```
/* See license.txt for terms of usage */

define([
    "firebug/lib/object",
    "firebug/firebug",
    "firebug/chrome/firefox",
    "firebug/chrome/reps",
    "firebug/lib/locale",
    "firebug/lib/events",
    "firebug/lib/wrapper",
    "firebug/lib/array",
    "firebug/lib/css",
    "firebug/lib/dom",
    "firebug/lib/xml",
    "firebug/chrome/window",
    "firebug/lib/system",
    "firebug/html/highlighterCache"
],
function(Obj, Firebug, Firefox, FirebugReps, Locale, Events, Wrapper, Arr, Css, Dom, Xml,
    Win, System, HighlighterCache) {

//
     ******************************************************************************************
     ** //
// Constants

const inspectDelay = 200;
const highlightCssUrl = "chrome://firebug/content/html/highlighter.css";
const ident = HighlighterCache.ident;
const Cu = Components.utils;

//
     ******************************************************************************************
     ** //
// Globals

var boxModelHighlighter = null;
var frameHighlighter = null;

//
     ******************************************************************************************
     ** //

/**
 * @module Implements Firebug Inspector logic.
 */
Firebug.Inspector = Obj.extend(Firebug.Module,
{
    dispatchName: "inspector",
    inspecting: false,
    inspectingPanel: null,

    /**
     * Main highlighter method. Can be used to highlight elements using the box model,
     * frame or image map highlighters. Can highlight single or multiple elements.
     *
     * Examples:
     * Firebug.Inspector.highlightObject([window.content.document.getElementById("gbar"),
     *     window.content.document.getElementById("logo")],
     *     window.content, "frame", null,
     *     ["#ff0000",{background:"#0000ff", border:"#ff0000"}])
     * or
     * Firebug.Inspector.highlightObject([window.content.document.getElementById("gbar"),
     *     window.content.document.getElementById("logo")], window.content, "boxModel", null,
     *     [{content: "#ff0000", padding: "#eeeeee", border: "#00ff00", margin: "#0000ff"},
     *         {content: "#00ff00", padding: "#eeeeee", border: "#00ff00", margin: "#0000ff"}])
     *
     * @param {Array} elementArr Elements to highlight
     * @param {Window} context Context of the elements to be highlighted
     * @param {String} [highlightType] Either "frame" or "boxModel". Default is configurable.
     * @param {String} [boxFrame] Displays the line guides for the box model layout view.
     *      Valid values are: "content", "padding", "border" or "margin"
     * @param {String | Array} [colorObj] Any valid html color e.g. red, #f00, #ff0000, etc.,
     *      a valid color object or any valid highlighter color array.
     */
    highlightObject: function(elementArr, context, highlightType, boxFrame, colorObj)
    {
        var i, elt, elementLen, oldContext, usingColorArray;
        var highlighter = highlightType ? getHighlighter(highlightType) : this.
            defaultHighlighter;

        if (!elementArr || !Arr.isArrayLike(elementArr))
        {
            // Not everything that comes through here is wrapped - fix that.
            elementArr = Wrapper.wrapObject(elementArr);

            // highlight a single element
            if (!elementArr || !Dom.isElement(elementArr) ||
                (typeof elementArr === "object" && !Xml.isVisible(elementArr)))
            {
                if (elementArr && Dom.isRange(elementArr))
                    elementArr = elementArr;
                else if (elementArr && elementArr.nodeType == Node.TEXT_NODE)
                    elementArr = elementArr.parentNode;
                else
                    elementArr = null;
            }

            if (elementArr && context && context.highlightTimeout)
            {
                context.clearTimeout(context.highlightTimeout);
                delete context.highlightTimeout;
            }

            oldContext = this.highlightedContext;
            if (oldContext && oldContext.window)
                this.clearAllHighlights();

            // Stop multi element highlighting
            if (!elementArr)
                this.repaint.element = null;

            this.highlighter = highlighter;
            this.highlightedContext = context;

            if (elementArr)
            {
                if (elementArr.nodeName && !isVisibleElement(elementArr))
                    highlighter.unhighlight(context);
                else if (context && context.window && context.window.document)
                    highlighter.highlight(context, elementArr, boxFrame, colorObj, false);
            }
            else if (oldContext)
            {
                oldContext.highlightTimeout = oldContext.setTimeout(function()
                {
                    if (FBTrace.DBG_INSPECT)
                        FBTrace.sysout("Removing inspector highlighter due to setTimeout loop");

                    if (!oldContext.highlightTimeout)
                        return;

                    delete oldContext.highlightTimeout;

                    if (oldContext.window && oldContext.window.document)
                    {
                        highlighter.unhighlight(oldContext);
                    }
                }, inspectDelay);
            }
        }
        else
        {
            // Highlight multiple elements
            if (context && context.highlightTimeout)
            {
                context.clearTimeout(context.highlightTimeout);
                delete context.highlightTimeout;
            }

            this.clearAllHighlights();
            usingColorArray = Arr.isArray(colorObj);

            if (context && context.window && context.window.document)
            {
                for (i=0, elementLen=elementArr.length; i<elementLen; i++)
                {
                    // Like above, wrap things.
                    elt = Wrapper.wrapObject(elementArr[i]);

                    if (elt && elt instanceof HTMLElement)
                    {
                        if (elt.nodeType == Node.TEXT_NODE)
                            elt = elt.parentNode;

                        var obj = usingColorArray ? colorObj[i] : colorObj;
                        highlighter.highlight(context, elt, null, obj, true);
                    }
                }
            }

            storeHighlighterParams(null, context, elementArr, null, colorObj, highlightType,
                   true);
        }
    },

    /**
     * Clear all highlighted areas on a page.
     */
    clearAllHighlights: function()
    {
        HighlighterCache.clear();
    },

    /**
     * Toggle inspecting on / off
     * @param {Window} [context] The window to begin inspecting in, necessary to toggle
           inspecting on.
     */
    toggleInspecting: function(context)
    {
        if (this.inspecting)
            this.stopInspecting(true);
        else
            this.startInspecting(context);
    },

    /**
     * Check if the new panel has the inspectable property set. If so set it as the new
           inspectingPanel.
     */
    onPanelChanged: function()
    {
        if (this.inspecting)
        {
            var panelBar1 = Firebug.chrome.$("fbPanelBar1");
            var panel = panelBar1.selectedPanel;

            if (panel && panel.inspectable)
            {
                this.inspectNode(null);
                this.inspectingPanel = panel;
            }
        }
    },

    /**
     * Turn inspecting on.
     * @param {Window} context The main browser window
     */
    startInspecting: function(context)
    {
        if (this.inspecting || !context || !context.loaded)
            return;

        this.clearAllHighlights();

        this.inspecting = true;
        this.inspectingContext = context;

        Firebug.chrome.setGlobalAttribute("cmd_firebug_toggleInspecting", "checked", "true");
        this.attachInspectListeners(context);

        var inspectingPanelName = this._resolveInspectingPanelName(context);
```

```
        this.inspectingPanel = Firebug.chrome.switchToPanel(context, inspectingPanelName);

        if (Firebug.isDetached())
            context.window.focus();
        else if (Firebug.isMinimized())
            Firebug.showBar(true);

        this.inspectingPanel.panelNode.focus();
        this.inspectingPanel.startInspecting();

        Events.dispatch(this.fbListeners, "onStartInspecting", [context]);

        if (context.stopped)
            Firebug.Debugger.thaw(context);

        var hoverNodes = context.window.document.querySelectorAll(":hover");

        if (hoverNodes.length != 0)
            this.inspectNode(hoverNodes[hoverNodes.length-1]);
    },

    /**
     * Highlight a node using the frame highlighter. Can only be used after inspecting has
           already started.
