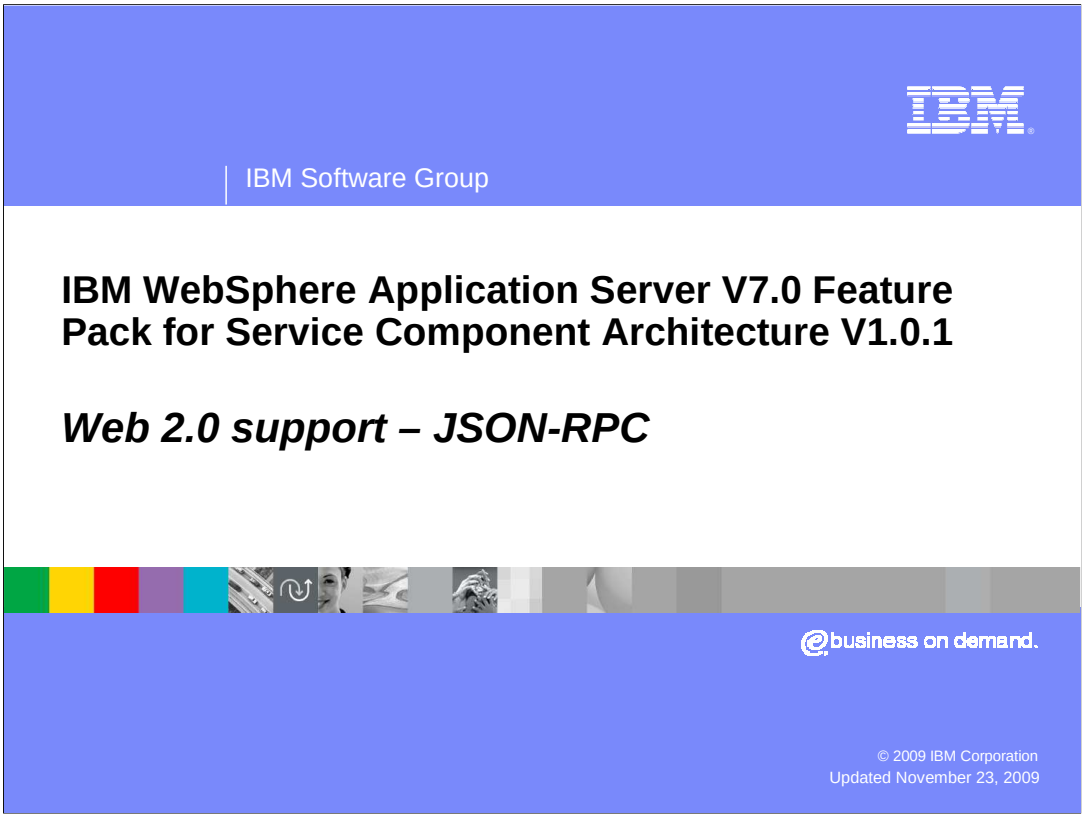
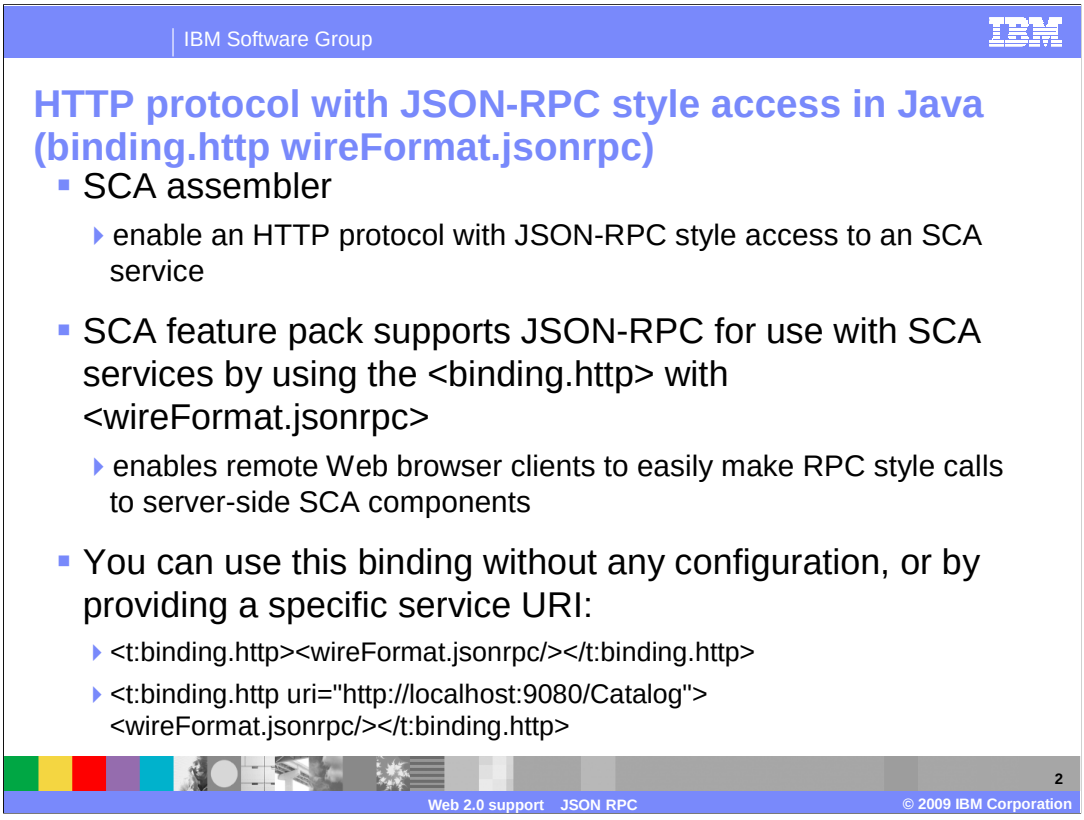


