



vCortex User Manual

I4Cortex Visual Component

Table of Contents

1. Introduction:	3
2. Prerequisites:	3
3. Setup and Usage:	3
4. Trial mode:	3
5. Playground Project:	4
6. Versioning System	4
7. Components:	5
7.1. Color Picker:	5
Properties:	5
Events:	5
7.2. Cascader:	6
Properties:	6
Events:	7
Future upgrades:	7
7.3. Date Time Range Picker:	8
Properties:	8
Events:	8
7.4. Time Picker:	9
Properties:	9
Events:	10
Future upgrades:	10
7.5. Range Slider:	10
Properties:	10
Events:	10

7.6. Odometer:	11
Properties:	11
Events:	11
Tips and Tricks:	11
7.7. Rich Text Editor:	12
Properties:	13
Events:	13
Future upgrades:	13
7.8. Calendar:	14
Properties:	14
Events:	15
Event return Example:	15
7.9. ECharts:	16
Properties:	16
Events:	17
Functions:	17
Example:	18
Future upgrades:	18
7.10. Zoom & Pan:	19
Properties:	19
Events:	19
Functions:	20
Tips and Tricks:	20
7.11. Flow Diagram:	21
Properties:	21
Events:	26
Functions:	26
8. System Functions:	28
System.i4cortex.gui.getViewObjects	28
System.i4cortex.gui.getViewSize	28
System.i4cortex.gui.getViewTree	29
9. Support:	30

1. Introduction:

The vCortex module is collection of perspective components to fill the gap of missing functionality in Perspective application. The module will not be limited to the current number of components and will be grown based on community requests.

All components unified by global styling in the module to make all of them look and feel the same way.

Current Components list:

- Rich Text Editor
- Color Picker
- Date Time Range Picker
- Time Picker
- ECharts
- Range Slider
- Odometer
- Cascader
- Calendar
- Zoom and Pan Viewer
- Node Base Editor

Future Planned Components:

- Timeline
- Excel Editor
- Circular Slider

2. Prerequisites:

- Ignition v8.1.20 Upward
- Perspective Module

3. Setup and Usage:

After installing the module, all components will be available to use under i4Corex section in the Perspective Components Panel. For more complex components we added some variants to help user start quickly.

We try to expose most of the styles of components in props section so there will be no need for debugging to find out related style classes.

You can also access this help file from the Designer toolbar by clicking on “i4Cortex Help Center” icon.



4. Trial mode:

After installing the module, it can be used infinitely with all its functionality only for testing and development during 2-hour Ignition Trial Mode. When the 2-hour trial mode is finished, then it will need to be reset.

5. Playground Project:

After installing the module, a demo project called **vCortex Playground** will be added to your gateway. As the name suggests, this project serves as a guide to help you effectively use the vCortex Module and understand how to apply its components and functions across different applications. You are free to modify the demo; however, please note that it will be overwritten whenever a new version is released. To preserve your changes or integrate the demo into your own projects, we recommend either project inheritance or copy the necessary resources into your target project.

6. Versioning System

The vCortex versioning system is as follows:

MAJOR.MINOR.PATCH

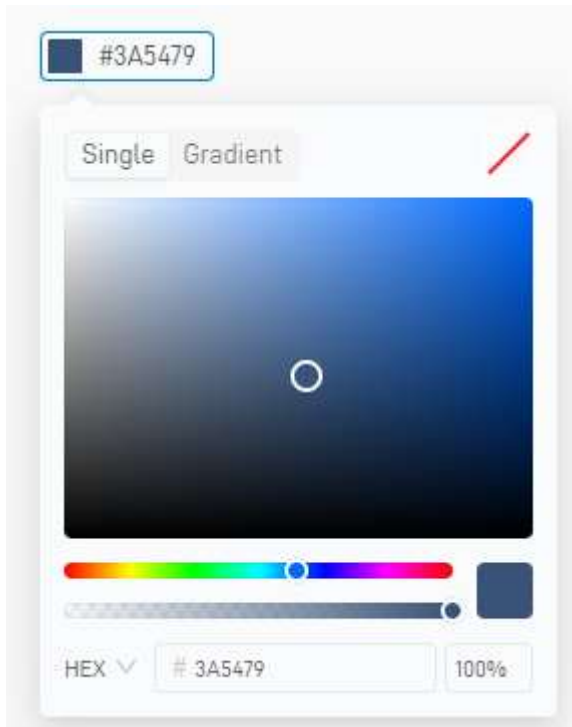
1. **MAJOR** → Indicates the Ignition compatibility
 - 0 → Ignition 8.1.x
 - 1 → Ignition 8.3.x
2. **MINOR** → Represents the number of included components in the module
3. **PATCH** → Identifies incremental updates, fixes, or optimizations

As vCortex supports multiple Ignition platforms (8.1, 8.3), this versioning system makes it clearer and more consistent across releases. This helps users easily identify which Ignition version a release targets and how many components are included in the module.

7. Components:

7.1. Color Picker:

This component is based on Ant Design 5.0 UI framework. As its name implies, by clicking on color icon, any color from a graphical popup color picker can be selected. The 3 famous color formats are with alpha channel is supported: HEX, RGBA, HSBA.



Properties:

- **value:** Selected color pick by user. Also, can pass color in hex format to the component as initial value. The CSS variables and color names like "red" does not supported but you can pass color in #xxxxx, rgba(), hsl() and hsv() format.
- **enabled:** Whether the components disable or enable. False is disable.
- **size:** Specify the color icon size. The support value is small, middle, large.
- **showText:** Whether the selected color code will appear next to color icon.
- **theme:** An object of styling properties which is generated from <https://ant.design/theme-editor>. You can copy and paste your desired style into this property or simply manipulated available options or adding them one by one. The traditional CSS styling does not work for this component, but you can use var() CSS color to make your theme dynamic base on selected Perspective Theme.

Events:

- **onActionPerformed:** Triggered when user selects a color.
event.hex: selected color code in hex string format
event.hsb: selected color code in hsb string format.
event.rgb: selected color code in rgb string format.

Event return Example:

```

{
  "hex": "#2167c800",
  "hsb": "hsba(215, 83%, 78%, 0)",
  "rgb": "rgba(33,103,200,0)"
}

```

Future upgrades:

- Adding color pallet
- Gradient color

7.2. Cascader:

- This component is based on Ant Design 5.0 UI framework. The main function of this component is when you need to select from a set of associated data set or when selecting from a large data set, with multi-stage classifications separated for easy selection. Such as province/city/district, company level, things classification. The component can also selection multiple items from different levels. The search option is also available at any time to filter the tree and fast selection.



Properties:

- **value**: An array of selected items from the tree. Each element is another array which represents the sequence of selection in order. This prop is not bidirectional and cannot accept any initial value for the moment.
- **selectedOptions**: An array of selected items which is rendered like items props.
- **multiple**: If True user can select multiple items from different levels by clicking on the check box next to the item.
- **items**: An array of object, each representing an item in the tree. Each item object has following properties:
 - **label**: Text display for this item.
 - **value**: Extra field use has id or data for the item. It can be string or number. For the moment, the object or array does not support.
 - **children**: An array of children of the current item in the tree, each representing an item in the tree.
 - **disableCheckbox**: If True the item can not be selected by the user.

