Parameter Implementations

Yellowfin has a number of Parameter implementations which can be used out of the box. They are defined in the enum **InputType**. Below is a list of all the the available types, and their options, along with code samples.

Type	Description	View Options	Code Sample
TEXTBOX	A text input box. blocked URL	• **textClassName:** Specify a class to add to the HTML input element. • **style:** CSS styles added to the HTML input element. • **numericOnly:** Set to true if the text box should accept only numbers. • **numericType:** Set to "INTEGER" to disallow decimal numbers. numericOnly should be true. • **unsignedOnly:** Set to true to not allow negative values. Works only when numericOnly is true. • **thousandSeparatorOn:** Set to true to show the thousandths separator when numericOnly is true. • **decimalPlaces:** Number of decimal places when numericOnly is true and numericType is not INTEGER. • **minValue/maxValue:** Min and Max number which can be entered. • **maxStringLength:** Maximum length of the input when numericOnly is false. • **readonly:** Sets the "readOnly" property of the HTML input element. • **disabled:** True sets the "disabled" property of the HTML input element. • **name:** Sets the "name" property of the HTML input element. • **tooltip:** Sets the "title" property of the HTML input element. • **disableBrowserAutoComplete:** Disable browser's autocomplete for that field. • **placeholder:** placeholder text to show before a value has been entered. • **placeholderCss:** Style the placeholder. • **placeholderAccountForScroll:** Set to true to subtract the scroll position from the placeholder. • **suffix:** a display suffix which should be shown appended to the end of the actual value. • **width:** override the width value of this input with a custom width. Note: This is a css property, so you can define whatever display unit you like. No unit defaults to px.	`ParameterImpl parameter = new ParameterImpl();` `parameter.setProperty ("DECIMALS");` `parameter.setInputType(InputType. TEXTBOX);` `parameter.setName("Decimal Places");` `parameter.setMinAllowed(0);` `parameter.addViewOption ("numericOnly", true);` `parameter.addViewOption ("numericType", "INTEGER");`
SELECT	A drop down list. The value and description are taken from the Possible Values added to this parameter. blocked URL	• **width:** override the width value of this input with a custom width. Note: This is a css property, so you can describe in whatever display unit you like. No unit defaults to px. • **style:** CSS styles added to the HTML input element • **size:** Sets the size attribute of the dropdown list. The default is 1. Setting it to a higher number makes it a scrollable list. • **multiSelect:** Set to true to make this a scrollable multiselect list. • **selectAllAvailable:** Add controls for selecting and deselecting all options. multiSelect should be true to use this • **multiJoinString:** If this is "|", all the selected values in a multi-select will be joined with "|" between them into a single string. multiSelect should be true to use this • **disabled:** Set to true to disable input • **showOptionTitle:** Show tooltips for options when set to true.	`ParameterImpl p = new ParameterImpl();` `p.setProperty("DATE_FORMAT");` `p.setName("Format");` `p.setInputType(InputType.SELECT);` `List<CustomValue<?>> optValues = new ArrayList<>();` `optValues.add(new CustomValue ("","- - Select - - ");` `optValues.add(new CustomValue ("YEAR","Year"));` `optValues.add(new CustomValue ("YEAR_START","Year Start Date"));` `optValues.add(new CustomValue ("YEAR_END","Year End Date"));` `p.setPossibleValues(optValues);`

