



TWaver[®] HTML5

Developer Guide

Version 1.0

Apr 2012

Serva Software

info@servasoftware.com

<http://www.servasoftware.com>

PO Box 8143, Wichita Falls, Texas, USA 76307



For more information about Serva Software and TWaver please visit the web site at:

<http://www.servasoftware.com>

Or send e-mail to:

info@servasoftware.com

Apr, 2012


Notice:

This document contains proprietary information of Serva Software. Possession and use of this document shall be strictly in accordance with a license agreement between the user and Serva Software, and receipt or possession of this document does not convey any rights to reproduce or disclose its contents, or to manufacture, use, or sell anything it may describe. It may not be reproduced, disclosed, or used by others without specific written authorization of Serva Software.

TWaver, servasoft, Serva Software and the logo are registered trademarks of Serva Software. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S.A. and other countries. Other company, brand, or product names are trademarks or registered trademarks of their respective holders. The information contained in this document is subject to change without notice at the discretion of Serva Software.

Copyright © 2012 Serva Software LLC
All Rights Reserved

Table of Contents

- Home 
 - Getting Started
 - Convention
 - TWaver HTML5 Quick Start
 - Setup Environment
 - Basic Knowledge
 - The Design Model and Frame of TWaver HTML5
 - The Data Model of TWaver HTML5
 - The Data Element of TWaver HTML5
 - The Data Container of TWaver HTML5
 - The View Components of TWaver HTML5
 - Network Introduction
 - Table Introduction
 - Tree Introduction
 - Data Serialization
 - Data Element
 - Basic Data Element
 - Alarm Data
 - Layer Data Element
 - Topology Element
 - twaver.Node
 - twaver.Link
 - Links Binding
 - Links Type
 - twaver.Follower
 - twaver.SubNetwork
 - twaver.Group
 - twaver.ShapeNode
 - twaver.Grid
 - Data Container
 - Events of Data Container
 - Quick Finder Mechanism
 - Selection Model Appliance
 - Using Network
 - Network Hierarchy
 - Network Filter
 - Network Function for Style Rule
 - Network GUI Interaction
 - Using Tree
 - Tree Node Presentation
 - Tree's Hierarchy & Order
 - Check Mode of Tree
 - Using Table
 - Table Configuration

- Table Data Orders
- Using Chart
 - Chart Value & Chart Values
 - BarChart
 - LineChart
 - PieChart
 - DialChart
 - BubbleChart
 - RadarChart
- Using Alarm
 - Alarm Severity
 - Alarm State & Statistics
 - Alarm Display
- Appendix
 - Network Element Stylesheet

Home

This documentation introduce you how to use TWaver HTML5 to build HTML5 based software.

- [Getting Started](#)
- [Basic Knowledge](#)
- [Data Element](#)
- [Data Container](#)
- [Using Network](#)
- [Using Tree](#)
- [Using Table](#)
- [Using Chart](#)
- [Using Alarm](#)
- [Appendix](#)

Getting Started

The following chapter is very important because it provides developers with core development information. It demonstrates how to build the programming environment, how to use TWaver HTML5 to create a new project, and how to initially utilize TWaver HTML5 components.

Chapter Content:

- [Convention](#)
- [TWaver HTML5 Quick Start](#)
- [Setup Environment](#)

Convention

There are name conventions for TWaver HTML5 and phrase comparison with TWaver Java. All these are to make you more easily understand this tutorial.

MVC: Model-View-Control model

Data Model: Correspond to Model of MVC, such as Data, DataBox, ElementBox, AlarmBox, LayerBox, Element and etc in this document.

View: Correspond to View of MVC, such as ElementUI, Network, Tree, Table and etc in this document.

Data Element: Data, Element, Node and etc.

Data Container: Data Element container, for example DataBox, ElementBox, AlarmBox, LayerBox.

The class name comparison between TWaver Java and TWaver HTML5.

TWaver HTML5	TWaver Java	Comment
ElementBox	TDataBox	Elements container
AlarmBox	AlarmModel	Alarms container
LayerBox	LayerModel	Layers container
Network	TNetwork	Topology component
Tree	TTree	Tree component
Table	TElementTable	Table Component

The element property comparison between TWaver HTML5 and TWaver Java

TWaver HTML5	TWaver Java	Comment
node.setStyle	node.putClientProperty	Style property
node.setClient	node.putUserProperty	User property

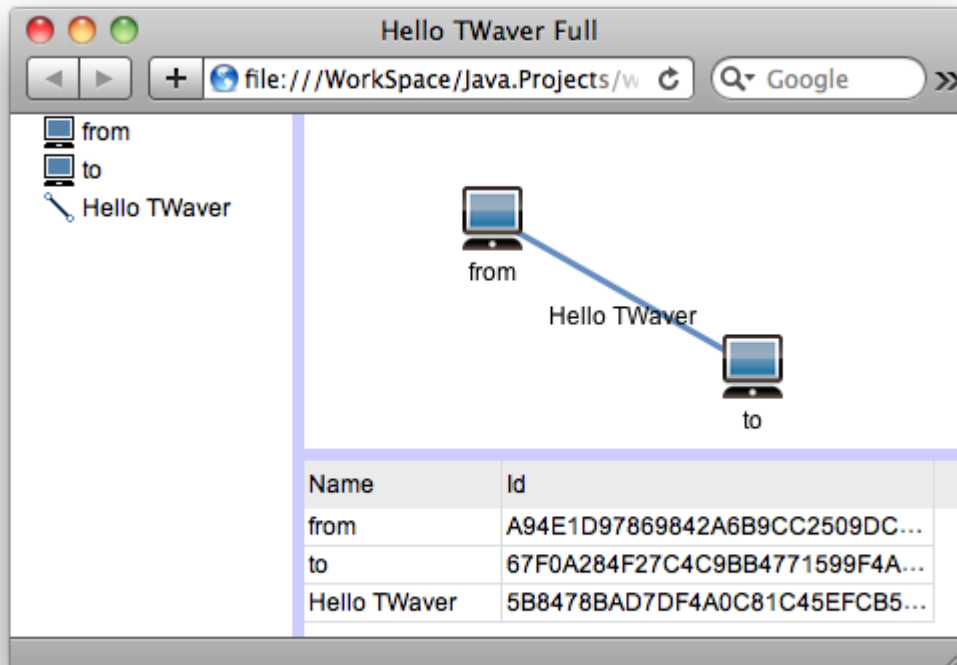
The filter comparison between TWaver HTML5 and TWaver Java

TWaver HTML5	TWaver Java	Comment
getAlarmLabel	alarmLabelGenerator	Generator for alarm label
isVisible	visibleFilter	Visible filter
isMovable	movableFilter	Movable filter
isEditable	elementLabelEditableFilter	Editable filter
getLabel	elementLabelGenerator	Generator for element label
getToolTip	elementToolTipTextGenerator	Generator for element tooltips

getInnerColor	elementBodyColorGenerator	Generator for body color of element
getOuterColor	elementOutlineColorGenerator	Generator for border color of element
getSelectColor	elementSelectColorGenerator	Generator for selected element
getAlarmFillColor	alarmColorGenerator	Generator for alarm
getAlarmLabel	alarmLabelGenerator	Generator for alarm label

TWaver HTML5 Quick Start

The best way to learn TWaver HTML5 is to make an example. Let's create an example application like below. This is a very simple application with a network view, tree and table components.



Now let's start step by step:

- First, create a new HTML file, import twaver.js in your project. Notice html DOCTYPE settings.

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello TWaver Full</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

Here notice that you should set the file head: "<!DOCTYPE html>", it must be set in some browse so that can be created HTML Canvas Element.

- Create topology network.

Firstly, create a network contains Link and Node.

```
function init() {
```

```

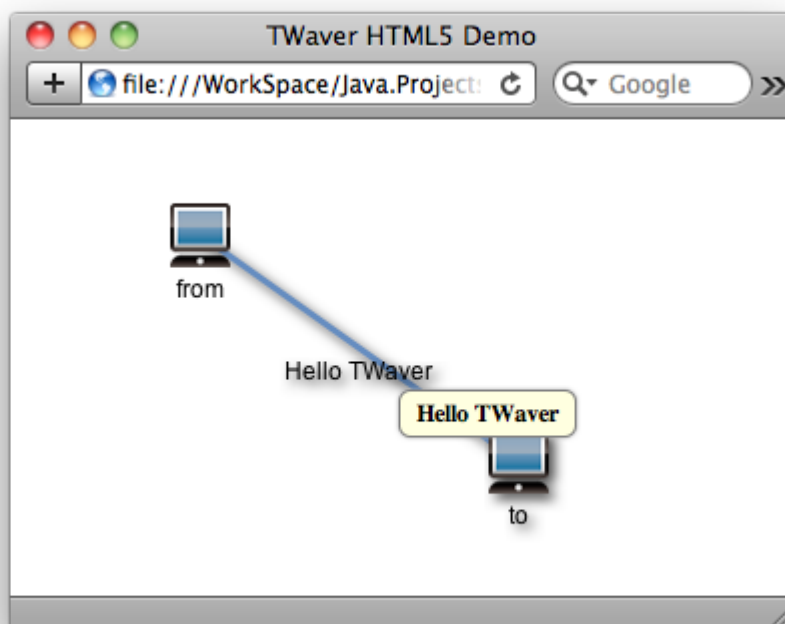
var network = new twaver.network.Network();

var box = network.getElementBox();
var node = new twaver.Node();
node.setName("from");
node.setLocation(100, 100);
box.add(node);
var node2 = new twaver.Node();
node2.setName("to");
node2.setLocation(300, 300);
box.add(node2);
var link = new twaver.Link(node, node2);
link.setName("Hello TWaver");
link.setToolTip("<b>Hello TWaver</b>");
box.add(link);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}

```

Run this example



- Add more components, use split pane to layout.

```

function init() {
    var box = new twaver.ElementBox();
    var node = new twaver.Node();
    node.setName("from");
    node.setLocation(100, 100);
    box.add(node);
    var node2 = new twaver.Node();
    node2.setName("to");

```

```

        node2.setLocation(300, 300);
        box.add(node2);
        var link = new twaver.Link(node, node2);
        link.setName("Hello TWaver");
        link.setToolTip("<b>Hello TWaver</b>");
        box.add(link);

        var network = new twaver.network.Network(box);
        var tree = new twaver.controls.Tree(box);
        var table = new twaver.controls.Table(box);
        var tablePane = new twaver.controls.TablePane(table);
        createColumn(table, 'Name', 'name', 'accessor', 'string');
        createColumn(table, 'Id', 'id', 'accessor', 'string');

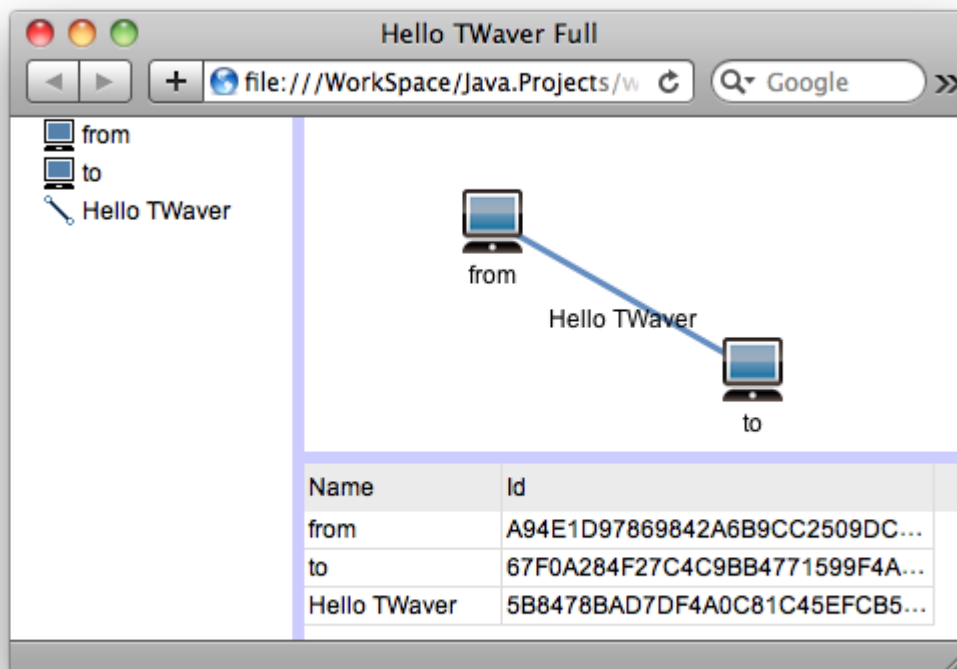
        var rightSplit = new twaver.controls.SplitPane(network, tablePane,
            'vertical', 0.7);
        var mainSplitPane = new twaver.controls.SplitPane(tree, rightSplit,
            'horizontal', 0.3);

        var networkDom = mainSplitPane.getView();
        networkDom.style.width = "100%";
        networkDom.style.height = "100%";
        document.body.appendChild(networkDom);
        network.getView().style.backgroundColor = "#f3f3f3";
        network.getView().style.cursor = "hand";

        window.onresize = function() {
            mainSplitPane.invalidate();
        };
    }
    function createColumn(table, name, propertyName, propertyType, valueType) {
        var column = new twaver.Column(name);
        column.setName(name);
        column.setPropertyName(propertyName);
        column.setPropertyType(propertyType);
        if (valueType)
            column.setValueType(valueType);
        table.getColumnBox().add(column);
        return column;
    }
}

```

Display below:



Setup Environment

To use TWaver HTML5, you need twaver.js library; this can be downloaded from Serva Software website www.servasoftware.com. If you want to use TWaver HTML5 in your project as commercial use, you need to purchase it.

- **Development Tools**

There are several tools to develop Web Project. Such as Plain Text Editor, Eclipse Spket, Netbeans. We don't want to give a recommend. Compare with Java, .NET, using plain text editor to develop JavaScript, HTML, CSS is better.

- **Supported browse**

TWaver HTML5 uses JavaScript to develop; it can run in some browse, IE9+, Firefox, Chrome, Safari, and Opera.

- **Reference documents**

[HTML5 at W3C](#) - standard reference document

[HTML5 Labs](#) - web standard prototype in early period (such as IndexedDB, FileAPI)

[ScriptJunkie.com](#) - web developing topic and information

[CanIUse.com](#) - support detail information about HTML5, CSS3 for different browses

Basic Knowledge

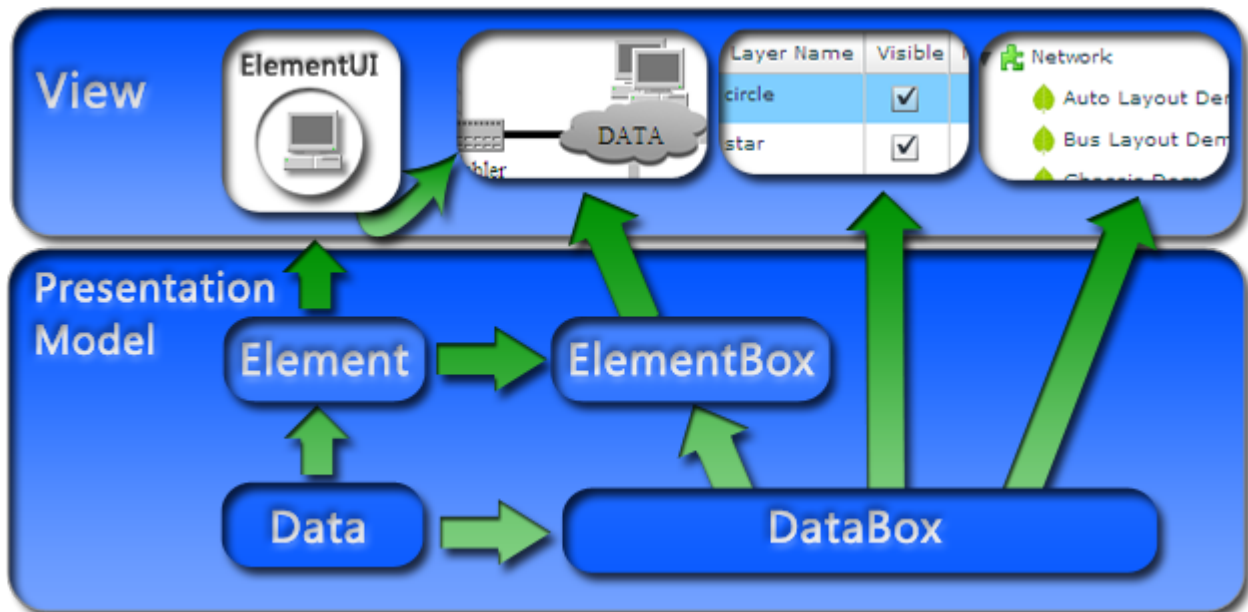
This chapter introduces the UI components and design model of TWaver HTML5. Also included is a brief overview of the data model and the types of visual components. After reading this chapter you will know the principles of TWaver HTML5.

- [The Design Model and Frame of TWaver HTML5](#)
- [The Data Model of TWaver HTML5](#)
 - [The Data Element of TWaver HTML5](#)
 - [The Data Container of TWaver HTML5](#)
- [The View Components of TWaver HTML5](#)
 - [Network Introduction](#)
 - [Table Introduction](#)
 - [Tree Introduction](#)
- [Data Serialization](#)

The Design Model and Frame of TWaver HTML5

TWaver HTML5 is based on MVC (Model-View-Controller) model. The data model of the client side is a nesting MV as a group. The basic data element is `twaver.Data`, and `twaver.DataBox` is the data container. Up to the application layer, `twaver.network.ElementUI` will serve as the basic component, and `twaver.Network`, `twaver.controls.Tree`, `twaver.controls.Table` and etc will serve as the data container components. This is the frame of TWaver HTML5.

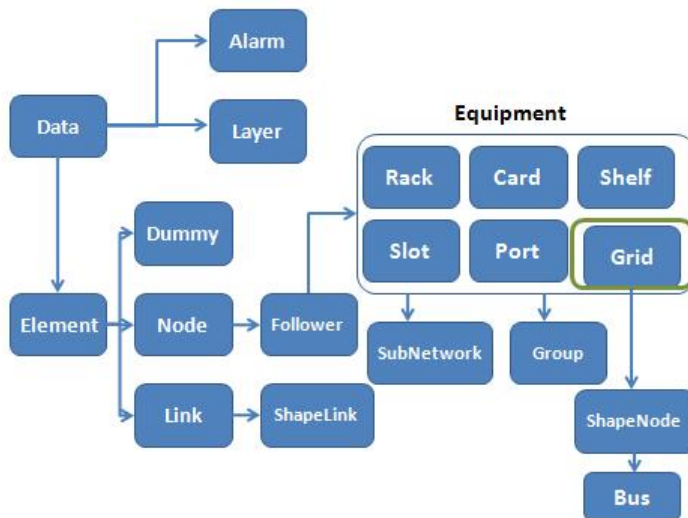
TWaver HTML5 Design Model



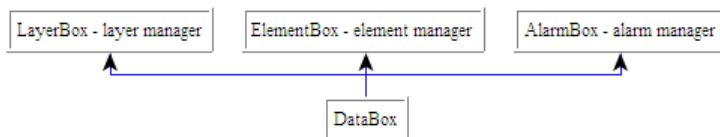
The Data Model of TWaver HTML5

Base on two basic elements, which twaver.Data and twaver.DataBox correspond to be the basic data element and data container, TWaver HTML5 predefined series of business objects, network elements and data containers. For example alarm element (twaver.Alarm) and alarm container (twaver.AlarmBox), view layer (twaver.Layer) and view layer container (twaver.LayerBox), topology element (twaver.Element) and topology container (twaver.ElementBox)...

The structure of data elements



The structure of data containers



Topology container (twaver.ElementBox) integrates other containers, provides abundance topology elements including Dummy, Node, Link, Bus, ShapeNode, ShapeLink, Follower, Rack, Shelf, Slot, Card, Port, Grid, Group, SubNetwork and etc, offers strong support for the design model and business function of webmaster interface developing.

- [The Data Element of TWaver HTML5](#)
- [The Data Container of TWaver HTML5](#)

The Data Element of TWaver HTML5

TWaver HTML5 takes twaver.Data as the basic data element, extends series of data element with GUI and business logic including Alarm, Layer, Element...

- **twaver.Data**

Data is the basic interface of TWaver HTML5 data element, take twaver.Data as its implementation class. It defined its own properties such as id, name, icon, toolTip, parent, children and etc, packaged event dispatcher.

Data contains a twaver.EventDispatcher which gives it's the event dispatching and listening function. It realizes event dispatching and listening by following methods:

```
firePropertyChange: function (property, oldValue, newValue)
addPropertyChangeListener : function (listener, scope, ahead)
removePropertyChangeListener : function (listener)
onPropertyChanged : function(listener)
```

More, Data owns other functions:

```
getChildren: function ()
getChildrenSize: function ()
toChildren: function (matchFunction, scope)
addChild: function (child, index)
removeChild: function (child)
getChildAt: function (index)
clearChildren: function ()
getParent: function ()
setParent: function (parent)
hasChildren: function ()
isRelatedTo: function (data)
isParentOf: function (data)
isDescendantOf: function (data)
```

Let's introduce the implementation classes one by one:

- **twaver.Layer**

Layer, which is the layer management component of TWaver, take twaver.Layer as its implementation class, has three special properties: visible, editable, movable. LayerBox is to maintain all relations of TWaver HTML5 layers, the default order is decided by father-children relationship and adding indexes. In topology, the view of each network is decided by layer ID of corresponding element.

- **twaver.Alarm**

Alarm, which is the data model of equipment fault and network exception of webmaster system, takes Alarm as its implementation class. The connection between Element and Alarm can response the alarm status of network. Alarm owns critical levels to express whether cleared, acknowledged and etc.

Severity	Letter	Value	Color
CRITICAL	C	500	Red

MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

AlarmBox is to manage all alarm issues in TWaver. Alarm and Network can be mutual reference directly, but can be connected by AlarmBox. AlarmState can be referenced by Network to express the level and quality of new alarm events.

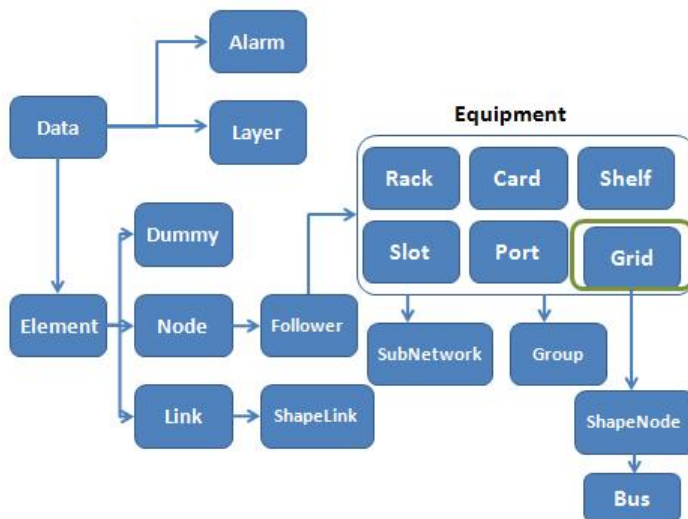
• twaver.Element

Element is the most important data in TWaver which takes Element as its implementation class. Element is to be the object of topology such as Node, Link, SubNetwork, Bus, Card and etc.

TWaver predefined abundance kinds of Network such as Dummy, Node, Link, Bus, ShapeNode, ShapeLink, Follower, Rack, Shelf, Slot, Card, Port, Grid, Group, SubNetwork and etc. Each network correspond one ElementUI class, and the network present its style, these two consists of the MV model.

We can demonstrate the effectiveness and several of presentations by setting the properties and style of Network. At the same time, user can extends all these predefined Elements to cope with special requirement.

ElementBox is to maintain all data of Element. It can drive multiple views such as twaver.Network, twaver.Tree, twaver.Table.



Dummy

It is invisible in topology, but visible in Tree and Table. It is usually used to group logical things which no topology significance.

Node

It is the basic class of other nodes, stands for one node in topology.

Link

It is the basic class of other links, stands for link in topology.

Follower

It stands for follower, can be attached in another node. Once the host moves, Follower will follow up.

Bus

It implements ShapeNode, is one kind of layout node. It can do Bus Layout with other nodes in it.

ShapeNode

Its shape is determined by a series of control point. It present as abundant styles.

ShapeLink

It implements Link but different with Link. Its direction is determined by a series of control point, and it can customize special Connection Layout.