This presentation will cover SCA feature pack Web 2.0 support with JSON-RPC



Now you will look closer at the HTTP protocol with JSON-RPC style access in Java™. As an SCA Assembler, you want to have the ability to enable an HTTP protocol with JSON-RPC style access to an SCA service. This will enable a consumer to use the JSON-RPC style protocol to access your service without rewriting your service side implementations. The Feature Pack for SCA supports JSON-RPC as a protocol for use with SCA services by using the <binding.http> with <wireFormat.jsonrpc>. This enables remote Web browser clients to easily make RPC style calls to server-side SCA components.

You can use this binding without any configuration, or by providing a specific service URI.

<t:binding.http><wireFormat.jsonrpc/></t:binding.http>

<t:binding.http uri="http://localhost:9080/Catalog">
<wireFormat.jsonrpc/></t:binding.http>

The **t:** above represents the Tuscany namespace.

Consuming JSON-RPC services on the client application

- Any JSON-RPC client can be used to access the SCA services which use the <binding.http> with <wireFormat.jsonrpc>
- Ways to consume JSON-RPC services on the client application
  - Utilizing implementation.widget
  - Using SCA JSON-RPC services with Dojo

© 2009 IBM Corporation

Any JSON-RPC client can be used to access the SCA services which use the <binding.http> with <wireFormat.jsonrpc>. There are two ways to consume JSON-RPC services on the client application. One way is to use the implementation.widget which was described already. The other way is to use SCA JSON-RPC services with Dojo. The next few slides will show you in more detail the two ways to consume JSON-RPC services.

Utilizing implementation.widget for JSON-RPC

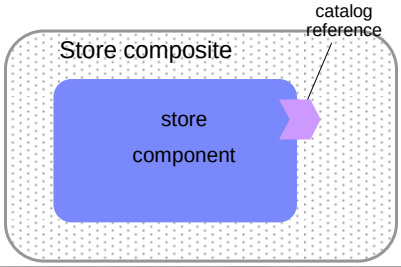
- Widget Implementation (implementation.widget), extends the SCA programming model to HTML or Web 2.0 client applications
  - For JSON-RPC allows you to model your HTML or Web 2.0 as an SCA component
  - Implementation artifact is an HTML resource where you can define a JSON-RPC reference, that can be wired to server side services

© 2009 IBM Corporation

As mentioned earlier the Widget Implementation, implementation.widget, extends the SCA programming model to HTML or Web 2.0 client applications. Using this for JSON-RPC allows you to model your HTML or Web 2.0 as an SCA component. The implementation artifact is an HTML resource where you can define a JSON-RPC reference, that can be wired to server side services.