RADIO	A group of Radio Buttons. Has a radio button for each option. blocked URL	• **alignment:** When 'right', render the radio buttons to the right of their labels. The default is 'left.' • **fullDescription:** Set this to true to display descriptions for each option. • **descriptions:** This should be an object of the format {value: 'description' }. For e.g.: `{ "LEFT": "Align Left", "RIGHT": "Align Right", "MIDDLE": "Centre Align" }` This works only when fullDescription is true. • **textAlign:** Set the text-align CSS property. This is applicable only when the alignment property is set. • **valign:** specifies the valign CSS property for this element. • **radioColour:** Set this to 'red' to render a red radio button. This does not affect the label. • **disabled:** Set to true to disable input • **tooltip:** Show tooltips for options when set to true. • **maxRows:** Available options will be rendered in this many rows.	`ParameterImpl parameter = new ParameterImpl(); parameter.setProperty ("STARTING_POINT"); parameter.setInputType(InputType. RADIO); parameter.addPossibleValue ("LEFT", "Left"); parameter.addPossibleValue ("RIGHT", "Right"); parameter.addPossibleValue ("MIDDLE", "Middle");`
DATE, TIMESTAMP	Date/Timestamp input element. These are mostly similar and have the same view options. The key set using **setProperty** stores the input value. **Format:** • Static dates are stored as yyyy-MM-dd • Static timestamps are stored as yyyy-MM-dd hh:mm:ss **Dynamic Dates:** In case of Dynamic Dates, the unit is stored against the key specified in the view option "unitCodeProperty". Dynamic date values are stored as "SYSDATE+x". For example, if property=STARTDATE and unitCodeProperty=STARTUNIT Today - 1 Year is stored as: STARTDATE: SYSDATE-1 STARTUNIT: YEAR blocked URL	• **useCurrentDate:** Set this to true to use Dynamic Dates. It renders a toggle, which can be switched off to use a specific date /timestamp. • **unitCodeProperty:** In case of Dynamic Dates, this is used to identify the selected date unit. • **width:** override the default width of the textbox. • **disabled:** Set to true to disable input.	`ParameterImpl p = new ParameterImpl(); p.setProperty("STARTDATE"); p.addViewOption ("useCurrentDate", "true"); p.addViewOption ("unitCodeProperty", "STARTUNIT"); p.setInputType(InputType.DATE);`
TIME	Text input element which verifies time data input. Valid time input is saved against the specified property in hh:mm:ss format (24 hour). This widget does not have any styling options. blocked URL	• **showSeconds:** Set to true to show a textbox for seconds entry. This is false by default. • **disabled:** Set to true to disable.	`ParameterImpl p = new ParameterImpl(); p.setProperty("RUN_TIME"); p.setInputType(InputType.TIME);`
CHECKBOX	Displays one or more standard Yellowfin checkboxes. blocked URL	• **alignment:** When 'right', render the checkboxes to the right of their labels. The default is 'left'. • **checkboxColour:** Set this to 'red' to render red checkboxes. This does not affect the label. • **tooltip:** Show tooltips for options when set to true.	`ParameterImpl p = new ParameterImpl(); p.setInputType(InputType. CHECKBOX); p.setProperty("INCLUDED_FIELDS"); p.setParameterClassName ("included-fields"); p.addViewOption("tooltip", true);` `p.addPossibleValue("field001", "PERSONID"); p.addPossibleValue("field002", "CAMPID"); p.addPossibleValue("field003", "INVOICEDAMOUNT");`