Grid

It is the basic class of Rack, Shelf, Slot, Card and Port, stands for equipment panel. It will display as grid in topology, and can change columns and rows.

Group

It stands for a group, contains child Network. It can be expended or combined, the location and scope of children will decide the location and scope of Group.

SubNetwork

SubNetwork means a lot for topology. In topology will not show all Networks once generally, but only show Networks in current SubNetwork. It can resolve large data volume problem by switching SubNetwork and data lazy loading.

The Data Container of TWaver HTML5

Data Management Container is a container to maintain data. DataBox in TWaver HTML5 is Data Management Container, play an important role as Model in TWaver HTML5 MVC model. One DataBox can drive multiple views, and the data change of DataBox can be shown its related view components. DataBox of TWaver HTML5 supports views including twaver.controls.Tree, twaver.controls.Table. ElementBox of TWaver HTML5 has its own special view component twaver.network.Network.

- **DataBox**

DataBox contains EventDispatcher property, adds listener to data change and container change. User can master all Element transformation by listening DataBox, and can handle the event by using the following function:

```
addDataBoxChangeListener: function (listener, scope, ahead)
removeDataBoxChangeListener: function (listener)
addDataPropertyChangeListener: function (listener, scope, ahead)
removeDataPropertyChangeListener: function (listener)
addHierarchyChangeListener: function (listener, scope, ahead)
removeHierarchyChangeListener: function (listener)
```

One container needs to have way to maintain data alternation. User can add/remove data element by following methods in DataBox:

```
add: function (data, index)
remove: function (data)
clear: function ()
```

There are iterating methods in DataBox as below:

```
getSiblings: function (data)
getRoots: function ()
getDats: function ()
getDataAt: function (index)
toDats: function (matchFunction, scope)
forEach: function (f, scope)
forEachReverse: function (f, scope)
forEachByDepthFirst: function (callbackFunction, data, scope)
forEachByBreadthFirst: function (callbackFunction, data, scope)
```

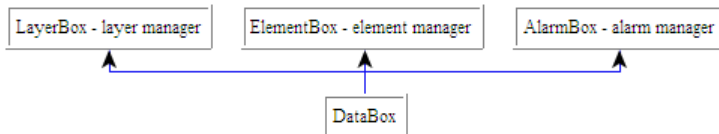
DataBox maintains all layers of data Elements. The following methods is to do moving or inserting operation:

```
moveTo: function (data, newIndex)
moveUp: function (data)
moveDown: function (data)
moveToTop: function (data)
moveToBottom: function (data)
```

More, DataBox packed the selection mechanism of data Element, realized the SelectionModel of Data, offer the simple operation approach

```
getSelectionModel: function ()
moveSelectionUp: function (sm)
moveSelectionDown: function (sm)
moveSelectionToTop: function (sm)
moveSelectionToBottom: function (sm)
removeSelection: function ()
```

TWaver HTML5 defined LayerBox to do layer management, AlarmBox to do alarm management, ElementBox to do network management. And the LayerBox and AlarmBox service for ElementBox, used to maintain the layer and alarm information of network.



• LayerBox

LayerBox is the layer management container, need to be attached to one ElementBox. It defined methods to add/remove layers, and one default layer:

```
twaver.LayerBox = function (elementBox)
getElementBox: function ()
getDefaultLayer: function ()
getLayerByElement: function (element)
```

AlarmBox

AlarmBox is to maintain alarm information, also needs to be attach one ElementBox:

```
twaver.AlarmBox = function (elementBox)
getElementBox: function ()
```

AlarmBox also defined the AlarmElementMapping to manage the matchup between network and alarm information. User can customize the relevance to implement requirements such as one alarm effect multi network:

```
getAlarmElementMapping: function ()
setAlarmElementMapping: function (alarmElementMapping)
```

More, We still offer many properties:

```
//When element is removed from ElementBox, alarm is removed from AlarmBox
//When alarm serverity is set to CLEAR, whether need to remove alarm automatically when clear alarm
isRemoveAlarmWhenAlarmIsCleared: function ()
setRemoveAlarmWhenAlarmIsCleared: function (removeAlarmWhenAlarmIsCleared)
```

ElementBox

ElementBox includes AlarmBox, LayerBox, is the network management container. TWaver HTML5 defined proper view components for ElementBox to show the topology relationships.

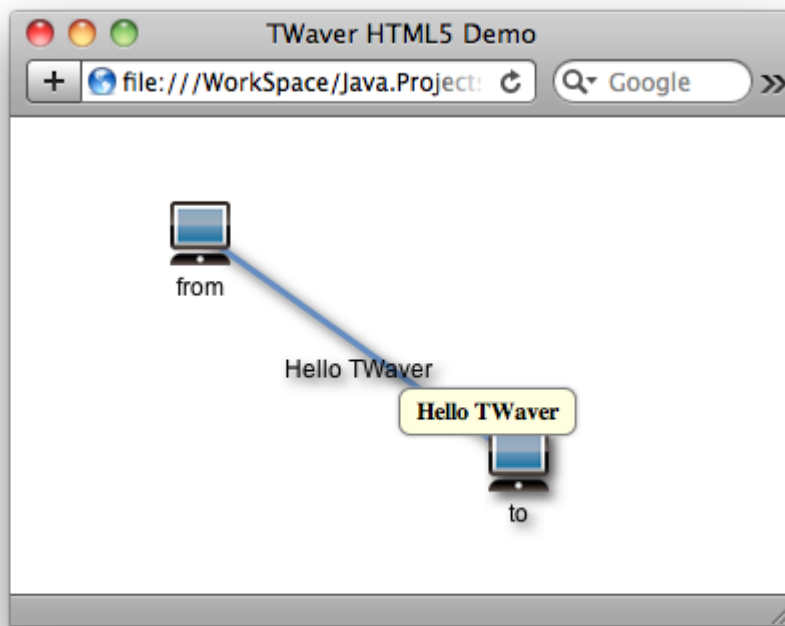
ElementBox HelloWorld application:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello TWaver Full </title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();

  var box = network.getElementBox();
  var node = new twaver.Node();
  node.setName("from");
  node.setLocation(100, 100);
  box.add(node);
  var node2 = new twaver.Node();
  node2.setName("to");
  node2.setLocation(300, 300);
  box.add(node2);
  var link = new twaver.Link(node, node2);
  link.setName("Hello TWaver");
  link.setToolTip("<b>Hello TWaver</b>");
  box.add(link);

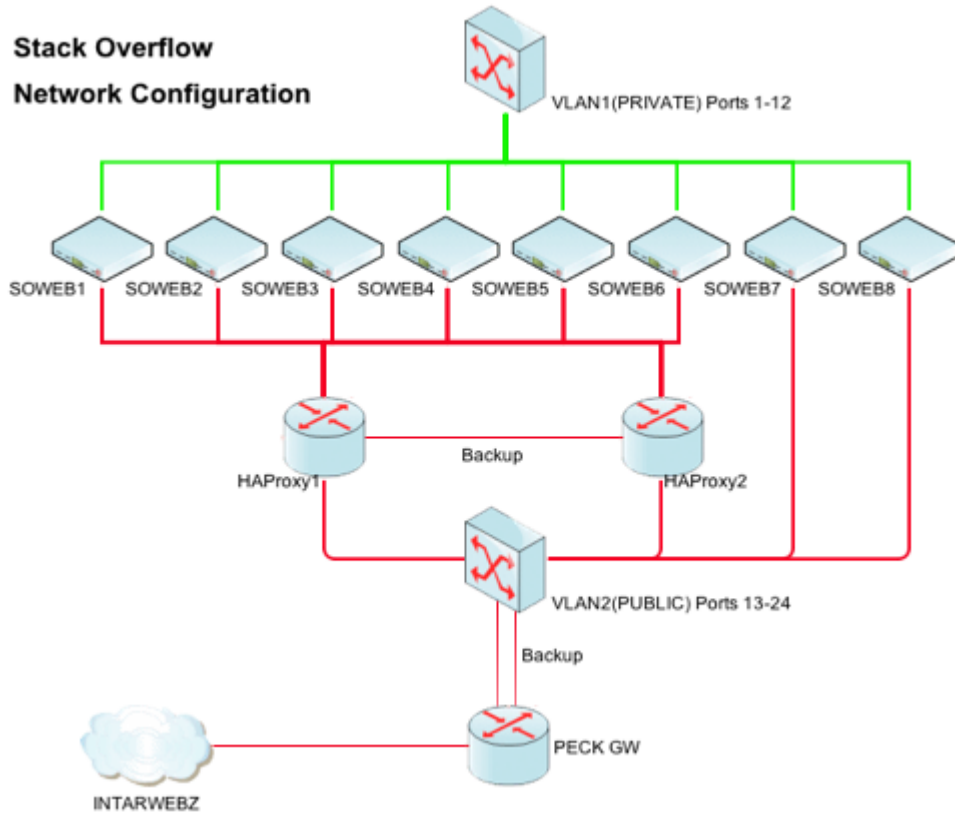
  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

It runs as below:



The View Components of TWaver HTML5

TWaver HTML5 defines some components including Network, Table, Tree and etc driving by DataBox. Network is GUI components to show all topology relations, Table and Tree are based on DataGrid and Tree in HTML5. All these components packed more properties and offer more convenient methods to let user use TWaver HTML5.



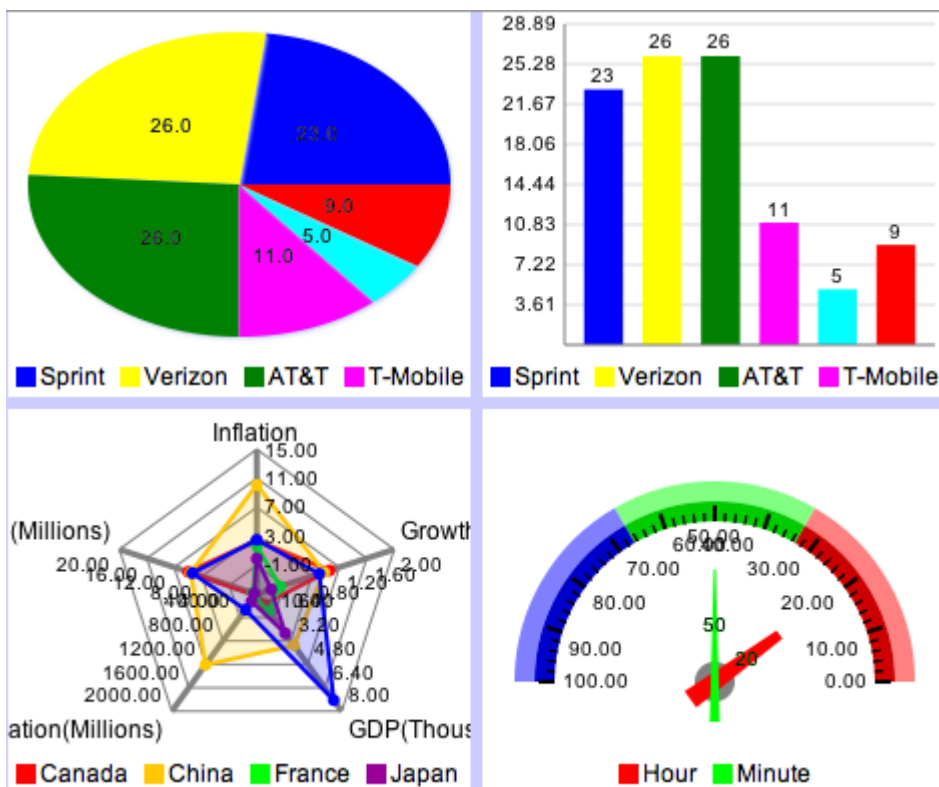
Network

Table

Mapping ID	Alarm Severity	Acked	Cleared	Raised Time
1	Indeterminate	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
2	Warning	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
3	Minor	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
4	Major	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
5	Critical	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
6	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07

Tree

- Alarm Demos
 - Alarm Statistics Demo
 - Alarm Mapping Demo
 - Alarm Propagation Demo
- Network Demos
 - Topology Demos
 - PSTN Demo
 - Bundle Demo
 - Auto Layout Demo
 - Spring Layout Demo
 - States Map Demo
 - Bus Layout Demo
 - Layer Vector Demo
 - Success Story Demo



Chart

- [Network Introduction](#)
- [Table Introduction](#)
- [Tree Introduction](#)

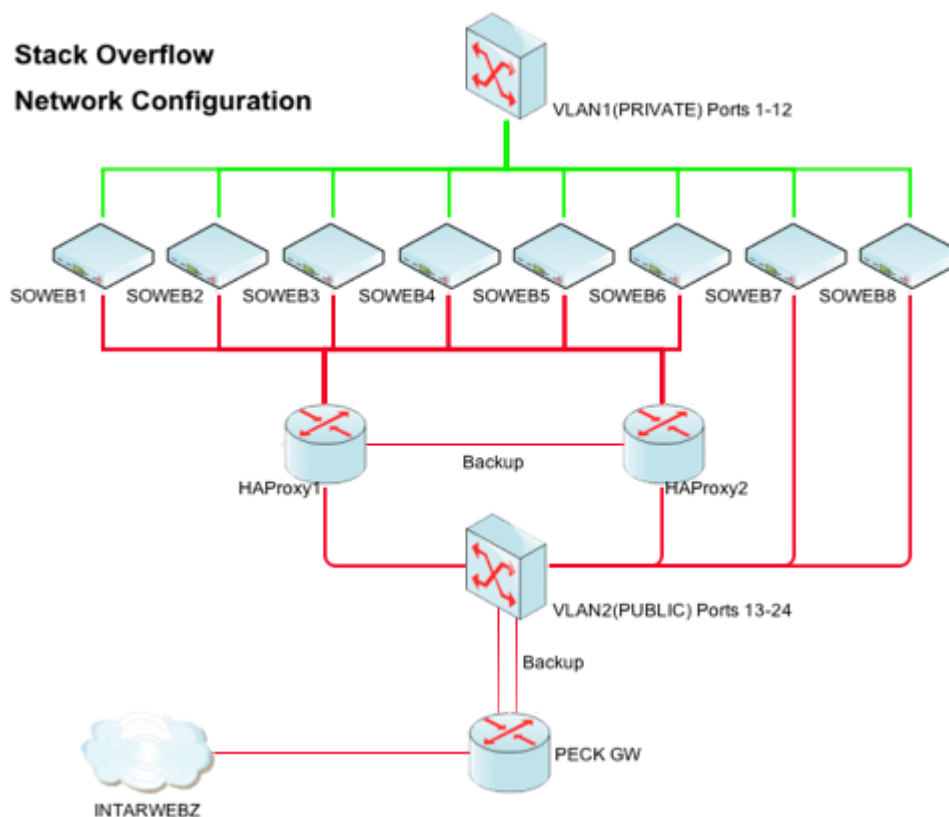
Network Introduction

• Network Overview

Network component is used to present network information in graphical way, has many functions such as element filter, SubNetwork switching, background supporting, zoom viewing, multi layers, attachment components showing and etc. It is the most intuitive and most shapely platform for network master system. More than that, Network still offers hundreds of properties and functions to let user to extend and configure, meet the special requirements.

Let's show one sample of Network to learn common features. The detail will be introduced in subsequence chapters.

We can make a topology picture as below easily by using Network:



• The Model and Hierarchy of Network Component

Network uses HTML DIV element as its implementation, it contains kinds of graphics element, such as <div>, . It also contains <canvas>, <video> element. Even supports HTML embed element of browse, such as flash, SVG. In addition, network has hierarchy structure: TopDiv, BackgroundDiv.

The hierarchy of Network:

```
-> rootDiv //root panel
-> topDiv //top panel
-> attachmentDiv //attachment panel
-> layerDiv //network view panel, to put ElementUI
-> layer n //the hierarchy of ElementUI is maintained by ElementBox.layerBox
-> layer ...
-> layer 0
```

```
-> bottomDiv //bottom panel
-> backgroundDiv //background panel
```

Network in Topology

Each Network element is to correspond with one ElementUI view component, Network express each view component by operating network element data. The method at below is the way to get correspond component of network element.

```
getElementUI : function(element)
```

Attachment Component

Network is not only related to ElementUI component, but also can be bounded with multi Attachment components such as network element tags, alarm bubbles. User can add new Attachment into network element to express itself more completely.

Attachment is a view component, it can be attached to Element, shows in Network component. Attachment can be shown in top, you can use showInAttachmentDiv property controls whether shows in attachment div.

```
//Attachment constructor function
twaver.network.Attachment = function (elementUI, showInAttachmentDiv)

twaver.network.Network#
//attachment canvas in network, Attachment components can be added into ElementUI component
//Attachment also can be detached with ElementUI component, can be added into attachment canvas directly which
will make sure attachment top show.
getAttachmentDiv: function ()

twaver.network.ElementUI
//ElementUI can get all attachments of network element
getAttachments: function ()
```

• Network Commonly Features

Switch Interactive Model

Switching interactive model in Network can make the interactive of different mouse events in different models come true. The most commonly use model is Default Model and Edit Model. User can handle mouse and key events by implementing Interaction as interface, thus, user can customize own interactive model. All these InteractionHandler can be used alternately and draw up together.

```
//switch to Default Interaction Model
setDefaultInteractions: function (lazyMode)
//switch to Edit Interaction Model
setEditInteractions: function (lazyMode)
//switch to Create Element Interaction Model
setCreateElementInteractions: function (type)d
//switch to Create Link Interaction Model
setCreateLinkInteractions: function (type)
//switch to Create ShapNode Interaction Model
```

```
setCreateShapeNodeInteractions: function (type)
```

Filter

```
network.visibleFunction = function(node){
    return node.getChildrenCount() > 0;
};
```

Some Properties Listed

There are hundreds of properties of Network, let's list parts of them:

```
getCurrentSubNetwork: function ()

//set current SubNetwork
getCurrentSubNetwork: function ()

//up into higher level SubNetwork
upSubNetwork: function (animate, finishFunction)

//mouse and keyboard add/remove interaction listener
addInteractionListener = function (listener, scope, ahead)
removeInteractionListener = function (listener)

//refresh view of network element
invalidateElementUI: function (element, checkAttachments)
invalidateElementUIs: function (checkAttachments)

//get view of network element
getElementUI: function (element)

//zoom operation
getZoom = function ()
setZoom = function (value, animate)
zoomIn = function (animate)
zoomOut = function (animate)
zoomReset = function (animate)

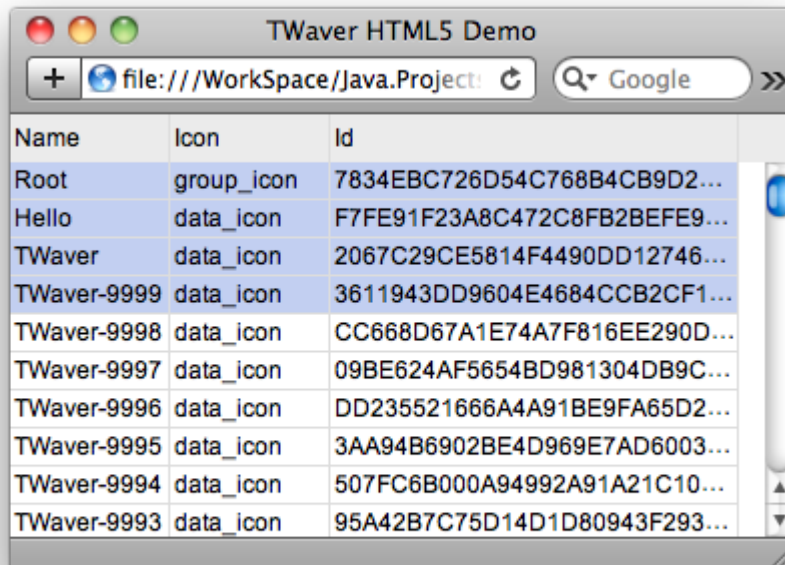
//get container of components in topology
getRootDiv: function ()
getTopDiv: function ()
getAttachmentDiv: function ()
getLayerDiv: function ()
getBottomDiv: function ()
getBackgroundDiv: function ()

//get view div of network
getView: function ()
```

Table Introduction

twaver.controls.Table is table component, it is bound with DataBox, shows data in databox. Every row data is a Element in databox, every column is a property of Element.

Following gives a table example:



Name	Icon	Id
Root	group_icon	7834EBC726D54C768B4CB9D2...
Hello	data_icon	F7FE91F23A8C472C8FB2BEFE9...
TWaver	data_icon	2067C29CE5814F4490DD12746...
TWaver-9999	data_icon	3611943DD9604E4684CCB2CF1...
TWaver-9998	data_icon	CC668D67A1E74A7F816EE290D...
TWaver-9997	data_icon	09BE624AF5654BD981304DB9C...
TWaver-9996	data_icon	DD235521666A4A91BE9FA65D2...
TWaver-9995	data_icon	3AA94B6902BE4D969E7AD6003...
TWaver-9994	data_icon	507FC6B000A94992A91A21C10...
TWaver-9993	data_icon	95A42B7C75D14D1D80943F293...

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();
      var box = table.getDataBox();

      var i = 10000;
      while(i-->0){
        data = new twaver.Data();
        data.setName("TWaver-" + i);
        data.setParent(root);
        box.add(data);
      }

      var tablePane = new twaver.controls.TablePane(table);
      createColumn(table,'Name', 'name', 'accessor', 'string');
      createColumn(table,'Id', 'id', 'accessor', 'string');
      createColumn(table,'Icon', 'icon', 'accessor');

      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
    }

    function createColumn(table, name, propertyName, propertyType, valueType) {
```

```
var column = new twaver.Column(name);
column.setName(name);
column.setPropertyName(propertyName);
column.setPropertyType(propertyType);
if (valueType) column.setValueType(valueType);
table.getColumnBox().add(column);
return column;
}
</script>
</head>
<body onload="init()" style="margin:0;" ></body>
</html>
```

Tree Introduction

twaver.controls.Tree is a view in TWaver. It can be bounded with any DataBox containers such as LayerBox, AlarmBox, ElementBox and etc. The hierarchy of Tree view is decided by the father-child relationships in DataBox container. Tree view own itself selection model, also can use the one of DataBox, in this way, it can be selected synchronization. A advantage of TWaver HTML5 Tree is High performance, it only renders visible data. It can be loaded 100,000 elements.

twaver.controls.Tree is pretty easy to use, once related to DataBox container, it can express the hierarchy of this container.

Filter:

```
tree.setVisibleFunction(function(data){
    return data.getName().length < 10;
});
```

Example:

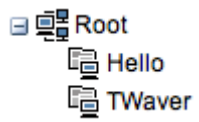
```
<!DOCTYPE html>
<html>
<head>
    <title>TWaver HTML5 Demo</title>
    <script type="text/javascript" src="../demo/twaver.js"></script>
    <script type="text/javascript">
        function init(){
            var tree = new twaver.controls.Tree();

            var box = tree.getDataBox();
            var root = new twaver.Data();
            root.setIcon("group_icon");
            root.setName('Root');
            box.add(root);
            var data = new twaver.Data();
            data.setParent(root);
            data.setName("Hello");
            box.add(data);
            data = new twaver.Data();
            data.setName("TWaver");
            data.setParent(root);
            box.add(data);

            var treeDom = tree.getView();
            treeDom.style.width = "100%";
            treeDom.style.height = "100%";
            document.body.appendChild(treeDom);

            tree.expandAll();
        }
    </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

The application run as below:



Data Serialization

TWaver HTML5 supports importing and exporting data container which facilitate data saving and transmission. TWaver HTML5 defined the serialize/deserialize methods by XML Serializer for DataBox to accomplish this job.

