : Collection<Integer>) : Integer Returns the maximum element of the collection or <code>null</ccode> if the collection is empty.

```
max([0,1,2,3,-2]) //returns 3
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

max(collection∗ : Collection<String>) : String Returns the string that is first as per Java lexicographic ordering of strings. Returns `null` if the collection is empty.

```
max(["haha", "yes", "no"]) //returns "yes"
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

min(collection∗ : Collection<Date>) : Date Returns the minimum element of the collection or `null` if the collection is empty.

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

min(collection∗ : Collection<LocalDate>) : LocalDate Returns minimal value from collection; or null, if the collection is empty.

Parameters:

- *collection*

Throws:

- *NullParameterError* if a mandatory parameter is null

min(collection∗ : Collection<Decimal>) : Decimal Returns the minimum element of the collection or `null` if the collection is empty.

```
min([0.9,1.0,2.4,3.1,-2.9]) //returns -2.9
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

min(collection∗ : Collection<Integer>) : Integer Returns the minimum element of the collection or <code>null</ccode> if the collection is empty.

```
min([0,1,2,3,-2]) //returns -2
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

min(collection∗ : Collection<String>) : String Returns the string that is first as per Java lexicographic ordering of strings. Returns `null` if the collection is empty.

```
min(["haha", "yes", "no"]) //returns "haha"
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is null

notEmpty(collection∗ : Collection<Object>) : Boolean Returns `true`, if the specified collection contains at least one element.

Parameters:

- *collection*

Throws:

- *NullParameterError* if a mandatory parameter is null

notEmpty(map∗ : Map<Object, Object>) : Boolean Returns `true` if the input map contains at least one element.

```
notEmpty([1->1]) // returns true
```

Parameters:

- *map* input map

Throws:

- *NullParameterError* if a mandatory parameter is null

remove(collection∗ : Collection<E>, elements : Object...) : Collection<E> Removes the first occurrence of each of the elements from the input collection and returns the resulting collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
remove([1,1,2,3] ,1,2);
// returns: [1,3]
remove({1,2,3} , 2)
// returns = {1,3}
```

Parameters:

- *collection* input collection
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

remove(list∗ : List<E>, elements : Object...) : List<E> Removes the first occurrence of the *elements* from the *input list* and returns the resulting list.

```
removeAll([1,1,2,3] ,[1]);
// returns: [2,3]
```

Parameters:

- *list* input list
- *elements* elements to remove from the input list

Throws:

- *NullParameterError* if a mandatory parameter is not specified

remove(set∗ : Set<E>, elements : Object...) : Set<E> Removes the elements from the input set and returns the resulting set.

```
remove({1,2,3,"hello"} , "hello", 2)
// returns = {1,3}
```

Parameters:

- *set* input set

- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeAll(collection∗ : Collection<E>, elements∗ : Collection<Object>) : Collection<E> Removes all occurrences of the elements from the input collection and returns the resulting collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
removeAll([1,1,2,3] ,[1]);
// returns: [2,3]
removeAll({ "hello", 1,2,3} , ["hello"])
// returns = {1,2,3}
```

Parameters:

- *collection* input collection
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeAll(list∗ : List<E>, elements∗ : Collection<Object>) : List<E> Removes all occurrences of the elements from the input list and returns the resulting list.

```
removeAll([ "hello", "hello", 1,2,3], ["hello"])
// returns  [1,2,3]
```

Parameters:

- *list* input list
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeAll(set∗ : Set<E>, elements∗ : Collection<Object>) : Set<E> Removes the elements from the input set and returns the resulting set.

```
{1,2,3} - [1];
// returns: {2,3}
```

Parameters:

- *set* input set
- *elements* elements to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeAt(collection∗ : Collection<E>, index∗ : Integer) : Collection<E> Removes the element at the *index* position in the input collection and returns the new collection. If the *input* collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
removeAt([1,2,3,4], 2)
// returns [1,2,4]
```

Parameters:

- *collection* input collection
- *index* element to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(collection))

removeAt(list∗ : List<E>, index∗ : Integer) : List<E> Removes the element at the *index* position in the input list and returns the new list.

```
removeAt([1,2,3,4], 2)
// returns [1,2,4]
```

Parameters:

- *list*
- *index*

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(list))

removeAt(set∗ : Set<E>, index∗ : Integer) : Set<E> Removes the element at the *index* position in the input set and returns the new set.

```
removeAt({1,2,3,4}, 2)
// returns {1,2,4}
```

Parameters:

- *set*
- *index*

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(set))

removeKey(map∗ : Map<K, V>, keys : Object...) : Map<K, V> Removes all key-value pairs with the keys defined by the *keys* parameter and returns the resulting map.

```
removeKey([ 1 -> "a", "a" -> 2, 3 -> "c"], "a", "c")
 // returns ["a"->2,3->"c"]
```

Parameters:

- *map* input map
- *keys* key to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeKeys(map∗ : Map<K, V>, keys∗ : Collection<Object>) : Map<K, V> Removes all key-value pairs with the keys defined by the *keys* parameter and returns the resulting map.

```
removeKeys([ 1 -> "a", "a" -> 2, 3 -> "c"], ["a", "c"])
 // returns ["a"->2,3->"c"]
```

Parameters:

- *map* input map
- *keys* keys to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

removeRange(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer) : Collection<E>
Returns a collection with the elements of the *input collection* that are out of the specified element range. The range of removed elements starts with *fromIndex* and finishes at the element preceding the *toIndex* element. If *fromIndex* is `null` or is less than 0, the range starts from the beginning of the collection (index 0). If *toIndex* is `null` or greater or equal to the size of the collection, the range ends at the end of the collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

removeRange([0, 1, 2, 3, 4], 2, 4) // returns [0,1,4]

Parameters:

- *collection* input collection
- *fromIndex* first element of the range that is removed
- *toIndex* first element after the range that is retained

Throws:

- *NullParameterError* if a mandatory parameter is null
- *OutOfBoundsError* fromIndex > toIndex

removeRange(list∗ : List<E>, fromIndex : Integer, toIndex : Integer) : List<E> Removes the elements of the *input list* that are in the specified element range from the input list and return the resulting list. The range of removed elements starts with *fromIndex* and finishes at the element preceding the *toIndex* element. If *fromIndex* is `null` or is less than 0, the range starts from the beginning of the list (index 0). If *toIndex* is `null` or greater or equal to the size of the list, the range ends at the end of the list.

removeRange({0, 1, 2, 3, 4}, 2, 4) // returns {0,1,4}

Parameters:

- *list* input list
- *fromIndex* first element of the range that is removed
- *toIndex* first element after the range that is retained

Throws:

- *NullParameterError* if a mandatory parameter is null
- *OutOfBoundsError* fromIndex > toIndex

removeRange(set∗ : Set<E>, fromIndex : Integer, toIndex : Integer) : Set<E> Removes the elements of the *input set* that in the specified element range and returns the resulting set. The range of removed elements starts with *fromIndex* and finishes at the element preceding the *toIndex* element. If *fromIndex* is `null` or is less than 0, the range starts from the beginning of the set (index 0). If *toIndex* is `null` or greater or equal to the size of the set, the range ends at the end of the set.

removeRange([0, 1, 2, 3, 4], 2, 4) // returns [0,1,4]

Parameters:

- *set* input set
- *fromIndex* first element of the range that is removed
- *toIndex* first element after the range that is retained

Throws:

- *NullParameterError* if a mandatory parameter is null
- *OutOfBoundsError* fromIndex > toIndex

removeValue(map∗ : Map<K, V>, values : Object...) : Map<K, V> Returns a map created by removing all key-value pairs with the values defined by the values parameter.

```
removeValue([ 1 -> "a", "a" -> 2, 3 -> "c"], "a", "c")
 // returns ["a"->2]
```

Parameters:

- *map* input map
- *values* values to remove

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replace(collection∗ : Collection<E>, element1 : Object, element2 : E) : Collection<E> Returns a collection created by replacing all occurrences of *element1* with *element2* in the input collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
replace([1,2,3,1,2,3],1, "hello")
// returns ["hello",2,3,"hello",2,3]
```

Parameters:

- *collection* input collection
- *element1* value to replace with the new value
- *element2* new value

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replace(list∗ : List<E>, element1 : Object, element2 : E) : List<E> Returns a list created by replacing all occurrences of *element1* with *element2* in the input list.

```
replace([1,1,2,3], 1, "hello")
// returns ["hello","hello",2,3]
```

Parameters:

- *list* input list
- *element1* element value to replace with the new element
- *element2* the new element

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replace(set∗ : Set<E>, element1 : Object, element2 : E) : Set<E> Returns a set created by replacing *element1* with *element2* in the input set.

```
replace({1,1,2,3}, 1, "hello")
 = {"hello",2,3}
```

Parameters:

- *set* input set
- *element1* set element to replace with the new element
- *element2* new element

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replaceValue(map∗ : Map<K, V>, value1 : Object, value2 : V) : Map<K, V> Returns a map created by replacing all values defined by *value1<i> with <i>value2* in the input map.

```
replaceValue([ 1 -> 1.0, 2 -> 1.0, 3 -> 2.0], 1.0, 4.2)
// return [1 -> 4.2, 2 -> 4.2, 3 -> 2]
```

Parameters:

- *map* input map
- *value1* value to be replaced with the new value

- *value2* new value

Throws:

- *NullParameterError* if a mandatory parameter is not specified

retainAll(collection∗ : Collection<E>, collections∗ : Collection<E>...) : Collection<E> Returns a collection of the elements from the input collection that are contained in each of the *collections*. The relative order of the retained elements remains unchanged. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
retainAll([1,1,2,3], [1,3,4,5], [1,1,1,3,4,5]);
// returns [1,1,3]
retainAll({1,2,3}, [1,2,3,4,5], [1,3,4])
//returns  {1,3}
```

Parameters:

- *collection* input collection
- *collections* collection with elements that should be added to the output collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

retainAll(list∗ : List<E>, collections∗ : Collection<E>...) : List<E> Returns a list of the elements from the input list that are contained in each of the *collections*. The relative order of the retained elements remains unchanged.

```
retainAll([1,1,2,3], [1,3,4,5], [1,1,1,3,4,5]);
// returns [1,1,3]
```

Parameters:

- *list* input list
- *collections* collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

retainAll(set∗ : Set<E>, collections∗ : Collection<E>...) : Set<E> Returns a set created by the intersection all the collections, the input *set* and the *collections*. The relative order of the retained elements remains unchanged.

```
retainAll({1,2,3}, [1,2,3,4,5], {1,3,4})
//returns  {1,3}
```

Parameters:

- *set* input set
- *collections* collections

Throws:

- *NullParameterError* if a mandatory parameter is not specified

reverse(collection∗ : Collection<E>) : Collection<E> Reverses the order of the elements in the input collection and returns them in a new collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
reverse([1,1,2.1,2.1,3,4.2])
 // returns [4.2,3,2.1,2.1,1,1];
 reverse({1,1,2.1,3,4.2})
 // returns {4.2,3,2.1,1}
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

reverse(list∗ : List<E>) : List<E> Reverses the order of the elements in the input list and returns them in a new list.

```
reverse([1,1,2.1,2.1,3,4.2])
 // returns [4.2,3,2.1,2.1,1,1]
```

Parameters:

- *list* input list

Throws:

- *NullParameterError* if a mandatory parameter is not specified

reverse(set∗ : Set<E>) : Set<E> Reverses the order of the elements in the input set and returns them in a new set.

```
 reverse({1,1,2.1,3,4.2})
 // returns {4.2,3,2.1,1}
```

Parameters:

- *set* input set

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Collection<E> Returns a colection with those elements from the input collection for which the condition is true. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
select([1,1,2.1,3,4.2],{ d:Decimal -> d > 2})
// returns [2.1,3,4.2]
```

Parameters:

- *collection* input collection
- *condition* condition the collection elements must satisfy

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(collection∗ : Collection<Object>, type∗ : Type<E>) : Collection<E> Returns a collection with those input-collection elements that are of the given type. The `null` elements are excluded. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
select([1,1,2.1,3,4.2, null],Integer)
 // returns  [1,1,3]
select({1,2.1,3,4.2, null},Integer)
// returns {1,3}
```

Parameters:

- *collection* input collection
- *type* type of elements to return

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(list∗ : List<E>, condition∗ : {E : Boolean}) : List<E> Returns a list with those elements from the input list which meet the condition.

```
select([1,1,2.1,2.1,3,4.2],{ d:Decimal -> d > 2})
// returns [2.1,2.1,3,4.2]
```

Parameters:

- *list* input list
- *condition* condition the list elements must satisfy

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(list∗ : List<Object>, type∗ : Type<E>) : List<E> Returns a list with those input-list elements that are of the given type. The `null` elements are excluded.

```
select({1,2.1,3,4.2, null},Integer)
// returns {1,3}
```

Parameters:

- *list* input list
- *type* type of elements to return

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(map∗ : Map<K, V>, condition∗ : {K, V : Boolean}) : Map<K, V> Returns a map with those elements of the input map for which the condition closure return true

```
select([1 -> "a", 2 -> "b", 3 -> "b"], {k:Integer, v:String -> k > 1 and v =="b"})
// returns [2->"b",3->"b"]
```

Parameters:

- *map* input map
- *condition* condition to check

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(set∗ : Set<E>, condition∗ : {E : Boolean}) : Set<E> Returns a set with those elements from the input set which meet the condition.

```
select({1,1,2.1,3,4.2},{ d:Decimal -> d > 2})
 // returns {2.1,3,4.2}
```

Parameters:

- *set* input set
- *condition* condition the set elements must satisfy

Throws:

- *NullParameterError* if a mandatory parameter is not specified

select(set∗ : Set<Object>, type∗ : Type<E>) : Set<E> Returns a set with those input-set elements that are of the given type. The `null` elements are excluded.

```
select({1,2.1,3,4.2, null},Integer)
// returns {1,3}
```

Parameters:

- *set* input set
- *type* type of elements to return

Throws:

- *NullParameterError* if a mandatory parameter is not specified

setAt(collection∗ : Collection<E>, index∗ : Integer, element : E) : Collection<E> Replaces the element of the collection at the *index* position with the new element and returns the new collection. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

```
setAt([1,2,4], 2, 3) //returns [1,2,3]
```

Parameters:

- *collection* input collection
- *index* index of the element to replace
- *element* new element

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(collection))

setAt(list∗ : List<E>, index∗ : Integer, element : E) : List<E> Replaces the element of the list at the *index* position with the new element and returns the new list.

```
setAt([1,2,4], 2, 3) //returns [1,2,3]
```

Parameters:

- *list*
- *index*
- *element*

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(list))

setAt(set∗ : Set<E>, index∗ : Integer, element : E) : Set<E> Replaces the element of the set at the *index* position with the new element and returns the new set.

```
setAt([1,2,4], 2, 3) //returns [1,2,3]
```

Parameters:

- *set* input set
- *index* index of the element to replace
- *element* new element

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified index is out of range (index < 0 or index >= size(set))

size(collection∗ : Collection<Object>) : Integer Returns the number of elements in the collection.

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

size(map∗ : Map<Object, Object>) : Integer Returns the number of elements in the map.

