Data Lake Visualization

Developer Guide

Issue 01

Date 2020-08-14

HUAWEI TECHNOLOGIES CO., LTD.

Contents

1 Custom Component Development

Development Process

1. **Environment Preparation**
2. **Installing Developer Tools**
3. **Generating Component Packages**
4. **Developing Components**
5. **Previewing Components**
6. **Publishing Components**

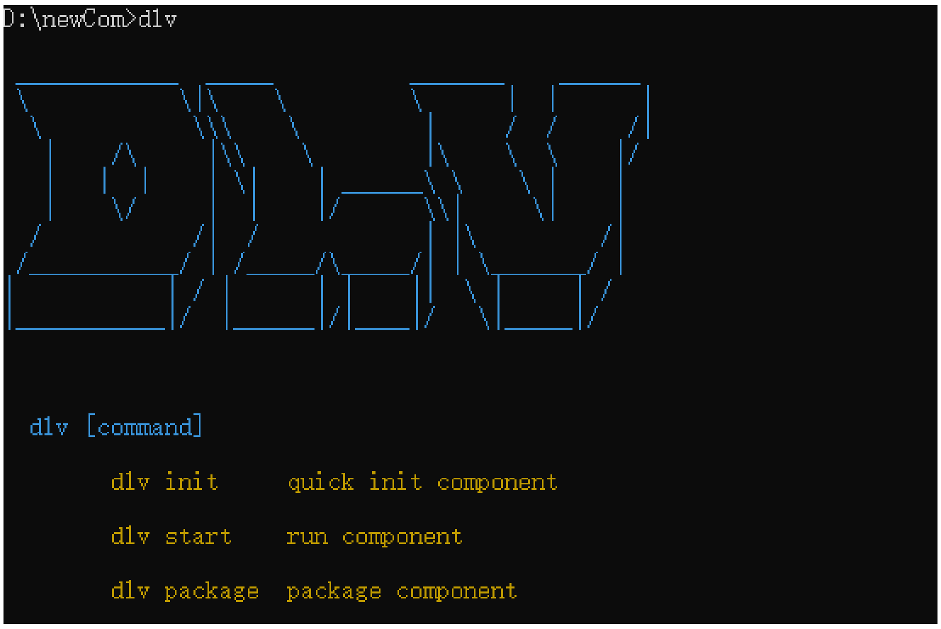
Environment Preparation

Go to the **Node.js** official website and download and install a Node.js version suitable for your operating system. Node.js supports the Windows, macOS, and Linux operating systems.

Installing Developer Tools

1. Log in to the DLV management console. On the **Control Center** > **Components** page, click **Download Developer Tools** in the upper right corner of the page to download the developer tool package named **dlv-cli-***x.x.x*, where *x.x.x* indicates the tool version.
2. Decompress the **dlv-cli-***x.x.x* package to the local host. Open the command prompt window in Windows or open the CLI terminal in Linux or Mac. Run the **cd** command to go to the **dlv-cli-***x.x.x* directory, and run the **npm i** command to install dependency packages, and then run the **npm link** command to install the **dlv-cli** developer tool.
3. After the installation is successful, run the **dlv** command to view information about the current developer tool.

Figure 1-1 Running the **dlv** command



```
D:\newCom>dlv

 dlv [command]

       dlv init     quick init component

       dlv start    run component

       dlv package  package component
```

Table 1-1 Description of developer tool commands

Command	Description
dlv init	Quickly initialize component templates.
dlv start	Start component packages to preview components.
dlv package	Package components.

Generating Component Packages

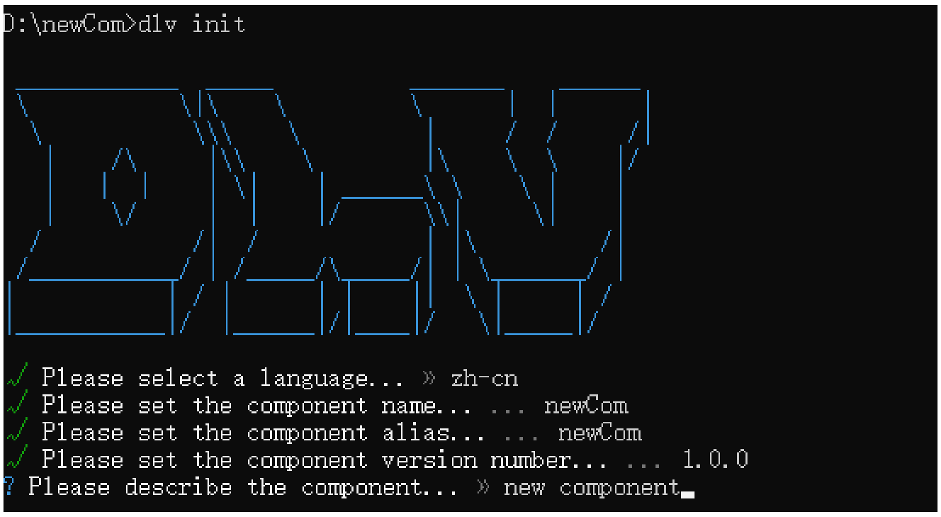
A component package is a custom component template provided by DLV. You can develop components based on the custom template.

