This presentation will review methods available to assist in debugging Web 2.0 applications.

Goals

- To provide an introduction to Firebug and motivate you to learn it
- To introduce trace statements available in the WebSphere® Commerce Ajax framework
- To review common Web 2.0 application errors and how to diagnose them

There are three main goals for this presentation. The first is to introduce the Firebug tool and motivate you to learn it. The second is to introduce the trace statements provided by the WebSphere Commerce Ajax framework that will help you with store front debugging. The final goal is to review some common Web 2.0 application errors in the Madisons starter store and how to diagnose them.

Agenda

- Web 2.0 debugging challenges
- Tools available
- Firebug overview
- Debugging in JavaScript
- Ajax framework trace statements
- Sample scenarios

This presentation starts with a discussion of Web 2.0 debugging challenges. It then moves on to a brief overview of debugging tools before focusing in more detail on the Firebug tool and methods for debugging JavaScript. Trace statements generated by the WebSphere Commerce Ajax framework are also discussed. The presentation concludes with some sample debugging scenarios.

Web 2.0 debugging challenges

- Increased use of JavaScript
- Asynchronous server communication
- Partial page updates
- CSS files can also add page function
- Browser differences

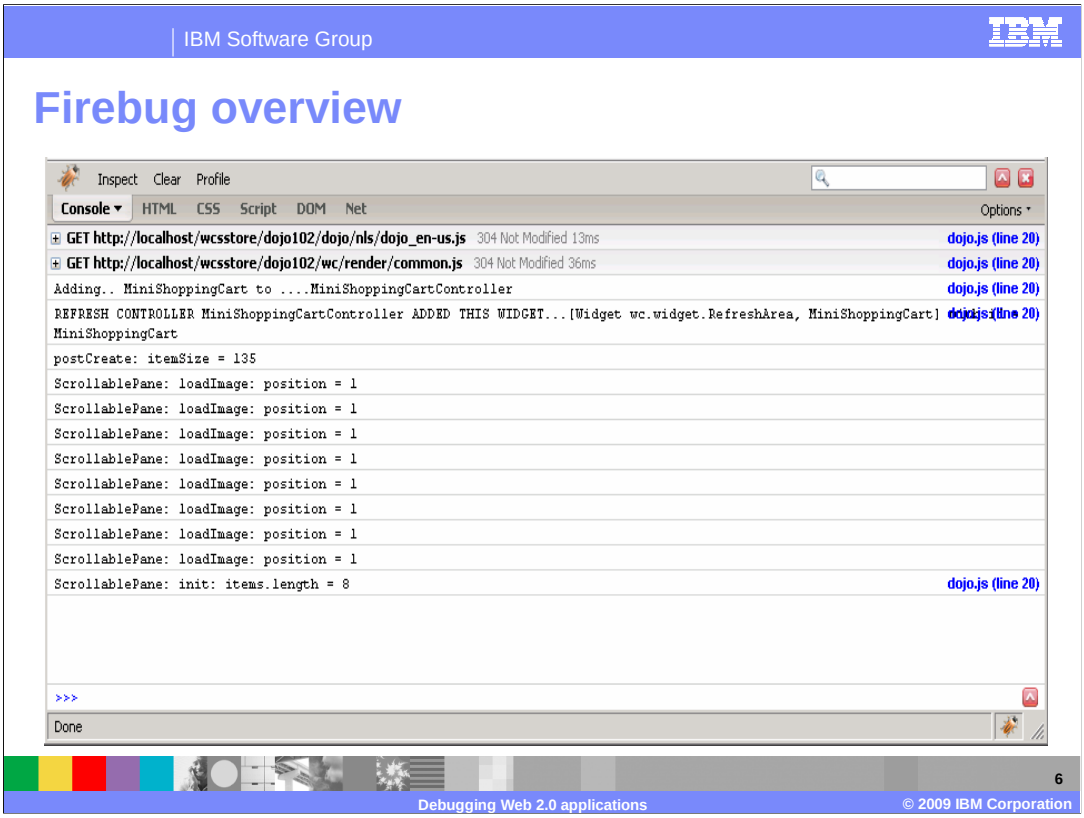
The same techniques that make Web 2.0 applications desirable from an end-user perspective also make them more challenging to write and debug. JavaScript provides much of the interactivity that is a key part of Web 2.0 applications but JavaScript development and debugging tools are not as mature as they are for languages like Java, C and C++. Asynchronous server communication and the resulting partial page updates provide another challenge by adding more points of failure for a page. Testing the page that initially loaded is not enough anymore. New content and function can be added to the page as a result of user interaction and these subsequent changes can introduce new problems. Often with Web 2.0 applications, the attempt to achieve a desktop like appearance drives more and more complex styling using CSS files. CSS files can add function on top of what is implemented in JavaScript files, further complicating the debugging process. Browser differences have been a debugging challenge in Web applications for many years and Web 2.0 applications are no exception. The Dojo toolkit helps to mitigate the problem. But with new applications pushing browser capabilities to their limits, implementation differences are bound to come up.

