

# API for ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints

Leave Authors Anonymous  
for Submission

## ABSTRACT

We provide an application programming interface (API) to formalize GUI layouts that can be solved at interactive rates by our solver and also enable users to efficiently create layouts. Our API mainly contains the classes for *ORCLayout* (Abstract), *ORCWidget*, *Pivot*, *ORCColumn*, *ORCRow*, *Flow*, *HorizontalFlow*, *VerticalFlow*, *FlowAroundFix*. We only include public classes, methods, and parameters in this document.

In addition, we also provide some examples to show how to specify layouts with API.

## CLASS ORCLAYOUT [ABSTRACT CLASS]

An abstract class to define all the sub-layouts

*@param name:* the name of the sub-layout

*@param parent:* the parent sub-layout of this sub-layout

*@param weight:* the weight (priority) of the sub-layout (default: 1)

### set\_weight(float weight)

Sets the weight of the sub-layout which represents the priority of the sub-layout.

*@param weight:* the weight (priority) of the widget

### constraint\_spec() [Abstract Method]

Constructs the constraint system including optimization variables, constraints, and objective functions for this layout.

### solve()

Solves the constraint systems for sub-layouts recursively. We perform the branch-and-bound algorithm through recursive calls, so in the end, we only need to call *root.solve()* to perform the algorithm, where *root* is the root of the branch-and-bound tree.

### get\_best()

Gets the best solution for the layout after *solve()* is called.

*@return best\_leaf:* the leaf node containing the best solution in the branch-and-bound tree

*@return best\_leaf\_result:* the best solution for the layout

*@return best\_leaf\_loss:* the loss value of the best solution for the layout

## CLASS ORCWIDGET (EXTENDS ORCLAYOUT)

Defines a widget or a sub-layout if the sub-layout only contains one widget.

*@param name:* the name of the sub-layout

*@param width\_and\_height:* the min/pref/max width and height of the widget

*@param parent:* the parent sub-layout of this sub-layout

*@param optional:* True if the widget is optional in the layout

### set\_optional()

Sets the widget to be optional, which can be removed if there is not enough space for it.

## CLASS PIVOT (EXTENDS ORCLAYOUT)

Specifies an OR constraint that may flip the orientation of a layout from horizontal to vertical, or vice versa. Meanwhile, it also switches between horizontal and vertical flows in the sub-layouts.

*@param name:* the name of the sub-layout

*@param parent:* the parent sub-layout of this sub-layout

*@param window\_width:* window width (default: None)

*@param window\_height:* window height (default: None)

### set\_layout(column\_or\_row)

If the input is column, then column has higher priority than row when the losses are the same.

*@param column\_or\_row:* the layout Pivot wants to switch

## CLASS ORCCOLUMN (EXTENDS ORCLAYOUT)

Constructs a *Column* layout dividing the whole window into different areas as a column. A *Column* is a rectangular vertical arrangement of sub-layouts.

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param window\_width: window width (default: None)

@param window\_height: window height (default: None)

#### define\_sublayouts(sublayouts)

Defines the sub-layouts contained in the column.

@param sublayouts: the set of sub-layouts contained in the column

### CLASS ORCROW (EXTENDS ORCLAYOUT)

Constructs a Row layout dividing the whole window into different areas as a row. A Row is a rectangular horizontal arrangement of sub-layouts.

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param window\_width: window width (default: None)

@param window\_height: window height (default: None)

#### define\_sublayouts(sublayouts)

Defines the sub-layouts contained in the row.

@param sublayouts: the set of sub-layouts contained in the row

### CLASS FLOW [ABSTRACT CLASS] (EXTENDS ORCLAYOUT)

An abstract class to Construct a flow sub-layout.

@param name: the name of the sub-layout

@param widget\_list: widgets to flow in this sub-layout

@param parent: the parent sub-layout of this sub-layout

@param balanced: True if we want the flow to be balanced.

