Tickets

Wiki

Table of contents

I. Types and operators

Types

Four types of data can be manipulated inside the predicates and the actions:

bool — Boolean (*true* or *false*)
They are used by writing *true* or *false*, with capital letters or not.

int — Integers (…, -2, -1, 0, 1, 2, …)
They are used by simply writing numbers.

float — Floating point numbers (2.54, 1.12, 5.0, …)
They are used by writing decimal numbers with a dot.

string — String of characters (“hello”, ‘bonjour’)
They are used by enclosing the words with a pair of quotes or a pair of apostrophe.

It is also possible to use lists of these types. A list is a collection of elements of the same type, written between a pair of brackets, and separated by commas.

List<bool> — [true, false, true]

List<int> — [1, 2, 3]

List<float> — [1.45, 2.89, 3.14]

List<string> — [“hello”, ‘bonjour’]

Lists of lists are possible too. Remember a list must have its element of the same type.

List<List<int>> — [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

A list can also not have any elements:

List<?> — []

Type conversions

Implicit conversion between types exists, so that an argument can be given to a function or to an operator even if it is not of the type required.

bool, int and float are equivalent and can be converted to each other type:

bool	*true*	->	1 or 1.0	
	false	->	0 or 0.0	
int	-1	->	*true* or -1.0	
	0	->	*false* or 0.0	(*false* when 0)
	1	->	*true* or 1.0	
	4	->	*true* or 4.0	
float	-1.4	->	*true* or -1	
	0.0	->	*false* or 0	(*false* when 0.0 only)
	0.8	->	*true* or 0	
	4.89	->	*true* or 4	

bool, int and float can be converted into string, but string can't be converted into these types.

bool	*true*	->	"true"
	false	->	"false"
int	-1	->	"-1"
	0	->	"0"
	1	->	"1"
	4	->	"4"
float	-1.4	->	"-1.4"
	0.0	->	"0"
	0.8	->	"0.8"
	4.89	->	"4.89"

Type conversions are also used for lists to determine which is the type of the elements in the list. The type chosen is the most generic type among bool, int, float and string:

bool < int < float < string where < means "less generic than".

[true, 0, 5]	->	List<int>
[true, 0.5]	->	List<float>
[1, 1.2, "hello"]	->	List<string>
[[false, 0], [true, "hello"]]	->	List<List<string>>

Operators

Operators take two operands (left and right) and convert them into the most generic type between them. Comparison and logical operators return a bool, and math operators return the most generic type found. Here are all the operators, and the types they can accept:

&&	and	bool	(logical)
||	or	bool	(logical)
!	not	bool	(logical)
==	equal	bool, int, float, string	(comparison)
!=	not equal	bool, int, float, string	(comparison)
>	more than	float	(comparison)
>=	more or equal	float	(comparison)
<	less than	float	(comparison)
<=	less or equal	float	(comparison)
*	multiplication	float	(math)
/	division	float	(math)
-	minus	float	(math)
+	addition	float, string (concatenation)	(math)

true + 0.54	->	1.0 + 0.54	->	1.54
"hello" + *false*	->	"hello" + "false"	->	"hellofalse"
1.54 > *true*	->	1.54 > 1.0	->	*true*
"hello" == -2	->	"hello" == "-2"	->	*false*
"hello" && 1.54	->	error, "hello" can't be converted into a bool		

II. Ticket

Description

A ticket is a web form whose content can be stored in a dashboard system. It is made of components which are various fields that can have dynamic behavior with each other.

Options

Mandatory options:

Name Defines the name of the ticket. It must be unique among all the other tickets.

DashboardIntegration Defines which dashboard system the ticket is exported in. The available values are: *Jira*.

Specific to JIRA dashboard options:

Priority Defines the JIRA priority of the ticket, such as Critical, Major, *Minor* and *Cosmetic*. These words depend on the language the JIRA account creating the ticket is set on. JIRA API provides numbers to avoid the translation problems: 2: *Critical*, 3: *Major*, 4: *Minor*, 5: *Cosmetic*.

Type Defines what type of JIRA ticket must be created. If this option is not defined, a *Task* ticket will be created.

CSS options:

CssPath Defines another CSS file to load with ***Site.css***.

DefaultCss Defines if the default CSS classes must be used. Values accepted are *true* and *false*. By default, its value is set to *true*.

Css Defines the global CSS class applied to the ticket.
Default class: *ticket-default*
Generated class: *TicketName*

CssTitle Defines the CSS class applied to the ticket name.
Default class: *ticket-default-title*
Generated class: *TicketName-title*

CssId Defines the CSS class of the id created in the dashboard system. It is only visible when editing the ticket.
Default class: *ticket-default-id*
Generated class: *TicketName-id*

CssStatus Defines the CSS class of the status of the ticket in the dashboard system. It is only visible when editing the ticket.
Default class: *ticket-default-status*
Generated class: *TicketName-status*

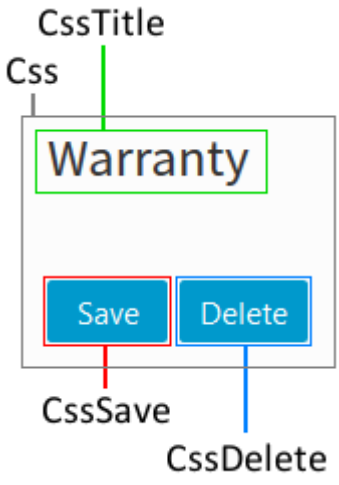
CssSave Defines the CSS class of the save button when creating a ticket.
Default class: *ticket-default-save*
Generated class: *TicketName-save*