Debugging tools

- Mozilla Firefox
  - Firebug
  - Web Developer
- Internet Explorer®
  - Developer Toolbar
  - Microsoft® Script Debugger
  - Microsoft Script Editor
- Cross-browser
  - Firebug Lite

A number of different tools exist to help you in debugging Web applications. The WebSphere Commerce Information Center gives a short summary of some common tools for the Mozilla and Internet Explorer browsers. You can locate that information by searching on the term “debug dojo”. The Firebug add-on for Firefox is recommended by both WebSphere Commerce and the Dojo toolkit. The remainder of this presentation will discuss Firebug only.

Firebug overview



Firebug is a plug-in that can be added to Firefox browsers. It provides many useful tools for understanding and debugging Web applications. You can see the Console tab highlighted in this screen capture.

The Console tab displays messages and errors generated by the application. It also supports command statements issued against the currently loaded page.

The HTML tab allows you to inspect any HTML within the page including page content modified by Ajax requests. The HTML view also shows the styles applied to the selected page element.

The CSS tab allows you to view the contents of all CSS files loaded for the current page.

The Script tab allows you to view all JavaScript files loaded for the current page. You can also set breakpoints and step through function with the debugger.

The DOM tab allows you to view the DOM structure for the entire page.

The Net tab shows you what has been downloaded by the browser and what Ajax requests have been made.

Firebug console functions

- console.log()
- console.dir()
- console.time()
- console.profile()
- console.trace()
- console.group()

The Firebug console functions provide many useful debugging options. These functions can be called from the console command line or added directly to your JavaScript. A global console variable is added to your Web application automatically by Firebug.

Console.log writes messages and objects to the console. Multiple severity levels are available. These are discussed on the next slide.

Console.dir generates an interactive listing of all the properties and functions of a JavaScript object.

Console.time allows you to time the execution of a portion of JavaScript by adding a start and end point.

Console.profile launches the JavaScript profiler to track which functions were called and how long each took to complete.

Console.trace lists an interactive stack trace of JavaScript execution from the point where it was called.

Console.group adds indentation to your logging output.

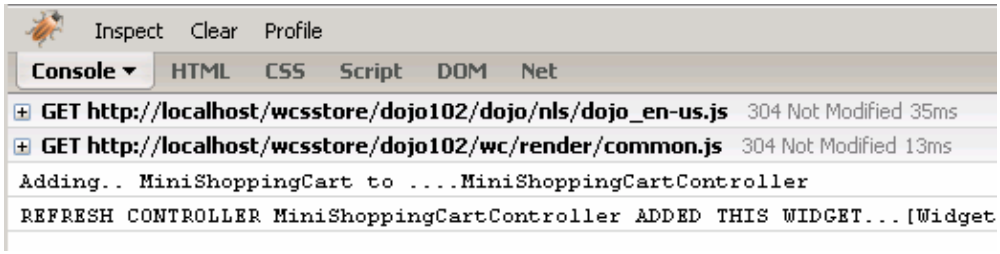
JavaScript logging

- Offers five severity levels
  - Log
  - Debug
  - Info
  - Warn
  - Error
- Can print out simple types: strings, integers
- Sample usage
  - console.debug(“Properties are: ” + properties);

The Madisons starter store takes advantage of the console logging functions to provide trace information for many JavaScript functions including the WebSphere Commerce Dojo extensions. There are five severity levels that can be used to format the output: log, debug, info, warn and error. In addition to string messages, the logging functions can also output JavaScript objects. Simple object types are printed directly to the console. More complex types are shown as a link to the DOM inspector.