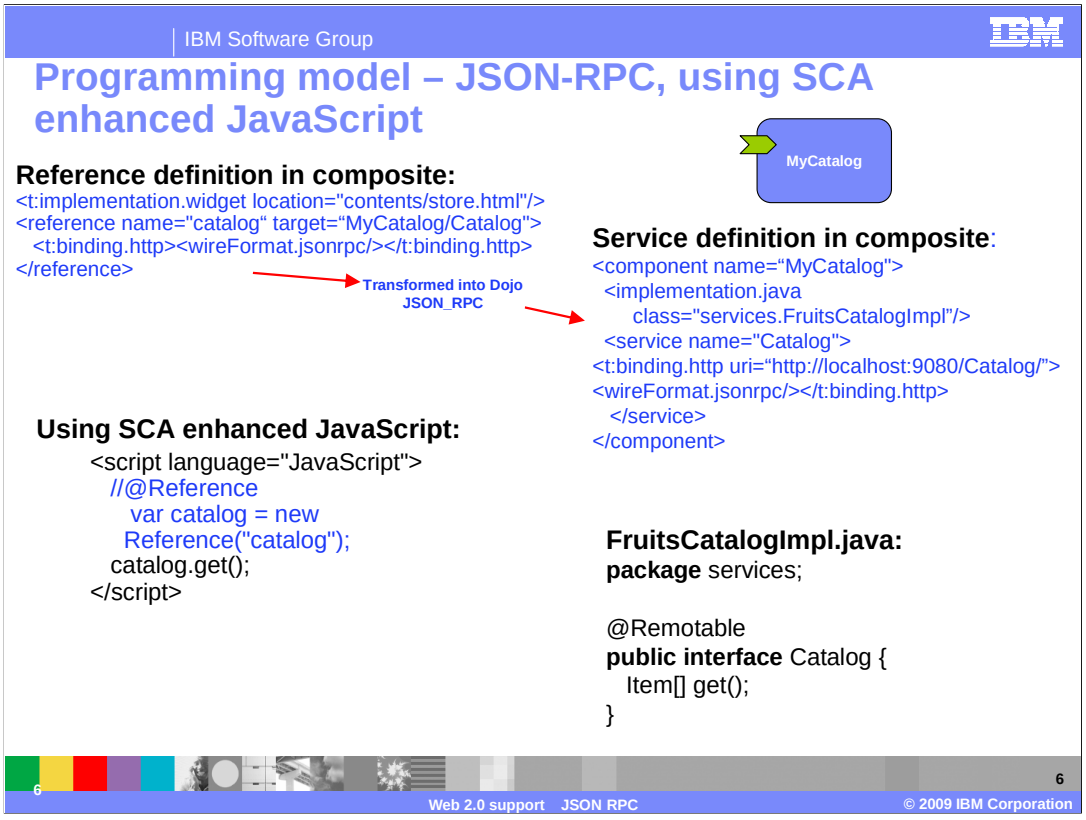
JSON-RPC style access in widget

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://store" name="store">
  <component name="store">
    <t:implementation.widget location="contents/store.html"/>
      <reference name="catalog" target="MyCatalog/Catalog">
        <t:binding.http><wireFormat.jsonrpc/></t:binding.http>
      </reference>
  </component> .....
</composite>
```

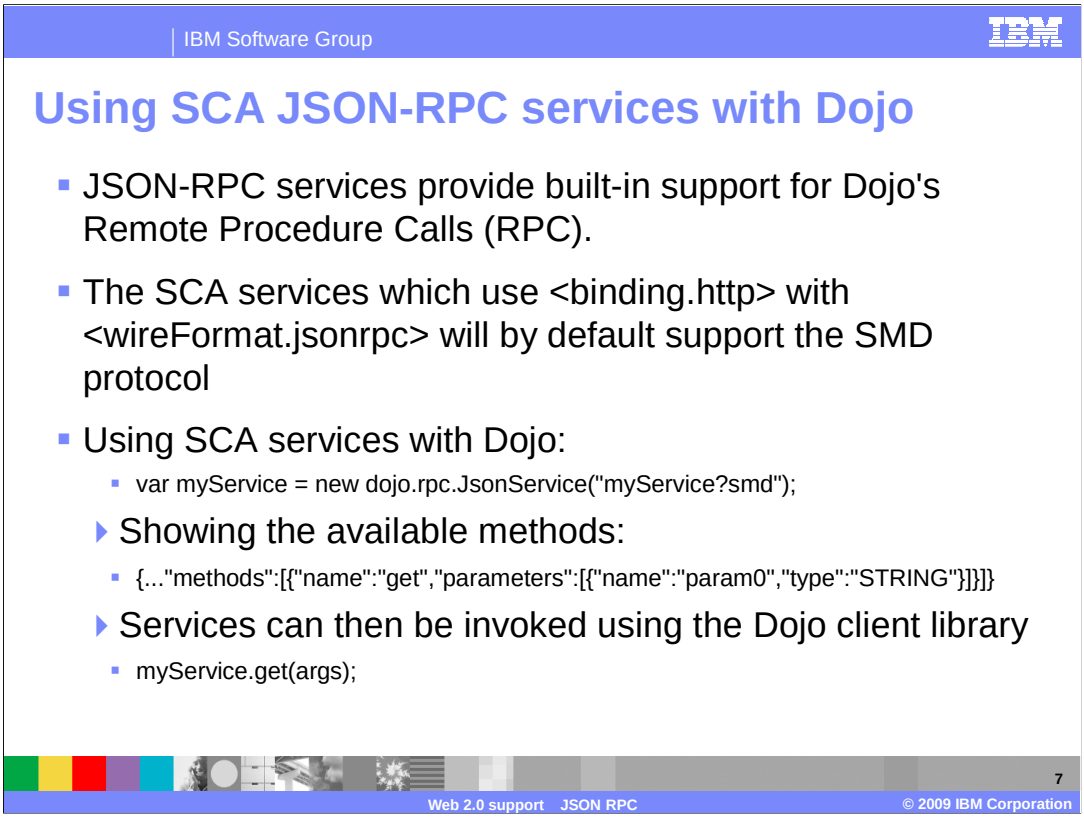


© 2009 IBM Corporation

The code above shows a sample composite file. The component store contains the implementation.widget. The implementation.widget component type allows you to model your HTML as an SCA component. The implementation artifact is an HTML resource, and above you can see how you can then define SCA References that can be wired to server side services. A catalog reference is defined for the store component which contains an HTTP binding with a wire format of JSON RPC. This catalog reference can be wired through HTTP binding with wire format JSON-RPC to server side services.



The code above depicts a more complete picture of how the catalog reference can be wired through an HTTP binding with wire format JSON-RPC to server side services using implementation.widget. You have starting in the left corner, a composite