IQL Components and Dialogs

1.1 Components

An Input Query Language (IQL) dialog is composed by IQL components. There are two types of components: base components and composite components. Base components store user input and have their own name, unique identifier, default values and constraints. The base component structure looks like this:

```
<name> '<prompt>' <type> ('<default value>'){<constraints>}
```

as for example

```
age 'insert your age:' Integer('5'){min=1 max=120}
```

Composite components contain other components. Their basic structure is the following:

```
'<prompt>'  <type> {<components>}
```

As in

```
'personal details' Tab{name 'Insert your name' String}
```

In the following section, we explain in details how to use each of the IQL components and provide examples.

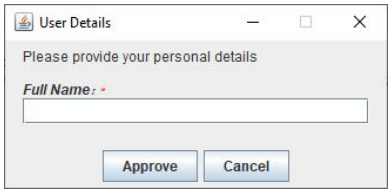
1.1.1 String Component

The simplest most basic base component is the String component; an example dialog containing just a String component would look as follows:

```
IQL.run("""
  'User Details' Single('Please provide your personal details')
  name 'Full Name:' String
  """)
```

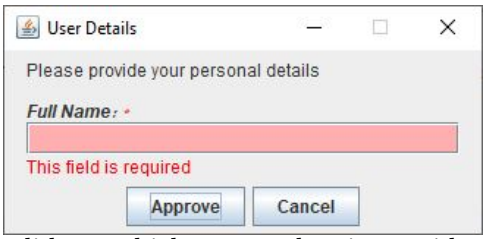
For simplicity, from now on we will avoid the Java boilerplate, `IQL.run(..)`, and focus only on IQL. So, the code above will look as follows:

```
'User Details' Single(
  'Please provide your personal
   details')
name 'Full Name:' String
```



The String component stores a string input from the user. Its default value is displayed to the user when the dialog opens. In addition, the String component allows the following constraints:
`min` – The minimum number of letters that the user must provide.
`max` - The maximum number of letters that the user cans provide.
`optional` – If the field is optional or required (required by default).
`placeholder` – A string of characters that temporarily takes the place of the final data. To distinguish the placeholder from the contained text, the color of the placeholder text is grey and it will be removed if the field is in focus, if there is a default value or if some input is provided. Note that even if there is a default value the placeholder text can still be visualized if the user manually empties the field content.
`display` – Define the component display look. The supported displays are `inline` and `block`. The string component default display is `block`.
`regex` – Regular expressions are a very commonly used tool. By using regex you can use a sequence of characters to define a pattern. Although there is a standard for regex that was defined by IEEE associated with POSIX, many programming languages do not follow it and implementing their own regular expression. IQL is using Java regex.

By default, the String component will check if the user input matches the provided constraints and provides an error message otherwise. In the first example, we can see that IQL generates a dialog with one field of type String. The prompt of the field is "Full Name:". We can see that this field is required: red star mark. Required is the default behavior for all the components.
If the user tried to press on approve without providing any information he will get the following error:

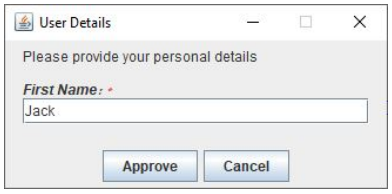


As we can see IQL uses an inline validate, which means that it provides the specific error message next to its field. This way of providing error massages next to its field show a good result by increasing the success rates and satisfaction rating while decreasing the errors made and completion times of

the users. IQL identify that the user failed to fulfill the required constraint and provide him an error message. When the user enters his name and press approve the user data will be saved and can be accessed later.

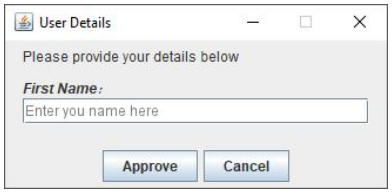
Here some more examples using the various constraints and default value: Defaults value can be provided by using circular brackets after the component type as following:

```
'User Details' Single(
  'Please provide your personal
   details')
name 'Full Name:' String('Jack')
```

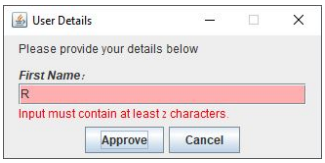


Consider the following dialog with many constraints:

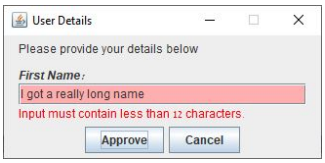
```
'User Details' Single(
  'Please provide your personal
   details')
name 'Full Name:' String{
  min=2
  max=12
  optional
  placeholder='Enter you name
   here'
  regex='[a-zA-Z]+'
  }
```



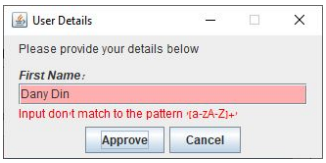
The placeholder constraint displays for a user a string of characters when he did not enter any value or if the field is not in focus. Those characters will not be saved if we press on approve button. In addition, we can see that there is no red star next to the prompt "First Name:". This is because we added the optional constraint.
Here are some examples of error messages that the user will get if they violate the constraints.