CssDelete Defines the CSS class of the delete button when creating a ticket, to abort the ticket creation.
Default class: *ticket-default-delete*
Generated class: *TicketName-detele*

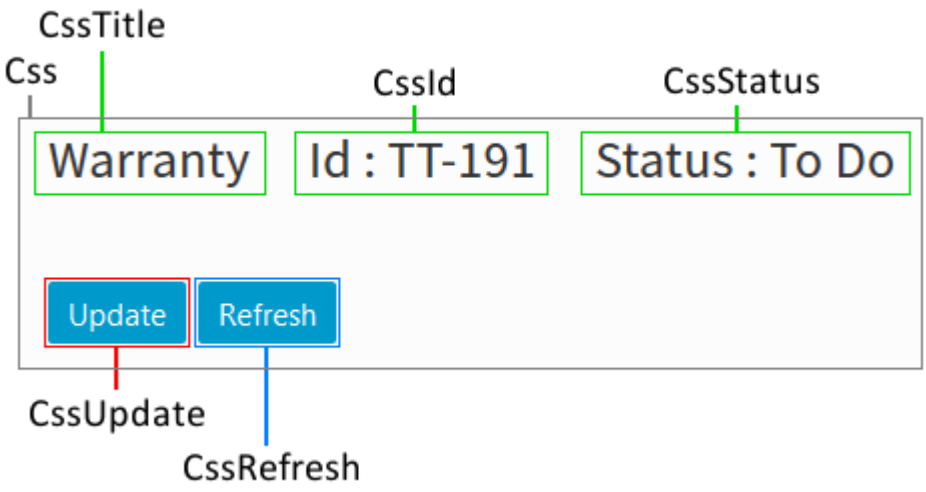
CssUpdate Defines the CSS class of the update button when editing a ticket, to commit changes in the dashboard system.
Default class: *ticket-default-update*
Generated class: *TicketName-update*

CssRefresh Defines the CSS class of the refresh button when editing a ticket, to fetch again the ticket information in the dashboard system.
Default class: *ticket-default-refresh*
Generated class: *TicketName-refresh*

Visual



When creating



When editing

III. Components

Common options for all components

Mandatory options:

Name Defines the name of the component. It must be unique among all the other components and be a one-word name.

Specific to JIRA dashboard options:

Dashboard Defines where the component is exported in the dashboard ticket. If this option is not defined, the component is not exported. Values accepted are: *Body*, *Summary*, *Reporter*, *Assignee* and *Custom*.
Custom is used to bind the component to custom fields in JIRA and should be used as *Custom:MyJiraFieldName*.

General options:

Hidden Defines if the component is visible or not. Values accepted are *true* or *false*. By default, its value is set to *false*.

DisplayedName Defines the name of the component to display in the ticket. It can be whatever, contrary to the *Name* option.

Preview Defines if the content of the component is displayed when a preview of the ticket appears. It is currently only used in Warranty tickets in the Web Viewer.

CSS options:

DefaultCss Defines if the default CSS classes must be used. Values accepted are *true* and *false*. By default, its value is set to *true*.

Common actions and functions for all components

Functions:

bool **IsVisible**() Returns *true* if the component is shown, *false* if hidden.

Actions:

SetVisible(bool) Shows the component if the argument is *true*, hides if *false*.

Dual actions:

Show(bool=*true*) Shows the component if the argument is *true*, hides if *false*.

Hide(bool=*true*) Hides the component if the argument is *true*, shows if *false*.

1. Attachment

Description

This component allows to upload files that will be stored in the dashboard ticket. All its options are cosmetic options, none of them changes the component behavior. This component doesn't have a label containing its name unlike many others and can only be once in a ticket.

Options

General options:

FileTitle Defines the text displayed above the uploaded files list. Default value is *Files*.

NoAttachment Defines the text displayed when no file is uploaded. Default value is *No attachment*.

UploadTitle Defines the text displayed above the upload part. Default value is *Upload*.

UploadButton Defines the text displayed in the upload button. Default value is *Upload.*

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-attachment*
Generated class: *TicketName-ComponentName*

CssFiles Defines the CSS class applied to the files part.
Default class: *ticket-default-attachment-files*
Generated class: *TicketName-ComponentName-files*

CssFilesTitle Defines the CSS class applied to the title of the files part.
Default class: *ticket-default-attachment-files-title*
Generated class: *TicketName-ComponentName-files-title*

CssFilesContent Defines the CSS class applied to the lines of uploaded lines.
Default class: *ticket-default-attachment-files-content*
Generated class: *TicketName-ComponentName-files-content*

CssFilesNoAttachment Defines the CSS class applied to the text displayed when no file is uploaded.
Default class: *ticket-default-attachment-files-noattachment*
Generated class: *TicketName-ComponentName-files-noattachment*

CssFilesElement Defines the CSS class applied to a single line of uploaded file.
Default class: *ticket-default-attachment-files-element*
Generated class: *TicketName-ComponentName-files-element*

CssFilesElementDelete Defines the CSS class applied to the delete icon of a single line of uploaded file.
Default class: *ticket-default-attachment-files-element-delete*
Generated class: *TicketName-ComponentName-files-element-delete*

