WHITESTEIN Technologies

Living Systems® Process Suite

Standard Library

Living Systems Process Suite Documentation

3.1
Tue Jan 12 2021

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

Contents

Chapter 1

Standard Library

Chapter 2

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.

2.1 Data Types

2.1.1 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 ui::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.

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, ...).

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

2.1.2 Components

UIComponent[ABSTRACT] **extends human::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 Base on return type of content closure, 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: {html, plaintext, preformatted}

for all other types {Boolean, Record, List, Set, Map, Reference, Closure, TypeNull, Object} format is ignored and core::toString() is used to transform 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 A single option of a *List UI components.

value : Object Value itself.

label : String Label for the value.

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 : ui::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.

Tree[DEPRECATED] **extends InputComponent** **children : {Null, Integer : List<TreeItem>}** Returns children of given parent node. Accepts two parameters: first one is the parent data Object reference. Resulting list of items will be nested in this parent. May be null - in this case the root items are returned. Second parameter is the depth of the parent Object in the tree. If parent is null, depth is 0 and all its children will have depth of 1, etc.

binding : Reference<Object> Reference to a slot containing the value.

TreeTable[DEPRECATED] **extends UIComponent** **children : {Null, Integer : List<TreeTableItem>}** Returns children of given parent node. Accepts two parameters: first one is the parent data Object reference. Resulting list of items will be nested in this parent. May be null - in this case the root items are returned. Second parameter is the depth of the parent Object in the tree. If parent is null, depth is 0 and all its children will have depth of 1, etc.

iterator : Reference<Object>

columns : List<TableColumn> The table columns definition.

TreeTableItem[DEPRECATED] **data : Object** The business data reference. May be null.

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

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

GeographicCoordinate 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.

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[ABSTRACT] An abstract type used to define filter of table column.

FilterUI[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.

TableColumnState This record holds column state. It can be used to store or restore the table column's state.

columnId : Object The identifier of the table column.

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

2.1.3 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 ui::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[ABSTRACTSYSTEM] **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 ui::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 ui::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 ui::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 ui::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[SYSTEMDEPRECATED] **extends Event** DEPRECATED.

source : UIComponent The component which has the menu attached

data : Object The value of MenuItem.id

ItemCollapsedEvent[SYSTEMDEPRECATED] **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 ui::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.

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.

2.1.4 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.

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

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.

ValueChangeListener extends Listener Listener which listens for ValueChangeEvents.

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

InitListener extends Listener Listener which listens for InitEvent.

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

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

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

ActionListener extends Listener Listener which listens for ActionEvents.

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

FileDownloadListener extends Listener Listener which listens for FileDownloadEvent.

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

FileUploadListener extends Listener Listener which listens for FileUploadEvents.

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

ApplicationEventListener extends Listener Listener which listens for ApplicationEvents.

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

WidgetChangeListener extends Listener Listener which listens for WidgetChangeEvents.

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

CalendarRescheduleListener extends Listener Listener which listens for CalendarRescheduleEvents.

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

CalendarCreateListener extends Listener Listener which listens for CalendarCreateEvents.

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

CalendarEditListener extends Listener Listener which listens for CalendarEditEvents.

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

MapClickedListener extends Listener Listener which listens for MapClickedEvents.

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

MarkerClickedListener extends Listener Listener which listens for MarkerClickedEvents.

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

MarkerDraggedListener extends Listener Listener which listens for MarkerDraggedEvents.

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

GeolocationListener extends Listener Listener which listens for GeolocationEvents.

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

PopupCloseRequestListener extends Listener Listener which listens for PopupCloseRequestEvents.

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

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

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.

2.2 Functions

2.2.1 UI

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.

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.

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.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

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.

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.

getBrowserWindowSize() : Dimension Retrieves the size, in DPIs, 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.

2.2.2 Dynamic UI

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

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

invoke(targets∗ : Collection<Container>, methodName∗ : String, parameters∗ : List<Object>) : Null Executes "Container methods".

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

refresh(components : Collection<UIComponent>) : Null Slates given components for refresh. The components are refreshed when the listener ends.

persist() : Null Persists immediately.

requestSubmit() : Null Requests submit after all listeners are processed

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

Throws:

- "NullParameterError" if mandatory parameter is null.

merge(viewModels∗ : Collection<ViewModel>) : Null Merges given view models to upper levels (one level up).

Throws:

- "NullParameterError" if mandatory parameter is null.

clear(viewModels∗ : Collection<ViewModel>) : Null Clears given view models.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

createAndShow(def∗ : Popup) : Popup 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.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

addColumn(what∗ : TableColumn, where∗ : UIComponent) : TableColumn 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.

Throws:

- "NullParameterError" if mandatory parameter is null.

removeAndDestroy(what∗ : UIComponent, where : UIComponent) : Null 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.

Throws:

- "NullParameterError" if mandatory parameter is null.

removeAll(container∗ : UIComponent) : Null Removes all children from given container. Only horizontal layout, vertical layout and form layout are supported.

Throws:

- "NullParameterError" if mandatory parameter is null.

getColumns(table∗ : UIComponent) : List<TableColumn> 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.

getColumnStates(table∗ : UIComponent) : List<TableColumnState> Returns the state of columns of a Table or TreeTable2. The state can be restored to the table by calling 'restoreColumnStates' function.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

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.

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.

2.3 Hints

2.3.1 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

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.

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%".

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 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).

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 and UITableColumn. When this hint is applied to the child of the UITableColumn 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 UITableColumn 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

Hint value type:Boolean

Predefined hint options:

Label	Expression
True	true
False (default)	false

If the value is true, table 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.

Chapter 3

Module human

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

The human module imports the core module.

3.1 Data Types

3.1.1 Human

Performer[ABSTRACTSYSTEM] An abstract record type referring to any process performer.

name : String Name (identifier) of the process performer.

Person[SHAREDSYSTEM] **extends Performer** Reference to a person (user of the application). The attribute name inherited from Performer is set to the person's login name.

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 used to commonly represent an organization role or an organization unit.

parameters : Map<String, String> Map of the actual organization unit parameters or organization role parameters. Keys represent the parameter names, values represent the parameter values.

metadata : Map<String, String> Set of metadata (key-value pairs) of the corresponding model element.

Role[SYSTEM] **extends RoleUnit** Reference to an organization role.

OrganizationUnit[SYSTEM] **extends** RoleUnit Reference to an organization unit.

Todo[SHAREDSYSTEM] A record used for representing a to-do.

id : Integer Unique identifier of the to-do.

title : String Title of the to-do.

start : Date Issuing date of the to-do.

finish : Date Date for finishing the to-do, whereas null is used if the to-do is not finished yet.

state : String Name of the current execution state of the to-do. Possible values are: "ALIVE", "ACCOMP↩ LISHED", "INTERRUPTED", and "SUSPENDED".

interruptionReason : String Reason for interruption, if the state is "Interrupted".

task : String Identification of the corresponding task in the process model.

modelInstanceId : Integer Identified of the parent model instance.

allocatedToId : String Identified of the person who allocated to-do. It is null, if the to-do is not allocated.

TodoEscalation[SHAREDSYSTEM] A record used for representing the data of a signal for escalating a to-do.

id : Integer Unique identifier of the to-do escalation.

reason : String Description of the reason for escalation.

todoStatus : String Name of the status of the to-do at the time of its escalation. Possible values are: "Unlocked", "Locked", "Accomplished", and "Interrupted".

time : Date Time of escalation triggering

DocumentType[SYSTEM] A record used for representing the type of document.

name : String Name of the document type.

SavedDocument[SHAREDSYSTEM] A record used for representing a saved document.

id : Integer Unique identifier of the saved document.

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

savedDate : Date The date when the document was saved.

isDeleted : Boolean If true, the saved document has been deleted. If false, the document is still saved.

UIDefinition[ABSTRACT] An abstract record type used to commonly represent any kind of user interface (UI) definition, for instance, UI components or screen flows or any other UI kind defined in future.

TodoListCriteria A record used to specify which to-dos will be used for joining of queried records with person's to-do list (Join Todo List section in query editor).

person : Person A person of which to-dos will be used for joining.

includeSubstituted : Boolean If true, also the to-dos from substituted persons are included. If false, only person's own to-dos

are included.

includeAllocatedByOthers : Boolean If true, also the to-dos already allocated by other persons are included. If false, only unallocated to-dos or to-dos allocated by the specified person are included.

QueryTodo[SHAREDSYSTEM] A record representing a to-do used for joining queried records with person's to-do list (Join Todo List section in query editor). The record contains information about a to-do. Instances of this record are temporary and can be used just in queries.

id : Integer Unique identifier of the to-do.

title : String Title of the to-do.

start : Date Issuing date of the to-do.

finish : Date Date for finishing the to-do, whereas null is used if the to-do is not finished yet.

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

interruptionReason : String Reason for interruption, if the state is "Interrupted".

task : String Identification of the corresponding task in the process model.

allocatedToId : String Identifier of the person who allocated to-do. It is null, if the to-do is not allocated.

modelInstanceId : Integer Identified of the parent model instance.

3.1.2 Navigation

Navigation[ABSTRACT] An abstract record type used to represent any navigation.

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 containing pairs parameter name - parameter value.

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. In the Living Systems Process Application, the following codes are supported: "todoList", "documents", and "runModel". Custom applications can extend and/or change this set of codes.

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 used to represent any navigation in history of browsing application pages. Its concrete sub-types are created by the application and put to the navigation history.

id : String Unique 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 containing pairs parameter name - parameter value.

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

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

code : String Code of the application page. In the Living Systems Process Application, the following codes are supported: "todoList", "documents", and "runModel". Custom applications can extend and/or change this set of codes.

