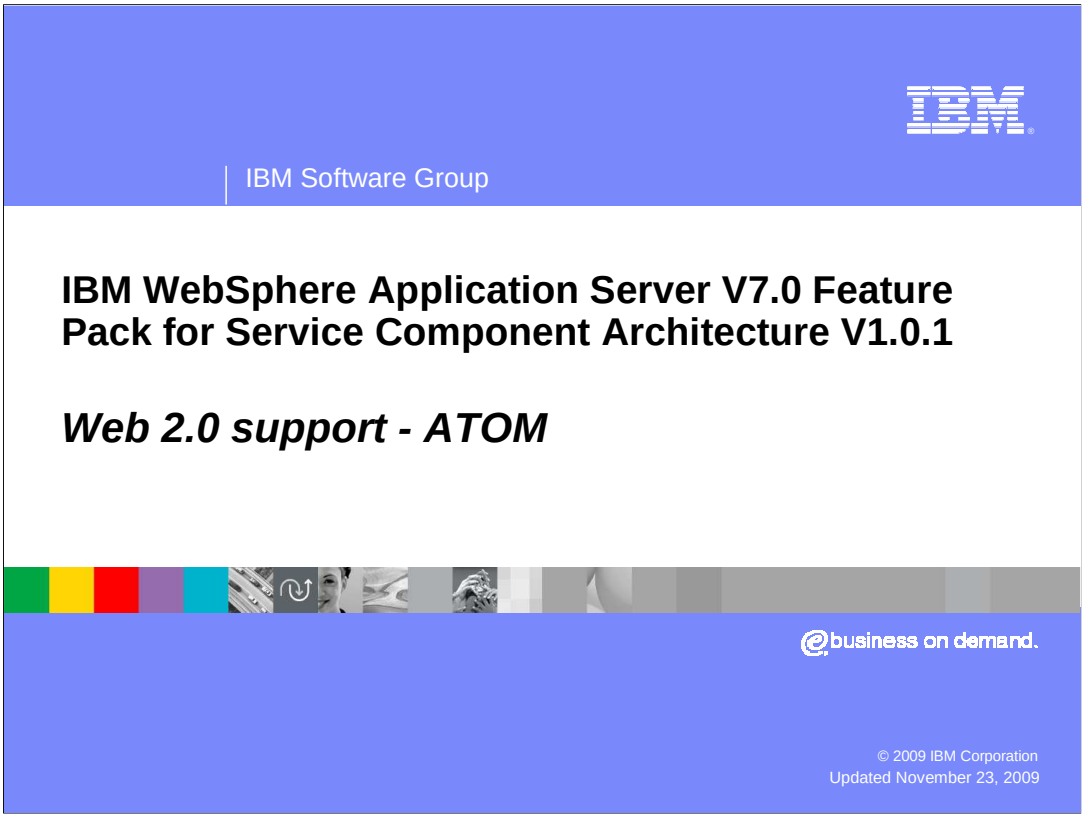
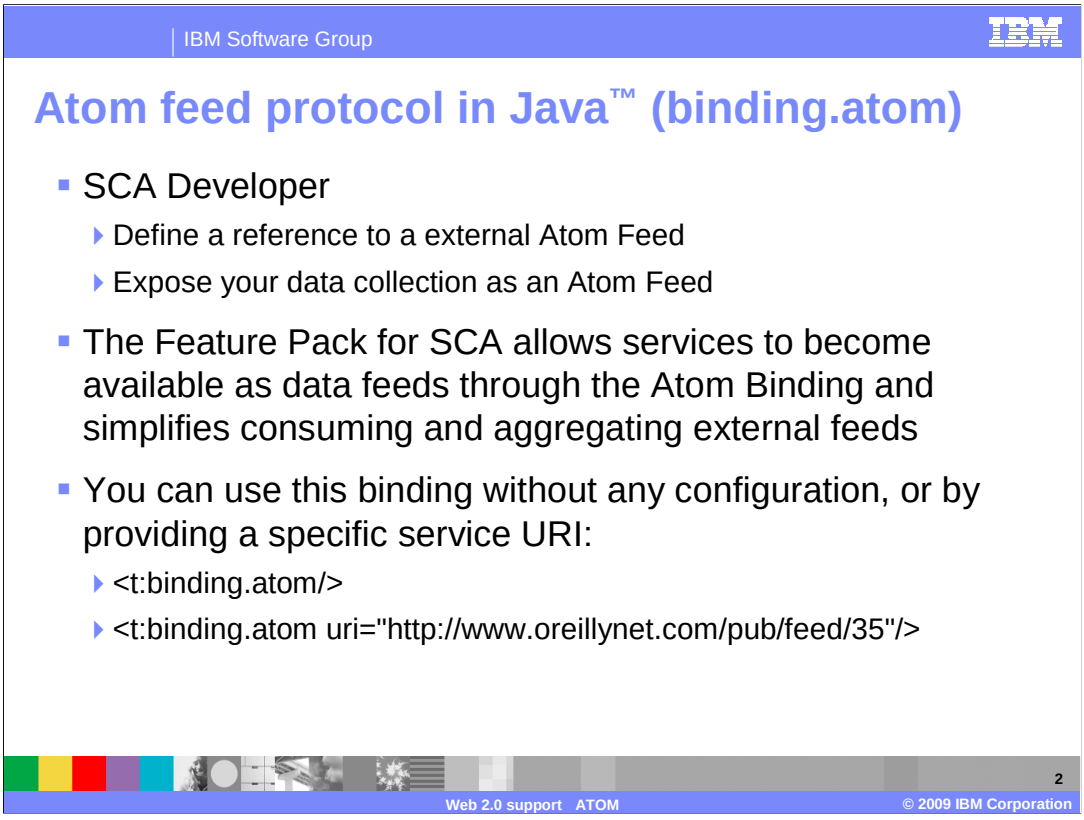


