Front-end Web Development Example

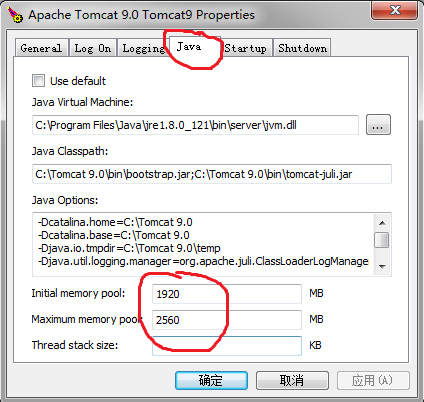
**Note:**

This document shows you how to do front-end web development.

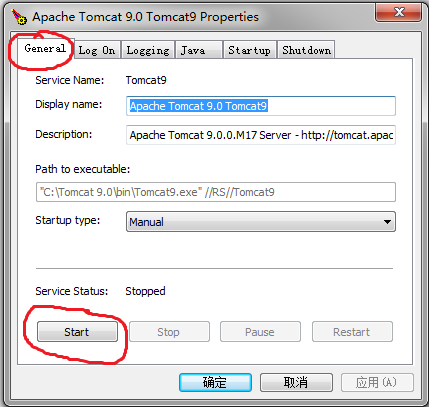
## Preparation

1. Install JAVA SDK and TOMCAT.
2. Copy JAVA file lfy.jar to directory webapps\ROOT\WEB-INF\lib of TOMCAT installation directory. Copy JSP file interface.jsp to directory webapps\ROOT\ of TOMCAT installation directory.
3. Extract scene data files from file data.rar to *any* directory. Create temporary directory for storing temporary data.
4. Configure scene data through modifying interface.jsp.
5. Start TOMCAT

First set JAVA memory pool as following dialog.



Then start TOMCAT as following dialog.



Now web server has been started. You are ready to do your front-end development.

If your web server is running at remote computer and maintained by other people, you can ignore the preparation above and do you front-end web development directly.

## How to do front-end web development

The WebGL draws 3D-content to a special Web tag <canvas>. To exploit WebGL in you web page, you should first declare a <canvas> tag in body of you webpage, and you <canvas> tag must have a tabindex:

*<canvas id="my\_canvas"* tabindex="0" *width=1500 height=500></canvas>*

To exploit client-side JavaScript program, you should reference it in the head of your web page, this can be done by <script> tag like this:

<script type="text/javascript" src="http://localhost:8080/interface.jsp?function\_name=construct\_render\_object"></script>

function\_name identifies initialization function name

And also you should declare an initialization function by <script> tag like this:

*<script type="text/javascript">*

*var render\_object;*

*function body\_onload()*

*{*

*construct\_render\_object(*

*// initialization function name , identified by function\_name above*

*canvas\_object, //canvas object*

*user\_name,pass\_word, //user name and passwprd*

*language\_name, // language name*

*scene\_name, link\_name, //scene name and link name*

*initialization\_parameter, // initialization\_parameter*

*initialization\_function // initialization complete execution function*

*);*

*}*

*</script>*

Scene initialization is done by calling function *construct\_render\_object()*. Its parameters include:

1. Canvas object: Usually obtained by *document.getElementById()*; This parameter identifies where to draw the scene.
2. User name: identify who you are.
3. Password: identify password.
4. Language name: *“english”* or *“chinese”.* This parameter identifies which language is used to display information.
5. Scene name: This parameter identifies which scene to create by its scene name.
6. Link name: This parameter is for scene sharing. If two client browsers view a scene with same scene name and same link name, they will share the scene.
7. *Initialization parameter: this parameter is an array. If it is not an array, this parameter will be ignored. Every element of this parameter is also an array with two items. The first item is parameter name, the second parameter is parameter value. Client-side engine will code the parameters into URL and transfer them to server-side engine. Server-side component driver and instance driver can get parameter value through get\_parameter() method of* client\_request\_response object.
8. initialization\_function: When scene initialization is done. Client-side program calls this function after creating render object. Usually in this function saves scene object in a variable. In following example above, saves scene object to a variable render\_object.

## Example

Put the following code in an html file, modify web-server URL. Then use your browser to view the webpage.

You can deploy your front-end html file at different website from your WebGL engine server. You can even deploy your front-end html file at local computer without any web server.