Construct a XML

```
var xmlSerializer = new twaver.XMLSerializer(box);
```

Serialize and Deserialize

```
serialize : function()
deserialize : function(xml, rootParent)
```

XML Serializer also can appoint properties to output or not. Let's give an example for not outputting Element ID:

```
xmlSerializer.settings.registerProperty("id",null);
```

The test application:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var network = new twaver.network.Network();

      var box = network.getElementBox();
      var node = new twaver.Node();
      node.setName("from");
      node.setLocation(100, 100);
      box.add(node);
      var node2 = new twaver.Node();
      node2.setName("to");
      node2.setLocation(300, 300);
      box.add(node2);

      var link = new twaver.Link(node, node2);
      link.setName("Hello TWaver");
      link.setToolTip("<b>Hello TWaver</b>");
      box.add(link);

      var xmlSerializer = new twaver.XMLSerializer(box);
      xmlSerializer.settings.registerProperty("id",null);
      alert(xmlSerializer.serialize());

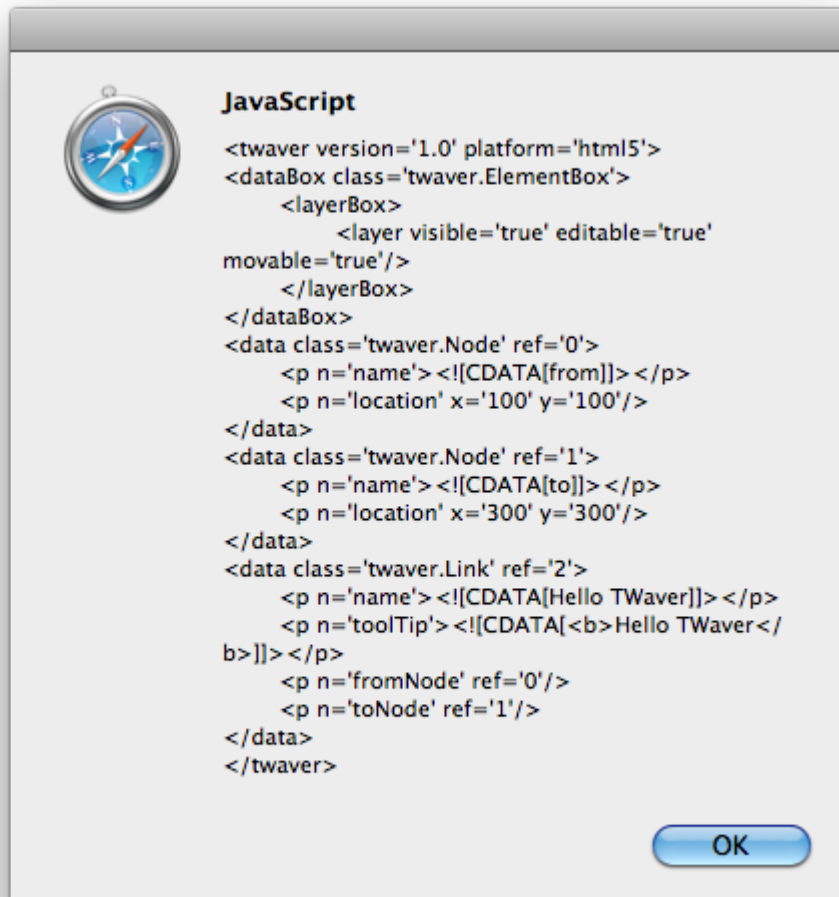
      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }
  </script>
</head>
</html>
```

```

</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>

```

The output interface:



Data Element

Data Element is the basic element of Data model. It is applied to describe graphic and business of network element or it is just the data added in network. All Data Element of TWaver HTML5 inherit `twaver.Data`. TWaver HTML5 pre-defined three kinds of data to meet with different requirements that `twaver.Element` is used for describing network element, `twaver.Alarm` is for describing alarm element and `twaver.Layer` is for describing layers in topology. Therein, `twaver.Element` extends several kinds of network element to express abundant characteristics of network, including basic network element, links, bus, equipment and etc.

This chapter introduces the peculiarities, usage and extended application of these network elements and other data elements.

- [Basic Data Element](#)
- [Alarm Data](#)
- [Layer Data Element](#)
- [Topology Element](#)

Basic Data Element

twaver.Data is the basic element of TWaver HTML Data Element. Data defined basic properties such as id, name, icon, toolTip, parent and children and etc, packed event dispatching methods, twaver.Element , twaver.Alarm , twaver.Layer inherit twaver.Data.

Data contains twaver.EventDispatcher class which gives it's the event dispatching and listening function. It realizes event dispatching and listening by following methods:

```
firePropertyChange: function (property, oldValue, newValue)
addPropertyChangeListener : function (listener, scope, ahead)
removePropertyChangeListener : function (listener)
onPropertyChanged : function(listener)
```

More, Data defined some basic properties including id, name, icon, toolTip and etc.

Thereinto, In Data container ID is the unique identification of data element. It can't be duplicated. TWaver HTML5 will set a unique identification for element when constructing Data, of course, user also can set value for ID, making sure the id can't be duplicated.

```
twaver.Data : function (id)

getId : function()
getName : function()
setName : function(value)
getIcon : function()
setIcon : function(value)
getToolTip : function()
setToolTip : function(value)
```

User can put up other properties for element by implementing setClient(...)method which is similar to HashMap in Java.

```
setClient = function (clientProp, newValue)
getClient = function (clientProp)
```

Data Element provides other function lists as below.

```
getChildren: function ()
getChildrenSize: function ()
toChildren: function (matchFunction, scope)
addChild: function (child, index)
removeChild: function (child)
getChildAt: function (index)
clearChildren: function ()
getParent: function ()
setParent: function (parent)
hasChildren: function ()
isRelatedTo: function (data)
isParentOf: function (data)
isDescendantOf: function (data)
```

Alarm Data

Each alarm defined by TWaver Alarm own level which is used for reacting degree of urgency. AlarmBox maintain the alarm element, and associate alarm with the network element in topology. Network element doesn't store specific alarm but store current alarm states information.

Alarm

Alarm is the data model of equipment fault and network exception in webmaster system, connected with Element to express alarm information of network element. Alarm pre-defined alarm levels, alarm cleared or not, alarm acknowledged or not and ID of alarm network element, More, user can add other properties by implementing setClient(...) method.

The properties of Alarm:

```
twaver.Alarm : function (alarmID, elementID, isAked, isCleared)
getElementID : function()
isAked : function()
setAked : function(value)
isCleared : function()
setCleared : function(value)
getAlarmSeverity : function()
setAlarmSeverity : function(value)
```

Alarm Severity

Alarm Severity is used for reacting degree of urgency. TWaver HTML5 pre-defined six alarm levels, the larger of default value, the more critical this alarm is.

```
twaver.AlarmSeverity : function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.CRITICAL = twaver.AlarmSeverity.add(500, "Critical", "C", '#FF0000');
twaver.AlarmSeverity.MAJOR = twaver.AlarmSeverity.add(400, "Major", "M", '#FFA000');
twaver.AlarmSeverity.MINOR = twaver.AlarmSeverity.add(300, "Minor", "m", '#FFFF00');
twaver.AlarmSeverity.WARNING = twaver.AlarmSeverity.add(200, "Warning", "W", '#00FFFF');
twaver.AlarmSeverity.INDETERMINATE = twaver.AlarmSeverity.add(100, "Indeterminate", "N", '#C800FF');
twaver.AlarmSeverity.CLEARED = twaver.AlarmSeverity.add(0, "Cleared", "R", '#00FF00');
```

Severity	Letter	Value	Color
CRITICAL	C	500	Red
MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

All levels of Alarm are static variables, user can register or unregister alarm levels in overall situation. In addition, TWaver HTML5 offers method to clear all alarm levels (Pay attention to use clearing all levels method, it will effect whole program).

```
twaver.AlarmSeverity.add : function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.remove : function (name)
twaver.AlarmSeverity.clear : function ()
```

Alarm State

A project can appear thousands of alarm at one time in practical terms. Obviously network element connected with alarm directly is very heavy in the alarm storm, So TWaver HTML5 detached network element and alarm element. Network element save the alarm state only including how much alarm it has, what is the highest level and etc. The detail information of alarm is stored in AlarmBox. How to response alarm storm and how to use AlarmBox will be introduced in following chapter.

Alarm state is defined by twaver.AlarmState, is to express the level and quantity of new alarm issued.

Alarm state properties including highest level of acknowledged alarms, highest level of new alarms, highest level of all alarms, the highest but one of new alarms, highest level of self alarm, transmission alarm level and quantity of each alarm level.

```
getHighestAcknowledgedAlarmSeverity: function ()
getHighestNewAlarmSeverity: function ()
getHighestOverallAlarmSeverity: function ()
getHighestNativeAlarmSeverity: function ()
hasLessSevereNewAlarms: function ()
getAcknowledgedAlarmCount: function (severity)
getAlarmCount: function (severity)
getNewAlarmCount: function (severity)
setNewAlarmCount: function (severity, count)
getPropagateSeverity: function ()
setPropagateSeverity: function (propagateSeverity)
isEmpty: function ()
isEnabledPropagation: function ()
setEnabledPropagation: function (enablePropagation)
```

The methods to modify alarm state: acknowledge alarm, clear alarm, add/decrease acknowledged alarm, remove alarms and etc.

```
acknowledgeAlarm: function (severity)
acknowledgeAllAlarms: function (severity)
increaseAcknowledgedAlarm: function (severity, increment)
increaseNewAlarm: function (severity, increment)
decreaseAcknowledgedAlarm: function (severity, decrement)
decreaseNewAlarm: function (severity, decrement)
removeAllNewAlarms: function (severity)
setAcknowledgedAlarmCount: function (severity, count)
removeAllAcknowledgedAlarms: function (severity)
clear: function ()
```

The Usage of Alarm

User needs to pay attention to alarm adding and removing, it should be operated via AlarmBox which is consistent with adding and removing Element via ElementBox.

For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo - Alarm</title>
  <script type="text/javascript" src="../demo/twaver.js"> </script>
  <script type="text/javascript">
    function init(){
      var network = new twaver.network.Network();

      var box = network.getElementBox();
      var node = new twaver.Node();
      node.setName("from");
      node.setLocation(100, 100);
      box.add(node);

      addAlarm("alarm 1",node.getId(),twaver.AlarmSeverity.CRITICAL, box.getAlarmBox());

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }
    function addAlarm(alarmID,elementID,alarmSeverity,alarmBox){
      var alarm = new twaver.Alarm(alarmID,elementID,alarmSeverity);
      alarmBox.add(alarm);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>
```



Layer Data Element

Layer is used for describing layer information of network element. Layer owns three special properties including visible, editable and movable.

```
getVisible : function()
setVisible : function(value)
getMovable : function()
setMovable : function(value)
getEditable : function()
setEditable : function(value)
```

The hierarchy of TWaver HTML5 is maintained by LayerBox, the default order is decided by father-child relations and the join sequence. Each element connected with certain layer by setting layer ID, thus, all network elements in topology will be shown in hierarchy.

Let's explain how to use layer and its three properties: the up-down move of layer will be explained more in LayerBox.

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo - Layer</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var layerBox, box;
    function init() {
      var network = new twaver.network.Network();

      box = network.getElementBox();
      layerBox = box.getLayerBox();

      var layer1 = new twaver.Layer("unmovable", "unmovable layer");
      layer1.setMovable(false);
      var layer2 = new twaver.Layer("uneditable", "uneditable layer");
      layer2.setEditable(false);
      var layer3 = new twaver.Layer("invisible", "invisible layer");
      layer3.setVisible(false);

      layerBox.add(layer1);
      layerBox.add(layer2);
      layerBox.add(layer3, 0);

      createNode(layer1, "circle", 10, 40, 100, 100, "#ff0000");
      createNode(layer2, "diamond", 30, 60, 100, 100, "#00ff00");
      createNode(layer3, "rectangle", 50, 80, 100, 100, "#0000ff");
      createNode(layerBox.getDefaultLayer(), "rectangle", 70, 20, 150, 150, "#808080");

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);

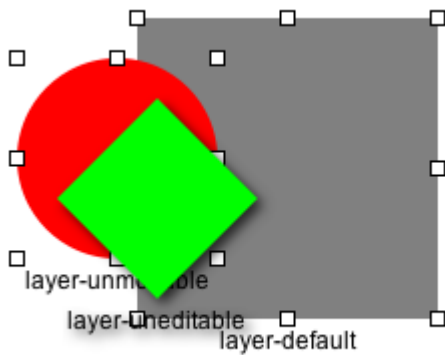
      network.setEditInteractions();
    }
    function createNode(layer, shape, x, y, width, height, fillColor) {
      var node = new twaver.Node();
      node.setLayerId(layer.getId());
      node.setName("layer-" + layer.getId());
      node.setStyle("body.type", "vector");
```



```

node.setStyle("vector.fill.alpha", 0.7);
node.setStyle("vector.shape", shape);
node.setSize(width, height);
node.setLocation(x, y);
node.setStyle("vector.fill.color", fillColor);
box.add(node);
return node;
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```



Topology Element

twaver.Element in TWaver HTML5 stands for topology element, it is the most important data. Topology element shows in Topology Network, it contains Dummy, Node and Link.

Dummy

Dummy is not visible in topology, but visible in the tree component. Generally dummy will be parent node of other nodes to classify and be in category. For example, put all links network element under one dummy, that mean it's the link category.

Node

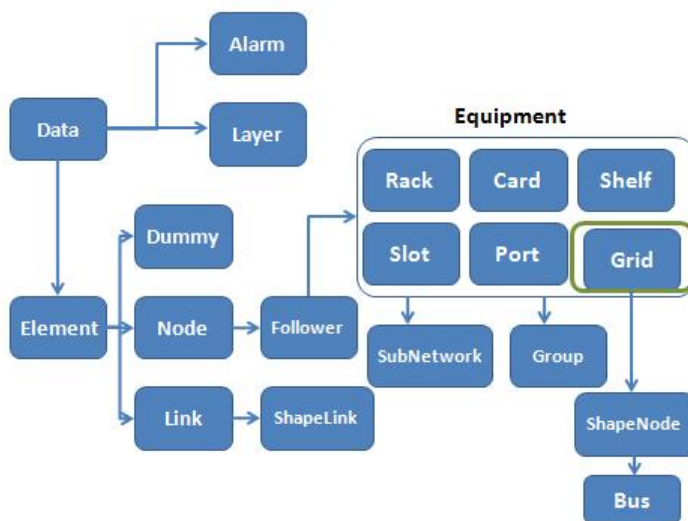
Node is the most commonly used network element. It stands for entity object including nodes, group, subNetwork, equipment and etc.

Link

Link stands for connection relations between nodes. ShapeLink extends Link, is used for representing irregular trend links.

Data structure

The following graph is topology element structure, equipment element is provided in TWaver HTML5 Demo.



Element

Element inherits Data class, extends properties such as alarmState, layerID, elementUIClass and etc to represent alarm state, layer ID, view type of network element correspondingly.

```

getLayerID : function()
setLayerID : function(layerId)
getAlarmState: function ()
getElementUIClass: function ()
  
```

Thereinto elementUIClass is the ElementUI class or extends ElementUI. Different network element corresponds with certain ElementUI, for example, twaver.Node corresponds with twaver.network.NodeUI, twaver.Grid to twaver.network.GridUI.

ElementUI is the view component of network element in topology, these two points construct a data view separation model. The detail will be introduced in view component chapter.

Element inherits Style interface, defined get/setStyle() methods to set network element style including color, border, background, alignment and etc. Generally different network element has different style properties, Please reference to stylesheet for detail.

```
setStyle : function(styleProp, newValue)
getStyle : function(styleProp, returnDefaultIfNull)
```

Set label color to be red for network element:

```
var node = new twaver.Node();
node.setStyle("label.color", "#ff0000");
node.setName("Node A");
```

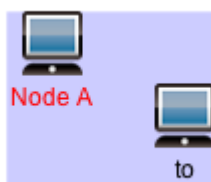
Run interface as follow:



Otherwise, Element defined isAdjustedToBottom():Boolean method to represent whether view in bottom, the default value is false.

```
isAdjustedToBottom: function ()
```

twaver.Group overwrite this method, and the return value is true which mean element will be in bottom. Thus, it will not cover the child node but locate at the bottom of this node:



- [twaver.Node](#)
- [twaver.Link](#)
- [twaver.Follower](#)
- [twaver.SubNetwork](#)
- [twaver.Group](#)
- [twaver.ShapeNode](#)
- [twaver.Grid](#)

twaver.Node

twaver.Node implements twaver.Element, stands for node in topology. Generally node represent entity object. It can be the endpoint of link, and its location and size can be fixed by x, y, width and height properties. It also has picture property.

Register Image

Notice that every image must be registered before using as Node image.

```
twaver.Util.registerImage : function (name, source, width, height)
```

Example

```
twaver.Util.registerImage('loading', 'images/loading.gif', 32, 32);
var node3 = new twaver.Node();
node3.setImage('loading');
```

Get Image size

Notice that you must give image's width and height when registering image, such as 32*32. Of source there has some unknown condition, we can use the following to handle it. Firstly create an Image class, listening the image's loading event. We can get image's width and height after loading image, then register the image.

```
registerImage: function (url) {
    var image = new Image();
    image.src = url;
    image.onload = function () {
        twaver.Util.registerImage(demo.Util.getImageName(url), image, image.width, image.height);
        image.onload = null;
        if (window.network) {
            window.network.invalidateElementUIs();
        }
        if (window.tree) {
            window.tree.invalidateDisplay();
        }
    };
}
```

For example:

```
<!DOCTYPE html>
<html>
<head>
    <title>TWaver HTML5 Demo</title>
    <script type="text/javascript" src="../demo/twaver.js"></script>
    <script type="text/javascript">
        function init() {
            registerImage("images/twaver.png", "twaver");

            var network = new twaver.network.Network();

            var box = network.getElementBox();
```

```

var node = new twaver.Node();
node.setImage("twaver");
node.setName("from");
node.setLocation(100, 100);
box.add(node);

var node2 = new twaver.Node();
node2.setName("to");
node2.setLocation(200, 200);
box.add(node2);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
function registerImage(url, name) {
var image = new Image();
image.src = url;
image.onload = function() {
twaver.Util.registerImage(name, image, image.width, image.height);
image.onload = null;
if (window.network) {
window.network.invalidateElementUIs();
}
if (window.tree) {
window.tree.invalidateDisplay();
}
}
};
}
</script>
</head>
<body onload="init()" style="margin:0;" ></body>
</html>

```



Related properties and methods for the location and size of node

```

get/setX : function(x)
get/setY : function(y)
get/setCenterLocation : function(point)
get/setLocation : function(point)
get/setWidth : function(width)
get/setHeight : function(height)
translate: function (x, y)

```

Get the connected links methods: if there is no links for this node, it will return null.

```
getLoopedLinks: function ()  
getLinks: function ()  
getAgentLinks: function ()  
getFromLinks: function ()  
getToLinks: function ()  
hasAgentLinks: function ()  
getFromAgentLinks: function ()  
getToAgentLinks: function ()
```

In addition, Node can add followers which can be moved with node together. This will be introduced in following chapters.

```
getFollowers: function ()
```

twaver.Link

Link generally stands for connection between nodes with start point and end point. TWaver supports loop link that the start node and the end node are the same one.

Start Node, End Node and Link

```
get/setFromNode: function (fromNode)
get/setToNode: function (toNode)
```

Start Node Agent and End Node Agent

The node connecting with link directly named fromNode, toNode in TWaver HTML5. If the start or end node is in one Group, It will look like one group connect with one node when the group is in combination state, in this situation the group will be agent node.

```
getFromAgent: function ()
getToAgent: function ()
```

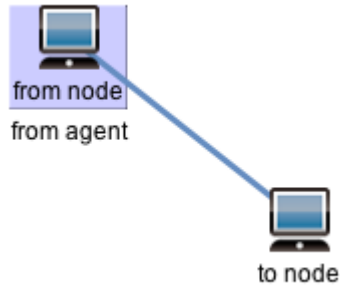
For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"> </script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var node1=new twaver.Node();
  node1.setLocation(20,20);
  node1.setName("from node");
  var node2=new twaver.Node();
  node2.setLocation(150,50);
  node2.setName("to node");
  var group=new twaver.Group();
  group.addChild(node1);
  group.setName("from agent");
  var link6=new twaver.Link(node1,node2);
  box.add(node1);
  box.add(node2);
  box.add(group);
  box.add(link6);

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
```

```
</html>
```



Self-Loop, The Start and End Point of This Link Share the Same Node

```
isLooped: function ()
```

For example:

```
var node = new twaver.Node();
box.add(node);
var link=new twaver.Link(node, node);
link.setName("Looped Link");
box.add(link);
```



The Expend and Bindle of Link

When there are multi links between nodes, TWaver supports expending and bundle links. Double click one link can realize the states switchover by default. The visible link when links binding together is named bundle agent, the default agent will be the first link. User can set all these default setting.

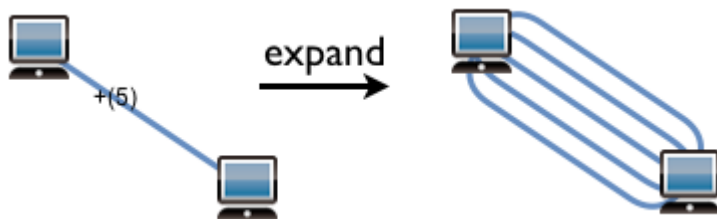
```
getBundleLinks: function ()
getBundleCount: function ()
getBundleIndex: function ()
reverseBundleExpanded: function ()
isBundleAgent: function ()
```

For example:

```
var from = new twaver.Node();
from.name = "from";
from.location = {x:20, y:20};
box.add(from);
var to = new twaver.Node();
```



```
to.name = "to";
to.location = {x:150, y:60};
box.add(to);
var link = new twaver.Link(from,to);
link.name = "bundle agent";
box.add(link);
var link2=new twaver.Link(from,to);
box.add(link2);
var link3=new twaver.Link(from,to);
box.add(link3);
var link4=new twaver.Link(from,to);
box.add(link4);
var link5=new twaver.Link(from,to);
box.add(link5);
```



- [Links Binding](#)
- [Links Type](#)

Links Binding

The chapter of twaver.Link introduces links binding and expending in brief. We will introduce functions such as the gap between extended links, binging in group and self-loop binding and etc.

Normal Links Binding(not self-loop)

TWaver defined six bundle properties:

"link.bundle.id" : The identification of bingding, links will be in same group with same ID
 "link.bundle.independent" : Whether the bundle is independent, whether bundle independently when has multi link groups
 "link.bundle.gap" : The gap between links
 "link.bundle.offset" : The link offset from endpoint
 "link.bundle.enable" : Whether join the binding or not
 "link.bundle.expanded" : Whether extended links, **false** stands **for** binding

For example: set two boudle links with ID "1" and "2"

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
var box;
function init() {
  var network = new twaver.network.Network();
  box = network.getElementBox();

  var from = new twaver.Node();
  from.setName("from");
  from.setLocation({
    x : 20,
    y : 20
  });
  box.add(from);
  var to = new twaver.Node();
  to.setName("to");
  to.setLocation({
    x : 150,
    y : 60
  });
  box.add(to);

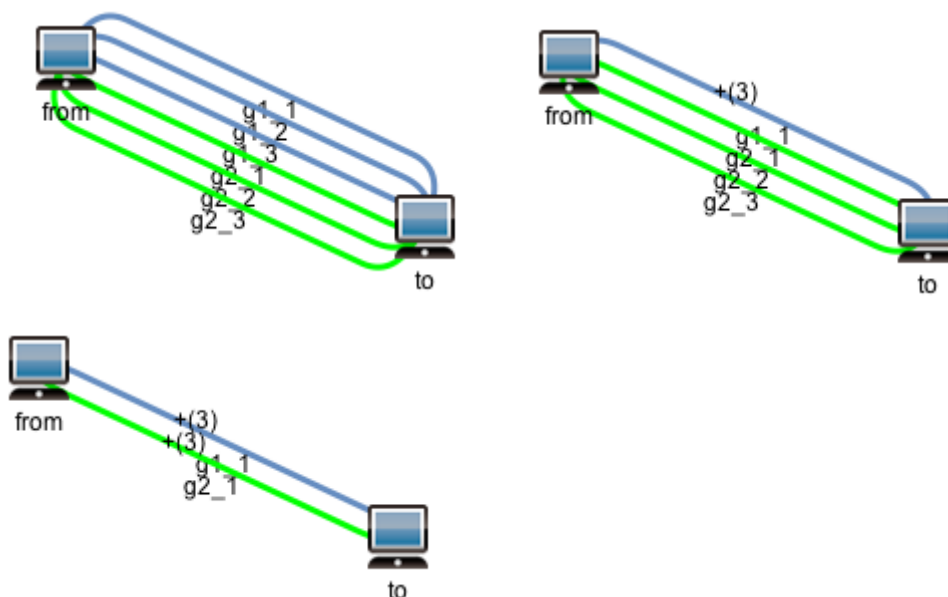
  createLink(from, to, "g1_1", 1);
  createLink(from, to, "g1_2", 1);
  createLink(from, to, "g1_3", 1);

  createLink(from, to, "g2_1", 2, "#00ff00");
  createLink(from, to, "g2_2", 2, "#00ff00");
  createLink(from, to, "g2_3", 2, "#00ff00");

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
```

```
function createLink(from, to, name, groupID, color, type, groupIndependent,
    gap, offset, bundleEnable) {
    var link = new twaver.Link(from, to);
    link.setName(name);
    if (type) {
        link.setStyle("link.type", type);
    }
    if (color) {
        link.setStyle("link.color", color);
    }
    if (groupID >= 0) {
        link.setStyle("link.bundle.id", groupID);
    }
    if (gap > 0) {
        link.setStyle("link.bundle.gap", gap);
    }
    if (offset > 0) {
        link.setStyle("link.bundle.offset", offset);
    }
    link.setStyle("link.bundle.independent", groupIndependent);
    link.setStyle("link.bundle.enable", bundleEnable);
    box.add(link);
    return link;
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>
```