This presentation will talk about Web 2.0 support within the ATOM



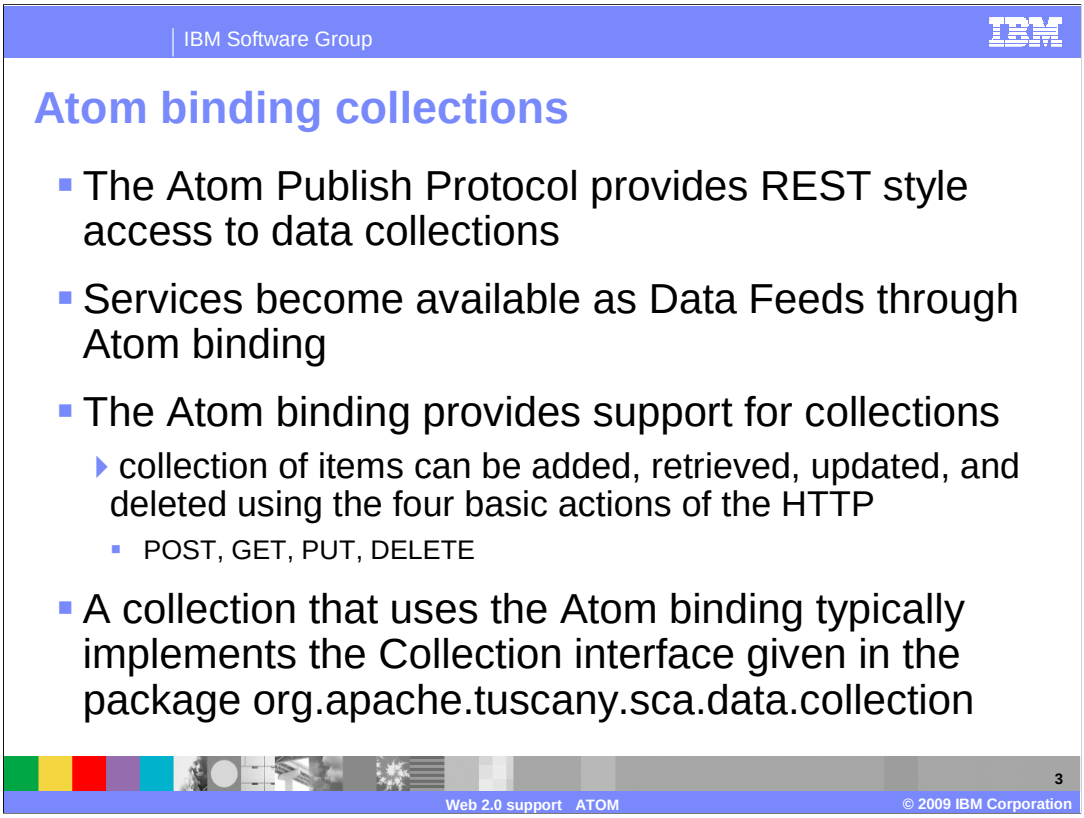
As an SCA Developer, you want to be able to define a reference to a external Atom Feed. As an SCA Developer, you want to expose your data collection as an Atom Feed. The Feature Pack for SCA allows services to become available as data feeds through the Atom Binding and simplifies consuming and aggregating external feeds. You will use the <binding.atom> extension. With the Feature Pack for SCA you can communicate with services that provide or consume items described in the Atom Syndication Format and Atom Publishing Protocol.