3.2 Functions

3.2.1 To-Dos

getTodoSubmitter(todo∗ : Todo) : Person If the specified todo is in the "Accomplished" state, the function returns the person who submitted that todo. Otherwise, the function returns null.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getTodoPerformers(todo∗ : Todo) : Set<Performer> Returns a set of the original performers of the specified todo, as given by the corresponding task.

Throws:

- "NullParameterError" if mandatory parameter is null.

getTodoCurrentAssignees(todo∗ : Todo) : Set<Person> Returns a set of persons the specified todo is currently issued (visible) to, taking into account also delegation, rejection and substitution.

Throws:

- "NullParameterError" if mandatory parameter is null.

getTodosFor(person∗ : Person) : Set<Todo> Returns a set of active to-dos assigned to the specified person and not allocated by other persons.

Throws:

- "NullParameterError" if mandatory parameter is null.

getTodosFor(person∗ : Person, includeAllocatedByOthers∗ : Boolean) : Set<Todo> Returns a set of active to-dos assigned to the specified person. The parameter includeAllocatedByOthers determines whether the output set contains also to-dos assigned to the person, but currently allocated by other persons.

Throws:

- "NullParameterError" if mandatory parameter is null.

allocateTodo(todo∗ : Todo, person∗ : Person) : Null[SIDE EFFECT] Allocates the specified todo to the person. The given person must belong to the set of persons resolved from the current todo assignees, else an exception is thrown. If the todo is currently allocated to another person, the function performs reallocation.

Throws:

- "NullParameterError" - Mandatory parameter is null.

- "PersonIsNotAssigneeError" - The specified person does not belong to the set of the current toto assignees.

unallocateTodo(todo∗ : Todo) : Null[SIDE EFFECT] Unallocates the specified todo. If the todo is currently not allocated, the function has no effect.

Throws:

- "NullParameterError" - Mandatory parameter is null.

reassignTodo(todo∗ : Todo, performers∗ : Set<Performer>) : Null[SIDE EFFECT] Changes the set of performers of the specified todo. Possible delegations and allocation of the todo are removed. If the todo has already been accomplished, the function has no effect.

Throws:

- “NullParameterError” - Mandatory parameter is null.

rejectTodo(todo∗ : Todo, persons∗ : Set<Person>, reason : String) : Null[SIDE EFFECT] Forces rejection of the todo by the specified persons.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

resetTodo(todo∗ : Todo) : Null Resets the specified todo. The saved state of the todo (if any) will be deleted.

Throws:

- "NullParameterError" - Mandatory parameter is null.

3.2.2 Organization

addPersonToRole(person∗ : Person, roleUnit∗ : RoleUnit) : Null Adds the person to the role.

Throws:

- "NullParameterError" - Mandatory parameter is null.

removePersonFromRole(person∗ : Person, roleUnit∗ : RoleUnit) : Null Remove the person from the role.

Throws:

- "NullParameterError" - Mandatory parameter is null.

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

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "PersonNotFoundError" if there is no such a person.

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

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "PersonNotFoundError" if there is no such a person.

getPersonFullName(person∗ : Person) : String Returns the full name of the specified person.

Throws:

- "NullParameterError" if mandatory parameter is null.

getPersonPicture(person∗ : Person) : File Returns a profile picture of person.

Throws:

- "NullParameterError" if mandatory parameter is not specified.

setPersonPicture(person∗ : Person, picture : File) : Null[SIDE EFFECT] Sets or changes a profile picture for the person. If the picture is null, the person has no profile picture.

Throws:

- "NullParameterError" if mandatory parameter is not specified.

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

Throws:

- "NullParameterError" if mandatory parameter is null.

getCurrentPerson() : Person Returns the person who has initiated the current model processing request. For instance, the person who has submitted a to-do or called a web service. In cases of receiving time events or signals the technical person “ProcessAgent” is returned.

anyPerformer() : Performer Returns a special performer representing any known process performer. This function is used to specify access to entities (e.g., to-dos, reports, or widgets) available to all known process performers.

getRoleUnitByName(module∗ : String, name∗ : String) : RoleUnit Returns an organization role or an organization unit specified by their name defined in the given module.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "RoleUnitNotFoundError" if there is no such a role or a unit.

getRoleUnitByName(module∗ : String, name∗ : String, parameters : Map<String, String>) : RoleUnit
Returns an organization role or an organization unit specified by their name and parameters defined in the given module.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "RoleUnitNotFoundError" if there is no such a role or a unit.

children(roleUnit∗ : RoleUnit) : Set<RoleUnit> Returns a set of direct children of an organization role or unit referred to by the roleUnit parameter. In the case of a parametric RoleUnit the values of children's parameters are derived from the specified roleUnit's parameters; i.e., the value of a particular child's parameter is identical with the value of the parent's parameter of the same name.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

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

Throws:

- "NullParameterError" if mandatory parameters are not specified.

isPersonStrictlyIn(person∗ : Person, role∗ : Role) : Boolean Returns true, if the given person plays strictly the specified role, but none of its sub-roles. Therefore, this is a non-recursive version of the isPersonIn() function applied to organization roles.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

personsWith(roleUnit∗ : RoleUnit) : Set<Person> Returns a set of all persons that belong to the given roleUnit in the current organization model. A person "belongs" to a given organization role or unit if it belongs to it directly or belongs to any of its descendants (evaluated recursively).

Throws:

- "NullParameterError" if mandatory parameters are not specified.

personsStrictlyWith(role∗ : Role) : Set<Person> Returns a set of all persons that play the given role, but none of its sub-roles. Therefore, this is a non-recursive version of the personsWith() function applied to organization roles.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getPersonRoles(person∗ : Person) : Set<Role> Returns a set of organization roles from the current model directly assigned to the specified person.

Throws:

- "NullParameterError" if mandatory parameter is null.

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

emailAddresses(performers∗ : Set<Performer>) : Set<String> Returns a set of email addresses of the performers. If a performer is of 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.

Throws:

- "NullParameterError" if mandatory parameter is null.

3.2.3 Documents

deleteSavedDocuments(documents∗ : Set<SavedDocument>) : Null[SIDE EFFECT] Deletes the specified saved documents.

Throws:

- “NullParameterError” - Mandatory parameter is null.

3.2.4 Utilities

sendEmail(subject : String, body∗ : String, attachments : Set<File>, recipientsTo∗ : Set<String>, recipientsCc : Set<Strir Sends an e-mail.

Deprecated: use function sendEmail with extended parameters.

Throws:

- "NullParameterError" - 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.

Throws:

- "NullParameterError" - Mandatory parameter is null.

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

Throws:

- "NullParameterError" - Mandatory parameter is null.

getUIHistory() : Map<Integer, HistoricalNavigation> A function which returns UI history. Under key 0 is always "current" page. Under positive keys are older entries (back), under negative keys less old entries (forward).

3.3 Tasks

3.3.1 Human

User Task type used to create form-based to-dos. It enables to simultaneously present information to users and to collect their input.

title : String The title of the generated to-do.

performers : Set<Performer> Specification of those 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 : Object} Closure executed after the to-do is generated. The generated to-do is passed as input parameter of the closure.

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

Throws:

-"NullParameterError" if mandatory parameter is null.

AddPersonToRole Adds a person to a role.

person : Person Person to be added to the role.

role : Role Role the person is added to. If the role does not exist in the system yet, it is created.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

RemovePersonFromRole Removes the person from the role.

person : Person Person that will be removed from the role.

role : Role Role that the person should be removed from.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

SendEmail Sends an email. For configuration of the mail server see the implementation of the task.

subject : String email subject

body : String email body paragraphs

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 2 platform, see Java docu- mentation for details.

Chapter 4

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.

This module is necessary for proper functioning of GO-BPMN models containing expressions and it is, therefore, recommended to include it (directly or indirectly) into each custom module.

4.1 Constants

4.1.1 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 e-mail address.

4.2 Constraint Types

4.2.1 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 or greater than the specified lowerBound.

Max(upperBound : Decimal, message : String) Applied to: Decimal

The Decimal value must be equal or less than the specified upperBound.

Range(lowerBound : Decimal, upperBound : Decimal, message : String) Applied to: Decimal

The Decimal value must be in range given by interval [lowerBound, upperBound] (inclusive).

MinLength(min : Integer, message : String) Applied to: String

The String value must be of length equal or greater than the specified min value.

MaxLength(max : Integer, message : String) Applied to: String

The String value must be of length equal 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 match to 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 to e-mail format.

Past(before : Date, message : String) Applied to: Date

The Date value must be the same or older than the specified before parameter.

Future(after : Date, message : String) Applied to: Date

The Date value must be the same or newer than the specified after parameter.

CardinalityMin(min : Integer, condition : {T : Boolean}, message : String) Applied to: Collection<T>

Cardinality of the Collection items satisfying the specified condition must be at least as the specified min.

CardinalityMax(max : Integer, condition : {T : Boolean}, message : String) Applied to: Collection<T>

Cardinality of the Collection items satisfying the specified condition must be at most as the specified max.

CardinalityRange(min : Integer, max : Integer, condition : {T : Boolean}, message : String) Applied to: Collection<T>

Cardinality of the Collection items satisfying the specified condition must be in interval [min, max] (inclusive).

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 value must be valid.

4.3 Data Types

4.3.1 Core

GoalPlan[ABSTRACTSYSTEM] An abstract record type used to commonly 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 valid in the current module instance.
- **metadata : Map<String, String>** Set of metadata (key-value pairs) of the element.

Goal[ABSTRACTSYSTEM] **extends GoalPlan** Represents a single goal (of any kind).

- **children : Set<GoalPlan>** Set of children.
- **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 running direct sub-plans. Therefore, it is an empty set for a goal decomposed to goals.

AchieveGoal[SYSTEMFINAL] **extends Goal** Represents a single achieve goal.

- **preCondition : { : Boolean}** Precondition of the achieve goal.
- **deactivateCondition : { : Boolean}** Deactivate condition of the achieve goal.