@param fixed\_boundary: True if all the boundaries are fixed for this flow sub-layout. (default: False)

@param boundary\_distance: when fixed\_boundary is True, if the flow is horizontal, then the distance is the distance between top boundary and bottom boundary, and if the flow is vertical, then the distance is the distance between left boundary and right boundary. (default: None)

#### connect\_to\_flow(other\_flow)

Sets the connected flow the current flow connects to.

@param other\_flow: the flow which the current flow connects to

#### solve() [Method overriding]

Solves the constraint systems for sub-layouts recursively. We perform the branch-and-bound algorithm through recursive

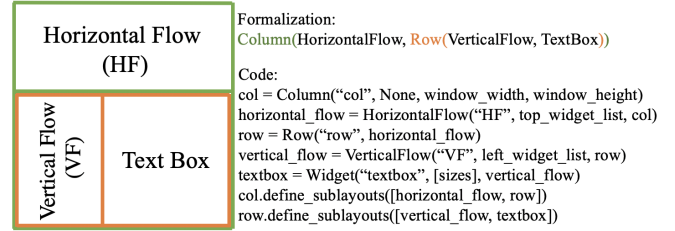
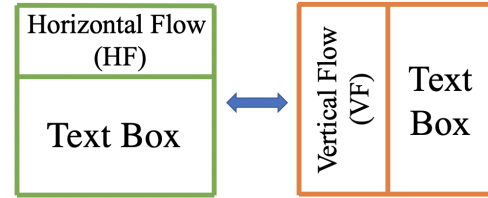


Figure 1. The formalization and corresponding code for a Column layout which divides the window into a horizontal flow and a Row layout containing a vertical flow and a text box.



Formalization:

`Pivot(Column(HorizontalFlow, TextBox))`

Code:

```
pivot = Pivot("p", None, window_width, window_height)
col = Column("col", pivot)
horizontal_flow = HorizontalFlow("HF", top_widget_list, col)
textbox = Widget("textbox", [sizes], horizontal_flow)
pivot.set_layout(column)
col.define_sublayouts([horizontal_flow, textbox])
```

Figure 2. The formalization and corresponding code for a Pivot layout which can switch between top toolbar and left toolbar.

calls. We override the method in the class *ORCLayout* to allow creating branches in the branch-and-bound tree.

### CLASS HORIZONTALFLOW (EXTENDS FLOW)

Defines a horizontal flow sub-layout which uses heuristics in our ORCSolver to solve the horizontal flow sub-layout.

### CLASS VERTICALFLOW (EXTENDS FLOW)

Defines a vertical flow sub-layout which uses heuristics in our ORCSolver to solve the vertical flow sub-layout.

### CLASS FLOWAROUNDFIX (EXTENDS FLOW)

Defines a flow sub-layout where all the widgets flows around a fixed area. It uses heuristics in our ORCSolver to solve the flow sub-layout.

@param name: the name of the sub-layout

@param widget\_list: widgets to flow in this sub-layout

@param parent: the parent sub-layout of this sub-layout

## EXAMPLES

The layout in Figure 1 can be formalized as *Column(HorizontalFlow, Row(VerticalFlow, TextBox))*. The layout uses a *Column* layout dividing the window into a horizontal flow and the lower part which is a *Row* layout containing a vertical flow and a text box.

The layout in Figure 2 can be formalized as *Pivot(Column(HorizontalFlow, TextBox))*. The *Pivot* pattern can turn horizontal arrangements into vertical ones.

The layout uses a *Column* with a horizontal flow, *e.g.*, of toolbar widgets. The flow is normally placed above the text box, *i.e.*, toolbar at the top. Because of the *Pivot*, the layout solver can turn the *Column* into a *Row* and at the same time, turn the horizontal flow into a vertical one, *i.e.*, toolbar on the left. Given this layout formalization, the solver can break the layout down into two alternatives, *i.e.*, *Column(HorizontalFlow, TextBox)* OR *Row(VerticalFlow, TextBox)*.