Parameters:

- *map* input map

Throws:

- *NullParameterError* if a mandatory parameter is not specified

sort(collection∗ : Collection<Date>, ascending∗ : Boolean) : Collection<Date> Sorts the input-collection elements sorted in ascending or descending order and returns the resulting collection.

```
sort([d'2015-12-24 20:00:00.000', now()] , true)
//returns  [d'2015-12-24 20:00:00.000',d'2018-06-04 10:25:01.142']
```

Parameters:

- *collection* input collection
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(collection∗ : Collection<Decimal>, ascending∗ : Boolean) : Collection<Decimal> Sort the input-collection elements in ascending or descending order and returns the resulting collection.

```
sort([ 2e+12, 0, -1.2342e0, -4.2] , true) //returns [-4.2,-1.2342,0,2000000000000]
```

Parameters:

- *collection* input collection
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(collection∗ : Collection<Integer>, ascending∗ : Boolean) : Collection<Integer> Sorts the input-collection elements in ascending or descending order and returns the resulting collection.

```
sort([ 1, 0, 1, -4], true) //returns [-4,0,1,1]
```

Parameters:

- *collection* input collection
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(collection∗ : Collection<String>, ascending∗ : Boolean) : Collection<String> Sorts the input-collection elements in ascending or descending order and returns the resulting collection.

```
sort(["č", "4","aa", "#", "C", "c"], true) //returns ["#","4","C","aa","c","č"]
```

Parameters:

- *collection* input collection
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(collection∗ : Collection<E>, comparator∗ : {E, E : Integer}) : Collection<E> Sorts the collection elements according to the comparator closure and returns the resulting collection. The comparator defines the total ordering function: It takes two elements of the input collection and returns a *negative integer* when the first parameter is less than the second parameter, *zero* when the first parameter is equal to the second parameter, or a *positive integer* when the first parameter is greater than the second parameter. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

sort({ 100, 1, 123, 342}, { a:Integer, b:Integer -> if a > b then 1 elsif a=b then 0 else -2 end}) //returns {1,100,123,342}

Parameters:

- *collection* input collection
- *comparator* comparator closure

Throws:

- *NullParameterError* if a mandatory parameter is not specified

sort(list∗ : List<Date>, ascending∗ : Boolean) : List<Date> Sorts the input-list elements in ascending or descending order and returns the resulting list.

```
sort([d'2015-12-24 20:00:00.000', now()] , true)
//returns  [d'2015-12-24 20:00:00.000',d'2018-06-04 10:25:01.142']
```

Parameters:

- *list* input list
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(list∗ : List<Decimal>, ascending∗ : Boolean) : List<Decimal> Sorts the input-list elements in ascending or descending order and returns the resulting list.

```
sort([ 2e+12, 0, 0,-1.2342e0, -4.2] , true) //returns [-4.2,-1.2342,0,0,2000000000000]
```

Parameters:

- *list* input list
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(list∗ : List<Integer>, ascending∗ : Boolean) : List<Integer> Sorts the input-list elements in ascending or descending order and returns the resulting list.

```
sort([ 1, 0, 1, -4], true) //returns [-4,0,1,1]
```

Parameters:

- *list* input list
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(list∗ : List<String>, ascending∗ : Boolean) : List<String> Sorts the input-list elements in ascending or descending order and returns the resulting list.

```
sort(["č", "4","aa", "#", "C", "c", "aa"], true) //returns ["#","4","C","aa","aa","c","č"]
```

Parameters:

- *list* input list
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(list∗ : List<E>, comparator∗ : {E, E : Integer}) : List<E> Sorts the elements of the collection according to the comparator closure and returns the resulting list. The comparator defines the total ordering function: it takes two elements of the input collection and returns a *negative integer* when the first parameter is less than the second parameter, *zero* when the first parameter is equal to the second parameter, or a *positive integer* when the first parameter is greater than the second parameter.

```
sort( [4, 1, 3, 0, 2, 2], { a:Integer, b:Integer -> if a > b then 1 elsif a=b then 0 else -2 end})
//returns [0,1,2,2,3,4]
```

Parameters:

- *list* input list
- *comparator* comparator closure

Throws:

- *NullParameterError* if a mandatory parameter is not specified

sort(set∗ : Set<Date>, ascending∗ : Boolean) : Set<Date> Sorts the input-set elements sorted in ascending or descending order and returns the resulting set.

```
sort({d'2015-12-24 20:00:00.000', now(), now()} , true)
//returns  [d'2015-12-24 20:00:00.000',d'2018-06-04 10:25:01.142']
```

Parameters:

- *set* input set
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(set∗ : Set<Decimal>, ascending∗ : Boolean) : Set<Decimal> Sorts the input-set elements in ascending or descending order and returns the resulting set.

Parameters:

- *set* input set
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(set∗ : Set<Integer>, ascending∗ : Boolean) : Set<Integer> Sorts the input-set elements in ascending or descending order and returns the resulting set.

```
sort({ 1, 0, 1, -4, -4}, true) //returns {-4,0,1,1}
```

Parameters:

- *set* input set
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(set∗ : Set<String>, ascending∗ : Boolean) : Set<String> Sorts the input-set elements in ascending or descending order and returns the resulting set.

```
sort({"č", "4","aa", "aa","#", "C", "c", "aa"}, true)
//returns {"#","4","C","aa","c","č"}
```

Parameters:

- *set* input set
- *ascending* whether to sort in ascending or descending order

Throws:

- *NullParameterError* if a mandatory parameter is null

sort(set∗ : Set<E>, comparator∗ : {E, E : Integer}) : Set<E> Sort the elements of the collection according to the comparator closure and returns the resulting set. The comparator defines the total ordering function: it takes two elements of the input collection and returns a *negative integer* when the first parameter is less than the second parameter, *zero* when the first parameter is equal to the second parameter, or a *positive integer* when the first parameter is greater than the second parameter.

```
sort({ 4, 1, 3, 0, 2, 2}, { a:Integer, b:Integer -> if a > b then 1 elsif a=b then 0 else -2 end})
//returns {0,1,2,3,4}
```

Parameters:

- *set* input set
- *comparator* comparator closure

Throws:

- *NullParameterError* if a mandatory parameter is not specified

sortByKey(collection∗ : Collection<E>, keyExtractor∗ : {E : Object}, ascending∗ : Boolean, nullsFirst∗ : Boolean) : List<E>
Sorts the input-collection elements by the keys produced by the *keyExtractor* closure and returns the resulting collection. The keys must be comparable: they must be strings, decimals, integers, or dates. *ascending* defines whether the list elements are in natural or reverse order: When `true`, natural order is used. *nullsFirst* specifies whether `null` keys are considered smaller or greater in natural order: When `true`, `null` values are at the beginning of the list.

```
sortByKey(
  collection -> [10,3,2,13, null, 4],
  keyExtractor -> { i:Integer -> i },
  ascending -> true,
  nullsFirst -> true
) //returns [null, 2,3,4,10,13]
```

Parameters:

- *collection* input collection
- *keyExtractor* key extractor closure
- *ascending* sorting order
- *nullsFirst* ordering position of null values

Throws:

- *NullParameterError* if a mandatory parameter is null

subCollection(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer) : Collection<E>
Returns the part of the input collection which starts with the element on the *fromIndex* position and finishes at the element preceding the *toIndex*. If *fromIndex* is `null` or is less than 0, the sub-collection starts from the beginning of the collection. If *toIndex* is `null` or, equal or greater than the size of the collection, the sub-collection ends at the end of the input collection.

```
subCollection([0, 1, 2, 3, 4], -2, 10) // returns [0,1,2,3,4]
```

Parameters:

- *collection* input collection
- *fromIndex* index of the first element to include in the resulting collection
- *toIndex* index of the first element to exclude from the resulting collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if fromIndex > toIndex

subList(list∗ : List<E>, fromIndex : Integer, toIndex : Integer) : List<E> Returns the part of the input list which starts with the element on the *fromIndex* position and finishes at the element preceding the *toIndex*. If *fromIndex* is `null` or is less than 0, the sub-list starts from the beginning of the list. If *toIndex* is `null` or, equal or greater than the size of the list, the sub-list ends at the end of the input list.

```
subList([1,1,2,3,4, 5], 2, 4) //returns [2,3]
```

Parameters:

- *list* input list
- *fromIndex* index of the first element to include in the returned list
- *toIndex* index of the first element to exclude from the returned list

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if fromIndex > toIndex

subset(set∗ : Set<E>, fromIndex : Integer, toIndex : Integer) : Set<E> Returns the part of the input set which starts with the element on the *fromIndex* position and finishes at the element preceding the *toIndex*. If *fromIndex* is `null` or is less than 0, the sub-set starts from the beginning of the set. If *toIndex* is `null` or, equal or greater than the size of the set, the sub-set ends at the end of the input set.

```
subset({0, 1, 2, 3, 4}, null, 3) // returns {0,1,2}
```

Parameters:

- *set* input set
- *fromIndex* index of the first element to include in the resulting set
- *toIndex* index of the first element to exclude from the resulting set

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if fromIndex > toIndex

sum(collection∗ : Collection<E>) : E Returns the sum of the decimals in the given collection.

```
sum( [1,3.2]) //returns 4.2
```

Parameters:

- *collection* collection of Decimals

Throws:

- *NullParameterError* if a mandatory parameter is not specified

swap(list∗ : List<T>, index1∗ : Integer, index2∗ : Integer) : List<T> Swaps the elements at the given indexes in the input list and returns the resulting list. s swap ([1, 3, 2], 1, 2) // returns [1,2,3]

Parameters:

- *list* input list
- *index1* index of the element to swap

- *index2* ndex of the element to swap

Throws:

- *NullParameterError* if a mandatory parameter is not specified

toList(collection∗ : Collection<E>) : List<E> Returns a list of elements of the input collection arranged in an arbitrary order.

```
toList([1,1,2,3,3]) ///returns [1,1,2,3,3]
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

toSet(collection∗ : Collection<E>) : Set<E> Returns a set with the elements of the input collection.

```
toSet([1 , 1, 2]) // returns {1,2}
```

```
toSet({1 , 1, 2}) // returns {1,2}
```

Parameters:

- *collection* input collection

Throws:

- *NullParameterError* if a mandatory parameter is not specified

values(map∗ : Map<K, V>) : List<V> Returns a list of values in the input map.

```
values([1 -> "b", 2 -> "c"]) // returns ["b","c"]
```

Parameters:

- *map* input map

Throws:

- *NullParameterError* if a mandatory parameter is not specified

8.4.3 Binary

createBinaryHandle(fileName∗ : String, description : String, content∗ : String, mime : String, charset : String) : BinaryHand
Creates a persisted binary handle that represents a file. If the mime type is not specified, it is determined based on the file extension. If no charset is defined, UTF-8 is used.

Parameters:

- *fileName* filename of the binary handle
- *description* description of the binary handle
- *content* content of the binary handle saved in binary
- *mime* mime type
- *charset* charset

Throws:

- *NullParameterError* if a mandatory parameter is null
- *UnsupportedEncodingError* if the specified charset is not supported

getBinaryHandle(id∗ : Integer) : BinaryHandle[DEPRECATED] Returns a binary handle with the identifier.

Parameters:

- *id* identifier of the binary handle

Throws:

- *NullParameterError* if a mandatory parameter is null
- *BinaryHandleNotFoundError* if there is no such a binary handle

getBinaryHandleMetadata(binaryData∗ : BinaryHandle) : Map<String, String> Returns a binary handle metadata.

Parameters:

- *binaryData* binary handle

Throws:

- *NullParameterError* if a mandatory parameter is null

getResource(module∗ : String, path∗ : String) : File Creates a file from the resource of a module of the model. If a model is based on module A and module A imports modules B and C, module B can access resourceses from C.

```
def File myFile := getResource("my_ imported_module", "output/myfile.txt")
```

Parameters:

- *module* name of the model module with the resource
- *path* relative path to the file from the module root

Throws:

- *NullParameterError* if a mandatory parameter is null
- *ResourceNotFoundError* if the module is not available or the resource is not found at the given path

resource(module∗ : String, path∗ : String) : BinaryHandle[SIDE EFFECTDEPRECATED] Creates a binary handle from a module resource located at the specified path

Parameters:

- *module*
- *path*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *ResourceNotFoundError* if the specified module is not available or the resource is not found at the given path

size(binary : Binary) : Integer Returns the size of the binary in bytes.

```
def Integer contentSize : = size(myBinaryHolder.content)
```

Parameters:

- *binary* binary

toBinary(string : String) : Binary Converts the string to binary.

Parameters:

- *string* string to convert

toString(binary∗ : Binary, charset : String) : String Converts binary to a string encoded in the given charset. If no charset is defined, UTF-8 is used. The supported charsets are defined in java.nio.charset.Standard↩ Charsets and include US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16, etc.

Parameters:

- *binary* binary to encode
- *charset* target charset

Throws:

- *NullParameterError* if a mandatory parameter is null
- *UnsupportedEncodingError* if the encoding is not supported

8.4.4 Enumeration

literalToName(literal∗ : Enumeration) : String Transforms the input enumeration literal to its name represented as a string

```
literalToName(Level.BEGINNER).toLowerCase()
//returns "beginner"
```

Parameters:

- *literal* input enumeration literal to transform

Throws:

- *NullParameterError* if mandatory parameter is null

literals(enumeration∗ : Type<E>) : List<E> Returns the list of literals of the input enumeration.

```
literals(Level)
//returns [common::Level.BEGINNER, common::Level.INTERMEDIATE, common::Level.ADVANCED]
```

Parameters:

- *enumeration* input enumeration

Throws:

- *NullParameterError* if mandatory parameter is null

nameToLiteral(enumeration∗ : Type<E>, name∗ : String) : E Transforms the input name to the corresponding literal value of the input enumeration.

```
nameToLiteral(Level, "BEGINNER")
//returns the enumeratation literal common::Level.BEGINNER
```

Parameters:

- *enumeration* input enumeration
- *name* input name to transform

Throws:

- *NullParameterError* if mandatory parameter is null
- *DoesNotExistError* if the specified name does not corespond to any literal

toEnumeration(string∗ : String) : Enumeration Transforms the input enumeration name to the corresponding enumeration value. The enumeration name must be in the following format: MODULE_NAME::ENUMERA↩ TION_NAME.LITERAL_NAME

```
toEnumeration("common::Level.BEGINNER")
//returns the enumeration literal common::Level.BEGINNER
```

Parameters:

- *string* input enumeration name

Throws:

- *NullParameterError* if a mandatory parameter is null or empty string
- *FormatError* if the string is not a valid enumeration full name

8.4.5 Label

getLabel(literal∗ : Enumeration) : String Returns the enumeration *literal* label.

Parameters:

- *literal* enumertation literal

Throws:

- *NullParameterError* if mandatory parameter is null

getLabel(property∗ : Property) : String Returns the property label of the record or interface *property*.

```
getLabel(Registration.firstName)
```

Parameters:

- *property* record or interface property

Throws:

- *NullParameterError* if mandatory parameter is null

getLabel(type∗ : Type<Object>) : String Returns the label of the record, enumeration, or interface *type*.