- **placeholder**: Prompt Text to display when no items are selected.
- **enabled**: Whether the components disable or enable. False is disable.
- **size**: Specify the color icon size. The support value is small, middle, large.
- **theme**: An object of styling properties which is generated from <https://ant.design/theme-editor>. You can copy and paste your desired style into this property or simply manipulated available options or adding them one by one. The traditional CSS styling does not work for this component but you can use var() CSS color to make your theme dynamic base on selected Perspective Theme.

Events:

- **onActionPerformed**: Triggered when an item is selected by user.
event.value: An object of selected items from the tree.

Event return example for multiple selection:

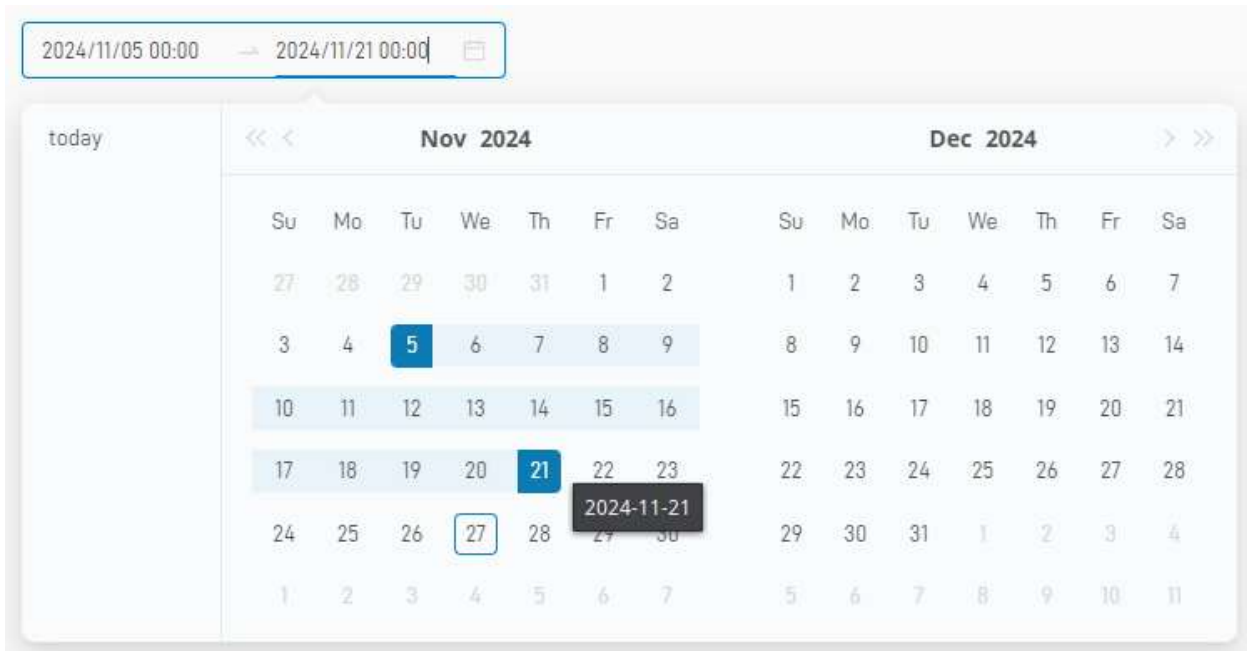
```
[
  [
    "light"
  ],
  [
    "bamboo",
    "little",
    "cards"
  ]
]
```

Future upgrades:

- Make value prop bidirectional.
- Object type support for value property in item object.
- Add placement option in props for specify the position of the popup.
- Support inline view case.

7.3. Date Time Range Picker:

This component is based on Ant Design 5.0 UI framework. User can select a date range from a calendar popup in one-shot. The Today date is highlighted in calendar and user can select Today Date.



Properties:

- **value:** An object which returns user selected start and end range in JAVA date format. Also, can pass date range to update the component for initial values.
- **size:** Specify the date range picker size. The support value is small, middle, large.
- **minDate:** Minimum Date/time as a date object. If null, the option is disable.
- **maxDate:** Maximum Date/time as a date object. If null, the option is disable.
- **format:** Template for formatting date display in UI and formattedValue property. E.g: 'YYYY/MM/DD'
- **formattedValue:** An object which returns user selected start and end range in formatted string supplied by format property.
- **placeholder:** Prompt Text to display when no range dates are selected.
- **locale:** Code for localization of language and formatting.
- **enabled:** Whether the components disable or enable. False is disable.
- **showTime:** True make Date Range picker also selected time as well. In this mode user needs to select date and time for start and click OK and then select end date and time.
- **placement:** The position where the selection box pops up.
- **picker:** Specify the picker mode. Options are date, time, decade, month, year
- **theme:** An object of styling properties which is generated from <https://ant.design/theme-editor>. You can copy and paste your desired style into this property or simply manipulated available options. The traditional CSS styling does not work for this component but you can use var() CSS color to make your theme dynamic base on selected Perspective Theme.

Events:

- **onActionPerformed:** Triggered when a date range is selected by user.
event.start: selected start date in string, formatted by format property.
event.end: selected end date in string, formatted by format property.

Event return Example:

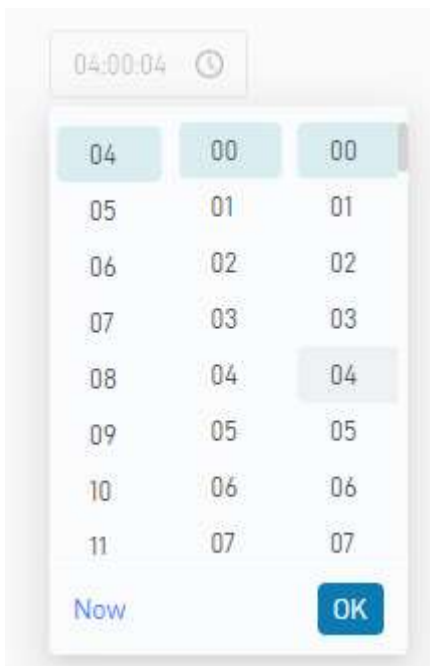
```
{  
  "end": "2025-01-03 06:05",  
  "start": "2024-12-19 03:04"  
}
```

Future upgrades:

- Support locale for Arabic Regions.

7.4. Time Picker:

This component is based on Ant Design 5.0 UI framework. User can select a time from a popup or select current time. For the time range selection use Date Time Range Picker Component.