(1)



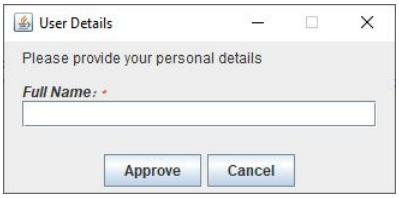
(2)



(3)

As we can see, in the first example (image 1) the user fails to fulfill the min constraint. It expects at least 2 characters in order to be a valid name. In the second example (image 2) the user fails to fulfill the max constraint. It expects a maximum of 12 characters in order to be a valid name. In the third example (image 3) the user fails to fulfill the regex constraint. It expects from the user a specific sequence of characters. In this example, it expects the user to match the following regex '[a-zA-Z]+'.

The String component supports two display types: `inline` and `block`.



(4)

(5)

By default, the component display is block (image 4). In order to change the String component display to inline (image 5) we need to provide the inline display constraint as follows:

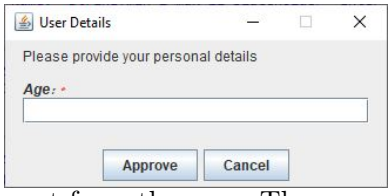
```
'User Details' Single('Please provide your personal
   details')
name 'Full Name:' String{inline}
```

As we can see from the examples if the component display is `block` then the title is placed above the input field, and when it is `inline` the title is placed beside the input field.

1.1.2 Integer Component

An Integer is a base component. An example dialog containing just an Integer component would look as follows:

```
'User Details' Single(
  'Please provide your personal
   details')
  age 'Age:' Integer
```

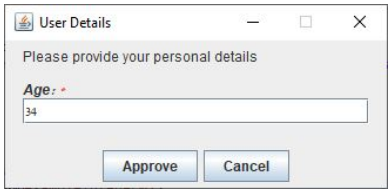


The Integer component stores an integer input from the user. The component prevents the user to enter characters that will not form an integer. Its default value is displayed to the user when the dialog opens. In addition, the Integer component allows the following constraints:
`min` – The minimum integer value that the user cans provide.
`max` - The maximum integer value that the user cans provide.
`optional`, `placeholder`, `display`, `regex` – Those constraints have the same behavior as in the String component.

The Integer component validates the user input and provides an error message if needed exactly as we have seen in the String component. In addition, also the placeholder has the same behavior at the String component.

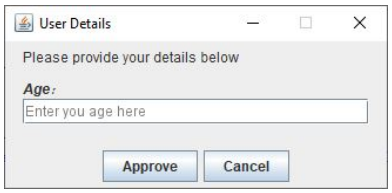
Here some examples using the various constraints and default value:
Defaults value can be provided by using circular brackets after the component type as following:

```
'User Details' Single(
  'Please provide your personal
   details')
age 'Age:' Integer('34')
```

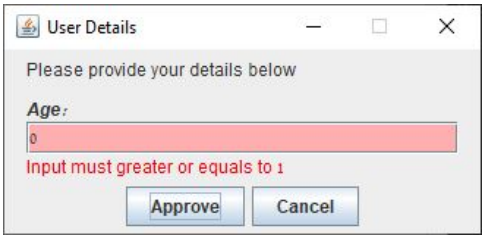


Please notice that the Integer default value must be surrounded by a single quote.
Consider the following dialog with many constraints.

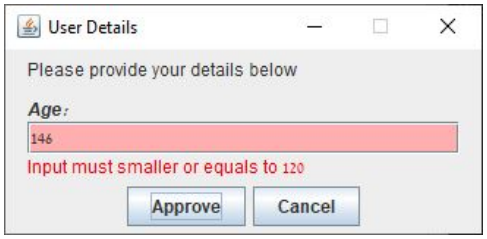
```
'User Details' Single(
  'Please provide your personal
   details')
age 'Age:' Integer{
  min=1
  max=120
  placeholder='Enter you age here'
  optional
  }
```



Here are examples of error messages that the user will get if violated the min and max constraints.

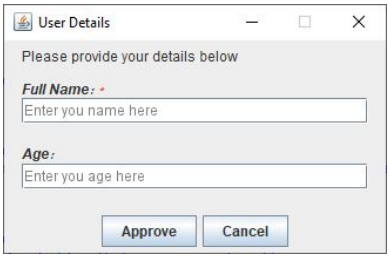


(1)



(2)

As we can see, in the first example (image 1) the user fails to fulfill the min constraint. It expects to get an integer value greater or equal to 1 in order to be a valid age. In the second example (image 2) the user fails to fulfill the max constraint. It expects to get an integer value smaller or equal to 120 in order to be a valid age. As we saw for the String component, also the Integer component supports exact two types of display types: `inline` and `block` and have the same behavior as the String component.