MaintainGoal[SYSTEMFINAL] **extends Goal** Represents 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] Represents a model.

- **id : Integer** Unique identifier of the model.

name : String Name of the model.

version : String Version of the model.

uploadDate : Date Model's upload date.

ModelInstance[SHAREDSYSTEM] Represents 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 Starting date of the model instance.

finish : Date Date of finishing the model instance, whereas null is used if the model instance is not finished yet.

ProcessInstance[SYSTEM] Represents a process instance.

id : Integer Identifier of process instance.

process : String Name of the instantiated 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 corresponding process model element.

Duration Represents a duration expressed in various time units. The duration value is given as the sum of all values of different units. Each field value can be either 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.

File Binary file.

content : Binary Binary content.

filename : String Filename of the binary 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. This shared record is also used internally by LSPS server to store some binary data. Therefore, modification and deleting of BinaryHandle records must be performed carefully.

id : Integer Unique identifier of the binary data.

description : String Description of the data.

RepeatedGoal Represents 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.

ConstraintViolation A record type used to report violation of a data constraint. Execution of a single validation results in a list of constraint violations.

id : String ID of the associated constraint. It is assigned automatically right after constraint execution to all created ConstraintViolation objects that have null id.

guid : String GUID of the associated constraint. It is assigned automatically right after constraint execution to all created ConstraintViolation objects that have null guid.

message : String Message returned by constraint expression defined in corresponding ConstraintType.

record : Record Record which did not pass the validation.

property : Property Optional property of the record which did not pass the validation.

payload : Object Optional application data.

Tag A record type used to classify data constraints and their runtime filtering in different validations.

name : String A full name of the validation tag (e.g. "mymodule::mytag")

Activity

UserTrack$^{\text{SHAREDSYSTEM}}$ Record for tracking of active user.

id : Integer

userLogin : String

startDate : Date

endDate : Date

4.4 Functions

4.4.1 Collection

add(list∗ : List<E>, elements : E...) : List<E> Returns a list created by adding the specified elements at the end of the list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

add(set∗ : Set<E>, elements : E...) : Set<E> Returns a set created by adding the specified elements to the set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

add(collection∗ : Collection<E>, elements : E...) : Collection<E> Returns a collection created by adding the specified elements at the end of 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

add(map∗ : Map<K, V>, key : K, value : V) : Map<K, V> Returns a map created by adding the specified key-value pair to the map. If the key already exists in the map, the specified value overwrites the original one.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

addAll(list∗ : List<E>, index∗ : Integer, elements∗ : Collection<E>...) : List<E> Returns a list created by adding the specified elements (given as collections) to the list, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and any subsequent elements to the right and increments the indices of the shifted elements with the number of the elements added to the specified list. The original order of the added elements is preserved also in the returned list.

Throws:

- "NullParameterError" if mandatory parameters are 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 created by adding the specified elements to the set parameter, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and any subsequent elements to the right and increments the indices of the shifted elements with the number of the different elements added to the specified set. The original order of the added elements is preserved also in the returned set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index $<$ 0 or index $>$ size(set)).

addAll(collection∗ : Collection<E>, index∗ : Integer, elements∗ : Collection<E>...) : Collection<E> Returns a collection created by adding the specified elements to the collection parameter, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and any subsequent elements to the right and increments the indices of the shifted elements with the number of the elements added to the specified collection. The original order of the added elements is preserved also in the returned 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index $<$ 0 or index $>$ size(collection)).

addAll(list∗ : List<E>, collections∗ : Collection<E>...) : List<E> Returns a list created as concatenation of the list and collections.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

addAll(set∗ : Set<E>, collections∗ : Collection<E>...) : Set<E> Returns a set created by concatenation of the set and collections.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

addAll(collections∗ : Collection<E>...) : Collection<E> Returns a list created as concatenation of the specified lists. 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

addAll(maps∗ : Map<K, V>...) : Map<K, V> Returns a map created as union of keys and their respective values from all specified 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 containing that key.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

addAt(list∗ : List<E>, index∗ : Integer, elements : E...) : List<E> Returns a list created by adding the specified elements to the list, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and its subsequent elements (if any) to the right and increments the indices of the shifted elements with the number of the elements added to the list.

Throws:

- "NullParameterError" if mandatory parameters are 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 specified elements to the set parameter, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and its subsequent elements (if any) to the right and increments the indices of the shifted elements with the number of the different elements added to the set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the 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 specified elements to the collection, starting from the position specified by the index. Shifts the element currently at that position (if there is any) and its subsequent elements (if any) to the right and increments the indices of the shifted elements with the number of the elements added to the list. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the index is out of range (index $<$ 0 or index $>$ size(collection)).

collect(list∗ : List<E>, function∗ : {E : T}) : List<T> Returns a list of values obtained by applying the function to each item of the list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

collect(set∗ : Set<E>, function∗ : {E : T}) : Set<T> Returns a set of values obtained by applying the function to each item of the set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

collect(collection∗ : Collection<E>, function∗ : {E : T}) : Collection<T> Returns a collection of values obtained by applying the function to each item of 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

collectWithIndex(list∗ : List<E>, function∗ : {Integer, E : T}) : List<T> Returns a list of values obtained by applying the function to each item of the list.

The function parameter takes 2 values: index of the item and the item itself.

Throws:

- "NullParameterError" - Mandatory parameter is null.

collectWithIndex(set∗ : Set<E>, function∗ : {Integer, E : T}) : Set<T> Returns a set of values obtained by applying the function to each item of the set. The function parameter takes 2 values: index of the item and the item itself.

Throws:

- "NullParameterError" - Mandatory parameter is null.

collectWithIndex(collection∗ : Collection<E>, function∗ : {Integer, E : T}) : Collection<T> Returns a collection of values obtained by applying the function to each item of the collection. The function parameter takes 2 values: index of the item and the item itself.

Throws:

- "NullParameterError" - Mandatory parameter is null.

compact(list∗ : List<E>) : List<E> Returns a list containing only non-null elements from a given list.

Throws:

- "NullParameterError" - Mandatory parameter is null.

compact(set∗ : Set<E>) : Set<E> Returns a set containing only non-null elements from a given set.

Throws:

- "NullParameterError" - Mandatory parameter is null.

compact(collection∗ : Collection<E>) : Collection<E> Returns a collection containing only non-null elements from a given collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

contains(collection∗ : Collection<Object>, elements : Object...) : Boolean Returns true, if the collection contains all specified elements.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

containsAll(collection∗ : Collection<Object>, elements∗ : Collection<Object>) : Boolean Returns true, if the collection contains all specified elements.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

containsKeys(map∗ : Map<Object, Object>, keys : Object...) : Boolean Returns true, if the map contains all specified keys.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

containsValues(map∗ : Map<Object, Object>, values : Object...) : Boolean Returns true, if the map contains all specified values.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

count(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the number of elements in the given collection which satisfy the given condition.

Throws:

- "NullParameterError" if mandatory parameters are 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

findIndex(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the index of the first occurrence of the collection element which satisfies the specified condition; returns -1 if the condition is false for all elements of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

findIndex(collection∗ : Collection<E>, fromIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the first occurrence of the collection element, starting at fromIndex (inclusive), which satisfies the specified condition; returns -1 if the condition is false for all elements of the collection starting at fromIndex. If the fromIndex parameter is null or is less than 0, the finding starts from the beginning of the collection (index 0).

Throws:

- "NullParameterError" - Mandatory parameter is null.

findIndex(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the first occurrence of the collection element, starting at the fromIndex (inclusive) and ending at the toIndex (exclusive), which satisfies the specified condition; returns -1 if the condition is false for all elements in the specified range. If the fromIndex parameter is null or is less than 0, the finding starts from the beginning of the collection (index 0). If the toIndex parameter is null or greater or equal to the size of the collection, the finding ends at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

findLastIndex(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Integer Returns the index of the last occurrence of the collection element which satisfies the specified condition; returns -1 if the condition is false for all elements of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

findLastIndex(collection∗ : Collection<E>, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the last occurrence of the collection element which satisfies the specified condition, searching backwards starting at index toIndex - 1; returns -1 if no such element is found. If the toIndex parameter is null or greater or equal to the size of the collection, the finding starts at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

findLastIndex(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer, condition∗ : {E : Boolean}) : Integer Returns the index of the last occurrence of the collection element, searching backwards in the range starting at fromIndex (inclusive) and ending at toIndex (exclusive), which satisfies the specified condition; returns -1 if the condition is false for all elements in the specified range. If the fromIndex parameter is null or is less than 0, the search ends at the beginning of the collection (index 0). If the toIndex parameter is null or greater or equal to the size of the collection, the finding starts at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

fold(collection∗ : Collection<E>, initialValue : T, function∗ : {T, E : T}) : T Combines the elements of the collection together using the binary function, from left to right, and starting with the initialValue. The function is applied iteratively on the previously computed interim value and a next element of the collection. In each iteration the function computes the subsequent interim value. The last computed value represents the result of the fold function.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

foldWithIndex(collection∗ : Collection<E>, initialValue : T, function∗ : {Integer, T, E : T}) : T Combines the elements of the collection together using the binary function, from left to right, and starting with the initial↩ Value. The function is applied iteratively on the previously computed interim value and a next element of the collection. In each iteration the function computes the subsequent interim value. The last computed value represents the result of the fold function. 0-based index of element is passed to function as the first argument.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

forAll(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Boolean Returns true, if the condition is true for all elements in the collection.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getFirst(collection∗ : Collection<E>) : E Returns the first item of the collection, or null if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

getItemType(collectionType∗ : Type<Collection<Object>>) : Type<Object> Returns the inner Type of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.

getLast(collection∗ : Collection<E>) : E Returns the last item of the collection, or null if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

hasDuplicates(collection∗ : Collection<Object>) : Boolean Returns true if the collection contains duplicate elements, false otherwise.