JavaScript logging

- Log messages written to Firebug Console



This screen capture shows a sample of the console logging methods being used to output information about the MiniShoppingCart. The messages are generated by the RefreshArea and RefreshController classes of the WebSphere Commerce Ajax framework.

Debugging Dojo

- Use djConfig to specify debugging options
- isDebug: true
  - Enables use of Firebug console functions
  - Provides Firebug Lite debugging console for non Firefox browsers
- debugAtAllCosts: true
  - Provides more accurate error flagging
  - Do not use in production code

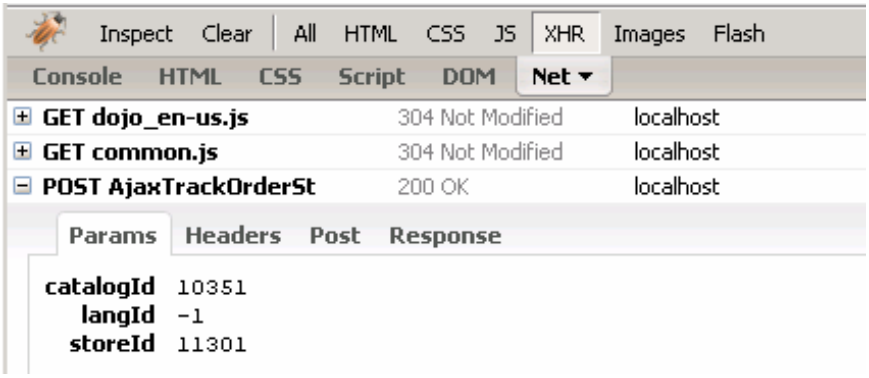
The djConfig attribute allows you to turn Dojo debugging options on and off.

The isDebug parameter, when set to true, displays a debug area at the bottom of your browser using Firebug Lite. This gives you access to the Firebug console messages and provides an area where you can type console commands. There are additional parameters that allow you to control the size and location of the debugging console. The Firebug plug-in for Firefox is able to output console messages even when isDebug is set to false.

The debugAtAllCosts parameter provides greater accuracy in locating errors within Dojo widgets. If you are extending or creating new Dojo widgets you might want to turn this on during development. The Madisons starter store sets both debug parameters to false by default.

Debugging Ajax calls

- Firebug's XHR tab



The XHR filter in the Net tab in Firebug is very valuable for debugging Ajax calls. Entries will only display here if they were loaded using the XMLHttpRequest object or, in other words, Ajax calls. When an Ajax call does not behave as expected, it can be tough to tell at what point the error occurred.

The first step is to confirm the Ajax request is displayed here. If there is no request ,there was either a failure before the call was invoked, or there was a communication error with the server. The Console tab and the JavaScript debugger can help you determine which. Any call that is successfully initiated will display here along with the parameters, header information, and posted data associated with the request. Your next debugging step is to confirm the request sent to the server was correct.