BUTTON	Renders a standard Yellowfin button. blocked URL	• **text:** the text label to display on the button • **img:** the image to display on the button. Note that if both the 'img' and 'text' options are specified, only the image will be displayed, and the 'text' option's value will be used as a tooltip for the button. The image may be a URL or a Base64 string with the image type: data:<MIME Type>;base64, <base64 String> • **activeImg:** the image to display on the button when hovering over it. • **imgCss:** styles to apply to the image on the button (active or not). • **style:** colour of the button, which may be one of 'red', 'blue', 'redhover', 'grey', 'medgrey', 'darkgrey'. • **flat:** true renders the button without 3D effects. • **tooltip:** HTML tooltip on hovering over the button. • **css:** styles to apply to the button. • **cssClass:** this class will be added to the button. • **width:** width of the button. • **height:** height of the button.	`ParameterImpl p = new ParameterImpl();` `p.setProperty("apply");` `p.setInputType(InputType.BUTTON);` `p.addViewOption("text", "Apply");` `p.addViewOption("flat", true);` `p.addViewOption("style", "blue");` `p.addViewOption("width", "50px");` `p.addViewOption("height", "40px");`
STATIC	Renders a block of static text. blocked URL	• **text:** Content to be displayed	`ParameterImpl p = new ParameterImpl();` `p.addViewOption("text", "This is some text");` `p.setInputType(InputType.STATIC);`
COLOUR PICKER	Standard Yellowfin colour-picker. blocked URL	None.	`ParameterImpl p = new ParameterImpl();` `p.setInputType( InputType.COLOURPICKER);`
SLIDERANDTEXT	Slider input with a linked text box which allows input via either method. blocked URL	• **maxValue:** Maximum value of the slider which is saved against the Parameter's property. Defaults to 100. • **textMaxValue:** Max value which can be entered in the textbox. • **minValue:** Minimum value of the slider which is saved against the Parameter's property. Defaults to 0. • **textMinValue:** Min value which can be entered in the textbox. • **sliderWidth:** override the default width of the slider element.	`ParameterImpl p = new ParameterImpl();` `p.setName("Group Size");` `p.setProperty("size");` `p.setInputType(InputType. SLIDERANDTEXT);` `p.addViewOption("textMaxValue", "20");` `p.addViewOption("maxValue", "20");` `p.setDefaultValue(5);`
TOGGLE	Renders an On/Off toggle slider. blocked URL	• **onValue:** the value to save when the toggle slider is switched to the On position. Defaults to true. • **offValue:** the value to save when the toggle slider is switched to the Off position. Defaults to false. • **disabled:** Set to true to disable.	`ParameterImpl p = new ParameterImpl();` `p.setProperty ("FIRST_ROW_HEADER");` `p.setName("First row contains unique titles");` `p.setDefaultValue(true);` `p.setInputType(InputType.TOGGLE);`

FIELDMATCHDROPDOWN	Accepts two lists of FieldObject (see the section on Helper Objects) which can be matched to one another. There may be one or more matches. The resulting relationships are returned in an array of objects which look like: `{ 'from': fieldA.id, 'to': fieldB.id }` blocked URL	• **fieldListA:** The first list of fields. • **fieldListB:** The second list of fields • **titleA:** The title for field list A. • **titleB:** The title for field list B. • **titleClassA:** Optional CSS class to format the title for field list A. • **titleClassB:** Same as above, but applies to title of field list B. • **viewOptionsA:** Options passed to drop down list A. • **viewOptionsB:** Options passed to drop down list B.	`ParameterImpl p = new ParameterImpl();` `p.setProperty ("fieldMatchProperty");` `p.setList(true); //List of matches` `ListOptions lo = new ListOptions();` `lo.setAddButtonText("Match More");` `p.setListOptions(lo);` `FieldObjectDataType textType = FieldObjectDataType.TEXT;` `FieldObjectDataType numericType = FieldObjectDataType.NUMERIC;` `// LIST A` `List<FieldObject> fieldListA = new LinkedList<>();` `FieldObject field1 = new FieldObject("1", "", "Field 1", textType);` `FieldObject field2 = new FieldObject("2", "", "Field 2", numericType);` `fieldListA.add(field1);` `fieldListA.add(field2);` `// LIST B` `List<FieldObject> fieldListB = new LinkedList<>();` `FieldObject field3 = new FieldObject("3", "", "Field 3", textType);` `FieldObject field4 = new FieldObject("4", "", "Field 4", numericType);` `FieldObject field5 = new FieldObject("5", "", "Field 5", textType);` `FieldObject field6 = new FieldObject("6", "", "Field 6", numericType);` `fieldListB.add(field3);` `fieldListB.add(field4);` `fieldListB.add(field5);` `fieldListB.add(field6);` `// Add the lists to the Match widget` `p.addViewOption("fieldListA", fieldListA);` `p.addViewOption("fieldListB", fieldListB);` `p.addViewOption("titleA", "Left Field");` `p.addViewOption("titleB", "Right Field");` `p.setInputType(InputType.FIELDMATCHDROPDOWN);`
FIELDMATCH	This serves the same function as FIELDMATCHDROPDOWN. Fields are matched by dragging them around. They are matched by data type, which are colour coded. Fields which cannot be matched are flagged.		`ParameterImpl p = new ParameterImpl();`

Fields from one side may also be excluded by dragging them to the “exclude” zone. The resulting relationships are returned in a similar format as FIELDMATCHDROPDOWN.