     * @param {Element} node The element to inspect
     */
    inspectNode: function(node)
    {
        if (node && node.nodeType != Node.ELEMENT_NODE)
            node = node.parentNode;

        if (node && Firebug.shouldIgnore(node) && !node.fbProxyFor)
            return;

        var context = this.inspectingContext;

        if (this.inspectTimeout)
        {
            context.clearTimeout(this.inspectTimeout);
            delete this.inspectTimeout;
        }

        if (node && node.fbProxyFor)
            node = node.fbProxyFor;

        var inspectingPanel = this.inspectingPanel;

        // Some panels may want to only allow inspection of panel-supported objects
        node = inspectingPanel ? inspectingPanel.getInspectNode(node) : node;

        var highlightColor = inspectingPanel ? inspectingPanel.inspectHighlightColor : "";
        this.highlightObject(node, context, "frame", undefined, highlightColor);

        this.inspectingNode = node;

        if (node)
        {
            var _this = this;

            this.inspectTimeout = context.setTimeout(function()
            {
                var selection = inspectingPanel ? inspectingPanel.inspectNode(node) : null;
                Events.dispatch(_this.fbListeners, "onInspectNode", [context, node]);
                if (selection)
                    inspectingPanel.select(node);
            }, inspectDelay);
        }
    },

    /**
     * Stop inspecting and clear all highlights.
     * @param {Boolean} canceled Indicates whether inspect was canceled (usually via the escape
           key)
     * @param {Boolean} [waitForClick] Indicates whether the next click will still forward you
     *      to the clicked element in the HTML panel.
     */
    stopInspecting: function(canceled, waitForClick)
    {
        if (!this.inspecting)
            return;

        var context = this.inspectingContext;

        if (context.stopped)
            Firebug.Debugger.freeze(context);

        if (this.inspectTimeout)
        {
            context.clearTimeout(this.inspectTimeout);
            delete this.inspectTimeout;
        }

        this.detachInspectListeners(context);
        if (!waitForClick)
            this.detachClickInspectListeners(context.window);

        Firebug.chrome.setGlobalAttribute("cmd_firebug_toggleInspecting", "checked", "false");

        this.inspecting = false;

        if (this.inspectingPanel)
        {
            Firebug.chrome.unswitchToPanel(context, this.inspectingPanel.name, canceled);
            this.inspectingPanel.stopInspecting(this.inspectingNode, canceled);
        }
        else
        {
            FBTrace.sysout("inspector.stopInspecting; ERROR? inspectingPanel is NULL");
        }

        Events.dispatch(this.fbListeners, "onStopInspecting", [context, this.inspectingNode,
              canceled]);

        this.inspectNode(null);

        // Make sure there are no (indirect) references to the page document.
        this.inspectingPanel = null;
        this.inspectingContext = null;

        if (Firebug.isDetached())
            window.focus();
    },

    /**
     * Get the name of the inspectable panel.
     * @param {Window} context Context of the panel
     */
    _resolveInspectingPanelName: function(context)
    {
        var requestingPanel = context && context.getPanel(context.panelName);

        return (requestingPanel && requestingPanel.inspectable) ? requestingPanel.name : "html";
    },

    /**
     * Inspect from context menu.
     * @param {Element} elt The element to inspect
     */
    inspectFromContextMenu: function(elt)
    {
        var panel;
        var inspectingPanelName = "html";

        Firebug.toggleBar(true, inspectingPanelName);
        Firebug.chrome.select(elt, inspectingPanelName);
        panel = Firebug.chrome.selectPanel(inspectingPanelName);
        panel.panelNode.focus();
    },

    /**
     * Navigate up and down through the DOM and highlight the result. This method is used by
     * the key handlers for the up and down arrow keys.
     *
     * @param {String} dir Direction to navigate the Dom, either "up" or "down"
     */
    inspectNodeBy: function(dir)
    {
        var target;
        var node = this.inspectingNode;

        if (dir == "up")
        {
            target = Firebug.chrome.getNextObject();
        }
        else if (dir == "down")
        {
            target = Firebug.chrome.getNextObject(true);
            if (node && !target)
            {
                target = node.contentDocument ?
                    node.contentDocument.documentElement : Dom.getNextElement(node.firstChild);
            }
        }

        if (target && Dom.isElement(target))
            this.inspectNode(target);
        else
            System.beep();
    },

    /**
     * Repaint the highlighter. Called from the window scroll and resize handlers.
     */
    repaint: function()
    {
        var rp = this.repaint;
        var highlighter = rp.highlighter;
        var context = rp.context;
        var element = rp.element;
        var boxFrame = rp.boxFrame;
        var colorObj = rp.colorObj;
        var highlightType = rp.highlightType;
        var isMulti = rp.isMulti;

        if (!context || (!highlighter && !isMulti))
            return;

        if (isMulti && element)
        {
            this.highlightObject(element, context, highlightType, null, colorObj);
        }
        else if (!isMulti)
        {
            var highlighterNode = HighlighterCache.get(highlighter.ident);

            if (highlighterNode && highlighter.ident === ident.boxModel)
                highlighterNode = highlighterNode.offset;

            if (highlighterNode && highlighterNode.parentNode)
            {
                this.clearAllHighlights();
                highlighter.highlight(context, element, boxFrame, colorObj, isMulti);
            }
        }
    },

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          * //

    /**
     * Attach the scroll and resize handlers to elt's window. Called from every highlight call.
     * @param {Element} elt Passed in order to reliably obtain context
     */
    attachRepaintInspectListeners: function(context, elt)
    {
        if (!elt || !elt.ownerDocument || !elt.ownerDocument.defaultView)
            return;

        var win = elt.ownerDocument.defaultView;

        if (FBTrace.DBG_INSPECT)
            FBTrace.sysout("inspector.attachRepaintInspectListeners to " + win.location.href,
                   elt);
        // there is no way to check if the listeners have already been added and we should
        // avoid adding properties to the users page.
        // Adding them again will do no harm so lets just do that.

        // xxxHonza: I think that adding them twice could actually do harm,
        // so make sure they are removed before.
        context.removeEventListener(win.document, "resize", this.onInspectingResizeWindow, true)
              ;
        context.removeEventListener(win.document, "scroll", this.onInspectingScroll, true);

        // Register again.
        context.addEventListener(win.document, "resize", this.onInspectingResizeWindow, true);
        context.addEventListener(win.document, "scroll", this.onInspectingScroll, true);
    },

    /**
     * Attach key and mouse events to windows recursively.