CssFilesElementDelete Defines the CSS class applied to the file name a single line of uploaded file.
Default class: *ticket-default-attachment-files-element-title*
Generated class: *TicketName-ComponentName-files-element-title*

CssUpload Defines the CSS class applied to the upload part.
Default class: *ticket-default-attachment-upload*
Generated class: *TicketName-ComponentName-upload*

CssUploadTitle Defines the CSS class applied to the title of the upload part.
Default class: *ticket-default-attachment-upload-title*
Generated class: *TicketName-ComponentName-upload-title*

CssUploadContent Defines the CSS class applied to the content of the upload part.
Default class: *ticket-default-attachment-upload-content*
Generated class: *TicketName-ComponentName-upload-content*

CssUploadBrowse Defines the CSS class applied to the browse button.
Default class: *ticket-default-attachment-upload-browse*
Generated class: *TicketName-ComponentName-upload-browse*

CssUploadValidate Defines the CSS class applied to the upload button.
Default class: *ticket-default-attachment-upload-validate*
Generated class: *TicketName-ComponentName-upload-validate*

Actions and functions

Functions:

bool **HasAttachment**() Returns *true* if at least one file is uploaded, *false* otherwise.

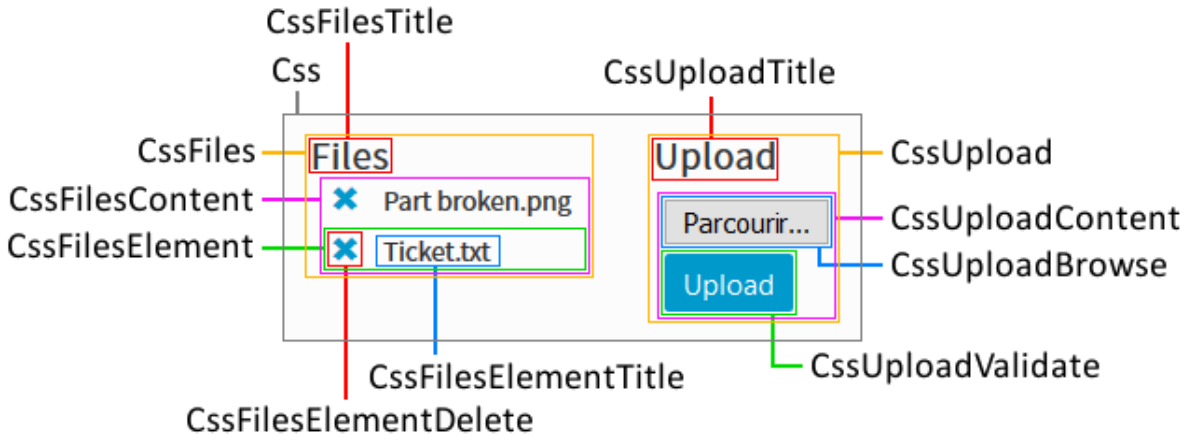
int **GetCount**() Returns the number of files uploaded.

bool **HasChanged**() Returns *true* (once) if an attachment is added or removed.

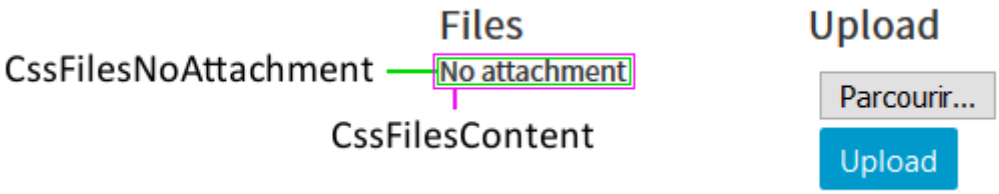
Actions:

RemoveAllAttachments() Removes all the files uploaded.

Visual



With attachments



Without attachments (all the other CSS classes work too)

2. Button

Description

This component is a simple button that triggers events when clicked on. This component doesn't have a label containing its name unlike many others, instead the text displayed inside it is either its name, or its displayed name if it exists.

Options

CSS options:

Css	Defines the global CSS class applied to the component.	
	Default class:	*ticket-default-button*
	Generated class:	*TicketName-ComponentName*

Actions and functions

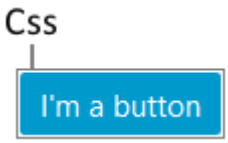
Functions:

bool **IsClicked**()	Returns *true* (once) when the button is clicked on.

Actions:

Click()	Triggers programmatically the button click.

Visual



3. CheckBox

Description

This component is a simple checkbox preceded by a label of its name or displayed name.

Options

General options:

Default Defines the default state of the checkbox. Values can be *true* or *checked* to check the checkbox, or *false* or *unchecked* to uncheck it.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-checkbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-checkbox-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the checkbox itself.
Default class: *ticket-default-checkbox-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

bool **IsChecked**() Returns *true* if the checkbox is checked, *false* otherwise.

bool **HasChanged**() Returns *true* (once) if the checkbox state changes.

Actions:

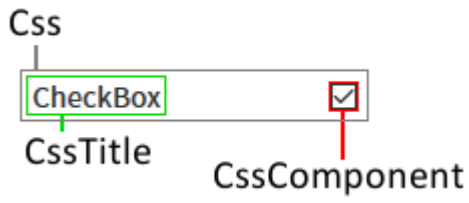
SetChecked(bool) Checks the checkbox if the argument is *true*, unchecks if false.