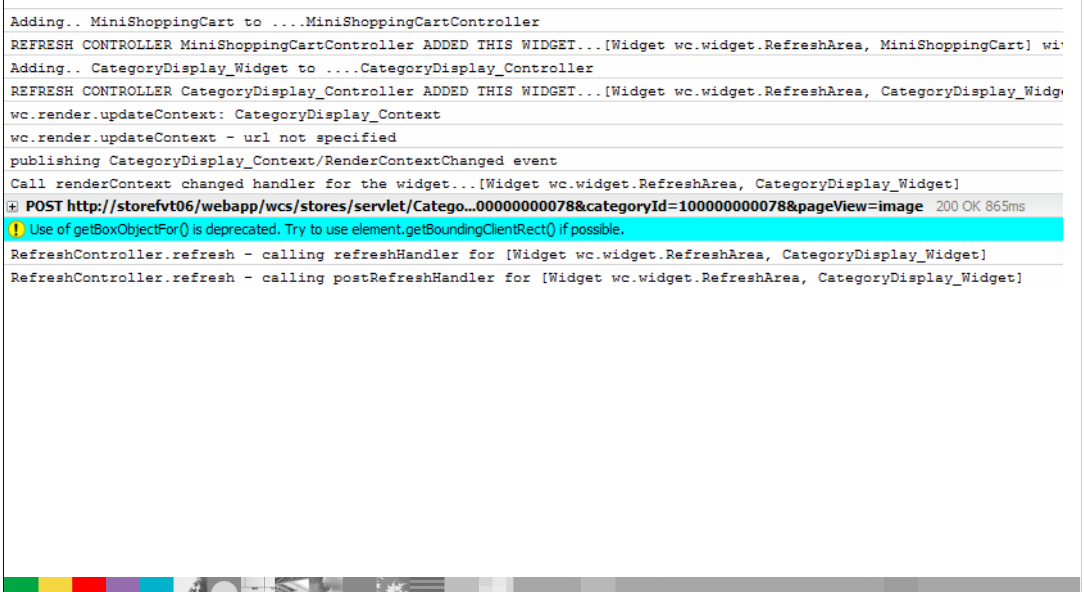
The fourth tab is Response. This tab contains all the data the server returned to the browser. Your next step in debugging is to make sure the response is what you expected. If an error response is returned, you can find more information in the server log files. If the response looks correct but something still seems to be wrong, it is time to move on and examine the code that processes the response data. You can use the JavaScript debugger to step through the response processing code, or use the HTML inspector to examine the updated page section in detail.

Tracing Ajax framework requests

- WebSphere Commerce custom Dojo classes include console.debug() trace statements
- Errors that are traced
  - Invoking an unregistered service
  - Declaring a duplicate RenderContext
  - Adding a duplicate widget to a RefreshController
  - Communication errors with the server

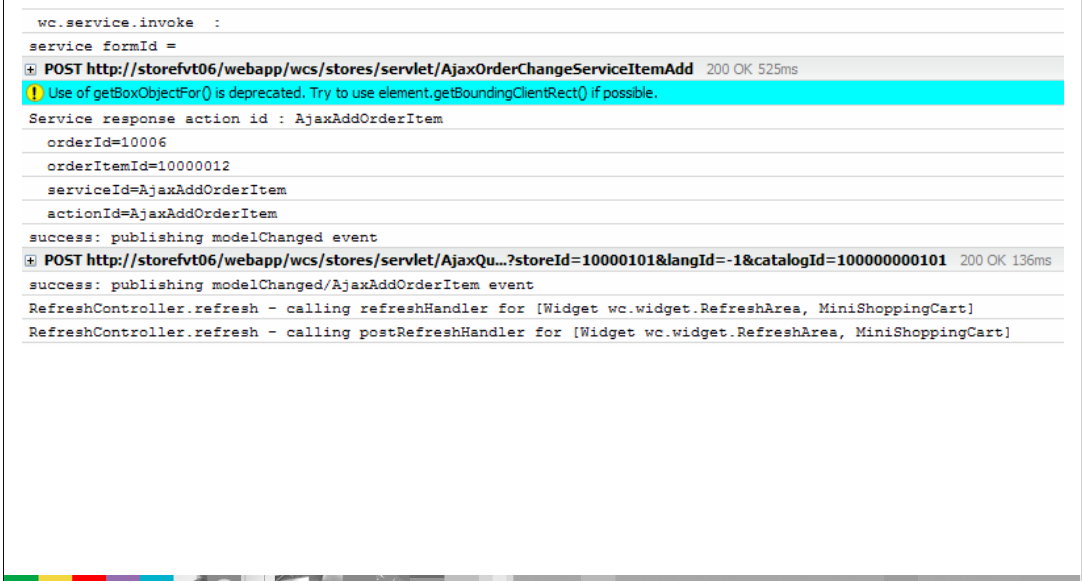
Since Web 2.0 applications push more functionality to the client, there is limited opportunity to trace back what went wrong in a shopper's browser. The console.debug function is used to provide some trace statements, but these are captured only within Firebug. There are four main error messages output by the WebSphere Commerce Ajax framework. They are: attempting to invoke an unregistered service, declaring a duplicate render context, adding a duplicate widget to a refresh controller, and communication errors with the server.

