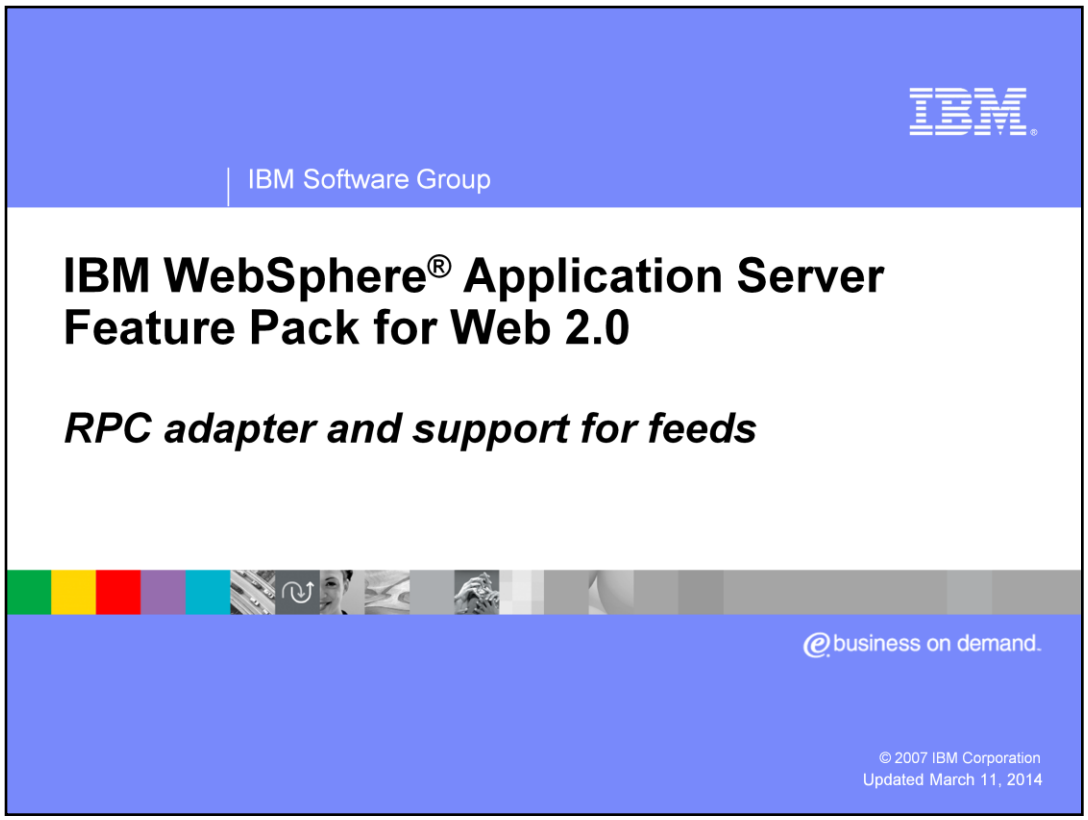
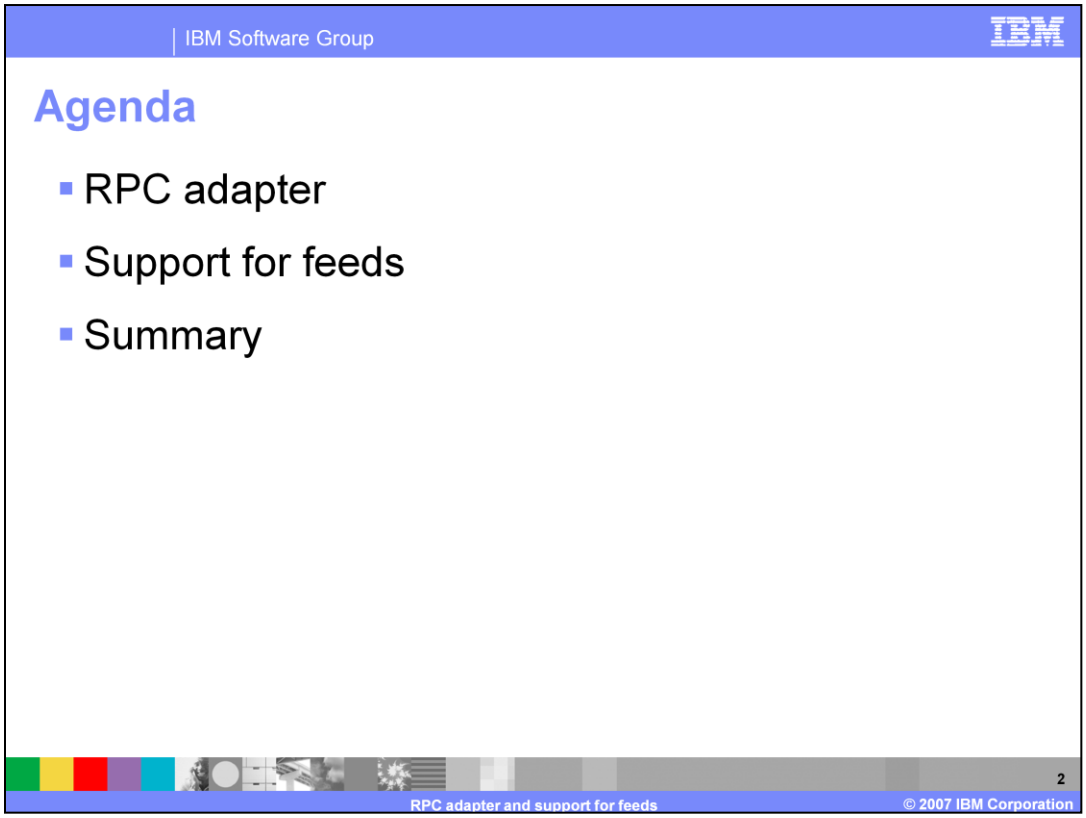
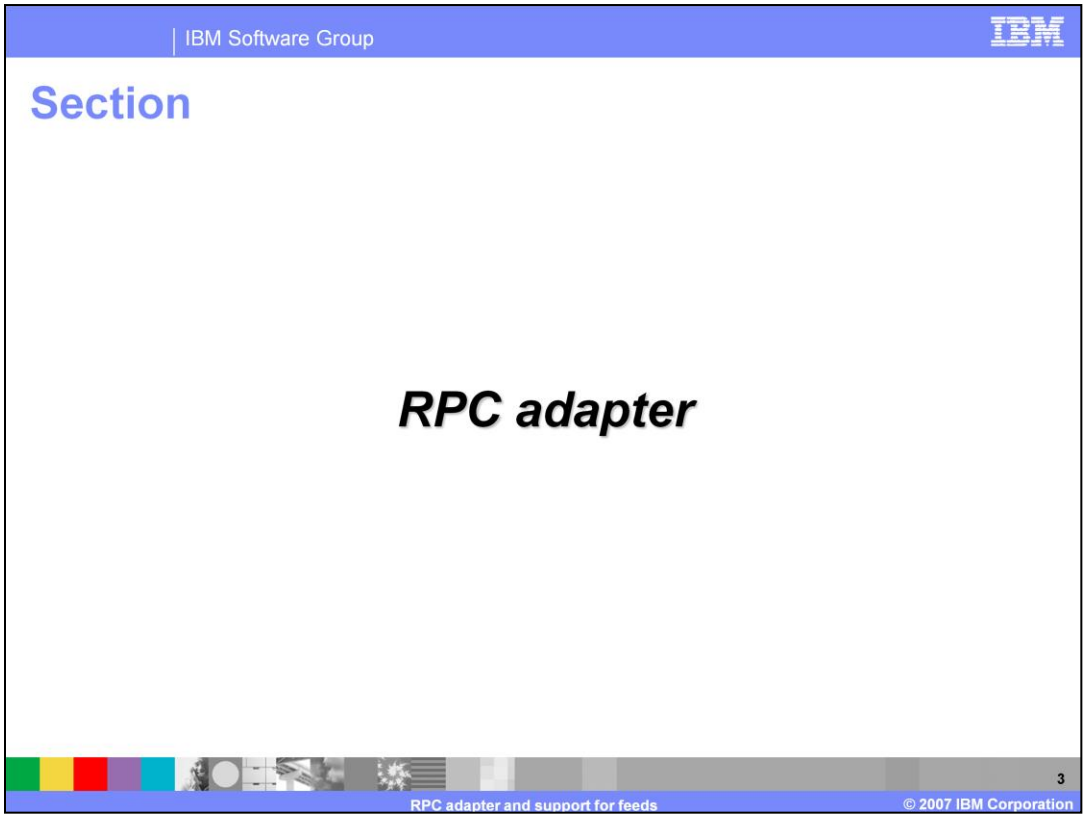