Throws:

- "NullParameterError" - 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

isEmpty(collection∗ : Collection<Object>) : Boolean Returns true, if the specified collection contains no elements.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

isEmpty(map∗ : Map<Object, Object>) : Boolean Returns true, if the specified map contains no elements.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

isSubset(set1∗ : Set<Object>, set2∗ : Set<Object>) : Boolean Returns true, if the set1 is a subset of the set2.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

join(collection∗ : Collection<Object>) : String Concatenates all non-empty string representations of the objects within collection, separating them by comma and space (", ").

Throws:

- "NullParameterError" if mandatory parameters are not specified.

join(collection∗ : Collection<Object>, joinString : String) : String Concatenates all non-empty string representations of the objects within collection, separating them by joinString.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

join(collection∗ : Collection<Object>, joinString : String, includeEmpty : Boolean) : String Concatenates all string representations of the objects within collection, separating them by joinString. If includeEmpty is true, also null and empty strings are included in the result string.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

keys(map∗ : Map<K, V>) : Set<K> Returns a set of keys contained in the specified map.

Throws:

- "NullParameterError" if mandatory parameters are 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.

Throws:

- "NullParameterError" if mandatory parameters are 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

listUnion(collections : Collection<Collection<E>>...) : List<E> Returns a list containing all elements from all given collections. Null values of collection are treated as empty collections.

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 created map. If the keys parameter contains the same key several times, the most recently used key-value pair is included in the resulting map.

Throws:

- "NullParameterError" if mandatory parameters are 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(keys∗ : Collection<K>, valueFunction∗ : {K : V}) : Map<K, V> Returns a map by combining given keys and computed values. For each key in the given collection of keys, the given closure is executed to obtain a map value.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

map(collection∗ : Collection<E>, keyFunction∗ : {E : K}, valueFunction∗ : {E : V}) : Map<K, V> Returns a map by combining computed keys and computed values. For each element in the given collection, the keyFunction is executed to obtain a map key and valueFunction is executed to obtain a map value.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

max(collection∗ : Collection<Date>) : Date Returns maximal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

max(collection∗ : Collection<Decimal>) : Decimal Returns maximal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

max(collection∗ : Collection<Integer>) : Integer Returns maximal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

max(collection∗ : Collection<String>) : String Returns maximal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

min(collection∗ : Collection<Date>) : Date Returns minimal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

min(collection∗ : Collection<Decimal>) : Decimal Returns minimal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

min(collection∗ : Collection<Integer>) : Integer Returns minimal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

min(collection∗ : Collection<String>) : String Returns minimal value from collection; or null, if the collection is empty.

Throws:

- "NullParameterError" - Mandatory parameter is null.

notEmpty(collection∗ : Collection<Object>) : Boolean Returns true, if the specified collection contains some elements.

Throws:

- "NullParameterError" - Mandatory parameter is null.

notEmpty(map∗ : Map<Object, Object>) : Boolean Returns true, if the specified map contains some elements.

Throws:

- "NullParameterError" - Mandatory parameter is null.

remove(list∗ : List<E>, elements : Object...) : List<E> Returns a list created by removing the first occurrences of the specified elements from the list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

remove(set∗ : Set<E>, elements : Object...) : Set<E> Returns a set created by removing the specified elements from the set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

remove(collection∗ : Collection<E>, elements : Object...) : Collection<E> Returns a collection created by removing the first occurrences of the specified elements from 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeAll(list∗ : List<E>, elements∗ : Collection<Object>) : List<E> Returns a list created by removing all occurrences of the elements from the list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeAll(set∗ : Set<E>, elements∗ : Collection<Object>) : Set<E> Returns a set created by removing all elements from the set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeAll(collection∗ : Collection<E>, elements∗ : Collection<Object>) : Collection<E> Returns a collection created by removing all occurrences of the elements from 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeAt(list∗ : List<E>, index∗ : Integer) : List<E> Returns a list created by removing that element of the list, which is at the position specified by the index.

Throws:

- "NullParameterError" if mandatory parameters are 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> Returns a set created by removing that element of the set, which is at the position specified by the index.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index $<$ 0 or index $>=$ size(set)).

removeAt(collection∗ : Collection<E>, index∗ : Integer) : Collection<E> Returns a collection created by removing that element of the collection, which is at the position specified by the index. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index < 0 or index >= size(collection)).

removeKey(map∗ : Map<K, V>, keys : Object...) : Map<K, V> Returns a map created by removing the specified keys (and their corresponding values) from the map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeKeys(map∗ : Map<K, V>, keys∗ : Collection<Object>) : Map<K, V> Returns a map created by removing the specified keys (and their corresponding values) from the map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

removeRange(set∗ : Set<E>, fromIndex : Integer, toIndex : Integer) : Set<E> Returns a portion of the set representing removing elements from the specified fromIndex (inclusive) to the toIndex (exclusive). If the fromIndex parameter is null or is less than 0, the range starts from the beginning of the collection (index 0). If the toIndex parameter is null or greater or equal to the size of the collection, the range ends at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "OutOfBoundsError" - fromIndex > toIndex.

removeRange(list∗ : List<E>, fromIndex : Integer, toIndex : Integer) : List<E> Returns a portion of the list representing removing elements from the specified fromIndex (inclusive) to the toIndex (exclusive). If the fromIndex parameter is null or is less than 0, the range starts from the beginning of the collection (index 0). If the toIndex parameter is null or greater or equal to the size of the collection, the range ends at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "OutOfBoundsError" - fromIndex > toIndex.

removeRange(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer) : Collection<E>
Returns a portion of the collection representing removing elements from the specified fromIndex (inclusive) to the toIndex (exclusive). If the fromIndex parameter is null or is less than 0, the range starts from the beginning of the collection (index 0). If the toIndex parameter is null or greater or equal to the size of the collection, the range ends at the end of the collection.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "OutOfBoundsError" - fromIndex > toIndex.

removeValue(map∗ : Map<K, V>, values : Object...) : Map<K, V> Returns a map created by removing all occurrences of the specified values (and their corresponding keys) from the map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

replace(list∗ : List<E>, element1 : Object, element2 : E) : List<E> Returns a list created by replacing all occurrences of the element1 with the element2 in the specified list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

replace(set∗ : Set<E>, element1 : Object, element2 : E) : Set<E> Returns a set created by replacing the element1 with the element2 in the specified set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

replace(collection∗ : Collection<E>, element1 : Object, element2 : E) : Collection<E> Returns a collection created by replacing all occurrences of the element1 with the element2 in the specified 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

replaceValue(map∗ : Map<K, V>, value1 : Object, value2 : V) : Map<K, V> Returns a map created by replacing all occurrences of the values corresponding to the value1 with the value2 in the specified map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

retainAll(list∗ : List<E>, collections∗ : Collection<E>...) : List<E> Returns a list of those elements of the list that are contained in the specified collections. The relative order of the retained elements remains unchanged.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

retainAll(set∗ : Set<E>, collections∗ : Collection<E>...) : Set<E> Returns a set created by intersection of the specified sets.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

retainAll(collection∗ : Collection<E>, collections∗ : Collection<E>...) : Collection<E> Returns a list of those elements of the collection that are contained in the specified 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

reverse(list∗ : List<E>) : List<E> Returns a list created by reversing the order of the elements in the specified list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

reverse(set∗ : Set<E>) : Set<E> Returns a set created by reversing the order of the elements in the specified set.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

reverse(collection∗ : Collection<E>) : Collection<E> Returns a collection created by reversing the order of the elements in the specified collection.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

select(list∗ : List<E>, condition∗ : {E : Boolean}) : List<E> Returns a list containing those elements from the list for which the condition is true.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

select(set∗ : Set<E>, condition∗ : {E : Boolean}) : Set<E> Returns a set containing those elements from the set for which the condition is true.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

select(collection∗ : Collection<E>, condition∗ : {E : Boolean}) : Collection<E> Returns a colection containing 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

select(map∗ : Map<K, V>, condition∗ : {K, V : Boolean}) : Map<K, V> Returns a map containing those elements from the map for which the condition is true.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

setAt(list∗ : List<E>, index∗ : Integer, element : E) : List<E> Returns a list created by replacing that element of the list, which is at the position given by the index, with the specified element.

Throws:

- "NullParameterError" if mandatory parameters are 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> Returns a set created by replacing that element of the set, which is at the position given by the index, with the specified element.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index $<$ 0 or index $>=$ size(set)).

setAt(collection∗ : Collection<E>, index∗ : Integer, element : E) : Collection<E> Returns a list created by replacing that element of the list, which is at the position given by the index, with the specified element. If the input collection is a set, the returned type is Set. If the input collection is a list, the returned type is List.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified index is out of range (index $<$ 0 or index $>=$ size(collection)).

size(collection∗ : Collection<Object>) : Integer Returns the number of elements in the specified collection.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

size(map∗ : Map<Object, Object>) : Integer Returns the number of elements in the specified map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

sort(list∗ : List<E>, comparator∗ : {E, E : Integer}) : List<E> Returns items of the specified list ordered according to the given comparator. The comparator defines a total ordering function returning a negative integer, zero, or a positive integer if the first parameter is less than, equal to, or greater than the second, respectively.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

sort(set∗ : Set<E>, comparator∗ : {E, E : Integer}) : Set<E> Returns items of the specified set ordered according to the given comparator. The comparator defines a total ordering function returning a negative integer, zero, or a positive integer if the first parameter is less than, equal to, or greater than the second, respectively.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

sort(collection∗ : Collection<E>, comparator∗ : {E, E : Integer}) : Collection<E> Returns items of the specified collection ordered according to the given comparator. The comparator defines a total ordering function returning a negative integer, zero, or a positive integer if the first parameter is less than, equal to, or greater than the second, respectively. If the input collection is a set, the returned

Throws:

- "NullParameterError" if mandatory parameters are not specified.

sort(set∗ : Set<Date>, ascending∗ : Boolean) : Set<Date> Returns a set with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(set∗ : Set<Decimal>, ascending∗ : Boolean) : Set<Decimal> Returns a set with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(set∗ : Set<Integer>, ascending∗ : Boolean) : Set<Integer> Returns a set with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(set∗ : Set<String>, ascending∗ : Boolean) : Set<String> Returns a set with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(list∗ : List<Date>, ascending∗ : Boolean) : List<Date> Returns a list with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(list∗ : List<Decimal>, ascending∗ : Boolean) : List<Decimal> Returns a list with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(list∗ : List<Integer>, ascending∗ : Boolean) : List<Integer> Returns a list with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(list∗ : List<String>, ascending∗ : Boolean) : List<String> Returns a list with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(collection∗ : Collection<Date>, ascending∗ : Boolean) : Collection<Date> Returns a collection with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(collection∗ : Collection<Decimal>, ascending∗ : Boolean) : Collection<Decimal> Returns a collection with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(collection∗ : Collection<Integer>, ascending∗ : Boolean) : Collection<Integer> Returns a collection with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

sort(collection∗ : Collection<String>, ascending∗ : Boolean) : Collection<String> Returns a collection with items sorted in ascending or descending order.