Tracing render context updates



This slide shows the Firebug output generated when a shopper pages through products in the catalog. The first four lines show refresh areas for the mini shopping cart and category display widget being registered with their respective refresh controllers. Next you see the render context being updated. This takes place when the shopper clicks on the next page arrow. A render context changed event is published and the render context changed handler for the category display widget is called. Next, you see a call to the server to get the next page of catalog data. This is the Ajax request. Ignoring the deprecation warning, the final two messages are for the refresh handler and post refresh handler being called.

Tracing service calls



This slide is similar to the last one but shows the Firebug output generated when a shopper adds an item to the cart. The messages showing the mini shopping cart being registered with its refresh controller are logged, but they have been omitted from this example for brevity. This example picks up with the **add to cart service** being invoked. The first call to the server is to update the shopper's order with the new item. When the service call returns, the service response actionId and response properties are logged. Next, the **model changed** events are published. The second call to the server is to get the updated mini shopping cart data. Ignoring the deprecation warning, the final two messages are for the refresh handler and post refresh handler being called.

Sample scenarios

- Dojo widget does not render correctly
- Service call does not update server
- Refresh area not updating

The next section of the presentation takes a look at some common Web 2.0 application problems, their possible causes, and ways to diagnose the problem. The three examples discussed are Dojo widgets not rendering correctly, service calls not resulting in an update on the server, and refresh areas not being updated correctly.

Dojo widget does not render correctly

- Missing or misspelled dojo.require
  - Flagged by Firebug
- Undefined or misspelled dojoType
  - Flagged by Firebug
- Widget not parsed
  - Class attribute does not contain a Dojo digit type
- Stylesheet error
  - Use Firebug to inspect style properties

The first sample situation is a Dojo widget not rendering correctly. Depending on the type of widget and the source of the error you might see nothing on the page, or you might see a portion of the widget content that is not correctly styled. Four possible reasons are discussed here.

First you should check for missing or misspelled **dojo.require** statements. Require statements provide dynamic loading of Dojo JavaScript as needed for the widgets you actually use. If you try to use a widget without the associated require statement you will see an error in Firebug.

After the require statement, the dojoType attribute is the other key part in declaring a Dojo widget. If the dojoType attribute is missing altogether, you are not declaring a Dojo widget. If you specify an undefined type, or misspell the dojoType attribute, Firebug will show an error when it fails to create an instance of the requested type.

The errors discussed so far are contingent on the widget being parsed because it is the parser that will recognize the errors or omissions. The one exception is a misspelled **require** statement which will fail when the browser attempts to load the corresponding JavaScript file. Not parsing a Dojo widget is the same as not declaring the dojoType attribute. That is, you do not have a Dojo widget. One way to identify if a widget has been parsed is to examine it using the Firebug HTML inspector. Check the class attribute of the Dojo widget you are trying to define. When Dojo parses a widget, it injects additional class names into the class attribute. These names typically include the word dijit.

Another possible source of errors is any additional styling you have applying to the widget using inline styles or a CSS file. Use Firebug to inspect the style properties and ensure they are being applied as expected. If necessary, check the styles against the actual stylesheet on the server. Some errors can result in individual attributes being omitted when the stylesheet is loaded.

Service call does not update server

- Service not defined
  - Debug messages in Firebug console
- Struts action not defined
  - Firebug error – Access to restricted URI denied
- User does not have access to run command
  - Access control error in server log file
- Command or Web service fails
  - Errors in server log file
- Communication error
  - Error and debug messages in Firebug console

The next sample situation is an Ajax framework service call not updating the server. The first thing to check in this case is that the service is correctly defined. Attempting to invoke an undeclared service will print out a debug statement in Firebug. The next thing to check is that the Struts action mapping is correctly defined. If the declaration is missing in the Struts extension file, you will see an error in Firebug when the service is invoked.