Dual actions:

Check(bool=*true*) Checks the component if the argument is *true*, uncheck otherwise.

Uncheck(bool=*true*) Unchecks the component if the argument is *true*, check otherwise.

Visual



4. DatePicker

Description

This component allows selecting a date in a calendar. It is preceded by a label of its name or displayed name.

Options

General options:

Format Defines the format used for the dates. It can be *dd/mm/yyyy*, *dd/mm/yy*, or any arrangement of the days, months, and years (ex: *yyyy/mm/dd*).

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-datepicker*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default- datepicker -title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the date displayed.
Default class: *ticket-default- datepicker -component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

string **GetDate**() Returns the date in the format specified in the *Format* option.

bool **HasChanged**() Returns *true* (once) if the date has changed.

bool **IsEmpty**() Returns *true* if the date is an empty field.

bool **Equal**(string) Returns *true* if the date given as argument is equal to the date in the component. The argument must be in the same format.

bool **NotEqual**(string) Returns *true* if the date given as argument is not equal to the date in the component. The formats must be the same.

bool **More**(string) Returns *true* if the date given as argument is greater than the date in the component. The formats must be the same.

bool **MoreEqual**(string) Returns *true* if the date given as argument is greater or equal to the date in the component. The formats must be the same.

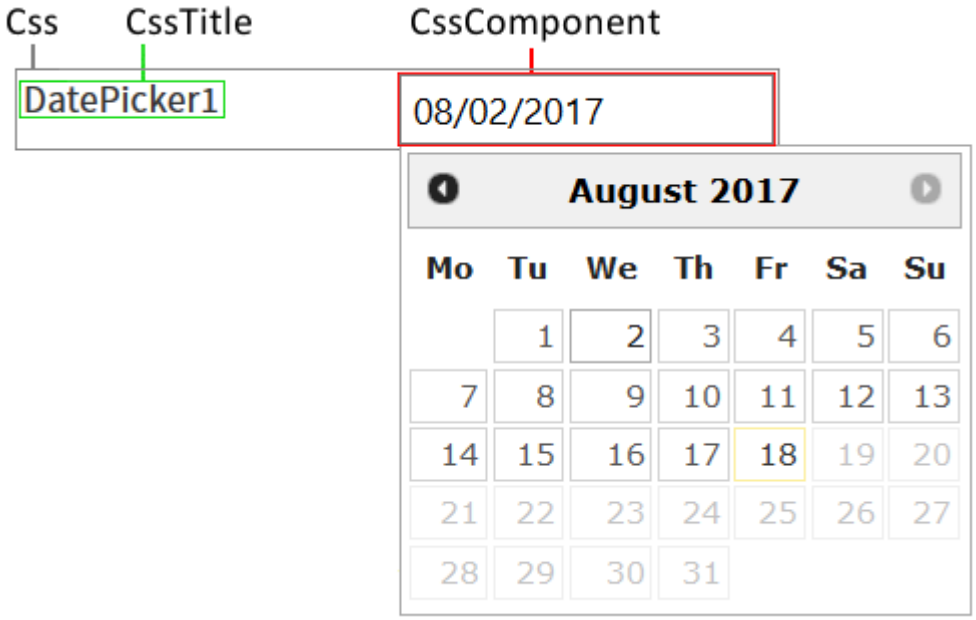
bool **Less**(string) Returns *true* if the date given as argument is less than the date in the component. The formats must be the same.

bool **LessEqual**(string) Returns *true* if the date given as argument is less or equal to the date in the component. The formats must be the same.

Actions:

SetDate(string)	Set the date to the one given as argument. The formats must be the same.
Reset()	Clear the date previously entered.

Visual



5. DropBox

Description

This component is a simple dropdown preceded by a label of its name or displayed name. It allows to select one value among several predefined values.

Options

General options:

Values Defines all the value likely to be selected. The values must be separated by semicolons (ex: *Option 1 ; Option 2 ; Option 3*).

Default Defines the default value selected. By default, the first value is selected.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-dropbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-dropbox-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the dropbox itself.
Default class: *ticket-default-dropbox-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

string **GetSelected**() Returns the value selected.

bool **GetIndexSelected**() Returns the index of the value selected (starts at 1).

bool **HasChanged**() Returns *true* (once) if the state of the dropbox changes.

Actions:

Select(string) Selects the value given as argument.

SelectAt(int) Selects the value at the given index. Nothing happens if the index is wrong.

Visual

6. Label

Description

This component is a line of text preceded by a label of its name or displayed name. It can only be changed programmatically.

Options

General options:

Default Defines the default text is the label. By default, it is empty.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-label*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-label-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the text itself.
Default class: *ticket-default-label-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

string **GetText** () Returns the text in the label.

int **GetLength**() Returns the length of the text in the label.

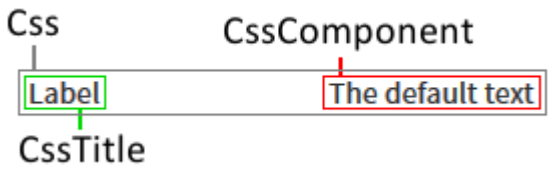
bool **IsEmpty**() Returns *true* if there is no text in the label, *false* otherwise.

bool **HasChanged**() Returns *true* (once) if the text changes.

Actions:

SetText(string) Sets the text in the label.

Visual



7. Mail

Description

This component allows to send mails by binding other component contents as the body of the mail, the subject, etc... It does not appear visually and doesn't have any CSS options. Proper SMTP configurations in ***profile.config*** are needed to send mails.