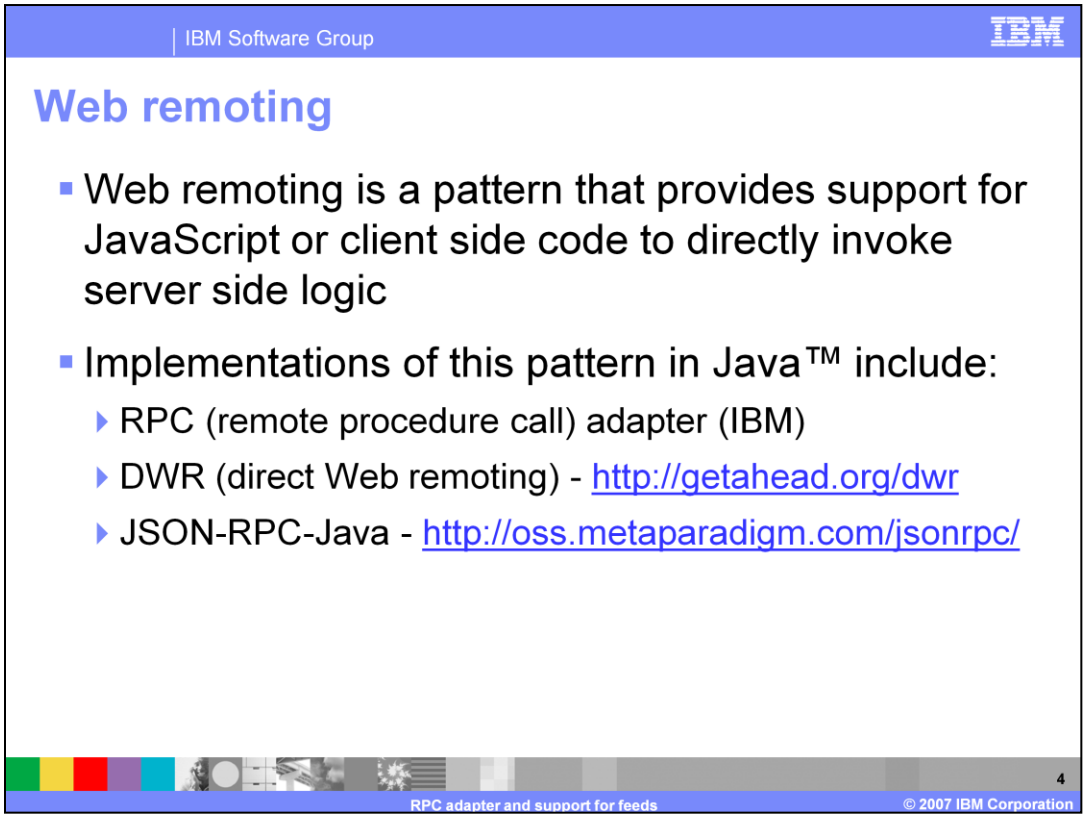
This presentation will cover the RPC adapter and support for feeds as part of the connectivity included in the IBM WebSphere Application Server Feature Pack for Web 2.0.