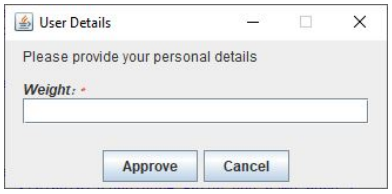
(4)

(5)

1.1.3 Decimal Component

Decimal is a base component. An example dialog containing just a Decimal component would look as follows:

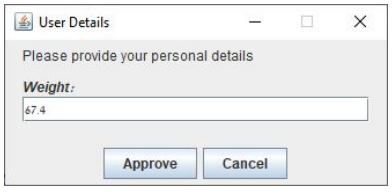
```
'User Details' Single(
  'Please provide your personal
   details')
weight 'Weight:' Decimal
```



The Decimal component stores a decimal input from the user. The component prevents from the user to enter characters that will not form a decimal value. Its default value is displayed to the user when the dialog opens. In addition, the Decimal component allows the following constraints:
`min` – The minimum decimal value that the user cans provide.
`max` - The maximum decimal value that the user cans provide.
`optional`, `placeholder`, `regex`, `display` – Those constraints have the same behavior as in the Integer component.

the Decimal component validates the user input and provide an error message if needed exactly as we have seen in the Integer component. In addition, all the other constraints behave exactly as in the Integer component with one exception. `min` and `max` constraints validate that the user input is a decimal and not an integer.
An example for dialog that contains a Decimal component with many constraints will look as follows:

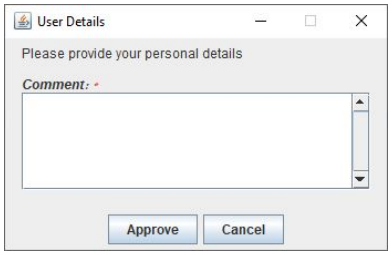
```
'User Details' Single(
  'Please provide your personal
   details')
weight 'Weight:' Decimal('67.4'){
  min=10.1
  max=220.6
  placeholder='Enter you weight
   in Kg'
  optional
  }
```



1.1.4 TextArea Component

TextArea is a base component. An example dialog containing just a TextArea component would look as following:

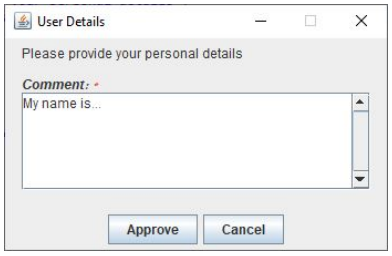
```
'User Details' Single(
  'Please provide your personal
   details')
comment 'Comment:' TextArea
```



The TextArea component stores multiple lines of string input from the user. Its default value is displayed to the user when the dialog opens. In addition, the String component allows the following constraints:
`min`, `max`, `optional`, `placeholder`, and `display` – All of the TextArea constraints have the same behavior as in the String component.

TextArea component validates the user input and provide an error message if needed exactly as we have seen in the String component.
Here some more examples using the various constraints and default value:
Defaults value can be provided by using circular brackets after the component type as following:

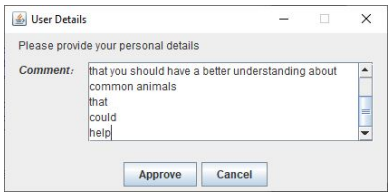
```
'User Details' Single(
  'Please provide your personal
   details')
comment 'Comment:' TextArea(
  'My name is...')
```



(1)

Consider the following dialog with many constraints.

```
'User Details' Single(
  'Please provide your personal
   details')
weight 'Weight:' TextArea{
  min=10
  max=300
  placeholder='Enter your comment
   here'
  optional
  }
```



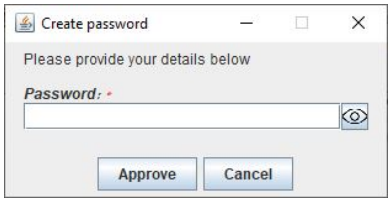
(2)

In the first example (image 1) we can see the use of the default values. The default value is display at the top left corner of the text box. In the second example (image 2) we can see a use of the `inline` display. In this display, the prompt is aligned with the top left corner of the `TextArea`. Also, we can see that if there are more rows than the `TextArea` can contain then the user can use the scroll bar in order to go up and down to see all the text.

1.1.5 Password Component

Password is a base component. An example dialog containing just a Password component would look as follows:

```
'Create password' Single(
  'Please provide your details
   below')
password 'Password:' Password
```

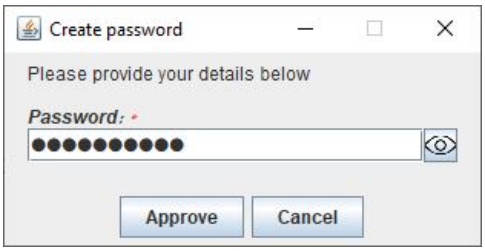


The Password component stores a text input from the user. Its default value is displayed to the user when the dialog opens. It has a button to hide and show the user input. In addition, the Password component allows the following constraints:
`min`, `max`, `optional`, `placeholder`, `regex` and `display` – All of the Password constraints have the same behavior as in the String component.
While using the regex constraint it is important to select a regex that will force the user to insert a strong password. Some basic guidelines for a strong password:
- The length of the password needs to be at least 6 to 10 characters
- It needs to contain upper case and lower case characters
- It needs to contain digits - It needs to contain at least one special characters (e.g @, *, [)
An example for a regex that follows those rules is:

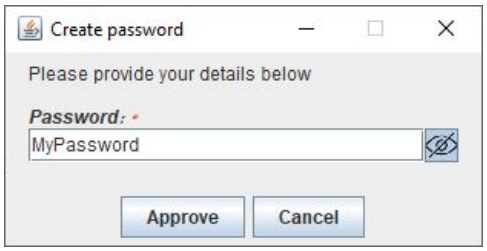
Password component validates the user input and provide an error message if needed exactly as we have seen in the String component.