Create a directory, for example, **newCom**, go to the directory, and run the **dlv init** command to create a component, as shown in **Figure 1-2**. Enter the information about the new component as prompted.

NOTE

Do not run the **dlv init** command in the directory of developer tool **dlv-cli-***x.x.x*. Otherwise, the developer tool cannot be used properly.

Figure 1-2 dlv init



```
D:\newCom>dlv init

√ Please select a language... » zh-cn
√ Please set the component name... ... newCom
√ Please set the component alias... ... newCom
√ Please set the component version number... ... 1.0.0
? Please describe the component... » new component
```

Table 1-2 Component messages

Message	Description
Please select a language...	Use ↑ and ↓ to select a language.
Please set the component name...	Set a component name, which consists of 1 to 32 characters, including letters, digits, and underscores (_), and must start with a letter or underscore (_). Unless otherwise specified, the names in this document comply with the naming rules.
Please set the component alias...	Set a component alias, which consist of 1 to 32 characters, including Chinese, letters, and special characters. .
Please set the component version number...	Set a component version number. The default version number is 1.0.0.
Please describe the component...	Describe a component.

If the corresponding template file is generated in the new directory, the component package is successfully generated.

├──node_modules # npm dependency package

├──gui.json # Component configuration

├──index.js # Component entry

├──index.less # Component style

└──package.json # npm module description

Developing Components

After a component package is generated, you can customize components based on the generated template. For details, see **Component Development Guide**.

Previewing Components

Go to the component directory and run the **dlv start** command to preview a component. If the command output shows that the service is started, the Chrome browser is automatically launched and navigates to the component preview page.

The preview page consists of the central canvas area and the toolbar on the right. The details are as follows:

- Central canvas area

  The central canvas area displays components and allows you to observe component changes in real time.

  All configurations and data modifications made on the toolbar on the right are displayed on the components of the central canvas in real time.

  The black frame of a component indicates the container size of the component. The black frame can be scaled in each direction to test the scaling performance of the component.
- Toolbar

  The toolbar on the right consists of the pattern, data, and interaction panels.

Table 1-3 Panel description

Panel	Description
Pattern	The **Pattern** panel provides configuration items for components. After you set the configuration items, the corresponding configurations take effect on the components immediately.
Data	The **Data** panel provides data API configuration items for components. Once the data on the **Data** panel is changed, the changes are made on the components accordingly.
Interaction	The **Interaction** panel describes the component interaction.

Publishing Components

Go to the component directory and run the **dlv package** command. A **tar.gz** package named in the format of *component name-version number* is generated outside the component directory. Upload the package to a component package on the **Components** page of the DLV management console to publish the package.

2 Component Development Guide

2.1 Component Development Package Files

This section describes the file structure that developers must comply with when developing custom components of DLV

After you run the **dlv init** command to generate a component package, the component package contains the following files:

Table 2-1 Development package files

File	Description
gui.json	Configuration file, recording the pattern, data, and interaction settings.
Index.js	Main entry file.
Index.less	CSS configuration file.

2.2 Index.js File

The **Index.js** file is the main entry file of a component. This file provides an example for your reference and describes the life cycles and related functions of the common components in the **index.js** file.

Table 2-2 Function description

Function	Description
refresh()	Default rendering function, which is called when a component is initialized and redrawn. (Custom function implementation)

Function	Description
resize()	Scaling function, which is called when a component is dragged or zoomed.
loadData()	Data loading function, which is called when a component loads data. (Custom function implementation)
dispatch(String: eventType, object: data)	This function is optional. It is used to trigger event interaction. If a component supports interaction, this function can be called by you to configure events in the **gui.json** file. ● eventType: event type. ● data: data transferred when an event is triggered.
getRelation()	Mapping acquisition function, which is called when you obtain the data mapping from the data panel.