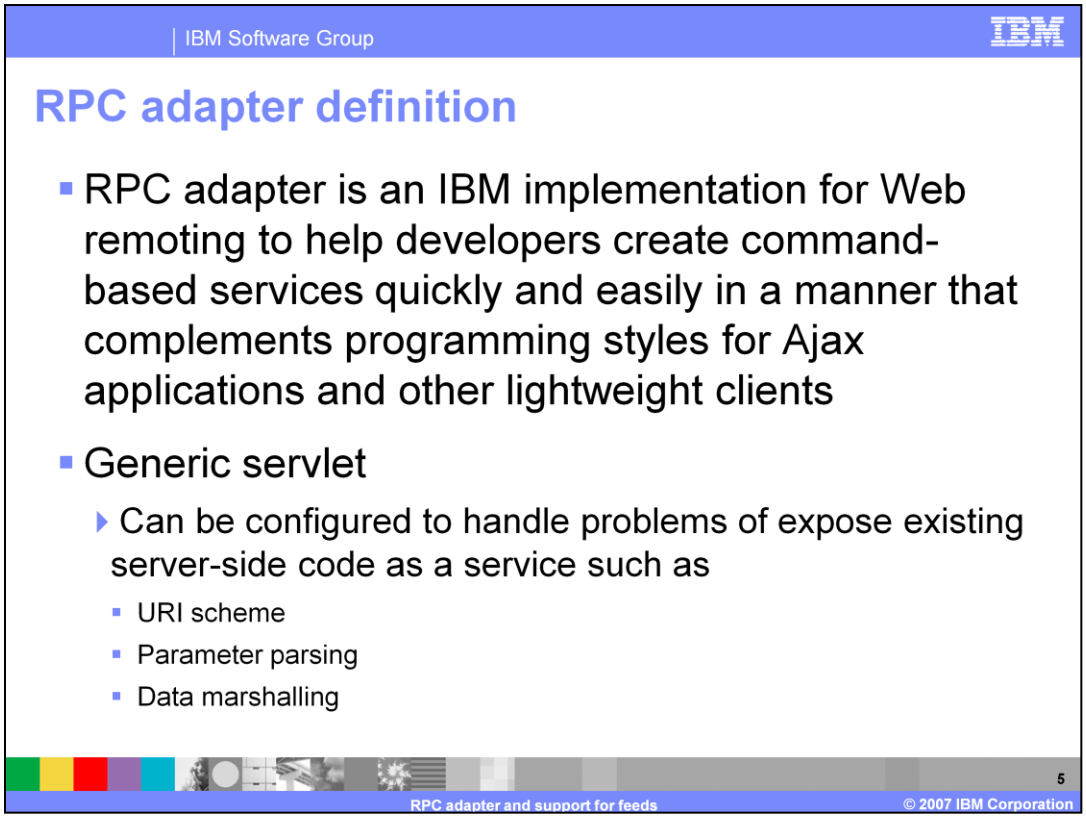
The presentation will touch on the RPC adapter and some of the included sample applications.



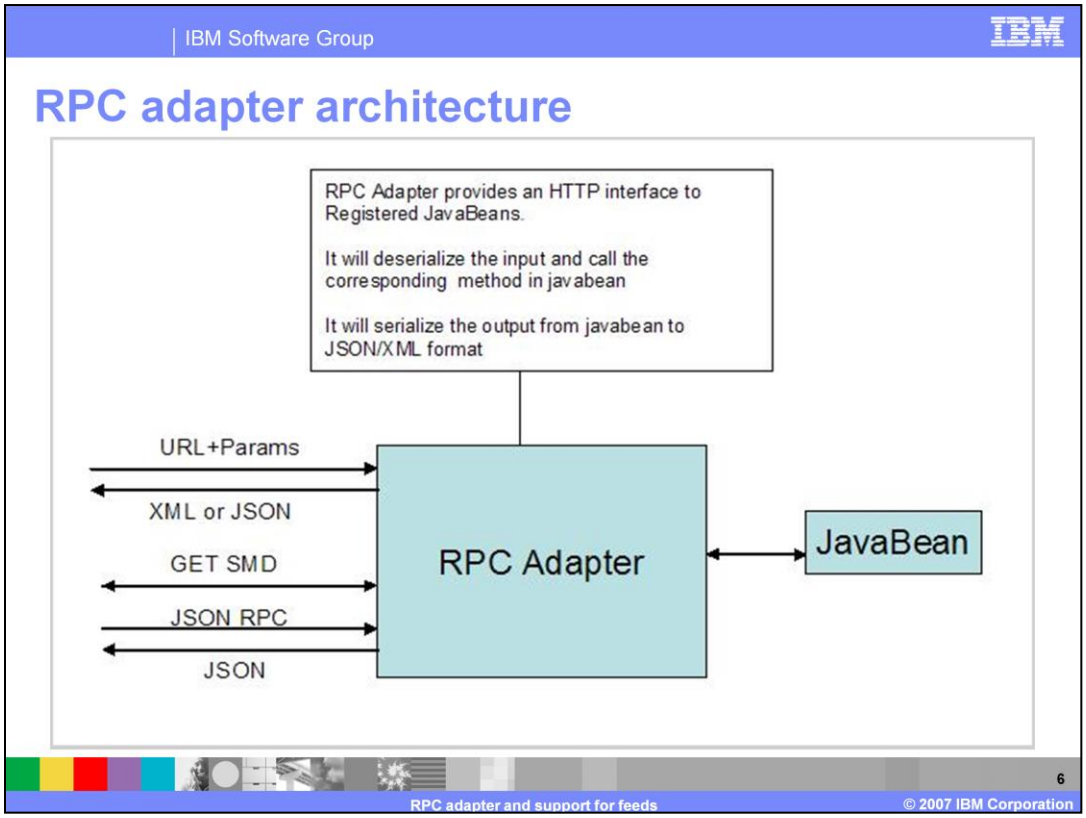
This section covers the RPC adapter



Before you understand the RPC adapter, you will need to know a little bit about Web-remoting. Web-remoting is a pattern that provides support for JavaScript or client side code to directly invoke server side logic. There are a few implementations of this pattern in Java, besides the current IBM one. DWR and the JSON-RPC-Java implementations are examples. DWR allows JavaScript in a browser to interact with Java on a server and helps you manipulate Web pages with the results. JSON-RPC-Java is based on the JSON-RPC specification.



IBM's Web remoting implementation is the RPC-adapter. The RPC adapter is designed to help developers create command-based services quickly and easily in a manner that complements programming styles for Ajax applications and other lightweight clients. Implemented as a generic servlet, the RPC adapter provides an HTTP interface to registered EJBs. The generic servlet can be configured to handle problems of exposing existing server-side code as a service such as URI scheme, parameter parsing and data marshalling.



This picture shows the RPC adapter architecture. As mentioned earlier, the RPC adapter provides an HTTP interface to registered beans. It will de-serialize the input and call the corresponding method in the Java bean. It will also serialize the output from the Java bean to JSON/XML format.