Properties:

- **value:** returns user selected time in string, formatted by format property. Also, can pass time to update the component for initial value.
- **size:** Specify the time picker size. The support value is small, middle, large.
- **format:** Template for formatting time display in UI. E.g: 'HH:mm:ss'
- **enabled:** Whether the components disable or enable. False is disable.
- **placement:** The position where the selection box pops up.
- **theme:** An object of styling properties which is generated from <https://ant.design/theme-editor>. You can copy and paste your desired style into this property or simply manipulated available options. The traditional CSS styling does not work for this component but you can use var() CSS color to make your theme dynamic base on selected Perspective Theme.

Events:

- **onActionPerformed**: Triggered when user selects a time.
event.value: selected time in string formatted by format property.

Event return Example:

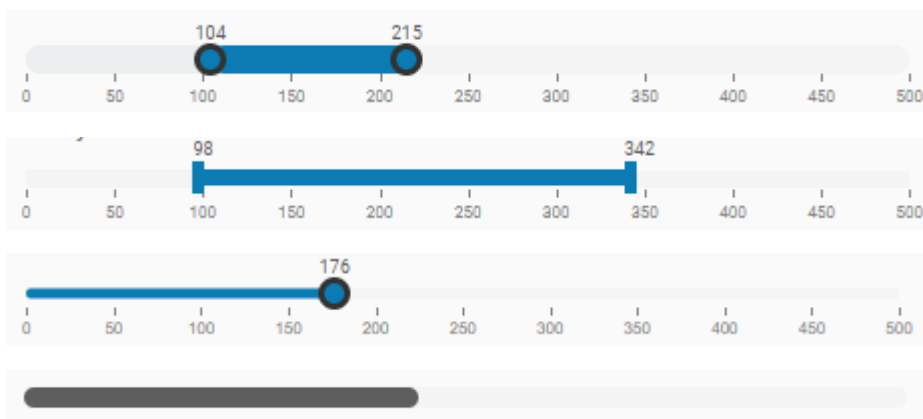
```
{  
  "value": "1:03:30"  
}
```

Future upgrades:

- Add Clear icon on UI to reset component value

7.5. Range Slider:

This component makes it possible to select multiple range for users in slider fashion. There are a lot of variants of this component for quick start or you can easily style it as you want.



Properties:

- **values**: An array of values which return the user selected values Also, can pass value to update the component for initial values.
- **step**: Intervals along track at which value may be set by slider.
- **mode**: define the interaction mode. Value of 1 allow handles to cross each other. Value of 2 will keep the slider from crossing. Value of 3 make the handles push able and keep them a step apart.
- **range**: define the start and end range of the slider.
- **count**: number of ticks on slider.
- **reversed**: reversed the slider direction.
- **enabled**: Whether the components disable or enable. False is disable.
- **styles**: a collection of available styles for the component. Use CSS code to style each part. you can use var() CSS color to make your theme dynamic base on selected Perspective Theme.

Events:

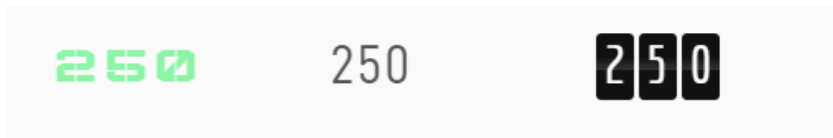
- **onActionPerformed**: Triggered when a range is selected by user.
event.value: selected range in array format. Index 0: start range value, index 1: end range value.

Event return Example:

```
{ "value": [78, 272 ] }
```

7.6. Odometer:

Odometer make transition numbers with ease and smoothly by applying animation. When the value is changed the number starts to count up or down to reach the target value.



Properties:

- **value:** input target value. The component starts transition numbers from previous value to the new value.
- **initValue:** Set the starting number for transition to target value on props.value when the component first loads. This value is used only for initialization; later updates use the props.value.
- **duration:** Specify the total transition animation rate in second.
- **format:** Allows to configure how the digit groups are formatted, and how many digits are shown after the decimal point.

Format - Example

(,ddd).dd - 12,345,678.09

(ddd),dd - 12 345 678,09

d - 12345678

- **type:** style templates to choose from drop box list. There are 6 different styles which also can be modified by user in Perspective stylesheet.

Events: None

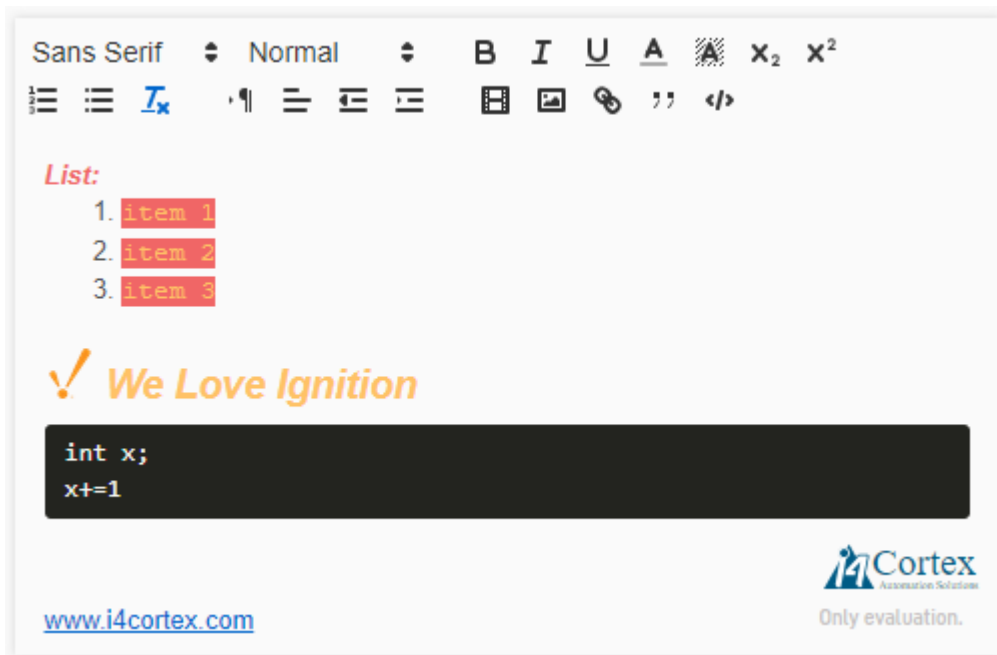
Tips and Tricks:

- To change the odometer size, you can simply apply font-size in the style section.
- For changing digit width, apply text-indent in the style section.
- As the wrapper of component is a flexbox you can apply flexbox CSS code to align the component.
- Use Persistent option on props.value when you don't setup any binding otherwise it revert back to default value.

7.7. Rich Text Editor:

This component is based on quill.js library. The component gives users an interface for editing rich text within web browsers, which presents the user with a "what-you-see-is-what-you-get" (WYSIWYG) editing area. The aim is to reduce the effort for users trying to express their formatting directly in markdown component.

The component also supports read only mode which is perfect for MES application to show users, the instructions or similar documents.