The following is a **Index.js** file example:

```
import gui from './gui.json';
import './index.less';
//global: echarts jQuery

class App extends BaseChart {
  constructor(props = {}) {
    super(props);
    Object.assign(props);
    this.chart = echarts.init(this.el);
  }
  refresh() {
    this.loadData();
  }
  loadData() {
    let option = this.getOption();
    if (option) {
      this.chart.setOption(option, true);
    }
  }
  resize() {
    this.chart.resize();
  }
  getOption() {
    let styledata = this.config.styledata,
        relation = this.getRelation();
    let yData = [], data = [];
    this.data.forEach(item => {
      yData.push(item[relation.x]);
      data.push(item[relation.y]);
    });
    return {
      backgroundColor: '222',
      grid: {
       top: '20',
       left: '20',
       right: '20',
       bottom: '20',
       containLabel: true
      },
      yAxis: [{
```

```
      type: 'category',
      data: yData,
      inverse: true,
      axisTick: {
        show: false
      },
      axisLabel: {
        margin: 10,
        textStyle: {
          fontFamily: styledata['font'],
          fontSize: 24,
          color: '#fff'
        }
      },
      axisLine: {
        show: false
      }
     }],
     xAxis: [{
       type: 'value',
       axisLabel: {
         show: false
       },
       axisLine: {
         show: false
       },
       splitLine: {
         show: false
       }
     }],
     series: [{
       type: 'bar',
       barWidth: 14,
       data,
       label: {
         normal: {
           show: true,
           position: 'insideBottomRight',
           formatter: '{c}%',
           distance: 0,
           offset: [30, -20],
           color: '#fff',
           fontSize: 16,
           padding: [5, 15, 10, 15]
         }
       },
       itemStyle: {
         normal: {
           color: new echarts.graphic.LinearGradient(1, 0, 0, 0, [{
             offset: 0,
             color: '#57eabf'
           }, {
             offset: 1,
             color: '#2563f9'
           }], false),
           barBorderRadius: 14
         }
       }
     }, {
       type: "bar",
       barWidth: 14,
       xAxisIndex: 0,
       barGap: "-100%",
       data: [120, 120],
       itemStyle: {
         normal: {
           color: "#444a58",
           barBorderRadius: 14
         }
        },
        zlevel: -1
      }]
    };
  }
}
App.gui = gui;export default App;
```

2.3 gui.json File

The **gui.json** file is the configuration file of a component. This section describes the fields in the **gui.json** file. You can modify the **gui.json** file and customize the component configuration panel based on the field description in this document.

The following is a **gui.json** file example:

```
{
  "name": "newCom",
  "attr": {
    "w": 650,
    "h": 378
  },
  "style": [...],
  "data": {...},
  "event": {...}
}
```

Table 2-3 Description of parameters in **gui.json**

Parameter	Mandatory	Type	Description
name	Yes	String	Name of the component.
attr	No	**attr** object	Basic configuration of the component width and height (unit: pixel).
style	Yes	Array of **Style** object	Configuration of the pattern panel.
data	Yes	Object	Configuration of the data panel. For details, see **Configuring Component Data**.
event	Yes	Object	Configuration of the interaction panel. For details, see **Configuring Component Interaction**.

Table 2-4 Description of the **attr** parameters

Parameter	Mandatory	Type	Description
w	No	Number	Component width (unit: pixel).
h	No	Number	Component height (unit: pixel).

Configuring the Component Style

```
"style": [
 {
   "label": "global",
   "isExpand": true,
   "children": [
     {
       "label": "Font",
       "name": "font",
       "type": "fontfamily",
       "value": "YouYuan"
     },
     {
       "label": "Color",
       "name": "color",
       "type": "color",
       "value": "rgba(70, 94, 212, 1)"
     }
   ]
 }
]
```

The **Style** parameter is in array format. Each data element forms a style configuration item, and can contain related sub-configuration items. A maximum of three levels are supported. The parameter is described as follows:

Table 2-5 Description of the **Style** parameters

Parameter	Mandatory	Type	Description
label	Yes	String	Label name of the configuration item.
isExpand	No	Boolean	Indicates whether sub-configuration items can be collapsed or expanded. true: yes; false: no
type	Yes	String	UI type of the configuration item. For details about the UI types supported by DLV, see **Supported Types**.
name	Yes	String	Key value of the configuration item. The value must be unique.
Other	No	-	Different UI types have different attribute parameters. For details, see **UI Types**.
children	No	Array of **children** object	Sub-configuration items of the configuration item. As shown in the following figure, the **Global** style contains **Font** and **Color**.