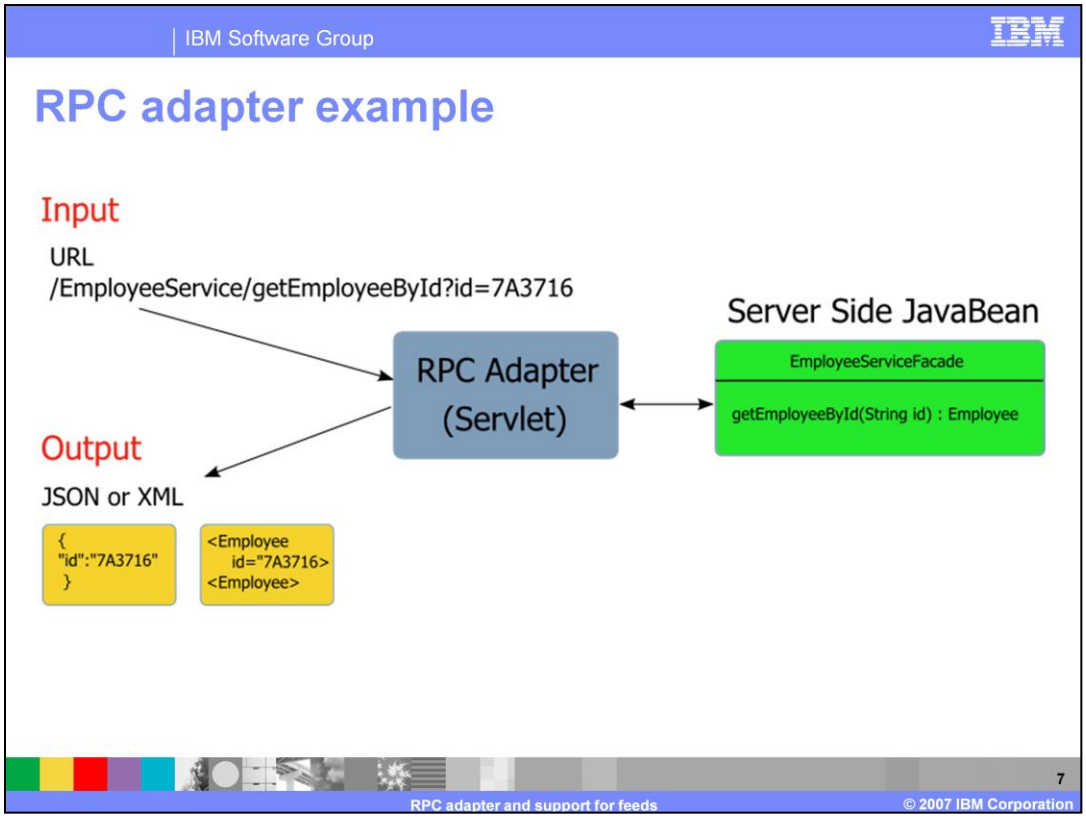
The RPC adapter currently supports two RPC protocols:

One is the "HTTP RPC", which encodes RPC invocations as URLs with query parameters (for HTTP GET) or form parameters (for HTTP POST)

The other is the "JSON-RPC", supporting the SMD service descriptor employed by Dojo's dojo.rpc.JsonService API.

In HTTP RPC, invocations are made using URLs with query parameters or form parameters. The RPC adapter will intercept the URL and deserialize the URL to get service name, method name and input parameters. Using this information the RPC adapter will invoke the corresponding method of the matching Java Bean.

On the other hand, in JSON-RPC, method invocation is made using JSON objects. The response generated will also be a JSON object. The registered Java Bean can be accessed through Dojo's JSON-RPC API.



In this RPC adapter example the input is a URL which is a request for an emplyoyee ID which goes through the RPC Adapter. The adapter exposes getEmployeeById method and the output is an xml or JSON output