     * @param {Window} context Context of the main browser window
     */
    attachInspectListeners: function(context)
    {
        var win = context.window;
        if (!win || !win.document)
            return;

        if (FBTrace.DBG_INSPECT)
            FBTrace.sysout("inspector.attachInspectListeners to all subWindows of " + win.
                   location);

        var chrome = Firebug.chrome;

        this.keyListeners =
        [
            chrome.keyCodeListen("RETURN", null, Obj.bindFixed(this.stopInspecting, this)),
            chrome.keyCodeListen("ESCAPE", null, Obj.bindFixed(this.stopInspecting, this, true))
                   ,
            chrome.keyCodeListen("UP", Events.isControl, Obj.bindFixed(this.inspectNodeBy, this,
                "up"), true),
            chrome.keyCodeListen("DOWN", Events.isControl, Obj.bindFixed(this.inspectNodeBy,
                   this,
                "down"), true),
        ];

        Win.iterateWindows(win, Obj.bind(function(subWin)
        {
            if (FBTrace.DBG_INSPECT)
                FBTrace.sysout("inspector.attacheInspectListeners to " + subWin.location +
                    " subWindow of " + win.location);

            Events.addEventListener(subWin.document, "mouseover", this.onInspectingMouseOver,
                true);
            Events.addEventListener(subWin.document, "mousedown", this.onInspectingMouseDown,
                true);
            Events.addEventListener(subWin.document, "mouseup", this.onInspectingMouseUp, true);
            Events.addEventListener(subWin.document, "click", this.onInspectingClick, true);
            Events.addEventListener(subWin.document, "keypress", this.onInspectingKeyPress, true
                   );
        }, this));
    },

    /**
     * Remove all event listeners except click listener from windows recursively.
     * @param {Window} context Context of the main browser window
     */
    detachInspectListeners: function(context)
    {
        var i, keyListenersLen;
        var win = context.window;

        if (!win || !win.document)
            return;

        var chrome = Firebug.chrome;

        if (this.keyListeners)  // XXXjjb for some reason this is null sometimes.
        {
            keyListenersLen = this.keyListeners.length;
            for (i = 0; i < keyListenersLen; ++i)
```

```
                chrome.keyIgnore(this.keyListeners[i]);
            delete this.keyListeners;
        }

        Win.iterateWindows(win, Obj.bind(function(subWin)
        {
            Events.removeEventListener(subWin.document, "mouseover", this.onInspectingMouseOver,
                true);
            Events.removeEventListener(subWin.document, "mousedown", this.onInspectingMouseDown,
                true);
            Events.removeEventListener(subWin.document, "mouseup", this.onInspectingMouseUp,
                  true);
            Events.removeEventListener(subWin.document, "keypress", this.onInspectingKeyPress,
                true);
        }, this));
    },

    /**
     * Remove the click listener independently from detachInspectListeners because if we remove
     * it after mousedown, we won't be able to cancel clicked links.
     *
     * @param {Window} context Context of the main browser window
     */
    detachClickInspectListeners: function(context)
    {
        Win.iterateWindows(context, Obj.bind(function(subWin)
        {
            Events.removeEventListener(subWin.document, "click", this.onInspectingClick, true);
        }, this));
    },

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         * //

    /**
     * Repaint last highlight in the correct position on window resize.
     * @param {Event} event Passed for tracing
     */
    onInspectingResizeWindow: function(event)
    {
        if (FBTrace.DBG_INSPECT)
           FBTrace.sysout("onInspectingResizeWindow event", event);

        this.repaint();
    },

    /**
     * Repaint last highlight in the correct position on scroll.
     * @param {Event} event Passed for tracing
     */
    onInspectingScroll: function(event)
    {
        if (FBTrace.DBG_INSPECT)
           FBTrace.sysout("onInspectingScroll event", event);

        this.repaint();
    },

    /**
     * Call inspectNode(event.target) highlighting the element that was moused over.
     * @param {Event} event Passed for tracing and to identify the target of inspection
     */
    onInspectingMouseOver: function(event)
    {
        if (FBTrace.DBG_INSPECT)
           FBTrace.sysout("onInspectingMouseOver event", event);

        this.inspectNode(event.target);
    },

    /**
     * Trap mousedown events to prevent clicking a document from triggering a document's
     * mousedown event when inspecting.
     *
     * @param {Event} event Used for tracing and canceling the event
     */
    onInspectingMouseDown: function(event)
    {
        if (FBTrace.DBG_INSPECT)
        {
            FBTrace.sysout("onInspectingMouseDown event", {originalTarget: event.originalTarget,
                tmpRealOriginalTarget:event.tmpRealOriginalTarget, event:event});
        }

        // Allow to scroll the document while inspecting
        if (event.originalTarget && event.originalTarget.tagName == "xul:thumb")
            return;

        Events.cancelEvent(event);
    },

    /**
     * Trap mouseup events to prevent clicking a document from triggering a document's mouseup
     * event when inspecting.
     *
     * @param {Event} event Used for tracing and canceling the event
     */
    onInspectingMouseUp: function(event)
    {
        if (FBTrace.DBG_INSPECT)
        {
            FBTrace.sysout("onInspectingMouseUp event", {originalTarget: event.originalTarget,
                tmpRealOriginalTarget:event.tmpRealOriginalTarget,event:event});
        }

        // Allow to release scrollbar while inspecting
        if (event.originalTarget && event.originalTarget.tagName == "xul:thumb")
            return;

        this.stopInspecting(false, true);

        Events.cancelEvent(event);
    },

    /**
     * Trap click events to prevent clicking a document from triggering a document's click event
     * when inspecting and removes the click inspect listener.
     *
     * @param {Event} event Used for tracing and canceling the event
     */
    onInspectingClick: function(event)
    {
        if (FBTrace.DBG_INSPECT)
            FBTrace.sysout("onInspectingClick event", event);

        var win = event.currentTarget.defaultView;
        if (win)
        {
            win = Win.getRootWindow(win);
            this.detachClickInspectListeners(win);
        }

        Events.cancelEvent(event);
    },

    /**
     * Trap keypress events to allow manipulation of the hovered elements
     *
     * @param {Event} event Used for canceling the event
     */
    onInspectingKeyPress: function(event)
    {
        if (event.keyCode == KeyEvent.DOM_VK_DELETE)
        {
            Events.dispatch(this.fbListeners, "onBeginFirebugChange", [this.inspectingNode, this
                   ]);
            this.inspectingNode.parentNode.removeChild(this.inspectingNode);
            Events.dispatch(this.fbListeners, "onEndFirebugChange", [this.inspectingNode, this])
                   ;
            Events.cancelEvent(event);
        }
    },

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         * //
    // extends Module
```

```
/**
 * Initialize the inspector
 */
initialize: function()
{
    Firebug.Module.initialize.apply(this, arguments);

    this.onInspectingResizeWindow = Obj.bind(this.onInspectingResizeWindow, this);
    this.onInspectingScroll = Obj.bind(this.onInspectingScroll, this);
    this.onInspectingMouseOver = Obj.bind(this.onInspectingMouseOver, this);
    this.onInspectingMouseDown = Obj.bind(this.onInspectingMouseDown, this);
    this.onInspectingMouseUp = Obj.bind(this.onInspectingMouseUp, this);
    this.onInspectingClick = Obj.bind(this.onInspectingClick, this);
    this.onInspectingKeyPress = Obj.bind(this.onInspectingKeyPress, this);
    this.onPanelChanged = Obj.bind(this.onPanelChanged, this);

    this.updateOption("shadeBoxModel", Firebug.shadeBoxModel);
    this.updateOption("showQuickInfoBox", Firebug.showQuickInfoBox);

    var panelBar1 = Firebug.chrome.$("fbPanelBar1");
    Events.addEventListener(panelBar1, "selectPanel", this.onPanelChanged, false);

    if (FBTrace.DBG_INSPECT)
        FBTrace.sysout("inspector.initialize;");
},

shutdown: function()
{
    Firebug.Module.shutdown.apply(this, arguments);

    var panelBar1 = Firebug.chrome.$("fbPanelBar1");
    Events.removeEventListener(panelBar1, "selectPanel", this.onPanelChanged, false);
},

/**
 * Stop inspecting and delete timers.
