ECMAScript Pop-up Window

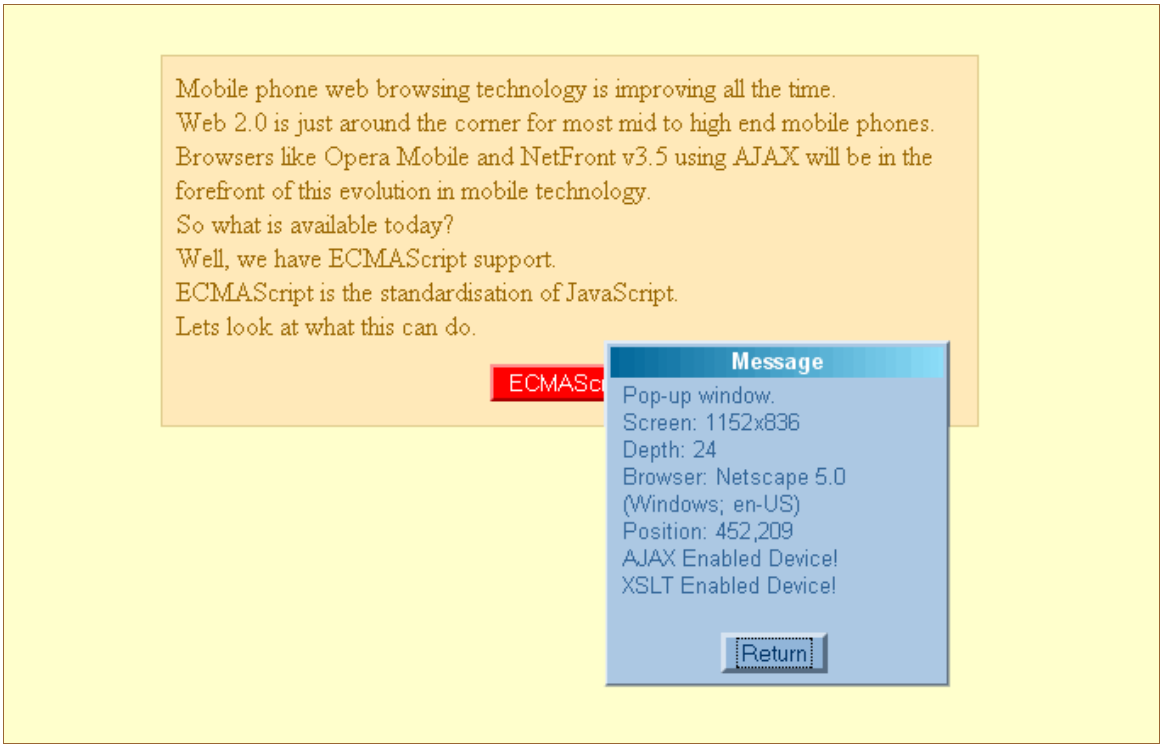
It is possible to create pop-up windows which can be displayed on mobile phones.

The following is a brief overview on what is required and the code to do it.

There are three files: popwin.html, popwin.js, popwin.css. Together these files produce a HTML page with a pop-up window as seen at http://vulpine.mobi/mobile/news/design/popup/

Below is an example of what a ECMAScript pop-up window will look like. The pale orange panel is the main HTML display. The blue pop-up window is displayed when the red ECMAScript button is pressed.

This ECMAScript pop-up window was designed to verify support for mobile phones using the NetFront v3.3 browser, so this example is quite simplistic in nature. The pop-up window should function in any browser that supports ECMA-262.



The following pages present the code required to create an ECMAScript pop-up window.

XHTML Code

The following code is constructed for mobile phones, however it can be access by any device that supports XHTML code.