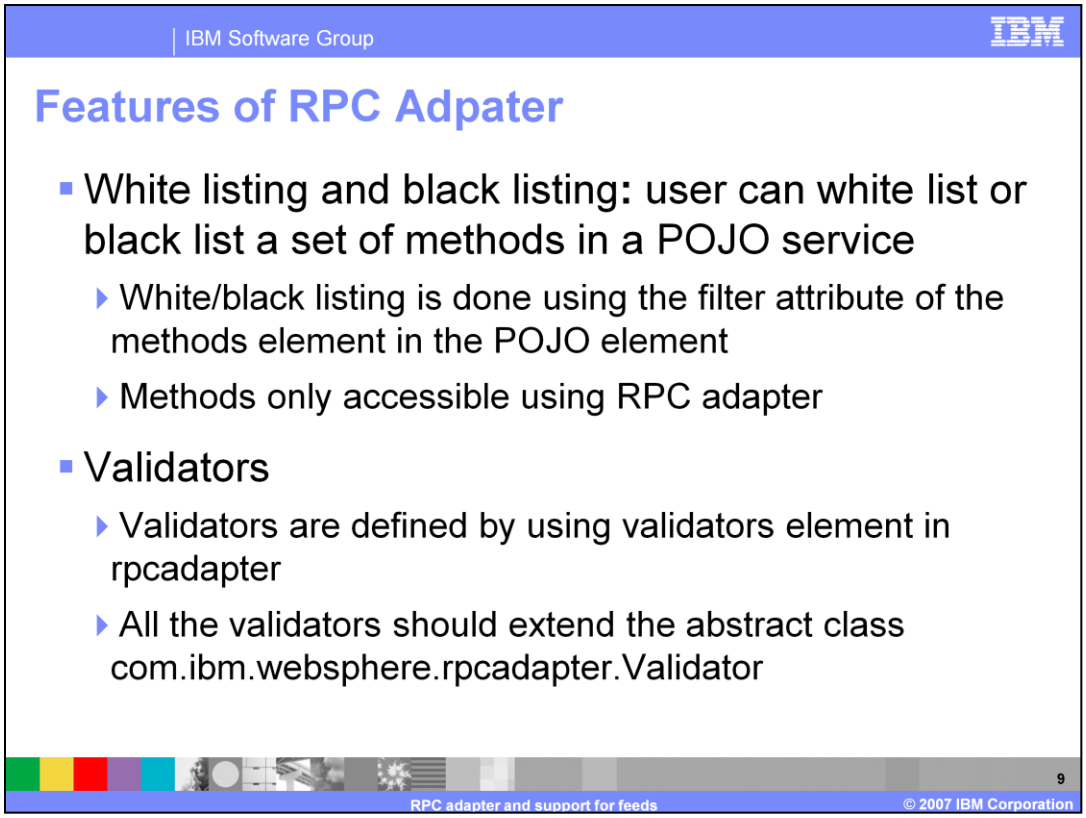
RPC adapter : JSON-RPC method

- In JSON-RPC method invocation is made using JSON objects
- The generated response will be a JSON object
- The registered Java Bean can be accessed through Dojo's JSON-RPC API
- Example

```
var example = new
  dojo.rpc.JsonService("/contextRoot/RPCadapter/jsonrpc/example");
  example.getEcho('Hello world').addCallback(exampleCallback);
function exampleCallback(result) {
// response is available as result.getEchoReturn.String }
```

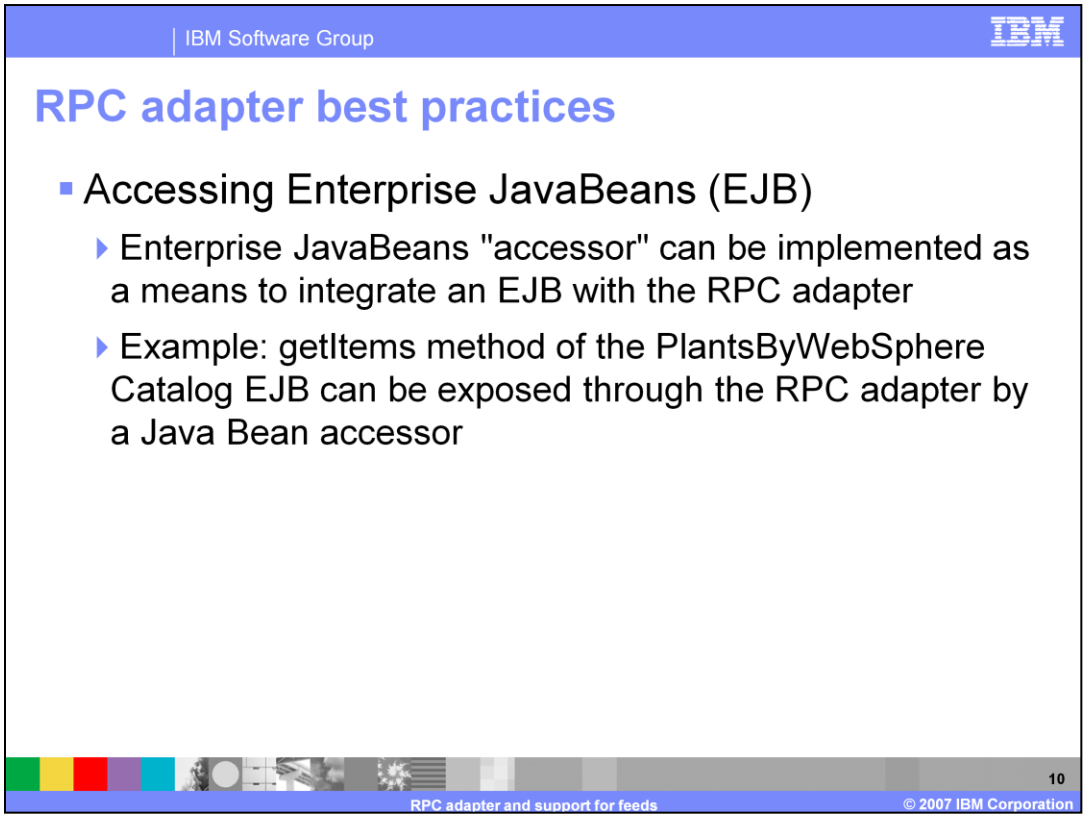

In JSON-RPC, method invocation is made using JSON objects. The generated response will be a JSON object and the registered Java Bean can be accessed through Dojo's JSON-RPC API.

The example shown shows the JSON-RPC method and how easy it is to create JSON objects.



Some of the features for the RPC adapter include white listing, black listing and validators. With White Listing and Black Listing, you can add a set of methods in a POJO service to either list. White or black listing is done using the filter attribute of the methods element in the POJO element. The values of the filter can be white listing and black listing. If the filter is not specified then all the methods are listed. All methods are accessible using the RPC adapter. If the methods element of a POJO service is not specified, or the filter attribute is not specified for the methods element, then by default all methods are white listed.

Validators are defined by using the validators element in the RPC adapter. You can specify a set of validators for individual POJO services. Before the method invocation, validate methods of all specified validators are called on the attributes of the methods which have references to at least one validator. All validators should extend the abstract class com.ibm.websphere.rpcadapter.Validator. Another way of validating is by using regular expressions. The validation-regex element can be used to specify regular expressions that should match the parameter values. If the parameter values are not matching the regular expression, then a validation error is thrown.