Options

Mandatory options:

To	Defines the component affected to the *To* field of a mail. The value can be between quotes to give it directly.
Body	Defines the component affected to the *Body* field of a mail. The value can be between quotes to give it directly.
Subject	Defines the component affected to the *Subject* field of a mail. The value can be between quotes to give it directly.

General options:

Attachment	Defines if the attachments of the ticket are also send in the mail. The value can be *true* or *false*. By default, it is *false**.***
Confirmation	Defines if a confirmation is sent back to the user. The value can be *true* or *false*. By default, it is *false**.***
Content	Defines which components content will be in the mail too. The component names must be separated with semicolons. Their content will be exported at the end of the mail.

Actions and functions

Functions:

string **GetTo**()	Returns the value affected to *To* option.
bool **HasToChanged**()	Returns the *true* (once) if the *To* option has changed.
string **GetSubject**()	Returns the value affected to *Subject* option.
bool **HasSubjectChanged**()	Returns the *true* (once) if the *Subject* option has changed.
string **GetBody**()	Returns the value affected to *Body* option.
bool **HasToChanged**()	Returns the *true* (once) if the *Body* option has changed.
bool **GetAttachment**()	Returns the value affected to the *Attachment* option.
bool **HasAttachmentChanged**()	Returns *true* (once) if the *Attachment* option has changed.
bool **GetConfirmation**()	Returns the value affected to the *Confirmation* option.

bool **HasConfirmationChanged**()	Returns the *true* (once) if the *Confirmation* option has changed.
List<string> **GetContent**()	Returns all the component names affected to the *Content* option.
bool **HasContentChanged**()	Returns *true* (once) if the *Content* option has changed.

Actions:

SetTo(string)	Sets a component name to the *To* option. A direct value can be given between quotes.
SetSubject(string)	Sets a component name to the *Subject* option. A direct value can be given between quotes.
SetBody(string)	Sets a component name to the *Body* option. A direct value can be given between quotes.
SetAttachment(bool)	Sets the *Attachment* option with the value passed as argument.
SetConfirmation(bool)	Sets the *Confirmation* option with the value passed as argument.
SetContent(List<string>)	Sets the components given as a list as argument to the *Content* option.

8. MultiCheckBox

Description

This component is a group of several checkboxes. The group is preceded by a label of its name or displayed name and each checkbox is followed by its value name.

Options

General options:

Values Defines all the value likely to be checked. The values must be separated by semicolons (ex: *Option 1 ; Option 2 ; Option 3*).

Defaults Defines the default values checked. The values must be separated by semicolons. By default, no value is checked.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-multicheckbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-multicheckbox-title*
Generated class: *TicketName-ComponentName-title*

CssContent Defines the CSS class applied to all the checkboxes.
Default class: *ticket-default-multicheckbox-content*
Generated class: *TicketName-ComponentName-content*

CssElement Defines the CSS class applied to one checkbox and its value name.
Default class: *ticket-default-multicheckbox-element*
Generated class: *TicketName-ComponentName-element*

CssElementTitle Defines the CSS class applied to the value names of the checkboxes.
Default class: *ticket-default-multicheckbox-element-title*
Generated class: *TicketName-ComponentName-element-title*

CssElementComponent Defines the CSS class applied to the checkboxes themselves.
Default class: *ticket-default-multicheckbox-element-component*
Generated class: *TicketName-ComponentName-element-component*

Actions and functions

Functions:

bool **IsChecked**(string) Returns *true* if the checkbox value given as argument is checked, *false* otherwise.

bool **IsCheckedAt**(int) Returns *true if* the checkbox at the index given as argument is checked, *false* otherwise. The index starts at 1. Returns *false* if the index is wrong.

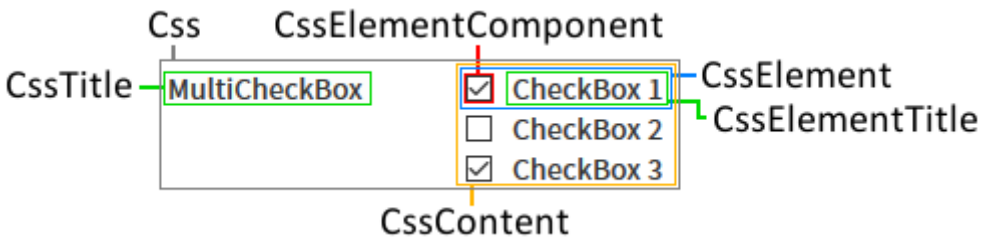
List<string> **GetChecked**() Returns all the checkbox values which are checked.

List<string> **GetUnChecked**()	Returns all the checkbox values which are not checked.
List<int> **GetIndexesChecked**()	Returns all the indexes of the checkboxes which are checked. Indexes starts at 1.
List<int> **GetIndexesUnchecked**()	Returns all the indexes of the checkboxes which are not checked. Indexes starts at 1.
int **GetCheckedCount**()	Returns the total number of checked checkboxes.
int **GetUncheckedCount**()	Returns the total number of unchecked checkboxes.
bool **HasChanged**()	Returns *true* (once) if a checkbox is checked or unchecked.