Following format is supported:

Inline

- Background Color - background
- Bold - bold
- Color - color
- Font - font
- Inline Code - code
- Italic - italic
- Link - link
- Size - size
- Strikethrough - strike
- Superscript/Subscript - script
- Underline - underline

Block

- Blockquote - blockquote
- Header - header
- Indent - indent
- List - list
- Text Alignment - align

- Text Direction - direction
- Code Block - code-block

Embeds

- Formula - formula (requires KaTeX)
- Image - image
- Video - video

Properties:

- **editorState**: Return user document content in markup encoding format. It is bidirectional so you can update the content as well. (like database)
You can copy and paste this value directly into Perspective Markdown source property with escapeHtml set to False to show it as well or copy text from word or website and paste them here.
- **placeholder**: Prompt Text to display when there is no content.
- **readOnly**: If True, hide the toolbar and disable editing mode.
- **editorStyle**: style the editor section by adding your CSS code here
- **toolbarStyle**: style the toolbar section by adding your CSS code here
- **toolbarButtonStyle**: style only buttons in toolbar by adding your CSS code here
- **toolbarIconStyle**: style only icons in toolbar by adding your CSS code here

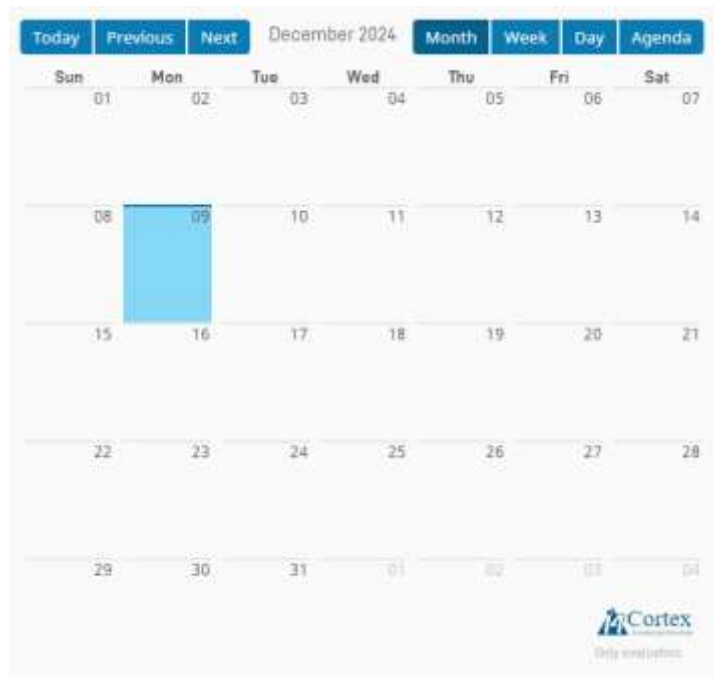
Events: None

Future upgrades:

- Add tooltip for toolbar buttons.
- Add support for table.
- Add support for checkbox.

7.8. Calendar:

This component is based on react-big-calendar library. The component makes it possible for users to select events, resize, move, delete and add events on the calendar. The recurring functionality should be defined in Python Script separately. This makes users open to developing their own version of recurring function. For quick start check out the demo project to see how to use rrule function to generate recurring events.



Properties:

- **events**: An array of event objects which display on the calendar and automatically create new event if addEventByClick is True. In this case the event object created by id: null to indicate that this new event is not save yet in database. After saving it into the database you can update the return id. You can pass your events from database by adding a simple query binding on this property.

Events Object items:

- **id**: Usually use for database primary Key value. When new events are created, the value is null, so you can loop through all events and if id is equal to null, you need to insert it into the database and update the value with new given id from database.
- **title**: Label or title for events that are displayed on the Calendar.
- **allDay**: A Boolean value that if set to "true" will make the event ignore any time portion of the "start" and "end" properties and show event in allDay area in Week, Day, Agenda mode. This props has no effect in Month mode
- **start**: A Java format date for starting of the event.
- **end**: A Java format date for ending of the event.
- **enabled**: A boolean value that if sent to "false" will make the event render in disabled mode in UI. The style of disabled mode is defined in styles.eventDisabled.
- **text**: A String value that use to store note to the events.
- **style**: Use for styling the event object on the Calendar. You can define the default style in the styles.newEvent property for addEventByClick: True. If you create the event by python code (like Perspective button) and leave the style empty the component will use the style definition on styles.event.

- **currentDate:** The current date displayed by Calendar. It is bidirectional property so you can pass date to update the Calendar date as well. Also when user click on event this value update to the start date of the selected event.
- **addEventByClick:** A Boolean value that if set to "true" will generate new event in events property each time user clicks or drag on Calendar.
- **locale:** Passing a language code will change the header/footer title, grid view headers and list item headers to the selected language and date format.
- **toolbarLabel:** Using to setup label for each button at the toolbar. We do not support Perspective translation for the moment so for multi-language applications, you can simply bind your label to an expression like this: `translate("Today", {session.props.locale})`
- **view:** A dropdown list for selecting the Calendar view mode.
- **resizable:** If True, allows for resizing of events on the Calendar. The user action will also update the related event in the events list.
- **draggable:** If True, allows for dragging of events on the Calendar. The user action will also update the related event in the events list.
- **styles:** Collection of all available styles for the component. You can define your own custom style for each part of the component. You can also benefit from defined CSS variables by passing them by `var()` function.

Events:

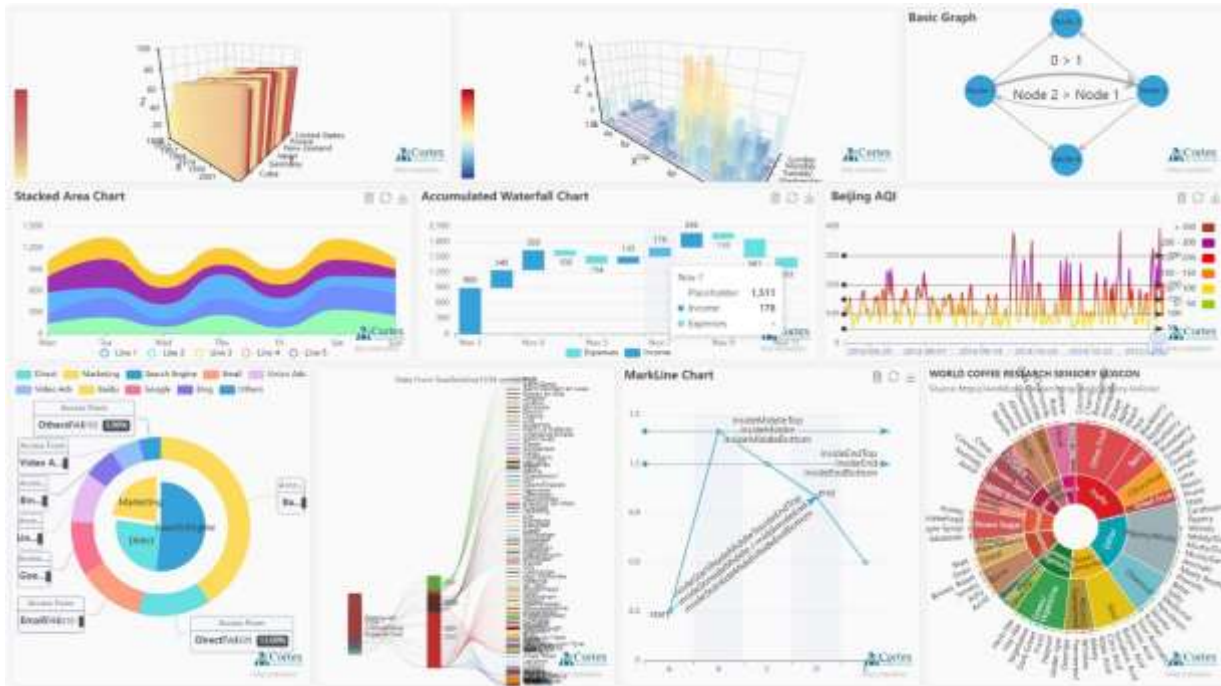
- **moveEvent:** Triggered when an event is moved by user.
- **resizeEvent:** Triggered when an event is resize by user.
- **newEvent:** Triggered when an event is created by user in calendar by clicking and dragging.
- **onSelectEvent:** Triggered when an event is click by user.
- **onDoubleClickEvent:** Triggered when an event is double click by user.
event.value: updated/created/resize Event object plus related index in the event list.