Defaults value can be provided by using circular brackets after the component type as follows:

```
'Create password' Single(
  'Please provide your details
   below')
password 'Password:' Password('MyPassword')
```



(1)



(2)

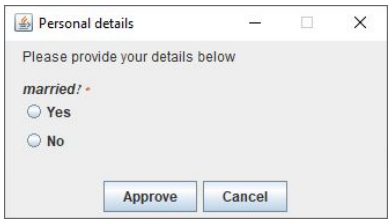
In the first example (image 1) we can see that the password is a mask with

circles. In order to see the password we need to click on the "eye" button and we will be able to see the password as we can see in image 2.

1.1.6 Boolean Component

Boolean is a base component. An example dialog containing just a Boolean component would look as follows:

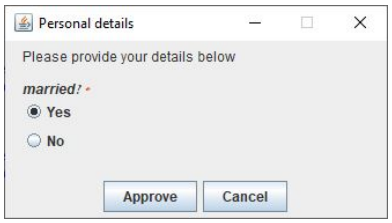
```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean
```



The Boolean component optionally stores a boolean from the user. Its default value is displayed to the user when the dialog opens. It has two options: yes and no. If none of them was selected then it will be saved as a null value. In addition, the Boolean component allows the following constraints:
`optional` – The behavior same as all the components above.
`display` – Define the component display look. The supported displays are `inline`, `block`, `inlineList` and `blockList`. The Boolean component default display is `block`.

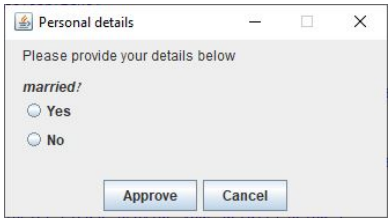
Here some examples using the various constraints and default value: Defaults value can be provided by using circular brackets after the component type as following:

```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean('true')
```



Consider the following dialog with optional constraints and different display values.

```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean{
  optional
  block
  }
```



(1)

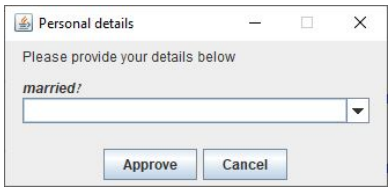
```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean{
  optional
  inline
  }
```

(2)

```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean{
  optional
  inlineList
  }
```

(3)

```
'Personal details' Single(
  'Please provide your details
   below')
married 'Married?' Boolean{
  optional
  blockList
  }
```



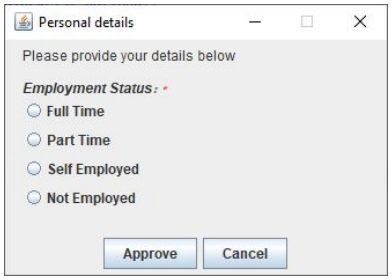
(4)

As we can see, in all examples we use the optional constraint, which means that the user is not required to select a value. In the first example (image 1) we can see the default display, which is `block`. In this display, the prompt is located above two radio buttons that align vertically and the user can select and deselect by clicking on a selected value. The second example (image 2) we use the `inline` display. In this display, the prompt is located at the side of two radio buttons that align horizontally. In the third example (image 3) we use the `inlineList` display. In this display, the prompt is located at the side of a list. In addition, in this display the first row is empty to provide the user the option to select none of the options. In the fourth example (image 4) we use the `blockList` display. In this display, the prompt is located above a list.

1.1.7 SingleOpt Component

SingleOpt is a base component. An example dialog containing just a SingleOpt component would look as following:

```
'Personal details' Single(
  'Please provide your details
   below')
employment 'Employment Status:'
  SingleOpt['Full Time|Part Time
    |Self Employed|Not Employed']
```



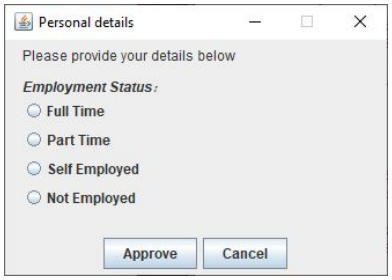
The SingleOpt component provides a set of options that the user can select one or none of them. The set of options must be provided by using square brackets in the component type. SingleOpt default value is displayed to the user when the dialog opens. If none of the options was selected then it will save a null value. In addition, the SingleOpt component allows the following constraints:
`optional` – If the field is optional or required (required by default).
`display` – Define the component display look. The supported displays are `blockRadio`, `blockList` and `inlineList`. SingleOpt component default display is `blockRadio`.

By default the SingleOpt component will check if the user input matches the provided constraints and provide an error message as other components do. SingleOpt component looks and feels like the Boolean component but has two main differences. First, we can define the number of items that the user will select from and second we can define those item values. These two differences provide the developer an option to generate a new SingleOpt component that looks and feels like a Boolean component but with different values. For example:

```
'Personal details' Single(
  'Please provide your details
   below')
married 'Are you married?'
  SingleOpt['True|False']
```

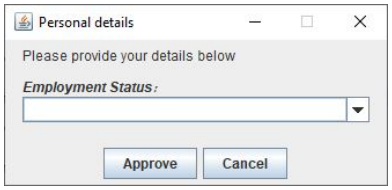
Consider the following dialog with optional constraint and different display values.

```
'Personal details' Single(
  'Please provide your details
   below')
employment 'Employment Status:'
  SingleOpt['Full Time|Part Time
    |Self Employed|Not Employed']{
  optional
  blockRadio
  }
```