file that has a component that uses the implementation widget. A reference named catalog is then defined using an HTTP binding with wire format JSON-RPC. This reference targets an existing service called Catalog. That service is defined to the top right in a composite file with a component name of MyCatalog. This component contains the service named Catalog that is defined with an HTTP binding with wire format JSON-RPC. This component defines its Java implementation to be FruitsCatalogImpl.java. Notice on the bottom right the FruitsCatalogImpl.java file contains a get() method. Now lastly, you will notice the SCA enhanced JavaScript code. Since widget implementation was used, you can now introduce SCA annotations to the JavaScript code. You can introduce the catalog reference and easily now call the get() method that was defined in FruitsCatalogImpl.java.



Dojo provides a basic RPC client class that has been extended to provide access to JSON-RPC services. It was designed so that it is easy to implement custom RPC services. Dojo's RPC clients simplify this process by taking a simple definition of the remote methods and application needs and generating client side functions to call these methods. The definition file, called a Simple Method Description (SMD) file, is a simple JSON string that defines a URL to process the RPC requests. It also processes any methods available at that URL, and the parameters those methods take. The Feature Pack for SCA provides built-in support for Dojo's RPC. It generates the SMD file. You then just have to initialize an RPC client object and then all of these remote methods are available for you to use as normal.

The SCA services which use <binding.http> with <wireFormat.jsonrpc> will by default support the SMD protocol. SMD is similar to ?wsdl for Web services, entering a service endpoint appended with ?smd will return a SMD descriptor for the service.

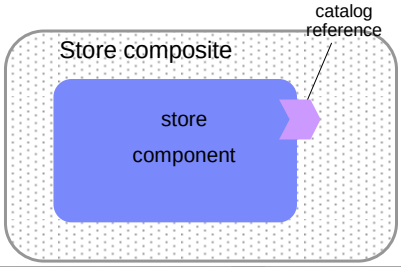
Using SCA services with Dojo can therefore be as simple as:

var myService = new dojo.rpc.JsonService("myService?smd");

The methods available can then be easily shown (for instance the get() method) and then the services can then be invoked using the Dojo client library: myService.get(args);

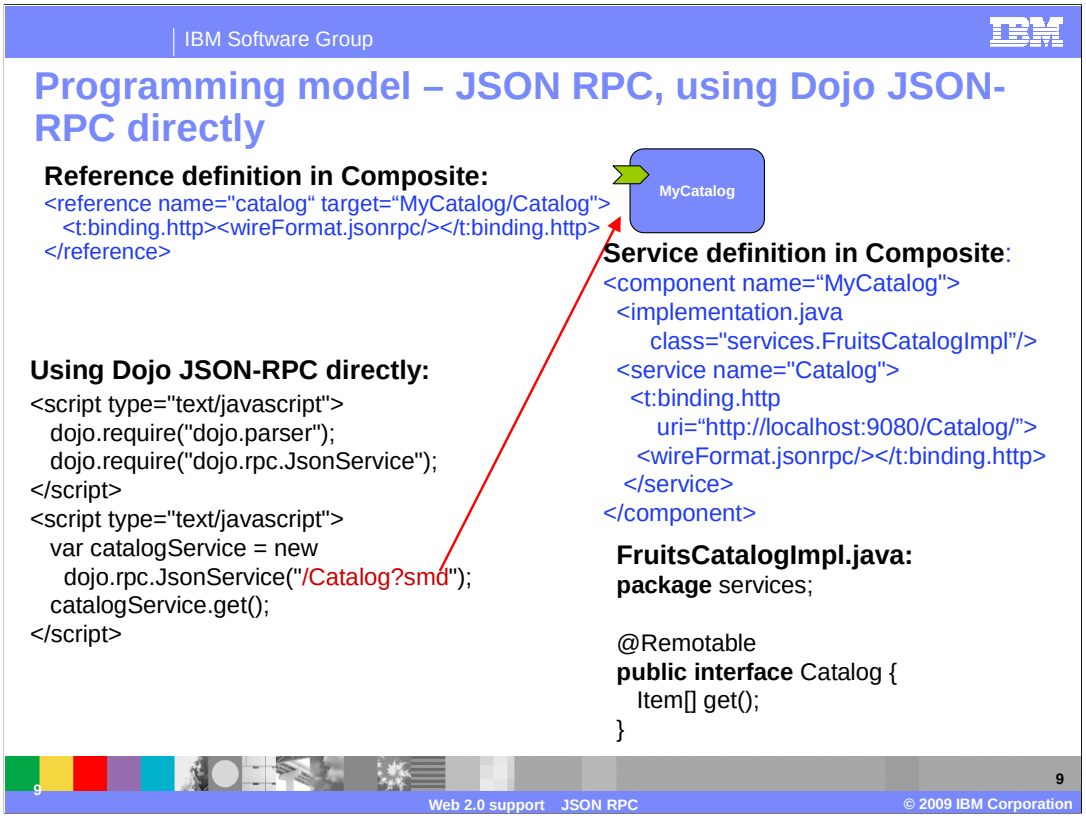
JSON-RPC style access

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://store" name="store">
  <component name="store">
    <reference name="catalog" target="MyCatalog/Catalog">
      <t:binding.http><wireFormat.jsonrpc/></t:binding.http>
    </reference>
  </component> .....
</composite>
```