Table 2-6 Description of the **children** parameters

Parameter	Mandatory	Type	Description
label	Yes	String	Label name of the configuration item.

Parameter	Mandatory	Type	Description
type	Yes	String	UI type of the configuration item. For details about the UI types supported by DLV, see **UI Types**.
name	Yes	String	Key value of the configuration item. The value must be unique.
Other	Yes	-	Different UI types have different attribute parameters. For details, see **UI Types**.

Configuring Component Data

data: data panel configuration of the component.

The following provides an example:

```
"data": {
  "fields": {...},
  "config": {...}
}
```

Table 2-7 Description of the **data** parameters

Parameter	Mandatory	Type	Description
fields	Yes	Object	Field mapping area. For details about the field mapping area on the data panel, see **fields**.
config	Yes	Object	Static data area on the data panel. For details, see **config**.

- **fields**

  In the example, the **fields** parameter is as follows:

```
"fields": {
  "x": {
    "value": "",
    "desc": "x"
  },
  "y": {
    "value": "",
    "desc": "y"
   },
   "s": {
     "value": "",
     "desc": "s",
     "type": "series",
     "gui": [
       {
          "label": "",
          "isCheck": true,
          "children": [
            {
               "label": "Name",
               "name": "series.name",
               "type": "input",
               "value": ""
            },
            {
               "label": "Color",
               "name": "series.color",
               "type": "color",
               "value": "rgba(195,53,53,1)"
            }
          ]
        }
      ]
   }
}
```

The **fields** parameter contains multiple objects in the key:value format. The value of **key** is the field name. The value of **value** corresponding to the **key** value is defined as follows:

Table 2-8 Description of the **Fields value** parameters

Parameter	Mandatory	Type	Description
value	Yes	String	Field name in the mapped source data.
desc	Yes	String	Field name description, which is the same as the key value.
type	No	String	The value is fixed to **series**. If the type parameter exists, the field is a series field (indicating that the field is used to distinguish two or more groups of data displayed in the same chart). Each series can map the value of the field. In the example, the **s** field contains a default series and a series corresponding to value **1**. Each series contains some component style configurations.
gui	No	Array	The **gui** parameter is available only when **type** is set to **series**. This parameter lists the style configuration required by the series.

- **config**

  In the example, the **config** parameter is as follows:

```
"config": {
  "data": [
    {
      "x": "2018",
      "y": 78,
      "s": 1
    },
   {
      "x": "2016",
      "y": 55,
      "s": 1
   },
  {
      "x": "2017",
      "y": 68,
      "s": 1
  },
  {
      "x": "2018",
      "y": 48,
      "s": 1
  },
  {
      "x": "2019",
      "y": 70,
      "s": 1
   },
  {
      "x": "2020",
      "y": 85,
      "s": 1
   }
   ]
}
```

The **data** parameter in the **Config** parameter corresponds to the static data displayed on the data panel. The dynamic data source can be connected after the component package is uploaded. The data format needs to be defined based on the source data of the user.

Configuring Component Interaction

event: component interaction configuration.

The following provides an example:

```
"event": {
   "Data change": {
     "enable": false,
     "fields": {
       "value": ""
     }
   }
}
```

The **event** parameter is an object in the key: value format. The value of **key** is the event name, and the value of **value** is the configuration of the event. The details are as follows:

Table 2-9 Description of the **event value** parameters

Parameter	Mandatory	Type	Description
Enable	Yes	Boolean	Indicates whether to enable the event. true: enable, false: disable
Fields	Yes	Object	Field mapping relationship corresponding to the interaction event, including one key:value at least. The value of **key** is the parameter name transferred during interaction, and the value of **value** is the transferred value that is obtained from the component data.

2.4 UI Types

This section describes the component configuration items supported by DLV. You can use the **type** field defined in the **gui.json** file to define types and configurations of components.