(1)

```
'Personal details' Single(
  'Please provide your details
   below')
employment 'Employment Status:'
  SingleOpt['Full Time|Part Time
    |Self Employed|Not Employed']{
  optional
  blockList
  }
```



(2)

```
'Personal details' Single(
  'Please provide your details
   below')
employment 'Employment Status:'
  SingleOpt['Full Time|Part Time
    |Self Employed|Not Employed']
    ('Self Employed'){
  optional
  inlineList
  }
```

(3)

As we can see, in all examples we use the optional constraint, which means that the user is not required to select a value. In the first example (image 1) we can see the default display, which is `blockRadio`. In this display, the prompt is located above the radio buttons that align vertically. In this display the user can select and deselect by clicking on a selected value. In the second example (image 2) we use the `blockList` display. In this display, the prompt is locate above a list. In this display the first row is empty to provide the user the option to select none of the options. In the third example (image 3) we use the `inlineList` display. In this display the prompt is locate at the side of a list.

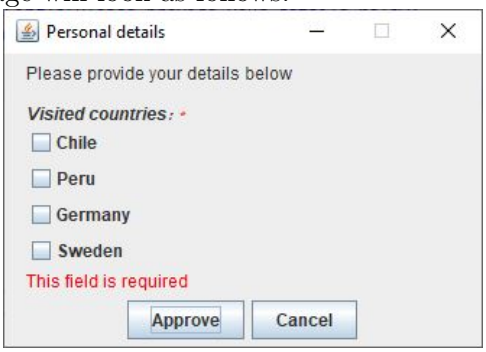
1.1.8 MultiOpt Component

MultiOpt is a base component. An example dialog containing just a MultiOpt component would look as follows:

```
'Personal details' Single(
  'Please provide your details
   below')
countries 'Visited countries:'
  MultiOpt['Chile|Peru
    |Germany|Sweden']
```

Like SingleOpt components the MultiOpt component provides a set of options that the user can select from. The differences are that in the MultiOpt component the users can select a multiple option or none while in the SingleOpt component they can select only one. Besides, the options in the MultiOpt component have checkboxes while in the SingleOpt component they have radio buttons. The set of options must be provided by using square brackets in the component type. MultiOpt default values are displayed to the user when the dialog opens. If none of the options was selected then it will save a null value. In addition, the MultiOpt component allows the following constraints:
`optional` – If the field is optional or required (required by default).
`display` – Define the component display look. The supported displays are `blockCheckbox`, `blockList` and `inlineList`. MultiOpt component default display is `blockCheckbox`.

Like all other components, also MultiOpt component will check if the user input matches the provided constraints and provide an error message otherwise. An example of an error message will look as follows:

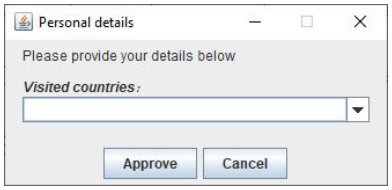


Like other components, MultiOpt uses input validation and uses circular brackets for default values.
Consider the following dialog with default values, optional constraint and different display values.

```
'Personal details' Single(
  'Please provide your details
   below')
countries 'Visited countries:'
  MultiOpt['Chile|Peru
    |Germany|Sweden']{
  optional
  blockCheckbox
  }
```

(1)

```
'Personal details' Single(
  'Please provide your details
   below')
countries 'Visited countries:'
  MultiOpt['Chile|Peru
    |Germany|Sweden']{
  optional
  blockList
  }
```



(2)

```
'Personal details' Single(
  'Please provide your details
   below')
countries 'Visited countries:'
  MultiOpt['Chile|Peru
    |Germany|Sweden']
    (Chile|Germany|Peru){
  optional
  inlineList
  }
```

(3)

As we can see, in all examples we use the optional constraint, which means that the user is not required to select a value. In the first example (image 1) we can see the default display, which is `blockCheckbox`. In this display, the prompt is located above the checkboxes that align vertically. In the second example (image 2) we use the `blockList` display. In this display, the prompt is located above a list. In third example (image 3) we use the `inlineList` display and define some default values. In this display the prompt is located at the side of a list.

1.1.9 Slider Component

The Slider is a base component. An example dialog containing just a Slider component would look as follows:

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']
```

The Slider component provides the user a lever control to select a value. When the user click on the slider or moves it, a tooltip that shows the current value is appears. The minimum and maximum slider ranges must be provided by using square brackets in the component type. Slider default

values are displayed to the user when the dialog opens. If a default value did not set then the slider default value will be set to the minimum value. In addition, the Slider component allows the following constraints:
`majorTicks` – Set the spacing for the major ticks. If not provided then the major ticks will not be displayed. An example is provided below.
`minorTicks` – Set the spacing for the minor ticks. If not provided then the minor ticks will not be displayed. An example is provided below.
`display` – Define the component display look. The supported displays are `inline` and `block`. Slider component default display is `block`.

Defaults value can be provided by using circular brackets after the component type as following:

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']('5')
```

Consider the following dialogs with different constraint values.