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

<t:binding.atom/>

<t:binding.atom uri="http://www.oreillynet.com/pub/feed/35"/>.

To access the feed directly, use the uniform resource indicator (URI).



With the Feature Pack for SCA you can communicate with services that provide or consume items described in the Atom Syndication Format and Atom Publishing Protocol. The Atom Publish Protocol provides REST style access to data collections. The Atom binding provides support for collections. The collection of items can be added, retrieved, updated, and deleted using the four basic actions of the HTTP protocol.

POST (create or add)

GET (retrieve or query)

PUT (update)

DELETE (destroy or remove)

A collection that uses the Atom binding typically implements the Collection interface given in the package org.apache.tuscany.sca.data.collection. This interface declares the basic access methods mentioned above (post, get, put, and delete). For example, one can view the collection as an Atom Feed, and manipulate items in the Feed as Atom Entries. It is up to you to provide the code that implements the Collection interface. You also need to provide the code that translates from the business objects to the Atom model objects Feed and Entry.

As an SCA Developer, you want to expose your data collection as an Atom Feed. The Feature Pack for SCA allows services to become available as data feeds through Atom Binding and simplifies consuming and aggregating external feeds. The Java coding that needs to be created to manipulate feeds, is minimal.

Consuming Atom services on the client application

- Any Atom client can be used to access the SCA services which use the <binding.atom>
- Ways to consume Atom services on the client application
  - Utilizing implementation.widget
  - Using SCA Atom services with Dojo
  - Directly from a browser or other feed client

Any Atom client can be used to access the SCA services which use the <binding.atom>. There are three ways to consume Atom services on the client application. One way is to use the implementation.widget which is described in the overview. The other way is to use SCA Atom services with Dojo. The last way is to just access the Atom services directly from a browser or other feed client. To access the feed directly, you can use the uniform resource indicator (URI) that the service specifies. The next few slides will show you in more detail the two ways to consume Atom services.

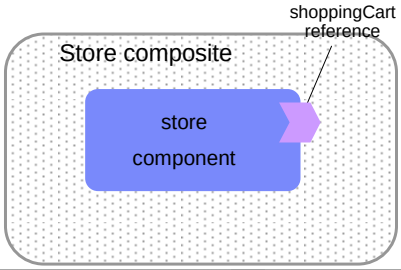
Utilizing implementation.widget for Atom

- Widget Implementation (implementation.widget):
  - Extends the SCA programming model to HTML or Web 2.0 client applications
  - Atom allows you to model your HTML or Web 2.0 as an SCA component
    - HTML resource that represents the application