Event return Example:

```
{
  "index": 0,
  "event": {
    "allDay": false,
    "end": "2024-12-14 00:00:00",
    "id": 1,
    "start": "2024-12-12 00:00:00",
    "style": {
      "classes": ""
    },
    "title": "Rework"
  }
}
```

7.9. ECharts:

This component is based on Apache ECharts library. The module adds support for Apache ECharts charting library as a Perspective component. The single Perspective component provides full support for leveraging all Apache ECharts library through a simple flexible property model. For quick start we add more than 20+ variants for different type of charting system. The component also supports 3D charting as well. Due to the data source complexity for some of the charting system, you provide some ready to use JSON template for different charting in our site, so you can just copy and paste them to get quick start.



Properties:

- **setOptions:** The ECharts option and configuration. You can check out following link to get familiar with available options. For fast start, you can use our eCharts variants or ready-to-use charts in our demo project and manipulate them base on your need.
<https://echarts.apache.org/en/option.html>
- **theme:** Accept Perspective dark and light theming. You must bind session.props.theme property with Persistent is set to True. This makes whenever user change Perspective Theme the chart update its color to match with the UI.
This prop also can accept Echart custom theme object. The easiest way to configure is go to eCharts theme maker tools and after making your own theme simply press download button and copy the JSON version and paste it into this property.
<https://echarts.apache.org/en/theme-builder.html>

Note: For the moment Echarts doesn't CSS variables colors var().

- **noMerge:** When enabled, the chart will be fully re-rendered each time setOption is changed, instead of merging new options with the existing configuration. Use this when your data structure changes significantly or you want to reset the chart state on update.
- **renderer:** Option to select between SVG and Canvas.
Generally, Canvas is more suitable for charts with a large number of elements (heat map, large-scale line or scatter plot in geo or parallel coordinates, etc.), and with visual effect. However, SVG has an

important advantage: It has less memory usage (which is important for mobile devices) and won't be blurry when zooming in.

The choice of renderer can be based on a combination of hardware and software environment, data volume and functional requirements.

- In scenarios where the hardware and software environment is good and the amount of data is not too large, both renderers will work and there is not much need to agonize over them.
- In scenarios where the environment is poor and performance issues arise that require optimization, experimentation can be used to determine which renderer to use.
- In situations where many instances of ECharts must be created and the browser is prone to crashing (probably because the number of Canvas is causing the memory footprint to exceed the phone's capacity), the SVG renderer can be used to make improvements. Roughly speaking, the SVG renderer may work better if the chart is running on a low-end Android.
- For larger amounts of data (>1k is an experience value), canvas renderer is always recommended.
- Note: Currently, some special effects still relies on Canvas: e.g. trail effect, heatmap with blending effect, etc.
- **lazyUpdate**: When enabled, the chart will not update immediately. The chart will be updated in the next animation frame. This will help to improve performance in the case you have frequently changes on data.
- **useDirtyRect**: When enabled with the Canvas renderer, the dirty rectangle rendering technique detects and updates only the parts of the view that have changed, rather than any changes causing a complete redraw of the canvas. This doesn't work with SVG renderer. In some cases, this causes visual glitches.

Events:

click: Triggered when user clicks the data elements on chart.

dblClick: Triggered when user double clicks the data elements on chart.

mousemove: Triggered when user move mouse over the data elements on chart.

legendSelectChanged: Triggered when user click on the chart legend.

Event return Example:

```
{  
  "name": "Income"  
}
```

Functions:

- **runJs(funBody, funcArgs)**: Allows user run Javascript code in client side within Perspective Session. When the JavaScript function is executed, the `chart` context inside the function will refer to the chart instance. Users can call all available API on eCharts object in Javascript.
<https://echarts.apache.org/en/api.html#echartsInstance>
 - **Parameters:**
 - **funcArgs**: *Optional*. Dictionary of arguments to use when calling the function. User can access keys value in JavaScript.
 - **funBody**: String of JavaScript code to run. There is no need to define function names and args here. The function runs in blocking mode.

Example:

This example runs on a custom property on eChart components, first use Tag History Binding and then pass it to Script Transform:

```
def transform(self, value, quality, timestamp):
    data = []
    for row in range(value.getRowCount()):
        data.append([system.date.toMillis(value.getValueAt(row, 0)),

    funcArgs = {"data": data};
    funcBody = '''
        con = chart.getOption()
        con.series[0].data = data
        chart.setOption(con, notMerge = true)
    '''

    self.getSibling("EChart").runJs(funcBody , funcArgs)
```

This example shows how to push data directly into the eChart object in memory instead of passing it to the Perspective Component props. This is very useful to prevent the Designer from crashing, in case you have millions of data to pass.

Future upgrades:

- Support auto detect of Perspective changing theme so there will be no need to bind session.props.theme to the theme property.
- Support CSS variables var().
- Support Locale.

7.10. Zoom & Pan:

This component allows for zooming and panning of any view inside it. The component also auto fits the view based on the parent container. So, it means even inside an embedded view your view will be fitted correctly. You can also search for objects inside your view and fly to them.