Throws:

- "NullParameterError" - Mandatory parameter is null.

subList(list∗ : List<E>, fromIndex : Integer, toIndex : Integer) : List<E> Returns a list representing such a portion of the list, which starts from the specified fromIndex (inclusive) to the toIndex (exclusive). If the from↩ Index parameter is null or is less than 0, the sub-list starts from the beginning of the list (index 0). If the toIndex parameter is null or equal/greater than the size of the list, the sub-list ends at the end of the list (index given by size(list)-1).

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if fromIndex $>$ toIndex.

subCollection(collection∗ : Collection<E>, fromIndex : Integer, toIndex : Integer) : Collection<E> Returns a collection representing such a portion of the collection, which starts from the specified fromIndex (inclusive) to the toIndex (exclusive). If the fromIndex parameter is null or is less than 0, the sub-list starts from the beginning of the list (index 0). If the toIndex parameter is null or equal/greater than the size of the collection, the sub-collection ends at the end of the collection (index given by size(collection)-1).

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if fromIndex $>$ toIndex.

subset(set∗ : Set<E>, fromIndex : Integer, toIndex : Integer) : Set<E> Returns a set representing such a portion of the collection, which starts from the specified fromIndex (inclusive) to the toIndex (exclusive). If the fromIndex parameter is null or is less than 0, the sub-list starts from the beginning of the list (index 0). If the toIndex parameter is null or equal/greater than the size of the collection, the sub-collection ends at the end of the collection (index given by size(set)-1).

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if fromIndex > toIndex.

sum(collection∗ : Collection<E>) : E Returns a sum of decimal or integer numbers in the given collection.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

toList(collection∗ : Collection<E>) : List<E> Returns a list of values, in an arbitrary order, contained in the specified collection.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

toSet(collection∗ : Collection<E>) : Set<E> Returns a set of values contained in the specified list.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

values(map∗ : Map<K, V>) : List<V> Returns a list of values contained in the specified map.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

4.4.2 Binary Data

createBinaryHandle(fileName∗ : String, description : String, content∗ : String, mime : String, charset : String) : BinaryHand Creates a binary handle representing a newly created file with the specified fileName, content, description, mime type and charset encoding. If the mime parameter is not specified, the MIME type is determined from the file extension automatically. If the charset is not specified, "UTF-8" is used by default.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "UnsupportedEncodingError" - The specified charset is not supported.

getBinaryHandle(id∗ : Integer) : BinaryHandle[DEPRECATED] Returns a binary handle of the specified identifier.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "BinaryHandleNotFoundError" if there is no such a binary handle.

getBinaryHandleMetadata(binaryData∗ : core::BinaryHandle) : Map<String, String> Returns a binary handle metadata

Throws:

- "NullParameterError" if mandatory parameter is null.

resource(module∗ : String, path∗ : String) : BinaryHandle[SIDE EFFECTDEPRECATED] Creates a binary handle from a module's resource located at the specified path.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "ResourceNotFoundError" if the specified module is not available or the resource is not found at the given path.

getResource(module∗ : String, path∗ : String) : File Creates a file from a module's resource located at the specified path.

Throws:

- "NullParameterError" if 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 data in bytes.

toBinary(string : String) : Binary Converts the specified string to binary.

toString(binary∗ : Binary, charset : String) : String Converts binary data to a string encoded by given charset. Standard charset strings are defined in java.nio.charset.StandardCharsets.

Possible values: US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16, etc.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "UnsupportedEncodingError" if encoding is unsupported.

4.4.3 Enumeration

literals(enumeration∗ : Type<E>) : List<E> Returns a list of literals from the specified enumeration.

Throws:

- “NullParameterError” if mandatory parameter is null.

literalToName(literal∗ : Enumeration) : String Transforms the specified enumeration literal to its name represented as string.

Throws:

- “NullParameterError” if mandatory parameter is null.

nameToLiteral(enumeration∗ : Type<E>, name∗ : String) : E Transforms the specified enumeration’s literal name to the corresponding literal value.

Throws:

- “NullParameterError” if mandatory parameter is null.
- “DoesNotExistError” if the specified name does not corespond to any literal.

4.4.4 Validation

validate(record : Record, property : Property, tags : Collection<Tag>, context : Map<String, Object>) : List<ConstraintVio
Performs a validation of the given record.

If the record is null, an empty list is returned. If the given property parameter is null, all properties of the record are validated.

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<
Performs a validation of the given collection of records.

If the collection is null, an empty list is returned. Null elements in the collection are ignored. If the given property parameter is null, all properties of the records are validated.

Throws:

- "IncompatibleTypeError" - if the given property is not a property of any given record in the collection

findTag(name∗ : String) : Tag Finds a tag by a given name which can be a simple name or full name. Returns <code>null</code> if the tag is not found. If the simple name is given and it is ambiguous (meaning there are more validation tags with the same name located in different imported modules), an error is thrown.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "AmbiguousNameError" if the given name is ambiguous

4.4.5 String

length(string∗ : String) : Integer Returns the length of the specified string.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

toInteger(string∗ : String) : Integer If possible, converts the specified string to an integer. If not possible, the function throws a runtime exception.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "FormatError" if the string does not represents an integer value.

toDecimal(string∗ : String) : Decimal If possible, converts the specified string to a decimal. If not possible, the function throws a runtime exception.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "FormatError" if the string does not represent a decimal value.

toString(object : Object) : String Converts the specified object to a string.

toLowerCase(string∗ : String) : String Converts all characters of the specified string to lower case.

Throws:

- "NullParameterError" if mandatory parameter is null.

toUpperCase(string∗ : String) : String Converts all characters of the specified string to upper case.

Throws:

- "NullParameterError" if mandatory parameter is null.

indexOf(string∗ : String, substring∗ : String) : Integer Returns the index of the first occurrence of the specified substring within the string.

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 string, starting at the specified index.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "OutOfBoundsError" if the index is out of range (index $<$ 0 or index $>=$ length(string)).

lastIndexOf(string∗ : String, substring∗ : String) : Integer Returns the index of the rightmost occurrence of the specified substring within the string.

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.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "OutOfBoundsError" if the index is out of range (index $<$ 0 or index $>=$ length(string)).

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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

substring(string∗ : String, fromIndex : Integer, toIndex : Integer) : String Returns a substring of the specified string starting at the position (inclusive) given by the fromIndex parameter and ending at the position (exclusive) given by the toIndex parameter. If the specified fromIndex is null or is less than 0, the substring starts from beginning of the string (index 0). If the specified toIndex is null or equal/higher than the length of the string, the substring ends at the end of the string (index given by length(string)-1). Throws a runtime exception if fromIndex $>$ toIndex.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if fromIndex $>$ toIndex.

replaceFirst(string∗ : String, regexp∗ : String, replacement∗ : String) : String Replaces the first substring of the specified string that matches the regular expression given by the regexp parameter with the string specified by the replacement parameter. The Java regular expression syntax (as defined in the java.util.regex.↩ Pattern class) is used.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

replaceAll(string∗ : String, regexp∗ : String, replacement∗ : String) : String Replaces each substring of the specified string that matches the regular expression given by the regexp parameter with the string specified by the replacement parameter. The Java regular expression syntax (as defined in the java.util.regex.Pattern class) is used.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

split(string∗ : String, regexp∗ : String) : List<String> Splits the specified string around matches of the regular expression given by the regexp parameter. The Java regular expression syntax (as defined in the java.util.↩ regex.Pattern class) is used.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

find(string∗ : String, regexp∗ : String) : List<String> Finds all occurrences of the regexp in the specified string. All the matched strings are returned as a list of strings. The function returns an empty list if there is no match. The Java regular expression syntax (as defined in the java.util.regex.Pattern class) is used.

Throws:

- "NullParameterError" - if mandatory parameter is null.

format(pattern∗ : String, arguments : Object...) : String Returns a string created by formatting the objects given by the arguments parameter with the format specified by the pattern parameter. The pattern parameter uses the syntax defined in the java.util.Formatter Java class restricted to formatting commands applied only for Java data types used to internally represent GO-BPMN expression language data types. The data types are transformed according to the following table:

Boolean -> java.lang.Boolean

String -> java.lang.String

Date -> java.util.Date

Integer -> java.math.BigDecimal

Decimal -> java.math.BigDecimal

all other data types -> java.lang.String using the function core::toString

The format() function uses a server locale for formatting numbers and dates.