 * @param {Window} context Context of the main window
 */
destroyContext: function(context)
{
    if (context.highlightTimeout)
    {
        context.clearTimeout(context.highlightTimeout);
        delete context.highlightTimeout;
    }

    if (this.inspecting)
        this.stopInspecting(true);
},

/**
 */
unwatchWindow: function(context, win)
{
    try
    {
        this.hideQuickInfoBox();
    }
    catch (ex)
    {
        // Get unfortunate errors here sometimes, so let's just ignore them since the
        // window is going away anyhow
    }
},

/**
 * Called when a FF tab is created or activated (user changes FF tab). We stop inspecting
 * in this situation.
 *
 * @param {xul:browser} [browser] Browser
 * @param {Window} [context] The main browser window
 */
showContext: function(browser, context)
{
    if (this.inspecting)
        this.stopInspecting(true);
},
/**
 * Called when a panel is shown.
 * @param {xul:browser} [browser] Browser
 * @param {Panel} [panel] Panel
 */
showPanel: function(browser, panel)
{
    // Don't disable the cmd_toggleInspecting command. The related shortcut <key> must
    // be available even if Firebug is not activated for the site. See 4452
    // The panel can be null (if disabled) so use the global context.
    // var context = Firebug.currentContext;
    // var disabled = (context && context.loaded) ? false : true;
    // Firebug.chrome.setGlobalAttribute("cmd_firebug_toggleInspecting", "disabled",
           disabled);
},

/**
 * Called after a context's page gets DOMContentLoaded. We enable inspection here.
 * @param {Window} [context] Context of the main window
 */
loadedContext: function(context)
{
    // See the comment in showPanel.
    // Firebug.chrome.setGlobalAttribute("cmd_firebug_toggleInspecting", "disabled",
           "false");
},

/**
 * Update the shadeBoxModel or showQuickInfoBox options
 * @param {String} name Either "shadeBoxModel" or "showQuickInfoBox"
 * @param {Boolean} value Enable or Disable the option
 */
updateOption: function(name, value)
{
    if (name == "shadeBoxModel")
    {
        this.highlightObject(null);
        this.defaultHighlighter = value ? getHighlighter("boxModel") :
               getHighlighter("frame");
    }
    else if(name == "showQuickInfoBox")
    {
        quickInfoBox.boxEnabled = value;
    }
},

/**
 * Gets stylesheet by Url.
 * @param {Window} context the main browser window
 * @param {String} url URL of the stylesheet
 */
getObjectByURL: function(context, url)
{
    var styleSheet = Css.getStyleSheetByHref(url, context);
    if (styleSheet)
        return styleSheet;
},

/**
 * Toggle the quick info box.
 */
toggleQuickInfoBox: function()
{
    var qiBox = Firebug.chrome.$("fbQuickInfoPanel");

    if (qiBox.state == "open")
        quickInfoBox.hide();

    quickInfoBox.boxEnabled = !quickInfoBox.boxEnabled;

    Firebug.Options.set("showQuickInfoBox", quickInfoBox.boxEnabled);
},

/**
 * Hide the quick info box.
 */
hideQuickInfoBox: function()
```

```
    {
        var qiBox = Firebug.chrome.$("fbQuickInfoPanel");

        if (qiBox.state==="open")
            quickInfoBox.hide();

        this.inspectNode(null);
    },

    /**
     * Pass all quick info box events to quickInfoBox.handleEvent() for handling.
     * @param {Event} event Event to handle
     */
    quickInfoBoxHandler: function(event)
    {
        quickInfoBox.handleEvent(event);
    }

});

//
     ***********************************************************************************************
     ** //
// Local Helpers

function getHighlighter(type)
{
    switch (type)
    {
        case "boxModel":
            if (!boxModelHighlighter)
                boxModelHighlighter = new BoxModelHighlighter();

            return boxModelHighlighter;

        case "frame":
            if (!frameHighlighter)
                frameHighlighter = new Firebug.Inspector.FrameHighlighter();

            return frameHighlighter;
    }
}

function pad(element, t, r, b, l)
{
    var css = "padding:" + Math.abs(t) + "px " + Math.abs(r) + "px "
         + Math.abs(b) + "px " + Math.abs(l) + "px !important;";

    if (element)
        element.style.cssText = css;
    else
        return css;
}

function moveImp(element, x, y)
{
    var css = "left:" + x + "px !important;top:" + y + "px !important;";

    if (element)
        element.style.cssText = css;
    else
        return css;
}

function resizeImp(element, w, h)
{
    var css = "width:" + w + "px !important;height:" + h + "px !important;";

    if (element)
        element.style.cssText = css;
    else
        return css;
}

//
     ***********************************************************************************************
     ** //
// Imagemap Highlighter
```

```
function getImageMapHighlighter(context)
{
    if (!context)
        return;

    var canvas, ctx, mx, my;
    var doc = context.window.document;

    var init = function(elt)
    {
        if (elt)
            doc = elt.ownerDocument;

        canvas = doc.getElementById("firebugCanvas");

        if (!canvas)
        {
            canvas = doc.createElementNS("http://www.w3.org/1999/xhtml", "canvas");
            hideElementFromInspection(canvas);
            canvas.id = "firebugCanvas";
            canvas.className = "firebugResetStyles firebugBlockBackgroundColor firebugCanvas";
            canvas.width = context.window.innerWidth;
            canvas.height = context.window.innerHeight;

            Events.addEventListener(context.window, "scroll", function()
            {
                context.imageMapHighlighter.show(false);
            }, true);

            Events.addEventListener(doc, "mousemove", function(event)
            {
                mx = event.clientX;
                my = event.clientY;
            }, true);

            doc.body.appendChild(canvas);
        }
    };

    if (!context.imageMapHighlighter)
    {
        context.imageMapHighlighter =
        {
            ident: ident.imageMap,

            show: function(state)
            {
                if (!canvas)
                    init(null);

                canvas.style.cssText = "display:"+(state ? "block" : "none")+" !important";
            },

            getImages: function(mapName, multi)
            {
                if (!mapName)
                    return;

                var xpe = new XPathEvaluator();
                var nsResolver = xpe.createNSResolver(doc.documentElement);

                var elts = xpe.evaluate("//map[@name='" + mapName + "']", doc,
                    nsResolver, XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);

                if (elts.snapshotLength === 0)
                    return;

                elts = xpe.evaluate("(//img | //input)[@usemap='#" + mapName + "']",
                    doc.documentElement, nsResolver, XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
                             null);
                var eltsLen = elts.snapshotLength;

                var images = [];
                for (var i = 0; i < eltsLen; ++i)
                {
                    var elt = elts.snapshotItem(i);
                    var rect = Dom.getLTRBWH(elt);
                    if (multi)
                    {
                        images.push(elt);
                    }
                    else if (rect.left <= mx && rect.right >= mx && rect.top <= my &&
                        rect.bottom >= my)
                    {
                        images[0] = elt;