```
getLabel(RegistrationData)
```

Parameters:

- *type* record, enumeration, or interface

Throws:

- *NullParameterError* if mandatory parameter is null

8.4.6 Pdf

fillPdfTemplate(pdfTemplate∗ : Binary, fdfDocument∗ : Binary) : Binary Populates the PDF form template with the values from the FDF document and returns the resulting PDF.

Parameters:

- *pdfTemplate*
- *fdfDocument*

fillPdfTemplate(pdfTemplate∗ : Binary, values : Map<Object, Object>) : Binary Populates the PDF form template with the values specified by the *values* parameter using the FDF merging method and returns the resulting PDF document. The parameter names and values are specified by the *parameters* map.

Parameters:

- *pdfTemplate* PDF template
- *values* names and values for the PDF form template

fillXfaPdfTemplate(pdfTemplate∗ : Binary, data∗ : Record) : Binary Returns a PDF document obtained by populating the XFA template with the specified data. The type of populated record must specify the XML mapping that conforms to the XML schema required by the template.

Parameters:

- *pdfTemplate* XFA template
- *data* input data

fillXfaPdfTemplate(pdfTemplate∗ : Binary, xml∗ : String) : Binary Populates the XFA template with the specified data and returns the resulting PDF document. The input data must conform to the XML schema required by the template.

Parameters:

- *pdfTemplate* PDF template
- *xml* input data

8.4.7 Validation

findTag(name∗ : String) : Tag Finds a tag with the given *name* which can be a simple name or full name. Returns `null` if the tag is not found. If the simple name is given and it is ambiguous (more validation tags with the name exist in different imported modules), an error is thrown.

Parameters:

- *name* name of the tag

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *AmbiguousNameError* if the given name is ambiguous

validate(record : Record, property : Property, tags : Collection<Tag>, context : Map<String, Object>) : List<ConstraintVio
Validates the given record property against its constraints with the given *tags*. If the record is null, an empty list is returned. If the *property* parameter is null, all properties of the record are validated.

```
validate(new Simple(field -> "a"), null, null, null)
```

Parameters:

- *record* input record
- *property* record property to validate
- *tags* tags
- *context* input for complex constraints

Throws:

- *IncompatibleTypeError* if the given property is not a property of the given record

validate(record : Collection<Record>, property : Property, tags : Collection<Tag>, context : Map<String, Object>) : List<
Validates the given *property* in all records in the collection against the property constraints with the given *tags*. If the collection is null, an empty list is returned. Null elements of the collection are ignored. If the *property* parameter is null, all properties of the records are validated.

Parameters:

- *record* collection of records
- *property* property to validate
- *tags* tags used
- *context* context data needed by the validation

Throws:

- *IncompatibleTypeError* if the given property is not a property of any given record in the collection

8.4.8 String

contains(string∗ : String, substring∗ : String) : Boolean Returns `true` if the input *string* contains the *substring*.

```
"hello".contains("he") //returns true
```

Parameters:

- *string* input string
- *substring* substring to search for

find(string∗ : String, regexp∗ : String) : List<String> Finds all occurrences of the *regexp* pattern in the specified *string* and returns them as a list of strings. If no string is matched, the function returns an empty list. The Java regular expression syntax as defined in the java.util.regex.Pattern class is used.

```
find("one two three fourteen", "([\w]+)")
//returns ["one", "two", "three", "fourteen"]
```

Parameters:

- *string* input string
- *regexp* regex pattern

Throws:

- *NullParameterError* if mandatory parameter is null

format(pattern∗ : String, arguments : Object...) : String Formats the objects in the *arguments* parameter using the format in the *pattern* parameter and returns the resulting string. The pattern uses the syntax defined in the java.util.Formatter class restricted to the formatting commands that apply to the Java data types used to represent the Expression Language data types. The *arguments* data type are transformed according to the following table:

Boolean -> java.lang.Boolean String -> java.lang.String Date -> java.util.Date Integer -> java.math.Big↩ Decimal Decimal -> java.math.BigDecimal other data types -> java.lang.String using the function core::to↩ String

The format() function uses the server locale for formatting numbers and dates.

```
format( "The boolean is 100%% %2$s, %1$s !",  "John", false);
 //returns "The boolean is 100% false, John !"
format( "Today is %tB %tm", now(), now());
 //returns "Today is June 06"
```

Parameters:

- *pattern* pattern to process
- *arguments* arguments for the pattern

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FormatError* if the specified format is incorrect

formatWithLocale(pattern∗ : String, locale : String, arguments : Object...) : String Formats the objects in the *arguments* parameter using the format in the *pattern* parameter and returns the resulting string. The pattern uses the syntax defined in the java.util.Formatter class restricted to the formatting commands that apply to the Java data types used to represent the Expression Language data types. The *arguments* data type are transformed according to the following table:

Boolean -> java.lang.Boolean String -> java.lang.String Date -> java.util.Date Integer -> java.math.Big↩ Decimal Decimal -> java.math.BigDecimal other data types -> java.lang.String using the function core::to↩ String

The *locale* parameter determines the locale used for formatting numbers and dates. If it is null, the server locale is used.

```
 formatWithLocale("%tB", "en_US", now());
//returns "June"
formatWithLocale("%tB.", "it", now());
//returns giugno
```

Parameters:

- *pattern* pattern to format
- *locale* target locale
- *arguments* arguments for the pattern

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FormatError* if the specified format is incorrect

indexOf(string∗ : String, substring∗ : String) : Integer Returns the index of the first occurrence of the specified substring within the string.

```
indexOf("abcabc", "b");//returns 1
indexOf("abcasbc", "ca") //returns 2
```

Parameters:

- *string* string to search in
- *substring* substring to search for

Throws:

- *NullParameterError* if mandatory parameter is null

indexOf(string∗ : String, substring∗ : String, fromIndex∗ : Integer) : Integer Returns the index of the first occurrence of the specified *substring* within the input *string* starting from the specified *index*.

```
indexOf("abcdefghabcdefgh", "ab", 2)
//returns 8
```

Parameters:

- *string* input string
- *substring* string to search for
- *fromIndex* start search at this index

Throws:

- *NullParameterError* if mandatory parameter is null
- *OutOfBoundsError* if the index is out of range (index $<$ 0 or index $>=$ length(string))

isBlank(string : String) : Boolean Returns true if the string is null, has zero length (contains no characters) or only contains only whitespaces.

```
"   ".isBlank() //returns true
```

Parameters:

- *string* input string

isEmpty(string : String) : Boolean Returns true if the string is either null or has zero length (contains no characters).

```
"   ".isEmpty(); //returns false
"".isEmpty() //returns true
```

Parameters:

- *string* input string

lastIndexOf(string∗ : String, substring∗ : String) : Integer Returns the index of the rightmost occurrence of the specified *substring* in the input *string*.

```
lastIndexOf("ababab", "a") //returns 4
```

Parameters:

- *string* input string
- *substring* substring to search for

Throws:

- *NullParameterError* if mandatory parameter is null

lastIndexOf(string∗ : String, substring∗ : String, fromIndex∗ : Integer) : Integer Returns the index of the last occurrence of the specified substring within the string searching backward starting at the specified index.

```
lastIndexOf("aabba", "a", 3) //returns 1
```

Parameters:

- *string* input string
- *substring* substring to search for
- *fromIndex* start backward from this index

Throws:

- *NullParameterError* if mandatory parameter is null
- *OutOfBoundsError* if the index is out of range (index $<$ 0 or index $>=$ length(string))

length(string∗ : String) : Integer Returns the length of the specified *string*.

```
"123 45".length() //returns 6
```

Parameters:

- *string* input string

Throws:

- *NullParameterError* if a mandatory parameter is not specified

localize(string∗ : String, language∗ : String) : String Replaces all localizable substrings in the string parameter by the corresponding localized strings from the specified language.

```
localize($save_button() + ", " + $load_button(), "EN")
//returns "Save, Load"
```

Parameters:

- *string* string to localize
- *language* language code

Throws:

- *NullParameterError* if a mandatory parameter is not specified

matches(string∗ : String, regexp∗ : String) : Boolean Returns true if the specified string matches the regular expression given by the regexp parameter, otherwise returns false. The Java regular expression syntax as defined in the java.util.regex.Pattern class is used.

```
"abccccD".matches("\a[^h]c{3}.*")
//returns true
```

Parameters:

- *string* input string

- *regexp* regex pattern

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replaceAll(string∗ : String, regexp∗ : String, replacement∗ : String) : String Replaces each occurence of the *regexp<i> parameter substring in the specified <i>string* with the *replacement* parameter. The Java regular expression syntax as defined in the java.util.regex.Pattern class is used.

```
replaceAll("replace replace me", "w++", "please ")
//returns "please please please"
```

Parameters:

- *string* input string
- *regexp* regex pattern
- *replacement* replace matches with this string

Throws:

- *NullParameterError* if a mandatory parameter is not specified

replaceFirst(string∗ : String, regexp∗ : String, replacement∗ : String) : String Replaces the first occurrence of the *regexp* parameter in the *string* parameter with the string specified by the *replacement* parameter and returns the resulting string. The Java regular expression syntax as defined in the java.util.regex.Pattern class is used.

```
replaceFirst("replace replace me", "w++", "Please ")
//returns "Please replace me"
```

Parameters:

- *string* input string
- *regexp* regex pattern
- *replacement* replace with this string

Throws:

- *NullParameterError* if a mandatory parameter is not specified

split(string∗ : String, regexp∗ : String) : List<String> Splits the *string* at the positions that match the regular expression. The Java regular expression syntax as defined in the java.util.regex.Pattern class is used.

```
split("one two three", "s")
//returns ["one", "two", "three"]
```

Parameters:

- *string* input string
- *regexp* regular expression

Throws:

- *NullParameterError* if a mandatory parameter is not specified

substring(string∗ : String, fromIndex : Integer, toIndex : Integer) : String Returns a substring of the input *string* starting at the *fromIndex* position (inclusive) and ending at the *toIndex* position (exclusive). If the fromIndex is null or is less than 0, it is considered 0. If the toIndex is null, equal, or greater than the length of the string, the substring ends at the end of the string (index given by length(string)-1).

```
substring("0123Hello World!", 4, 20 )
 //returns "Hello World!"
```

Parameters:

- *string* input string

- *fromIndex* substring's start index
- *toIndex* substring's end index

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if fromIndex $>$ toIndex

toDecimal(string$*$: String) : Decimal If possible, converts the specified string to a decimal. If not possible, the function throws a runtime exception.

```
toDecimal("4.2") //returns 4.2
```

Parameters:

- *string* string with a decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FormatError* if the string does not represent a decimal value

toInteger(string$*$: String) : Integer If possible, converts the specified *string* to an integer. If not possible, the function throws a runtime exception.

```
"123 45".toInteger();
//returns FormatError
"12345".toInteger() //returns 12345
```

Parameters:

- *string* string with integer value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FormatError* if the string does not represents an integer value

toInteger(string$*$: String, radix$*$: Integer) : Integer Converts the integer in the specified *string* to an integer using the specified *radix*. The radix must be between 2 and 36 (inclusive). If not possible, the function throws a runtime exception.

```
toInteger("15",36) //returns 41
```

Parameters:

- *string* string with an integer
- *radix* applied radix

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FormatError* if the string does not represents an integer value

toLowerCase(string$*$: String) : String Converts all characters of the specified string to lowercase.

```
toLowerCase("Hello World!");
toLowerCase("Hello World!");
"Hello World!".toLowerCase()
```

Parameters:

- *string* input string

Throws:

- *NullParameterError* if mandatory parameter is null

toString(object : Object) : String Converts the specified object to a string.

```
toString(findById(Model, 1))
```

Parameters:

- *object* object to convert to string

toUpperCase(string∗ : String) : String Converts all characters of the specified string to uppercase.

```
toUpperCase("Hello World!") //returns "HELLO WORLD!"
```

Parameters:

- *string* input string

Throws:

- *NullParameterError* if mandatory parameter is null

trim(string∗ : String) : String Eliminates leading and trailing spaces from the input *string* and returns the resulting string.

```
" hello world!   ".trim() //returns "hello world!"
```

Parameters:

- *string* input string

Throws:

- *NullParameterError* if a mandatory parameter is not specified

8.4.9 Xml

convertToXml(object∗ : Object) : String Converts internal data structures passed to the object parameter to X↩ ML document. The object parameter must be an instance of a record type, otherwise an exception is thrown. The function returns a string which represents the output XML document.

```
convertToXml(new MyRecord(item -> "value"))
```

Parameters:

- *object* record to convert to XML

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the type of the object is not record
- *UnableToConvertToXmlError* if an error occurred during conversion

parseXml(xml∗ : String, resultType∗ : Type<E>) : E **SIDE EFFECT** Parses *xml* to *resultType* record and returns the result in the *resultType* record.

```
parseXml(
   "<?xml version=""1.0"" encoding=""UTF-8""?>
     <ns0:ParsedXml xmlns:ns0=""restCall"" xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"">
       <ns0:item>value</ns0:item>
      </ns0:ParsedXml>",
ParsedXml)
```

Parameters:

- *xml* XML to parse to the record type
- *resultType* record type

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the resultType parameter is not record
- *UnableToParseXmlError* if the specified xml cannot be parsed

8.4.10 Reflection

achieved() : String[DEPRECATED] Returns a string representing "Achieved" state. Deprecated. Replaced by constant ACHIEVED.

activate(goals : Collection<Goal>) : void[SIDE EFFECT] Activates inactive and finished *goals*. If a goal from the collection is active, the goal status remains unchanged.

Parameters:

- *goals* goals to activate

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FinishedProcessInstanceError* if the process instance of a goal is finished

active() : String[DEPRECATED] Returns a string representing "Active" state. Deprecated. Replaced by constant ACTIVE.

alive() : String[DEPRECATED] Returns a string representing "Alive" state. Deprecated. Replaced by constant ALIVE.

deactivate(goals : Collection<Goal>) : void[SIDE EFFECT] Deactivates not finished *goals*. If a goal from the collection is finished or its goal's process is finished, the goal status remains unchanged.

Parameters:

- *goals* goals to deactivate

Throws:

- *NullParameterError* if a mandatory parameter is not specified

deactivated() : String[DEPRECATED] Returns a string representing "Deactivated" state. Deprecated. Replaced by constant DEACTIVATED.

failed() : String[DEPRECATED] Returns a string representing "Failed" state. Deprecated. Replaced by constant FAILED.

finished() : String[DEPRECATED] Returns a string representing "Finished" state. Deprecated. Replaced by constant FINISHED.

getAllSubGoals(goal∗ : Goal) : Set<AchieveGoal> Returns the set of all sub-achieve goals into which the specified *goal* is directly or indirectly decomposed.

```
getAllSubGoals(TripArranged)
```

Parameters:

- *goal* input achieve goal

getSubGoals(goal∗ : Goal) : Set<AchieveGoal> Returns the set of all direct sub-achieve goals into which the specified *goal* is decomposed.

```
getSubGoals(TripArranged)
```

Parameters:

- *goal* goal

inactive() : String[DEPRECATED] Returns a string representing "Inactive" state. Deprecated. Replaced by constant INACTIVE.

isInState(goalPlan∗ : GoalPlan, state : String) : Boolean Returns `true` if the goal or the plan defined by the *goalPlan* parameter is in the specified *state* or a sub-state of the state. If the *goalPlan* parameter is specified correctly but the *state* parameter contains an incorrect string representation of a state or is null, the function returns `false`.

```
Goal2.isInState(READY)
```

Parameters:

- *goalPlan* goal or plan to check the state of
- *state* state

Throws:

- *NullParameterError* if a mandatory parameter is not specified

notFinished() : String[DEPRECATED] Returns a string representing "Not finished" state. Deprecated. Replaced by constant NOT_FINISHED.

ready() : String[DEPRECATED] Returns a string representing "Ready" state. Deprecated. Replaced by constant R↩ EADY.

running() : String[DEPRECATED] Returns a string representing "Running" state. Deprecated. Replaced by constant RUNNING.

8.4.11 Type

getIndexType(mapType∗ : Type<Map<K, V>>) : Type<K>[DEPRECATED] Returns the key type of the given map type. Deprecated. Use getKeyType(Type<Map<K,V>>) instead.

Parameters:

- *mapType* input map type

Throws:

- *NullParameterError* if a mandatory parameter is null

getItemType(collectionType∗ : Type<Collection<E>>) : Type<E> Returns the item type of the given collection type.