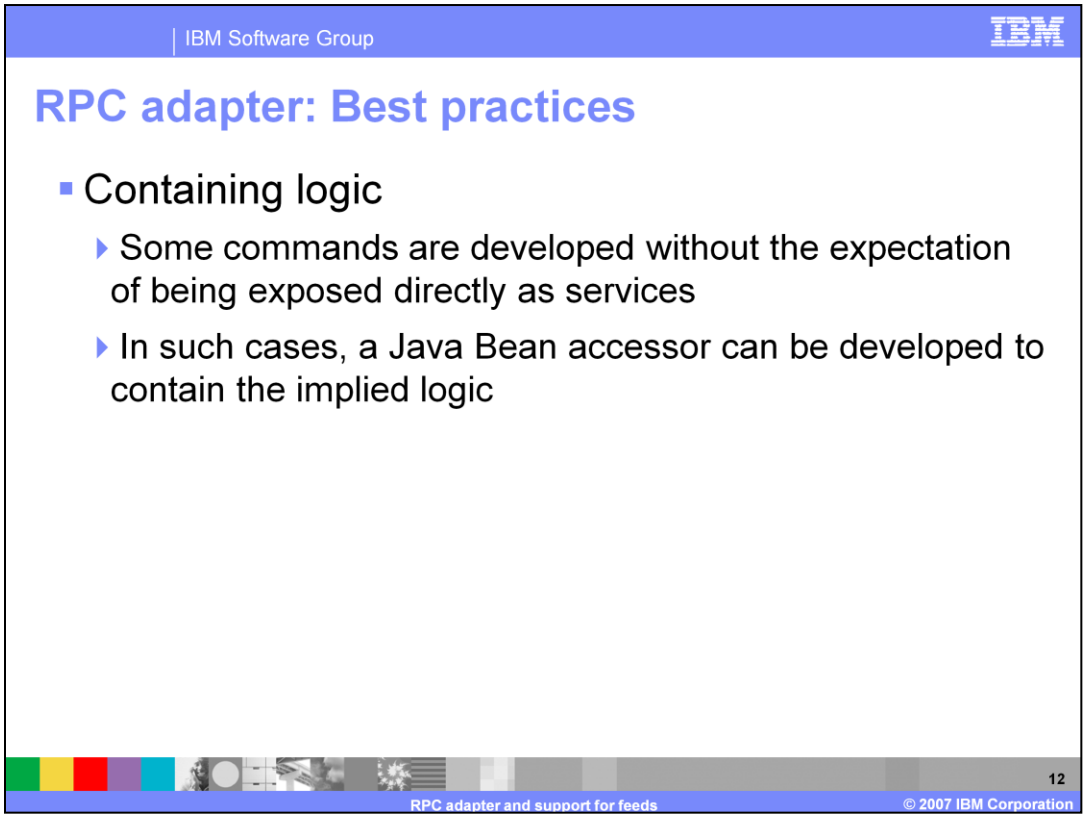


Some of the RPC adapter Best practices include accessing enterprise beans. A Java Bean "accessor" can be implemented as a means to integrate an EJB with the RPC adapter. For example, the getItems method of the PlantsByWebSphere Catalog EJB can be exposed through the RPC adapter by a Java Bean accessor.

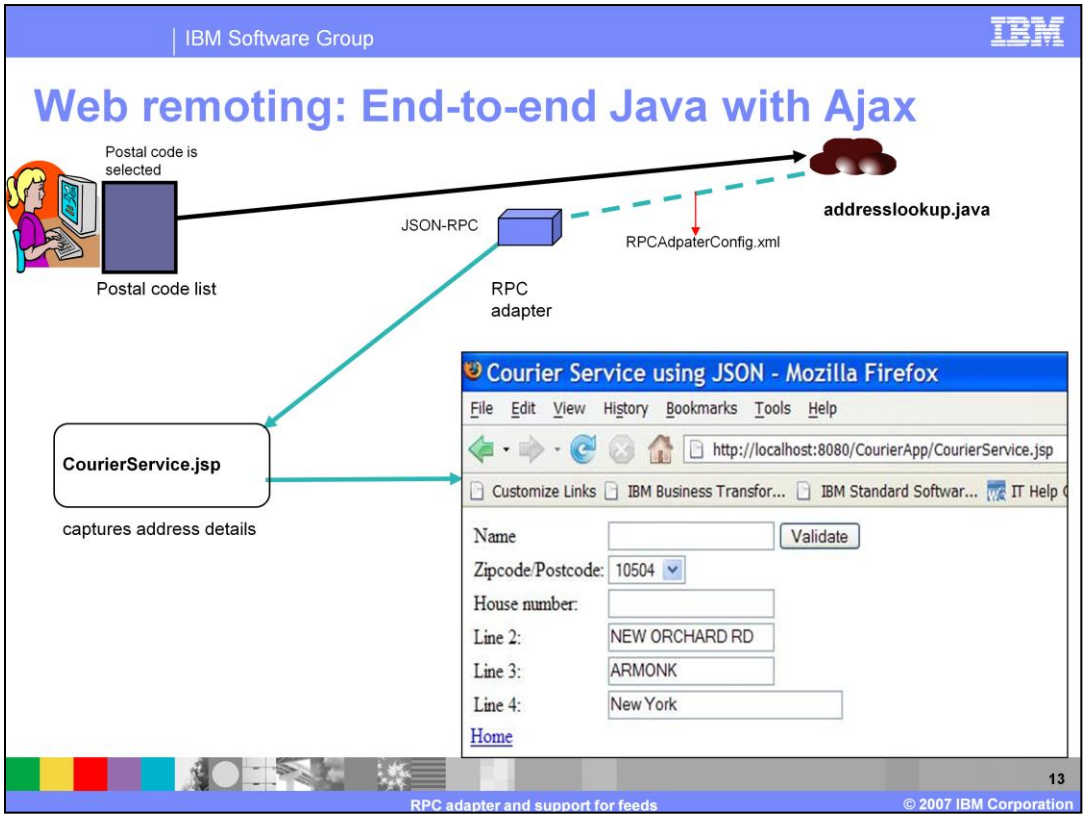
Example

```
public class CatalogAccessor implements Catalog,
  SelfBeanInfo {
private static CatalogHome _home = (CatalogHome)
  Util.getEJBHome("java:comp/env/ejb/Catalog",
  com.ibm.websphere.samples.plantsbyWebsphereejb.Catalo
  gHome.class);
private Catalog _bean;
public static String[][] getBeanDescriptorInfo() {
  String [][] descriptor = { {"method", "getItems",
  "Get all inventory items. Return: Vector of
  StoreItems.", "GET"}, }; return(descriptor); }
public CatalogAccessor() throws RemoteException,
  CreateException { _bean = _home.create(); }
public Vector getItems() throws RemoteException {
return(_bean.getItems()); } }
```