Run application as follows:



Let's set one bundle binding independently, and set different link type

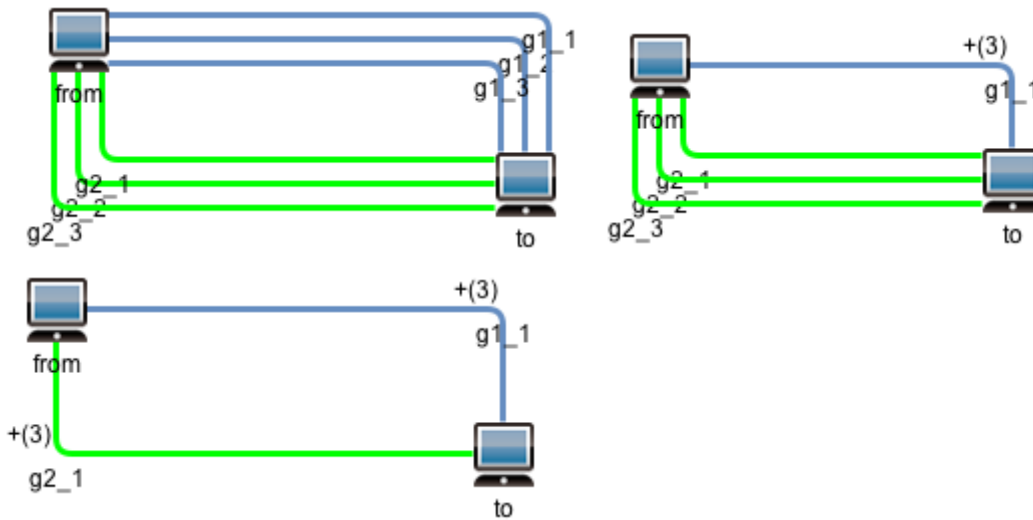
```
createLink(from, to, "g1_1", 1, null, "orthogonal.H.V");
createLink(from, to, "g1_2", 1, null, "orthogonal.H.V");
createLink(from, to, "g1_3", 1, null, "orthogonal.H.V");

createLink(from, to, "g2_1", 2, "#00ff00", "orthogonal.V.H", true);
createLink(from, to, "g2_2", 2, "#00ff00", "orthogonal.V.H", true);
createLink(from, to, "g2_3", 2, "#00ff00", "orthogonal.V.H", true);
```

```
}

```

Run as follows:



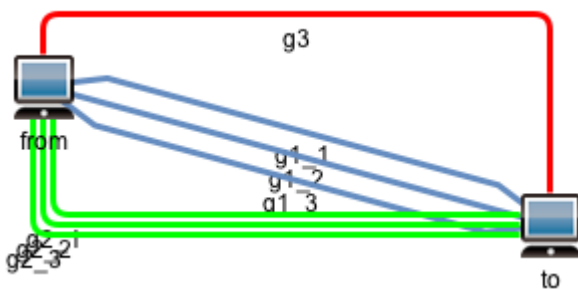
Let's have a try for parameters link.bundle.gap and link.bundle.offset. In addition, let's add one link without binding by setting link.bundle.enable parameter.

```
createLink(from, to, "g1_1", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_2", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_3", 1, null, "triangle", null, null, 30);

createLink(from, to, "g2_1", 2, "#00ff00", "orthogonal.V.H", true, 5);
createLink(from, to, "g2_2", 2, "#00ff00", "orthogonal.V.H", true, 5);
createLink(from, to, "g2_3", 2, "#00ff00", "orthogonal.V.H", true, 5);

createLink(from, to, "g3", 2, "#ff0000", "extend.top", null, null, null, false);
```

Run as follows:



Self-Loop Binding

TWaver HTML5 supports self-loop link to express the native connection relations of network element. It controls by a set of parameters on its own.

```
link.looped.gap : the gap of self-loop, 12 is the default value
link.looped.direction : the direction of links including east,south,west,north and etc eight values.
'northwest', 'north', 'northeast', 'east', 'west', 'south', 'southwest', 'southeast'
```

link.looped.type : the type of self-loop including circle and rectangle, 'arc', 'rectangle'

The example as below create three self-loop links, the gap is 20, and the type is rectangle

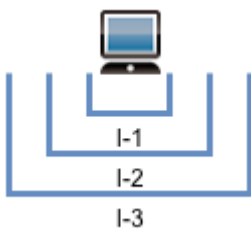
```
function init() {
    var network = new twaver.network.Network();
    var box = network.getElementBox();

    var from = new twaver.Node();
    from.setLocation({
        x : 20,
        y : 20
    });
    box.add(from);

    var i = 3;
    while(i>0){
        var link = new twaver.Link(from, from);
        link.setName("I-" + i);
        link.setStyle("link.looped.gap", 20);
        link.setStyle("link.looped.direction", "south");
        link.setStyle("link.looped.type", "rectangle");
        box.add(link);
        i--;
    }

    var networkDom = network.getView();
    networkDom.style.width = "100%";
    networkDom.style.height = "100%";
    document.body.appendChild(networkDom);
}
```

Run as follows:



Links Type

The links direction of TWaver HTML5 has two ways: One of is `twaver.Link` which decided the links direction by setting parameters, another is `twaver.ShapeLink` which decided by a series of control points. If from node and two node is the same, it means that node A link to node A, this time we said this link is "Loop Link".

`twaver.Link`

Link has several types, mainly has three kinds.

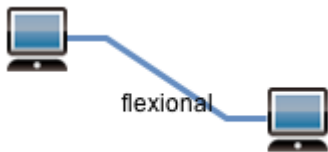
Straight

It has three types: 'arc', 'triangle', 'parallel'



Flexional

This kind of link has three types: 'flexional', 'flexional.horizontal', 'flexional.vertical'



Orthogonal

This link uses one point to control link path, it has the following types: 'orthogonal', 'orthogonal.horizontal', 'orthogonal.vertical', 'orthogonal.H.V', 'orthogonal.V.H', 'extend.top', 'extend.left', 'extend.bottom', 'extend.right'

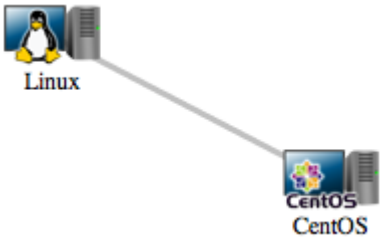

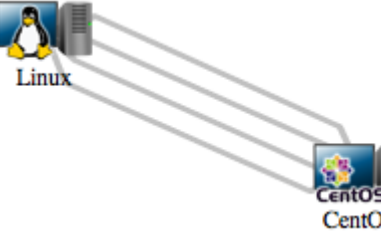
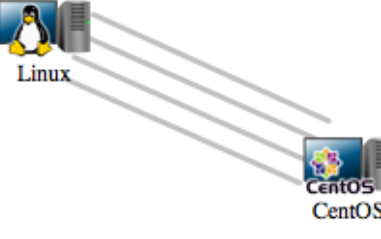
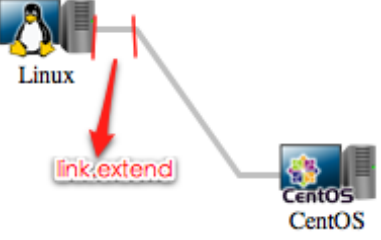
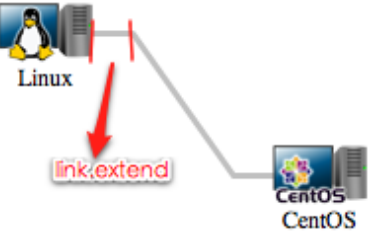


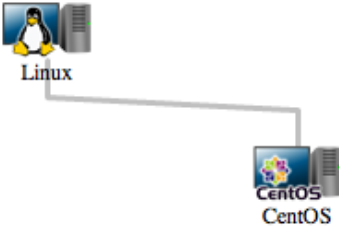
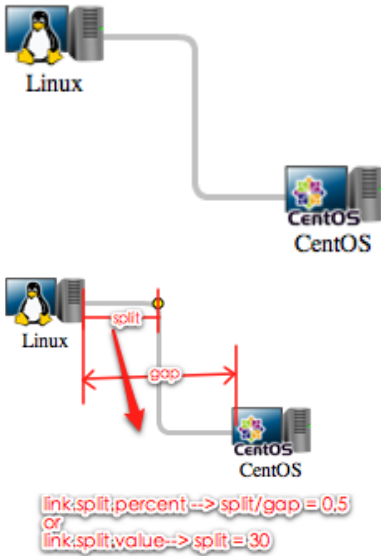
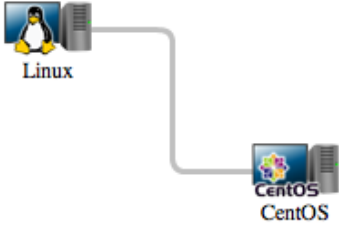


Set link types

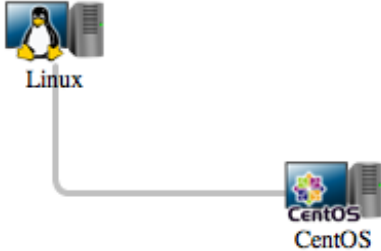

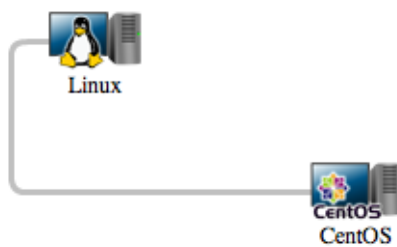

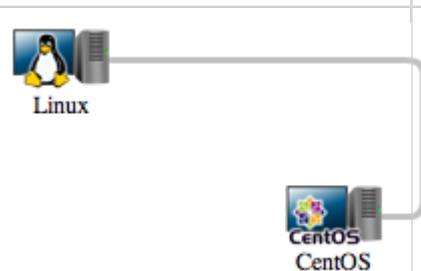
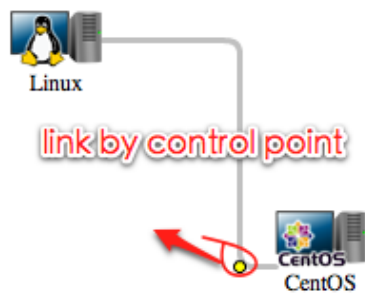
```
link.setStyle('link.type', 'orthogonal');
```

The table below lists all kinds of link types and corresponding control parameters

Link Type	View	Control Parameters
-----------	------	--------------------

<p>Basic Type is to connect start and end node directly. When there are more than one links between two nodes, TWaver classify three types by links expanding effects: arc, triangle, parallel These types only works when there are multi links between two nodes.</p>		<p>link.from.position Default center link.from.xoffset Default 0 link.from.yoffset Default 0 link.to.xoffset Default 0 link.to.yoffset Default 0</p>
<p>arc The knee point present as arc when links extend.</p>		<p>link.bundle.offset The knee point offset, default value is 20 link.bundle.gap The gap between links, default value is 12</p>
<p>triangle The knee point at right angle when links extend.</p>		<p>The same as above</p>
<p>parallel Links present as parallel lines when links extend.</p>		<p>The same as above</p>
<p>flexional This kind of link has three directions: Automatic direction: Value as horizontal direction when gap is bigger in X-axis. Horizontal direction: flexional.horizontal Vertical direction: flexional.vertical</p>		<p>link.from.at.edge Links to edge of start point Default value is true link.to.at.edge Links to edge of end point Default value is true link.extend Default value of extended gap is 20</p>
<p>flexional.horizontal</p>		<p>The same as above</p>

flexional.vertical		The same as above
Orthogonal Line Links Type		
<p>orthogonal</p> <p>Orthogonal Line</p> <p>This kind of link has three directions:</p> <p>Automatic direction: Value as horizontal direction when gap is bigger in X-axis.</p> <p>Horizontal direction: orthogonal.horizontal</p> <p>Vertical direction: orthogonal.vertical</p>	 <p>link:split:percent ==> split/gap = 0.5 or link:split:value ==> split = 30</p>	<p>link.from.at.edge Links to edge of start point Default value is true</p> <p>link.to.at.edge Links to edge of end point Default value is true</p> <p>link.split.by.percent Whether split by percent Default value is true</p> <p>link.split.percent The percent gap of split to start point Default value is 0.5 Set this property if split by percent</p> <p>link.split.value The split offset to start point Default value is 20 Set this property if split by offset</p>
orthogonal.horizontal		The same as above
orthogonal.vertical		The same as above
orthogonal.vertical From start point, align horizontal direction at first then vertical.		<p>link.from.at.edge Links to edge of start point Default value is true</p> <p>link.to.at.edge Links to edge of end point Default value is true</p>

vertical.horizontal From start point, align vertical directionat first then horizontal		The same as above
extend.left Extend upwards The extended value is split point to topmost		link.extend The extended value, Default value is 20
extend.left Extend left The extended value is split point to left-most		The same as above
extend.bottom Extend downwards The extended value is split point to bottom-most		The same as above
extend.right Extend right The extended value is split point to right-most		The same as above
All orthogonal lines above are control by control points and decide theirs direction. If these control points are set value at first, TWaver will graphic the split point by these control point priority.		LINK_CONTROL_POINT Control point This control point decided the split point of link The direction of links is decided by link type Automaticdirection: orthogonal Horizontal direction: orthogonal.horizontal Vertical direction:

		orthogonal.vertical
--	--	---------------------

twaver.Follower

The follower attaches itself to host node. Generally it is used for equipment panel such as ports rely on card.

Set Host Node

One node can be attached more than one followers

```
//get host node
get/setHost : function(host)
//Whether attach itself to certain node
isHostOn : function(node)
```

Looped Host

Moreover, followers can be attached to each other which mean they can be host and follower for each other.

```
//Whether looped host on each other
isLoopedHostOn : function(node)
```

For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var node=new twaver.Node();
  node.setLocation(20,20);
  node.setName("Host");
  box.add(node);

  var follower = new twaver.Follower();
  follower.setLocation(50,50);
  follower.setHost(node);
  follower.setName("Follower");
  box.add(follower);

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```



twaver.SubNetwork

twaver.SubNetwork stands for part of Net. TWaver HTML5 can switch SubNetwork in topology, and the components of interface are the network elements of current SubNetwork.

twaver.SubNetwork is the implement class of ISubNetwork. In topology double click can enter into SubNetwork, double click background can go back upper SubNetwork. The Null value of current SubNetwork stands for it is the highest SubNetwork.

Switch SubNetwork in Network:
twaver.network.Network

```
//Set current SubNetwork
get/setCurrentSubNetwork : function(subnetwork, animate)
//Go back to upper SubNetwork
upSubNetwork : function(animate)
```

For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var node=new twaver.Node();
  node.setLocation(20,20);
  var node2=new twaver.Node();
  node2.setLocation(150,50);
  var link=new twaver.Link(node,node2);

  var node=new twaver.Node();
  node.setName("node in subnetwork A");
  node.setLocation(100,50);
  var subNetworkA=new twaver.SubNetwork();
  subNetworkA.addChild(node);
  subNetworkA.setName("Subnetwork A");
  subNetworkA.setLocation(20,20);

  var node2=new twaver.Node();
  node2.setName("node in subnetwork B");
  node2.setLocation(100,50);
  var subNetworkB=new twaver.SubNetwork();
  subNetworkB.addChild(node2);
  subNetworkB.setName("Subnetwork B");
  subNetworkB.setLocation(120,20);

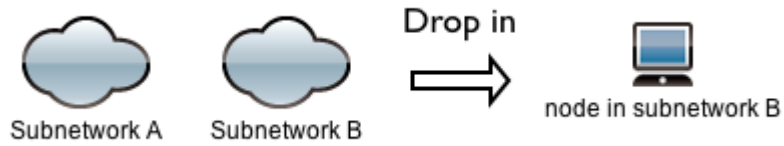
  box.add(node);
  box.add(subNetworkA);
  box.add(node2);
  box.add(subNetworkB);

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
```

```

}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>

```



twaver.Group

Generally Group contains multi child-network element, display group and its child-network when extends group, show icon of group network element when combine group.

The effect drawing of group extending and combining show as below:

Double click can extend or combine group by default.



For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();

      var node=new twaver.Node();
      node.setLocation(20,20);
      var node2=new twaver.Node();
      node2.setLocation(150,50);
      var link=new twaver.Link(node,node2);

      var childGroup=new twaver.Group();
      childGroup.addChild(node);
      childGroup.addChild(node2);
      childGroup.addChild(link);
      childGroup.setStyle("group.fill.color", "#ff0000");
      childGroup.setName("Child Group");
      childGroup.setExpanded(true);

      var group=new twaver.Group();
      group.addChild(childGroup);
      group.setStyle("group.fill.color", "#00ff00");
      group.setName("Group");
      group.setExpanded(true);

      box.add(node);
      box.add(node2);
      box.add(link);
      box.add(childGroup);
      box.add(group);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }
  </script>
</head>
```

```
<body onload="init()" style="margin:0;" > </body>  
</html>
```


twaver.ShapeNode

The location and figure of twaver.ShapeNode are decided by a series of control points. It is commonly used for representing irregular graphics such as coverage area, map block and etc.

Related methods to add/remove control point:

```
get/setPoint : function(points)
addPoint : function(point)
addPointAt : function(point, index)
setPointAt : function(point, index)
removePoint : function(point)
```

In addition, TWaver can offer methods to draw segments which will express ShapeNode more enrichment such as curve (QuadTo), interruption (moveTo) and etc.

TWaver has three drawing methods: moveTo, lineTo, QuadTo

```
"moveto", "lineto", "quadto"
```

Set segments
twaver.ShapeNode

```
get/setSegments : function(list)
```

Let's give an example: Create five control points at first, then set drawing method for each segment. Move the first control to draw curve, then draw until the end.

```
var shapeNode2 = new twaver.ShapeNode();
shapeNode2.addPoint({
    x : 30,
    y : 10
});
shapeNode2.addPoint({
    x : 80,
    y : 10
});
shapeNode2.addPoint({
    x : 100,
    y : 90
});
shapeNode2.addPoint({
    x : 10,
    y : 90
});
shapeNode2.addPoint({
    x : 30,
    y : 10
});
shapeNode2.setStyle("vector.fill.color", "#00ff00");
shapeNode2.setLocation(150, 10);

var segments = new twaver.List();
segments.add("moveto");
segments.add("quadto");
```

```
segments.add("quadto");
shapeNode2.setSegments(segments);
shapeNode2.setName("ShapeNode with Segments");
box.add(shapeNode2);
```



ShapeNode



ShapeNode with Segments



Note: One curve segment needs two control points

The whole program of this application:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var shapeNode=new twaver.ShapeNode();
  shapeNode.addPoint({x:30, y:10});
  shapeNode.addPoint({x:80, y:10});
  shapeNode.addPoint({x:100, y:90});
  shapeNode.addPoint({x:10, y:90});
  shapeNode.addPoint({x:30, y:10});
  shapeNode.setStyle("vector.fill.color", "#ff0000");
  shapeNode.setName("ShapeNode");
  box.add(shapeNode);

  var shapeNode2=new twaver.ShapeNode();
  shapeNode2.addPoint({x:30, y:10});
  shapeNode2.addPoint({x:80, y:10});
  shapeNode2.addPoint({x:100, y:90});
  shapeNode2.addPoint({x:10, y:90});
  shapeNode2.addPoint({x:30, y:10});
  shapeNode2.setStyle("vector.fill.color", "#00ff00");
  shapeNode2.setLocation(150, 10);

  var segments=new twaver.List();
  segments.add("moveto");
  segments.add("quadto");
  segments.add("quadto");
  shapeNode2.setSegments(segments);
  shapeNode2.setName("ShapeNode with Segments");
  box.add(shapeNode2);

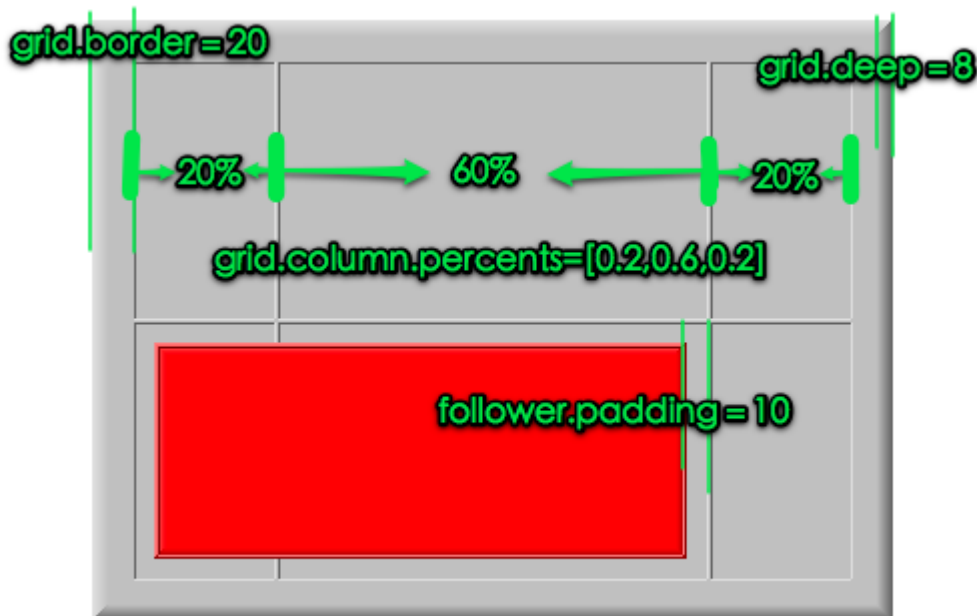
  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
</script>
</head>
```

```
<body onload="init()" style="margin:0;" > </body>  
</html>
```

twaver.Grid

twaver.Grid network element present as grid in topology, implements layout similar with TableLayout of java Swing. Thus, it realize positional placement by appointing column and row, the ranks of span.

The following example is a grid with three columns and two rows, and a child-network element in red color located at second row, first and two columns.



For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();

      var grid = new twaver.Grid();
      grid.setLocation(20, 20);
      grid.setSize(400, 300);
      grid.setStyle("grid.border", 20);
      grid.setStyle("grid.row.count", 2);
      grid.setStyle("grid.column.count", 3);
      grid.setStyle("grid.column.percents", [0.2, 0.6, 0.2]);
      grid.setStyle("grid.deep", 8);
      box.add(grid);

      var cell = new twaver.Grid();
      cell.setStyle("follower.column.index", 0);
      cell.setStyle("follower.row.index", 1);
      cell.setStyle("follower.column.span", 2);
      cell.setStyle("grid.fill.color", "#ff0000");
      cell.setStyle("follower.padding", 10);
      cell.setStyle("grid.deep", 5);
    }
  </script>
</head>
</html>
```

```
cell.setHost(grid);
grid.addChild(cell);
box.add(cell);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>
```

Data Container

Based on introduction of data container classify and basic application in [Getting Started](#) chapter, we will introduce the application of data container in detail. Here includes various events of data container, quick-finder mechanism and selection model appliance.

- [Events of Data Container](#)
- [Quick Finder Mechanism](#)
- [Selection Model Appliance](#)

Events of Data Container

Data container is not only the collection of network elements, but also is the container of layers and the manager of selection model. Different changes of elements dispatch different events. For example, element plus or minus dispatch `DataBoxChangeEvent`, property changes of element dispatch `PropertyChangeEvent`, layer changes of element dispatch `HierarchyChangeEvent`, and selection changes will be maintained by `SelectionModel`. Know more about [Selection Model](#).