```
getItemType(getType([1,2,3])),2}]))
//returns Set<Integer>
```

Parameters:

- *collectionType* collection type

Throws:

- *NullParameterError* if a mandatory parameter is null

getItemType(mapType∗ : Type<Map<K, V>>) : Type<V>[DEPRECATED] Returns the value type of the given map type. Deprecated. Use getValueType(Type<Map<K,V>>) instead.

Parameters:

- *mapType* input map type

Throws:

- *NullParameterError* if a mandatory parameter is null

getKeyType(mapType∗ : Type<Map<K, V>>) : Type<K> Returns the key type of the given map type.

```
getKeyType(getType([now()-> 1, now() -> "a"]))
//returns Date
```

Parameters:

- *mapType* input map type

Throws:

- *NullParameterError* if a mandatory parameter is null

getReferencedType(referenceType∗ : Type<Reference<E>>) : Type<E> Returns the referenced type of the given reference type.

```
def Integer i;
getReferencedType(typeOf(&i))
//returns Integer
```

Parameters:

- *referenceType*

Throws:

- *NullParameterError* if a mandatory parameter is null

getSupertype(recordType∗ : Type<Record>) : Type<Object> Returns the supertype of the given record type.

Throws: -"NullParameterError" - Mandatory parameter is null.

Parameters:

- *recordType*

getValueType(mapType∗ : Type<Map<K, V>>) : Type<V> Returns the value type of the given map type.

```
getValueType(typeOf([1 -> "a"]))
//returns String
```

Parameters:

- *mapType* input map type

Throws:

- *NullParameterError* if a mandatory parameter is null

toType(string∗ : String) : Type<Object> If possible, parses the specified string to a type. If the given string does not represent a valid type, the function throws an error. <pre>toType("{Integer:Integer}"); //returns {Integer:Integer}

Parameters:

- *string*

Throws:

- *NullParameterError* if a mandatory parameter is not specified.
- *InvalidTypeError* if the string does not represent a valid type.

8.4.12 Number

abs(number∗ : Decimal) : Decimal Returns the absolute value of the number.

```
abs(-3.1) //returns 3.1
```

Parameters:

- *number* decimal number

Throws:

- *NullParameterError* if a mandatory parameter is not specified

abs(number∗ : Integer) : Integer Returns the absolute value of the number.

```
abs(-42) //returns 42
```

Parameters:

- *number* integer

Throws:

- *NullParameterError* if a mandatory parameter is not specified

max(numbers∗ : Decimal...) : Decimal Returns the greatest number from the listed decimals.

```
max(1.0, -22.4, 188, 0, 2E+5) //returns 200000
```

Parameters:

- *numbers* decimals

Throws:

- *NullParameterError* if a mandatory parameter is not specified

max(numbers∗ : Integer...) : Integer Returns the greatest number from the listed integers.

```
max(1, -22, 188, 0) //returns 188
```

Parameters:

- *numbers* integers

Throws:

- *NullParameterError* if a mandatory parameter is not specified

min(numbers∗ : Decimal...) : Decimal Returns the smallest number from the listed decimals.

```
min(1.1, 2, -3.55) //returns -3.55
```

Parameters:

- *numbers* list of decimals

Throws:

- *NullParameterError* if a mandatory parameter is not specified

min(numbers∗ : Integer...) : Integer Returns the smallest number from the listed integers.

```
min(1, 2, -3) //returns -3
```

Parameters:

- *numbers* integers

Throws:

- *NullParameterError* if a mandatory parameter is not specified

random() : Decimal Returns a pseudorandom, uniformly distributed decimal value between 0.0 (inclusive) and 1.0 (exclusive).

random(upperBound∗ : Integer) : Integer Returns a pseudorandom, uniformly distributed integer between 0 (inclusive) and the specified upperBound (exclusive).

```
random(10)
```

Parameters:

- *upperBound*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundCeiling(number∗ : Decimal) : Integer Returns rounding of the specified number towards positive infinity. If the number is positive, rounds up; if the number is negative, rounds down. It does not decrease the number value.

```
roundCeiling(5.5);//returns 6
roundCeiling(-5.5)//returns -5
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundDown(number∗ : Decimal) : Integer Returns rounding of the specified number towards zero. It never increases the magnitude of the number.

```
roundDown(5.5); //returns 5
roundDown(1.6); //returns 1
roundDown(-1.1) //returns -1
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundFloor(number∗ : Decimal) : Integer Returns rounding of the specified number towards negative infinity. If the number is positive, rounds down. If the number is negative, rounds up.

```
roundFloor(5.55);//returns 5
roundFloor(-5.55)//returns -6
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundHalfDown(number∗ : Decimal) : Integer Returns rounding of the specified number towards the "closest neighbor"; if both neighbors are equidistant, rounds down.

```
roundHalfDown(5.4); //returns 5
roundHalfDown(5.5); //returns 5
roundHalfDown(-5.6); //returns -6
roundHalfDown(-5.4); //returns -5
roundHalfDown(-5.5); //returns -5
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundHalfEven(number∗ : Decimal) : Integer Returns rounding of the specified number towards the "closest neighbor"; if both neighbors are equidistant, rounds towards the even neighbor.

```
roundHalfEven(4.5); //returns 4
roundHalfEven(5.5); //returns 6
roundHalfEven(-5.5); //returns -6
roundHalfEven(-6.5); //returns -6
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundHalfUp(number∗ : Decimal) : Integer Returns rounding of the number towards the "closest neighbor"; if both neighbors are equidistant, it rounds up.

```
roundHalfUp(5.4); //returns 5
roundHalfUp(5.5); //returns 6
roundHalfUp(-5.6); //returns -6
roundHalfUp(-5.4); //returns -5
roundHalfUp(-5.5) //returns -6
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

roundUp(number∗ : Decimal) : Integer Returns rounding of the specified number away from zero; it never decreases the magnitude of the number.

```
roundUp(2.5) ;//returns 3
roundUp(-1.1) //returns -2
```

Parameters:

- *number* input decimal number

Throws:

- *NullParameterError* if a mandatory parameter is not specified

scale(number∗ : Decimal) : Integer Returns the scale, the number of decimal places, of the *number*.

```
scale(2.53)//returns 2
```

Parameters:

- *number* input decimal

Throws:

- *NullParameterError* if a mandatory parameter is not specified

setScale(number∗ : Decimal, scale∗ : Integer, roundingMode∗ : {Decimal : Integer}) : Decimal Returns a decimal with the specified scale, and the unscaled value of which is determined by multiplying or dividing the unscaled value of the number by the appropriate power of ten to maintain its overall value. If the scale is reduced by the operation, the unscaled value must be divided and the specified roundingMode is applied to the division. Formally, it can be expressed as roundingMode(number ∗ (10 ∗∗ scale)) / (10∗∗ scale).

Parameters:

- *number*
- *scale*
- *roundingMode*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

8.4.13 Model

createModelInstance(synchronous∗ : Boolean, model∗ : Model, properties : Map<String, String>) : ModelInstance SIDE EFFEC

Finds the specified model and starts a new instance of the specified *model* with the initialization *properties*, and returns the new model instance. If *synchronous* is set to true, the call waits until the new model instance becomes "started", that is, its first transaction is executed sucessfully. If this does not occur, the transaction is rolled back. If *synchronous* is false, the execution continues when the model instance becomes "created". Note that in the case of a roll-back, the data of the new model instance is deleted. It is recommended to set *synchronous* to true when calling from documents and to false when calling from processes.

```
createModelInstance(
 synchronous -> false,
 model -> getModel("registration","1.0"),
 properties-> ["Triggered by" -> thisModelInstance().id.toString()]
)
```

Parameters:

- *synchronous* whether the containing expression waits for the created model instance
- *model* model of the created model instance
- *properties* properties of the created model instance

Throws:

- *NullParameterError* if a mandatory parameter is null
- *ModelInstantiationError* if the model instance did not start due to an internal initiation error
- *ModelInterpretationError* if the model instance did not start due to an internal model interpretation (startup) error

createModelInstance(synchronous∗ : Boolean, model∗ : Model, processEntity∗ : Record, properties : Map<String, String>)

Finds the specified model and starts a new instance of the specified *model* with the initialization *properties* and the *process entity*, and returns the new model instance. If *synchronous* is set to true, the call waits until the new model instance becomes "started", that is, its first transaction is executed sucessfully. If this does not occur, the transaction is rolled back. If *synchronous* is false, the execution continues when the model instance becomes "created". Note that in the case of a roll-back, the data of the new model instance is deleted. It is recommended to set *synchronous* to true when calling from documents and to false when calling from processes. The process entity must be a shared record: the string representation of its primary key is added to the initialization properties.

Parameters:

- *synchronous* whether the containing expression waits for the model instance to be created
- *model* model of the created model instance
- *processEntity* shared record which is the process entity
- *properties* properties of the created model instance

Throws:

- *NullParameterError* if a mandatory parameter is null
- *ModelInstantiationError* if the model instance could not be created due to an internal initiation error
- *ModelInterpretationError* if the model instance could not be created due to an internal model interpretation (startup) error
- *IncompatibleTypeError* if the processEntity is not a shared record

findModelInstances(model : String, version : String, isRunning : Boolean, properties : Map<String, String>) : Set<ModelIn

Returns the set of model instances satisfying all specified criteria. All String parameters, including the properties parameter, support the following wildcards: "∗" matching none or several characters, and "?" matching exactly one character. Any other character in patterns matches itself. The value of the *isRunning* parameter must match exactly. null specified as the pattern value matches any value of the model instance field and is equivalent to the "∗" pattern.

```
findModelInstances(
   model -> "registration*",
   version -> "2.?",
   isRunning -> true,
   properties -> null
)
```

Parameters:

- *model* underlying model
- *version* version of the underlying model
- *isRunning* whether to return only running instances
- *properties* properties the model instances must have

findModels(name : String, version : String, latestOnly : Boolean) : Set<Model> Returns a set of uploaded models satisfying the specified filtering criteria. All Strings parameters support the following wildcards: "*" matching none or several characters, and "?" matching exactly one character. Any other character in patterns matches itself. The value of the *isRunning* parameter must match exactly. `null` specified as the pattern value matches any value of the model and is equivalent to the "*" pattern. When the *latestOnly* parameter is `true`, only the most recently uploaded models with the given name and version are returned if multiple models with the same name and version are stored on the server. When set to `false`, the returned set can contain several models with the same name and version.

```
findModels("registration", "1.0", false)
```

Parameters:

- *name* model name
- *version* model version
- *latestOnly* whether to return only the last uploaded model in the version

findProcessInstances(filter : {ProcessInstance : Boolean}) : Set<ProcessInstance> Returns the set of process instances of the current model instance that satisfy the *filter* criteria. The filter criteria is specified as a Boolean closure with one ProcessInstance parameter. If the Boolean expression returns true for a process instance, the process instance is included in the resulting set. If *filter* is `null`, all process instances are returned.

```
findProcessInstances(filter -> {p:ProcessInstance -> p.process == "approval"})
```

Parameters:

- *filter* filter closure

Throws:

- *NullParameterError* if a mandatory parameter is not specified.

findProcessInstances(modelInstance : ModelInstance, filter : {ProcessInstance : Boolean}) : Set<ProcessInstance> Returns the set of process instances generated by the *model instance* that satisfy the *filter* criteria. The filter criteria is specified as a Boolean closure with one ProcessInstance parameter. If the Boolean expression returns true for a process instance, the process instance is included in the resulting set. If *filter* is `null`, all process instances are returned.

WARNING: The execution of this function can be slow if accessing a model instance which is not loaded in memory.

```
findProcessInstances(
  modelInstance -> getModelInstance(8015),
  filter -> {p:ProcessInstance -> p.process == "approval"}
)
```

Parameters:

- *modelInstance* model instance
- *filter* filter closure

Throws:

- *NullParameterError* if mandatory parameters are not specified.

getModel(name∗ : String) : Model Returns the model with the specified name. If multiple models with the same name and any version are uploaded on the server, the most recently uploaded model is returned. Unloaded models are ignored.

```
getModel("registration")
```

Parameters:

- *name* string with model name

Throws:

- *NullParameterError* if mandatory parameters are not specified.
- *ModelNotFoundError* if there is no such a model.

getModel(name∗ : String, version∗ : String) : Model Returns a model with the specified name and version. If multiple models with the same name and any version are uploaded on the server, the most recently uploaded model is returned.

```
getModel("registration", "2.3")
```

Parameters:

- *name* string with model name
- *version* string with version

Throws:

- *NullParameterError* if mandatory parameters are not specified.
- *ModelNotFoundError* if there is no such a model.

getModelInstance(id∗ : Decimal) : ModelInstance Returns the model instance with the id.

```
getModelInstance(8013)
```

Parameters:

- *id* id of the model instance

Throws:

- *NullParameterError* if mandatory parameters are not specified.
- *ModelInstanceNotFoundError* if there is no such a model instance.

getModelInstanceProperties(modelInstance∗ : ModelInstance) : Map<String, String> Returns the map of initialization properties of the specified model instance.

```
getModelInstanceProperties(getModelInstance(8014))
```

Parameters:

- *modelInstance* model instance

Throws:

- *NullParameterError* if mandatory parameter is null.

getProcessEntity(type∗ : Type<T>) : T Returns the process entity of the specified type of the model instance. Process entity information, a string representation of its primary key, is stored in the model instance properties. The expected type must be specified and must be the shared record type which was used to create the model instance. If the types do not match, the behavior is unspecified. See the createModelInstance(Boolean synchronous, Model model, Record processEntity, Map<String,String> properties).

```
getProcessEntity(RegistrationData)
```

Parameters:

- *type* type of the process entity

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if type is not a shared record type

notifyModelInstance(modelInstanceId∗ : Decimal, reason : String, synchronously∗ : Boolean) : void
Invokes (wakes up) the model instance. The call is used, for example, when the data of the model instance were changed by an external service or by a direct change in the database. If synchronous, the notification call waits until the invocation of the model instance finishes.

Parameters:

- *modelInstanceId* model instance to notify
- *reason* log string
- *synchronously* whether to execute synchronously

Throws:

- *ErrorException* if fails, e.g. the model instance was not found, model instance state is invalid

sendSignal(synchronous∗ : Boolean, recipients∗ : Set<ModelInstance>, signal∗ : Object) : void[SIDE EFFECT]
Sends the *signal* object to the specified *recipients*. If *synchronous* is set to true, the execution waits until the signal is received and processed by its recipients. If *synchronous* is set to false, the execution continues immediately.

```
sendSignal(true, {getModelInstance(8014)}, "Dispatch cancel")
```

Parameters:

- *synchronous* whether the expression waits for signal processing to succeed
- *recipients* target model instances
- *signal* signal

Throws:

- *NullParameterError* if a mandatory parameter is null
- *SendSignalError* if sending or receiving of the signal was not successful

terminateModelInstance(modelInstance∗ : ModelInstance) : void Terminates a running model instance which becomes finished.

```
//example task parameters:
modelInstance -> getModelInstance(25009)
```

Parameters:

- *modelInstance* model instance

Throws:

- *NullParameterError* if a mandatory parameter is not specified

thisModel() : Model Returns the model of the context from which the function was executed.

thisModelInstance() : ModelInstance Returns the model instance that executed the function. Returns `null` when executed from a document.

thisModelInstanceProperties() : Map<String, String> Returns the map of the initialization properties of the model instance that executed the function. Returns `null` when executed from document. <pre>this↩ ModelInstanceProperties() //returns ["InitiatorId"->"admin", "my key"->"my value"]

thisModelInstanceProperty(propertyName∗ : String) : String Returns a value of the initialization property with the given name of the model instance that executed the function. Returns `null` if no property with the given name exists or if executed from a document.

