Tags Support for JavaScript

Table of contents

1. Introduction

The following topic explains how the NetUI JSP tags provide support for client side JavaScript development. The primary JavaScript support provides the ability to looking up HTML DOM elements produced by the tags. Lookup may be done by either the **id** or **name** attributes. In addition, the NetUI tags provide the ability to generate unique values for the HTML **id** attribute.

NetUI owns the value of the **name** attribute generated for the HTML form controls. It's very common to write JavaScript that would like to access these control DOM elements using the name. The NetUI JavaScript support provides a mapping between the local name and the generated name.

In addition, NetUI provides scoping facilities that ensure the HTML **id** attribute is unique as required by the HTML specifications. In a simple page, this is usually not a problem because all of the **id** attributes are specified in a single source file. There are situations, such as content within a portal, where a NetUI page is generated as a portion of a larger page. In these cases, unique **id** values must be generated when the page is generated. The JavaScript support in NetUI allows for looking up DOM elements in JavaScript using a "local" name and scope. This allows JavaScript to be written as if the page was a simple page and for it to become part of a composite page without modification.

Note: The pre-Beehive NetUI tags provided a different set of routines for looking up DOM elements. A configuration option is available which will turn this mode on. For more information see the id-javascript section of the configuration reference page.

2. Backward Compatibility

There are two JavaScript functions intended to support client side JavaScript, **lookupIdByTagId()** and **lookupNameByTagId()**. These are the public interfaces provided to lookup HTML DOM elements and Beehive will maintain backward compatibility on these methods. All of the other JavaScript methods and the values of the HTML **name** and **id** attributes output into the generated HTML may change over time. Developers should not build dependencies on these values. In some cases, this document describes the details of these low level features for explanation purposes.

3. JSP Tags and Attribute

This section describes the primary NetUI JSP tags and attributes used to support JavaScript.

3.1. tagId

The NetUI tags usually do not allow setting the **id** attribute directly. Instead, the generated HTML elements are given "local" names using the **tagId** attribute. The value of this attribute is used to generate the **id** attribute of the resulting HTML elements. In addition, for form controls, a mapping is maintained from the **tagId** value to the generated **name**.

3.2. ScriptContainer

Scoping of the generated HTML **id** attributes is done by the **ScriptContainer** tags. A ScriptContainer provides either an explicit name or a uniquely generated name. All of the tagId values found in its decedents will have this name prefixed to their tagId value when their qualified **id** value is created.

There are two NetUI JSP tags that are ScriptContainers, <netui:scriptContainer> and <netui:html>. The ScriptContainer tag represents the basic features. The Html tag is a subclass of ScriptContainer and provides additional HTML specific features. ScriptContainers may be nested. The **id** value is generated by prefixing all of the ancestor ScriptContainer names to the value of the **tagId**.

The ScriptContainer tags support two attributes for creating a name, **idScope** and **generateIdScope**. The **idScope** sets an explicit value for the generated scope. The **generateIdScope**, when **true**, will cause a unique id scope name to be generated. This value is unique for the request.

The NetUI tags dynamically generate JavaScript in certain situations. The top most ScriptContainer gathers up generated JavaScript then renders it into the HTML document. There isn't a requirement to have a ScriptContainer for most of the JavaScript enabled features; this includes the lookup methods. If the ScriptContainer is not present, the generated JavaScript will be output inline with the tags. The result is duplicate JavaScript will be rendered throughout the page for features like setting **tagId** or image rollovers. The advantages of using a ScriptContainer is this JavaScript is not duplicated and will be written out in a single script block in the generated HTML file.

Note: the generated id scope may not be the same from request to request on the same page. It may change if content preceding the JSP changes from request to request, or some conditional logic on the page, such as JSTL, changes the number or order of ScriptContainers.

3.3. ScriptBlock

The <netui:scriptBlock> tag is used to position user provided JavaScript in relationship to the framework generated JavaScript. This tag supports inserting JavaScript **before** or **after** the framework generated JavaScript. When you want to create valid XHTML or HTML this tag

is required. See the HTLM and XHTML Support topic for more information on creating valid XHTML and HTML using the NetUI JSP tags and information on the ScriptBlocks role.