File: popwin.html

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>ECMAScript Pop-up Window using NetFront v3.3</title>
<meta name="description"
 content="ECMAScript Pop-up Window using NetFront v3.3." />
<link href="/script/popwin.css" type="text/css" rel="stylesheet" />
<script src="/script/popwin.js" type="text/javascript"></script>
</head>
<body>
<div id="popwin">
<p>Mobile phone web browsing technology is improving all the time.
<br />Web 2.0 is just around the corner for most mid to high end mobile phones.
<br />Browsers like Opera Mobile and NetFront v3.5 using AJAX will be in the
forefront of this evolution in mobile technology.
<br />So what is available today?
<br />Well, we have ECMAScript support.
<br />ECMAScript is the standardisation of JavaScript.
<br />Lets look at what this can do.</p>
<button id="butt1" onclick="return popWin(event)" title="ECMAScript pop-up
window">ECMAScript</button>
<noscript>
<p class="pRed">
 Note - Your device does not have
 ECMAScript/JavaScript and this feature will NOT work.
</p>
</noscript>
</div>
</body>
```

Cascading Style Sheet code

The CSS file is used to provide formatting and presentation for the ECMAScript pop-up window. It is not recommended to use in-line style statements in ECMAScript where it can be avoided. The file bluehighlight.gif is a 210x1px image with a colour gradient for the pop-up windows title bar. This file is located in the /img folder.

This style sheet file is placed in the /script folder.

File: popwin.css

```
div#popwin{
 margin:1em auto 0 auto;
 width:30em;
 border:1px solid #caa862;
 color:#960;
 background-color:#ffe9b9;
 text-align:center;
}
button#butt1{
 text-align:center;
 color:#fff;
 background-color:#f00;
 border:2px outset #f33;
 margin:1em;
}
p{
 margin:0.5em 0.5em 0 0.5em;
 text-align:left;
}
p.pRed{
 color:#900;
 background-color:transparent;
 font-weight:bold;
}
p.js{
 margin:0;
 padding-left:0.5em;
 color:#369;
 background-color:transparent;
 text-align:left;
}
div#pop1{
 border:2px outset #acc8e3;
 color:#fff;
 background-color:transparent;
 font:normal 10pt arial;
 text-align:center;
}
div#tp1{
 font-weight:bold;
 background-image:url(/img/bluehighlight.gif);
 background-repeat:repeat-y;
}
div#msg{
 text-align:left;
 margin:0;
}
button#butt2{
 color:#036;
 border:3px outset #9cf;
 background-color:#91b5d9;
}
```

ECMAScript Code

The first thing we need to do in any ECMAScript and JavaScript for that matter, is identify the protagonists. In this script we assume support for ECMA-262 and DOM standards, however Microsoft has it's own way of doing things which also needs to be identified.

In the past to identify the Microsoft IE browser, most people just coded `if(document.all)`.

However Opera, due to backward compatibility for Microsoft IE, also identifies this method as valid.

So, as the focus of the exercise is to create a pop-up window for mobile phones we will only target standards based code and Microsoft IE.

This can be done by testing the use of the event handlers. Microsoft does not support DOM event handlers and thus allows us to use the standards and leaves Microsoft free to do the other thing.

If and when Microsoft supports DOM event handlers, they are more than likely to support its full features. This means browser sniffing is not required and as I read somewhere, sniffing anything is not good for you.

Both XMLHttpRequest (AJAX) and XSLTProcessor (XSLT) are checked for support.

Images displayed within the pop-up window are preloaded and the characteristics of the browser are retrieved.

When checking the Microsoft IE browser we only return the browser version, we do not return the full User-Agent that is provided by `navigator.appVersion`.

This script file is placed in the /script folder.

File: popwin.js

```
w3c=(window.addEventListener)?true:false;
var xmlHttp=cXmlHttp();
var xmlXslt=w3cXslt();
if(document.images){
  var img=new Object();
  img[1]=new Image();
  img[1].src ="/img/bluehighlight.gif";
};
var saw=(screen.availWidth)?screen.availWidth:'0';
var sah=(screen.availHeight)?screen.availHeight:'0';
var scd=screen.colorDepth;
var spd=screen.pixelDepth;
var bws=navigator.appName;
var bwv=navigator.appVersion;
if(!w3c){
  var myRegex=new RegExp("\\d\\.\\d{0,2}");
  bwv=myRegex.exec(bwv);
};
```

At this point we check some basic screen sizes.

If the screen size is greater than 320px we create a static 200x200px pop-up window.

If the screen size is between 160px and 320px wide, we place padding round the pop-up window and use the dimensions to create the window.

If the screen size is below 160px, we create minimal padding around the pop-up window and use a small fixed size of 122px which on very small screens may mean some sideways scrolling.