```
thisModelInstanceProperty("InitiatorId")
```

Parameters:

- *propertyName* property name

Throws:

- *NullParameterError* if a mandatory parameter is not specified.

thisProcessInstance() : ProcessInstance Returns the process instance that executed the function or `null` when the function is executed in the global context of a module or from a document.

throwEscalation(code∗ : String, payload : Object) : void[SIDE EFFECT] Throws an escalation signal with the specified code and payload. The escalation signal can be processed This function can be called only from an expression enclosed in a process that can handle the escalation. Otherwise, the calling of this function has no effect.

```
throwEscalation("Desk head", thisModelInstance())
```

Parameters:

- *code* escalation code
- *payload* escalation payload

Throws:

- *NullParameterError* if a mandatory parameter is null

8.4.14 Date

∗(duration : Duration, number : Decimal) : Duration Multiplies each field of the duration by the specified number and rounds the value down, and returns the resulting duration.

```
(new Duration(minutes -> 2, seconds -> 1))*1.55
//returns core::Duration {hours=0, seconds=1, months=0, weeks=0, minutes=3, days=0, millis=0, years=0}
```

Parameters:

- *duration*
- *number*

∗(number : Decimal, duration : Duration) : Duration Multiplies each field of the duration by the specified number and rounds the value down, and returns the resulting duration.

```
1.7 * (new Duration(minutes -> 2, seconds -> 1))
 =//returns core::Duration {hours=0, seconds=1, months=0, weeks=0, minutes=3, days=0, millis=0, years=0}
```

Parameters:

- *number*
- *duration*

+(duration : Duration) : Duration Unary minus operator which returns the duration.

```
+seconds(-2)
//returns core::Duration {hours=0, seconds=-2, months=0, weeks=0, minutes=0, days=0, millis=0, years=0}
```

Parameters:

- *duration* duration

+(date : Date, duration : Duration) : Date Adds the duration to the input date and returns the resulting date.

```
d'2018-06-07 00:00:00.000' + minutes(42)
//returns d'2018-06-07 00:42:00.000
```

Parameters:

- *date* input date
- *duration* duration to add to the date

+(date∗ : LocalDate, duration∗ : Duration) : LocalDate Adds a duration to a given local date.

Parameters:

- *date*
- *duration*

+(duration : Duration, date : Date) : Date Adds the date to the duration and returns the resulting date.

```
minutes(42)+d'2018-06-07 00:00:00.000'
// returns d'2018-06-07 00:42:00.000'
```

Parameters:

- *duration* duration
- *date* date

+(duration1 : Duration, duration2 : Duration) : Duration Adds the values of the corresponding fields of duration1 and duration2, and returns the resulting duration.

```
seconds(100)+minutes(3)
//returns core::Duration {hours=0, seconds=100, months=0, weeks=0, minutes=3, days=0, millis=0, years=0}
```

Parameters:

- *duration1*
- *duration2*

-(duration : Duration) : Duration Negates each field of the specified duration and returns the resulting duration.

```
-(seconds(10)+days(2))
 //returns core::Duration {hours=0, seconds=-10, months=0, weeks=0, minutes=0, days=-2, millis=0, years=0
```

Parameters:

- *duration*

-(date : Date, duration : Duration) : Date Subtracts the duration from the date and returns the resulting date.

```
d'2018-06-07 00:00:42.000' - seconds(42)
//returns d'2018-06-07 00:00:00.000'
```

Parameters:

- *date*
- *duration*

-(date∗ : LocalDate, duration∗ : Duration) : LocalDate Subtracts a duration from a given local date.

Parameters:

- *date*
- *duration*

-(date1 : Date, date2 : Date) : Duration Subtracts date2 from date1 and returns the resulting duration. The duration abides by the maximum allowed values for individual fields; that is, 60 seconds are resolved as 1 minute. If date2 $>$ date1, the resulting duration is negative, i.e., all its fields are negative.

```
d'2018-06-07 00:00:63.000' - d'2010-06-07 00:00:01.000'
 //result core::Duration {hours=0, seconds=2, months=0, weeks=0, minutes=1, days=0, millis=0, years=8}
```

Parameters:

- *date1*
- *date2*

-(date1∗ : LocalDate, date2∗ : LocalDate) : Duration Returns a duration obtained by subtracting two dates.

Parameters:

- *date1*
- *date2*

-(duration1 : Duration, duration2 : Duration) : Duration Subtracts each field of duration2 from the corresponding field of duration1 and returns thhe resutling duration.

```
(seconds(10)+minutes(1))-(minutes(2)+seconds(84))
 //returns core::Duration {hours=0, seconds=-74, months=0, weeks=0, minutes=-1, days=0, millis=0, years=0
```

Parameters:

- *duration1*
- *duration2*

currentDate() : LocalDate Returns the current local date in the time zone of the LSPS Server.

date(epochMillis∗ : Integer) : Date Creates a date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", which is January 1, 1970, 00:00:00 GMT.

```
date(42)
//returns d'1970-01-01 00:00:00.042'
```

Parameters:

- *epochMillis* milliseconds to add to the epoch

Throws:

- *NullParameterError* if a mandatory parameter is not specified

date(string∗ : String) : Date Creates a date from the input string. The format of the input string is a subset of formats given by ISO 86011. It is possible to specify date with the format `\<date\>`, or time with the format `T\<time\>`, or to combine dates with times with the format `\<date\>T\<time\>`: The following formats of dates are allowed: yyyy, yyyy-MM, yyyy-MM-dd, yyyy-DDD, yyyy-Www, and yyyy-Www-e; where `yyyy` stands for the year, `MM` stands for the month (01-12), `dd` stands for the day in the month (01-31), `DDD` stands for the day in the year (001-365), `ww` stands for the week number and must be prefixed by the letter 'W' (W01-W53), and `e` stands for the day of the week (1-7). The following formats of time are allowed: HH, HH:mm, and HH:mm:ss; where `HH` stands for hour (00-23), `mm` stands for minutes (00-59), and `ss` stands for seconds (00-59). The following formats of time zones are allowed: `\<time\>Z` where 'Z' stands for U↩ TC, `\<time\>±HH`, `\<time\>±HH:mm`, and `\<time\>±HH:mm:ss`. Decimal fractions, separated either by comma or by dot, may also be added to hours, minutes or seconds.

```
date("2100-12-18");
 //returns d'2100-12-18 00:00:00.000'
date("2100T13:10");
//returns d'2100-01-01 13:10:00.000'
date("2100T13:10Z");
//returns d'2100-01-01 13:10:00.000';
date("2100T13:10+02")
 //returns d'2100-01-01 11:10:00.000';
 date("2100T13:10.555+02")
//returns d'2100-01-01 11:10:33.300'
```

Parameters:

- *string* input string with the date

Throws:

- *NullParameterError* if mandatory parameters are not specified

- *FormatError* if the string parameter does not conform to the specified format
- *OutOfBoundsError* if the string parameter does not represent a correct date

date(string∗ : String, pattern∗ : String) : Date Creates a date from the input string based on the format defined by the pattern parameter. The pattern defines the format as specified in the java.text.SimpleDateFormat Java class.

```
date("2100.07.04 AD at 12:08:56 PDT","yyyy.MM.dd G 'at' HH:mm:ss z");
//returns d'2100-07-04 19:08:56.000'
date("12:08 PM","h:mm a")
//returns d'1970-01-01 12:08:00.000'
```

Parameters:

- *string* input string with date
- *pattern* format used by the string date

Throws:

- *NullParameterError* if mandatory parameter is not specified
- *FormatError* if the pattern parameter does not conform to the specified format, or the string parameter does not conform to the pattern
- *OutOfBoundsError* if the string parameter does not represent a correct date

date(string∗ : String, pattern∗ : String, locale : String) : Date Creates a date from the specified string, which adheres to the format given by the pattern parameter. The pattern string uses the format defined in the java.text.SimpleDateFormat Java class. The function takes an optional locale parameter.

```
date("2100.07.04 AD at 12:08:56 PDT","yyyy.MM.dd G 'at' HH:mm:ss z", "en_US");
//returns d'2100-07-04 19:08:56.000'
```

Parameters:

- *string* input string with the date
- *pattern* format used by the string date
- *locale* optional locale name, e.g. "de_AT"

Throws:

- *NullParameterError* if mandatory parameter is not specified
- *FormatError* if the pattern parameter does not conform to the specified format, or the string parameter does not conform to the pattern
- *OutOfBoundsError* if the string parameter does not represent a correct date

date(year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer) : Date Creates a date from the values given by the parameters. The date is the sum of all parameter values; i.e., the number of years is added up with the number of months, and then with the number of days of a month. There are no limitations on parameter values: 0 refers to the item preceeding 1; for months, 1 designates January; hence 0 is the preceeding December.

date(year -> 1991, month -> 0, dayOfMonth -> 0); //returns d'1990-11-30 00:00:00.000' date(year -> 1991, month -> 155, dayOfMonth -> 42) //returns d'2003-12-12 00:00:00.000'

Parameters:

- *year* year of the date
- *month* month of the date
- *dayOfMonth* day of the date month

Throws:

- *NullParameterError* if a mandatory parameter is not specified

date(year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer, hour∗ : Integer, minute∗ : Integer, second∗ : Integer, millis∗ : Creates a date from the values given by the parameters. The date is given as a sum of all parameter values; i.e., the number of years is added up with the number of months, then with the number of days of a month, then with the number of hours, etc. There are no limitations on parameter values, for example, 0 refers to the item preceeding 1; for months, 1 designates January; hence 0 is the preceeding December.

```
date(year -> 2030, month -> -2, dayOfMonth -> -13, hour -> 0, minute -> 0, second -> 0 , millis -> 42);
 //returns d'2029-09-16 23:00:00.042'
date(year -> 2030, month -> 12, dayOfMonth -> 13, hour -> 0, minute -> 0, second -> 0 , millis -> 42);
//returns d'2030-12-12 23:00:00.042'
date(year -> 2030, month -> 13, dayOfMonth -> 13, hour -> 0, minute -> 0, second -> 0 , millis -> 33333)
//returns d'2031-01-13 00:00:33.333'
```

Parameters:

- *year*
- *month*
- *dayOfMonth*
- *hour*
- *minute*
- *second*
- *millis*

Throws:

- *NullParameterError* if mandatory parameters are not specified.

days(number∗ : Integer) : Duration Returns a duration with the specified number of days.

```
days(42)
//returns core::Duration {hours=0, seconds=0, months=0, weeks=0, minutes=0, days=42, millis=0, years=0}
```

Parameters:

- *number* number of days

Throws:

- *NullParameterError* if a mandatory parameter is not specified

format(date∗ : LocalDate, pattern∗ : String) : String Returns a string created by formatting the date to the specified pattern. The pattern uses the syntax defined in the java.time.format.DateTimeFormatter class.

Parameters:

- *date*
- *pattern*

Throws:

- *NullParameterError* if a mandatory parameter are not specified
- *FormatError* if the specified format is incorrect

formatDate(date∗ : Date, pattern∗ : String, timeZone : String) : String Formats the date as a String in the specified pattern. The formatted date can also be transformed to a time zone given by the timeZone parameter. The pattern and timeZone parameters use the syntax defined by org.joda.time.format.DateTimeFormat.

```
formatDate(d'2030-12-12 23:00:00.042', "EEE MMM dd HH:mm:ss 'time zone offset: 'Z yyyy", "GMT");
 //returns "Thu Dec 12 23:00:00 time zone offset: +0000 2030"
formatDate(d'330-12-12 15:10:00.042', "EEE MMM dd G h:mm a Z", null)
//retuns "Sat Dec 13 AD 3:19 PM +0009"
```

Parameters:

- *date* input date

- *pattern* target pattern of the output string
- *timeZone* time zone of the input date

Throws:

- *NullParameterError* if a mandatory parameter are not specified
- *FormatError* if the specified format or timeZone is incorrect

formatDate(date∗ : Date, pattern∗ : String, timeZone : String, locale : String) : String Formats the date as a String in the specified pattern. The formatted date can also be transformed to a time zone given by the timeZone parameter. The pattern and timeZone parameters use the syntax defined by org.joda.time.format.DateTimeFormat. Additionaly, you can set the locale code in the locale parameter. If undefined or invalid, the default locale is used.

```
formatDate(d'2020-12-12 15:10:00.042', "EEE MMM dd G h:mm a Z", null, "fr_FR")
//returns "sam. déc. 12 ap. J.-C. 4:10 PM +0100"
```

Parameters:

- *date* input date
- *pattern* target pattern of the output string
- *timeZone* time zone of the input date
- *locale* optional locale name, e.g. "de_AT"

Throws:

- *NullParameterError* if a mandatory parameter are not specified
- *FormatError* if the specified format or timeZone is incorrect

getDayOfMonth(date∗ : Date) : Integer Returns the day of the month of the input date.

```
getDayOfMonth(d'2012-30-12 09:48:00.042')
 //returns 12
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDayOfMonth(date∗ : LocalDate) : Integer Returns the day of the month of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDayOfWeek(date∗ : Date) : Integer Returns the day of the week of the input date as an Integer where 1 is Monday and 7 is Sunday.

```
getDayOfMonth(d'2012-30-12 09:48:00.042')
 //returns 12
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDayOfWeek(date∗ : LocalDate) : Integer Returns the day of the week of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDayOfYear(date∗ : Date) : Integer Returns the day of the year of the input date.

```
getDayOfYear(d'2020-1-1 09:48:00.042');
 //returns 1
 getDayOfYear(d'2013-12-31 09:48:00.042')
 //returns 365
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getDayOfYear(date∗ : LocalDate) : Integer Returns the day of the year of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getEpochDay(date∗ : LocalDate) : Integer Returns the epoch day of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getEpochMillis(date∗ : Date) : Integer Returns the number of milliseconds since January 1, 1970, 00:00:00 G↩ MT represented by the input date.

```
getEpochMillis(d'2013-12-31 09:48:00.042')
//returns 1388483280042
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getHour(date∗ : Date) : Integer Returns the hour of the day of the input date.

```
getHour(d'2013-12-31 09:06:00.042')
 //returns 10
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getMillis(date∗ : Date) : Integer Returns the millisecond of the second of the input date.

```
getMillis(d'2013-12-31 09:48:00.042')
 //returns 42
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getMinute(date∗ : Date) : Integer Returns the minute of the hour of the input date.

```
getMinute(d'2029-09-16 23:02:53.7000');
//returns 3
getMinute(d'2029-09-16 23:02:52.7000');
//returns 2
getMinute(d'2029-09-16 23:02:544.7000');
 //returns 11
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if mandatory parameters are not specified

getMonth(date∗ : Date) : Integer Returns the month of the year of the specified date. <pre>getMonth(d'2029-09-16 23:00:56.7000') //returns 9<pre>

Parameters:

- *date* input date

Throws:

- *NullParameterError* if mandatory parameters are not specified

getMonth(date∗ : LocalDate) : Integer Returns the month of the year of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if mandatory parameters are not specified

getSecond(date∗ : Date) : Integer Returns the second of the current minute of the input date.

```
getSecond(d'2029-09-16 23:00:56.700')
//returns 56
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getWeek(date∗ : Date) : Integer Returns the week of the current year of the input date.