Properties:

- **view:** Path of the embedded view to display for zoom and panning.
- **params:** Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view.
- **Interaction:** Collections of options and functionality for user interactions with the Zoom and Pan Component.
 - **minZoom:** Minimum zoom level that user can apply in runtime.
 - **maxZoom:** Maximum zoom level that user can apply in runtime.
 - **zoomOnPinch:** Controls if the view should zoom by pinching on a touch screen.
 - **zoomOnDoubleClick:** Controls if the view should zoom by double-clicking somewhere on the container.
 - **zoomOnScroll:** Controls if the view should zoom by scrolling inside the container.
 - **panOnDrag:** Enabling this allows users to pan the view by clicking and dragging.
 - **panOnScroll:** Controls if the view should pan by scrolling inside the container.
 - **panOnScrollSpeed:** Controls how fast view should be panned on scroll.
 - **disableDragHTMLtag:** An array of HTML tags to control on which type of HTML tag the dragging(panning) should be disabled. It is useful if your components must use dragging functionality which conflicts with zoom and pan dragging like text input.
 - **panOnScrollMode:** Used to limit the direction of panning when panOnScroll is enabled. The "free" option allows panning in any direction.
- **controls:** Show/hide the controls button on the UI.
 - **show:** Show/hide all the buttons at once.
 - **zoom:** Show/hide the zoom buttons.
 - **fitView:** Show/hide the fit view button.
 - **duration:** Specify the transition animation time for the fit view function.
 - **fitViewPadding:** Adds extra space around when the view is automatically zoomed and centered. The value is a scale factor based on the total size of the graph (e.g. 0.2 adds about 20% padding). Use 0 for no padding.
- **viewSize:** return the actual size of the view that user pass in props.view. In a case where your view is huge (like more than 500 embedded view), the loading time of view will take time and sometimes the autofit functionality may missing. In this case user can add change script on this props and call self.autoFit().
- **fitView:** Toggling this props will autofit the view.

Events:

- **currentZoom:** Triggered when zoom level of the view is changed. Useful to implement the clutter/decluttering effect on the view by sending the zoom level to view objects. The object receive zoom level message and show/hide their details based on this value.
- **onResize:** Triggered when the size of zoom and pan component is changed in runtime. Especially good for creating responsive effect if using this component inside an embedview and not in the primaryView.

Functions:

- **flyTo(x, y, zoom, duration):** Center the Zoom and Pan viewport to x, y coordinate with a smooth pan-zoom animation.
 - **Parameters:**
 - **x:** Numeric value for x axis target location to fly to.
 - **y:** Numeric value for y axis target location to fly to.
 - **zoom:** Numeric value for zoom level on target location.
 - **duration:** Numeric value for transition animation to target location.
 - **Parameters:** Nothing
- **fitView(duration):** Fit the view inside the Zoom and Pan viewport.
 - **Parameters:**
 - **duration:** Numeric value for transition animation to fit the view.
 - **Parameters:** Nothing

Tips and Tricks:

Search Engine:

For searching and finding objects in the view inside the Zoom & Pan viewer, we provide a system function called `system.i4cortex.gui.getViewObjects()` to return all available objects positions in coordinate container. By using this function API and `flyTo()` function you can easily implement your own search functionality.

This can be used for general search by users or fly to alarm position in the view. For more information, please refer to [the System Functions](#) of this manual.

To see how we develop it check out the `vCortexPlayGround` project which is installed with module.

7.11. Flow Diagram:

This component allows for building customizable node-based flow diagrams with editing and interactive diagram. Each node can represent by a perspective view (embedded view) and have a custom number of handles on each side of the node for connections of lines to other nodes in the diagram.

Here are some use cases in Ignition Perspective Applications:

1. Material Track and Trace Genealogy Flow Diagram
2. ESD diagram
3. Energy Flow Diagram
4. Interactive P&ID editor
5. Logic Diagram (CFC, SFC)
6. Recipe Editor
7. Automated Workflow
8. Tracking business goals



Properties:

- **nodes:** An array of objects, each representing a node in the diagram. Each node object has the following properties:
 - **id:** A unique identifier. The component generates a UUID each time a new node is created in designer props. Users can generate their own id if needed like in the case of appending a new node by Python script. In this case, the user should handle the id. Duplicate in id result breaking the functionality.
 - **type:** Option for displaying node. Available options are default, input, output, custom, connector.
 - **Input:** Represent the node as a simple label with border and one output handle at bottom. Users can style it in the node.style object. The node.data.label is used for the node label. The performance of this type is good but graphically it is limited.
 - **output:** Represent the node as a simple label with border and one input handle at bottom. User can style it in the node.style object. The node.data.label is used for node label. The performance of this type is really good but graphically it is limited.





- **custom:** Represent the node by Perspective View. Node.data.params use here to pass the selected view parameters. Any number of handles can be added in the node.data.handles props. The performance of this type is lower than other types as it uses embedded view technology of Ignition Perspective but the graphical power of this node is unlimited and user can create anything.
- **connector:** Represent the node as a 4-ways connector. Each branch of connector can be show/hide in the node.data.handles[x].show props to make 3-ways connector. This node is used to create branch in edges(lines).



- **data:** Use to pass custom data like view, label, params and handles to the node based on node.type.
 - **handles:** An array of objects, each representing a handle for the node. Only apply for custom and connector nodes. Each handle has the following properties:
 - **position:** Specify the position of the handle related to the edge of the node.
 - **offset:** Specify the offset of the handles along the node edge.
 - **id:** An unique identifier. The handle generates a UUID each time a node is created in designer props. User can generate his own id if needed and this should only be unique for each node.
 - **style:** Customize the look and feel of handle by CSS code. Very useful in the case of putting the handle inside the node by using translate, top, bottom CSS code. For more information, check out the connector handles style.
 - **space:** Specify the space between multiple handles on the same edge of the node.
 - **view:** Path of the embedded view to display for the custom node.
 - **params:** Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view.
 - **label:** Display text for input, output and default node types.
 - **style:** CSS code to apply to the embedded view.
- **position:** Determine the position (top-left corner) of the node in the diagram. Moving the node in diagram will update this prop and writing to this prop will update the node in diagram.
- **hidden:** Set to True to hide the node in the diagram.
- **draggable:** Set to False will disable user from dragging the node in the diagram.
- **selected:** Update to True when the user selects the node and sets it to False when another node is selected.
- **style:** CSS code to apply to the Node. When using custom node and applying a view, this updates the default width and height of the view.
- **edges:** An array of objects, each representing an edge (line) in the diagram. Each edge object has the following properties:

- **source**: Id of the source node which edge is coming from. This prop will automatically update when the user creates edges in diagram.
- **target**: Id of the target node which edge is going to. This prop will automatically update when the user creates edges in diagram.
-
- **type**: Option for rendering edge between nodes. Available options are bezier, step, smoothstep and straight.
- **animated**: Whether to apply animation effect on edge or not.
- **id**: A unique identifier. The component generates a UUID each time a new edge is created in diagram by user. Users can generate their own id if needed by Python script.
- **markerEnd**: Set the marker on the end of an edge.
 - **type**: Specify the marker shape either Arrow or ArrowClosed.
 - **color**: Specify the color of the marker.
 - **width**: Specify the size of the marker. Both width and height should be same value.
 - **height**: Specify the size of the marker. Both width and height should be same value.
- **markerStart**: Set the marker on the start of an edge.
 - **type**: Specify the marker shape either Arrow or ArrowClosed.
 - **color**: Specify the color of the marker.
 - **width**: Specify the size of the marker. Both width and height should be same value.
 - **height**: Specify the size of the marker. Both width and height should be same value.
- **style**: CSS code to apply to the Edge. The most useful ones are "stroke" used for edge color and "strokeWidth" used for size of the edge.
- **Interaction**: Collections of options and functionality for user interactions with the Flow Diagram.
 - **snapToGrid**: Control options for activating background grid of the flow diagram. The top-left corner of the node will snap to the grid.
 - **snapGrid**: An array of two elements to specify the grid space in x and y axis.
 - **panOnScrollMode**: Used to limit the direction of panning when panOnScroll is enabled.
 - The "free" option allows panning in any direction.
 - **selectionMode**: Controls how nodes and edges are selected within the flow diagram. This is only activated if **panOnDrag** is set to False.
 - Full: Nodes and edges are only selected when their entire bounding box is within the selection area.
Partial: Nodes and edges are selected even if only a part of their bounding box is within the selection area.
 - **connectionMode**: Controls how connections (edges) are created between nodes. A "loose" connection mode will allow to connect handles of any type to one another. The "strict" mode will only allow to connect source to target handles.
 - **minZoom**: Minimum zoom level that user can apply in runtime.
 - **maxZoom**: Maximum zoom level that user can apply in runtime.
 - **zoomOnPinch**: Controls if the view should zoom by pinching on a touch screen.
 - **zoomOnDoubleClick**: Controls if the view should zoom by double-clicking somewhere on the container.
 - **zoomOnScroll**: Controls if the view should zoom by scrolling inside the container.
 - **disableDragHTMLtag**: An array of HTML tags to control on which type of HTML tag the dragging(panning) should be disabled. It is useful if your components must use dragging functionality which conflicts with zoom and pan dragging like text input.
 - **panOnDrag**: Enabling this allows users to pan the view by clicking and dragging. You can also set this prop to an array of numbers to limit which mouse buttons can activate panning. For example, [0,2] would allow panning with the left and right mouse buttons.

- **panOnScroll**: Controls if the view should pan by scrolling inside the container.
 - **panOnScrollSpeed**: Controls how fast the view should be panned on scroll.
 - **autoPanOnConnect**: Controls whether the view automatically pans when creating a new connection (edge) between nodes and moves beyond the viewport's boundaries.
 - **autoPanOnNodeDrag**: Controls whether the viewport automatically pans when drag a node toward the edge of the current view. This feature helps to keep the node in view without manually adjusting the viewport.
 - **nodesFocusable**: When True, focus between nodes can be cycled with the Tab key. This option can be overridden by individual node setting their focusable prop.
 - **edgesFocusable**: When True, focus between edge can be cycled with the Tab key. This option can be overridden by individual edge setting their focusable prop.
 - **selectionOnDrag**: Controls whether dragging a node or edge also triggers a selection.
 - **selectNodesOnDrag**: Controls whether nodes are automatically selected when you start dragging them.
 - **elevateNodesOnSelect**: Controls whether selected nodes are visually "elevated" above other elements in the diagram. This can help users clearly distinguish selected nodes when node overlap on each others.
 - **connectOnClick**: Controls whether you can create connections (edges) between nodes by simply clicking on the source and target handles, instead of dragging. Provides a more precise and accessible way to create edges, especially in dense or complex diagrams. Set to False, if you want to reduce accidental connections or enforce more deliberate connection behaviors by dragging to create connections.
 - **nodesDraggable**: Controls whether nodes in the flow can be moved around the canvas. This setting is important for managing user interactions, especially when building read-only or fixed-layout diagrams. Individual nodes can override this setting by setting their draggable prop.
 - **nodesConnectable**: Controls whether all nodes can be used as source or target points for connections (edges). This setting is important for customizing which nodes are interactive in terms of making connections. Individual nodes can override this setting by setting their connectable prop.
 - **elementsSelectable**: Controls whether nodes and edges can be selected when users click on them. This setting allows you to customize the selection behavior for all elements in the flow (nodes and edges). This is useful for read-only diagrams where you don't want users to modify the selection state. If nodesDraggable is True and elementSelectable is False, user can drag and move nodes but can't select it.
- **controls**: Show/hide the controls button on the Flow Diagram.
 - **show**: Show/hide all the buttons at once.
 - **zoom**: Show/hide the zoom buttons.
 - **interactive**: Show/hide the lock buttons.
 - **fitView**: Show/hide the fit view button.
 - **duration**: Specify the transition animation time for the fit view function.
 - **resize**: Control options for resizing the node in diagram.
 - **show**: Show/hide all the resize handle for selected node.
 - **color**: Specify the color of the resize handle.
 - **style.width**: Specify the width of resize handles.
 - **style.height**: Specify the height of resize handles.

- **miniMap**: Control options for small overview version of the graph. min the node in diagram. It shows a high-level view of the entire graph, allowing users to see the full layout and quickly navigate to different parts of the graph without having to scroll or zoom in/out extensively.
 - **show**: Show/hide all mini map in the UI.
 - **pannable**: Whether pan disable or enable on mini map.
 - **zoomable**: Whether zoom disable or enable on mini map.
 - **inversePan**: Reverse the pan functionality on mini map.
 - **nodeColor**: Specify the color of the node in mini map.
 - **nodeStrokeColor**: Specify the stroke color of the node in mini map.
 - **nodeBorderRadius**: Specify the border radius of the node in mini map.
 - **nodeStrokeWidth**: Specify the border width of the node in mini map.
 - **maskColor**: Specify the color of mask or overlay area in min map which highlights the portion of the graph that is visible in the viewport, helping users identify which part of the graph they are currently focused on.
 - **maskStrokeColor**: Specify the stroke color of mask cutout area.
 -
- **layouting**: Apply the automatic layout algorithms on connect nodes to arrange them in different layouts. This is useful in applications where the position of nodes is unknown like Material flow diagram in MES.
 - **enabled**: Set True to active the automatic layouting.
 - **autoLayout**: Toggling this prop will apply the automatic layouting on the nodes.
 - **spacingNodeBetweenLayers**: Defines the vertical or horizontal distance between nodes in successive layers, depending on the layout direction.
 - **spacingNode**: Determines the minimum distance between any two nodes within the same layer.
 - **direction**: Specifies the primary axis along which nodes are arranged during layout computation. This setting is particularly relevant when using the **layered** layout algorithm, as it influences the overall orientation of the graph.
 - **algorithm**: Controls layout algorithms that arrange graph structures in a way that minimizes edge crossings, optimizes node placement, and produces readable, aesthetically pleasing layouts.
 - **layered**: Arranges nodes in layers and optimizes the layout by minimizing edge crossings. Nodes are arranged into layers based on their relationships. Edge crossings are minimized as much as possible. Works well for directed acyclic graphs, like flowcharts or organizational charts.
 - **stress**: This algorithm aims to create an optimal layout by minimizing the stress function. (minimize the "stress" the distance based on graph connectivity and the actual distances in the layout). Iteratively adjusts node positions to reduce "stress." Uses a force-based approach to determine node placement. Works well for graphs where the relationship between nodes needs to be represented by their physical distance.
 - **mrtree**: The Tree algorithm is specifically designed for layouts where nodes have a clear hierarchical structure, like trees or decision trees. This algorithm arranges nodes in a tree-like structure, where each node has one parent (except for the root) and can have multiple children.
 - **force**: The **Force-Directed** algorithm is a popular layout technique based on physical forces. It models nodes as if they are physical objects, with forces acting upon them to achieve an optimal layout. This spreads out nodes in the graph so that no two nodes are too close or too far apart, resulting in a natural-looking layout. Great for

undirected graphs and graphs with complex relationships. It can be slower than other algorithms for large graphs.