Supported Types

The **Type** field supports the following:

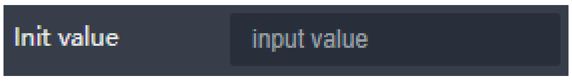
- Input: **Text Box**.
- Number: **Numeric Control**. The value can be entered and the maximum and minimum values can be customized.
- Select: **Drop-down List**. The filtering and custom input are supported.
- Color: **Color Selector**.
- Checkbox: **Check Box**.
- Slider: **Slider**

Text Box

Table 2-10 Parameter description

Field Name	Mandatory	Parameter Type	Description
placeholder	No	String	Message displayed when the value is empty.
value	Yes	String	Value in the text box.

The following provides an example:



```
{
  "label": "Init value",
  "name": "initvalue",
  "type": "input",
  "placeholder": "input value",
  "value": ""
}
```

Numeric Control

Table 2-11 Parameter description

Field Name	Mandatory	Parameter Type	Description
value	Yes	Number	Value of the numeric control. The default value is **0**.
min	No	Number	The minimum value. The default value is **-30,000**.
max	No	Number	The maximum value. The default value is **30,000**.
precision	No	Number	Number of decimal places to be reserved. The default value is **0**.
step	No	Number	Adjustment range. The default value is **1**.

The following provides an example:

```
{
  "label": "Size",
  "name": "size",
  "type": "number",
  "min": 0.1,
  "max": 1.5,
  "precision": 1,
  "step": 0.1,
  "value": "0.6"
}
```

Drop-down List

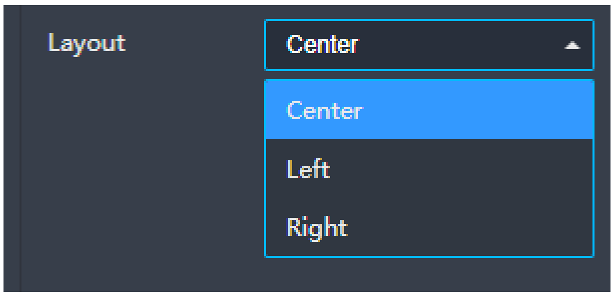
Table 2-12 Parameter description

Field Name	Mandatory	Parameter Type	Description
value	Yes	string	Value selected from the drop-down list.
data	Yes	Array<Object>	All the optional values contained in the drop-down list.
isSearch	No	Boolean	Indicates whether the drop-down list supports the search function. The default value is **false**.

The following provides a data example:

```
{
  "label": "Layout",
  "name": "initvalue",
  "type": "select",
  "data": [
    {
      "key": "Center",
      "value": "center"
    },
    {
      "key": "Left",
      "value": "left"
    },
    {
      "key": "Right",
      "value": "right"
     }
   ],
   "value": "center"
}
```

Figure 2-1 Drop-down list



Color Selector

Table 2-13 Parameter description

Field Name	Mandatory	Parameter Type	Description
value	Yes	String	Value selected by the color selector. The value can be a gradient color or an echarts color value.
gradient	No	Boolean	Indicates whether gradient colors are supported. The default value is **true**.

The following provides an example:

```
{
  "label": "Color",
  "name": "color",
  "type": "color",
  "value": "rgba(70,94,212,1)",
  "gradient": true
},
```

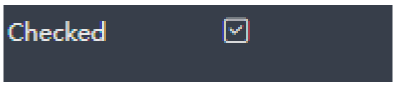
Check Box

Table 2-14 Parameter description

Field Name	Mandatory	Parameter Type	Description
value	Yes	Boolean	Value of the check box. The value can be **true** or **false**.

The following provides an example:

Figure 2-2 Check box



```
{
  "label": "Checked",
  "name": "checked",
  "type": "checkbox",
  "value": true
}
```

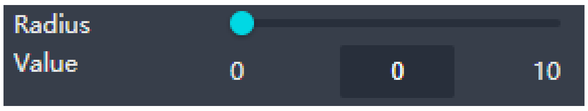
Slider

Table 2-15 Parameter description

Field Name	Mandatory	Parameter Type	Description
value	Yes	Number	Value of the slider. The default value is **0**.
min	No	Number	Minimum value of the slider. The default value is **0**.
max	No	Number	Maximum value of the slider. The default value is **100**.
step	No	Number	Adjustment range. The default value is **1**.

The following provides an example:

Figure 2-3 Slider



```
{
  "label": "Radius Value",
  "name": "radius",
  "type": "slider",
  "value": 0,
  "min": 0,
  "max": 10,
  "step": 1
}
```

A Change History

Released On	Description
2020-08-14	This is the first official release.