For Review

Web Controls as a Replacement for ActiveX in the MOS Protocol

Enhancements to the MOS protocol to deprecate ActiveX based plugins in place of web controls.

By Shawn Snider
Team Leader, Inception
Ross Video Limited
August 13, 2013

Contents

The Problem with ActiveX

ActiveX is an old technology, widely used in the late 1990's and early 2000's, but has since fallen out of favor for numerous reasons, notably it's gaping security flaws, and lack of support on platforms other then Windows desktop PC's. In the coming years, it is expected that Microsoft will phase this technology out altogether, limiting our options as it is a required part of the existing MOS specification for device plugins.

Other Plugin-Based Technologies

During my research for this whitepaper, various other replacement technologies have been considered, including Flash/Flex, Java based technologies such as applets and OSGI-based plugins, and even browser-based plugin technology. Unfortunately, none of these options are particularly flexible, and there is a high probability that most of these technologies will be unfeasible in the next ten years as plugin based technologies, ultimately suffering the same fate as ActiveX.

Deprecating ActiveX

The proposal put forth is not an immediate replacement to ActiveX, it is an alternative technology in which to design and build device plugins, and a standard that can be adopted to eventually replace ActiveX for new plugins. Unfortunately, due to the existing established base of technology and legacy devices, it is unlikely we will be able to remove all support for ActiveX based technologies in the protocol for the foreseeable future. However, this part of the specification should be deprecated, and developers and device vendors should be highly encouraged to adopt the new technology platform for all future plugins, and possibly adapt existing products still in development where resources allow.

HTML-Based Web Controls

This document outlines support, including sample proof-of-concept implementations of HTML-based device plugins. There are many benefits to this technology over ActiveX, they are extremely customizable and flexible, the technology is will understood and continuing to be evolved, and allow for rich creative freedom. They are also much more stable than native plugins, which is beneficial to the user experience as a whole. They also open up MOS-based plugin technologies to other platforms aside from Windows, including Mac and Linux desktops, tablets, and mobile devices, as well as potential future devices.

The specification will note various points of integration for data communication between the NCS and device plugins, and will address some of the unique challenges of web-based technologies such as the same-origin policy and cross-domain restrictions and drag and drop behavior.

The HTML-based containers to be implemented inside the NCS to host the device-based plugins should support HTML5, and be of at least IE8 compatibility or higher (or Firefox, Chrome versions of a similar era). The inclusion of support for additional plugins embedded within a device "webpage" (such as Flash, VLC, etc.) is to be determined.

Cross Site Scripting and Cross Domain Restrictions

One of the key challenges with integrating messaging between web-based controls hosted on different systems are overcoming restrictions on cross-domain communication. Typically, we would love an NCS to be able to invoke any JavaScript method on a device control that the specification defined in the interface, however due to these restrictions this is not possible. We are going to overcome these restrictions by effectively proxying MOS messages between the NCS and devices through a standardized messaging channel defined in the HTML specification for cross-domain messaging. See http://www.w3.org/TR/webmessaging/ for more details on this specification.

Origins

In order to ensure security in cross-domain messaging applications, the notion of origins must be considered. An origin is the source of a message, and the recipient of any message must check the origin of that message to ensure it is coming from a known and trusted source.

When the NCS opens up a web control and loads the URL as provided by the device, it will append a parameter to the URL called “origin’, indicating the origin of the NCS. This is typically the URL of the NCS server, for example, a valid origin could be http://MYNCSHOST. The device, then, can read this origin out of the parameter list, and determine that it will only accept messages with an origin matching that supplied from the NCS.

When the device sends a message to the NCS, it needs to supply its own origin. The device’s origin can be determined by parsing the hostname, protocol, and port out of the device plugin’s URL. When the NCS receives a message from a device, it should validate the origin of the device to ensure that the message can be trusted.

Drag and Drop

Drag and drop of MOS objects from device plugins into the NCS is fully supported through standard HTML native drag and drop mechanisms in all major browsers (Internet Explorer, Firefox, Chrome, and Safari). The use of ondragstart and ondragend events attached to elements in the device plugin’s HTML DOM enables this support. An example is provided in the device implementation section of this document.

Examples

The examples provided in the following sections are proof-of-concept, and have been demonstrated as working across all major browser implementations. They use standard behavior as noted in the HTML specification for messaging. See http://www.w3.org/TR/webmessaging/ for more information.

Newsroom Implementation Example

The newsroom is responsible for hosting a container capable of running the web control. This could be an embedded web browser control (such as an Internet Explorer, Firefox, or Chrome embedded browser inside the NCS). The specific interface between the NCS and this browser is proprietary, and defined by the vendor. The browser itself must host an iframe or similar container that is reachable in the device plugin via the window.parent value, and the NCS must subscribe to 'message' window events supplied by the browser.

The example below illustrates this, with the assumption that the NCS itself is web-based for simplicity, and using an iframe control to host the device plugin.

What is the Origin?