                        break;
                    }
                }

                return images;
            },

            highlight: function(eltArea, multi)
            {
                if (!eltArea || !eltArea.coords)
                    return;

                var images = this.getImages(eltArea.parentNode.name, multi) || [];

                init(eltArea);

                var v = eltArea.coords.split(",");

                if (!ctx)
                    ctx = canvas.getContext("2d");

                ctx.fillStyle = "rgba(135, 206, 235, 0.7)";
                ctx.strokeStyle = "rgb(44, 167, 220)";
                ctx.lineWidth = 2;

                if (images.length == 0)
                    images[0] = eltArea;

                for (var j = 0, imagesLen = images.length; j < imagesLen; ++j)
                {
                    var rect = Dom.getLTRBWH(images[j], context);

                    ctx.beginPath();

                    if (!multi || (multi && j===0))
                        ctx.clearRect(0, 0, canvas.width, canvas.height);

                    var shape = eltArea.shape.toLowerCase();

                    if (shape === "rect")
                    {
                        ctx.rect(rect.left + parseInt(v[0], 10), rect.top + parseInt(v[1], 10),
                                  v[2] - v[0], v[3] - v[1]);
                    }
                    else if (shape === "circle")
                    {
                        ctx.arc(rect.left + parseInt(v[0], 10) + ctx.lineWidth / 2, rect.top +
                                  parseInt(v[1], 10) + ctx.lineWidth / 2, v[2], 0, Math.PI / 180
                                  * 360, false);
                    }
                    else
                    {
                        var vLen = v.length;
                        ctx.moveTo(rect.left + parseInt(v[0], 10), rect.top + parseInt(v[1], 10)
                                  );
                        for (var i = 2; i < vLen; i += 2)
                            ctx.lineTo(rect.left + parseInt(v[i], 10), rect.top + parseInt(v[i +
                                       1], 10));
                        ctx.lineTo(rect.left + parseInt(v[0], 10), rect.top + parseInt(v[1], 10)
                                  );
                    }

                    ctx.fill();
                    ctx.stroke();
                    ctx.closePath();
                }

                this.show(true);
            },
            destroy: function()
            {
                this.show(false);
                canvas = null;
                ctx = null;
            }
        };
    }

    return context.imageMapHighlighter;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * //

var quickInfoBox =
{
    boxEnabled: undefined,
    dragging: false,
    storedX: null,
    storedY: null,
    prevX: null,
    prevY: null,

    show: function(element)
    {
        if (!this.boxEnabled || !element)
            return;

        this.needsToHide = false;

        var domAttribs = ["nodeName", "id", "name", "offsetWidth", "offsetHeight"];
        var cssAttribs = ["position"];
        var compAttribs = [
            "width", "height", "zIndex", "position", "top", "right", "bottom", "left",
            "margin-top", "margin-right", "margin-bottom", "margin-left", "color",
            "backgroundColor", "fontFamily", "cssFloat", "display", "visibility"];
        var qiBox = Firebug.chrome.$("fbQuickInfoPanel");

        if (qiBox.state==="closed")
        {
            this.storedX = this.storedX || Firefox.getElementById("content").tabContainer.
                   boxObject.screenX + 5;
            this.storedY = this.storedY || Firefox.getElementById("content").tabContainer.
                   boxObject.screenY + 35;

            // Dynamically set noautohide to avoid mozilla bug 545265.
            if (!this.noautohideAdded)
            {
                this.noautohideAdded = true;
                qiBox.addEventListener("popupshowing", function runOnce()
                {
                    qiBox.removeEventListener("popupshowing", runOnce, false);
                    qiBox.setAttribute("noautohide", true);
                }, false);
            }
            qiBox.openPopupAtScreen(this.storedX, this.storedY, false);
        }

        qiBox.removeChild(qiBox.firstChild);
        var vbox = document.createElement("vbox");
        qiBox.appendChild(vbox);

        var needsTitle = this.addRows(element, vbox, domAttribs);
        var needsTitle2 = this.addRows(element.style, vbox, cssAttribs);

        var lab;
        if (needsTitle || needsTitle2)
        {
            lab = document.createElement("label");
            lab.setAttribute("class", "fbQuickInfoBoxTitle");
            lab.setAttribute("value", Locale.$STR("quickInfo"));
            vbox.insertBefore(lab, vbox.firstChild);
        }

        lab = document.createElement("label");
        lab.setAttribute("class", "fbQuickInfoBoxTitle");
        lab.setAttribute("value", Locale.$STR("computedStyle"));
        vbox.appendChild(lab);

        this.addRows(element, vbox, compAttribs, true);
    },

    hide: function()
    {
        // if mouse is over panel defer hiding to mouseout to not cause flickering
        if (this.mouseover || this.dragging)
        {
            this.needsToHide = true;
            return;
        }

        var qiBox = Firebug.chrome.$("fbQuickInfoPanel");

        this.prevX = null;
        this.prevY = null;
        this.needsToHide = false;
        qiBox.hidePopup();
    },

    handleEvent: function(event)
    {
        switch (event.type)
        {
            case "mousemove":
                if(!this.dragging)
                    return;

                var diffX, diffY,
                    boxX = this.qiBox.screenX,
                    boxY = this.qiBox.screenY,
                    x = event.screenX,
                    y = event.screenY;

                diffX = x - this.prevX;
                diffY = y - this.prevY;

                this.qiBox.moveTo(boxX + diffX, boxY + diffY);

                this.prevX = x;
                this.prevY = y;
                this.storedX = boxX;
                this.storedY = boxY;
                break;
            case "mousedown":
                this.qiPanel = Firebug.chrome.$("fbQuickInfoPanel");
                this.qiBox = this.qiPanel.boxObject;
                Events.addEventListener(this.qiPanel, "mousemove", this, true);
                Events.addEventListener(this.qiPanel, "mouseup", this, true);
                this.dragging = true;
                this.prevX = event.screenX;
                this.prevY = event.screenY;
                break;
            case "mouseup":
                Events.removeEventListener(this.qiPanel, "mousemove", this, true);
                Events.removeEventListener(this.qiPanel, "mouseup", this, true);
                this.qiPanel = this.qiBox = null;
                this.prevX = this.prevY = null;
                this.dragging = false;
                break;
            // this is a hack to find when mouse enters and leaves panel
            // it requires that #fbQuickInfoPanel have border
            case "mouseover":
                if(this.dragging)
                    return;
                this.mouseover = true;
                break;
            case "mouseout":
                if(this.dragging)
                    return;
                this.mouseover = false;
                // if hiding was defered because mouse was over panel hide it