4. Simple Example

This section presents a very simple example which demonstrates the use of the **tagId** attribute, scoping ids, and the lookup methods. In this example there is a simple form containing a checkbox. The document is automatically scoped to generate unique ids within the document.

4.1. JSP Code

This JSP file demonstrates the use of **tagId** and scoping. It also contains JavaScript that looks up DOM elements and dynamically creates some output verifying these elements are found. The screen shot at the end of this document displays the expected dynamically generated content.

tagIdSupport.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib prefix="netui"
uri="http://beehive.apache.org/netui/tags-html-1.0"%>
<netui:html generateIdScope="true">
    <head>
        <title>tagIdSupport</title>
        <netui:base/>
    </head>
    <netui:body>
        <netui:form tagId="form" action="begin">
        <span id="scopeOneSpan" />
        Checkbox:<netui:checkBox tagId="check" dataSource="pageFlow.check"
/>
         <netui:button type="submit" value="Post Form" />
        </netui:form>
        <hr>
        <p style="color: green">
        The following calls the generated JavaScript support to lookup DOM
elements.
        </p>
        <p id="javaOut"></p>
    <netui:scriptBlock placement="after">

    // find the element that will provide the output <p> and the scope for
lookup scope
    var p = document.getElementById("javaOut");
    var scope = document.getElementById("scopeOneSpan");
```

```
    // create the contents of the output paragraph
    var val = "<b>Lookup the Form by name and id</b><br><br>";
    var form = document.getElementById(lookupIdByTagId("form",scope));
    val = val + "Form by <b>id</b> is: " +
        form.id + ", type: " + form.nodeName + "<br>";
    form = document[lookupNameByTagId("form",scope)];
    val = val + "Form by <b>name</b> is: " +
        form .name + ", type: " + form.nodeName +  "<br>";

    val = val + "<br><b>Lookup Checkbox by name and id</b><br><br>";
    var check = document.getElementById(lookupIdByTagId("check",scope));
    val = val + "Checkbox by <b>id</b> is: " +
        check.id + ", type: " + check.nodeName + "<br>";
    check = form[lookupNameByTagId("check",scope)];
    val = val + "Checkbox) by <b>name</b> is: " +
        check .name + ", type: " + check.nodeName +  "<br>";
    p.innerHTML = val;

    </netui:scriptBlock>
    </netui:body>
</netui:html>
```

In the JSP code, the <netui:form> and <netui:checkbox> elements are both named using the **tagId** attribute. The values of the tagId ("form" and "check") are the local name of these elements.

```
<netui:form tagId="form" action="begin">
<netui:checkBox tagId="check" dataSource="pageFlow.check" />
```

In this example, the <netui:html> tag generates a scope name for all **tagId** values. This scoping isn't required in this example, but demonstrates the use of scoping within a page. The **generateIdScope** attribute on the <netui:html> tag will automatically generate a unique "name" for the scope defined by the tag. The fully qualified name for each DOM element is creating by prefixing the local name with each scope name defined by all ancestor ScriptContainer JSP tags. Prefixing is done moving up the hierarchy until the root is found. In this example, the <netui:html> tag will generate a name of **n0**. The generated name for the <netui:form> tag will then become **n0.form**. The '.' character is used to separate the names.

```
<netui:html generateIdScope="true">
```

The source JavaScript in the page is found inside of a ScriptBlock. The reason for using a ScriptBlock is to control the placement of the script in relationship to the framework generated lookup functions. In this example the **placement** attribute is set to the value **after** indicating that the contents of this block should appear after the framework generated JavaScript. This is required because the JavaScript contains calls to functions which are generated by the framework.

```
<netui:scriptBlock placement="after">
```

When one of the NetUI JSP tags contains a **tagId** attribute, the framework will generate some JavaScript which supports looking up DOM elements based upon the local name and a scope. There are two primary JavaScript functions generated, **lookupIdByTagId** and **lookupNameByTagId**. The former will return the real id of a DOM element based upon the tagId value and the later will return the real name based upon the tagId value. In the example, we lookup the form based upon by id. Then we lookup the DOM input element representing the checkbox based upon its name.