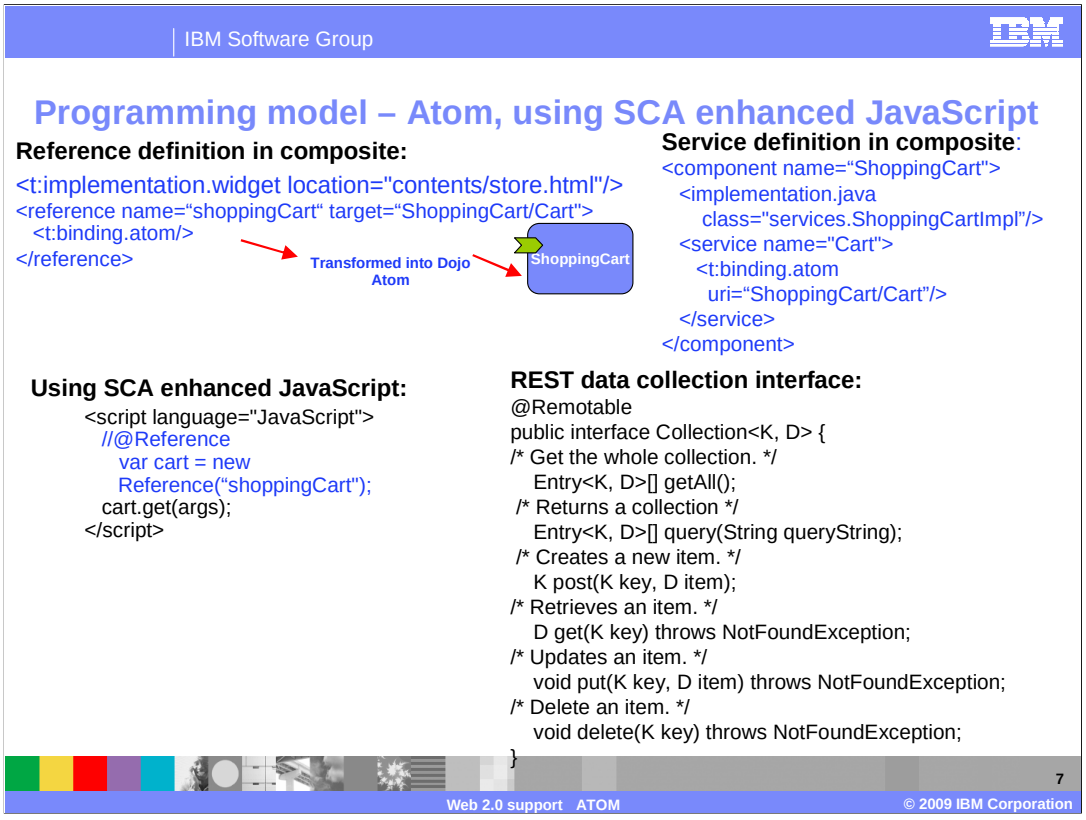
As mentioned earlier the Widget Implementation, implementation.widget, extends the SCA programming model to HTML or Web 2.0 client applications. Using this for Atom allows you to model your HTML or Web 2.0 as an SCA component. The implementation artifact can be HTML resource that represents the application, and you can then define an Atom reference, that can be wired to server side services.

Atom style access in widget (binding.atom)

```
<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="shoppingCart"
        target="ShoppingCart/Cart">
        <t:binding.atom/>
      </reference>
  </component> .....
</composite>
```



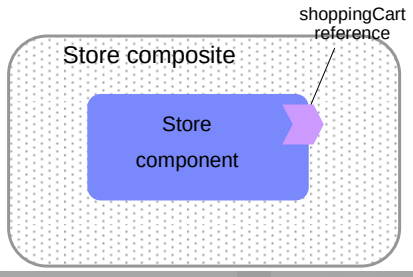
The code above shows a sample composite file. The component store contains the implementation.widget. The implementation.widget component type allows you to model your HTML as an SCA component. The implementation artifact is an HTML resource, and above you can see how you can then define SCA References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



