



IBM Software Group

IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture V1.0.1

Web 2.0 support - ATOM



@business on demand.

© 2009 IBM Corporation
Updated November 23, 2009

This presentation will talk about Web 2.0 support within the ATOM

Atom feed protocol in Java™ (binding.atom)

- SCA Developer
 - ▶ Define a reference to a external Atom Feed
 - ▶ 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 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"/>`



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).

Atom binding collections

- The Atom Publish Protocol provides REST style access to data collections
- Services become available as Data Feeds through Atom binding
- The Atom binding provides support for collections
 - ▶ collection of items can be added, retrieved, updated, and deleted using the four basic actions of the HTTP
 - POST, GET, PUT, DELETE
- A collection that uses the Atom binding typically implements the Collection interface given in the package `org.apache.tuscany.sca.data.collection`

3

Web 2.0 support ATOM

© 2009 IBM Corporation

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

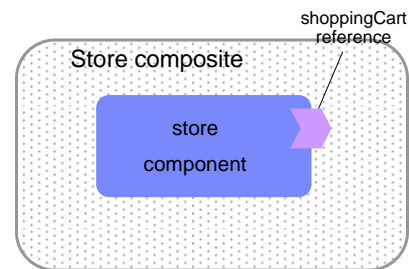
- Widget Implementation (`implementation.widget`):
 - ▶ Extends the SCA programming model to HTML or Web 2.0 client applications
 - ▶ Atom allows you to model your HTML or Web 2.0 as an SCA component
 - HTML resource that represents the application



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.oxa.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.

Programming model – Atom, using SCA enhanced JavaScript

Reference definition in composite:

```
<t:implementation.widget location="contents/store.html"/>
<reference name="shoppingCart" target="ShoppingCart/Car">
  <t:binding.atom/>
</reference>
```

Transformed into Dojo
Atom

ShoppingCart

Service definition in composite:

```
<component name="ShoppingCart">
  <implementation.java
    class="services.ShoppingCartImpl"/>
  <service name="Car">
    <t:binding.atom
      uri="ShoppingCart/Car"/>
  </service>
</component>
```

Using SCA enhanced JavaScript:

```
<script language="JavaScript">
  //@Reference
  var cart = new
  Reference("shoppingCart");
  cart.get(args);
</script>
```

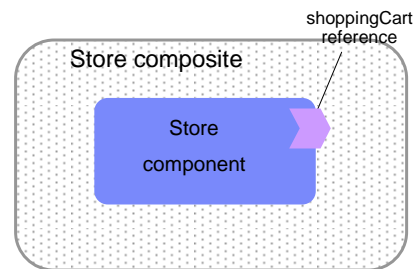
REST data collection interface:

```
@Remotable
public interface Collection<K, D> {
  /* Get the whole collection. */
  Entry<K, D>[] getAll();
  /* Returns a collection */
  Entry<K, D>[] query(String queryString);
  /* Creates a new item. */
  K post(K key, D item);
  /* Retrieves an item. */
  D get(K key) throws NotFoundException;
  /* Updates an item. */
  void put(K key, D item) throws NotFoundException;
  /* Delete an item. */
  void delete(K key) throws NotFoundException;
}
```

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 Car. That service is defined to the top right in a composite file with a component name of ShoppingCart. This component contains the service named Car 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.oxa.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.

Programming model – Atom, using Dojo Atom directly

Reference definition in Composite:

```
<reference name="shoppingCart"
  target="ShoppingCart/Car">
  <t.binding.atom/>
</reference>
```

Transformed into Dojo
Atom



Service definition in Composite:

```
<component name="ShoppingCart">
  <implementation.java
    class="services.ShoppingCartImpl"/>
  <service name="Car">
    <t.binding.atom
      uri="ShoppingCart/Car"/>
  </service>
</component>
```

Using Dojo Atom directly:

```
<script type="text/javascript">
  dojo.require("dojox.atom.io.model");
  dojo.require("dojox.atom.io.Connection");

  //This function performs some basic dojo initialization
  function initSimpleAtom () {
    var conn = new dojox.atom.io.Connection();
    conn.getFeed("/ShoppingCart/Car",
      function(feed){ ...},function(err){...});
  }

  dojo.addOnLoad(initSimpleAtom);
</script>
```

REST Data Collection Interface:

```
@Remotable
public interface Collection<K, D> {
  /* Get the whole collection. */
  Entry<K, D>[] getAll();
  /* Returns a collection */
  Entry<K, D>[] query(String queryString);
  /* Creates a new item. */
  K post(K key, D item);
  /* Retrieves an item. */
  D get(K key) throws NotFoundException;
  /* Updates an item. */
  void put(K key, D item) throws NotFoundException;
  /* Delete an item. */
  void delete(K key) throws NotFoundException;
}
```

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 Car. That service is defined to the top right in a composite file with a component name of ShoppingCart. This component contains the service named Car 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.

Programming model – Referencing feeds within Java

Reference definition in composite:

```
<reference name="shoppingCart"
  target="ShoppingCart/Car">
  <t.binding.atom/>
</reference>
```

Transformed into Dojo
Atom



Service definition in composite:

```
<component name="ShoppingCart">
  <implementation.java
    class="services.ShoppingCartImpl"/>
  <service name="Car">
    <t.binding.atom
      uri="ShoppingCart/Car"/>
  </service>
</component>
```

Referencing feed within Java file:

```
@Reference
public Cart shoppingCart;

public Item get(String id) {
  if (shoppingCart.get(id) != null )
    return shoppingCart.get(id);
}
```

REST Data Collection Interface:

```
@Remotable
public interface Collection<K, D> {
  /* Get the whole collection. */
  Entry<K, D>[] getAll();
  /* Returns a collection */
  Entry<K, D>[] query(String queryString);
  /* Creates a new item. */
  K post(K key, D item);
  /* Retrieves an item. */
  D get(K key) throws NotFoundException;
  /* Updates an item. */
  void put(K key, D item) throws NotFoundException;
  /* Delete an item. */
  void delete(K key) throws NotFoundException;
}
```



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

- 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

Web 2.0 support ATOM
© 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



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

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.