When a field is excluded from one column, the match object for that row will not have the corresponding “from” or “to” attribute. For example, the object for ADDRESSID in the below screenshot would be

```
        {
  'to': ADDRESSID.id
}
```

blocked URL

- **fieldListA:** The first list of fields.
- **fieldListB:** The second list of fields
- **titleA:** The title for field list A.
- **titleB:** The title for field list B.
- **titleClassA:** Optional CSS class to format the title for field list A.
- **titleClassB:** Same as above, but applies to title of field list B.
- **viewOptionsA:** Options passed to drop down list A.
- **viewOptionsB:** Options passed to drop down list B.
- **includeUnmatched:** Allow fields from one list to be matched to nothing in the other list and show no error. False by default.
- **roughMatch:** Allow Dates and Text fields to be matched. False by default.
- **imageMap:** Specify various images in a map. Available keys are:
- **primary:** For fields which form a Primary Key
- **dragHandle:** The handle for dragging fields around
- **error:** Field match error image
- **forceAllConnections:** If set to true when connections are being returned, if there are warnings an empty list will bereturned instead of a partially connected list. False by default.

```
p.setInputType(InputType.
FIELDMATCH);
p.setProperty
("UNION_FIELD_MATCH");
p.addViewOption
("includeUnmatched", true);

//Match exact types
p.addViewOption("roughMatch",
false);

List<FieldObject> foListA = new
ArrayList<>();
List<FieldObject> foListB = new
ArrayList<>();

FieldObjectDataType textDataType
= FieldObjectDataType.TEXT;
FieldObjectDataType
numericDataType =
FieldObjectDataType.NUMERIC;

// LIST A
FieldObject field1 = new
FieldObject("1", "primary",
"Field 1", textDataType);
//Primary Key
FieldObject field2 = new
FieldObject("2", "", "Field 2",
numericDataType);//This is a
metric

foListA.add(field1);
foListA.add(field2);

// LIST A
FieldObject field3 = new
FieldObject("3", "", "Field 3",
textDataType);
FieldObject field4 = new
FieldObject("4", "", "Field 4",
numericDataType);//This is a
metric
FieldObject field5 = new
FieldObject("5", "", "Field 5",
textDataType);
FieldObject field6 = new
FieldObject("6", "", "Field 6",
numericDataType);//This is a
metric

foListB.add(field3);
foListB.add(field4);
foListB.add(field5);
foListB.add(field6);

p.addViewOption("fieldListA",
foListA);
p.addViewOption("fieldListB",
foListB);

//Input 1
p.addViewOption("titleA", "Input
1");

//Input 2
p.addViewOption("titleB", "Input
2");
```

			The above code will generate the following: blocked URL
AUTOCOMPLETE TEXTBOX	This renders a searchable dropdown list. blocked URL Searchable items should be CustomValue objects added as “possibleValues” of the Parameter.	• **initialValue:** the input's initial value. This should be a CustomValue object where value = search string description = search string • **width:** Sets the width of the view • **maxDisplayItems:** The max number of items displayed below the list, default is 4 • **isDataGrouped:** Data can have a group • **resultListStyle:** Adds the specified class(es) to HTML element containing the results list	``` ParameterImpl p = new ParameterImpl(); p.setProperty("TABLETEXTBOX"); p.setName("New Table Name"); p.setInputType(InputType.AUTOCOMPLETETEXTBOX); p.addViewOption("width", 200); CustomValue<String> initialValue = new CustomValue<String>("all_objects", "all_objects"); p.addViewOption("initialValue", initialValue); List<CustomValue<?>> possibleValues = new ArrayList<>(); possibleValues.add(new CustomValue<String>("all_columns", "all_columns")); possibleValues.add(new CustomValue<String>("all_parameters", "all_parameters")); possibleValues.add(new CustomValue<String>("all_sql_modules", "all_sql_modules")); possibleValues.add(new CustomValue<String>("all_tables", "all_tables")); p.setPossibleValues(possibleValues); p.addViewOption("width", 200); p.addViewOption("resultListStyle", "outputStepAutoCompleteList"); ```