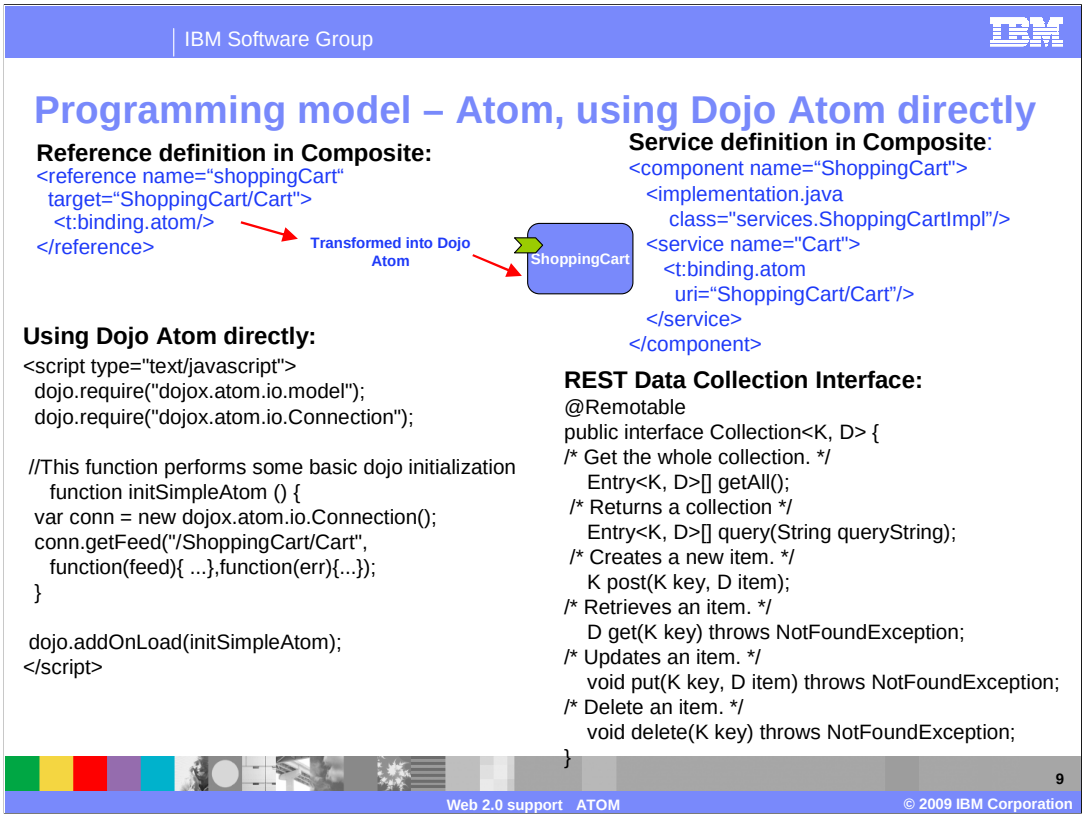
The code above depicts a more complete picture of how the shoppingCart reference can be wired through Atom binding to server side services using implementation.widget. You have starting in the left corner, a composite file that has a component that uses the implementation widget. A reference named shoppingCart is then defined using a Atom binding. This reference targets an existing service called Cart. That service is defined to the top right in a composite file with a component name of ShoppingCart. This component contains the service named Cart that is defined with a Atom binding. This component defines its Java implementation to be ShoppingCartImpl.java. Notice on the bottom right the ShoppingCartImpl.java file that contains REST data collection methods such as getAll(), query(), post(), get(), put(), and delete(). 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 shoppingCart reference and easily now call the get() method that was defined in ShoppingCartImpl.java.

Atom binding (binding.atom)

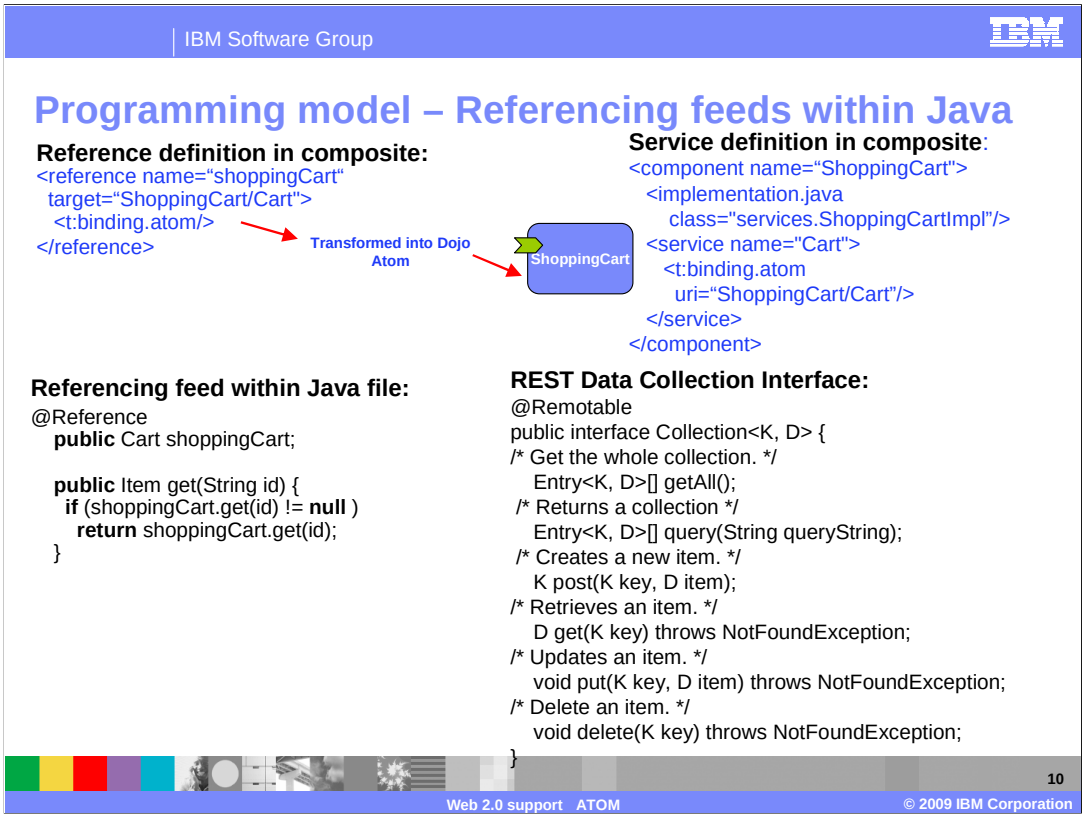
```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://store" name="store">
  <component name="store">
      <reference name="shoppingCart"
        target="ShoppingCart/Cart">
        <t:binding.atom/>
      </reference>
  </component> .....
</composite>
```