```
var wd=122,hd=200,padw=15,padh=15;
if(saw&&saw>=160&&saw<=320){
 wd=saw-(padw*2);
 hd=sah;
}
else if(saw&&saw>320){wd=200;}
else{
 padw=2;
 if(saw){wd=saw-4;}
};
```

An Object is created to listen for key clicks on the pop-up window. The key clicks are bound to the return button on the pop-up window which has been given focus.

When a key click is detected the close function is called. The pop-up window is hidden and focus is returned to the button on the original HTML page. It is possible to de-construct the pop-up window when returning to the original page, however there is the overhead of building it again if the pop-up window is revisited.

```
closeButton=new Object();
closeButton.ctrl=function(obj)
{
 if(!w3c){obj.onclick=closeButton.close;}
 else{obj.addEventListener("click",closeButton.close,false);}
};
closeButton.close=function(e)
{
 this.exit.oB.style.display="none";
 this.exit.oB.visible="hidden";
 document.getElementById("butt1").focus();
};
```

Each text node is a separate object, so this function is repeatedly called for each text node displayed. To provide the required formatting we refer to the CSS class. As the formatting is a constant we should not declare it as an in-line style within the ECMAScript. By doing so you are breaking the separate of code and presentation, creating a maintenance nightmare for yourself in the future.

```
para=function(txt){
  var p=document.createElement("p");
  p.className="js";
  p.appendChild(txt);
  return p;
};
```

The DOMDv method is repeatedly called to create the HTML DIV regions to contain the content of the pop-up window. Here we do not separate the ECMAScript and CSS as the dimension of each DIV container will be dynamic. It is best to only use position absolute as many mobile browsers do not support position relative and thus, will only cause problems.

```
function DOMDv(x,y,w,h,col){
  var oDv=document.createElement("DIV");
  oDv.style.position="absolute";
  oDv.style.left=x+"px";
  oDv.style.top=y+"px";
  oDv.style.width=w+"px";
  oDv.style.height=h+"px";
  oDv.style.visibility="visible";
  oDv.style.backgroundColor=col;
  return oDv;
}
```

As the DOM standard and Microsoft use different methods to create an AJAX object we have to test both. The W3C DOM method is relatively straight forward and only requires a one line check, however the Microsoft method adds several lines of code as it has to check for different versions which may be used in any given IE browser.

```
function cXmlHttp(){
  var xmlHttp;
  try{xmlHttp=new XMLHttpRequest();}
  catch(e){
    var XmlHttpVersions=new Array("MSXML2.XMLHTTP.6.0"
                                 ,"MSXML2.XMLHTTP.5.0"
                                 ,"MSXML2.XMLHTTP.4.0"
                                 ,"MSXML2.XMLHTTP.3.0"
                                 ,"MSXML2.XMLHTTP"
                                 ,"Microsoft.XMLHTTP");
    for(var i=0;i<XmlHttpVersions.length&&!xmlHttp; i++){
      try{xmlHttp=new ActiveXObject(XmlHttpVersions[i]);}
      catch(e){}
    }
  }
  if(!xmlHttp)return false;
  else return xmlHttp;
}
```

This script uses only the DOM method to check the browser for XSLT support. IE browsers are not checked in this script as like XMLHttpRequest this will bloat the code unnecessarily and this is mobile phone proof on concept only.

```
function w3cXslt(){
  var xsltHttp;
  try{xsltHttp=new XSLTProcessor();}
  catch(e){}
  if(!xsltHttp)return false;
  else return xsltHttp;
}
```

The main function and entry point from the button click on the main HTML page is located here. The event object is passed to the pop-up window which contains positional details of the main browser window.