Actions:

CheckAll()	Checks all the checkboxes.
UncheckAll()	Unchecks all the checkboxes.
Check(string)	Checks the checkbox value given as argument.
Uncheck(string)	Unchecks the checkbox value given as argument.
MultiCheck(List<string>)	Checks the checkbox values given as argument.
MultiUncheck(List<string>)	Unchecks the checkbox values given as argument.
CheckAt(int)	Checks the checkbox value at the index given as argument.
UncheckAt(int)	Unchecks the checkbox value at the index given as argument.
MultiCheckAt(List<int>)	Checks the checkbox values at the indexes given as argument.
MultiUncheckAt(List<int>)	Unchecks the checkbox values at the indexes given as argument.

Visual



9. MultiDropBox

Description

This component is a dropdown where multiple values can be selected. The component is preceded by a label of its name or displayed.

Options

General options:

Values Defines all the value likely to be selected. The values must be separated by semicolons (ex: *Option 1 ; Option 2 ; Option 3*).

Defaults Defines the default values selected. The values must be separated by semicolons. By default, no value is selected.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-multidropbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-multidropbox-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the dropdown.
Default class: *ticket-default-multidropbox-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

bool **IsSelected**(string) Returns *true* if the value given as argument is selected, *false* otherwise.

bool **IsSelectedAt**(int) Returns *true if* the value at the index given as argument is selected, *false* otherwise. The index starts at 1. Returns *false* if the index is wrong.

List<string> **GetSelected**() Returns all the values which are selected.

List<string> **GetUnselected**() Returns all the values which are not selected.

List<int> **GetIndexesSelected**() Returns all the indexes of the values which are selected. Indexes starts at 1.

List<int> **GetIndexesUnselected**() Returns all the indexes of the values which are not selected. Indexes starts at 1.

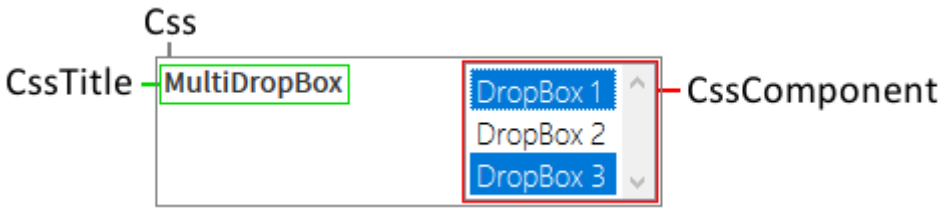
int **GetSelectedCount**() Returns the total number of selected values.

int **GetUnselectedCount**()	Returns the total number of unselected checkboxes.
bool **HasChanged**()	Returns *true* (once) if a value is selected or unselected.

Actions:

SelectAll()	Selects all the values.
UnselectAll()	Unselects all the values.
Selected(string)	Selects the value given as argument.
Uncheck(string)	Unselects the value given as argument.
MultiSelect(List<string>)	Selects the values given as argument.
MultiUnselect(List<string>)	Unselects the values given as argument.
SelectAt(int)	Selects the value at the index given as argument.
UncheckAt(int)	Unselects the value at the index given as argument.
MultiSelectAt(List<int>)	Selects the values at the indexes given as argument.
MultiUnselectAt(List<int>)	Unselects the values at the indexes given as argument.

Visual



10. MultiTextBox

Description

This component is a textbox which can contain several lines, and can be resized at will. It is preceded by a label of its name or displayed name.

Options

General options:

Default Defines the default text in the textbox.

Columns Defines the default horizontal size of the textbox, in number of characters (integer). Default value is 50.

Rows Defines the default vertical size of the textbox, in number of characters (integer). Default value is 2.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-multitextbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-multitextbox-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the text.
Default class: *ticket-default-multitextbox-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

string **GetText**() Returns the text in the textbox.

bool **IsEmpty**() Returns *true* if there is no text in the textbox, *false* otherwise.

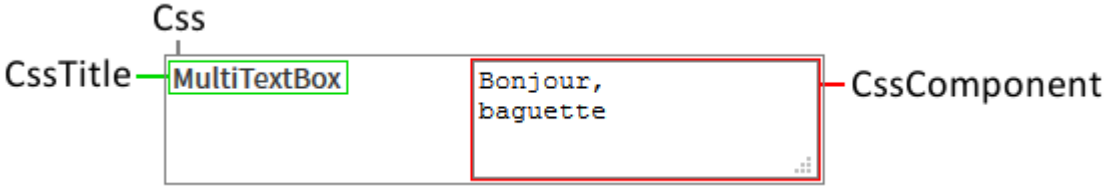
bool **HasChanged**() Returns *true* (once) if the text in the textbox is modified.

bool **HasRegex**(string) Returns *true* if the regular expression given as argument matches the text in the textbox.

Actions:

SetText(string) Sets the text in the textbox with the one given as argument.

Visual



11. PartList

Description

This component is a list of parts. It is a table where each line has a part name, a part number, and a quantity that can be modified. All its options are cosmetic options, none of them changes the component behavior. The first cell of the table (top left corner) is the name or the displayed name of the component. Quantity fields can have values between 0 and 999 (included).

Options

General options:

NameTitle Defines the placeholder text in the part name fields. By default, it is *Part*.

NumerTitle Defines the placeholder text in the part number fields. By default, it is *Part number*.

DetailsTitle Defines the text of the header above the part names and the part numbers. By default, it is *Part details*.