Once the service is defined and the Struts configuration is correct, the Ajax call should be reaching the server. The next problem you might run into is a user not having access to run the requested command or Web service. When this occurs you will see an error in the server log file. The response tab of the request in the Firebug XHR view will also show that an error occurred. If the command or Web service fails for any other reason, you can follow a similar debugging method. Check the server log files and the Ajax response in Firebug. Certain errors might be handled by the application, such as redirecting the shopper to the login page if their session has timed out.

Communication errors can have many causes and are difficult to diagnose from the client side. When a communication error occurs, you will see a debug message in the Firebug console that is generated by the Ajax framework. The request itself might be flagged as an error in the Firebug console as well. If the server is unreachable, every request will be failing. If the particular request suffered a non-recoverable error, there should be some information in the server log files.

Refresh area not updating

- Ensure widget is being parsed and service is returning correctly
- Refresh controller not defined
  - Message in Firebug console
- Errors in HTML returned
  - Firebug HTML inspector
- JavaScript in HTML returned
  - Firebug HTML inspector
- Errors in or working with JSON returned
  - Firebug DOM inspector or JavaScript debugger

The last scenario to look at is a refresh area not updating as expected. Since this is the final step in the update process, you need to confirm that the preceding steps completed correctly. The refresh area is a Dojo extension widget which must have been correctly defined and parsed. Also, the service update that led to the refresh area being updated must have completed correctly. One possible problem specific to refresh areas is that the refresh controller might not have been defined, or there are spelling errors. If this is the case, a message is displayed in Firebug, but is not flagged as an error.

Once you have determined that everything is correct with the refresh area definition and service request, the next place to look is the response itself. If HTML is returned, there might be errors in the response that are causing the problem. You can use the Firebug HTML inspector to examine the element and look for problems. The browser's "View source" menu option will not include any dynamically loaded HTML. One thing to look out for when examining the HTML is any JavaScript being returned. Returning JavaScript in a refresh area is not supported and will not function.

If the return format is JSON, then you need to look for errors both in the returned JSON data and in the JavaScript code that modifies the page DOM. The Firebug DOM inspector can be useful for examining both the JSON response, once it has been converted to JavaScript, and the modifications made to the page DOM. You can also use the JavaScript debugger to step through the code performing the updates.

Summary

- Learn Firebug
- Firebug console functions are available through a JavaScript object
- If you are new to Dojo review djConfig options and parser function
- Remember to check the Firebug console even if no errors are flagged
- Understanding the flow of the Ajax framework makes debugging easier

In summary, it is worth investing some time to learn the many features Firebug provides. Firebug's console functions are made available through a JavaScript object that you can use for trace and debug statements in your code.

If you are new to Dojo, it is worth getting familiar with the djConfig options and the parser functionality. This information is useful for debugging efforts.

The trace statements provided by the WebSphere Commerce Ajax framework are written to the Firebug console but do not generate the red X error indicator. When developing and debugging it is important to check the Firebug console trace statements even when no error is flagged.

Overall, understanding the flow of the Ajax framework will make your debugging work easier. When problems do occur it will be much easier to narrow down where the problem is located and what the likely causes are.

References

- Information center debugging tools summary

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.madisons-starterstore.doc/refs/rsmmadisondebug.htm

- Firebug logging
  - http://getfirebug.com/logging.html
- Dojo debugging tutorial

http://www.dojotoolkit.org/book/book-dojo/part-4-meta-dojo-making-your-dojo-code-run-faster-and-better/debugging-facilities/deb

This slide contains some useful references for Web 2.0 debugging.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WCS6005_DebuggingWeb20Applications.ppt

This module is also available in PDF format at: ../WCS6005_DebuggingWeb20Applications.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Internet Explorer, Microsoft, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JavaScript, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.