- **radial**: Arranges the nodes of a graph in a circular or radial pattern where nodes are arranged along the perimeter of a circle. Useful for representing cyclic graphs or graphs where a central node has many connections.
- **background**: Control options for background grid of the flow diagram. For activating the grid set `interaction.snapToGrid` to `True`. The top-left corner of the node will snap to the grid.
 - **show**: Show/hide the grid in diagram.
 - **gap**: An array of two elements to specify the grid space in x and y axis.
- **fitView**: Toggling this prop will autofit the whole diagram.

Events:

- **currentZoom**: Triggered when zoom level of the view is changed. Useful to implement the clutter/decluttering effect on the view by sending the zoom level to view objects. The object receives zoom level message and show/hide their details based on this value.
- **onNodeClick**: Triggered when a user clicks on a node. This event is commonly used for handling interactions with specific nodes, such as opening a detail panel, updating node data, or triggering other actions when a node is clicked. Return node object and node index.
- **onEdgeClick**: Triggered when a user clicks on an edge. This event is commonly used for handling interactions with specific edges, such as opening a detail panel, updating node data, or triggering other actions when an edge is clicked. Return edge object and edge index.
- **onNodeDelete**: Triggered when a node is deleted via user action, such as clicking a delete button (backspace key). This event allows to handle the removal of nodes from the flow, including performing additional tasks like cleaning up related data or updating the state. The Flow diagram will also delete connected edges automatically. In the case you delete a node from nodes props by script, you need to delete related edges as well otherwise those connected edges will remain orphan.
- **onEdgeDelete**: Triggered when an edge is deleted via user action, such as clicking a delete button (backspace key). This event allows to handle the removal of the edge from the flow, including performing additional tasks like cleaning up related data or updating the state.
- **onNodeDoubleClick**: Triggered when a user double clicks on a node. This event is commonly used for handling interactions with specific nodes, such as opening a detail panel, updating node data, or triggering other actions when a node is clicked. Return node object and node index.
- **onEdgeDoubleClick**: Triggered when a user double clicks on an edge. This event is commonly used for handling interactions with specific edges, such as opening a detail panel, updating node data, or triggering other actions when an edge is clicked. Return edge object and edge index.
- **onClick**: Triggered when user click on Flow diagram viewport. Return position of click in x and y.
- **onRightClick**: Triggered when user right click on Flow diagram viewport. Return position of click in x and y.

Functions:

- **self.flyToNode(nodeId, duration)**: Center the viewport to a node (nodeId) coordinate with a smooth pan-zoom animation.
 - **Parameters**:
 - **nodeId**: The id of the target node to fly to.

- **duration:** Numeric value for transition animation to target location.
- **Parameters:** Nothing
- **flyTo(x, y, zoom, duration):** Center the Zoom and Pan viewport to x, y coordinate with a smooth pan-zoom animation.
 - **Parameters:**
 - **x:** Numeric value for x axis target location to fly to.
 - **y:** Numeric value for y axis target location to fly to.
 - **zoom:** Numeric value for zoom level on target location.
 - **duration:** Numeric value for transition animation to target location.
 - **Parameters:** Nothing
- **fitView(duration):** Fit the view inside the Zoom and Pan viewport.
 - **Parameters:**
 - **duration:** Numeric value for transition animation to fit the view.
 - **Parameters:** Nothing

8. System Functions:

The i4Cortex scripting API, which is available under the *system.i4cortex*, is a set of functions that are useful when designing projects in Ignition with vCortex Components.

System.i4cortex.gui.getViewObjects

Description:

Return the objects position inside the given Perspective view. This is only working for coordinate containers as other ones don't have components position. This is especially useful for implementing search on zoom and pan viewer. To skip components from search simply add a custom property called search and set its value to false.

Parameters:

- **view (string):** Perspective coordinate container view path to search on.
- **project (string):** The name of the project which the view in it. Leaving this empty, cause function uses the current project or the parent project which this function is called from. In the case of nested inheritance parent projects all of them will be searching for the view.

Return (object list): List of python objects with following details:

Key	Value
name	The name(meta.name) of the component.
x	The x position (top left corner) of the component.
y	The y position (top left corner) of the component.
mx	The x position of the component center.
my	The y position of the component center.
w	The component width.
h	The component height.

Code Example:

```
# Return list of all components inside the view called 'overview' from project 'project1'.  
system.i4cortex.gui.getViewObjects('overview', 'project1')
```

System.i4cortex.gui.getViewSize

Description:

Return the default dimensions of the given Perspective view. It is useful for view scaling in Perspective applications.

Parameters:

- **view (string):** Perspective coordinate container view path to search on.

- **project (string):** The name of the project which the view in it. Leaving this empty, cause function uses the current project or the parent project which this function is called from. In the case of nested inheritance parent projects all of them will be searching for the view.

Return (object): A python objects with following details:

Key	Value
w	The default view width.
h	The default view height.

Code Example:

Return the size of the view called 'overview' from project 'project1'.

```
system.i4cortex.gui.viewSize ('overview', 'project1')
```

System.i4cortex.gui.getViewTree

Description:

Return the all the views hierarchy and format as a Perspective Tree items from the project, which is called, so you can simply bind the result to the Perspective Tree Components (props.items).

In the case the project has multiple nested parent projects, all related views are also bring to the tree.

Parameters:

- **startView (string):** Starting view Path that return views from there. If it leaves empty function start from root level and return all the views.
- **project (string):** The name of the project for returning the views. Leaving this empty, cause function uses the current project and the parent projects which this function is called from. In the case of multi-deep level of inheritance of parent projects all of them will be in the view tree.

Return (object list): An array of objects, each of which represents a view or folder on the tree with the following details:

Key	Value
label	The name of the view or folder.
expanded	Whether or not the tree appears with all levels expanded.
items	An array of objects, each of which represents a view or folder on the tree
data	View Object. path: view path folder: Whether or not the item is folder or view.

Code Example:

Return the tree object of all views (from root level) inside the project1.

```
system.i4cortex.gui.getViewTree('', 'project1')
```

9. Support:

- Free Upgrades with Ignition Updates for 8.1 and 8.3
- Support via email modules.support@i4corex.com