Property Changes of Data Container -PropertyChange

The property of `DataBox` self can be changed such as name, client property. Event style property can be changed for `ElementBox`.

```
databox.setName("my databox");
databox.setClient(propertyName, value);
elementBox.setStyle(styleName, value);
...
```

The changes of data container property will dispatch `PropertyChangeEvent`. Add listener as following codes:

```
//databox property change listener
databox.addPropertyChangeListener(function(e){ });
```

Element Plus or Minus - DataBoxChange

As the name suggests, the quantity of elements in container changes will dispatch event. Adding new elements is corresponding to "add". Removing elements is correspond to "remove", And clearing elements is correspond to "clear".

```
//databox change listener
databox.addDataBoxChangeListener(function(e){ });
```

Event of `DataBox` change contains their properties: kind, data, datas

kind:String - three event types

- add
- remove
- clear

data- target element. For example: `var dataA = new twaver.Data()`, adding dataA into dataBox, the dataA property of correspondent event is data.

datas- target element collection. For example: removing elements, the elements property of correspondent event is datas.

```
databox.addDataBoxChangeListener(function(e){
    var kind = e.kind;
    if(kind == "add"){
        console.log("add a new data [" + e.data + "]");
    }
});
```

Property Changes of Element- DataPropertyChange

TWaver forward element changes by data container. In this way, we don't need to add listener on each element but only on data container.



The difference between DataBoxChange and DataPropertyChange is the target object. The DataBoxChange is point to databox self properties, and the DataPropertyChange is point to element properties.

```
//data property change listener
databox.addDataPropertyChangeListener(function(e){ });
```

The following code is used for listening element name changed.

```
databox.addDataPropertyChangeListener(function(e){
    if(e.property == "name"){
        trace("data name changed - old name: " + e.oldValue + ", new name: " + e.newValue);
    }
});
```

Layer Changes of Element - HierarchyChange

Data container organizes the hierarchy by parent-child relationships of elements. Such as the tree view responds to the hierarchy of databox. You can change the hierarchy relationships by modifying parent-child relationships, or calling related functions. Data container trigger HierarchyChange event when hierarchy is modified.

```
//hierarchy change listener
databox.addHierarchyChangeListener(function(e){ });
```

The two important messages of HierarchyChange event are oldIndex and newIndex. The oldIndex stands for original position and the newIndex is for new one.

Sample:

The following codes demonstrate all events and results.

```
var databox;
function init() {
    databox = new twaver.DataBox();

    // databox property change listener
    databox.addPropertyChangeListener(function(e) {
        console.log("databox property: " + e.property
            + " changed, old value: " + e.oldValue
            + ", new value: " + e.newValue + "");
    });

    // databox change listener
    databox.addDataBoxChangeListener(function(e) {
        if (e.data) {
            console.log(" " + e.data.getName() + " " + e.kind);
        } else {
            console.log("databox " + e.kind);
        }
    });
}
```



```

    });

    // data property change listener
    databox.addDataPropertyChangeListener(function(e) {
        console.log("data property: " + e.property
            + " changed, old value: " + e.oldValue
            + ", new value: " + e.newValue + "");
    });

    // hierarchy change listener
    databox.addHierarchyChangeListener(function(e) {
        console.log("data hierarchy changed, old index: " + e.oldIndex
            + ", new index: " + e.newIndex + "");
    });

    databox.setName("Box A");

    var dataA = createData("A");
    var dataB = createData("B");
    dataA.setName("AA");
    dataB.setClient("NO", 1);
    databox.clear();

    var dataC = createData("C");
    var dataD = createData("D");
    var dataE = createData("E");
    var dataF = createData("F");

    databox.remove(dataC);

    databox.moveToTop(dataE);

    databox.moveToBottom(dataD);

    console.log("Root datas: " + databox.getRoots());
}

function createData(name) {
    var data = new twaver.Data();
    data.setName(name);
    databox.add(data);
    return data;
}

```

Output results:

```

databox property: 'name' changed, old value: 'DataBox', new value: 'Box A'
'A' add
'B' add
data property: 'name' changed, old value: 'A', new value: 'AA'
data property: 'C:NO' changed, old value: 'undefined', new value: '1'
databox clear
'C' add
'D' add
'E' add
'F' add
'C' remove
data hierarchy changed, old index: '1', new index: '0'
data hierarchy changed, old index: '1', new index: '2'
Root datas: E,F,D

```

Quick Finder Mechanism

QuickFinder is used to find out the kind of elements with same property quickly, such as alarms with same security level and etc.

Let's explain how to use QuickFinder, tacking elements with certain name for an example:

At first, create a QuickFinder bundled with name property. The first property is the data container that will be the target of QuickFinder, and the second one is the property you are looking for.

```
var nameFinder = new twaver.QuickFinder(databox, "name");
```

Secondly, find out all elements named "groupA_1".

```
var datas = nameFinder.find("group-1");
```

nameFinder.find("group-1") means find all elements named "group-1", return the result collection.

Introduction of QuickFinder in detail:

Construct QuickFinder

```
twaver.QuickFinder = function (dataBox, propertyName, propertyType, valueFunction, filterFunction)
```

dataBox: data container, the target of QuickFinder.

propertyName: property name.

propertyType: property type, the default one is "accessor" stands for being able to getting this property directly, such as data.getName()

Three property types:

accessor: getting property directly by datapropertyName, just like the javaBean property of java.

client: getting property by data.getClient(propertyName), client property.

style: getting property by data.getStyle(propertyName), style property.

valueFunction: the function to get results. The default way is by the three methods as above. You can custom the function valueFunction yourself.

filterFunction: filter function is to filter elements again based on above custom parameters. function(data){}
Returning true or false to indicate whether participate in finding.

How to Find

QuickFinder provides two finding functions:

```
findFirst : function(value)
find : function(value)
```

findFirst: return the first found out element.

find: return all found out elements.

Sample

Construct one property "name" QuickFinder and one client property "NO" QuickFinder.

```
function init() {
    var box = new twaver.DataBox();

    for (var i = 0; i < 20; i++) {
        var data = new twaver.Data();
        data.setName("group-" + parseInt(i / 4));
        data.setClient("NO", i % 4);
        box.add(data);
    }

    var nameFinder = new twaver.QuickFinder(box, "name");
    var noFinder = new twaver.QuickFinder(box, "NO", "client");

    data = nameFinder.findFirst("group-1");
    log("nameFinder.findFirst(\"group-1\")", data);

    var datas = nameFinder.find("group-1");
    log("nameFinder.find(\"group-1\")", datas);

    data = noFinder.findFirst(1);
    log("noFinder.findFirst(1)", data);

    datas = noFinder.find(1);
    log("noFinder.find(1)", datas);
    var finder = new twaver.QuickFinder(box, "name");
    finder.find("")
}

function getDataDescription(data) {
    return "name:" + data.getName() + ", NO:" + data.getClient("NO");
}

function log(title, data) {
    console.log(title);
    if (data instanceof twaver.Data) {
        console.log(getDataDescription(data));
    } else if (data instanceof twaver.List) {
        data.forEach(function(item) {
            console.log(getDataDescription(item));
        });
    }
}
```

Output results:

```
nameFinder.findFirst("group-1")
name:group-1, NO:0
nameFinder.find("group-1")
name:group-1, NO:0
name:group-1, NO:1
name:group-1, NO:2
name:group-1, NO:3
noFinder.findFirst(1)
name:group-0, NO:1
noFinder.find(1)
name:group-0, NO:1
name:group-1, NO:1
name:group-2, NO:1
```

name:group-3, NO:1
name:group-4, NO:1

Selection Model Appliance

SelectionModel is to maintain the selection information of elements in DataBox. All adding, clearing selection for elements are realized by SelectionModel.Make "data" element in databox selected by calling following API:

```
databox.getSelectionModel().appendSelection(data);
```

View components own their SelectionModel, and bundle with their DataBox. For example: make element selected in topology by calling following API:

```
network.getSelectionModel().appendSelection(element);
```

Construct SelectionModel

You can get default SelectionModel of DataBox by databox.getSelectionModel(). The SelectionModel of view components is the same as DataBox by default. Invoke it by following code: view.getSelectionModel(), for example, network.getSelectionModel() and tree.getSelectionModel().

User can setup your SelectionModel both for DataBox and view components. Construct a custom SelectionModel at first.

```
var mySelectionModel = new twaver.SelectionModel(databox);
```

Then bundle SelectionModel with databox/view components in constructor function.

```
databox.setSelectionModel(mySelectionModel);
network.setSelectionModel(mySelectionModel);
...
```

The SelectionModel of view components is the one of databox by default. In this way, TWaver can make sure synchronize for all view components selection. Constructing SelectionModel for view components separately is to deal with the independent selection issue.

How to Make Element Selected and How to Remove Selection

The following functions of SelectionModel can realize selection appending, element selected, all elements selected, the selection of all elements removed and etc.

```
//append selected element, the parameter can be both one single element and elements collection
appendSelection : function(datas)

//setting selection for elements is different with appending selected element.
//This function will remove all original selection at first.
appendSelection : function(datas)

//select all elements in databox
selectAll : function()

//remove selected element, the parameter can be both one single element and elements collection
removeSelection : function(datas)
```

```
//clear all selected elements
clearSelection : function()
```

Get Selection Information of Elements

You can get the selection information in SelectionModel. Such as all selected elements, the quantity of selected elements, the selection status of elements and etc.

```
//get all selected elements. Note: the return result is passed by
//reference. So you shouldn't do any changes to this collection.
getSelection : function()

//get new collection from SelectionModel. Note: the return result is the
//new constructed collection which is different with above method.
//matchFunction: the parameter is IData and the return result is
//true/false. The false result stands for excluding this element.
toSelection: function (matchFunction, scope)

//the quantity of selected elements
size: function ()

//the selection status of element
contains: function (data)

//get the last element in selected elements collection
getLastData: function ()

//get the first element in selected elements collection
getFirstData: function ()

//whether the element is selected
isSelectable: function (data)
```

Selection Change Event

SelectionModel maintains selection state of all elements. The status changes can dispatch respondent selection change event. For example: calling method to append element selection would dispatch "append" event. TWaver uses the following code to listening selection change event.

```
selectionModel.addSelectionChangeListener(function(e){
    console.log("Kind: " + e.kind + ", datas: " + e.datas.toString());
});
```

Selection change event includes two properties, "kind", "datas". "kind" stands for selection event type, 'datas' is for event data

There are five types of event: append, set, remove, all, clear. They are responding to five functions: appendSelection, setSelection, removeSelection, selectAll, clearSelection.

Selection Model

SelectionModel provides three models: multiple-selection, single-selection and unselectable which are in common use for UI. At the same time, SelectionModel support the data layer of view components SelectionModel, taking multiple selections as default.

Change the selection model:

```
selectionModel.setSelectionMode("singleSelection");
```



Note: when the SelectionModel changes from multiple-selection to single-selection or unselectable, TWaver will invoke clear method to clear all selection status.

Sample

Let's show how to do element selection, how to change element selection and how to switch the selection model.

```
function init() {
    var databox = new twaver.DataBox();
    var selectionModel = databox.getSelectionModel();

    for (var i = 0; i < 10; i++) {
        var data = new twaver.Data(i);
        data.setName("data_" + i);
        databox.add(data);
    }

    selectionModel.addSelectionChangeListener(function(e) {
        console.log("Kind: " + e.kind + ", datas: "
            + e.datas.toString());
    });

    selectionModel.appendSelection(databox.getDataById(0));
    selectionModel.appendSelection(databox.getDataById(1));
    selectionModel.removeSelection(databox.getDataById(0));
    selectionModel
        .setSelection([databox.getDataById(2), databox.getDataById(6)]);
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);

    // single selection mode
    console.log("setting single selection mode, remove all selected elements at first, then append no more than one
    element at the same time");
    selectionModel.setSelectionMode("singleSelection");
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);
    console.log("selection size: " + selectionModel.size());

    console.log("none selection mode, remove all selected elements at first, then unable select data any more");
    selectionModel.setSelectionMode("noneSelection");
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);

    console.log("The default selection model is multiple selection");
    selectionModel.setSelectionMode("multipleSelection");

    console.log("\nsetting filterFunction , filter data bigger than 5");
    selectionModel.setFilterFunction(function(data) {
        return data.getId() > 5;
    });
    selectionModel.selectAll();
}
```

Output results:

Kind: append, datas: data_0

Kind: append, datas: data_1
Kind: remove, datas: data_0
Kind: set, datas: data_1,data_2,data_6
Kind: append, datas: data_3,data_4
setting single selection mode, remove all selected elements at first, then append no more than one element at the same time
Kind: clear, datas: data_2,data_6,data_3,data_4
Kind: append, datas: data_4
selection size: 1
33
none selection mode, remove all selected elements at first, then unable select data any more
Kind: clear, datas: data_4

The default selection model is multiple selection

setting filterFunction , filter data bigger than 5.
Kind: all, datas: data_6,data_7,data_8,data_9

Using Network


This chapter we introduce the design and usage of topology components including hierarchy of topology, setting background, filter and switching interactive models:

- [Network Hierarchy](#)
- [Network Filter](#)
- [Network Function for Style Rule](#)
- [Network GUI Interaction](#)

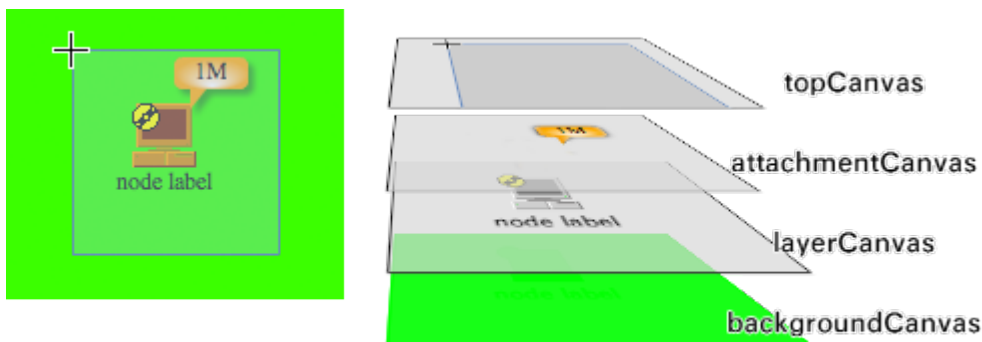
Network Hierarchy

All components in topology are put into rootDiv. The following is its hierarchy:

- rootDiv - - main canvas, contains all topology components
- topDiv - - top canvas, is to draw interaction frame, adjust the size of drag block
- attachmentDiv - - top components placement canvas
- layerDiv - - the view of all components canvas
 - layer n
 - layer ...
 - default layer
- bottomDiv - - bottom canvas, is used for drawing shade on background

 Attachment will be attached with ElementUI by default, if the value of showInAttachmentDiv is true, Attachment will be placed in attachmentDiv layer

Layer diagram:



User can get any div in Network, then add components as requirement:

```
geRootDiv : function()
getTopDiv : function()
getAttachmentDiv : function()
getLayerDiv : function()
getBottomDiv : function()
```

Let's add button in layer as above to deep impression:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();

      var node = new twaver.Node();
      node.setLocation(20,20);
      box.add(node);
      var node2 = new twaver.Node();
      node2.getAlarmState().setNewAlarmCount(twaver.AlarmSeverity.MAJOR, 1);
      node2.setLocation(150,50);
```

```

    box.add(node2);
    var link = new twaver.Link(node, node2);
    box.add(link);

    var buttonInRootDiv = createDiv("div in rootDiv", 20, 90, 255, 100, 100);
    network.getRootDiv().appendChild(buttonInRootDiv);

    var buttonInTopDiv = createDiv("div in topDiv", 70, 115, 100, 255, 100);
    network.getTopDiv().appendChild(buttonInTopDiv);

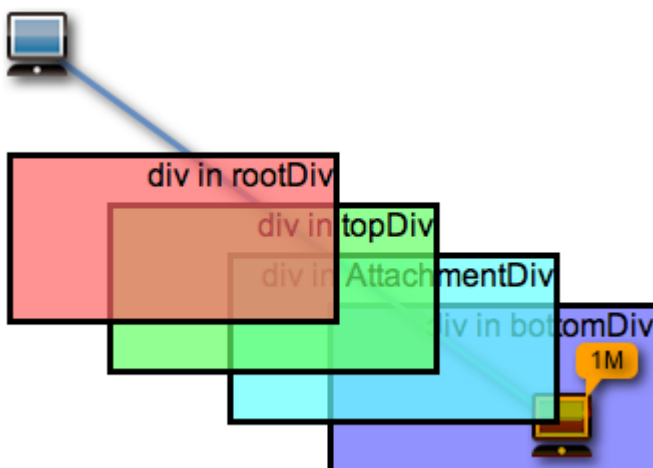
    var buttonInAttachmentDiv = createDiv("div in AttachmentDiv", 130, 140, 100, 255, 255);
    network.getAttachmentDiv().appendChild(buttonInAttachmentDiv);

    var buttonInBottomDiv = createDiv("div in bottomDiv", 180, 165, 100, 100, 255);
    network.getBottomDiv().appendChild(buttonInBottomDiv);

    var networkDom = network.getView();
    networkDom.style.width = "100%";
    networkDom.style.height = "100%";
    document.body.appendChild(networkDom);
}

function createDiv(text, x, y, r, g, b){
    var div= document.createElement("div");
    div.innerHTML = text;
    div.style.position = "absolute";
    div.style.textAlign = 'right';
    div.style.fontSize = "16px";
    div.style.left = x + "px";
    div.style.top = y + "px";
    div.style.border = "solid";
    div.style.backgroundColor = "rgba(" + r + ", " + g + ", " + b + ", 0.7)";
    div.style.height = "80px";
    div.style.width = "160px";
    return div;
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```



Network Filter

Continuing the style of TWaver, TWaver HTML5 offers a series of filter including visible filter, movable filter, editable filter and etc. TWaver HTML5 realizes one data model, different view information and different operation model by setting filter. Generally user can get different interactions and views by setting permission and setting network element type.

Network filters includes:

```
//visible filter
get/setVisibleFunction : function(filter)
//movable filter
get/setMovableFunction : function(filter)
//editable filter
get/setEditableFunction : function(filter)
```

Let's give an example to show filter usage. Pay attention that parameter type is Element, and the return value is Boolean. The example show how to set nodes has children to be visible:

```
network.setVisibleFunction(function(node){
    return node.getChildrenCount() > 0;
});
```

Network Function for Style Rule

TWaver HTML5 set component style by rule-making functions including tree view, color rendering of node in topology, border, bubble and etc. This chapter will introduce related regulations in topology.

Color rendering, border color, alarm color, alarm text, bundle line text, toolbar prompt text of network element in topology is realized by rule-making functions. User also can customize rule-making functions to meet requirements.

Style Rule of Network:

getLabel : Label function for network element text, is to set label for network element.

getToolTip : The hint of toolbar

getInnerColor : Rendering color function inner network element

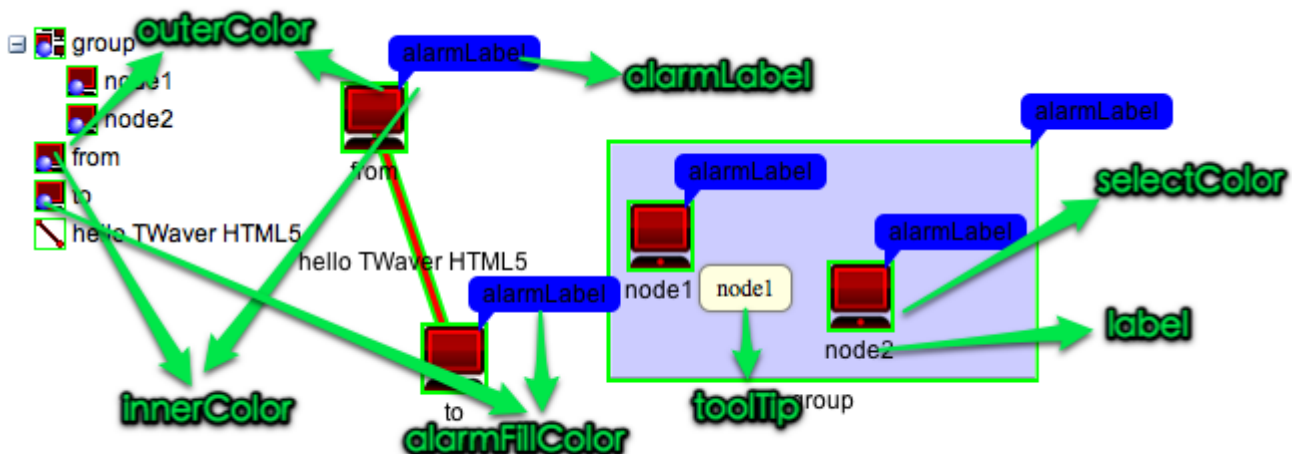
getOuterColor : Border color of network element

getSelectColor : Selected color of network element

getAlarmFillColor : Alarm bubble color

getAlarmLabel : Alarm bubble text

getLinkHandlerLabel : Bundle link text



Default

The default definition rule of twaver.Defaults, list as below:

```
//label
Network#getLabel: function (data) {
    return data.getStyle('network.label') || data.getName();
};
Tree#getLabel = function (data) {
    return data.getName();
};
//toolTip
Network#getToolTip = function (data) {
    return data.getToolTip();
};
```

```
//innerColor
Network/Tree#
getInnerColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getHighestNativeAlarmSeverity();
        if (severity) {
            return severity.color;
        }
        return data.getStyle('inner.color');
    }
    return null;
};

//outerColor
Network/Tree#
getOuterColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getPropagateSeverity();
        if (severity) {
            return severity.color;
        }
        return data.getStyle('outer.color');
    }
    return null;
};

//selectColor
Network#getSelectColor: function (element) {
    return element.getStyle('select.color');
};

//alarmFillColor
Network/Tree#
getAlarmFillColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getHighestNewAlarmSeverity();
        if (severity) {
            return severity.color;
        }
    }
    return null;
};

//alarmLabel
getAlarmLabel: function (element) {
    var severity = element.getAlarmState().getHighestNewAlarmSeverity();
    if (severity) {
        var label = element.getAlarmState().getNewAlarmCount(severity) + severity.nickName;
        if (element.getAlarmState().hasLessSevereNewAlarms()) {
            label += "+";
        }
        return label;
    }
    return null;
};

//linkHandlerLabel
Network#
getLinkHandlerLabel: function (link) {
    if (link.isBundleAgent()) {
        return "(" + link.getBundleCount() + ")";
    }
    return null;
};
```

```
};
```

Rule Making

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
var box;
var number;
function init() {
  number = 0;
  var network = new twaver.network.Network();
  network.setToolTipEnabled(true);
  box = network.getElementBox();
  var tree = new twaver.controls.Tree(box);

  var group = new twaver.Group();
  group.setName("group");
  box.add(group);
  group.addChild(createTWaverNode("node1", 200, 100));
  group.addChild(createTWaverNode("node2", 300, 130));
  group.setExpanded(true);

  var from = createTWaverNode("from", 30, 30);
  var to = createTWaverNode("to", 70, 150);
  var link = new twaver.Link(from, to);
  link.setName("hello TWaver HTML5");
  box.add(link);

  // tree inner color
  tree.getInnerColor = function(data) {
    return "#ff0000";
  };

  // tree node outer color
  tree.getOuterColor = function(data) {
    return "#00ff00";
  };

  // tree alarm color, if return null, show nothing
  tree.getAlarmFillColor = function(data) {
    if (data instanceof twaver.Element && !data.getAlarmState().isEmpty()) {
      return "#0000ff";
    }
    return null;
  };

  // network is the same
  // innerColor - node inner color
  // outerColor - node border color
  // alarmFillColor - alarm bubble color
  // alarmLabel - alarm label text
  // Body color
  network.getInnerColor = function(data) {
    return "#ff0000";
  };

  // node border color
  network.getOuterColor = function(data) {
```

```

        return "#00ff00";
    };

    // alarm bubble color
    network.getAlarmFillColor = function(data) {
        if (data instanceof twaver.Element && !data.getAlarmState().isEmpty()) {
            return "#0000ff";
        }
        return null;
    };

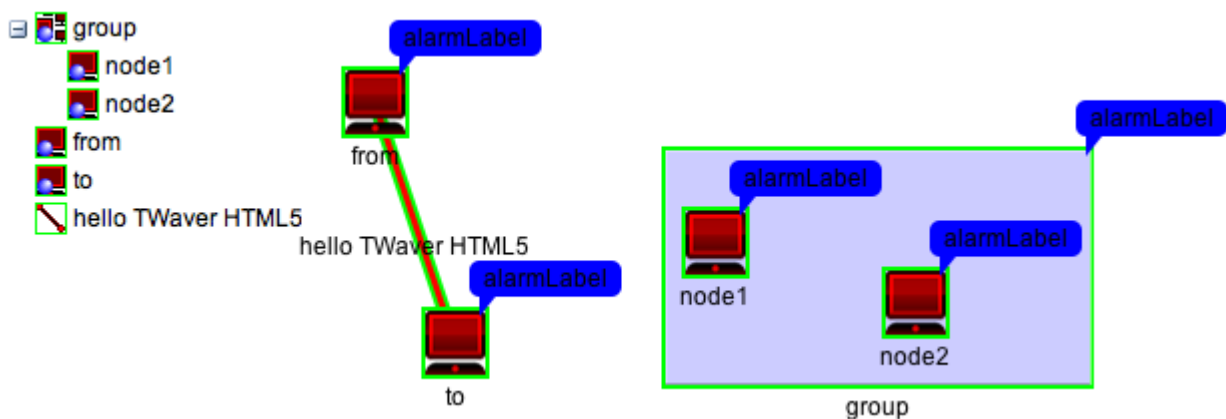
    network.getAlarmLabel = function(element) {
        if (!element.getAlarmState().isEmpty()) {
            return "alarmLabel";
        }
        return null;
    };

    network.getSelectColor = function(element) {
        return "#ffff00";
    };

    var treeDom = tree.getView();
    treeDom.style.width = "150px";
    treeDom.style.height = "100%";
    var networkDom = network.getView();
    networkDom.style.left = "150px";
    networkDom.style.width = "100%";
    networkDom.style.height = "100%";
    document.body.appendChild(treeDom);
    document.body.appendChild(networkDom);
}

function createTWaverNode(name, x, y) {
    var node = new twaver.Node();
    node.setName(name);
    node.setToolTip(name);
    node.getAlarmState().increaseNewAlarm(twaver.AlarmSeverity.MAJOR);
    node.setClient("number", number++);
    node.setLocation(x, y);
    box.add(node);
    return node;
}
</script>
</head>
<body onload="init()" style="margin:10;"></body>
</html>

```



Network GUI Interaction

InteractionHandler

Mouse and keyboard interaction in topology compose by a set of InteractionHandler. And each InteractionHandler contains independent event listening and uninstalling operation. The basic class is twaver.network.interaction.BaseInteraction, other interaction class extends this basic class, the following code is BaseInteraction, setup function is used for installing listening, tearDown function is used for uninstalling listening. Usually we need to rewrite these two method when implements listening class.