                if (this.needsToHide && event.target.nodeName == "panel")
                    this.hide();
                break;
        }
    },

    addRows: function(domBase, vbox, attribs, computedStyle)
    {
        if (!domBase)
            return;

        var needsTitle = false;
        for (var i = 0; i < attribs.length; i++)
        {
            var value;
            if (computedStyle)
            {
                var cs = getNonFrameBody(domBase).ownerDocument.defaultView.getComputedStyle
                        (domBase, null);
                value = cs.getPropertyValue(attribs[i]);

                if (value && /rgb\(\d+,\s\d+,\s\d+\)/.test(value))
                    value = rgbToHex(value);
            }
            else
            {
                value = domBase[attribs[i]];
            }

            if (value)
            {
                needsTitle = true;
                var hbox = document.createElement("hbox");
                var lab = document.createElement("label");
                lab.setAttribute("class", "fbQuickInfoName");
                lab.setAttribute("value", attribs[i]);
                hbox.appendChild(lab);
                var desc = document.createElement("label");
                desc.setAttribute("class", "fbQuickInfoValue");
                desc.appendChild(document.createTextNode(": " + value));
                hbox.appendChild(desc);
                vbox.appendChild(hbox);
            }
        }

        return needsTitle;
    }
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * //

Firebug.Inspector.FrameHighlighter = function()
{
};

Firebug.Inspector.FrameHighlighter.prototype =
{
    ident: ident.frame,

    doNotHighlight: function(element)
    {
        return false; // (element instanceof XULElement);
    },

    highlight: function(context, element, extra, colorObj, isMulti)
    {
        if (this.doNotHighlight(element))
            return;

        // if a single color was passed in lets use it as the border color
        if (typeof colorObj === "string")
            colorObj = {background: "transparent", border: colorObj};
        else
            colorObj = colorObj || {background: "transparent", border: "highlight"};

        Firebug.Inspector.attachRepaintInspectListeners(context, element);
        storeHighlighterParams(this, context, element, null, colorObj, null, isMulti);

        var cs;
        var offset = Dom.getLTRBWH(element);
        var x = offset.left, y = offset.top;
        var w = offset.width, h = offset.height;

        if (FBTrace.DBG_INSPECT)
            FBTrace.sysout("FrameHighlighter HTML tag:" + element.tagName + " x:" + x +
                " y:" + y + " w:" + w + " h:" + h);

        var wacked = isNaN(x) || isNaN(y) || isNaN(w) || isNaN(h);
        if (wacked)
        {
            if (FBTrace.DBG_INSPECT)
                FBTrace.sysout("FrameHighlighter.highlight has bad boxObject for " + element.
                        tagName);

            return;
        }

        if (element.tagName !== "AREA")
        {
            if (FBTrace.DBG_INSPECT)
                FBTrace.sysout("FrameHighlighter " + element.tagName);
            var body = getNonFrameBody(element);
            if (!body)
                return this.unhighlight(context);

            this.ihl && this.ihl.show(false);

            quickInfoBox.show(element);
            var highlighter = this.getHighlighter(context, isMulti);
            var bgDiv = highlighter.firstChild;
            var css = moveImp(null, x, y) + resizeImp(null, w, h);

            if (Dom.isElement(element))
            {
                cs = body.ownerDocument.defaultView.getComputedStyle(element, null);

                if (cs.transform && cs.transform != "none")
                    css += "transform: " + cs.transform + " !important;" +
                           "transform-origin: " + cs.transformOrigin + " !important;";
                if (cs.borderRadius)
                    css += "border-radius: " + cs.borderRadius + " !important;";
                if (cs.borderTopLeftRadius)
                    css += "border-top-left-radius: " + cs.borderTopLeftRadius + " !important;";
                if (cs.borderTopRightRadius)
                    css += "border-top-right-radius: " + cs.borderTopRightRadius + " !
                        important;";
                if (cs.borderBottomRightRadius)
                    css += "border-bottom-right-radius: " + cs.borderBottomRightRadius + " !
                        important;";
                if (cs.borderBottomLeftRadius)
                    css += "border-bottom-left-radius: " + cs.borderBottomLeftRadius + " !
                        important;";
            }
            css += "box-shadow: 0 0 2px 2px "+
                (colorObj && colorObj.border ? colorObj.border : "highlight")+"!important;";

            if (colorObj && colorObj.background)
            {
                bgDiv.style.cssText = "width: 100%!important; height: 100%!important;" +
                    "background-color: "+colorObj.background+"!important; opacity: 0.6!
                        important;";
            }
            else
            {
                bgDiv.style.cssText = "background-color: transparent!important;";
            }

            highlighter.style.cssText = css;

            var needsAppend = !highlighter.parentNode || highlighter.ownerDocument != body.
                ownerDocument;
            if (needsAppend)
            {
                if (FBTrace.DBG_INSPECT)
                    FBTrace.sysout("FrameHighlighter needsAppend: " + highlighter.ownerDocument.
                             documentURI +
                        " !?= " + body.ownerDocument.documentURI, highlighter);
                attachStyles(context, body.ownerDocument);

                try
                {
                    body.appendChild(highlighter);
                }
                catch(exc)
                {
                    if (FBTrace.DBG_INSPECT)
                        FBTrace.sysout("inspector.FrameHighlighter.highlight body.appendChild
                             FAILS for body " +
                            body + " " + exc, exc);
                }

                // otherwise the proxies take up screen space in browser.xul
                if (element.ownerDocument && element.ownerDocument.contentType.indexOf("xul") ==
                        = -1)
                    createProxiesForDisabledElements(body);
            }
        }
        else
        {
            this.ihl = getImageMapHighlighter(context);
            this.ihl.highlight(element, false);
        }
    },

    unhighlight: function(context)
    {
        if (FBTrace.DBG_INSPECT)
            FBTrace.sysout("FrameHighlighter unhighlight", context.window.location);

        var highlighter = this.getHighlighter(context);
        var body = highlighter.parentNode;
        if (body)
        {
            body.removeChild(highlighter);
            quickInfoBox.hide();
        }

        this.ihl && this.ihl.destroy();
        this.ihl = null;
    },

    getHighlighter: function(context, isMulti)
    {
        if (!isMulti)
        {
            var div = HighlighterCache.get(ident.frame);
            if (div)
                return div;
        }

        var doc = context.window.document;
        div = doc.createElementNS("http://www.w3.org/1999/xhtml", "div");
        var div2 = doc.createElementNS("http://www.w3.org/1999/xhtml", "div");