```
var form = document.getElementById(lookupIdByTagId("form",scope));
check = form[lookupNameByTagId("check",scope)];
```

4.2. Controller

The Page Flow controller simply supports the sample. It defines the required begin action and a property bound to by the <netui:checkbox>.

Controller.java

```
package tagIdSupport;

import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="tagIdSupport.jsp")
    }
)
public class Controller
    extends PageFlowController
{
    private boolean _check;

    public boolean isCheck() {
        return _check;
    }

    public void setCheck(boolean check1) {
        _check = check1;
    }
}
```

4.3. Generated HTML

This section contains the generated HTML page sent to the browser. The output of the NetUI JSP tags and the literal content are contained in the page. The tagId has been used to name HTML elements and the framework generated JavaScript is present.

Generate HTML on Client Browser

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
>
<html lang="en" netui:idScope="n0">

    <head>
        <title>tagIdSupport</title>
        <base
href="http://localhost:8080/dev/tagIdSupport/tagIdSupport.jsp">
    </head>
    <body>
        <form name="n0.form" id="n0.form"
action="/dev/tagIdSupport/begin.do" method="post">
        <span id="scopeOneSpan" />
        Checkbox:<input type="hidden"
name="wlw-checkbox_key:{pageFlow.check}OldValue" value="false"><input
type="checkbox" name="wlw-checkbox_key:{pageFlow.check}" id="n0.check">
         <input type="submit" value="Post Form">
        </form>
        <hr>
        <p style="color: green">
        The following calls the generated JavaScript support to lookup DOM
elements.
        </p>
        <p id="javaOut"></p>

    <script language="JavaScript" type="text/JavaScript">
<!--

// **** Start the NetUI Framework Generated JavaScript ****

// Build the netui_names table to map the tagId attributes
// to the real id written into the HTML
if (netui_tagIdNameMap == null)
   var netui_tagIdNameMap = new Object();
netui_tagIdNameMap.n0___form="n0.form"
netui_tagIdNameMap.n0___check="wlw-checkbox_key:{pageFlow.check}"


// lookup by tagId to "real id"
function lookupIdByTagId(id, tag)
{
   var idScope = lookupIdScope(tag,".");
   return (idScope == "") ? id : idScope + id;
}

// lookup by tagId to "real name"
function lookupNameByTagId(id, tag)
{
   var idScope = lookupIdScope(tag,"_");
   if (idScope == "")
      return netui_tagIdNameMap[id];
```

```
   else
      return netui_tagIdNameMap[idScope  + "__" + id];
}

//Non-Legacy lookup method creating a fully qualified scope id
function lookupIdScope(tag,sep)
{
   var val = "";
   if (sep == null) sep = "";
   while (tag != null && tag.getAttribute != null) {
      try {
         var attrVal = tag.getAttribute("netui:idScope");
      } catch (e) { /* ignore, in IE6 calling on a table results in an
exception */ }
      if (attrVal != null)
         val = attrVal + sep + val;
      tag = tag.parentNode;
   }
   return val;
}
-->
</script>
<script language="JavaScript" type="text/JavaScript">
<!--
    // find the element that will provide the output <p> and the scope for
lookup scope
    var p = document.getElementById("javaOut");
    var scope = document.getElementById("scopeOneSpan");

    // create the contents of the output paragraph
    var val = "<b>Lookup the Form by name and id</b><br><br>";
    var form = document.getElementById(lookupIdByTagId("form",scope));
    val = val + "Form by <b>id</b> is: " +
        form.id + ", type: " + form.nodeName + "<br>";
    form = document[lookupNameByTagId("form",scope)];
    val = val + "Form by <b>name</b> is: " +
        form .name + ", type: " + form.nodeName +  "<br>";

    val = val + "<br><b>Lookup Checkbox by name and id</b><br><br>";
    var check = document.getElementById(lookupIdByTagId("check",scope));
    val = val + "Checkbox by <b>id</b> is: " +
        check.id + ", type: " + check.nodeName + "<br>";
    check = form[lookupNameByTagId("check",scope)];
    val = val + "Checkbox) by <b>name</b> is: " +
        check .name + ", type: " + check.nodeName +  "<br>";
    p.innerHTML = val;
-->
</script>
</body>
</html>
```