```
twaver.network.interaction.BaseInteraction = function (network) {
    this.network = network;
};
twaver.Util.ext('twaver.network.interaction.BaseInteraction', Object, {
    setUp: function () {

    },
    tearDown: function () {

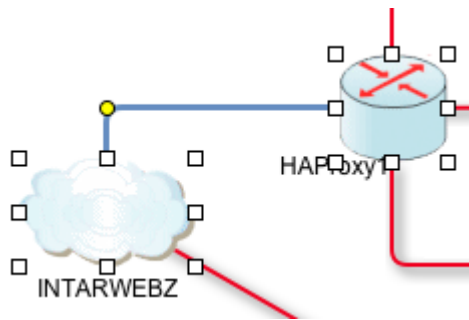
    },
    addListener: function () {
        for (var i = 0; i < arguments.length; i++) {
            var type = arguments[i];
            twaver.Util.addEventListener(type, 'handle_' + type, this.network.getView(), this);
        }
    },
    removeListener: function () {
        for (var i = 0; i < arguments.length; i++) {
            twaver.Util.removeEventListener(arguments[i], this.network.getView(), this);
        }
    }
});
```

Commonly Used Interaction Model

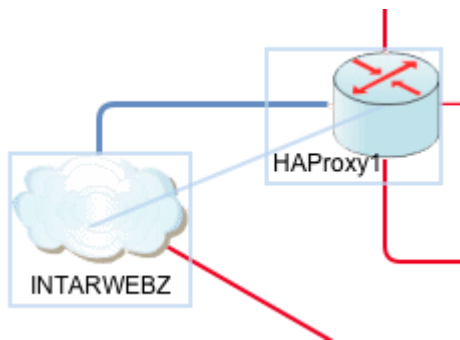
Network pre-defined several commonly used interaction models including default, editable, creating links, creating shapelink and polygonal network element models. Moreover, magnifier interaction and fish-eye interaction also are easy to implement.

```
setDefaultInteractions: function (lazyMode)
setPanInteractions: function ()
setEditInteractions: function (lazyMode)
setCreateElementInteractions: function (type)
setCreateLinkInteractions: function (type)
setCreateShapeLinkInteractions: function (type)
setCreateShapeNodeInteractions: function (type)
setTouchInteractions: function ()
```

Editor Model



Creating Link Interaction Model



Creating ShapeLink Interaction Model

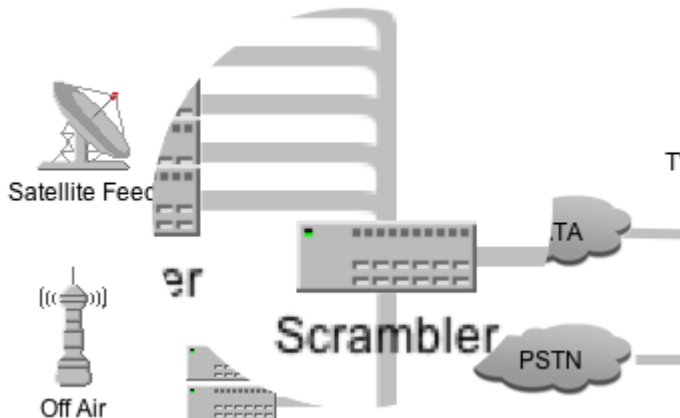


Creating Polygon Interaction Model



Magnifier Interaction:

```
network.setInteractions([
    new twaver.network.interaction.SelectInteraction(network),
    new twaver.network.interaction.MoveInteraction(network),
    new twaver.network.interaction.MagnifyInteraction(network),
    new twaver.network.interaction.DefaultInteraction(network)
]);
```



The following example realizes the switch between any interaction models, and implements creating network element by dragging button:

```
<!DOCTYPE html>
<html>
<head>
    <title>TWaver HTML5 Demo</title>
    <script type="text/javascript" src="../demo/twaver.js"></script>
    <script type="text/javascript">
        var network;
        function init(){
            network = new twaver.network.Network();

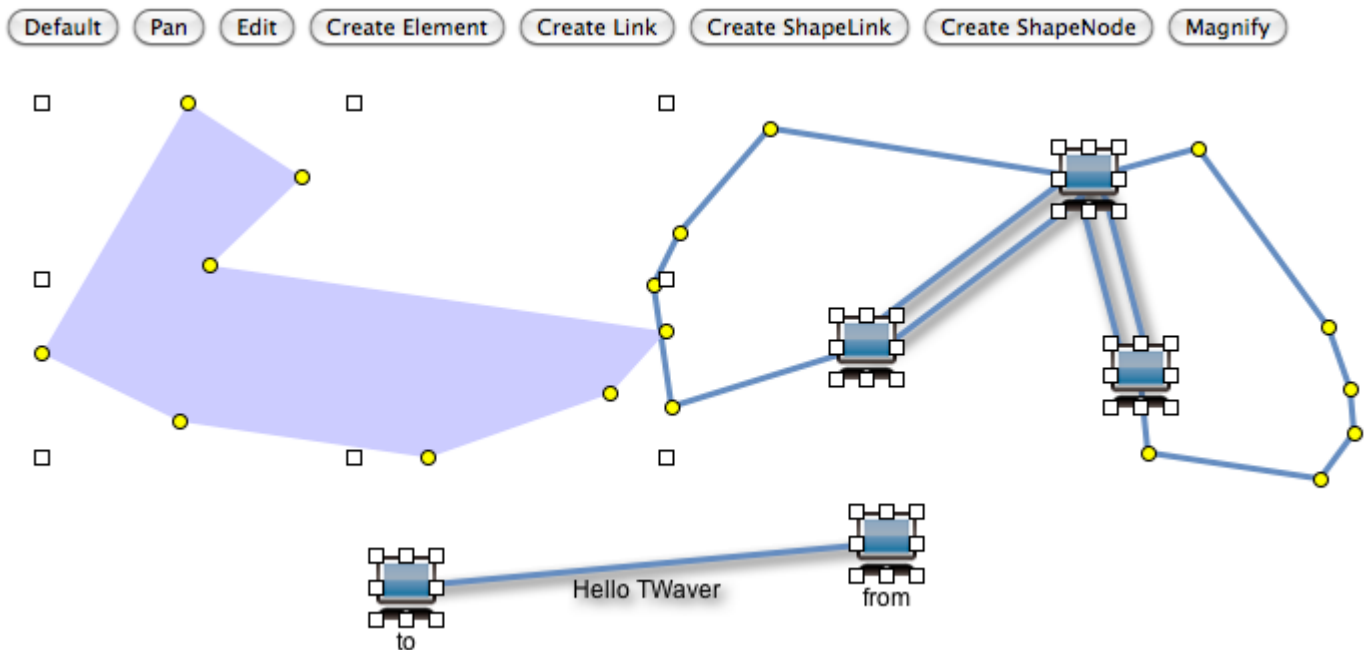
            var box = network.getElementBox();
            var node = new twaver.Node();
            node.setName("from");
            node.setLocation(100, 100);
            box.add(node);
            var node2 = new twaver.Node();
            node2.setName("to");
            node2.setLocation(300, 300);
            box.add(node2);

            var link = new twaver.Link(node, node2);
            link.setName("Hello TWaver");
            link.setToolTip("<b>Hello TWaver</b>");
            box.add(link);

            var networkDom = network.getView();
            networkDom.style.width = "100%";
            networkDom.style.height = "400px";
            document.getElementById("network").appendChild(networkDom);
        }
    </script>
</head>
</html>
```

```
function setMagnifyInteraction(){
    network.setInteractions([
        new twaver.network.interaction.SelectInteraction(network),
        new twaver.network.interaction.MoveInteraction(network),
        new twaver.network.interaction.MagnifyInteraction(network),
        new twaver.network.interaction.DefaultInteraction(network));
    ]
}
</script>
</head>
<body onload="init()" style="margin:0px;">
<div style="width: 100%;" >
<button onclick="network.setDefaultInteractions();">Default</button>
<button onclick="network.setPanInteractions();">Pan</button>
<button onclick="network.setEditInteractions();">Edit</button>
<button onclick="network.setCreateElementInteractions();">Create Element</button>
<button onclick="network.setCreateLinkInteractions();">Create Link</button>
<button onclick="network.setCreateShapeLinkInteractions();">Create ShapeLink</button>
<button onclick="network.setCreateShapeNodeInteractions();">Create ShapeNode</button>
<button onclick="setMagnifyInteraction();">Magnify</button>
</div>
<div id="network" />
</body>
</html>
```

Run the example:



Using Tree

twaver.controls.Tree binds with a DataBox, for presenting the hierarchy of elements in that DataBox, the element's hierarchy depends on the Parent-Child relationship.

- [Tree Node Presentation](#)
- [Tree's Hierarchy & Order](#)
- [Check Mode of Tree](#)

Creating a tree

Tree component is the same as network component, it can be created by API, you can give a databox when create it,

```
var tree = new twaver.controls.Tree();
```

Example

Let's create a tree, and append some data

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var tree = new twaver.controls.Tree();

      var box = tree.getDataBox();
      var group = new twaver.Group();
      group.setName('Group');
      box.add(group);

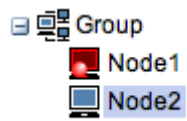
      var node1 = new twaver.Node();
      node1.setName("Node1");
      node1.setParent(group);
      node1.getAlarmState().increaseNewAlarm(twaver.AlarmSeverity.CRITICAL);
      box.add(node1);
      var node2 = new twaver.Node();
      node2.setName("Node2");
      node2.setParent(group);
      box.add(node2);
      tree.getSelectionModel().setSelection(node2);

      var treeDom = tree.getView();
      treeDom.style.width = "100%";
      treeDom.style.height = "100%";
      document.body.appendChild(treeDom);

      tree.expandAll();
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
```

```
</html>
```

Running Result

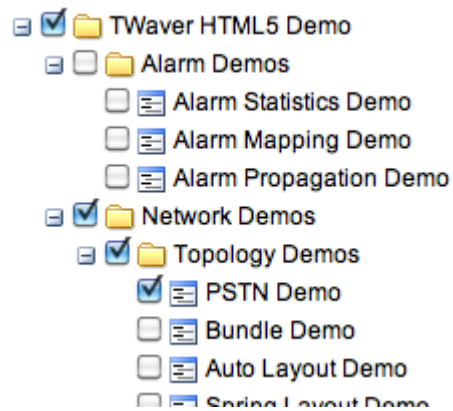


Tree Presentation

Tree node with a alarm bubble



Tree with Checking Mode



Tree Node Presentation

By default, a tree node includes one icon and one label, TWaver made some extensions to the tree, tree node provides some other parts, message block and some other icons, it also provides icon color rendering, alarm bubble mounting...

Tree Node Styles

Setting Tree Node Icon

```
data.setIcon('iconName');
```

Setting Tree Node Label

```
data.setName('001');
```

Example

The following example gives the common use of tree, contains setting icon, custom label and render alarm bubble.

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var box = new twaver.ElementBox();
      var tree = new twaver.controls.Tree(box);

      var group = new twaver.Group();
      group.setName('Group');
      box.add(group);

      for(var i = 0; i < 3; i++){
        var node1 = new twaver.Node();
        node1.setIcon("images/twaver-small.png");
        node1.setName("Node-" + i);
        node1.setParent(group);
        box.add(node1);
        for(var j = 0; j < 3; j++){
          var node2 = new twaver.Node();
          node2.setName("Node-" + i + "-" + j);
          node2.getAlarmState().setNewAlarmCount(twaver.AlarmSeverity.MAJOR, 1);
          node2.setParent(node1);
          box.add(node2);
        }
      }

      var treeDom = tree.getView();
      treeDom.style.width = "100%";
      treeDom.style.height = "100%";
    }
  </script>
</head>
</html>
```

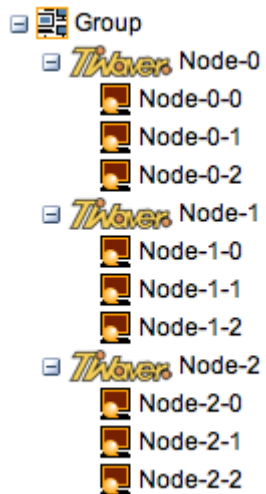
```

        document.body.appendChild(treeDom);

        tree.expandAll();
    }
</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>

```

Running Results :



Tree's Hierarchy & Order

The hierarchy of tree nodes depends on the Parent-Child relationship, that means we can adjust the hierarchies by setting element's parent or children, we also can change the indexes of elements by moving methods.

Setting Parent-Child Relationship

```
node.setParent(parent);
□node.addChild(child);
```

Adjusting Element's Index

□By default, tree node orders as same as the DataBox's, we can change this order by move* methods of DataBox

```
□□box.move**(node);
```

Setting the Sorting Compare

```
□tree.setSortFunction(compareFunction);
```

If we want to sort by name, we can set compare function as below:

```
tree.setSortFunction(function(d1, d2){
    return d1.getName() > d2.getName() ? -1 : 1;
});
```

Sorting Example

In the following example, we'll create ten random tree nodes, and sorts them in ascending.

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var tree = new twaver.controls.Tree();

      var box = tree.getDataBox();

      for(var i = 0; i < 3; i++){
        var node1 = new twaver.Node();
        node1.setName("Node - " + Math.floor(Math.random()*10));
        box.add(node1);
        for(var j = 0; j < 3; j++){
          var node2 = new twaver.Node();
          node2.setName("Child - " + Math.floor(Math.random()*10));
```

```

        node2.setParent(node1);
        box.add(node2);
    }
}

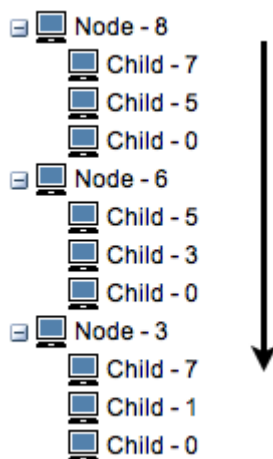
tree.setSortFunction(function(d1, d2){
    return d1.getName() > d2.getName() ? -1 : 1;
});

var treeDom = tree.getView();
treeDom.style.width = "100%";
treeDom.style.height = "100%";
document.body.appendChild(treeDom);

tree.expandAll();
}
</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>

```

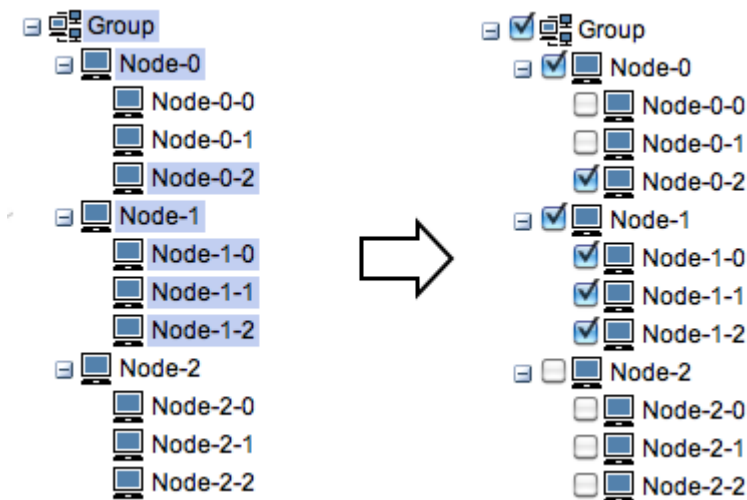
Running Interface



Check Mode of Tree

Tree component of TWaver HTML5 supports check module. If check a tree node, it will select this node.

Default Select-Mode to Check-Mode



Check Mode

Tree component provides four types of check modes, set by the following method :

```
Tree.setCheckMode(...);
```

Default Check Mode - default

Every node is checkable

Children Check Mode - children

Every leaf node is checkable

Descendant Check Mode - descendant

If the parent node is checked, its children and grandchildren would be checked, too.

Ancestor Descendant Check Mode - descendantAncestor

If the parent node is checked, its children and grandchildren would be checked, and its parent and grandparent would be checked, too.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
```

```
function init(){
    var tree = new twaver.controls.Tree();

    var box = tree.getDataBox();
    var group = new twaver.Group();
    group.setName('Group');
    box.add(group);

    for(var i = 0; i < 3; i++){
        var node1 = new twaver.Node();
        node1.setName("Node-"+i);
        node1.setParent(group);
        box.add(node1);
        for(var j = 0; j < 3; j++){
            var node2 = new twaver.Node();
            node2.setName("Node-"+ i + "-" + j);
            node2.setParent(node1);
            box.add(node2);
        }
    }

    tree.setCheckMode("descendantAncestor");

    var treeDom = tree.getView();
    treeDom.style.width = "100%";
    treeDom.style.height = "100%";
    document.body.appendChild(treeDom);

    tree.expandAll();
}
</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>
```

Using Table

twaver.controls.Table component realizes on combination with DataBox. This chapter will introduce the usage of table, the definition of table column, the customize of renderer and editor, the usage of filter and sorting order.

- [Table Configuration](#)
- [Table Data Orders](#)

Let's show properties name, id, icon for Table component:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();

      var box = table.getDataBox();
      var root = new twaver.Data();
      root.setIcon("group_icon");
      root.setName('Root');
      box.add(root);

      var i = 100;
      while(i-->0){
        data = new twaver.Data();
        data.setName("TWaver-" + i);
        data.setParent(root);
        box.add(data);
      }

      createColumn(table,'Name', 'name', 'accessor', 'string');
      createColumn(table,'Id', 'id', 'accessor', 'string');
      createColumn(table,'Icon', 'icon', 'accessor');

      var tablePane = new twaver.controls.TablePane(table);
      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
      window.onresize = function(){
        tablePane.invalidate();
      }
    }

    function createColumn(table, name, propertyName, propertyType, valueType) {
      var column = new twaver.Column(name);
      column.setName(name);
      column.setPropertyName(propertyName);
      column.setPropertyType(propertyType);
      if (valueType) column.setValueType(valueType);
      table.getColumnBox().add(column);
      return column;
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

Name	Id	Icon	
Root	C5597E4FB7D94A209666B2F50...	group_icon	
TWaver-99	94D22BF7B54E444A91EFF430...	data_icon	
TWaver-98	BEC49F6B3EE2449A817D8995...	data_icon	
TWaver-97	AAE7AFC73DC946C9981F1FBC...	data_icon	
TWaver-96	6AE75956CC5F421D8BC2B70A...	data_icon	
TWaver-95	74AFFD229557407CB AFF100B...	data_icon	
TWaver-94	C46BCA2BF2474DA69BD652B0...	data_icon	
TWaver-93	DDC280D6973A44638D17C3C5...	data_icon	
TWaver-92	FF5C11F51A994A6D8273361C4...	data_icon	

Table Configuration

Table component is used to present attributes of elements that in the dataBox, each element means one row, and each column means one type of element's attribute, like 'name', 'location' or others.

twaver.Column is the implement class of table column in TWaver HTML5. Firstly we need to which property is bounded with this column, and then we can set column name, column type and etc.

Table Column Setting

The following code gives a basic setting of table column, shows name property.

```
var column = new twaver.Column();
column.setName("Name");//set table column name
column.setPropertyName("name");//set property name
column.setPropertyType("accessor");//set property type, default value is accessor
column.setValueType("string");//set value type, default value is string
table.getColumnBox().add(column);//add column to columnBox
```

Above code gives the setting about column name, column property, property type and value. Property Type has four values: field, accessor, style, client. Different type has different method to get value.

field: for example:node.name

accessor: node.getName(),node.setName(...);

style: node.getStyle("name"),node.setStyle("name", ...);

client: node.getClient("name"), node.setClient("name", ...).

The following code gives an example using style property to show a column.