To overcome cross-site scripting and cross-domain/same-origin policy limitations inside a web browser, we're building the message flow over a standard messaging service built into all major browsers, and defined as part of the HTML5 specification. We're effectively piggybacking the MOS messages across this secure messaging service. One of the requirements for this to work is the concept of origin, to ensure that you're only receiving and handling messages from windows that you expect. In this case, the NCS origin is the URL/host of the newsroom server. The origin should appear in the form is PROTOCOL://URL[:PORT], for example http://192.168.1.100, or a hostname is also valid, such as https://myncsserver. If a non-standard (not 80/443) port is used, it should be suffixed onto the origin.

When we open the URL for the device, we add the origin as a parameter at the end that URL, indicating to the device the origin of the NCS that it should expect to receive messages from. The NCS can determine the origin of the device by inspecting the URL to that device plugin, the origin being the protocol, hostname, and (optional) port elements of that URL.

To Send a Message to the Device

In this case, the NCS would invoke the mosMsgFromHost message, which locates the iframe container by ID, and invokes a "postMessage" method, passing in both the message, and the NCS origin URL. If there is a reply to be expected to this message, it will be handled as an inbound message in the mosMsgFromPlugin event.

To Receive and Reply to a Message from the Device

The mosMsgFromPlugin function is defined, and receives an event. This function is registered against the current window, so that whenever an event from the iframe is received, this function is invoked. The body of the message is located in event.data, as defined as part of the HTML specification. To reply to this message, simply invoke the postMessage method on the event.source, passing in the reply message, and the event origin.

Sample Code for Messaging

```
<html>
<head>

<script type=”text/javascript”>

function mosMsgFromPlugIn(event) {

	var message = event.data;

	// Handle the Message
	// To Reply, issue a postMessage on the event source.

	var reply = “SOME MESSAGE”;
	event.source.postMessage(reply, event.origin);

}

function mosMsgFromHost(message) {

	document.getElementById(‘plugin’).contentWindow.postMessage(message,
	‘http://NCS_ORIGIN’);

}

// Register the Event Handler – Cross Browser
if (window.addEventListener) {
	window.addEventListener('message', mosMsgFromPlugIn, false);
} else if (window.attachEvent) {
	window.attachEvent('message', mosMsgFromPlugIn, false);
}

</script>
</head>

<body>
<iframe id=”plugin” src=”DEVICE_PLUGIN_URL?origin=http://NCS_ORIGIN”/>
</body>

</html>
```

Device Implementation Example

The device implementation is similar to the newsroom implementation, but slightly simpler as it does not have to concern itself with embedding any control, it is entirely written in HTML and hosted on the device as an HTTP service.

The device is expected to provide the NCS with a public URL for which, when requested, serves up an HTML document. This document needs to register the appropriate listeners (as noted in the code example below) to subscribe to MOS messages received by the window.

Sample Code for Messaging

The NCS ORIGIN is provided to the device as a parameter in the invoked URL called “origin”. This should be validated whenever we receive a message to ensure it came from the expected window.

The DEVICE ORIGIN can be determined via the URL that was invoked to reach the device, as the following: PROTOCOL://HOSTNAME[:PORT]. For example, if the URL http://mydevice/somepage was invoked to reach the device’s plugin, then the device origin is http://mydevice. This can be determined by inspecting the window.location.host and window.location.protocol values.

```
<html>
<head>

<script type=”text/javascript”>

function mosMsgFromHost(event) {

        var message = event.data;

        // Check the Origin in event.origin to ensure it matches our expected NCS origin
        // parameter.

        // Handle the Message
        // To Reply, issue a postMessage on the event source.

        var reply = “SOME MESSAGE”;
        event.source.postMessage(reply, event.origin);

}

function mosMsgFromPlugIn(message) {

        window.parent.postMessage(message, ‘http://DEVICE_ORIGIN’);

}

// Register the Event Handler – Cross Browser
if (window.addEventListener) {
        window.addEventListener('message', mosMsgFromHost, false);
} else if (window.attachEvent) {
        window.attachEvent('message', mosMsgFromHost, false);
}

</script>
</head>

<body>
<!—My Plugin HTML→
</body>

</html>
```

Sample Code for Drag and Drop

MOS devices that wish to support object drag and drop into the NCS will be required to add the following code to the specific object they wish to make draggable. If IE10 is used as the web browsing container, a security setting must be enabled to allow for cross-origin/cross-domain drag and drop.

```
<html>
<head>
<script type=”text/javascript”>

function startObjectDrag(event) {

	event.dataTransfer.setData('Text', ‘MOS_OBJECT’);
	event.dataTransfer.effectAllowed = "copyMove";
	event.dataTransfer.dropEffect = "copy";

}

function endObjectDrag(event) {

}

</script>

<body>
<img src=”SOME_IMAGE” ondragstart=”startObjectDrag(event)” ondragend=”endObjectDrag(event)”/>
</body>

</html>
```