SHOWADVANCED	An accordion-like widget to show/hide other UI elements. This should be used with ParameterDisplayRules. blocked URL	• **showText:** Text for the link which opens the accordion • **hideText:** Text for the link which hides the accordion	Boolean value = true; Boolean negative = false; // SHOW_ADVANCED_SETTINGS == true ParameterDisplayRule advancedSettingsDisplayRule = new ParameterDisplayRule("AND", "SHOW_ADVANCED_SETTINGS", value, negative); ParameterImpl p = new ParameterImpl(); p.setProperty("ALL_ROWS"); p.setName("All Rows"); p.setDefaultValue(true); p.setInputType(InputType.TOGGLE); p.addDisplayRule (advancedSettingsDisplayRule); paramList.add(p); p = new ParameterImpl(); p.setProperty("SEPARATOR"); p.setName("Separator"); p.setInputType(InputType. TEXTBOX); p.addViewOption("width", "40px"); p.addDisplayRule (advancedSettingsDisplayRule); paramList.add(p); p = new ParameterImpl(); p.setProperty ("SHOW_ADVANCED_SETTINGS"); p.setInputType(InputType. SHOWADVANCED); p.setDefaultValue(false); p.addViewOption("showText", "Advanced Settings ↓"); p.addViewOption("hideText", "Less ↑"); p.addViewOption("width", "150px"); paramList.add(p);

PAIRCOLUMNS	Renders paired data in two columns. There is a control for deleting rows. Whenever the Parameter data changes, the columns are re-rendered to show the new data. The data should be an array having objects of the format: `{ column1Key: dataRow1Col1 column2Key: dataRow2Col2 }` blocked URL	• **column1Header:** Heading of the first column • **column2Header:** Heading of the second column • **column1Key:** Key to identify the first column's data • **column2Key:** Key to identify the second column's data	`ParameterImpl p = new ParameterImpl(); p.setProperty("RULES_PARAM"); p.setName("Rules"); p.setInputType(InputType.PAIRCOLUMNS); p.addViewOption("column1Header", "Find"); p.addViewOption("column2Header", "Replace"); p.addViewOption("column1Key", "FIND_PARAMETER"); p.addViewOption("column2Key", "REPLACE_PARAMETER");`  `List<Map<String, String>> defaults = new ArrayList<>();`  `Map<String, String> row = new HashMap<>(); row.put("FIND_PARAMETER", "Farenheit"); row.put("REPLACE_PARAMETER", "Celsius"); defaults.add(row);`  `row = new HashMap<>(); row.put("FIND_PARAMETER", "miles"); row.put("REPLACE_PARAMETER", "km"); defaults.add(row);`  `p.setDefaultValue(defaults);`
DELIMITEREXAMPLE	Shows example output on applying a delimiter to split a row of data. blocked URL	• **text:** Sample row having delimited data	`ParameterImpl p = new ParameterImpl(); p.setProperty("DELIMITER"); p.addViewOption("text", "7 Rosella Avenue;Boronia;3155; Victoria"); p.setInputType(InputType.DELIMITEREXAMPLE);`

INPUTLIST		• **inputOptions:** a list of options that allows InputListView to render a collection of views automatically. Each input option will have: ◦ **controlType:** the type of control of that view (TEXTBOX, TOGGLE, CHECKBOX, SELECT,...) ◦ **viewOptions:** view options of that particular control	`Map<String, String> viewOptions1 = new HashMap<>(); viewOptions1.put("property", "FIND"); Map<String, String> viewOptions2 = new HashMap<>(); viewOptions2.put("property", "REPLACE");` `Map<String, Map<String, String>> inputOption1 = new HashMap<>(); inputOption1.put("controlType", "TEXTBOX"); inputOption1.put("viewOptions", viewOptions1);` `Map<String, Map<String, String>> inputOption2 = new HashMap<>(); inputOption2.put("controlType", "TEXTBOX"); inputOption2.put("viewOptions", viewOptions2);` `List<Map<String, Map<String, String>> inputOptions = new ArrayList<>();` `ParameterImpl p = new ParameterImpl(); p.setInputType(InputType.INPUTLIST); p.setProperty ("FIND_AND_REPLACE_INPUTS"); p.addViewOption("inputOptions", inputOptions);`