QuantityTitle Defines the text of the header above the quantities. By default, it is *Qty*.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-partlist*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the first cell of the component.
Default class: *ticket-default-partlist-title*
Generated class: *TicketName-ComponentName-title*

CssDetails Defines the CSS class applied to the details header.
Default class: *ticket-default-partlist-details*
Generated class: *TicketName-ComponentName-details*

CssQuantity Defines the CSS class applied to the quantity header.
Default class: *ticket-default-partlist-quantity*
Generated class: *TicketName-ComponentName-quantity*

CssElementName Defines the CSS class applied to the part name fields.
Default class: *ticket-default-partlist-element-name*
Generated class: *TicketName-ComponentName-element-name*

CssElementNumber Defines the CSS class applied to the part number fields.
Default class: *ticket-default-partlist-element-number*
Generated class: *TicketName-ComponentName-element-number*

CssElementQuantity Defines the CSS class applied to the quantity fields.
Default class: *ticket-default-partlist-element-quantity*
Generated class: *TicketName-ComponentName-element-quantity*

Actions and functions

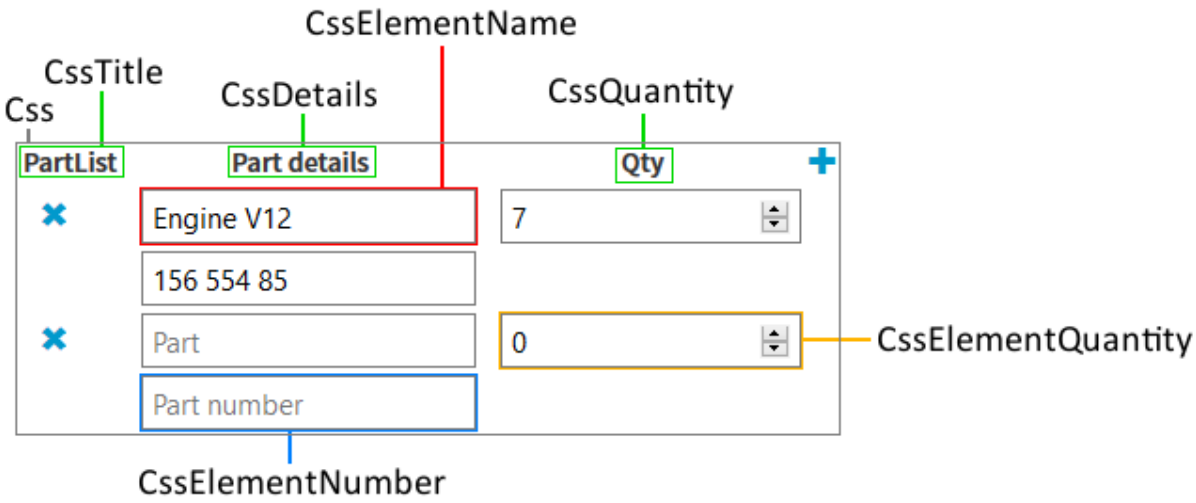
Functions:

bool **GetPartCount**()	Returns the total number of parts (lines of the table).
bool **HasPart**()	Returns *true* if the number of parts (lines in the table) is greater than 0.
bool **HasChanged**()	Returns *true* (once) if a field is changed or if a part is added or removed.
bool **AllPartsValid**()	Returns *true* if all fields are completed and if the quantities are greater than 0.
int **GetQuantities**()	Returns the sum of all the quantities.

Actions:

AddPart()	Adds a new part.
RemoveAllParts()	Removes all the parts, clearing the table.
ResetAllQuantities()	Sets all the quantity fields to 0.
IncrementAllQuantities(int=1)	Increments all the quantity fields with the given value.
DecrementAllQuantities(int=1)	Decrements all the quantity fields with the given value.
MultiplyAllQuantities(int)	Multiplies all the quantity fields with the given value.
DivideAllQuantities(int)	Divides all the quantity fields with the given value. The resulting quantities remain integers.

Visual



12. Quantity

Description

This component is a field allowing integer selection. It is preceded by a label of its name or displayed name.

Options

General options:

Default	Defines the default number in the field (integer). Default value is 0.
Min	Defines the smallest value possible of the field (integer). Default value is 0.
Max	Defines the highest value possible of the field (integer). Default value is 999.
Step	Defines the increment and decrement value of the field when using the arrows (integer). Default value is 0.

CSS options:

Css	Defines the global CSS class applied to the component. Default class: *ticket-default-quantity* Generated class: *TicketName-ComponentName*
CssTitle	Defines the CSS class applied to the label of the component. Default class: *ticket-default-quantity-title* Generated class: *TicketName-ComponentName-title*
CssComponent	Defines the CSS class applied to the text. Default class: *ticket-default-quantity-component* Generated class: *TicketName-ComponentName-component*

Actions and functions

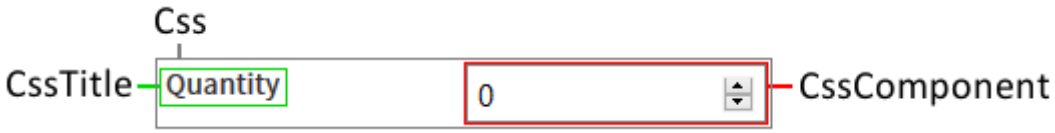
Functions:

int **GetValue**()	Returns the value in the component.
int **GetMin**()	Returns smallest value possible of the component.
int **GetMax**()	Returns highest value possible of the component.
int **GetStep**()	Returns the increment value of the component.
bool **IsMin**()	Returns *true* if the value in the component is the smallest possible.
bool **IsMax**()	Returns *true* if the value in the component is the highest possible.
bool **HasChanged**()	Returns *true* (once) if the value in the component is modified.
bool **HasMinChanged**()	Returns *true* (once) if the smallest value possible is modified.
bool **HasMaxChanged**()	Returns *true* (once) if the highest value possible is modified.
bool **HasStepChanged**()	Returns *true* (once) if the value in the field is modified.