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']{
  majorTicks=3
  }
```

(1)

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']{
  majorTicks=3
  minorTicks=1
  }
```

(2)

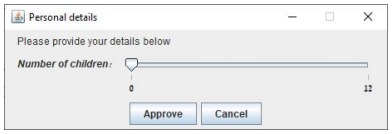
As we can see, in the first example (image 1) we are using `majorTicks` constraints. This constraint adds to the slider ticks with values. In the second example (image 2) we add also the `minorTicks` constraints which add to the slider shorter lines without any values.

Consider the following dialogs with different display values.

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']{
  block
  }
```

(1)

```
'Personal details' Single(
  'Please provide your details
   below')
children 'Number of children:'
  Slider['0,12']{
  inline
  }
```



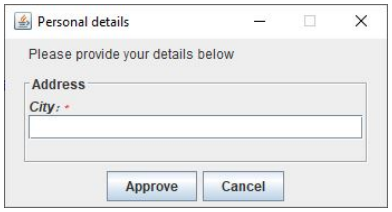
(2)

As we can see, in the first example (image 1) we can see the default display, which is `block`. In this display, the prompt is located above the slider. In the second example (image 2) we use the `inline` display. In this display, the prompt is located at the side of the slider.

1.1.10 Group Component

The Group is a composite component. It can contain only base components. The simplest example for dialog containing a Group component would look as follows:

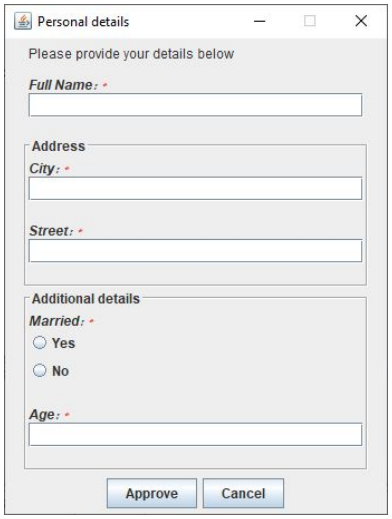
```
'Personal details' Single(
  'Please provide your details
   below')
'Address' Group{
  city 'City:' String
  }
```



As we can see, the Group component wraps the base component and give them a title. The group provides a way to have a separation between the components in the same dialog. The Group component does not have any constraints or default values.

Here is a more complex example using the group component:

```
'Personal details' Single(
  'Please provide your details
   below')
name 'Full Name:' String
'Address' Group{
  city 'City:' String
  street 'Street:' String
  }
'Additional details' Group{
  married 'Married:' Boolean
  age 'Age:' Integer
  }
```

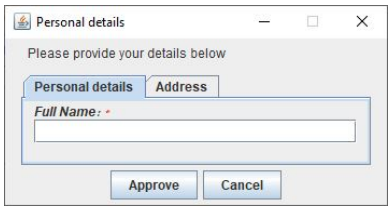


In this example, we can see that we can define multiple groups. Also, we can see that some components inside of the groups and some of them are not. Each group contains its components and the components that outside of the groups will be on the top level of the dialog.

1.1.11 Tab Component

The Tab is a composite component. It can contain base and group components and cannot contain other Tab components. The simplest example for dialog containing a Tab component would look as follows:

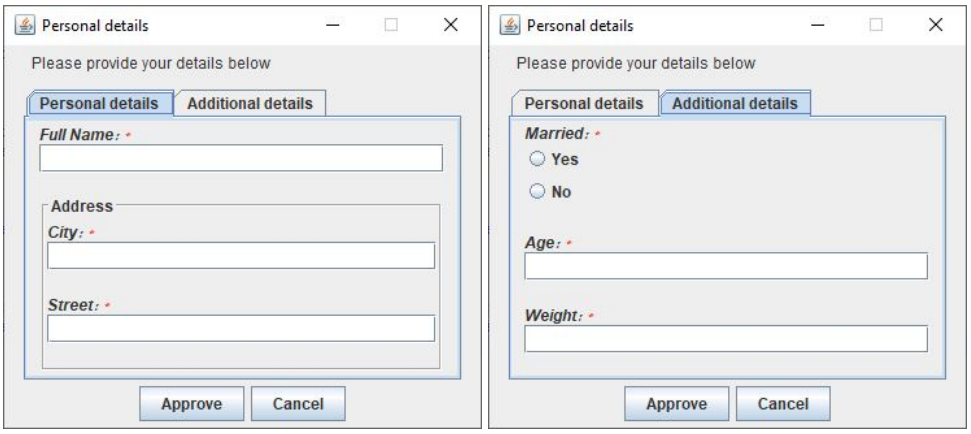
```
'Personal details' Single(
  'Please provide your details
   below')
'Personal details' Tab{
  name 'Full Name:' String
  }
'Address' Tab{
  city 'City:' String
  }
```



As we can see from the example above, the Tab component can contain two base components. It separates the components into different tabs that the user can see when he presses on a specific tab. The Tab component must contain at least two tabs. The tab component does not have any constraints or default values.
Here is a more complex example using the Tab component:

```
'Personal details' Single('Please provide your details below')
'Personal details' Tab{
```

```
  name 'Full Name:' String
  'Address' Group{
    city 'City:' String
    street 'Street:' String
    }
  }
'Additional details' Tab{
  married 'Married:' Boolean
  age 'Age:' Integer
  weight 'Weight:' Decimal
  }
```



In this example, we can see that we define two tabs. Each tab contains some components. In addition, the example illustrates how to generate a tab that contains a Group component.

1.2 Dialogs

In IQL there are three types of dialogs: Single, Pages and Tabular. The dialog structure looks like this:

```
'<title>'  <type> ('<description>'){<constraints>}
```

for example

```
'Account details' Single('Insert Your details below')
```

All dialog types accept input from the user. After the user fulfills all the dialog requirements and presses the "Approve" button, the dialog will close and return as a result a `List<Map<String, String>>`. Each item in the list represents a page in the case of Pages dialog or a row in the case of Tabular dialog. For the Single dialog the list will be in a length of one. In the map, each key is the component `name` and each value is the input that the user inserts in that specific field. Here is a simple example using Java to get the user data:

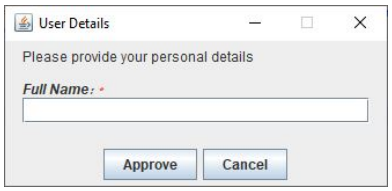
```
List<Map<String, String>> results =
  IQL.run('User Details' Single('Please provide your
   personal details')
    name 'Full Name:' String)
```

In the following section, we explain in details how to use each of the IQL dialogs and provide examples.

1.2.1 Single Dialog

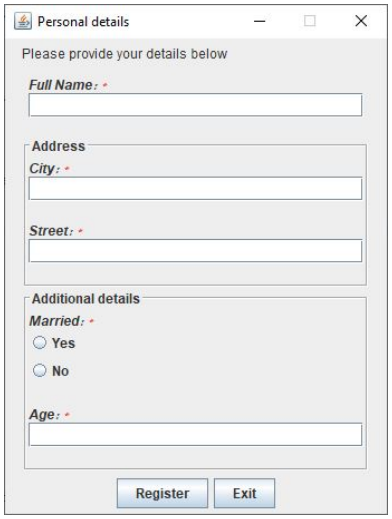
The Single dialog is the simplest dialog. It's a single form that contains all the components. An example of a Single dialog that contains only String component would look as follows:

```
'User Details' Single(
  'Please provide your personal
   details')
name 'Full Name:' String
```



As we can see the Single dialog has two buttons: "Approve" and "Cancel". The name of the buttons can be changed by using constraints. When pressing on the "Approve" button the dialog will check all the inputs and will provide an error message if needed. While pressing on the "Cancel" or the close window button ('x') the dialog will close and return an EmptyEntriesMap object which extends a `List<Map<String, String>>` with the keys equal to the component `name` and the values equals to null. We return the EmptyEntriesMap object to make it easier for the developers to check if the user clicks the "Approve" button or just close the form. The Single dialog supports all the component types and their displays. In addition, it supports the following constraints:
`approve` – The text to display in the "Approve" button.
`cancel` - The text to display in the "Cancel" button.
Here is a more complex example using the various constraints:

```
'Personal details' Single(
  'Please provide your details
   below'){
    approve='Register'
    cancel='Exit'
  }
name 'Full Name:' String
'Address' Group{
  city 'City:' String
  street 'Street:' String
  }
'Additional details' Group{
  married 'Married:' Boolean
  age 'Age:' Integer
  }
```

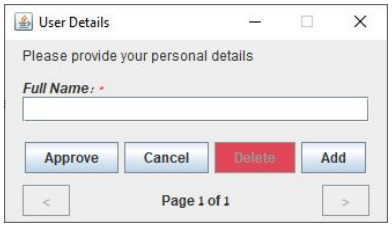


In this example, we can see that we define a Single dialog that contains two groups named: "Address" and "Additional details". We can see that by using the `approve` and the `cancel` constraints we were able to modify the default label for the "Approve" and "Cancel" buttons.

1.2.2 Pages Dialog

Pages dialog is like the Single dialog but it can contain multiple pages of the same dialog. An example of Pages dialog that contains only String component would look as follows:

```
'User Details' Pages(
  'Please provide your personal
   details')
name 'Full Name:' String
```



As we can see the Pages dialog have six buttons: "Approve", "Cancel", "Add", "Delete", "<" and ">". The name of the "Approve" and "Cancel" buttons can be changed by using constraints. When pressing on the "Approve" button the dialog will check all the inputs and will provide an error message if needed. While pressing on the "Cancel" or the close window button ('x') the dialog will close and return a list of hashmap in the same way the Single dialog does. The Pages dialog supports all the component types and their displays. In addition, it support the following constraints:
`approve` – The text to display in the "Approve" button.
`cancel` - The text to display in the "Cancel" button.
`min` – The minimum number of pages that the user requires to fill.

`max` - The maximum number of pages that the user requires to fill.

Here is a more complex example using the various constraints:

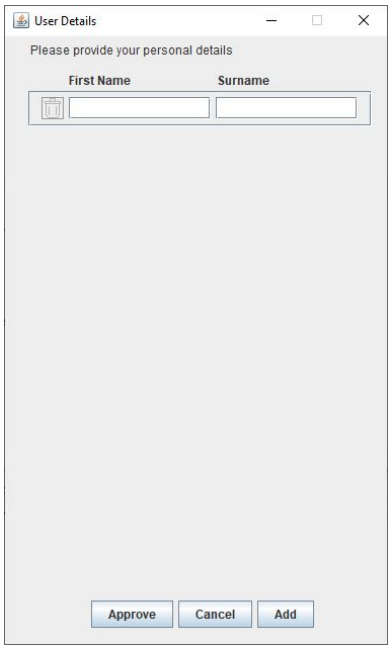
```
'Personal details' Pages(
  'Please provide your details
   below'){
    min=2
    max=8
    approve='Register'
    cancel='Exit'
  }
name 'Full Name:' String
'Address' Group{
  city 'City:' String
  street 'Street:' String
  }
'Additional details' Group{
  married 'Married:' Boolean
  age 'Age:' Integer
  }
```

In this example, we can see that we define a Pages dialog that contains two groups. We can see that by using the `approve` and the `cancel` constraints we were able to modify the default label for the "Approve" and "Cancel" buttons. The "Add" button is used to add a new page that the user can fill. If the page that the user fills has errors so this button will not add a new page and will display to the user the relevant errors. The two arrows buttons ("<" and ">") are used to navigate between the pages that the user filled. If the user is at the first page then the "<" button will become unclickable. In the same way, if the user is on the last page then the ">" button becomes unclickable. The "Delete" button removes the current page. If there is only one page left then this button becomes unclickable. In addition to that there is a pages index at the bottom of the dialog. This index shows the current page and the total pages. The `max` constraint defines the maximum number of pages that the user can add. If the user get to the maximum number of pages then the "Add" button becomes unclickable. The `min` constraint defines the minimum number of pages that the user must fill. In the case that, the user presses the "Register" button without fill the necessary amount of pages, a popup that contains an error message will display. Here is an example of the error message:

1.2.3 Tabular Dialog

The Tabular dialog shows all the fields in a tabular way. It can contain multiple rows that each row contains all the components. An example of a Tabular dialog that contains two String components would look as follows:

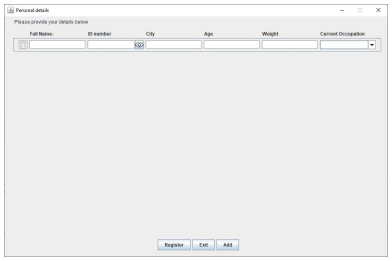
```
'User Details' Tabular(
  'Please provide your personal
   details')
first 'First Name' String
surname 'Surname' String
```



As we can see the Tabular dialog have three buttons at the bottom: "Approve", "Cancel", "Add" and one button to the left of each row. The name of the "Approve" and "Cancel" buttons can be changed by using constraints. When pressing on the "Approve" button the dialog will check all the inputs and will provide an error message if needed. While pressing on the "Cancel" or the close window button ('x') the dialog will close and return a list of hashmap in the same way the Single dialog does. The Tabular dialog does not support Tab, Group, Slider and TextArea components. In addition, it has its default display for each component that cannot be changed. The default display for String, Integer and Decimal components is `block` and for Boolean, SingleOpt and MultiOpt is `blockList`. In addition, Tabular dialog supports the following constraints:
`approve` – The text to display in the "Approve" button.
`cancel` - The text to display in the "Cancel" button.
`min` – The minimum number of rows that the user requires to fill.
`max` - The maximum number of rows that the user requires to fill.

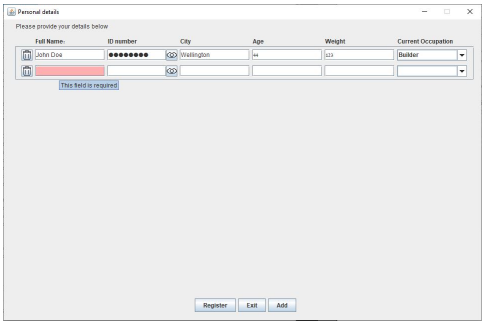
Here is a more complex example using the various constraints:

```
'Personal details' Tabular(
  'Please provide your details below'){
    min=2
    max=30
    approve='Register'
    cancel='Exit'
    }
name 'Full Name' String{optional}
id 'ID number' Password{optional}
city 'City' String{optional}
age 'Age' Integer{optional}
weight 'Weight' Decimal{optional}
occupation 'Current Occupation' SingleOpt[
  'Builder|Farmer|Baker|No occupation']
  {optional}
```

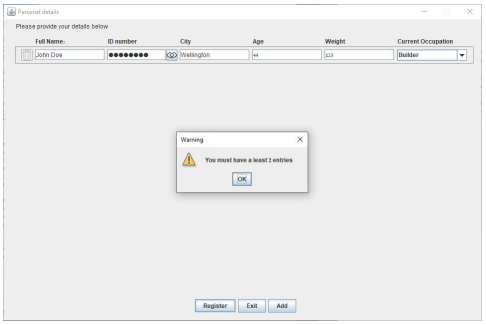


This example shows a Tabular that contain various fields. We can see that by using the `approve` and the `cancel` constraints we were able to modify the default label for the "Approve" and "Cancel" buttons. The "Add" button is use to add a new row. If one of the rows that the user filled has errors the button will not add a new row and will display to the user the fields that have errors in pink color. In order to see the error information, the user needs to hover with the mouse over the pink field (image 1). By pressing the button with the garbage icon the user can delete a specific row. If there is only one row then this button become unclickable. The `max` constraint defines the maximum number of rows that the user can have. If the user gets

to the maximum number of rows then the "Add" button become unclickable. The `min` constraint defines the minimum number of rows that the user must fill. If the user press the "Register" button without fill the necessary amount of rows, a popup that contains an error message will be displayed (image 2).



(1)



(2)