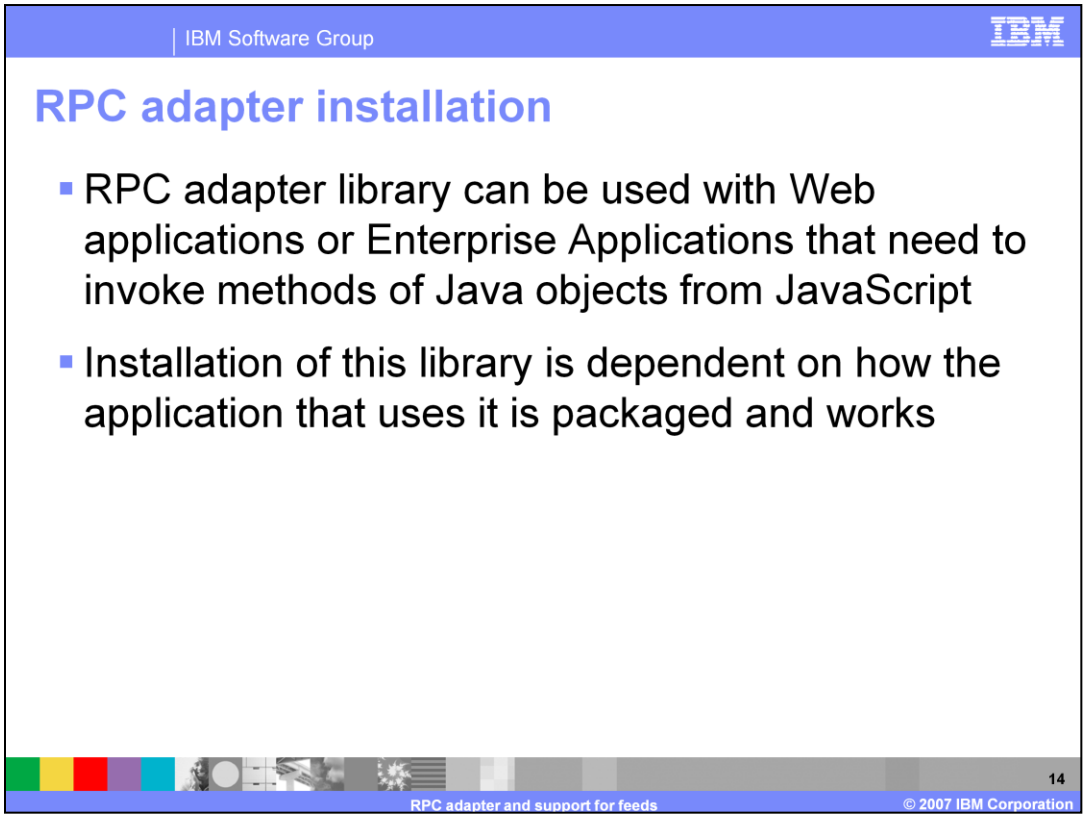

In the given example, note that CatalogHome is cached as a static field so that it can be reused to create new instances. Instances of Catalog are created in the no-arg constructor of the Java Bean, then used in the getItems method.



Another feature of the RPC adapter is that it contains logic in that some commands are developed without the expectation of being exposed directly as services. In such cases, a Java Bean accessor can be developed to contain the implied logic. For example, the ShoppingCart EJB in the PlantsByWebSphere sample includes an addItem(StoreItem item) method. The StoreItem object includes the item price.



This is an example of an end-to-end scenario of Java with Ajax using Web remoting. The sample is a courier application using Web-remoting to auto-populate address details when a postal code is selected. The **addresslookup** Java bean is registered with RPC adapter through **RPCadapterConfig.xml**. CourierService.jsp is a simple jsp page that captures address details. When a postal code is selected from the postalcode list, the page uses the RPC adapter to autopopulate the corresponding address details.

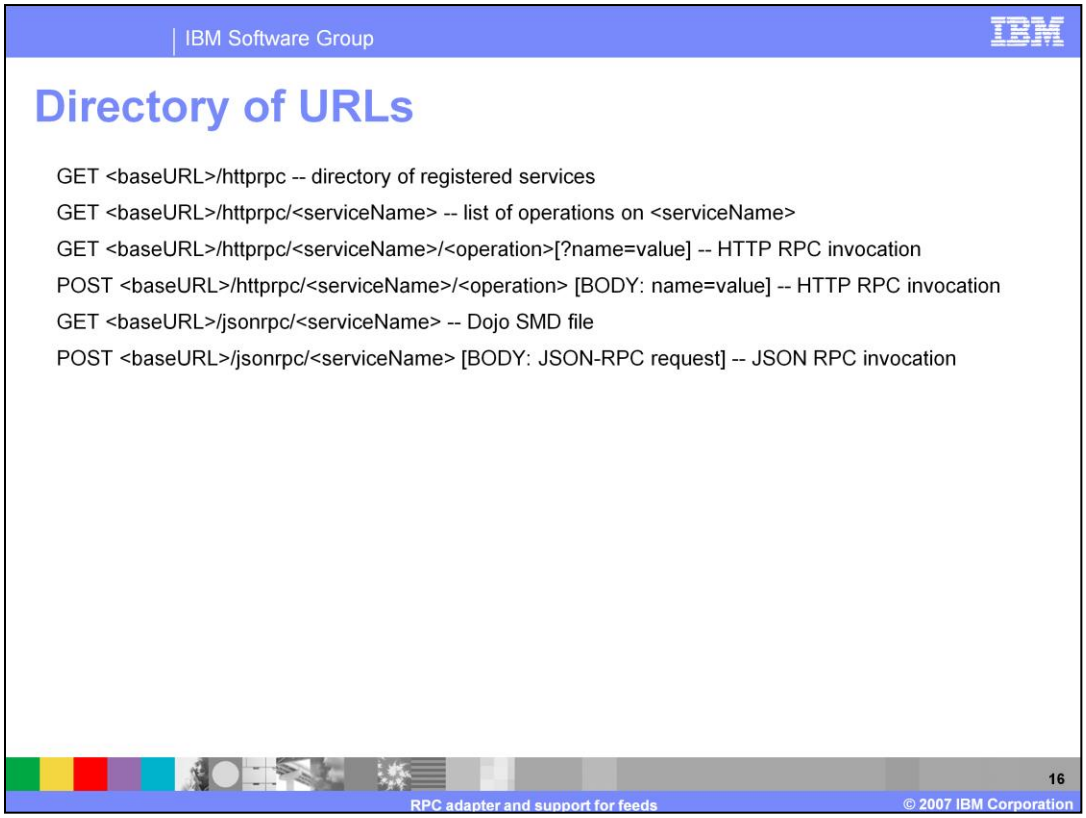


Installation of the RPC adapter library is dependent on how the application that uses it is packaged and works. This stand-alone library can be used with Web applications or Enterprise Applications that need to invoke methods of Java objects from JavaScript.