Note: The format() function returns incorrect value of time zone offset from GMT for Daylight Saving Time (“$tz” or “$Tz” conversions) on Java 6. It is recommended to use formatDate() function for formatting dates. Java 7 returns correct values of time zone offset.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "FormatError" if the specified format is incorrect.

formatWithLocale(pattern∗ : String, locale : String, arguments : Object...) : String Returns a string created by formatting the objects given by the arguments parameter with the format specified by the pattern parameter. The pattern parameter uses the syntax defined in the java.util.Formatter Java class restricted to formatting commands applied only for Java data types used to internally represent GO-BPMN expression language data types. The data types are transformed according to the following table:

Boolean -> java.lang.Boolean

String -> java.lang.String

Date -> java.Functions.Date

Integer -> java.math.BigDecimal

Decimal -> java.math.BigDecimal

all other data types -> java.lang.String using the function core::toString

The locale parameter determines a locale used for formatting numbers and dates. If it is null, the server locale is used.

Note: The formatWithLocale() function returns incorrect value of time zone offset from GMT for Daylight Saving Time ("$tz" or "$Tz" conversions) on Java 6. It is recommended to use formatDate() function for formatting dates. Java 7 returns correct values of time zone offset.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "FormatError" if the specified format is incorrect.

trim(string∗ : String) : String Returns a copy of the specified string, while omitting the leading and trailing whitespaces.

Throws:

- "NullParameterError" if mandatory parameters are 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

4.4.6 XML

parseXml(xml∗ : String, resultType∗ : Type<E>) : E SIDE EFFECT Parses XML document to internal data structures. The xml parameter represents the XML document to parse and the resultType parameter determines the type of the returned data. The referred type must be record type, otherwise an exception is thrown.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "IncompatibleTypeError" - The resultType parameter is not record.
- "UnableToParseXmlError" - The specified xml cannot be parsed.

convertToXml(object∗ : Object) : String Converts internal data structures passed to the object parameter to XML 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.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "IncompatibleTypeError" - The type of the object is not record.
- "UnableToConvertToXmlError" - An error occurred during conversion.

4.4.7 Goal Reflection

activate(goals : Collection<Goal>) : Null (Re)activates inactive or finished goals. If a goal from the collection is active, the goal status remains unchanged.

Throws:

- "NullParameterError" if mandatory parameter is not specified.
- "FinishedProcessInstanceError" if the process instance of any activated goal is finished.

deactivate(goals : Collection<Goal>) : Null Deactivates not finished goals. If a goal from the collection is finished or the goal's process is finished, the goal status remains unchanged.

Throws:

- "NullParameterError" if mandatory parameter is not specified.

isInState(goalPlan* : GoalPlan, state : String) : Boolean Returns true, if the goal or the plan referenced by the goalPlan parameter is in the specified state or any of its sub-states. If the goal or plan parameter is specified correctly, but the state parameter contains an incorrect string representation of a state or is null, the function returns false.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

alive() : String[DEPRECATED] Returns a string representing "Alive" state.

Deprecated. Replaced by constant ALIVE.

notFinished() : String[DEPRECATED] Returns a string representing "Not finished" state.

Deprecated. Replaced by constant NOT_FINISHED.

inactive() : String[DEPRECATED] Returns a string representing "Inactive" state.

Deprecated. Replaced by constant INACTIVE.

active() : String[DEPRECATED] Returns a string representing "Active" state.

Deprecated. Replaced by constant ACTIVE.

ready() : String[DEPRECATED] Returns a string representing "Ready" state.

Deprecated. Replaced by constant READY.

running() : String[DEPRECATED] Returns a string representing "Running" state.

Deprecated. Replaced by constant RUNNING.

finished() : String[DEPRECATED] Returns a string representing "Finished" state.

Deprecated. Replaced by constant FINISHED.

achieved() : String[DEPRECATED] Returns a string representing "Achieved" state.

Deprecated. Replaced by constant ACHIEVED.

failed() : String[DEPRECATED] Returns a string representing "Failed" state.

Deprecated. Replaced by constant FAILED.

deactivated() : String[DEPRECATED] Returns a string representing "Deactivated" state.

Deprecated. Replaced by constant DEACTIVATED.

4.4.8 Number

abs(number* : Decimal) : Decimal Returns an absolute value of the number.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

abs(number* : Integer) : Integer Returns an absolute value of the number.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

min(numbers∗ : Decimal...) : Decimal Returns the minimal number from a list of decimals.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

min(numbers∗ : Integer...) : Integer Returns the minimal number from a list of integers.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

max(numbers∗ : Decimal...) : Decimal Returns the maximal number from a list of decimals.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

max(numbers∗ : Integer...) : Integer Returns the maximal number from a list of integers.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundCeiling(number∗ : Decimal) : Integer Returns rounding of the specified number towards positive infinity.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundDown(number∗ : Decimal) : Integer Returns rounding of the specified number towards zero.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundFloor(number∗ : Decimal) : Integer Returns rounding of the specified number towards negative infinity.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundHalfDown(number∗ : Decimal) : Integer Returns rounding of the specified number towards the "nearest neighbor"; if both neighbors are equidistant, rounds down.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundHalfEven(number∗ : Decimal) : Integer Returns rounding of the specified number towards the "nearest neighbor"; if both neighbors are equidistant, rounds towards the even neighbor.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundHalfUp(number∗ : Decimal) : Integer Returns rounding of the specified number towards the "nearest neighbor"; if both neighbors are equidistant, rounds up.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

roundUp(number∗ : Decimal) : Integer Returns rounding of the specified number away from zero.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

scale(number∗ : Decimal) : Integer Returns the scale of the specified number.

Throws:

- "NullParameterError" if mandatory parameters are 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).

Throws:

- "NullParameterError" if mandatory parameters are 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).

Throws:

- "NullParameterError" if mandatory parameters are not specified.

4.4.9 Model Reflection and Execution

createModelInstance(synchronous∗ : Boolean, model∗ : Model, properties : Map<String, String>) : ModelInstance SIDE EFFEC
Returns a new instance of the model having the specified initialization properties. If the properties is null, an empty map is created in the instantiated model. If the synchronous parameter is set to true, the model interpretation phase is performed synchronously and execution of the containing expression waits until its finish. If the synchronous parameter is set to false, the model interpretation phase is performed asynchronously and execution of the containing expression does not wait until its finish. For usage this function in actions of documents, it is recommended to set the synchronous parameter to true, and for usage in processes it is recommended to set the parameter value to false.

Throws:

- “NullParameterError” - Mandatory parameter is null.
- “ModelInstantiationError” - The model instance did not start due to an internal initiation error.
- “ModelInterpretationError” - The model instance did not start due to an internal model interpretation (startup) error.

findModelInstances(model : String, version : String, isRunning : Boolean, properties : Map<String, String>) : Set<ModelIn
Returns a set of model instances satisfying the specified filtering criteria. If the corresponding fields of a model instance match the patterns specified as function parameters, the model instance is included in the resulting set. All strings in parameters, including the properties parameter, are specified as wildcard strings patterns. A wildcard string can contain the following wildcards: "∗" matching none or several arbitrary characters, and "?" matching exactly one arbitrary character. Any other character in patterns matches itself. Value of the isRunning parameter must match exactly. null specified as the pattern value matches any value of the model instance field (which is equivalent to the "∗" pattern). All the specified properties must occur also in the selected model instances.

findModels(name : String, version : String, latestOnly : Boolean) : Set<Model> Returns a set of all uploaded models satisfying the specified filtering criteria. If the corresponding fields of a model match the patterns specified as function parameters, the model is included in the resulting set. All strings in parameters are specified as wildcard strings patterns. A wildcard string can contain the following wildcards: "∗" matching none or several arbitrary characters, and "?" matching exactly one arbitrary character. Any other character in patterns matches itself. ‘null’ specified as the pattern value matches any value of the model field (which is equivalent to the "∗" pattern). Setting the latestOnly parameter to true causes including only the most

recently uploaded models with given name and version, if there are several models with the same name and version on the server. If the latestOnly parameter is set to false, all models satisfying the filtering criteria are returned; the returned set can contain several models of the same name and version.

findProcessInstances(filter : {ProcessInstance : Boolean}) : Set<ProcessInstance> Returns a set of process instances generated by this model instance up to this point and satisfying the specified filtering criteria. The criteria specified by the filter parameter are given as a Boolean closure with one formal parameter of the type ProcessInstance. 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

findProcessInstances(modelInstance : ModelInstance, filter : {ProcessInstance : Boolean}) : Set<ProcessInstance> Returns a set of process instances generated by the specified model instance up to this point and satisfying the specified filtering criteria. The criteria specified by the filter parameter are given as a Boolean closure with one formal parameter of the type ProcessInstance. 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: Execution of this function can be relatively slow if accessing a model instance which is not currently loaded in memory.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getModel(name∗ : String) : Model Returns a model with the specified name. If there are several models with the same name (and any version), the most recently uploaded model is returned. Unloaded models are ignored.

Mandatory parameters:

-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 there are several models with the same name and version on the server, the most recently uploaded model is returned.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "ModelNotFoundError" if there is no such a model.

getModelInstance(id∗ : Decimal) : ModelInstance Returns a model instance with the specified id.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "ModelInstanceNotFoundError" if there is no such a model instance.

getModelInstanceProperties(modelInstance∗ : ModelInstance) : Map<String, String> Returns a map of initialization properties of the specified model instance.