```
getWeek(date("2025-03-14"))
 //returns 11
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getYear(date∗ : Date) : Integer Returns the year of the input date.

```
getYear(date(0))
 //returns 1970
```

Parameters:

- *date* input date

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getYear(date∗ : LocalDate) : Integer Returns the year of the specified local date.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

hours(number∗ : Integer) : Duration Returns a duration with the specified number of hours.

```
hours(25)
//returns core::Duration {hours=25, seconds=0, months=0, weeks=0, minutes=0, days=0, millis=0, years=0}
```

Parameters:

- *number* number of hours

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInDays(date1∗ : Date, date2∗ : Date) : Integer Returns the number of days between the date. If date1 $>$ date2, the resulting number is negative.

```
intervalInDays(date("2030-3-2"), date("2032-3-1"))
//returns 730
```

Parameters:

- *date1* start date for the day interval
- *date2* end date of the day interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInHours(date1∗ : Date, date2∗ : Date) : Integer Returns the number of hours between the dates. If date1 $>$ date2, the resulting number is negative.

```
intervalInHours(date("2030-3-2"), date("2032-3-1"))
 //returns 17520
```

Parameters:

- *date1* start date for the hour interval
- *date2* end date of the hour interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInMillis(date1∗ : Date, date2∗ : Date) : Integer Returns the number of milliseconds between the dates. If date1 > date2, the resulting number is negative.

```
intervalInMillis(date("2030-3-2"), date("2030-3-3"))
 //returns 86400000
```

Parameters:

- *date1* start date for the millisecond interval
- *date2* end date of the millisecond interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInMinutes(date1∗ : Date, date2∗ : Date) : Integer Returns the number of minutes between the dates. If date1 > date2, the resulting number is negative.

```
intervalInMinutes(date("2030-3-2"), date("2032-3-1"))
//returns 1051200
```

Parameters:

- *date1* start date for the minute interval
- *date2* end date of the minute interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInMonths(date1∗ : Date, date2∗ : Date) : Integer Returns the number of months between the dates. If date1 > date2, the resulting number is negative.

```
intervalInMonths(date("2030-3-2"), date("2032-3-1"))
//returns 23
```

Parameters:

- *date1* start date for the months interval
- *date2* end date of the months interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInSeconds(date1∗ : Date, date2∗ : Date) : Integer Returns the number of seconds between the dates. If date1 > date2, the resulting number is negative.

```
intervalInSeconds(date("2030-3-2"), date("2032-3-1"))
//returns 63072000
```

Parameters:

- *date1* start date for the seconds interval
- *date2* end date of the seconds interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInWeeks(date1∗ : Date, date2∗ : Date) : Integer Returns the number of weeks between the dates. If date1 > date2, the resulting number is negative.

```
intervalInWeeks(date("2030-3-2"), date("2032-3-2"))
 //returns 104
```

Parameters:

- *date1* start date for the week interval
- *date2* end date of the week interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

intervalInYears(date1∗ : Date, date2∗ : Date) : Integer Returns the number of years between the dates. If date1 > date2, the resulting number is negative.

```
intervalInYears(date("2030-3-2"), date("2032-3-1"))
 //returns  1
```

Parameters:

- *date1* start date for the year interval
- *date2* end date of the year interval

Throws:

- *NullParameterError* if a mandatory parameter is not specified

localDate(epochDay∗ : Integer) : LocalDate Creates a local date from the specified epochDay. The epoch day is a simple incrementing count of days where day 0 is 1970-01-01. Negative numbers represent earlier days.

Parameters:

- *epochDay*

Throws:

- *NullParameterError* if mandatory parameters are not specified
- *OutOfBoundsError* if the value of epochDay is out of range

localDate(string∗ : String) : LocalDate Creates a local date from the specified string. The format of the input string is yyyy-MM-dd (ISO 8601).

Parameters:

- *string*

Throws:

- *NullParameterError* if mandatory parameters are not specified
- *FormatError* if the string parameter does not conform to the specified format

localDate(year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer) : LocalDate Creates a local date from the values given by the following parameters: year, month (month of year), and dayOfMonth. year must be between -999,999,999 and +999,999,999, month must be between 1 and 12, day must be between 1 and 31 (depends on the month).

Parameters:

- *year*
- *month*
- *dayOfMonth*

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the value of any field is out of range

max(dates∗ : Date...) : Date Returns the latest date from the input dates.

```
max(date("2100-12-18"), now(), date("1990-12-18"))
//returns d'2100-12-17 23:00:00.000'
```

Parameters:

- *dates* input dates

Throws:

- *NullParameterError* if a mandatory parameter is not specified

max(dates∗ : LocalDate...) : LocalDate Returns the maximal date from a list of local dates.

Parameters:

- *dates*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

millis(number∗ : Integer) : Duration Returns a duration with the specified number of milliseconds.

```
millis(120)
//returns core::Duration {hours=0, seconds=0, months=0, weeks=0, minutes=0, days=0, millis=120, years=0}
```

Parameters:

- *number* number of miliseconds

Throws:

- *NullParameterError* if a mandatory parameter is not specified

min(dates∗ : Date...) : Date Returns the soonest date from the input dates.

```
min(date("2100-12-18"), now(), date("1990-12-18"))
//returns d'1990-12-17 23:00:00.000'
```

Parameters:

- *dates* input dates

Throws:

- *NullParameterError* if a mandatory parameter is not specified

min(dates∗ : LocalDate...) : LocalDate Returns the minimal date from a list of local dates.

Parameters:

- *dates*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

minutes(number∗ : Integer) : Duration Returns a duration with the specified number of minutes.

```
minutes(64)
 //returns core::Duration {hours=0, seconds=0, months=0, weeks=0, minutes=64, days=0, millis=0, years=0}
```

Parameters:

- *number* number of minutes

Throws:

- *NullParameterError* if a mandatory parameter is not specified

months(number∗ : Integer) : Duration Returns a duration with the specified number of months.

```
months(15)
//returns core::Duration {hours=0, seconds=0, months=15, weeks=0, minutes=0, days=0, millis=0, years=0}
```

Parameters:

- *number* number of months

Throws:

- *NullParameterError* if a mandatory parameter is not specified

now() : Date Returns the current date and time.

```
now()
//returns d'2018-06-08 09:36:14.015'
```

seconds(number∗ : Integer) : Duration Returns a duration with the specified number of seconds.

```
seconds(64)
//returns core::Duration {hours=0, seconds=64, months=0, weeks=0, minutes=0, days=0, millis=0, years=0}
```

Parameters:

- *number* number of seconds

Throws:

- *NullParameterError* if a mandatory parameter is not specified

setDate(date∗ : Date, year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer) : Date Changes the fields of the input date according to the values of the Integer parameters and returns the resulting date. The value must fall into valid ranges. The dayOfMonth must be in the correct range, that is from 1 to the last day of the given month in the year.

```
setDate(date("2030-1-1"), 2040, 2, 28)
 //results d'2040-02-28 00:00:00.000'
```

Parameters:

- *date* input day
- *year* required year value
- *month* required month value
- *dayOfMonth* required day of month value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if any of the parameters is out of range

setDayOfMonth(date∗ : Date, dayOfMonth∗ : Integer) : Date Changes the day field to point to the day of month defined by the dayOfMonth parameter value and returns the resulting date. The dayOfMonth must be in the correct range, that is from 1 to the last day of the given month in the year.

```
setDayOfMonth(date -> date(0), dayOfMonth -> 13)
//returns d'1970-01-13 00:00:00.000'
//date(0) resolves to d'1970-01-01 00:00:00.000'
```

Parameters:

- *date* input date
- *dayOfMonth* required date of month

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified dayOfMonth is out of range

setDayOfWeek(date∗ : Date, dayOfWeek∗ : Integer) : Date Changes the day and month fields to point to the closes day of week defined by the dayOfWeek parameter value and returns the resulting date.

```
setDayOfWeek(date -> date(0), dayOfWeek -> 2);
//results in the closes Tuesday: d'1969-12-30 00:00:00.000'
```

Parameters:

- *date* input date
- *dayOfWeek* required day of week

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified dayOfWeek is out of range (dayOfWeek $<$ 1 or dayOfWeek $>$ 7)

setDayOfYear(date∗ : Date, dayOfYear∗ : Integer) : Date Changes the day and month fields of the input date to correspond to the dayOfYear parameter value and returns the resulting date.

```
setDayOfYear(date -> date(0), dayOfYear -> 33)
 //returns d'1970-02-02 00:00:00.000'
```

Parameters:

- *date* input date
- *dayOfYear* new day of year value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified dayOfYear is out of range (dayOfYear $<$ 1 or dayOfYear $>$ 365 or 366, depending on the year)

setHour(date∗ : Date, hour∗ : Integer) : Date Changes the hour field of the input date to the hour parameter value and returns the resulting date.

```
setHour(date -> date(0), hour -> 10)
 = d'1970-01-01 10:00:00.000'
```

Parameters:

- *date* input date
- *hour* new hour field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified hour is out of range (hour $<$ 0 or hour $>$ 23)

setMillis(date∗ : Date, millis∗ : Integer) : Date Changes the milliseconds field of the input date to the millis parameter value and returns the resulting date.

```
setMillis(date-> d'2100-11-01 10:10:10.111', millis -> 42)
//returns d'2100-11-01 10:10:10.042'
```

Parameters:

- *date* input date
- *millis* new millis field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified smillis is out of range (millis $<$ 0 or millis $>$ 999)

setMinute(date∗ : Date, minute∗ : Integer) : Date Changes the minute field of the input date to the minute parameter value and returns the resulting date.

```
setMinute(date-> d'2100-11-01 12:12:13.000', minute -> 42)
 //returns d'2100-11-01 12:42:13.000'
```

Parameters:

- *date* input date
- *minute* new minute field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified minute is out of range (minute $<$ 0 or minute $>$ 59)

setMonth(date∗ : Date, month∗ : Integer) : Date Changes the month field of the input date to the month parameter value and returns the resulting date.

```
setMonth(date-> d'2100-11-01 12:10:13.000', month -> 2)
 //returns d'2100-02-01 12:10:13.000'
```

Parameters:

- *date* input date
- *month* new month field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified month is out of range (month $<$ 1 or month $>$ 12)

setSecond(date∗ : Date, second∗ : Integer) : Date Changes the second field of the input date to the second parameter value and returns the resulting date.

```
setSecond(date-> d'2100-01-01 12:10:13.000', second -> 42)
 //returns d'2100-01-01 12:10:42.000'
```

Parameters:

- *date* input date
- *second* new second field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified second is out of range (second $<$ 0 or second $>$ 59)

setTime(date∗ : Date, hour∗ : Integer, minute∗ : Integer, second∗ : Integer, millis∗ : Integer) : Date Changes the time fields of the input date to the values of the parameters and returns the resulting date. Parameters defining the time must not exceed their ranges: 0 =$<$ hour $<$ 23; 0 =$<$ minute $<$ 0 or minute $>$ 59 or second $<$ 0 or second $>$ 59 or millis $<$ 0 or millis $>$ 999)

```
setTime(date -> d'2100-01-01 12:10:00.000', hour -> 20, minute -> 33, second -> 12, millis -> 200)
 //returns d'2100-01-01 19:33:12.200'
```

Parameters:

- *date* input date
- *hour* new hour field value
- *minute* new minute field value
- *second* new second field value
- *millis* new millisecond field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if any of the parameters is out of range

setWeek(date∗ : Date, week∗ : Integer) : Date Changes the week field of the year of the input date to the week parameter value and returns the resulting date.

```
setWeek(date("2035-1-15"), 12)
//returns d'2035-03-19 00:00:00.000'
```

Parameters:

- *date* input date
- *week* new week field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *OutOfBoundsError* if the specified week is out of range (week $<$ 1 or week $>$ 52)

setYear(date∗ : Date, year∗ : Integer) : Date Changes the year field of the input date to the year parameter value and returns the resulting date.

```
setYear(date->now(), year-> 2100)
 //returns d'2100-06-08 12:54:46.141'
```

Parameters:

- *date* input date
- *year* new year field value

Throws:

- *NullParameterError* if a mandatory parameter is not specified

toDate(localDate∗ : LocalDate) : Date Returns the date of this local date at start of day. The system default time-zone is used for the conversion.

Parameters:

- *localDate*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

toLocalDate(date∗ : Date) : LocalDate Returns the local date of this date. The system default time-zone is used for the conversion.

Parameters:

- *date*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

today() : Date Returns the current date with the time set to midnight in the time zone of the LSPS Server (by default UTC).

```
today()
//returns d'2018-06-07 22:00:00.000'
```

weeks(number∗ : Integer) : Duration Returns a duration with the specified number of weeks.

```
weeks(12)
 //returns core::Duration {hours=0, seconds=0, months=0, weeks=12, minutes=0, days=0, millis=0, years=0}
```

Parameters:

- *number* number of weeks

Throws:

- *NullParameterError* if a mandatory parameter is not specified

years(number∗ : Integer) : Duration Returns a duration with the specified number of years.

```
years(35)
 //returns core::Duration {hours=0, seconds=0, months=0, weeks=0, minutes=0, days=0, millis=0, years=35}
```

Parameters:

- *number* number of years

Throws:

- *NullParameterError* if a mandatory parameter is not specified

8.4.15 Restart

clearApplicationRestartData() : void Clears the information about previous application restart.

explicitFinishing() : void Specifies that the current model instance starting is finished explicitly, by calling the modelInstanceStartSucceeded() function.

getStartStatus(modelInstance : ModelInstance) : ModelInstanceStartStatus Returns the starting status of the given model instance. If the modelInstance parameter is is null, the current model instance is used.

Parameters:

- *modelInstance* model instance

getStartStatuses(modelInstances∗ : Set<ModelInstance>) : Map<ModelInstance, ModelInstanceStartStatus> Returns map of starting statuses for given model instances. If the modelInstances parameter is is null, statuses for all currently monitored instances are returned.

Parameters:

- *modelInstances* list of model instances

modelInstanceStartSucceeded() : void Tells that the model instance starting succeded.

setCountOfModelInstancesToStart(count∗ : Integer) : void Sets the number of model instances to be restarted. It can be set in advance of the actual model instance restarting.

Parameters:

- *count* count of model instances to start

watchStarting(modelInstance : ModelInstance) : void Adds the model instance to a list of started model instances. If the modelInstance parameter is is null, the current model instance is used.

Parameters:

- *modelInstance* model instance to watch

8.4.16 Record

createInstance(recordType : Type<T>, properties : Map<String, Object>) : T Creates and returns a new instance of the specified *record type*. If the properties are specified, the instance is initialized with the specified values of properties.

```
createInstance(RegistrationData, [ "approved" -> false ])
```

Parameters:

- *recordType* record type
- *properties* map with properties

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the specified properties do not belong to the specified record type

deleteRecords(records* : Record...) : void[SIDE EFFECT] Deletes the specified records and their related data links navigable from the records. Works for shared and for non-shared records. Values of the deleted records in all slots (variables, function or task call parameters, etc.) referring to the records become `null`.

```
deleteRecords(persona1, persona2, order1)
```

Parameters:

- *records* records to delete

Throws:

- *NullParameterError* if mandatory parameter is null
- *RecordNotFoundError* if the records parameter contains a shared record with an incorrect reference to a database record

deleteRecords(records* : Collection<Record>) : void[SIDE EFFECT] Deletes the specified records and their related data links navigable from the records. Works for shared and for non-shared records. Values of the deleted records in all slots (variables, function or task call parameters, etc.) referring to the records become `null`. If the record is referred to from a collection or a key in a map, it is removed.

```
deleteRecords([persona1, persona2, order1])
```

Parameters:

- *records* records to delete

Throws:

- *NullParameterError* if mandatory parameter is null
- *RecordNotFoundError* if the records parameter contains a shared record with an incorrect reference to a database record

flushAndRefresh(records : Record...) : void Executes flush on all relevant sessions and refreshes (re-reads) from the database on the records. It is useful in the cases when the records might have been modified by custom objects or third-party systems.

Parameters:

- *records*

8.4.17 Property

getProperties(type∗ : Type<Record>) : List<Property> Returns all properties of the given record type.