        hideElementFromInspection(div);
        hideElementFromInspection(div2);

        div.className = "firebugResetStyles firebugBlockBackgroundColor";
        div2.className = "firebugResetStyles";
        div.appendChild(div2);
        div.ident = ident.frame;
        HighlighterCache.add(div);
        return div;
    }
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * //

function BoxModelHighlighter()
{
}

Firebug.Inspector.BoxModelHighlighter = BoxModelHighlighter;
```

```
BoxModelHighlighter.prototype =
{
    ident: ident.boxModel,

    highlight: function(context, element, boxFrame, colorObj, isMulti)
    {
        var line, contentCssText, paddingCssText, borderCssText, marginCssText,
            nodes = this.getNodes(context, isMulti),
            highlightFrame = boxFrame ? nodes[boxFrame] : null;

        // if a single color was passed in lets use it as the content box color
        if (typeof colorObj === "string")
            colorObj = {content: colorObj, padding: "SlateBlue", border: "#444444", margin:
                   "#EDFF64"};
        else
            colorObj = colorObj || {content: "SkyBlue", padding: "SlateBlue", border: "#444444",
                   margin: "#EDFF64"};

        Firebug.Inspector.attachRepaintInspectListeners(context, element);
        storeHighlighterParams(this, context, element, boxFrame, colorObj, null, isMulti);

        if (context.highlightFrame)
            Css.removeClass(context.highlightFrame, "firebugHighlightBox");

        if (element.tagName !== "AREA")
        {
            this.ihl && this.ihl.show(false);

            quickInfoBox.show(element);
            context.highlightFrame = highlightFrame;

            if (highlightFrame)
            {
                Css.setClass(nodes.offset, "firebugHighlightGroup");
                Css.setClass(highlightFrame, "firebugHighlightBox");
            }
            else
                Css.removeClass(nodes.offset, "firebugHighlightGroup");

            var win = (element.ownerDocument ? element.ownerDocument.defaultView : null);
            if (!win)
                return;

            var style = win.getComputedStyle(element, "");
            if (!style)
            {
                if (FBTrace.DBG_INSPECT)
                    FBTrace.sysout("highlight: no style for element " + element, element);
                return;
            }

            var styles = Css.readBoxStyles(style);
            var offset = Dom.getLTRBWH(element);
            var x = offset.left - Math.abs(styles.marginLeft);
            var y = offset.top - Math.abs(styles.marginTop);
            var w = offset.width - (styles.paddingLeft + styles.paddingRight + styles.borderLeft
                   + styles.borderRight);
            var h = offset.height - (styles.paddingTop + styles.paddingBottom + styles.borderTop
                   + styles.borderBottom);

            moveImp(nodes.offset, x, y);
            marginCssText = pad(null, styles.marginTop, styles.marginRight, styles.marginBottom,
                   styles.marginLeft);
            borderCssText = pad(null, styles.borderTop, styles.borderRight, styles.borderBottom,
                   styles.borderLeft);
            paddingCssText = pad(null, styles.paddingTop, styles.paddingRight, styles.
                   paddingBottom, styles.paddingLeft);
            contentCssText = resizeImp(null, w, h);

            // element.tagName !== "BODY" for issue 2447. hopefully temporary, robc
            var showLines = Firebug.showRulers && boxFrame && element.tagName !== "BODY";
            if (showLines)
            {
                var offsetParent = element.offsetParent;

                if (offsetParent)
                    this.setNodesByOffsetParent(win, offsetParent, nodes);
                var left = x;
                var top = y;
                var width = w-1;
                var height = h-1;

                if (boxFrame == "content")
                {
                    left += Math.abs(styles.marginLeft) + Math.abs(styles.borderLeft)
                        + Math.abs(styles.paddingLeft);
                    top += Math.abs(styles.marginTop) + Math.abs(styles.borderTop)
                        + Math.abs(styles.paddingTop);
                }
                else if (boxFrame == "padding")
                {
                    left += Math.abs(styles.marginLeft) + Math.abs(styles.borderLeft);
                    top += Math.abs(styles.marginTop) + Math.abs(styles.borderTop);
                    width += Math.abs(styles.paddingLeft) + Math.abs(styles.paddingRight);
                    height += Math.abs(styles.paddingTop) + Math.abs(styles.paddingBottom);
                }
                else if (boxFrame == "border")
                {
                    left += Math.abs(styles.marginLeft);
                    top += Math.abs(styles.marginTop);
                    width += Math.abs(styles.paddingLeft) + Math.abs(styles.paddingRight)
                         + Math.abs(styles.borderLeft) + Math.abs(styles.borderRight);
                    height += Math.abs(styles.paddingTop) + Math.abs(styles.paddingBottom)
                        + Math.abs(styles.borderTop) + Math.abs(styles.borderBottom);
                }
                else if (boxFrame == "margin")
                {
                    width += Math.abs(styles.paddingLeft) + Math.abs(styles.paddingRight)
                         + Math.abs(styles.borderLeft) + Math.abs(styles.borderRight)
                         + Math.abs(styles.marginLeft) + Math.abs(styles.marginRight);
                    height += Math.abs(styles.paddingTop) + Math.abs(styles.paddingBottom)
                        + Math.abs(styles.borderTop) + Math.abs(styles.borderBottom)
                        + Math.abs(styles.marginTop) + Math.abs(styles.marginBottom);
                }

                moveImp(nodes.lines.top, 0, top);
                moveImp(nodes.lines.right, left + width, 0);
                moveImp(nodes.lines.bottom, 0, top + height);
                moveImp(nodes.lines.left, left, 0);
            }

            var body = getNonFrameBody(element);
            if (!body)
                return this.unhighlight(context);

            if (colorObj.content)
                nodes.content.style.cssText = contentCssText + " background-color: " + colorObj.
                        content + " !important;";
            else
                nodes.content.style.cssText = contentCssText + " background-color: #87CEEB !
                        important;";

            if (colorObj.padding)
                nodes.padding.style.cssText = paddingCssText + " background-color: " + colorObj.
                        padding + " !important;";
            else
                nodes.padding.style.cssText = paddingCssText + " background-color: #6A5ACD !
                        important;";

            if (colorObj.border)
                nodes.border.style.cssText = borderCssText + " background-color: " + colorObj.
                        border + " !important;";
            else
                nodes.border.style.cssText = borderCssText + " background-color: #444444 !
                        important;";

            if (colorObj.margin)
                nodes.margin.style.cssText = marginCssText + " background-color: " + colorObj.
                        margin + " !important;";
            else
                nodes.margin.style.cssText = marginCssText + " background-color: #EDFF64 !
