Manufacturing Integration and Intelligence (SAP xMII) JavaScript Debugging

Applies to:

JavaScript debugging in SAP xApp Manufacturing Integration and Intelligence (SAP xMII) 11.5

Summary

SAP xMII enables you to build applications using HTML and Java applets to monitor and manage your shop floor. You can use JavaScript to enhance the functionality and usability of your applications. The SAP xMII installation includes a debugging tool to help you develop these applications and streamline your JavaScript. This article describes how to use this debugging tool.

Author(s): Bob Elam

Company: SAP

Created on: 14 July 2006

Author Bio

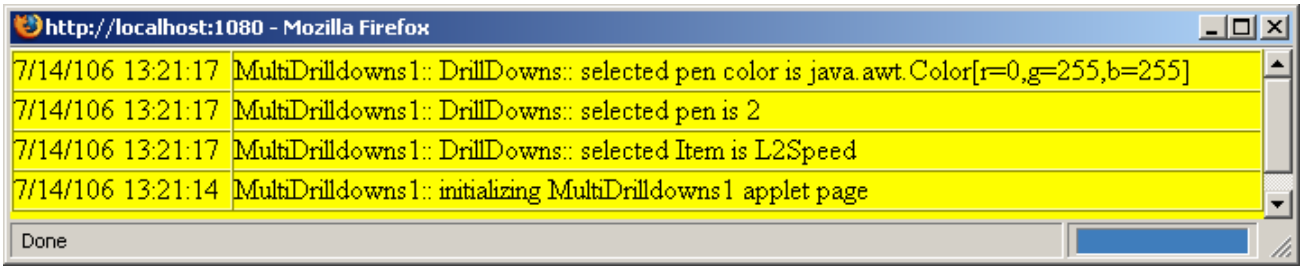
Bob Elam is a Java developer for the xMII application specializing in JavaScript front-end development.

Table of Contents

Debugging Tool Overview

JavaScript developers typically use the alert() function to display debug information as they work on their application. Alerts are easy to build, and they quickly show debug information on the screen as your application is running; however, alerts stop the processing of your application with their model dialogs. When you require more than two Requiring and displaying alerts, your application can get cumbersome.
SAP xMII provides a JavaScript debugging tool to make your application building and debugging more effective. When you use this tool, debug messages are written to a debug screen. This allows your application to perform its functions and write debug messages at the same time. The display window also acts as a log file and records your debug messages.

Below is an example of the JavaScript debugging display screen:



If you set it up correctly, this screen appears after your page initializes and records your debug messages as the page performs its function. Most recent messages are displayed at the top of the page. Each message gets a date and time stamp (in some browsers the year is displayed as the current year minus 1900). The messages are what would have been written to an alert.

Since the debugger can run across multiple pages or applications, I preface each of my messages with the page name and JavaScript function that is writing the message. In the example above, I opened a page named *MultiDrilldowns1* and performed an action that executed the *DrillDowns* function of that page. For debugging purposes, I want to check which item and pen were selected and the pen color when the function was executed.

Setting Up the Debugger

The debugger JavaScript object is included in the SAP xMII installation. It is located in the JavaScript directory of the Lighthammer Web application for 11.5.

1. Add the debugger JavaScript object reference to your page or application.
2. Add the following line in the header section (<head></head>) of your HTML (or .irpt):

```
   <SCRIPT type="text/JavaScript"
src="http://{yourservername}/Lighthammer/JavaScript/debugger.js"></SCRIPT>
```

3. Create an instance of the debugger in your page.
4. In the Script section (<script></script>) of your page, create the following object:

   ```
   var jsDebugger = new JSDebugger(true);
   ```

   You now have a debugger object.

Debugging Functions

The following list describes available debugging functions:

- setEnabled(boolean)

  You can use this function to enable or disable the debugger. If the debugger is disabled, you cannot write messages to it.

- show()

  You can use this function to display the debugger window. You can write messages to the debugger without displaying it. If it is running in the background, you can call it so it will display.

- close()

  You can use this function to close the debugger window that is open.

- clear()

  You can use this function to clear the debugger of any messages.

- write()

  You can use thi function to write messages in the debugger.

Displaying the Debugging Screen

There are many ways to display your debugger.

I like to have my debugger displayed and running when I open my page, so I create an initialize() function that is called during the Body onload() event. The code looks something like the following:

```
function initialize() {

jsDebugger.write('MultiDrilldowns1:: initializing MultiDrilldowns1 applet page');
jsDebugger.show();

}
```

I then call the initialize() function from the Body onload() event, so when the body of the page loads, my debugger displays.

You could also use Body onload=”jsDebugger.show()” if there is nothing else to initialize or execute when the page body is loaded.

You could keep the debugger hidden until another event, like a button or checkbox onClick event, is executed on the page. Or, you could create error handling code that displays the debugger when certain error conditions are met in your JavaScript. Unlike an alert, which displays a limited amount of information, your debugger window could detail any amount of error debugging information. Your users could then copy the information (unlike an alert) and forward it to the appropriate developer.

After you decide how and when to show your debugging screen, you can write messages to it. In my initialize() example above, I write a message as soon as the function is called to indicate the function was called successfully. You can then add a jsDebugger.write() where you would normally put an alert.

The debugger JavaScript object is included with your xMII install. It can be found under the JavaScript directory of the Lighthammer webapp for 11.5.

Disclaimer & Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