© 2009 IBM Corporation

The above sample composite shows a component named store that has a reference named catalog. This reference has an HTTP binding with wire format JSON-RPC defined.



The code above depicts a more complete picture of how the catalog reference can be wired through an HTTP binding with wire format JSON-RPC to server side services using SCA JSON-RPC services with Dojo. You have starting in the left corner, a composite file that has a reference named catalog defined with an HTTP binding with wire format JSON-RPC. This reference targets an existing service called Catalog. That service is defined to the top right in a composite file with a component name of MyCatalog. This component contains the service named Catalog that is defined with an HTTP binding with wire format JSON-RPC. This component defines its Java implementation to be FruitsCatalogImpl.java. Notice on the bottom right the FruitsCatalogImpl.java file contains a get() method. Now lastly, you will notice the on the lower left side the using Dojo JSON-RPC directly code. You will first have to define the proper Dojo requires. Then you can define your catalog service using SCA services with Dojo by entering a service endpoint appended with ?smd to return a SMD descriptor for the service. You can now call the get() method that was defined in FruitsCatalogImpl.java.

JSON-RPC services – JSON data binding

- In/Out parameters are passed in JSON format
  - { string : value }
  - { "method" : "get" }
  - { "result" : "12345" }
- JSON data binding is used to transform JSON input arguments into Java types, and to transform Java results into JSON

© 2009 IBM Corporation

Now that you did the HTTP binding with wire format JSON-RPC you can use JSON formats. The parameters are passed in JSON format name/value pairs. JSON data binding is used to transform JSON input arguments into Java types, and to transform Java results into JSON.

Section

Summary and references

© 2009 IBM Corporation

Next is summary of what you have learned and some references.

Summary

- SCA feature pack supports JSON-RPC as a protocol for use with SCA services
  - using the <binding.http> with <wireFormat.jsonrpc>
  - enables remote Web browser clients to easily make RPC style calls to server-side SCA components

© 2009 IBM Corporation

The Feature Pack for SCA supports JSON-RPC as a protocol for use with SCA services by using the <binding.http> with <wireFormat.jsonrpc>. This enables remote Web browser clients to easily make RPC style calls to server-side SCA components.

References

- JSON: http://www.json.org
- Dojo Toolkit: http://dojotoolkit.org
- OpenAjax Alliance: http://openajax.org
- Ajax Technical library: http://www.ibm.com/developerworks/views/web/libraryview.jsp?search_by=Mastering+Ajax
- IBM education assistant: Feature pack for Web 2.0

http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wasfpweb20/plugin_coverpage.html

- The WebSphere® Application Server Feature Pack for Web 2.0 service page:

http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/web20/

- WebSphere Application Server Feature Pack for SCA service page:

http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/sca/

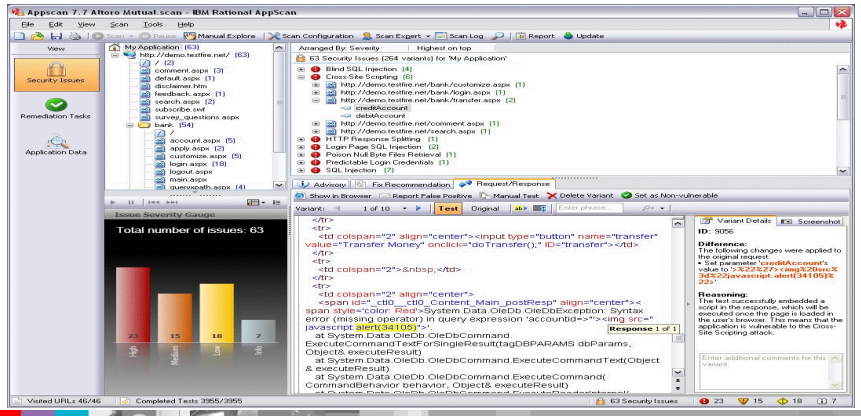
- Apache Tuscany: http://tuscany.apache.org/
  - Notice this presentation contains information from Apache Tuscany. You can find the license information here: http://www.apache.org/licenses/
- This article talks about authorization policy: http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsec_authsoa_policy.html

© 2009 IBM Corporation

Above are some useful references.

Web 2.0 vulnerability testing

- Explicit Security vulnerability testing is critical when working with JavaScript/Web2.0
- Rational® AppScan® (www.watchfire.com) is the industry-leading security vulnerability scanning tool for Ajax



© 2009 IBM Corporation

Outside-in security testing should be part of your quality assurance plan, watchfire.com is a security vulnerability testing tool.

Dojo Toolkit support

- For help with developing Ajax application look at the General Debugging Tools section
- The best information and help is on the www.dojotoolkit.org site especially the forums. Look through the forums for issues and post non-confidential issues on the forum.
- http://www.dojotoolkit.org/support
  - Frequently Asked Questions
  - Forums
- http://www.dojotoolkit.org/docs
  - The Book of Dojo 1.0
  - Dojo API Reference
  - Dojo Porting Guides

© 2009 IBM Corporation

Here are some useful links for Dojo Toolkit.

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_WASV7SCA101_Web20_JSON.ppt

This module is also available in PDF format at: ../WASV7SCA101_Web20_JSON.pdf

© 2009 IBM Corporation

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:

AppScan Rational 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

Rational is a trademark of International Business Machines Corporation and Rational Software 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.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.