```
column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
table.getColumnBox().add(column);
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript" src="../demo/demo.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();
      var box = table.getDataBox();

      var i = 100;
      while(i-->0){
        data = new twaver.Node();
        data.setName("TWaver-" + i);
        data.setStyle("inner.color", demo.Util.randomColor());
        box.add(data);
      }
    }
  </script>
</head>
</html>
```

```

var column = new twaver.Column();
column.setName("Name");
column.setPropertyName("name");
column.setPropertyType("accessor");
column.setValueType("string");
table.getColumnModel().add(column);

column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
table.getColumnModel().add(column);

var tablePane = new twaver.controls.TablePane(table);
var tableDom = tablePane.getView();
tableDom.style.width = "100%";
tableDom.style.height = "100%";
document.body.appendChild(tableDom);
window.onresize = function(){
    tablePane.invalidate();
}
}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>

```

The Result

Name	Inner Color
TWaver-99	Blue
TWaver-98	Yellow
TWaver-97	Orange
TWaver-96	Red
TWaver-95	Purple
TWaver-94	Brown
TWaver-93	Green
TWaver-92	Blue
TWaver-91	Yellow
TWaver-90	Purple

Table Data Orders

Adjusting Data Index

It's can adjust the index of the rows by method - `dataBox.move***(element)`, like move the first row to bottom or move the bottom row to top.

Table Sorting

Each column can be sort, by default we can click the table header to switch sort, ascending order, descending order, not order. It uses string compare by default. You can custom comparer, we will give an example using color value. Sort function is set to table column, using `Column#setSortFunction(...)`

```
var column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
column.setSortFunction(function(color1, color2) {
    if (!color1) {
        return -1;
    }
    if (!color2) {
        return 1;
    }
    var number1 = parseInt(color1.substring(1, 16));
    var r1 = (number1 >> 16) & 0xFF;
    var g1 = (number1 >> 8) & 0xFF;
    var b1 = number1 & 0xFF;
    var number2 = parseInt(color2.substring(1, 16));
    var r2 = (number2 >> 16) & 0xFF;
    var g2 = (number2 >> 8) & 0xFF;
    var b2 = number2 & 0xFF;
    return (r1 + g1 + b1) - (r2 + g2 + b2);
});
table.getColumnBox().add(column);
```

The default value of color is String type, such as "#FF0000", first we need to transfer to "rgb" value, and then use "r+g+b" value to compare.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>TWaver HTML5 Demo</title>
<script type="text/javascript" src="../demo/twaver.js"></script>
<script type="text/javascript" src="../demo/demo.js"></script>
<script type="text/javascript">
function init(){
    var table = new twaver.controls.Table();
    var box = table.getDataBox();

    var i = 100;
    while(i-->0){
        data = new twaver.Node();
        data.setName("TWaver-" + i + "-" + i%10);
        data.setStyle("inner.color", demo.Util.randomColor());
    }
}
```

```

        box.add(data);
    }

    var column = new twaver.Column();
    column.setName("Name");
    column.setPropertyName("name");
    column.setPropertyType("accessor");
    column.setValueType("string");
    table.getColumnBox().add(column);

    column = new twaver.Column();
    column.setName("Inner Color");
    column.setPropertyName("inner.color");
    column.setPropertyType("style");
    column.setValueType("color");
    column.setSortFunction(function(color1, color2){
        if(!color1){
            return -1;
        }
        if(!color2){
            return 1;
        }
        var number1 = parseInt(color1.substring(1, 16), 16);
        var r1 = (number1 >> 16) & 0xFF;
        var g1 = (number1 >> 8) & 0xFF;
        var b1 = number1 & 0xFF;
        var number2 = parseInt(color2.substring(1, 16), 16);
        var r2 = (number2 >> 16) & 0xFF;
        var g2 = (number2 >> 8) & 0xFF;
        var b2 = number2 & 0xFF;
        return (r1 + g1 + b1) - (r2 + g2 + b2);
    });
    table.getColumnBox().add(column);

    var tablePane = new twaver.controls.TablePane(table);
    var tableDom = tablePane.getView();
    tableDom.style.width = "100%";
    tableDom.style.height = "100%";
    document.body.appendChild(tableDom);
    window.onresize = function(){
        tablePane.invalidate();
    }
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

The Sorting Result

Name	Inner Color		Name	Inner Color	
TWaver-76-6		↓	TWaver-14-4		↑
TWaver-42-2			TWaver-43-3		
TWaver-30-0			TWaver-37-7		
TWaver-19-9			TWaver-67-7		
TWaver-4-4			TWaver-26-6		
TWaver-22-2			TWaver-80-0		

Using Chart

TWaver HTML5 provides a wealth of charts, TWaver chart also binds with a dataBox, to present element's chart values in the dataBox.

- [Chart Value & Chart Values](#)
- [BarChart](#)
- [LineChart](#)
- [PieChart](#)
- [DialChart](#)
- [BubbleChart](#)
- [RadarChart](#)

Chart Value & Chart Values

Chart binds with a DataBox to present element's chart values in that DataBox, there are two types of chart values: single value and list values, each these means single chart value and a set of chart values especially. In TWaver HTML5, sets chart value by two styles: `chart.value`, `chart.values`, as following codes:

```
element.setStyle("chart.value", 27);
element.setStyle("chart.values", [27, 37, 48]);
```

Chart	Chart Value	Chart Values
BarChart	True	True
LineChart	False	True
PieChart	True	False
DialChart	True	False
BubbleChart	False	True
RadarChart	False	True

BarChart

Bar chart has several types of present mode, by default each bar means one element, and the height of the bar depends on the chart value of the element. Besides, there is group mode, stack mode, layer mode and percent mode, each mode has a different display. Setting bar chart's present mode as follows:

```
chart.setType("group");
```

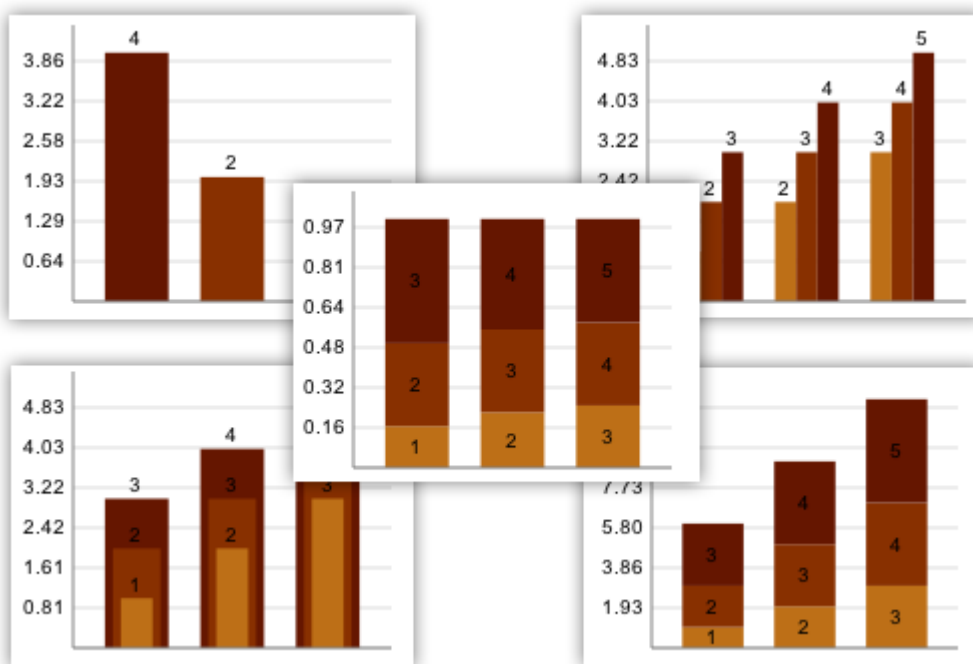
"default" - default mode

"group" - group mode

"stack" - stack mode

"layer" - layer mode

"percent" - percent mode



Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var chart = new twaver.charts.BarChart();
      var box = chart.getDataBox();

      // chart.setType("group");
      chart.setType("stack");
      // chart.setType("layer");
      // chart.setType("percent");

      var data = new twaver.Element();
      data.setStyle("chart.value", 4);
      data.setStyle("chart.values", [3,4,5]);
    }
  </script>
</head>
</html>
```

```

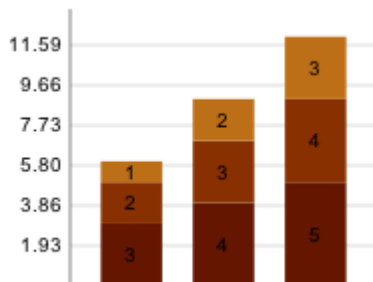
data.setStyle("chart.color", "#651600");
box.add(data);
data = new twaver.Element();
data.setStyle("chart.value", 2);
data.setStyle("chart.values", [2,3,4]);
data.setStyle("chart.color", "#883000");
box.add(data);
data = new twaver.Element();
data.setStyle("chart.value", 1);
data.setStyle("chart.values", [1,2,3]);
data.setStyle("chart.color", "#BD6F17");
box.add(data);

var chart = chart.getView();
chart.style.width = "200px";
chart.style.height = "150px";
document.body.appendChild(chart);

chart = chart.getView();
chart.style.width = "200px";
chart.style.height = "150px";
document.body.appendChild(chart);
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

Running:



LineChart

Line chart is one of the most common chart types, the line is performed as tendency, in TWaver HTML5, each line means a element, the points on the line correspond to the [chart values](#) of that element.

Example

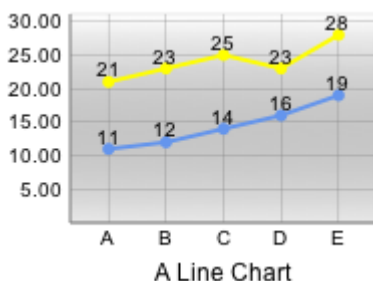
```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.LineChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [11, 12, 14, 16, 19]);
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [21, 23, 25, 23, 28]);
      box.add(data);

      chart.setXScaleTexts(new twaver.List(['A', 'B', 'C', 'D', 'E']));
      chart.setLowerLimit(0);
      chart.setYScaleValueGap(5);
      chart.setXAxisText('A Line Chart');
      chart.setBackgroundVisible(true);

      var chartDom = chart.getView();
      chartDom.style.width = "200px";
      chartDom.style.height = "150px";
      document.body.appendChild(chartDom);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

Running



PieChart

TWaver HTML5 pie chart also binds with a DataBox, each slice on it corresponds to an element, the size of the slice depends on the chart value of the element.

Pie chart has many different shapes: oval, circle, line, donut and ovalDonut.

```
chart.setType("ovalDonut");
```

Pie chart's shape:

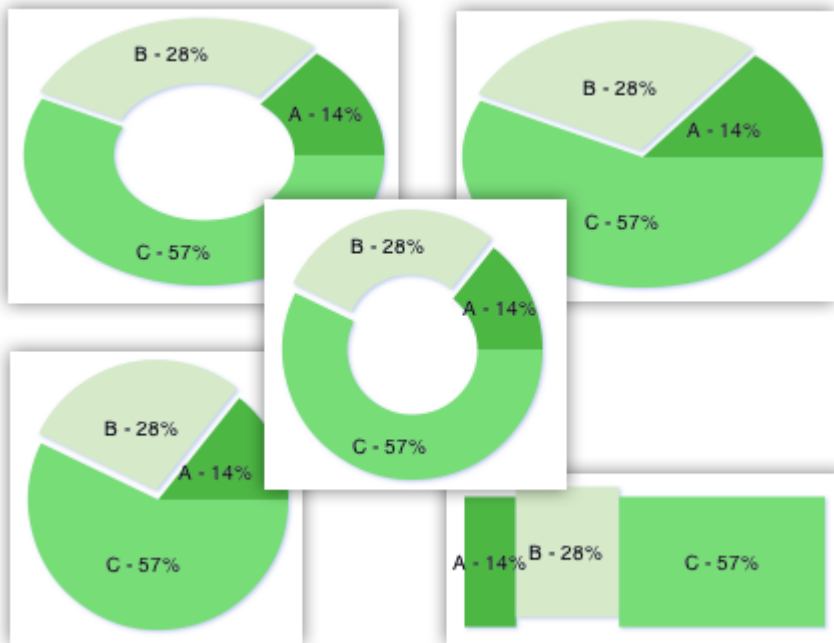
oval - oval

circle - circle

line - line

donut - donut

ovalDonut - oval donut



Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.PieChart();
      box = chart.getDataBox();

      createData('A', 10, "#4CB743");
      createData('B', 20, "#D6EAC9");
      createData('C', 40, "#77DD77");

      chart.setType("ovalDonut");
```



```

    chart.formatValueText = function(value, data){
        return data.getName() + ' - ' + parseInt(value/chart.getSum()*100) + '%';
    };

    var chartDom = chart.getView();
    chartDom.style.width = "200px";
    chartDom.style.height = "150px";
    document.body.appendChild(chartDom);
}
function createData(name, value, color){
    var data = new twaver.Element();
    data.setStyle("chart.value", value);
    data.setStyle("chart.color", color);
    data.setStyle("chart.value.color", 0x555555);
    data.setStyle("chart.value.font", 'hevetica');
    data.setName(name);
    box.add(data);
};
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

Running Interface



DialChart

Dial chart, using pointers and instrument panel to present the chart, usually each pointer corresponds to one element, sometimes one element also can present to several pointers.

A dial chart consists of three parts: the dial, shaft and pointers, these three parts all can set colors or rendering types.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.DialChart();
      box = chart.getDataBox();

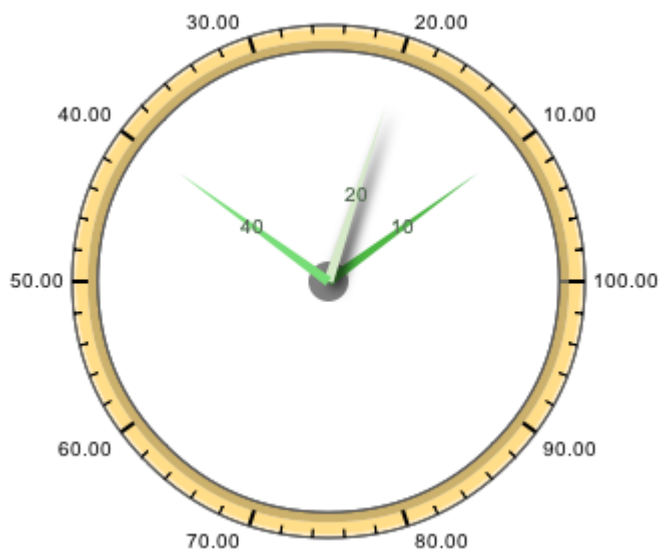
      createData('A', 10, "#4CB743");
      createData('B', 20, "#D6EAC9");
      createData('C', 40, "#77DD77");

      chart.setOutlineColor("#555555");
      chart.setOutlineWidth(1);
      chart.setOuterBrighterRadius(2);
      chart.setInnerDarkerRadius(5);
      chart.setColorRangeFillColor("#FFDD88");
      chart.setInnerRadius(0.9);
      chart.setScaleInside(false);
      chart.setScaleLowerLimitTextVisible(false);

      var chartDom = chart.getView();
      chartDom.style.width = "400px";
      chartDom.style.height = "300px";
      document.body.appendChild(chartDom);
    }

    function createData(name, value, color){
      var data = new twaver.Element();
      data.setStyle("chart.value", value);
      data.setStyle("chart.color", color);
      data.setStyle("chart.value.color", "#555555");
      data.setStyle("chart.value.font", 'hevetica');
      data.setStyle('dialchart.radius', 0.8);
      data.setName(name);
      box.add(data);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

The interface:



BubbleChart

Bubble chart, using three-dimensional numerical(x, y, value) to present element's features, each set of values corresponds to a bubble on the chart, x and y values determine the location, and the value determine bubble's size. Usually one element corresponds to one bubble, sets a bubble as follows:

```
var data = new twaver.Element();
data.setStyle("chart.values", [80]);
data.setStyle("chart.xaxis.values", [5]);
data.setStyle("chart.yaxis.values", [5]);
data.setStyle("chart.color", "#BD6F17");
box.add(data);
```

Also allows more than one bubbles corresponding to an element, sets like this:

```
data = new twaver.Element();
data.setStyle("chart.values", [50, 70]);
data.setStyle("chart.xaxis.values", [20,22]);
data.setStyle("chart.yaxis.values", [20,12]);
data.setStyle("chart.color", "#651600");
box.add(data);
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.BubbleChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [80]);
      data.setStyle("chart.xaxis.values", [5]);
      data.setStyle("chart.yaxis.values", [5]);
      data.setStyle("chart.color", "#BD6F17");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [100]);
      data.setStyle("chart.xaxis.values", [10]);
      data.setStyle("chart.yaxis.values", [30]);
      data.setStyle("chart.color", "#883000");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [50, 70]);
      data.setStyle("chart.xaxis.values", [20,22]);
      data.setStyle("chart.yaxis.values", [20,12]);
      data.setStyle("chart.color", "#651600");
      box.add(data);

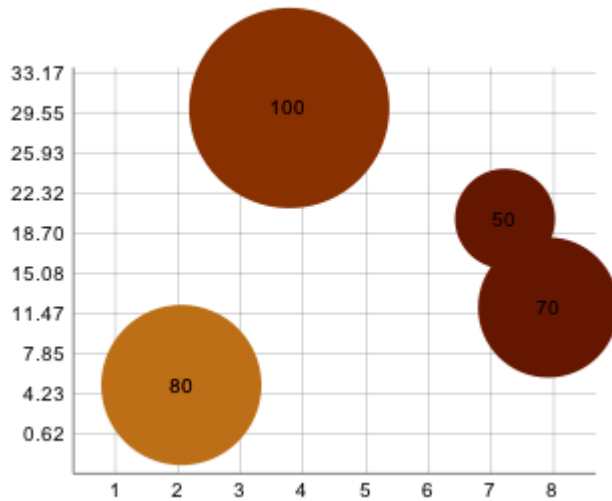
      chart.setXScaleTexts(new twaver.List(["1", "2", "3", "4", "5", "6", "7", "8"]));
```

```

var chartDom = chart.getView();
chartDom.style.width = "400px";
chartDom.style.height = "300px";
document.body.appendChild(chartDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

The Interface:



RadarChart

Radar chart, for performance of multidimensional data, each element can setting a set of [chart values](#), as the following example, it set five values for each element.

```
data.setStyle("chart.values", [80, 100, 150, 230, 40]);
```

Example:

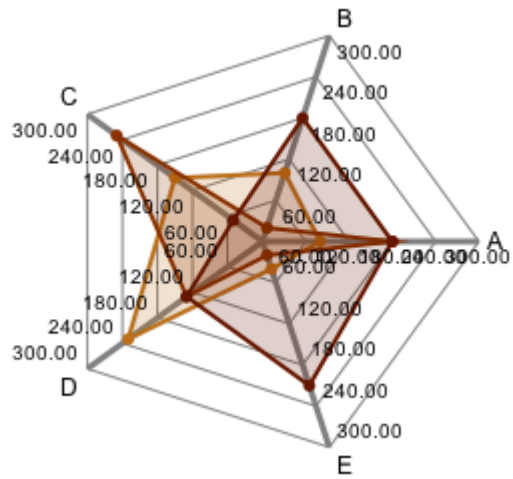
```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.RadarChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [80, 100, 150, 230, 40]);
      data.setStyle("chart.color", "#BD6F17");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [180, 20, 250, 130, 20]);
      data.setStyle("chart.color", "#883000");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [180, 180, 50, 130, 210]);
      data.setStyle("chart.color", "#651600");
      box.add(data);

      chart.setAxisList(new twaver.List([
        {text:"A", min:0, max:300},
        {text:"B", min:0, max:300},
        {text:"C", min:0, max:300},
        {text:"D", min:0, max:300},
        {text:"E", min:0, max:300}
      ]));

      var chartDom = chart.getView();
      chartDom.style.width = "300px";
      chartDom.style.height = "300px";
      document.body.appendChild(chartDom);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

Present:



Using Alarm

The front [Alarm Object](#) section made a brief introduction about how to use alarm in TWaver HTML5, this chapter will explain in detail about alarm, alarm model, alarm severity, alarm state and present of alarm.

twaver.Alarm - describes the element or device operation status, it is the basic data type of alarm, it contains alarm severity, alarm time, element id etc. it has introduced in section [Alarm Object](#).

twaver.AlarmSeverity - Reflects the urgency of the alarm contains name, nickname, value, color and other properties, and the 'value' property reflects the urgency of the alarm, there are six alarm severities predefined: CRITICAL, MAJOR, MINOR, WARNING, INDETERMINATE, CLEARED.

twaver.AlarmBox - the data container for managing alarm objects, it provides some basic operations to alarm, such as: add alarm, remove alarm or clear all alarms. Read section [Data Container](#) for details.

twaver.AlarmState - describes the current alarm status of elements or devices, it includes all statistics information about new alarms, acknowledged alarms and propagated alarms. Such as: device has alarm or not, how many alarms, how many new alarms, etc. But there is no specific alarm object is included.

twaver.AlarmStateStatistics - the AlarmState describes alarm status of every element, extended to the entire ElementBox, we can use AlarmStateStatistics to count the alarm states for all elements in the ElementBox, it includes all quantities of alarms for each alarm severity, and the numbers of all new alarms or acknowledged alarms. It also can statistics for specify elements.

- [Alarm Severity](#)
- [Alarm State & Statistics](#)
- [Alarm Display](#)

Alarm Severity

Alarm severity - Reflects the urgency of the alarm which contains name, nickname, value, color and other properties, we use class twaver. AlarmSeverity to define this type of object in TWaver HTML5

Default Alarm Severity

there are six alarm severities are predefined : CRITICAL, MAJOR, MINOR, WARNING, INDETERMINATE, CLEARED. and the 'value' property reflects the urgency of the alarm.

```
var AlarmSeverity = twaver.AlarmSeverity;
AlarmSeverity.CRITICAL = AlarmSeverity.add(500, "Critical", "C", '#FF0000');
AlarmSeverity.MAJOR = AlarmSeverity.add(400, "Major", "M", '#FFA000');
AlarmSeverity.MINOR = AlarmSeverity.add(300, "Minor", "m", '#FFFF00');
AlarmSeverity.WARNING = AlarmSeverity.add(200, "Warning", "W", '#00FFFF');
AlarmSeverity.INDETERMINATE = AlarmSeverity.add(100, "Indeterminate", "N", '#C800FF');
AlarmSeverity.CLEARED = AlarmSeverity.add(0, "Cleared", "R", '#00FF00');
```

Severity	Letter	Value	Color
CRITICAL	C	500	Red
MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

Custom Alarm Severity

Predefined alarm severities in TWaver HTML5 are all static variables, users can register their own alarm severities or remove the defaults in global. Because it is a global setting, be careful deleting alarm severity which would affect the whole process.

```
twaver.AlarmSeverity.add = function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.remove = function (name)
twaver.AlarmSeverity.clear = function ()
```

The following codes could clear all default severities and add a custom one.

```
twaver.AlarmSeverity.clear();
twaver.AlarmSeverity.add(1, "a", "a", "#FF0000", "AAA");
```

Custom Alarm Severity Example

The example clears all default severities and registers some new severities, then use them in network

```

<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var network = new twaver.network.Network();

      var box = network.getElementBox();

      twaver.AlarmSeverity.clear();
      var a = twaver.AlarmSeverity.add(1, "a", "a", "#FF0000", "AAA");
      var b = twaver.AlarmSeverity.add(2, "b", "b", "#0000FF", "BBB");
      var c = twaver.AlarmSeverity.add(3, "c", "c", "#00FF00", "CCC");

      var node1 = new twaver.Group();
      node1.setExpanded(true);
      var node2 = new twaver.Node();
      node2.setLocation(100, 100);
      var node3 = new twaver.Node();
      node3.setLocation(200, 150);
      node3.setParent(node1);
      node2.setParent(node1);
      box.add(node1);
      box.add(node2);
      box.add(node3);

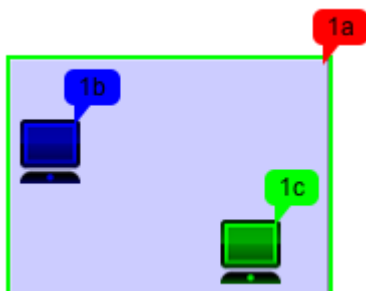
      addAlarm(box.getAlarmBox(),node1,a);
      addAlarm(box.getAlarmBox(),node2,b);
      addAlarm(box.getAlarmBox(),node3,c);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }

    function addAlarm(alarmBox, element, severity){
      var alarm = new twaver.Alarm(null, element.getId(), severity);
      alarmBox.add(alarm);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>

```

The Result



Alarm State & Statistics

Alarm State

Alarm state is one kind of element properties, it describes the current alarm status of elements or devices, it includes all statistics information about new alarms, acknowledged alarms and propagated alarms. Such as: the device has alarm or not, how many alarms, how many new alarms, etc. But there is not specific alarm object be included.

In TWaver HTML5, we use class `twaver.AlarmState` to define alarm state, we can get an instance from the following code:

```
var alarmState = node.getAlarmState();
```

`twaver.AlarmState` - Alarm State

The alarm state of element includes all statistics information about new alarms, acknowledged alarms, propagated alarms, it also includes what is the highest alarm severity of this element, etc. the methods for modifying alarm state is as follows:

```
getHighestAcknowledgedAlarmSeverity: function ()
getHighestNewAlarmSeverity: function ()
getHighestOverallAlarmSeverity: function ()
getHighestNativeAlarmSeverity: function ()
hasLessSevereNewAlarms: function ()
```

We can modify alarm state of element by these methods: increase or decrease acknowledge alarm count, increase or decrease new alarm count, clear alarm states...

```
increaseAcknowledgedAlarm: function (severity, increment)
increaseNewAlarm: function (severity, increment)
decreaseAcknowledgedAlarm: function (severity, decrement)
decreaseNewAlarm: function (severity, decrement)
acknowledgeAlarm: function (severity)
acknowledgeAllAlarms: function (severity)
getAcknowledgedAlarmCount: function (severity)
getAlarmCount: function (severity)
getNewAlarmCount: function (severity)
setNewAlarmCount: function (severity, count)
removeAllNewAlarms: function (severity)
setAcknowledgedAlarmCount: function (severity, count)
removeAllAcknowledgedAlarms: function (severity)
isEmpty: function ()
clear: function ()
```

Other Methods

```
getPropagateSeverity: function ()
setPropagateSeverity: function (propagateSeverity)
isEnabledPropagation: function ()
setEnabledPropagation: function (enablePropagation)
```

Alarm State Statistics

Alarm state statistics: make statistics for alarm state of all elements in the whole ElementBox, we use 'twaver.AlarmStateStatistics' class to achieve these functions in TWaver HTML5, it includes all quantities of alarms for each alarm severity, and the numbers of all new alarms or acknowledged alarms. It also can make statistics for specify elements.