Actions:

SetValue(int)	Sets the value of the component with the value given as argument.
Increment()	Increments the value of the component with the step value.
Decrement()	Decrements the value of the component with the step value.
SetMin(int)	Sets the smallest value possible with the value given as argument.
SetMin(int)	Sets the highest value possible with the value given as argument.
SetStep(int)	Sets the increment and decrement value of the component with the value given as argument.

Visual



13. Radio

Description

This component is a group of radio buttons, which are basically exclusive checkboxes. It is preceded by a label of its name or displayed name and each radio button is followed by its value name.

Options

General options:

Values Defines all the value likely to be selected. The values must be separated by semicolons (ex: *Option 1 ; Option 2 ; Option 3*).

Default Defines the default selected value. By default, the first value is selected.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-radio*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-radio-title*
Generated class: *TicketName-ComponentName-title*

CssContent Defines the CSS class applied to all the radio buttons.
Default class: *ticket-default-radio-content*
Generated class: *TicketName-ComponentName-content*

CssElement Defines the CSS class applied to one radio button and its value name.
Default class: *ticket-default-radio-element*
Generated class: *TicketName-ComponentName-element*

CssElementTitle Defines the CSS class applied to the value names of the radio buttons.
Default class: *ticket-default-radio-element-title*
Generated class: *TicketName-ComponentName-element-title*

CssElementComponent Defines the CSS class applied to the radio buttons themselves.
Default class: *ticket-default-radio-element-component*
Generated class: *TicketName-ComponentName-element-component*

Actions and functions

Functions:

string **GetSelected**() Returns the value name of the radio button selected.

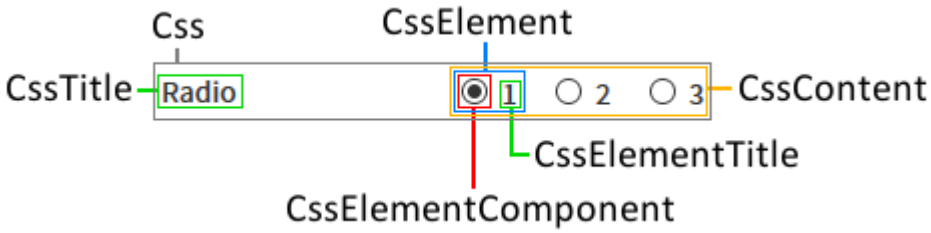
int **GetIndexSelected**() Returns the index of the radio button selected (starts at 1).

bool **HasChanged**() Returns *true* (once) when a radio button is selected.

Actions:

Select(string)	Selects the radio button given as argument.
SelectAt(int)	Selects the radio button whose index is given as argument.

Visual



14. Range

Description

This component is a slider to select integer values between an interval. It is preceded by a label with its name or its displayed name. The actual value selected is not shown, but can be displayed in other components using behaviors.

Options

General options:

Default Defines the default value of the slider (integer). Default value is 0.

Min Defines the smallest value possible of the slider (integer). Default value is 0.

Max Defines the highest value possible of the slider (integer). Default value is 999.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-range*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-range-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the slider.
Default class: *ticket-default-range-content*
Generated class: *TicketName-ComponentName-content*

Actions and functions

Functions:

int **GetValue**() Returns the value of the slider.

int **GetMin**() Returns smallest value possible of the slider.

int **GetMax**() Returns highest value possible of the slider.

bool **IsMin**() Returns *true* if the value in the component is the smallest possible.

bool **IsMax**() Returns *true* if the value in the component is the highest possible.

bool **HasChanged**() Returns *true* (once) if the value in the component is modified.

bool **HasMinChanged**() Returns *true* (once) if the smallest value possible is modified.

bool **HasMaxChanged**() Returns *true* (once) if the highest value possible is modified.

Actions:

SetValue(int)	Sets the value of the slider with the value given as argument.
SetMin(int)	Sets the smallest value possible with the value given as argument.
SetMin(int)	Sets the highest value possible with the value given as argument.

Visual

15. TextBox

Description

This component is a textbox which can contain only one line. It is preceded by a label of its name or displayed name.

Options

General options:

Default Defines the default text in the textbox.

CSS options:

Css Defines the global CSS class applied to the component.
Default class: *ticket-default-textbox*
Generated class: *TicketName-ComponentName*

CssTitle Defines the CSS class applied to the label of the component.
Default class: *ticket-default-textbox-title*
Generated class: *TicketName-ComponentName-title*

CssComponent Defines the CSS class applied to the text.
Default class: *ticket-default-textbox-component*
Generated class: *TicketName-ComponentName-component*

Actions and functions

Functions:

string **GetText**() Returns the text in the textbox.

bool **IsEmpty**() Returns *true* if there is no text in the textbox, *false* otherwise.

bool **HasChanged**() Returns *true* (once) if the text in the textbox is modified.

bool **HasRegex**(string) Returns *true* if the regular expression given as argument matches the text in the textbox.

Actions:

SetText(string) Sets the text in the textbox with the one given as argument.

Visual