                        important;";

            var needsAppend = !nodes.offset.parentNode
                || nodes.offset.parentNode.ownerDocument != body.ownerDocument;
```

```
            if (needsAppend)
            {
                attachStyles(context, body.ownerDocument);
                body.appendChild(nodes.offset);
            }

            if (showLines)
            {
                if (!nodes.lines.top.parentNode)
                {
                    if (nodes.parent)
                        body.appendChild(nodes.parent);

                    for (line in nodes.lines)
                        body.appendChild(nodes.lines[line]);
                }
            }
            else if (nodes.lines.top.parentNode)
            {
                if (nodes.parent)
                    body.removeChild(nodes.parent);

                for (line in nodes.lines)
                    body.removeChild(nodes.lines[line]);
            }
        }
        else
        {
            this.ihl = getImageMapHighlighter(context);
            this.ihl.highlight(element, true);
        }
    },

    unhighlight: function(context)
    {
        HighlighterCache.clear();
        quickInfoBox.hide();
    },

    getNodes: function(context, isMulti)
    {
        if (context.window)
        {
            var doc = context.window.document;

            if (FBTrace.DBG_ERRORS && !doc)
                FBTrace.sysout("inspector getNodes no document for window:" + window.location);
            if (FBTrace.DBG_INSPECT && doc)
                FBTrace.sysout("inspect.getNodes doc: " + doc.location);

            if (!isMulti)
            {
                var nodes = HighlighterCache.get(ident.boxModel);
                if (nodes)
                    return nodes;
            }

            var Ruler = "firebugResetStyles firebugBlockBackgroundColor firebugRuler
                  firebugRuler";
            var Box = "firebugResetStyles firebugBlockBackgroundColor firebugLayoutBox
                  firebugLayoutBox";
            var CustomizableBox = "firebugResetStyles firebugLayoutBox";
            var Line = "firebugResetStyles firebugBlockBackgroundColor firebugLayoutLine
                  firebugLayoutLine";

            function create(className, name)
            {
                var div = doc.createElementNS("http://www.w3.org/1999/xhtml", "div");
                hideElementFromInspection(div);

                if (className !== CustomizableBox)
                    div.className = className + name;
                else
                    div.className = className;

                return div;
            }
            nodes =
            {
                parent: create(Box, "Parent"),
                rulerH: create(Ruler, "H"),
                rulerV: create(Ruler, "V"),
                offset: create(Box, "Offset"),
                margin: create(CustomizableBox, "Margin"),
                border: create(CustomizableBox, "Border"),
                padding: create(CustomizableBox, "Padding"),
                content: create(CustomizableBox, "Content"),
                lines: {
                    top: create(Line, "Top"),
                    right: create(Line, "Right"),
                    bottom: create(Line, "Bottom"),
                    left: create(Line, "Left")
                }
            };

            nodes.parent.appendChild(nodes.rulerH);
            nodes.parent.appendChild(nodes.rulerV);
            nodes.offset.appendChild(nodes.margin);
            nodes.margin.appendChild(nodes.border);
            nodes.border.appendChild(nodes.padding);
            nodes.padding.appendChild(nodes.content);
        }

        nodes.ident = ident.boxModel;
        HighlighterCache.add(nodes);
        return nodes;
    },

    setNodesByOffsetParent: function(win, offsetParent, nodes)
    {
        var parentStyle = win.getComputedStyle(offsetParent, "");
        var parentOffset = Dom.getLTRBWH(offsetParent);
        var parentX = parentOffset.left + parseInt(parentStyle.borderLeftWidth, 10);
        var parentY = parentOffset.top + parseInt(parentStyle.borderTopWidth, 10);
        var parentW = offsetParent.offsetWidth-1;
        var parentH = offsetParent.offsetHeight-1;

        nodes.parent.style.cssText = moveImp(null, parentX, parentY) + resizeImp(null, parentW,
              parentH);

        if (parentX < 14)
            Css.setClass(nodes.parent, "overflowRulerX");
        else
            Css.removeClass(nodes.parent, "overflowRulerX");

        if (parentY < 14)
            Css.setClass(nodes.parent, "overflowRulerY");
        else
            Css.removeClass(nodes.parent, "overflowRulerY");
    }
};

//
     ***********************************************************************************************
     ** //

function getNonFrameBody(elt)
{
    if (Dom.isRange(elt))
    {
        elt = elt.commonAncestorContainer;
    }
    var body = Dom.getBody(elt.ownerDocument);
    return (body.localName && body.localName.toUpperCase() === "FRAMESET") ? null : body;
}

function attachStyles(context, doc)
{
    if (!context.highlightStyleCache)
        context.highlightStyleCache = new WeakMap();
    var highlightStyleCache = context.highlightStyleCache;

    var style;
    if (highlightStyleCache.has(doc))
    {
        style = highlightStyleCache.get(doc);
    }
    else
    {
        style = Css.createStyleSheet(doc, highlightCssUrl);
        highlightStyleCache.set(doc, style);
    }

    // Cater for the possiblity that someone might have removed our stylesheet.
    if (!style.parentNode)
        Css.addStyleSheet(doc, style);
}

function createProxiesForDisabledElements(body)
{
    var i, rect, div, node, cs, css,
        doc = body.ownerDocument,
        xpe = new XPathEvaluator(),
        nsResolver = xpe.createNSResolver(doc.documentElement);

    var result = xpe.evaluate("//*[@disabled]", doc.documentElement,
        nsResolver, XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);

    var l = result.snapshotLength;
    for (i=0; i<l; i++)
    {
        node = result.snapshotItem(i);
        cs = body.ownerDocument.defaultView.getComputedStyle(node, null);
        rect = node.getBoundingClientRect();
        div = doc.createElementNS("http://www.w3.org/1999/xhtml", "div");
        hideElementFromInspection(div);
        div.className = "firebugResetStyles fbProxyElement";

        css = moveImp(null, rect.left, rect.top + body.scrollTop) + resizeImp(null, rect.width,
              rect.height);
        if (cs.transform && cs.transform != "none")
            css += "transform:" + cs.transform + " !important;" +
                   "transform-origin:" + cs.transformOrigin + " !important;";
        if (cs.borderRadius)
            css += "border-radius:" + cs.borderRadius + " !important;";
        if (cs.borderTopLeftRadius)
            css += "border-top-left-radius:" + cs.borderTopLeftRadius + " !important;";
        if (cs.borderTopRightRadius)
            css += "border-top-right-radius:" + cs.borderTopRightRadius + " !important;";
        if (cs.borderBottomRightRadius)
            css += "border-bottom-right-radius:" + cs.borderBottomRightRadius + " !important;";
        if (cs.borderBottomLeftRadius)
            css += "border-bottom-left-radius:" + cs.borderBottomLeftRadius + " !important;";

        div.style.cssText = css;
        div.fbProxyFor = node;

        body.appendChild(div);
        div.ident = ident.proxyElt;
        HighlighterCache.add(div);
    }
}

function rgbToHex(value)
{
    return value.replace(/\brgb\((\d{1,3}),\s*(\d{1,3}),\s*(\d{1,3})\)/gi, function(_, r, g, b)
    {
        return "#"+((1 << 24) + (r << 16) + (g << 8) + (b << 0)).toString(16).substr(-6).
              toUpperCase();
    });
}

function isVisibleElement(elt)
{
    var invisibleElements =
        {
            "head": true,
            "base": true,
            "basefont": true,
            "isindex": true,
            "link": true,
            "meta": true,
            "script": true,
            "style": true,
            "title": true
        };

    return !invisibleElements[elt.nodeName.toLowerCase()];
}

function hideElementFromInspection(elt)
{
    if (!FBTrace.DBG_INSPECT)
        Firebug.setIgnored(elt);
}

// highlightType is only to be used for multihighlighters
function storeHighlighterParams(highlighter, context, element, boxFrame, colorObj,
    highlightType, isMulti)
{
    var fir = Firebug.Inspector.repaint;

    fir.highlighter = highlighter;
    fir.context = context;
    fir.element = element;
    fir.boxFrame = boxFrame;
    fir.colorObj = colorObj;
    fir.highlightType = highlightType;
    fir.isMulti = isMulti;

    Firebug.Inspector.highlightedContext = context;
}

//
     *******************************************************************************************
     ** //
// Registration

Firebug.registerModule(Firebug.Inspector);

return Firebug.Inspector;

//
     *******************************************************************************************
     ** //
});
```