In the generated HTML, the names and ids of the form and checkbox input elements have been generated by the NetUI JSP tags. The **n0** portion of the name is prefixed to the tagId because the <netui:html> tag generated that value. For the checkbox, the name is generated

by the JSP tag and contains an expression allowing the NetUI **check** property to be updated by the NetUI framework.

```
<form name="n0.form" id="n0.form" action="/dev/tagIdSupport/begin.do"
method="post">
<input type="checkbox" name="wlw-checkbox_key:{pageFlow.check}"
id="n0.check">
```

In addition, the framework generated JavaScript and the JavaScript defined in the JSP have the proper relationship with each other. The framework JavaScript precedes the JSP defined JavaScript.

Finally, the <html> element has an additional attribute name spaced into NetUI. In many cases the NetUI tags generate additional attributes on some HTML elements to provide information to JavaScript. These attributes will always be name spaced into the **netui** name space. In this case, the **netui:idScope** attribute is used by the lookup code to create the fully qualified name.

```
<html lang="en" netui:idScope="n0">
```

4.4. Framework Generated JavaScript

This section describes the details of the framework generated JavaScript. This JavaScript was generated because NetUI tags used the **tagId** attribute. This JavaScript will not be generated in pages that do not use tagId.

Framework Generated JavaScript

```
// **** Start the NetUI Framework Generated JavaScript ****

// Build the netui_names table to map the tagId attributes
// to the real id written into the HTML
if (netui_tagIdNameMap == null)
   var netui_tagIdNameMap = new Object();
netui_tagIdNameMap.n0___form="n0.form"
netui_tagIdNameMap.n0___check="wlw-checkbox_key:{pageFlow.check}"


// lookup by tagId to "real id"
function lookupIdByTagId(id, tag)
{
   var idScope = lookupIdScope(tag,".");
   return (idScope == "") ? id : idScope + id;
}

// lookup by tagId to "real name"
function lookupNameByTagId(id, tag)
{
   var idScope = lookupIdScope(tag,"_");
```

```
   if (idScope == "")
      return netui_tagIdNameMap[id];
   else
      return netui_tagIdNameMap[idScope  + "__" + id];
}

//Non-Legacy lookup method creating a fully qualified scope id
function lookupIdScope(tag,sep)
{
   var val = "";
   if (sep == null) sep = "";
   while (tag != null && tag.getAttribute != null) {
      try {
         var attrVal = tag.getAttribute("netui:idScope");
      } catch (e) { /* ignore, in IE6 calling on a table results in an
exception */ }
      if (attrVal != null)
         val = attrVal + sep + val;
      tag = tag.parentNode;
   }
   return val;
}
```

There are two JavaScript functions intended to support user JavaScript, **lookupIdByTagId** and **lookupNameByTagId**. Both methods take a name and scope. The name should be the local name of the DOM element you want to lookup. The scope is some type of DOM element that is found in the same scope as the element that is to be looked up. This is typically a button or link that is found in the same scope.

```
<a onclick="someFunction(this);return false;" href="#">
```

The user defined JavaScript function **someFunction** receives a reference to the anchor DOM element that can then be used to lookup other DOM elements in the same scope.

These two methods act as the public interface to the framework lookup code. NetUI will maintain backward compatibility for these two methods. The other code generated by the framework and the value of the **name** and **id** attributes may change over time. These features should be considered "internal". Developers should not build dependence on these features of the framework generated JavaScript support.

Note: In the example, we are using an element found in the document using the **id** attribute value. This is actually not a typical usage because the id value is not scoped by the framework. This JSP could not be included twice in a composite page because it would result in the same `id` appearing multiple times.

4.5. Browser Results

This section contains a screen shot of the page from the browser. The dynamically generated

content can be seen.

Page as seen in a browser

JavaScript result in the browser