Now, you will see an example using SCA Atom services with Dojo. The above sample composite shows a component named store that has a reference named shoppingCart. This reference has a Atom binding defined. There is no Widget implementation used.



The code above depicts a more complete picture of how the shoppingCart reference can be wired through Atom binding to server side services using SCA Atom services with Dojo. You have starting in the left corner, a composite file that has a reference named shoppingCart defined with a Atom binding. This reference targets an existing service called Cart. That service is defined to the top right in a composite file with a component name of ShoppingCart. This component contains the service named Cart that is defined with a Atom binding. This component defines its Java implementation to be ShoppingCartImpl.java. Notice on the bottom right the ShoppingCartImpl.java file that contains REST data collection methods such as getAll(), query(), post(), get(), put(), and delete(). Now lastly, you will notice the on the lower left side the using Dojo Atom directly code. You will first have to define the proper Dojo requires. Then using the proper Dojo code you can get your feed utilizing the get() method that was defined in ShoppingCartImpl.java.



The code above depicts a more complete picture of how the shoppingCart reference can be wired through Atom binding to server side services using Java. You have starting in the left corner, a composite file that has a reference named shoppingCart defined with a Atom binding. This reference targets an existing service called Cart. That service is defined to the top right in a composite file with a component name of ShoppingCart. This component contains the service named Cart that is defined with a Atom binding. This component defines its Java implementation to be ShoppingCartImpl.java. Notice on the bottom right the ShoppingCartImpl.java file that contains REST data collection methods such as getAll(), query(), post(), get(), put(), and delete(). Now lastly, you will notice the on the lower left side referencing feeds within Java code. You will introduce SCA annotations to define the shoppingCart reference and easily now call the get() method that was defined in ShoppingCartImpl.java.

Section

Summary and references

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

Summary

- The Feature Pack for SCA allows services to become available as data feeds through the Atom Binding and simplifies consuming and aggregating external feeds

The Feature Pack for SCA allows services to become available as data feeds through the Atom Binding and simplifies consuming and aggregating external feeds.

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

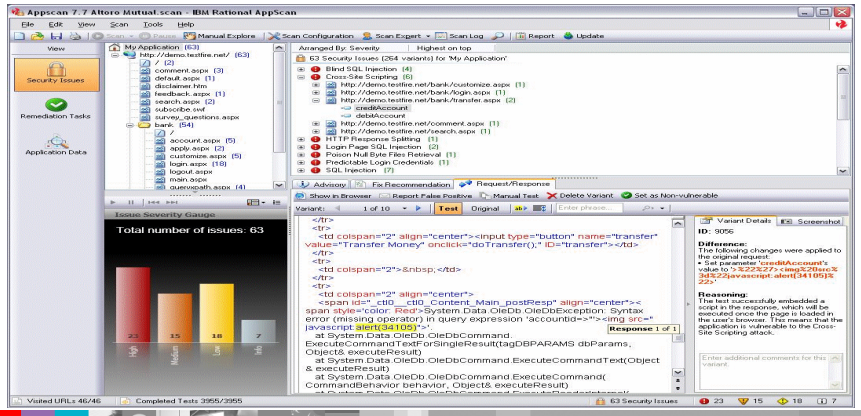
Above are some useful references.

IBM Software Group

Web 2.0 vulnerability testing

watchfire an IBM company

- 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



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

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_ATOM.ppt

This module is also available in PDF format at: ../WASV7SCA101_Web20_ATOM.pdf

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

Trademarks, copyrights, and disclaimers