Circular-Memory-Leak Mitigation: Windows® Internet Explorer® 8

Beta 1 for Developers

Making the Web Work for You

March 2008

For more information, press only:
Rapid Response Team
Waggener Edstrom Worldwide
(503) 443-7070
rrt@waggeneredstrom.com

OVERVIEW

Windows Internet Explorer 8 Beta 1 for Developers includes improvements to memory management. These improvements mitigate memory leaks that were previously created by circular references between Microsoft® JScript® objects and DOM objects.

COMPATIBILITY: Changes in Behavior from Internet Explorer 7

This feature may affect the behavior of Web pages that depend on garbage memory that, only as the result of a memory leak, existed in previous versions of Internet Explorer. In Internet Explorer 8 Beta 1 for Developers, these pages will reference unallocated memory and generate a fault.

This feature affects each Web site displayed in Internet Explorer 8 Beta 1 for Developers regardless of the site's chosen rendering mode.

FEATURE DETAILS

As described in detail in this MSDN article, the JScript garbage collector in previous versions of Internet Explorer manages the lifetime of JScript objects but not of DOM objects. As a result, the JScript garbage collector cannot break circular references between DOM objects and JScript objects, and memory leaks occur. In Internet Explorer 6, these circular references are broken when the Internet Explorer process terminates. In Internet Explorer 7, these circular references are broken when users navigate away from page that contain the leaks.

In Internet Explorer 8 Beta 1 for Developers, the JScript garbage collector treats DOM objects referenced by JScript objects as any other JScript object. Rather than wait until page navigation as in Internet Explorer 7 or process termination as in Internet Explorer 6, the garbage collector manages the lifetime of these DOM objects, and breaks circular references whenever possible throughout the lifetime of the site.

While Web developers should be aware of memory leaks created by use of programming patterns such as JScript closures in Internet Explorer 7 and earlier, those patterns will no longer result in leaks in Internet Explorer 8 Beta 1 for Developers.

Code Sample

This section contains simplified examples of code patterns that would previously result in memory leaks but will not leak in Internet Explorer 8 Beta 1 for Developers.

Direct Circular-Memory References

- Circular reference with DOM objects referring to itself and object not in the tree

```
function leaktest1()
{
    var elem1 = document.createElement("DIV");
    elem1.thing = elem1;
}
```

- Circular reference between DOM objects not in tree

```
function leaktest2()
{
    var elem1 = document.createElement("DIV");
    var elem2 = document.createElement("DIV");
    elem1.thing = elem2;
    elem2.item = elem1;
}
```

- Circular reference between DOM object and JScript object

```
function leaktest3()
{
    var x = new Object();
    x.obj = document.createElement("DIV");
    x.obj.jsobj = x;
}
```

- Circular reference between DOM objects when in temporary markup

```
function leaktest4()
{
    var elem1 = document.createElement("DIV");
    var elem2 = document.createElement("DIV");
    elem1.appendChild(elem2);
    elem1.thing = elem2;
    elem2.item = elem1;
}
```

- Circular reference between DOM objects when removed from the tree by using removeNode

```
function leaktest5()
{
    var elem1 = document.createElement("DIV");
    document.body.appendChild(elem1);
    elem1.thing = elem1;
    elem1.removeNode(true);
}
```

- Circular reference between DOM objects when removed from the tree using innerHTML

```
function leaktest6()
{
    var elem1 = document.createElement("DIV");
    document.body.appendChild(elem1);
    elem1.thing = elem1;
    elem1.parentElement.innerHTML = "";
}
```

Circular-Memory References Created by Closures

- Closure with element created dynamically

```
function leaktest7()
{
    var elem = document.createElement("DIV");
    elem.onload = function (){
      var y = elem;
    }
}
```

- Closure with element added to tree and removed using removeNode

```
function leaktest8(){
    var elem = document.createElement("DIV");
    document.body.appendChild(elem);
    elem.onload = function () {
      var y = elem;
    }
    elem.removeNode();
}
```

- Closure with element added to tree and removed using innerHTML

```
function leaktest9()
{
    var elem = document.createElement("DIV");
    document.body.appendChild(elem);
    elem.onload = function () {
      var y = elem;
    }
    elem.parentElement.innerHTML = "";
}
```

- Leaks caused by function pointers

```
function leaktest10()
{
    var elem = document.createElement("DIV");
    elem.thing = elem.setAttribute;
}
```