Throws:

- "NullParameterError" if mandatory parameter is null.

thisModel() : Model Returns the model from which context the function was executed.

thisModelInstance() : ModelInstance Returns the model instance that executed the function. Returns null if executed from document.

thisProcessInstance() : ProcessInstance Returns the process instance that executed the function; or null, if the function is executed in the global context of a module.

sendSignal(synchronous∗ : Boolean, recipients∗ : Set<ModelInstance>, signal∗ : Object) : NullSIDE EFFECT Sends a signal object to the specified recipients. If the synchronous parameter is set to true, the signal is sent synchronously and execution of the containing expression waits until the signal is received by recipients. If the synchronous parameter is set to false, the signal is sent asynchronously and execution of the containing expression does not wait until the signal is received by recipients. The function returns always null.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "SendSignalError" - Sending or receiving of the signal was not successful.

throwEscalation(code∗ : String, payload : Object) : NullSIDE EFFECT Throws an escalation with the specified code and payload. This function should be called only from an expression enclosed in a process that can handle the escalation. Otherwise, calling of this function has no effect. The function returns always null.

Throws:

- "NullParameterError" - Mandatory parameter is null.

4.4.10 Date

+(date : Date, duration : Duration) : Date Binary operator which returns a date obtained by adding the duration to the date.

+(duration : Duration) : Duration Unary operator which returns the specified duration.

+(duration1 : Duration, duration2 : Duration) : Duration Binary operator which returns a duration obtained by adding the values of the corresponding fields of duration1 and duration2.

-(date : Date, duration : Duration) : Date Binary operator which returns a date obtained by subtracting the duration from the date.

-(date1 : Date, date2 : Date) : Duration Binary operator which returns a duration obtained by subtracting date2 from date1. The resulting duration is in the "normal form"; i.e., it has the highest possible values (taken in absolute values) in highly ranked fields. If date2 $>$ date1, the resulting duration is negative, i.e., all its fields are negative.

-(duration : Duration) : Duration Unary operator which returns a duration obtained by negating of each field of the specified duration.

-(duration1 : Duration, duration2 : Duration) : Duration Binary operator which returns a duration obtained by subtracting each field of duration2 from the corresponding field of duration1.

∗(duration : Duration, number : Decimal) : Duration Binary operator which returns a duration obtained by multiplying each field of the duration by the specified number and rounding the value down.

∗(number : Decimal, duration : Duration) : Duration Binary operator which returns a duration obtained by multiplying each field of the duration by the specified number and rounding the value down.

date(year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer) : Date Creates a date from the values given by the following parameters: year, month (month of year), and dayOfMonth. The created date is given as a 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.

Throws:

- "NullParameterError" if mandatory parameters are 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 following parameters: year, month (month of year), dayOfMonth, hour (hour of day), minute (minute of hour), second (second of minute), and millis (millisecond of second). The created 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.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

date(string∗ : String) : Date Creates a date from the specified 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 year (0000-9999), MM stands for month in year (01-12), dd stands for day in month (01-31), DDD stands for day in year (001-365), Www stands for week number prefixed by the letter 'W' (W01-W53), and e stands for day in week (1-7).

The following formats of times are allowed: HH, HH:mm, and HH:mm:ss; where HH stands for hour in day (00-23), mm stands for minute in hour (00-59), and ss stands for second in minute (00-59). The following formats of time zones are allowed: <time>'Z', <time>±HH, <time>±HH:mm, and <time>±HH:mm:ss; where 'Z' is used for UTC. Decimal fractions, separated either by comma or by dot, may also be added to hours, minutes or seconds.

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 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.

Throws:

- "NullParameterError" if mandatory parameters are 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.

formatDate(date∗ : Date, pattern∗ : String, timeZone : String) : String Returns a string created by formatting the date to 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 in the Joda class org.joda.time.format.DateTimeFormat. The syntax is described in the date(string, pattern) function in more details.

Throws:

- "NullParameterError" if mandatory parameter are not specified.
- "FormatError" if specified format or timeZone is incorrect.

getDayOfMonth(date∗ : Date) : Integer Returns the day of the month of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getDayOfWeek(date∗ : Date) : Integer Returns the day of the week of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getDayOfYear(date∗ : Date) : Integer Returns the day of the year of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getHour(date∗ : Date) : Integer Returns the hour of the day of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getMillis(date∗ : Date) : Integer Returns the millisecond of the second of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getMinute(date∗ : Date) : Integer Returns the minute of the hour of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getMonth(date∗ : Date) : Integer Returns the month of the year of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getSecond(date∗ : Date) : Integer Returns the second of a minute of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getWeek(date∗ : Date) : Integer Returns the week of the year of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

getYear(date∗ : Date) : Integer Returns the year of the specified date.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInDays(date1∗ : Date, date2∗ : Date) : Integer Returns the number of days between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInHours(date1∗ : Date, date2∗ : Date) : Integer Returns the number of hours between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInMillis(date1∗ : Date, date2∗ : Date) : Integer Returns the number of milliseconds between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInMinutes(date1∗ : Date, date2∗ : Date) : Integer Returns the number of minutes between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInMonths(date1∗ : Date, date2∗ : Date) : Integer Returns the number of months between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInSeconds(date1∗ : Date, date2∗ : Date) : Integer Returns the number of seconds between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInWeeks(date1∗ : Date, date2∗ : Date) : Integer Returns the number of weeks between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

intervalInYears(date1∗ : Date, date2∗ : Date) : Integer Returns a number of years between two dates, date2 - date1. If date1 $>$ date2, the resulting number is negative.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

setDate(date∗ : Date, year∗ : Integer, month∗ : Integer, dayOfMonth∗ : Integer) : Date Returns a copy of the specified date with the fields given by year, month, and dayOfMonth parameters updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if any of the parameters is out of range (month $<$ 1 or month $>$ 12 or dayOfMonth $<$ 1 or dayOfMonth $>$ number of days in the month and year).

setDayOfMonth(date∗ : Date, dayOfMonth∗ : Integer) : Date Returns a copy of the specified date with the dayOfMonth field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified dayOfMonth is out of range (dayOfMonth $<$ 1 or dayOfMonth $>$ number of days in the date's month and year).

setDayOfWeek(date∗ : Date, dayOfWeek∗ : Integer) : Date Returns a copy of the specified date with the dayOfWeek field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified dayOfWeek is out of range (dayOfWeek $<$ 1 or dayOfWeek $>$ 7).

setDayOfYear(date∗ : Date, dayOfYear∗ : Integer) : Date Returns a copy of the specified date with the dayOfYear field updated.

Throws:

- "NullParameterError" if mandatory parameters are 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 Returns a copy of the specified date with the hour (meaning an hour of a day) field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified hour is out of range (hour $<$ 0 or hour $>$ 23).

setMillis(date$*$: Date, millis$*$: Integer) : Date Returns a copy of the date with the specified millis (meaning a millisecond of a second) field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified smillis is out of range (millis $<$ 0 or millis $>$ 999).

setMinute(date$*$: Date, minute$*$: Integer) : Date Returns a copy of the date with the specified minute (meaning a minute of an hour) field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified minute is out of range (minute $<$ 0 or minute $>$ 59).

setMonth(date$*$: Date, month$*$: Integer) : Date Returns a copy of the specified date with the month field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified month is out of range (month $<$ 1 or month $>$ 12).

setSecond(date$*$: Date, second$*$: Integer) : Date Returns a copy of the date with the specified second (meaning a second of a minute) field updated.

Throws:

- "NullParameterError" if mandatory parameters are 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 Returns a copy of the specified date with the fields given by the hour (meaning an hour of a day), minute (meaning a minute of an hour), second (meaning a second of a minute), and millis (meaning a millisecond of a second) parameters updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if any of the parameters is out of range (hour $<$ 0 or hour $>$ 23 or minute $<$ 0 or minute $>$ 59 or second $<$ 0 or second $>$ 59 or millis $<$ 0 or millis $>$ 999).

setWeek(date$*$: Date, week$*$: Integer) : Date Returns a copy of the specified date with the week (meaning a week of a year) field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "OutOfBoundsError" if the specified week is out of range (week $<$ 1 or week $>$ 52).

setYear(date$*$: Date, year$*$: Integer) : Date Returns a copy of the specified date with the year field updated.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

now() : Date Returns the current date and time.

today() : Date Returns the current date in the time zone of the LSPS Server, without specifying the time.

millis(number∗ : Integer) : Duration Returns a duration with the specified number of milliseconds.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

seconds(number∗ : Integer) : Duration Returns a duration with the specified number of seconds.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

minutes(number∗ : Integer) : Duration Returns a duration with the specified number of minutes.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

hours(number∗ : Integer) : Duration Returns a duration with the specified number of hours.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

days(number∗ : Integer) : Duration Returns a duration with the specified number of days.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

weeks(number∗ : Integer) : Duration Returns a duration with the specified number of weeks.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

months(number∗ : Integer) : Duration Returns a duration with the specified number of months.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

years(number∗ : Integer) : Duration Returns a duration with the specified number of years.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

max(dates∗ : Date...) : Date Returns the maximal date from a list of dates.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

min(dates∗ : Date...) : Date Returns the minimal date from a list of dates.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

4.4.11 Record

deleteRecords(records∗ : Record...) : Null[SIDE EFFECT] Deletes the specified records and all related data links navigable from the deleted records. Works for shared and for non-shared records. Values of deleted records become null in all slots (variables, function or task call parameters, etc.) referring to those records.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "RecordNotFoundError" if the records parameter contains a shared record with incorrect reference to a database record.

deleteRecords(records∗ : Collection<Record>) : Null[SIDE EFFECT] Deletes the specified records and all related data links navigable from the deleted records. Works for shared and for non-shared records. Values of deleted records become null in all slots (variables, function or task call parameters, etc.) referring to those records. If the record is referred to from a collection or a key in a map, it is removed.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "RecordNotFoundError" if the records parameter contains a shared record with incorrect reference to a database record.

createInstance(recordType : Type<Record>, properties : Map<String, Object>) : T Creates and returns new instance of record type T. If the properties are specified, the instance is initialized with the specified values of properties.