<html>

<head>

<script type="text/javascript" src="./interface.jsp?function\_name=construct\_render\_object"></script>

<script type="text/javascript">

var render\_object;

function body\_onload()

{

construct\_render\_object(

document.getElementById("my\_canvas"),

"NoName","NoPassword","chinese",

"","",

[],

function(my\_render\_object)

{

render\_object=my\_render\_object;

document.title=render\_object.title;

}

);

}

</script>

</head>

<body onload="body\_onload();">

<canvas id="my\_canvas" tabindex="0" width=1250 height=500></canvas>

</body>

</html>

## Try some modifications and see result

1. Try to modify scene name to the following names. These tries show you how to view different scenes.

map\_environment\_1 map\_environment\_2

map\_environment\_3 map\_environment\_4

bridge\_standard\_16 bridge\_standard\_16.72

bridge\_standard\_17.44 bridge\_standard\_18

bridge\_standard\_24 bridge\_standard\_32

bridge\_fish\_16 bridge\_fish\_16.72

bridge\_fish\_17.44 bridge\_fish\_18

bridge\_fish\_24 bridge\_fish\_32

1. Use “test” as scene name. Try to modify “Change part name” to the following names. These tries show you how to view use different in a scene.

display\_part:obj\_format\_file\_military\_part\_0

display\_part:obj\_format\_file\_military\_part\_1

display\_part:obj\_format\_file\_military\_part\_2

display\_part:obj\_format\_file\_military\_part\_3

display\_part:lession\_4\_video\_as\_texture\_part\_1

display\_part:lession\_4\_video\_as\_texture\_part\_2

display\_part:lession\_18\_particle\_system\_part\_1

display\_part:lession\_16\_use\_texture\_in\_vertex\_shader\_part\_0

display\_part:lession\_16\_use\_texture\_in\_vertex\_shader\_part\_1

## Add Interactivity

To test Interactivity, first add several button tags and processing functions to our HTML source file as following code. When users click on these buttons, the corresponding function will be executed.

<html>

<head>

<script type="text/javascript" src="http://localhost:8080/interface.jsp?function\_name=construct\_render\_object"></script>

<script type="text/javascript">

var render\_object;

function body\_onload()

{

construct\_render\_object(

document.getElementById("my\_canvas"),

"NoName","NoPassword","chinese",

"test","",[],

function(my\_render\_object)

{

render\_object=my\_render\_object;

document.title=render\_object.title;

}

);

}

function process\_button\_1()

{

};

function process\_button\_2()

{

};

function process\_button\_3()

{

};

</script>

</head>

<body onload="body\_onload();">

<canvas id="my\_canvas" tabindex="0" width=1250 height=500></canvas>

<br/>

<input name="buttion\_1" id="buttion\_1" type="button" value="button\_1"

onclick="process\_button\_1();"></input>

<input name="buttion\_2" id="buttion\_2" type="button" value="button\_2"

onclick="process\_button\_2();"></input>

<input name="buttion\_3" id="buttion\_3" type="button" value="button\_3"

onclick="process\_button\_3();"></input>

</body>

</html>

## Interactivity function

Interactivity is implemented through call\_server\_component function of render\_object, its format is:

render\_object.call\_server\_component (

component\_name, // call which component

driver\_id, // call which driver of the component

component\_parameter, //parameter

response\_function, //execute this function when success

error\_function //execute this function when fail

);

* component\_name identifies which component to call.
* driver\_id identifies which driver of the component to call.
* component\_parameter is an array, its element is also an array with two strings, one string is parameter name, the other is parameter value.
* The last two parameters are functions. When success, response\_function is execute, when fail, error\_function is execute.

Different components implement different functions. By calling render\_object.call\_server\_component, users can call component special functions. Next we show some examples. For more component function, see their manuals.

## Try some Interactivity and see result

* 1. Turn on or off background image

Modify the two function as following code. The two functions turn on or off background image by calling component "background\_component\_0".

You can try it by clicking the two buttons and see the result.

function process\_button\_1()

{

render\_object.call\_server\_component (

"background\_component\_0","0",

[

["operation", "turnonoff" ],

["state", "on" ]

]

);

};

function process\_button\_2()

{

render\_object.call\_server\_component (

"background\_component\_0","0",

[

["operation", "turnonoff" ],

["state", "off" ]

]

);

};

* 1. Turn on or off coordinate

Modify the two function as following code. The two functions turn on or off coordinate by calling component " coordinate ".

You can try it by clicking the two buttons and see the result.

function process\_button\_1()

{

render\_object.call\_server\_component (

"coordinate","0",

[

["camera\_display\_flag","off"]

]

);

};

function process\_button\_2()

{

render\_object.call\_server\_component (

"coordinate","0",

[

["camera\_display\_flag","on"]

]

);

};

* 1. Mount Bridge

Modify scene name to :

bridge\_old\_32

Modify the three function as following code. The three functions start bridge mounting, stop bridge mounting or start bridge dismounting.

You can try it by clicking the buttons and see the result.

function process\_button\_1()

{

render\_object.call\_server\_component (

"movement\_manager","0",

[

["operation", "movement"],

["move\_method", "continue"],

["direct", "forward"],

["time", "1000"]

]

);

};

function process\_button\_2()

{

render\_object.call\_server\_component (

"movement\_manager","0",

[

["operation", "movement"],

["move\_method", "continue"],

["direct", "backward"],

["time", "1000"]

]

);

};

function process\_button\_3()

{

render\_object.call\_server\_component (

"movement\_manager","0",

[

["operation", "movement"],

["move\_method", "stop"]

]

);

};