Special Implementations

Besides the commonly defined parameters, there are also a few special implementations. These are listed below:

Implementation	Description	Methods	View Options	Code Sample

FileUploadParameter	This is used to generate UI for uploading a file. It also has a way to delete an uploaded file. blocked URL	1. public FileUploadParameter (String property, String fileType, String typeCode, String uploadEvent) **property** is the name associated with a numeric ID of the uploaded file saved in the Yellowfin document store. The property and file ID could be saved as Transformation step configuration. **fileType** defines the extension of file to be uploaded (csv, xml etc). If any file can be uploaded, use * **typeCode** defines the file type code used for categorising the files in the Yellowfin document store **uploadEvent** is the event which will be triggered by the panel collection when the upload finishes. This can be null. 2. public void setUploadWidgetText(String text) Text that will be displayed on the file upload widget. 3. public void setUploadImageBase64 (String base64, String imgType) Use this to specify a Base64 image which will be displayed on the upload widget. Its type must also be specified. 4. public void setUploadImageHoverBase64(String base64, String imgType) Use this to specify how the widget should look on hover.	• **inputName:** Name of the file input HTML element. • **showProcessingBar:** Show a bar while a file is being saved to the Yellowfin config DB. This is true by default. blocked URL • **processingText:** Text displayed above the processing bar. This can be used when showProcessingBar is true. • **width:** Set the width of the upload widget. It defaults to 100%. • **height:** Set the height of the upload widget. It defaults to 100%. • **styleClass:** Add a CSS class name to the upload widget.	``` FileUploadParameter p = new FileUploadParameter ("UPLOAD_FILE_KEY", "*", "FILE", null); p.setParameterClassName("left-param"); p.setUploadImageBase64 (getBase64OffImage(), "image/svg+xml"); p.setUploadImageHoverBase64 (getBase64OnImage(), "image/svg+xml"); p.setUploadWidgetText ("Drag File Here"); ``` The uploaded file can be accessed from a data transformation step using: ``` byte[] rawData = this.getFile ("UPLOAD_FILE_KEY"); ```

PasswordParameter	This is used to generate UI for a password field. Things typed in the field are masked and they can be encrypted/decrypted by the framework. When used in Data Transformations, Yellowfin automatically encrypts the value when the current value is different from what was previously saved as a Step Option. The component using the parameter is expected to decrypt the password before using it. blocked URL	1. public PasswordParameter (String property, String value) **property** is the name associated which the password which will be saved in the Yellowfin configuration DB. **value** is the string to be encrypted or decrypted. 2. public String getValue 3. public void setValue(String value) Get/set the current value of the Password Parameter. 4. public String encrypt() throws Exception Encrypt the value set in the password parameter. 5. public String decrypt() throws Exception Decrypt the value set in the password parameter.	• **textClassName:** Specify a class to add to the HTML input element • **style:** CSS styles added to the HTML input element • **disabled:** True sets the "disabled" property of the HTML input element • **placeholder:** placeholder text to show before a value has been entered. • **width:** override the width value of this input with a custom width. Note: This is a css property, so you can describe in whatever display unit you like. No unit defaults to px.	Defining a password parameter: `PasswordParameter apiKey = new PasswordParameter ("API_KEY", "myApiKey"); apiKey.setName ("API Key"); apiKey. addViewOption ("width", "190px");` Using a password parameter: `List<Parameter> paramsList = getPanelCollection( ).getSectionMap(). get ("passwordSection") .getParameters();` `for (Parameter parameter : paramsList){ if ("API_KEY". equals(parameter. getProperty())){` `PasswordParameter pwd = ((PasswordParameter ) parameter; pwd. setValue (getStepOption ("API_KEY")); try { apiKey = pwd.decrypt();` `} catch (Exception e) {` `throwUnhandledETLEx ception(e); } } }`