Throws:

- "NullParameterError" if mandatory parameter is not specified.
- "IncompatibleTypeError" if the specified properties do not belong to the specified record type.

4.4.12 Property

getPropertyName(property∗ : Property) : String Returns the property name.

Throws:

- "NullParameterError" if mandatory parameter is null.

getPropertyType(property∗ : Property) : Type<Object> Returns the property type.

Throws:

- "NullParameterError" if mandatory parameter is null.

getPropertyRecordType(property∗ : Property) : Type<Record> Returns the property record type. It is the record type where the given property is declared.

Throws:

- "NullParameterError" if mandatory parameter is null.

getPropertyMetadata(property∗ : Property) : Map<String, String> Returns the property metadata.

Throws:

- "NullParameterError" if mandatory parameter is null.

getPropertyValue(record∗ : Record, property∗ : Property) : Object Returns the value of the given property of the given record object.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "IncompatibleTypeError" if the record the the given property are not compatible

setPropertyValue(record∗ : Record, property∗ : Property, value : Object) : Object Sets the value of the given property of the given record object.

Returns null.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "IncompatibleTypeError" if the record and the given property are not compatible

getProperties(type∗ : Type<Record>) : List<Property> Returns all properties of the given record type.

Throws:

- "NullParameterError" if 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.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "IncompatibleTypeError" if the given record and property are not compatible.

4.4.13 Generic

cast(object : Object, objectType∗ : Type<E>) : E If the type of the given object is compatible with the specified objectType, the function returns the object casted to the objectType. If not possible, the function throws a runtime exception.

Throws:

- "NullParameterError" if mandatory parameters are not specified.
- "IncompatibleTypeError" if the type of the object is not compatible with the specified objectType.

isInstance(object : Object, objectType∗ : Type<Object>) : Boolean Returns true, if the given object is assignment-compatible with (is kind of) the specified objectType.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

isSubtype(subtype∗ : Type<Object>, supertype∗ : Type<Object>) : Boolean Returns true, if the type specified by the sub-type parameter is a subtype of the type specified by the supertype parameter.

Throws:

- "NullParameterError" if mandatory parameters are not specified.

typeOf(object : E) : Type<E> Returns the type of specified object.

clone(object : T) : T SIDE EFFECT Creates and returns a shallow copy of a collection or of a user-defined record, or returns the object itself if it is of another type.

error(code∗ : String) : Null Throws an error with the specified code.

Throws:

- "NullParameterError" if mandatory parameter is not specified.

4.4.14 Auditing

getRevisions(record∗ : Record, from : Date, to : Date) : List<Integer> Returns a list of revision identifiers in which the shared record was modified during the period defined by the from and to parameters. If the from or to parameter is null, the period is unbounded from bottom or top respectively.

Throws:

- "NullParameterError" if mandatory parameter is null.
- "IncompatibleTypeError" if the record is not a shared record.

findByRevision(record∗ : Record, revision∗ : Integer) : E Returns the audited record in the given revision.

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, the revision entity is created even if no audited record was changed in the current transaction.

Throws:

- "NullParameterError" if a mandatory parameter is null.
- "IncompatibleTypeError" if the revisionEntitytype is not a custom revision entity record type.

4.4.15 Query

findAll(type∗ : Type<E>) : List<E> Returns all shared records of the given type.

Throws:

- “NullParameterError” - if mandatory parameter is null
- “IncompatibleTypeError” - if the type parameter is not a shared record type

findById(id∗ : Object, type∗ : Type<E>) : E Returns a shared record of the specified type and identified by the given primary key id. Composed primary key id 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 id and values represent the value of the properties. If the key is of type string then it contains the simple property name. If there is no record with the given id, the function returns null.

Throws:

- "NullParameterError" - Mandatory parameter is null.
- "IncompatibleTypeError" - The type parameter is not a shared record type.

4.4.16 Utilities

debugLog(message∗ : { : String}) : String Logs a message to the console at the debug logging level. If logging is not enabled for the debug level, the message closure is not even evaluated. The function returns a string that was logged, or null if the message was not logged.

Throws:

- "NullParameterError" if mandatory parameter is null.

debugLog(message∗ : { : String}, level∗ : Integer) : String Logs a message to the console at the specified logging level. If logging is not enabled at the specified level, the message closure is not even evaluated. The function returns a string that was logged, or null if the message was not logged.

Throws:

- "NullParameterError" if mandatory parameter is null.

log(message∗ : String, level : Integer) : Null Logs a specified message to the application log at the specified log level. Unspecified level value logs at the "Info" level (value 200). The function returns always null.

Throws:

- "NullParameterError" if mandatory parameter is null.

infoLevel() : Integer[DEPRECATED] Returns an integer (200) representing the "Info" log level.

Deprecated. Replaced by constant INFO_LEVEL.

debugLevel() : Integer[DEPRECATED] Returns an integer (100) representing the "Debug" log level.

Deprecated. Replaced by constant DEBUG_LEVEL.

warningLevel() : Integer[DEPRECATED] Returns an integer (300) representing the "Warning" log level.

Deprecated. Replaced by constant WARNING_LEVEL.

errorLevel() : Integer[DEPRECATED] Returns an integer (400) representing the "Error" log level.

Deprecated. Replaced by constant ERROR_LEVEL.

getApplicationData(key∗ : String) : Object Returns an application data object with the specified key. At the moment, the "locale" string and the "user" Person are supported by the LSPS Process Application.

Throws:

- "NullParameterError" if mandatory parameter is null.

splitPathname(name∗ : String) : List<String> Splits a fully specified element name to names of its components which are separated by the namespace separator (::). For instance, splitPathname("module::'role 1'") results in ["module", "role 1"].

Throws:

- "NullParameterError" if mandatory parameter is null.
- "IncorrectPathnameError" if syntax of the specified name is incorrect.

4.5 Tasks

4.5.1 Core

CreateModelInstance Creates a new model instance.

model : Model Model to be instantiated.

properties : Map<String, String> Map of model instance initialization properties. If null, an empty 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, the referred slot is set to null.

Throws:

-"NullParameterError" if mandatory parameters are 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.

modelInstance : ModelInstance Model instance to be terminated.

Throws:

-"NullParameterError" if mandatory parameter is not specified.

DeleteModelInstanceLogs Permanently removes process logs of the specified finished model instances from the database.

modelInstances : Set<ModelInstance> A set of model instances of which 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. If the goal is active, the task type does nothing.

goal : Goal Goal to be (re)activated.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

-"FinishedProcessInstanceError" if the process instance of the activated goal is finished.

Deactivate Deactivates a not finished goal. If the goal is finished or the goal's process is finished, the task type does nothing.

goal : Goal Goal to be deactivated.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

RepeatGoals Repeats a set of goals. In principle, this task type deactivates the specified achieve goals, then sets the slots with the desired values, and finally reactivates the goals.

goals : List<RepeatedGoal> Goals to be repeated. A runtime exception is thrown when the list includes a pair of achieve goals from which one goal is a subgoal of the other. Another runtime exception is thrown when process instance of any of the repeated goals is terminated.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

-"FinishedProcessInstanceError" if the process instance of the activated goal is finished.

-"ParentGoalError" if the goals list includes a pair of goals from which one goal is a subgoal of the other.

DeleteSharedRecords Deletes the specified shared records and related (by data relationships) shared records from database.

data : Set<Object> Set of shared records to delete.

Throws:

-"NullParameterError" if mandatory parameters are 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 incorrect reference to a database record.

RemoveBinaryData[DEPRECATED] Permanently removes the specified binary data from the database.

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 mandatory parameters are not specified.

Lock Creates a model instance-independent lock.

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 an already existing lock with the same name.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

Unlock Unlocks (removes) a previously created lock.

lockName : String Name of the lock to be removed.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

ParseXml[DEPRECATED] Parses XML document to internal data structures.

xml : String The XML document to parse.

output : Reference<Object> Reference to a slot where the result is stored. The type of reference provided determines which object should be parsed from XML. The referred object must be record type, otherwise an exception is thrown.

useDefaultMapping : Boolean If true, the default LSPS XML mapping is used. Otherwise, the meta- data defined on record types are taken into account when parsing XML. Parsing of types generated from XSD requires setting of this parameter 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 mandatory parameters are 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 internal data structures to XML document.

object : Object The object to be converted. It must be an instance of a record type, otherwise an exception is thrown.

output : Reference<String> Reference to the variable where the output XML document is stored.

useDefaultMapping : Boolean If true, the default LSPS XML mapping is used. Otherwise, the meta- data defined on record types are taken into account when creating XML. Creating XML from an object of which type was generated from XSD requires setting of this parameter to false. The default val- ue 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 mandatory parameters are not specified.

-"IncompatibleTypeError" if the type of the object is not record.

-"UnableToConvertToXmlError" if an error occured during conversion.

HttpCall Request payload. Can be either String or Binary. All other objects are converted to String. String is converted to data stream honoring charset specified in the requestContentType parameter. If there is no charset specified, UTF-8 is used.

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 honoring charset specified in the requestContent↩ Type parameter. If there is no charset 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 where the response payload is to be stored.

responseCode : Reference<Integer> Reference to the response code.

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.

message : String Message to be logged.

level : Integer Log level. Unspecified level value logs at the “Info” level.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

Assign[DEPRECATED] Writes values to slots.

slots : Map<Reference<Object>, Object> Sets objects as values of slots, whereas the slots are being pointed to by reference. References to the changed slots are given by the keys and corresponding objects by the values. References in the slots map cannot be null.

Throws:

-"NullParameterError" if mandatory parameters are not specified.

-"IncompatibleTypeError" if the type of an assigning value is not compatible with the type of the slot it is being assigned to.

Execute Executes the specified activity.

activity : Activity Activity to execute.

Throws:

-"NullParameterError" if mandatory parameter is not specified.