```
function popWin(oEvt){
```

Next we check to see if the pop-up window has been previously visited and if so we redisplay the pop-up window. This returns the focus back to the pop-up window button and waits on any onclick events on this button.

```
if(document.getElementById('pop1')){
    this.oB.style.display="block";
    this.oB.visible="visible";
    document.getElementById("butt2").focus();
```

If this is the first visit to the pop-up window be then start the building of the pop-up window.

First we check if the current key click position has been sent and map it to the X and Y coordinates of the screen. A lot of mobile phones do not return an X coordinate due to the logic of mobile screen layout.

Repeated calls are made to the DOMDv method to create containing DIV blocks.
Each unique DIV is given a unique id name which is used my CSS to format the presentation. The use of the style attribute within an element should be restrict to those styles which are dynamic. Where the presentation for an element is static the CSS file should always be referenced.

During the construction of the pop-up window the return button object is passed to the event handler which will then wait for any onclick events on the specified button.

```
}else{
    if(!document.getElementById||!document.createElement){
      alert("DOM methods not supported on this device.");
      return false;
    }

    var posY=(!w3c)?oEvt.clientY:oEvt.pageY;
    var posX=(!w3c)?oEvt.clientX:oEvt.pageX;
    if (posX&&wd<200)posX=0;

    this.oB=new DOMDv(posX+padw,posY-25,wd,hd,"#acc8e3");
    this.oB.id="pop1";

    this.oTT=new DOMDv(2,2,wd-4,18,"#369");
    this.oTT.appendChild(document.createTextNode("Message"));
    this.oTT.id="tp1";

    this.oCA=new DOMDv(2,22,wd-4,hd-52,"transparent");
    this.oCA.id="msg";

    this.oTB=document.createElement("button");
    this.oTB.id="butt2";
    this.oTB.appendChild(document.createTextNode("Return"));
    this.oTB.exit=this;
    closeButton.ctrl(this.oTB);
    this.oBA=new DOMDv(((wd-60)-6)/2,hd-30,60,20,"transparent");
    this.oBA.appendChild(this.oTB);

    this.oP1=para(document.createTextNode("Pop-up window."));
    this.oP2=para(document.createTextNode("Screen: "+saw+"x"+sah));
    this.oP3=para(document.createTextNode("Depth: "+scd));
    this.oP4=para(document.createTextNode("Browser: "+bws+" "+bwv));
    this.oP5=para(document.createTextNode("Position: "+posX+","+posY));

    if(xmlHttp){var aTxt="AJAX Enabled Device!"}
    else{var aTxt="NO AJAX Support"};
    var mt6=document.createTextNode(aTxt);
    this.oP6=para(mt6);

    if(xmlXslt){var xTxt="XSLT Enabled Device!"}
    else if (!w3c) {var xTxt="IE XSLT not checked."}
    else{var xTxt="NO XSLT Support"};
    var mt7=document.createTextNode(xTxt);
    this.oP7=para(mt7);
```

As each element is built up it is appended to it's parent. In this way a final body object is created which will be displayed on the screen.

Finally, once the pop-up window is displayed, focus is given to the return button on the pop-up window.

```
    this.oCA.appendChild(this.oP1);
    this.oCA.appendChild(this.oP2);
    this.oCA.appendChild(this.oP3);
    this.oCA.appendChild(this.oP4);
    this.oCA.appendChild(this.oP5);
    this.oCA.appendChild(this.oP6);
    this.oCA.appendChild(this.oP7);

    this.oB.appendChild(this.oTT);
    this.oB.appendChild(this.oCA);
    this.oB.appendChild(this.oBA);

    document.body.appendChild(this.oB);
    document.getElementById("butt2").focus();
  }
};
```

By putting all this together we have created the popwin.js file.

This has been a very basic introduction to the construction of a pop-up window using ECMAScript. There is no real world functionality provided by this pop-up window as it stands, however with a little bit of effort this could be used as the basis of any ECMA-262 script or extended for use with AJAX and XSLT.

There is no restriction on the use of this code, but you might like to say you found it at Vulpine Mobile (http://vulpine.mobi/).