AlarmStateStatistics - Alarm State Statistics

Create a Alarm State Statistics

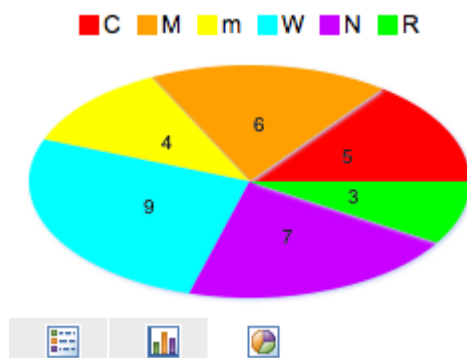
```
var statistics= new twaver.AlarmStateStatistics(elementBox);
```

Methods provided by Alarm State Statistics

```
//gets the count of new alarms
getNewAlarmCount: function (severity)
//gets the count of acknowledged alarms
getNewAlarmCount: function (severity)
//gets the count of all alarms
getTotalAlarmCount: function (severity)
//sets filter
setFilterFunction: function (f)
```

The Present of Alarm State Statistics

Puts alarm state statistics informations into a table, presents as follows:

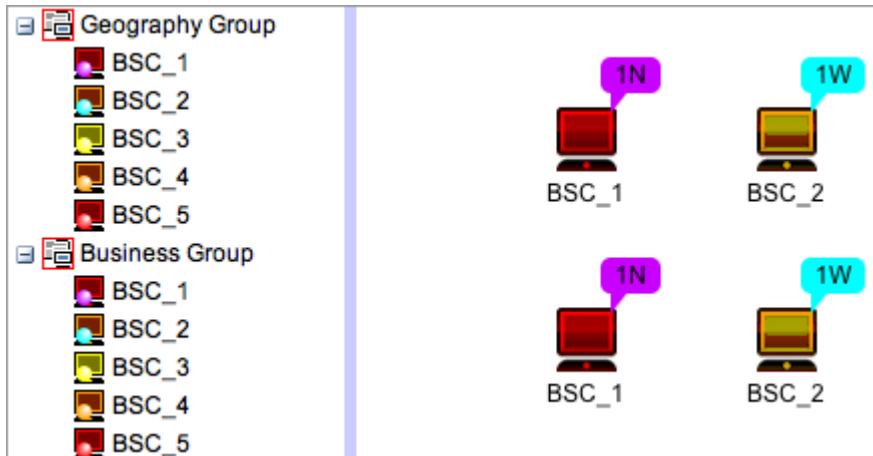


Alarm Display

There are several ways to present alarm, different components have different performances.

Alarm Bubble

TWaver HTML5 uses bubble to present new alarms, it displays on tree and network like this:



Alarm Dying (Color Fill or Outline)

The above image shows alarm dying effects, the icon of tree nodes and the image of elements on network.

Alarm Table

Alarm performance in a table is the most traditional way. First use the twaver.Table component, and bind with the AlarmBox, then all the alarms will display in the table, and each row means an alarm object.

Mapping ID	Alarm Severity	Acked	Cleared	Raised Time
1	Indeterminate	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
2	Warning	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
3	Minor	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
4	Major	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
5	Critical	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
5	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
4	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
3	Minor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54

Alarm State Statistics Chart

Alarm statistics usually are presented in chart, like this:

Severity	New	Acked	Total
MAJOR	5	1	6
WARNING	8	1	9
CLEARED	2	1	3
CRITICAL	3	2	5
MINOR	1	3	4
INDETERMINATE	0	7	7
TOTAL	19	15	34

Example for Alarm Table

The following example creates ten random alarms, and show in a table component, it uses some codes in TWaver HTML5 Demo source.

```

<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var box = new twaver.ElementBox();
      var alarmBox = box.getAlarmBox();
      var table = new twaver.controls.Table(alarmBox);

      var node = new twaver.Node();
      box.add(node);

      var i = 0;
      while(i++ < 10){
        var alarm = new twaver.Alarm(i, node.getId(), randomSeverity());
        alarm.setAcked(Math.random() >= 0.5);
        alarm.setCleared(Math.random() >= 0.5);
        alarmBox.add(alarm);
      }

      createColumn(table,'Id', 'id', 'accessor', 'string').setWidth(30);
      createColumn(table,'Alarm Severity', 'alarmSeverity', 'accessor').setHorizontalAlign('center');
      createColumn(table,'Acked', 'acked', 'accessor', "boolean");
      createColumn(table,'Cleared', 'cleared', 'accessor', "boolean");

      table.onCellRendered = function (params) {
        if (params.column.getName() === 'Alarm Severity') {
          params.div.style.backgroundColor = params.data.getAlarmSeverity().color;
        }
      };

      var tablePane = new twaver.controls.TablePane(table);
      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
      window.onresize = function(){
        tablePane.invalidate();
      }
    }

    function randomSeverity() {
      var severities = twaver.AlarmSeverity.severities;

```

```

        return severities.get(Math.floor(Math.random() * severities.size()));
    }
    function createColumn(table, name, propertyName, propertyType, valueType) {
        var column = new twaver.Column(name);
        column.setName(name);
        column.setPropertyName(propertyName);
        column.setPropertyType(propertyType);
        if (valueType) column.setValueType(valueType);
        table.getColumnModel().add(column);
        return column;
    }
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

The result

Id	Alarm Severity	Acked	Cleared
1	Warning	<input type="checkbox"/>	<input type="checkbox"/>
2	Major	<input type="checkbox"/>	<input type="checkbox"/>
3	Warning	<input type="checkbox"/>	<input type="checkbox"/>
4	Cleared	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Major	<input type="checkbox"/>	<input type="checkbox"/>
6	Minor	<input type="checkbox"/>	<input type="checkbox"/>
7	Warning	<input type="checkbox"/>	<input type="checkbox"/>
8	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Appendix

- [Network Element Stylesheet](#)

Network Element Stylesheet

All view properties of network element in TWaver HTML5 is realized by set style property. This chapter will list all properties of all kinds of network element including assignment range and instructions.

```
var link = new Link(from, to);
link.setName("hello TWaver!");
//style name: link.type
//set link type as flexional
link.setStyle("link.type", "flexional");
```

Alarm Bubble

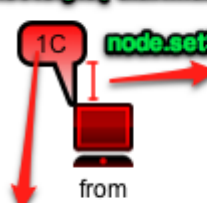
Style (twaver.Styles#)	Evaluation	Note
alarm.alpha	Number, default value is 1	transparency of alarm bubble
alarm.cap	String, default value is butt. Optional value is: butt, round, square	endpoint style of border
alarm.color	Color, default value is #000000	font color
alarm.corner.radius	Number, default value is 5	radius of alarm bubble
alarm.direction	String, default value is aboveright. Optional value is: aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	direction of alarm bubble
alarm.gradient	String, default value is none. Optional value is: linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	fill gradient type
alarm.gradient.color	Color, default value is #FFFFFF	fill gradient color
alarm.join	String, default value is miter. Optional value is: miter, round, bevel	joint point style of alarm border
alarm.outline.color	Color, default value is #000000	border color of alarm bubble
alarm.outline.width	Number, default value is -1	border width

alarm.padding	Number, default value is 0	alarm padding
alarm.padding.bottom	Number, default value is 0	alarm bottom padding
alarm.padding.left	Number, default value is 0	alarm left padding
alarm.padding.right	Number, default value is 0	alarm right padding
alarm.padding.top	Number, default value is 0	alarm top padding
alarm.pointer.length	Number, default value is 10	alarm pointer length
alarm.pointer.width	Number, default value is 8	alarm pointer width
alarm.position	String, default value is hotspot. Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	alarm position
alarm.shadowable	String, default value is true	whether shows shadowable
alarm.xoffset	Number, default value is 0	alarm xoffset
alarm.yoffset	Number, default value is 0	alarm yoffset

```

node.setStyle("alarm.position", "topleft.topleft");
node.setStyle("alarm.direction", "aboveleft");
node.setStyle("alarm.pointer.length", 20);
node.setStyle("alarm.gradient", "linear.south");
node.setStyle("alarm.gradient.color", "#EEEEEE");

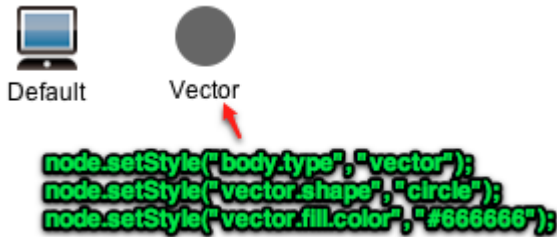
```



body style

Style (twaver.Styles#)	Evaluation	Note
--------------------------	------------	------

body.type	String, default value is default. value can be: 'none', 'default', 'vector', 'default.vector', 'vector.default'	body type
-----------	--	-----------



bus style

Style (twaver.Styles#)	Evaluation	Note
bus.style	String, default value is nearby Optional value is: 'nearby', 'north', 'south', 'west', 'east'	bus type

示例：

```
var node = new twaver.Node();
node.setSize(16, 16);
node.setLocation(200, 300);
box.add(node);

var node2 = new twaver.Node();
node2.setSize(16, 16);
node2.setLocation(100, 400);
box.add(node2);

var bus = new twaver.Bus();
bus.addPoint({x: 50, y: 350});
bus.addPoint({x: 200, y: 350});
bus.addPoint({x: 200, y: 420});
bus.addPoint({x: 50, y: 420});
bus.setStyle("vector.fill.color", "#00ff00");
bus.setStyle('vector.outline.width', 5);
bus.setStyle('bus.style', 'north');
box.add(bus);

box.add(new twaver.Link(node, bus));
box.add(new twaver.Link(node2, bus));
```



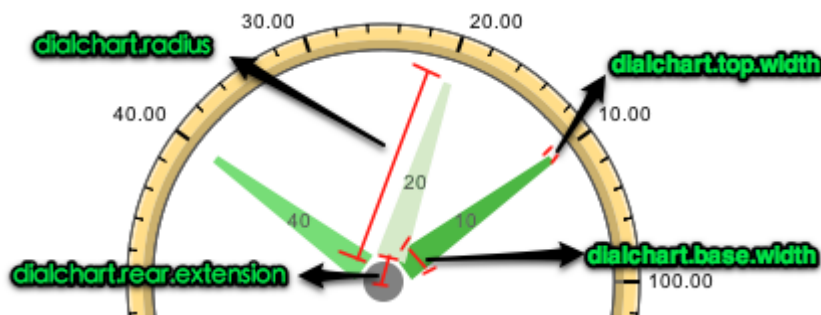


chart style

Style (twaver.Styles#)	Evaluation	Note
chart.bubble.shape	String, default value is circle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	bubble chart shape
chart.line.width	Number, default value is 2	line width
chart.marker.shape	String, default value is circle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	marker shape
chart.marker.size	Number, default value is 6	marker size
chart.value	Number, default value is 0	chart value
chart.value.color	Color, default value is #000000	chart font color

dialchart style

Style (twaver.Styles#)	Evaluation	Note
dialchart.radius	Number, default value is 0.8	pointer length percent
dialchart.base.width	Number, default value is 5	pointer width
dialchart.rear.extension	Number, default value is 0	pointer offset
dialchart.top.width	Number, default value is 0	pointer tip width



follower style

Style (twaver.Styles#)	Evaluation	Note
------------------------	------------	------

follower.column.index	Number, default value is 0	column index
follower.column.span	Number, default value is 1	span column number
follower.padding	Number, default value is 0	padding
follower.padding.bottom	Number, default value is 0	bottom padding
follower.padding.left	Number, default value is 0	left padding
follower.padding.right	Number, default value is 0	right padding
follower.padding.top	Number, default value is 0	top padding
follower.row.index	Number, default value is 0	row index
follower.row.span	Number, default value is 1	span row number

grid style

Style (twaver.Styles#)	Evaluation	Note
grid.border	Number, default value is 1	border width
grid.border.bottom	Number, default value is 0	bottom border width
grid.border.left	Number, default value is 0	left border width
grid.border.right	Number, default value is 0	right border width
grid.border.top	Number, default value is 0	top border width
grid.cell.deep	Number, default value is -1	cell deep
grid.column.count	Number, default value is 1	column count
grid.deep	Number, default value is 1	grid deep
grid.fill	String, default value is true	whether need to fill grid
grid.fill.color	Color, default value is #C0C0C0	fill color
grid.padding	Number, default value is 1	grid padding
grid.padding.bottom	Number, default value is 0	grid bottom padding
grid.padding.left	Number, default value is 0	grid left padding
grid.padding.right	Number, default value is 0	grid right padding
grid.padding.top	Number, default value is 0	grid top padding
grid.row.count	Number, default value is 1	row count

example:

```
<!DOCTYPE html>
<html>
<head>
```

```

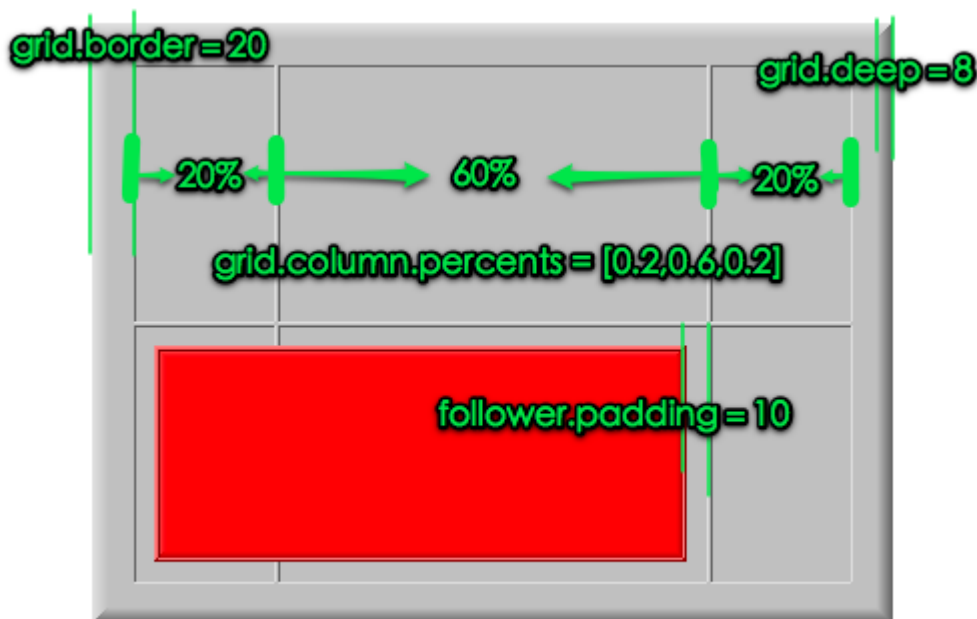
<title>TWaver HTML5 Demo</title>
<script type="text/javascript" src="../demo/twaver.js"></script>
<script type="text/javascript">
function init() {
var network = new twaver.network.Network();
var box = network.getElementBox();

var grid = new twaver.Grid();
grid.setLocation(20, 20);
grid.setSize(400, 300);
grid.setStyle("grid.border", 20);
grid.setStyle("grid.row.count", 2);
grid.setStyle("grid.column.count", 3);
grid.setStyle("grid.column.percents", [0.2, 0.6, 0.2]);
grid.setStyle("grid.deep", 8);
box.add(grid);

var cell = new twaver.Grid();
cell.setStyle("follower.column.index", 0);
cell.setStyle("follower.row.index", 1);
cell.setStyle("follower.column.span", 2);
cell.setStyle("grid.fill.color", "#ff0000");
cell.setStyle("follower.padding", 10);
cell.setHost(grid);
grid.addChild(cell);
box.add(cell);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>

```



group style

Style (twaver.Styles#)	Evaluation	Note
group.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border
group.deep	Number, default value is 1	group deep
group.fill	String, default value is true	whether fill group body
group.fill.color	Color, default value is #CCCCFF	fill color
group.gradient	String, default value is none Optional value is: linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	gradient type
group.gradient.color	Color, default value is #FFFFFF	gradient color
group.join	String, default value is miter Optional value is: miter, round, bevel	join style of border
group.outline.color	Color, default value is #5B5B5B	border color
group.outline.width	Number, default value is -1	border width
group.padding	Number, default value is 5	padding
group.padding.bottom	Number, default value is 0	bottom padding
group.padding.left	Number, default value is 0	left padding
group.padding.right	Number, default value is 0	right padding
group.padding.top	Number, default value is 0	toppadding
group.shape	String, default value is rectangle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	group shape when group is collapse

icons style

Style (twaver.Styles#)	Evaluation	Note
------------------------	------------	------

icons.orientation	String, default value is right Optional value is: left, right, top, bottom	alignment of icons
icons.position	String, default value is topleft.bottomright Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	icon position
icons.xgap	Number, default value is 1	x gap of icons
icons.xoffset	Number, default value is 0	x offset of icons
icons.ygap	Number, default value is 1	y gap of icons
icons.yoffset	Number, default value is 0	y offset of icons

image style

Style (twaver.Styles#)	Evaluation	Note
image.padding	Number, default value is 0	padding
image.padding.bottom	Number, default value is 0	bottom padding
image.padding.left	Number, default value is 0	left padding
image.padding.right	Number, default value is 0	right padding
image.padding.top	Number, default value is 0	top padding

label style

Style (twaver.Styles#)	Evaluation	Note
label.alpha	Number, default value is 1	tranparency
label.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border

label.color	Color, default value is #000000	font color
label.corner.radius	Number, default value is 0	radius of corner
label.direction	String, default value is below Optional value is: aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	label direction for network element. It is visible when set pointer length, refer to label.pointer.length
label.fill	String, default value is false	whether filling
label.fill.color	Color, default value is #C0C0C0	fill color
label.gradient	String, default value is none Optional value is: linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	fill gradient type
label.gradient.color	Color, default value is #FFFFFF	gradient color
label.join	String, default value is miter Optional value is: miter, round, bevel	join style of border
label.outline.color	Color, default value is #000000	border color
label.outline.width	Number, default value is -1	border color, blank when value is smaller than 0
label.padding	Number, default value is 0	padding
label.padding.bottom	Number, default value is 0	bottom padding
label.padding.left	Number, default value is 0	left padding
label.padding.right	Number, default value is 0	right padding
label.padding.top	Number, default value is 0	top padding
label.pointer.length	Number, default value is 0	length of pointer, default value is 0
label.pointer.width	Number, default value is 8	pointer width
label.position	String, default value is bottom.bottom Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright,	label position for network element

	topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	
label.shadowable	String, default value is true	whether show label shadow
label.xoffset	Number, default value is 0	offset in x-axis direction
label.yoffset	Number, default value is 2	offset in y-axis direction

link style

Style (twaver.Styles#)	Evaluation	Note
link.bundle.enable	String, default value is true	whether join binding
link.bundle.expanded	String, default value is true	whether expand
link.bundle.gap	Number, default value is 12	bundle gap
link.bundle.id	Number, default value is 0	the id of bundle group
link.bundle.independent	String, default value is false	whether bundle this group independent
link.bundle.offset	Number, default value is 20	offset of bundle link
link.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border
link.color	Color, default value is #658DC1	link color
link.corner	String, default value is round Optional value is: none, round, bevel	link corner type
link.extend	Number, default value is 20	link extend value
link.from.at.edge	String, default value is true	whether start point position is at the edge
link.from.position	String, default value is center Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright,	start point position

	top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	
link.from.xoffset	Number, default value is 0	start point offset in x-axis
link.from.yoffset	Number, default value is 0	start point offset in y-axis
link.handler.alpha	Number, default value is 1	link handler transparency
link.handler.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of link handler border
link.handler.color	Color, default value is #000000	handler font color
link.handler.corner.radius	Number, default value is 0	radius of handler corner
link.handler.direction	String, default value is below Optional value is: aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	handler direction
link.handler.fill	String, default value is false	filling handler
link.handler.fill.color	Color, default value is #C0C0C0	fill color for handler
link.handler.gradient	String, default value is none Optional value is: linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	handler gradient type
link.handler.gradient.color	Color, default value is #FFFFFF	handler gradient color
link.handler.join	String, default value is miter Optional value is: miter, round, bevel	join style of link handler
link.handler.outline.color	Color, default value is #000000	border color of link handler

link.handler.outline.width	Number, default value is -1	border width of link handler
link.handler.padding	Number, default value is 0	link handler padding
link.handler.padding.bottom	Number, default value is 0	link handler bottom padding
link.handler.padding.left	Number, default value is 0	link handler left padding
link.handler.padding.right	Number, default value is 0	link handler right padding
link.handler.padding.top	Number, default value is 0	link handler top padding
link.handler.pointer.length	Number, default value is 0	pointer length of link handler
link.handler.pointer.width	Number, default value is 8	pointer width of link handler
link.handler.position	String, default value is topleft.topleft Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	link handler position
link.handler.shadowable	String, default value is true	whether show shadow of link handler
link.handler.xoffset	Number, default value is 0	x offset of link handler
link.handler.yoffset	Number, default value is 0	y offset of link handler
link.join	String, default value is miter Optional value is: miter, round, bevel	joint style of link
link.looped.direction	String, default value is northwest Optional value is: north, northeast, east, southeast, south, southwest, west, northwest	direction of loop link
link.looped.gap	Number, default value is 6	the gap of loop link
link.looped.type	String, default value is arc	loop link type
link.split.by.percent	String, default value is true	split link by percent
link.split.percent	Number, default value is 0.5	the percent of split link

link.split.value	Number, default value is 20	split value of link
link.to.at.edge	String, default value is true	whether the endpoint is at the edge
link.to.position	String, default value is center Optional value is: topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	position of endpoint
link.to.xoffset	Number, default value is 0	endpoint offset in x-axis direction
link.to.yoffset	Number, default value is 0	endpoint offset in y-axis direction
link.type	String, default value is arc Optional value is: arc, triangle, parallel, flexional, flexional.horizontal, flexional.vertical, orthogonal, orthogonal.horizontal, orthogonal.vertical, orthogonal.H.V, orthogonal.V.H, extend.top, extend.left, extend.bottom, extend.right	link type
link.width	Number, default value is 3	link width
link.xradius	Number, default value is 8	x-axis radius of link
link.yradius	Number, default value is 8	y-axis radius of link

outer style

Style (twaver.Styles#)	Evaluation	Note
outer.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border
outer.join	String, default value is miter Optional value is: miter, round, bevel	join style of border
outer.padding	Number, default value is 1	padding

outer.padding.bottom	Number, default value is 0	bottom padding
outer.padding.left	Number, default value is 0	left padding
outer.padding.right	Number, default value is 0	right padding
outer.padding.top	Number, default value is 0	top padding
outer.shape	String, default value is rectangle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	border shape
outer.width	Number, default value is 2	border width

select border style

Style (twaver.Styles#)	Evaluation	Note
select.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border
select.color	String, default value is rgba(0, 0, 0, 0.7)	color
select.join	String, default value is miter Optional value is: miter, round, bevel	join style of border
select.padding	Number, default value is 2	padding
select.padding.bottom	Number, default value is 0	bottom padding
select.padding.left	Number, default value is 0	left padding
select.padding.right	Number, default value is 0	right padding
select.padding.top	Number, default value is 0	top padding
select.shape	String, default value is rectangle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	select shape
select.style	String, default value is shadow Optional value is: null, shadow, border	select style
select.width	Number, default value is 2	select border width

shadow style

Style (twaver.Styles#)	Evaluation	Note
shadow.blur	Number, default value is 6	blur value
shadow.xoffset	Number, default value is 3	xoffset of shadow

shadow.yoffset	Number, default value is 3	yoffset of shadow
----------------	----------------------------	-------------------

shapelink style

Style (twaver.Styles#)	Evaluation	Note
shapelink.type	String, default value is lineto Optional value is: 'lineto', 'quadto', 'cubicto'	shapeLink type

shapenode style

Style (twaver.Styles#)	Evaluation	Note
shapenode.closed	String, default value is false	whether closed

vector

Style (twaver.Styles#)	Evaluation	Note
vector.cap	String, default value is butt Optional value is: butt, round, square	endpoint style of border
vector.deep	Number, default value is 0	the depth of vector
vector.fill	String, default value is true	fill vector
vector.fill.color	Color, default value is #CCCCFF	fill color
vector.gradient	String, default value is none Optional value is: linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	gradient
vector.gradient.color	Color, default value is #FFFFFF	gradient color
vector.join	String, default value is miter Optional value is: miter, round, bevel	join style of border
vector.outline.color	Color, default value is #5B5B5B	outline color
vector.outline.width	Number, default value is -1	outline width

vector.padding	Number, default value is 0	padding
vector.padding.bottom	Number, default value is 0	bottom padding
vector.padding.left	Number, default value is 0	left padding
vector.padding.right	Number, default value is 0	right padding
vector.padding.top	Number, default value is 0	top padding
vector.shape	String, default value is rectangle Optional value is: rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	vector shape

whole

Style (twaver.Styles#)	Evaluation	Note
whole.alpha	Number, default value is 1	transparency