Parameter Level

This is the fourth and lowest level, and is described by the Parameter interface. User input widgets are rendered by implementation classes. Yellowfin has a number of implementations, most of which can be used by instantiating the **ParameterImpl** class and setting the correct input type. Yellowfin uses this information to generate html using its view engine.

The important methods of the ParameterImpl class are listed below:

Methods	Description
public void setInputType (InputType type)	This is used to set the parameter type to one defined in enum InputType. For example, InputType.SELECT renders a dropdown.
public void setName (String name)	The name of the Parameter gets rendered as a label to the left of the input element.
public void setDescription(String desc)	The description gets rendered below the label.
public void setProperty (String property)	User input is set in this property of a JSON object and sent back to the server when submitted.
public void setMinAllowed(int min) public void setMaxAllowed(int max)	Shows an error message if the input is outside of the min or max bounds. This is applicable for numeric input values.
public void addViewOption(String property, Object value) public void setViewOptions (Map<String, Object> viewOptions)	The rendered input widget can be customized by setting up options using these methods. The available views and their options are listed in the section on Input Types.
public void addDisplayRule (ParameterDisplayRule rule) public void setDisplayRules (List<ParameterDisplayRule> displayRules)	Display Rules are used to determine whether a panel should be displayed or not, based on user input. There's more on this in the section on Helper Objects.
public void addPossibleValue (String val, String desc) public void addPossibleValue (String val, String desc, String colour) public void setPossibleValues (List<CustomValue<?>> possibleValues)	These methods are used to set up the available option values for an input type such as Dropdown List. addPossibleValue is also overloaded to accept an integer value. The value, description and colour are used to build a CustomValue object which is then used to render an option. The string in the value field will be saved to the database as the selected option. The description is for display purposes only.
public void setParameterClassName(String className)	Set a CSS class to be applied to the parameter.
public void setCssRules (Set<CssRule> cssRules)	Define CSS rules which can be used for fine-grained control over styling of a parameter. This is described in detail in the section on Helper Objects.
public void addParameterValueDisplay(String key, ParameterValueDisplay pvd)	The ParameterValueDisplay object is used to define how a parameter is rendered for a specific input value. The input value is specified in the "key" method parameter. For example, in case of a toggle switch, this could be used to change the image and colour of the switch when the current selection is "false". ParameterValueDisplay is discussed in detail in the section on Helper Objects.

public void setList (boolean) public void setListOptions (ListOptions listOptions)	setList is used to determine whether or not to generate a list of parameters of a specified type. For example, when set to true for a TEXTBOX type, it will generate a list of textboxes that can be accumulated to indefinitely using an “add” button. Options for rendering the list are defined using a ListOptions object. This is described in detail in the section on Helper Objects.

public void setValidationRules (ParameterValidation rules)
 	
public void setObjectValidationRules(Map<String, ParameterValidation> rules)
	These methods are used to set up front-end validation rules. You can use this to ensure that an input field is not empty, for instance. The Object validation method is used to validate JSON objects. For example, a field matching parameter may hold the mapping between two fields as a JSON object.
	
	`{`
	`  "from": "field1",`
	`  "to": "field2"`
	`}`
	
	The keys in the map passed to setObjectValidationRules, should be the object properties “from” and “to”. The values should be the ParameterValidation objects for validating “field1” and “field2” respectively. For more details, see the section on Helper Objects.

public void setValueDependencies (List<ValueDependent> deps) public void addValueDependency (String prop, PropertyLocation loc) public void addValueDependency (ValueDependent dep)	This is used if the Parameter being set up is dependent on another. When the value of one parameter changes, dependent parameters get re-rendered. For more information, see the section on Helper Objects.
public void setEvent (String event) public void setEventParameters (List<ValueDependent> eventParameters) public void addEventParameter (ValueDependent parameter) public void addEventParameter (String prop, PropertyLocation loc) public void setEventData (Map<String, Object> eventData) public void addEventData(String key, Object value)	Events are similar to Value Dependencies, except in reverse. This is used to trigger an event when the parameter’s value changes. This may be used to change the values of “child” parameters. Event Data is used to send any associated data along with the triggered events. For more information, see the section on Helper Objects.
public String getDynamicKey() public void setDynamicKey(String dynamicKey)	These methods are used for working with Dynamic Parameters, which is discussed in the section on Helper Objects.