Parameters:

- *type*

Throws:

- *NullParameterError* if a mandatory parameter is null

getProperty(type∗ : Type<Record>, name∗ : String) : Property Returns the property of the given record type with the specified name, or null if the type has no such property.

Parameters:

- *type* record with the property
- *name* property name

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyMetadata(property∗ : Property) : Map<String, String> Returns the property metadata.

Parameters:

- *property*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyName(property∗ : Property) : String Returns the property name as a string.

Parameters:

- *property*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyNames(propertyPath∗ : PropertyPath) : List<String> Returns the property names.

Parameters:

- *propertyPath*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyPathRecordType(propertyPath∗ : PropertyPath) : Type<Record> Returns the property path record type. It is the record type where the given property path starts.

Parameters:

- *propertyPath*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyPathType(propertyPath∗ : PropertyPath) : Type<Object> Returns the property path type (the type of the last property).

Parameters:

- *propertyPath*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyPathValue(record∗ : Record, propertyPath∗ : PropertyPath) : Object Returns the value of the given property path of the given record object.

Parameters:

- *record*
- *propertyPath*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record the the given property are not compatible

getPropertyRecordType(property∗ : Property) : Type<Record> Returns the record type where the specified property is declared.

```
getPropertyRecordType(SubRecord.fieldInSuperRecord)
//returns SuperRecord
```

Parameters:

- *property*

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyReference(reference∗ : Reference<Object>, properties∗ : Property...) : Reference<Object> Returns the reference to the object defined by the last property parameter. The path to the object is resolved as a chain of associations starting in the reference parameter through the property parameters in specified order.

Parameters:

- *reference*
- *properties*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the given record and property are not compatible

getPropertyType(property∗ : Property) : Type<Object> Returns the property type.

```
getPropertyType(Person.name) //returns String
```

Parameters:

- *property* record property

Throws:

- *NullParameterError* if a mandatory parameter is null

getPropertyValue(record∗ : Record, property∗ : Property) : Object Returns the value of the given property of the given record object.

Parameters:

- *record*
- *property*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record the the given property are not compatible

isAutogenerated(property∗ : Property) : Boolean Returns true if the given property is autogenerated (usually a primary key from a sequence).

Throws: -"NullParameterError" if mandatory parameter is null.

Parameters:

- *property*

isPrimaryKey(property∗ : Property) : Boolean Returns true if the given property is a primary key of a shared record (or a part of composite primary key).

Throws: -"NullParameterError" if mandatory parameter is null.

Parameters:

- *property*

isReadOnly(propertyPath∗ : PropertyPath) : Boolean Returns true if the given property path is read only.

Throws: -"NullParameterError" if mandatory parameter is null.

Parameters:

- *propertyPath*

isVersion(property∗ : Property) : Boolean Returns true if the given property is a version field.

Throws: -"NullParameterError" if mandatory parameter is null.

Parameters:

- *property*

setPropertyPathValue(record∗ : Record, propertyPath∗ : PropertyPath, value : Object) : void[SIDE EFFECT] Sets the value of the given property path of the given record object.

Parameters:

- *record*
- *propertyPath*
- *value*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record and the given property are not compatible

setPropertyValue(record∗ : Record, property∗ : Property, value : Object) : void[SIDE EFFECT] Sets the value of the given property of the given record object. Returns null.

Parameters:

- *record*
- *property*
- *value*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record and the given property are not compatible

8.4.18 Generic

cast(object : Object, objectType∗ : Type<E>) : E If the type of the input object is compatible with the specified objectType, the function casts the object to the objectType.

```
cast(-1, Decimal);
 //returns -1
(-1.1).cast(Integer)
 //returns  IncompatibleTypeError
```

Parameters:

- *object* input object
- *objectType* target type

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the type of the object is not compatible with the input objectTyp

clone(object : T) : T[SIDE EFFECT] Creates and returns a shallow copy of a collection or a user-defined record (related records remain the same), or returns the object itself if it is of another type.

```
clone([[1,2,3],3, "a", date(0)]);
 //returns [[1,2,3],3,"a",d'1970-01-01 00:00:00.000']
 def Applicant john := new Applicant(name -> "John", surname -> "Doe", tutor -> new Tutor(name -> "Jane")
clone(john);
//returns a new applicant with the same tutor:
// Applicant(id->7, name->"John", surname->"Doe", tutor->Tutor);
clone(Applicant)
//retuns myModule::Applicant
```

Parameters:

- *object*

error(code∗ : String) : void[SIDE EFFECT] Throws an error with the specified code.

```
error("code red")
//returns null, error code: code red
```

Parameters:

- *code*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

error(code∗ : String, message : String) : void[SIDE EFFECT] Throws an error with the code and a message.

```
error("code red", "This is really serious.")
//returns This is really serious., error code: code red
```

Parameters:

- *code* error code
- *message* error message

Throws:

- *NullParameterError* if a mandatory parameter is not specified

getType(object : E) : Type<E> Returns the type of the input object. Note that, unlike typeOf(), getType() is an extension method.

```
now.getType();
//returns { : Date}
now().getType()
//returns Date
```

Parameters:

- *object* input object

isInstance(object : Object, objectType∗ : Type<Object>) : Boolean Returns true if the input object is assignment-compatible with the objectType.

```
isInstance(1, Decimal);
 //returns true
(1.1).isInstance(Integer)
//returns false
isInstance(null, String);
//returns true
```

Parameters:

- *object* input object
- *objectType* check type

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isSubtype(subtype∗ : Type<Object>, supertype∗ : Type<Object>) : Boolean Returns true if subtype is a subtype of supertype.

```
isSubtype(Decimal, Integer);
//returns false
isSubtype(Integer, Decimal);
//returns true
isSubtype(Null, Object)
//returns true
```

Parameters:

- *subtype* subtype
- *supertype* supertype

Throws:

- *NullParameterError* if a mandatory parameter is not specified

isSubtypeOf(subtype∗ : Type<Object>, supertype∗ : Type<Object>) : Boolean Returns true if subtype is a subtype of supertype. Note that, unlike isSubtype(), isSubtypeOf() is an extension method.

```
String.isSubtypeOf(Object)
//returns true
```

Parameters:

- *subtype*
- *supertype*

Throws:

- *NullParameterError* if a mandatory parameter is not specified

typeOf(object : E) : Type<E> Returns the type of the input object.

```
typeOf(now());
 //returns Date
typeOf(1.0);
//returns Integer
typeOf({ e-> String} );
 //returns {Object : Type<String>}
typeOf(now)
 //returns { : Date}
```

Parameters:

- *object* input object

8.4.19 Json

convertToJson(object∗ : Object) : String Converts the *object* to JSON. The object can be an instance of a Record, List, Set or Map with their data structure consisting of the types String, Boolean, Binary, Decimal, Integer, Date, Set, List, Map, Record, or Enumeration.

```
convertToJson(
//record with a related record:
  new Applicant(
    name -> "John",
    surname -> "Doe",
    tutor -> new Tutor(
       name -> "Jane",
       hireDate -> date("2018-1-1")
       )
     )
 )
 /*returns
{
  "id" : 2,
 "name" : "John",
  "surname" : "Doe",
  "tutor" : {
    "id" : 1,
    "name" : "Jane",
    "hireDate" : 1514761200000
  }
}*/
```

Parameters:

- *object* instance of a Record, List, Set or Map to convert to JSON

Throws:

- *NullParameterError* a mandatory parameter is null
- *IncompatibleTypeError* the object is not a Record, List, Set, Map or contains an unsupported type
- *JsonProcessingError* an error occurred during conversion to JSON

convertToJson(useExternalRef∗ : Boolean, object∗ : Object) : String Converts internal data structures passed as the object parameter to JSON representation. The object parameter must be an instance of a Record, List, Set or Map type otherwise an exception is thrown. Furthermore, the data structure represented by the object parameter can consist only of the following types: String, Boolean, Binary, Decimal, Integer, Date, Set, List, Map, Record, Enumeration. The function returns a string which represents the data in JSON format. The parameter useExternalRef defines how shared records are converted to JSON. In case the useExternalRef is true the shared record is stored as type and primary key string. In other case all fields of the shared record are stored.

Parameters:

- *useExternalRef*
- *object*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the object is not a Record, List, Set, Map or the data structure represented by the object parameter contains an unsupported type
- *JsonProcessingError* if an error occurred during conversion to JSON

jsonSchema(useExternalRef : Boolean, type : Type<Object>) : String

jsonToMap(json∗ : String) : Map<String, Object> Parses JSON string to a map. The json parameter represents the JSON string to parse.

```
jsonToMap("
  {
    ""name"" : ""John"",
    ""surname"" : ""Doe"",
    ""tutor"" : {
        ""id"" : 1,
        ""name"" : ""Jane"",
        ""hireDate"" : 2012-04-23T18:25:43.511Z
    }
}")
```

Parameters:

- *json*

Throws:

- *NullParameterError* a mandatory parameter is null
- *JsonProcessingError* the specified json cannot be parsed

jsonToMap(json∗ : String, doInsert∗ : Boolean, doUpdate∗ : Boolean) : Map<String, Object> Parses JSON string to a map. he *json* parameter represents the JSON string to parse and the resultType parameter determines the type of the returned data. The referred type must be a Record, List, Set or Map type, otherwise an exception is thrown. Furthermore the data structure represented by the resultType can consist only from these supported types: String, Boolean, Binary, Decimal, Integer, Date, Set, List, Map, Record, Enumeration. If there exist a shared record with the same primary key as specified in JSON the record is used by import. Here, it does not matter whether the shared record is specified by external reference or by value in JSON. If the record specified by reference in JSON does not exist in database it is ignored and null is used as its imported value or empty collection for collection of records. No exception is thrown in this case. If the parameter *doInsert* is true and the database does not contain a shared record specified by value in JSON or the JSON does not specify primary key, the record from JSON is inserted to database. For auto-generated primary keys, a new key value is created, for not auto-generated primary keys, the value of the key from JSON is used. If *doInsert* is false, a record from JSON which does not exist in the DB is ignored and null is used for its value in the imported data instead; for collections of non-existing records, empty collections are created by import. If the parameter *doUpdate* is true and the database contains a shared record with the same primary key as a record specified by value in JSON, values of this record fields are updated from JSON. If false, values of fields do not change.

```
jsonToMap("
  {
    ""name"" : ""John"",
    ""surname"" : ""Doe"",
    ""tutor"" : {
        ""id"" : 1,
        ""name"" : ""Jane"",
        ""hireDate"" : 2012-04-23T18:25:43.511Z
    }
}")
```

Parameters:

- *json*
- *doInsert*
- *doUpdate*

Throws:

- *NullParameterError* if a mandatory parameter is null
- *JsonProcessingError* if the specified json cannot be parsed

mapToJson(map∗ : Map<String, Object>) : String Converts the *map* to JSON. The string keys of the map represent the keys of the JSON object. The values of the map must be instances of String, Boolean, Binary, Decimal, Integer, Date, Record, Enumeration, Null, Set and List of the previously mentioned types, or Map<String, Object>. Otherwise an exception is thrown. Nested JSON objects are represented either by values of the type Map<String, Object> or record. The function returns a string which represents the data in JSON format.

```
mapToJson(
  [
    "name" -> "John",
    "address" -> ["City"->"Amsterdam"]
  ]
)
```

Parameters:

- *map*

Throws:

- *NullParameterError* a mandatory parameter is null
- *IncompatibleTypeError* the map value contains an unsupported type
- *JsonProcessingError* an error occurred during conversion to JSON

parseJson(json∗ : String, resultType∗ : Type<E>) : E[DEPRECATED] Deprecated: use *parseJson(String json, Type resultType, Boolean doInsert, Boolean doUpdate)*.

Parses the *json* string to internal data structures. The *json* parameter represents the JSON string to parse and the *resultType* parameter determines the type of the returned data. The type must be a Record, List, Set or Map type, otherwise an exception is thrown. Also the data structure represented by the resultType can consist only from these supported types: String, Boolean, Binary, Decimal, Integer, Date, Set, List, Map, Record, Enumeration.

Note that when parsing JSON to shared Records if json defines the primary-key value: if the primary key is autogeneratedit, the primary key in the json is ignored: A new shared record is created. If the primary key is not autogenerated and a record with the defined primary key already exists, the call fails with a runtime exception.

```
parseJson("
{
  ""id"" : 1,
  ""approved"" : false,
  ""firstName"" : ""Jane"",
  ""lastName"" : ""Doe""
}
", RegistrationData)
```

Parameters:

- *json* string with JSON
- *resultType* type to which to parse the JSON string

Throws:

- *NullParameterError* mandatory parameter is null
- *IncompatibleTypeError* resultType parameter is not a a Record, List, Set or Map type or the data structure represented by the resultType parameter contains an unsupported type
- *JsonProcessingError* json cannot be parsed

parseJson(json∗ : String, resultType∗ : Type<E>, doInsert∗ : Boolean, doUpdate∗ : Boolean) : E Parses JSON string to internal data structures. The *json* parameter is the JSON string and the resultType parameter is the type to switch it should be parsed. The referred type must be a Record, List, Set or Map type. The data structure of the resultType can consist only of the types String, Boolean, Binary, Decimal, Integer, Date, Set, List, Map, Record, or Enumeration.

If a shared record with the primary key equal to the primary key specified in JSON, the record is used by import. Here, it does not matter whether the shared record is specified by external reference or by value in JSON. If the record specified by reference in JSON does not exist in the database, it is ignored and null is used as its imported value or empty collection for collection of records. No exception is thrown in this case. If the parameter *doInsert* is true and the database does not contain a shared record specified by value in JSON or the JSON does not specify a primary key, the record from JSON is inserted into the database. For auto-generated primary keys, a new key value is created, for non-auto-generated primary keys, the value of the key from JSON is used. If *doInsert* is false, a record from JSON which does not exist in the DB is ignored and

null is used for its value in the imported data instead; for collections of non-existing records, empty collections are created by import. If the parameter *doUpdate* is true and the database contains a shared record with the same primary key as a record specified by value in JSON, values of this record fields are updated from JSON. If false, such records remain unchanged.

```
parseJson("
{
  ""id"" : 1,
  ""approved"" : false,
  ""firstName"" : ""Jane"",
  ""lastName"" : ""Doe""
}
", RegistrationData, true, true)
```

Parameters:

- *json* string with JSON
- *resultType* type to which to parse the JSON string
- *doInsert* whether to allow insert of new shared records
- *doUpdate* whether to allow update of shared records

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the resultType parameter is not a a Record, List, Set or Map type or if the data structure represented by the resultType parameter contains an unsupported type
- *JsonProcessingError* if the json cannot be parsed
- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the resultType parameter is not a a Record, List, Set or Map type or if the data structure represented by the resultType parameter contains an unsupported type
- *JsonProcessingError* if the json cannot be parsed

8.4.20 Ws

callHttp(sslConfig : SslConfig, endpointAddress∗ : String, httpMethod∗ : String, requestContentType∗ : String, input : Strin
Synchronous http call.

```
callHttp(
   sslConfig -> null,
   endpointAddress-> "http://localhost:9200/myindex/section/_search",
   httpMethod -> "GET",
   requestContentType -> "application/json",
   input -> "{""query"":{""match"":{""title"":""Controlling Flow""}}}",
   login -> null,
   password -> null,
   readTimeout -> null,
   requestHeaders -> null,
   responseHeaders -> &responseHeaders,
   resposeCode -> &responseCode,
   logMessages -> true
)
```

Parameters:

- *sslConfig* ssl configuration if secure communication is required
- *endpointAddress* url
- *httpMethod* GET|PUT|POST|DETETE
- *requestContentType* should be set if there is an input (payload)
- *input* a payload for PUT|POST
- *login* login name

- *password* password
- *readTimeout* timeout in milliseconds
- *requestHeaders* request headers
- *responseHeaders* reference to where response headers are set
- *resposeCode* reference to where to store the response code
- *error* reference to where the error object is set
- *logMessages* if true request and response messages are logged to console

callWebservice(soapMetadata∗ : Map<String, String>, sslConfig : SslConfig, endpointAddress : String, login : String, pass
Synchronous call of a webservice. It provides the same functionality as the generated webservice task but can be used in expressions. It uses the same data objects that are generated for the webservice task and the same metadata.

Parameters:

- *soapMetadata* the same metadata as for the webservice task
- *sslConfig* the ssl configuration if secure communication is required
- *endpointAddress* webservice url. If not specified, the endpoint from metadata is used.
- *login* login name
- *password* password
- *input* webservice input represented by a record
- *requestSoapHeaders* map of soap request headers
- *requestHeaders* map of request headers
- *readTimeout* timeout in milliseconds
- *outputType* lsps type of the return value
- *responseSoapHeaders* reference to where response soap headers are set
- *responseHeaders* reference to where response headers are set
- *error* reference to where the error object is set
- *logMessages* if true request and response messages are logged to console

8.4.21 Auditing

findByRevision(record∗ : Record, revision∗ : Integer) : E Returns the audited record in the revision.

```
findByRevision(getBookByTitle("Death in Venice"), 7)
```

Parameters:

- *record* audited shared record
- *revision* required revision of the record

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record is not a shared record

getCurrentRevision(revisionEntityType∗ : Type<T>, persist∗ : Boolean) : T Returns the current custom revision entity. If the persist parameter is set to `true`, a new revision entity is created even if no audited record is changed in the transaction.

getCurrentRevision(auditing::MyRevisionEntity, true)

Parameters:

- *revisionEntityType* revision entity type

- *persist* if true, a new revision entity is created even if no audited record was changed in the transaction

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the revisionEntity type is not a custom revision entity record type

getRevisions(record∗ : Record, from : Date, to : Date) : List<Integer> Returns the list of revision identifiers in which the audited shared record was modified during the period defined by the from and to parameters. If either of the parameters is null, the period is unbounded from start or end respectively.

Parameters:

- *record* shared record changed in the revisions
- *from* start of the period when the revisions were created
- *to* end of the period when the revision were created

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the record is not a shared record

8.4.22 Query

countAll(type∗ : Type<Record>) : Integer Returns the count of all shared records of the given type.

Parameters:

- *type* shared record type

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the type parameter is not a shared record type

findAll(type∗ : Type<E>) : List<E> Returns all shared records of the given type.

```
findAll(Model)
```

Parameters:

- *type* record type

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the type parameter is not a shared record type

findById(id∗ : Object, type∗ : Type<E>) : E[DEPRECATED] Returns a shared record of the specified type and identified by the given primary key. Composed primary key is represented as a map of type Map<String, Object> or of type Map<Property, Object>. The keys of the map identify the properties of the primary key and values represent the value of the properties. If the key is of the type string then it contains the simple property name. If there is no record with the given id, the function returns null.

```
findById(ModelInstance, 8000);
findById(RecWithComplexPK, ["id1" -> 1, "id2" -> 2])
```

Parameters:

- *id* primary key
- *type* shared record type

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the type parameter is not a shared record type

findById(type∗ : Type<E>, id∗ : Object) : E Returns a shared record of the specified type and identified by the given primary key. Composed primary key is represented as a map of type Map<String, Object> or of type Map<Property, Object>. The keys of the map identify the properties of the primary key and values represent the value of the properties. If the key is of the type string then it contains the simple property name. If there is no record with the given id, the function returns null.

```
ModelInstance.findById(8000);
RecWithComplexPK.findById(["id1" -> 1, "id2" -> 2])
```

Parameters:

- *type* shared record type
- *id* primary key

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncompatibleTypeError* if the type parameter is not a shared record type

8.4.23 Utilities

debugLevel() : Integer[DEPRECATED] Returns an integer (100) representing the "Debug" log level. Deprecated. Replaced by constant DEBUG_LEVEL.

debugLog(message∗ : { : String}) : String[SIDE EFFECT] Logs a message to the console at the debug logging level and returns the logged string or null if no message was logged. If logging is not enabled for the debug level on the server, the message closure is not evaluated.

```
debugLog({ -> "debug message" })
```

Parameters:

- *message* message to log

Throws:

- *NullParameterError* if a mandatory parameter is null

debugLog(message∗ : { : String}, level∗ : Integer) : String[SIDE EFFECT] Logs a message to the console at the debug logging level and returns the logged string or null if no message was logged. If logging is not enabled for the debug level on the server, the message closure is not evaluated.

```
debugLog({ -> "debug message" }, INFO_LEVEL )
```

Parameters:

- *message* message to log
- *level* log level of the message

Throws:

- *NullParameterError* if a mandatory parameter is null

errorLevel() : Integer[DEPRECATED] Returns an integer (400) representing the "Error" log level. Deprecated. Replaced by constant ERROR_LEVEL.

executeSqlFile(binaryInput∗ : Binary) : void[SIDE EFFECT] Execute the SQL in the *binary*. Empty lines and lines starting with `--` or `;` are ignored.

```
def File file := getResource(module -> "localization", path -> "sqlinsert.txt");
//the file contains SQL statements separated with semicolons
executeSqlFile(file.content);
```

Parameters:

- *binaryInput* binary with SQL

getApplicationData(key∗ : String) : Object Returns an application data object with the *key* "locale" string or "user": locale returns the locale and user the login name of the logged-in user in the LSPS Process Application.

```
getApplicationData("locale")
```

Parameters:

- *key* name of the application data

Throws:

- *NullParameterError* if a mandatory parameter is null

getCurrentDatabaseName() : String Returns the DBMS name of the current LSPS_DS datasource, such as `MYSQL`, `ORACLE`, etc.

getErrorCode() : String If used in the catch section of the try/catch command, the function returns the code of caught error. If used elsewhere, it returns null.

getErrorMessage() : String If used in the catch section of the try/catch command, the function returns the message of caught error. If used elsewhere, it returns null.

infoLevel() : Integer[DEPRECATED] Returns an integer (200) representing the "Info" log level. Deprecated. Replaced by constant INFO_LEVEL.

log(message∗ : String, level : Integer) : void[SIDE EFFECT] Logs the specified *message* to the application log at the specified log level. If the level is not specified, the level value is set to the "Info" level (value 200).

Parameters:

- *message* message to log
- *level* log level of the message

Throws:

- *NullParameterError* if a mandatory parameter is null

splitPathname(name∗ : String) : List<String> Splits a fully qualified element name to the names of its components as separated by the namespace separator (::).

```
splitPathname("module::'role 1'")
//returns ["module", "role 1"]
```

Parameters:

- *name* input fully qualified name

Throws:

- *NullParameterError* if a mandatory parameter is null
- *IncorrectPathnameError* if syntax of the specified name is incorrect

uuid() : String Returns a universally unique identifier (UUID).

```
uuid()
//returns "3021c2bf-b146-492b-9849-9ebd4abd56e8"
```

warningLevel() : Integer[DEPRECATED] Returns an integer (300) representing the "Warning" log level. Deprecated. Replaced by constant WARNING_LEVEL.

8.5 Tasks

8.5.1 Core

CreateModelInstance Creates a new model instance and waits until it receives a confirmation message with the model instance ID. Note that the message is sent asynchronously when the interpretation of the new model instance starts. Hence the task becomes finished even though the first transaction of the new model instance failed and was rolled back.

<pre>//example task parameters: model -> getModel("registration"), properties -> ["customer" -> "John Doe"], modelInstance -> &modelInstanceVar</pre>

model : Model Model to be instantiated.

properties : Map<String, String> Map of model instance initialization properties. If null, an empty properties map is created in the instantiated model.

modelInstance : Reference<ModelInstance> Reference to the created model instance. If the created model instance throws a runtime exception during its initialization, it is set to null.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *ModelInstaniationError* if the model instance did not start due to an internal initiation error, not due to a model interpretation error

TerminateModelInstance Terminates a running model instance which becomes finished. <pre>//example task parameters: modelInstance -> getModelInstance(25009)</pre>

modelInstance : ModelInstance Model instance to be terminated.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

DeleteModelInstanceLogs Permanently removes process logs of the specified model instances from the database. All model instances must be finished. Note that process logs hold data on each changes of model, process, and element instances. Other data, such as the logs created by the instance, its properties, entry on the fiinished model instance, the model-instance update log, etc. remain unchanged. <pre>//example task parameters: modelInstances -> {getModelInstance(8000)}</pre>

modelInstances : Set<ModelInstance> A set of finished model instances whose logs are deleted.

Throws:

- *NullParameterError* if mandatory parameter is null
- *ModelInstanceNotFinishedError* if at least one of the specified model instances is not finished

Activate (Re)activates an inactive or finished goal. An active goal remains unchanged. <pre>//example task parameter: goal -> booking::BookedOnlineGoal</pre>

goal : Goal Goal to be (re)activated.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FinishedProcessInstanceError* if the process instance of the activated goal is finished

Deactivate Deactivates an unfinished goal. If the goal is finished or the goal's process is finished, the task type does nothing. <pre>//example task parameter: goal -> booking::BookedOnlineGoal</pre>

goal : Goal Goal to be deactivated.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

RepeatGoals Repeats a set of goals: It deactivates and reactivates the specified achieve goals. If the list includes a pair of achieve goals with one being the subgoal of the other or the process instance of any of the goals is finished, a runtime exception is thrown. <pre>//example task parameter: goals -> [booking::BookedOnlineGoal]</pre>

goals : List<RepeatedGoal> Achieve goals to deactivate and reactivate.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *FinishedProcessInstanceError* if the process instance of the activated goal is finished
- *ParentGoalError* if the goals list contains a pair of goals from which one goal is a subgoal of the other

DeleteSharedRecords Deletes the specified shared records and all related (by data relationships) shared records from the database. Note that it is not possible to delete system shared records. <pre>//example task parameter: data -> {getApplicantByName("John")}</pre>

data : Set<Object> Set of shared records to delete.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the data parameter contains an object which is not a shared record
- *RecordNotFoundError* if the data parameter contains a shared record with an incorrect reference to a database record

RemoveBinaryData[DEPRECATED] Permanently removes the specified binary data from the database. The fields of the *handle* are cleaned up as well (set to the default values). The execution of the task required the `Binary:Delete` security right.

handle : BinaryHandle Binary handle pointing to the binary data to be removed. The fields of the handle are cleaned up as well (set to the default values).

Throws:

- *NullParameterError* if a mandatory parameter is not specified

Lock Creates a model instance-independent lock. The result is stored in the result slot. The lock can be removed byt the Unlock task.

lockName : String Name (identifier) of the lock.

message : String Description of the lock.

result : Reference<Boolean> If true, a new lock was successfully created. If false, a lock with the same name already exists.

existingMessage : Reference<String> If the result is true, the value of the referred string is not changed. If the result is false, the value of the referred string is replaced with the message of the already existing lock with the same name.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

Unlock Removes a lock created by a Lock task, possibly from another model instance.

lockName : String Name of the lock to be removed.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

ParseXml[DEPRECATED] Parses XML document to data structures. <pre>xml -> "<?xml version=""1.0"" encoding=""UTF-8""?> <ns0:RegistrationData xmlns:ns0=""urn:lsps:common"" xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""> <ns0:id>3</ns0:id> <ns0:firstName>John</ns0:firstName> <ns0:lastName>Doe</ns0:lastName> </ns0:RegistrationData>", output -> &parsedXML, useDefaultMapping -> true, typesNamespace -> null</pre>

xml : String XML document to parse.

output : Reference<Object> Reference to where the result is stored. The type of reference provided determines which object is parsed from the XML. The referred object must be of a record type; otherwise an exception is thrown.

useDefaultMapping : Boolean If true, the default LSPS XML mapping is used. Otherwise, the metadata defined on the record types are taken into account. If parsing types generated from XSD, this parameter must be set to false. The default value is false.

typesNamespace : String If the default mapping is used, this parameter determines the XML namespace used for elements. If the default mapping is not used, this parameter is ignored. The default value is null.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the type of the output is not record
- *UnableToParseXmlError* if the specified xml cannot be parsed to output

ConvertToXml[DEPRECATED] Converts data structures to XML document. <pre>//example task parameters: object -> new RegistrationData(firstName -> "Ellen", lastName -> "Doe", approved -> true), output -> &varWithXml, useDefaultMapping -> false, typesNamespace -> null</pre>

object : Object The instance of a record type to be converted to XML.

output : Reference<String> Reference to where to store the output XML.

useDefaultMapping : Boolean If true, the default LSPS XML mapping is used. Otherwise, the meta- data defined on record types are taken into account. If you are converting to XML an object of a type that was generated from XSD, the parameter must be set to false. The default value is false.

typesNamespace : String If the default mapping is used, this parameter determines XML namespace used for elements. If the default mapping is not used, this parameter is ignored. The default value is null.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the type of the object is not record
- *UnableToConvertToXmlError* if an error occured during conversion

HttpCall Performs an HTTP call and stores the response. <pre>httpMethod -> "GET", url -> "http://localhost.com:9500/myIndex/section/_search", request -> "{""query"":{""term"":{""title"":""search for me""}}}", requestContentType -> "application/json", isSynchronous -> true</pre>

httpMethod : String Type of the method to be performed. Possible values are “GET”, “POST”, “PUT”, and “DELETE”.

url : String Endpoint address of the target HTTP service.

request : Object Request payload. Can be either String or core::BinaryHandle. All other objects are converted to String. String is converted to data stream in the charset specified in the requestContentType parameter. If the charset is not specified, UTF-8 is used. If core::BinaryHandle object is used, request payload will contain data from this handle.

requestContentType : String Content type (MIME) of the request payload.

response : Reference<String> Reference to where the response payload is stored.

responseCode : Reference<Integer> Reference to where the response code is stored.

login : String Login for the HTTP BASIC authentication.

password : String Password for the HTTP BASIC authentication.

readTimeout : Integer Socket timeout for the call.

requestHeaders : Map<String, String> Map of the request HTTP headers to be sent together with the request.

responseHeaders : Reference<Map<String, String>> Reference to the map of the HTTP response headers.

isSynchronous : Boolean If true, the task is executed synchronously in the context of process instance. If false or null, the task is executed asynchronously, outside of the process context.

Throws:

- *NullParameterError* if mandatory parameter is null
- *HttpCallError* if an error occured during the HTTP call

Log Logs a specified message to the application log at the specified log level. <pre>//example task parameters: message -> "The item was successfully created.", level -> 100</pre>

message : String Message to be logged.

level : Integer Log level. Unspecified level value logs at the “Info” level.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

Assign[DEPRECATED] Writes the objects to the target slots: map keys define the target slots as references and values define the objects. References in the slots keys must not be null.

slots : Map<Reference<Object>, Object> Map with references to target slots as keys and values to assign to the slots as map values.

Throws:

- *NullParameterError* if a mandatory parameter is not specified
- *IncompatibleTypeError* if the type of the value is not compatible with the type of the target slot

Execute Executes the specified activity. <pre>//example task parameter: activity -> new registration::Approval() //Approval is the Activity of a process with the Create activity reflection type flag</pre>

activity : Activity Activity to execute.

Throws:

- *NullParameterError* if a mandatory parameter is not specified

NoOperation Task with no execution logic. It is intended for prototyping and logical structuring of BPMN processes.