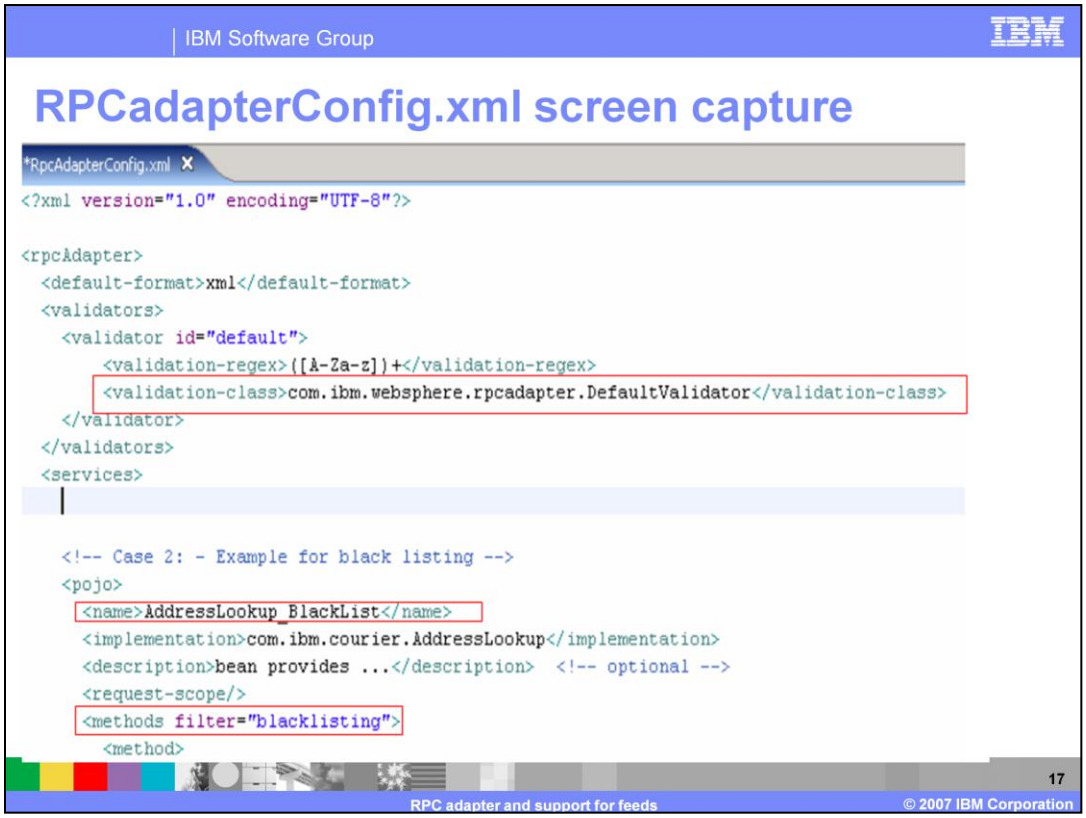
RPC adapter installation: J2EE Web application

- Place RpcadapterConfig.xml in the WEB-INF/lib directory of the Web application containing the Java Beans that you want to make accessible
- Configure Web.xml accordingly
- Start the application server
- Interact with the RPC adapter for information using directory URLs
  - Example: GET <baseURL>/httprpc -- directory of registered services

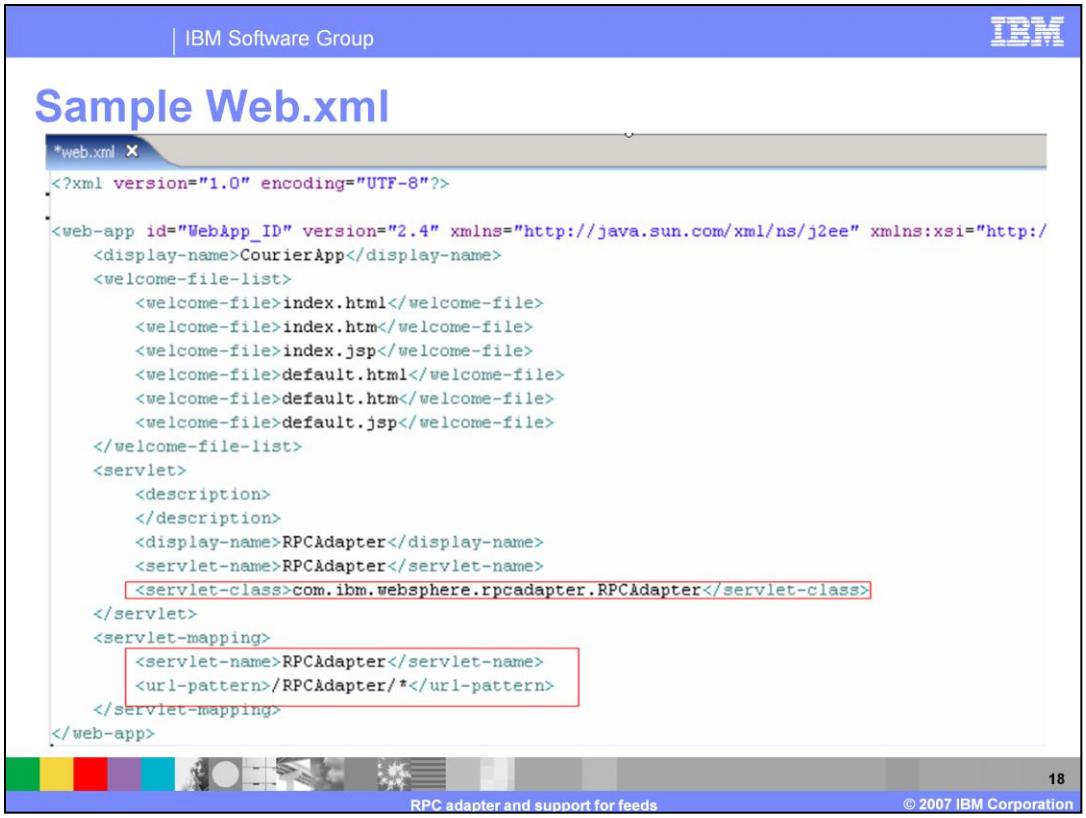
Installing an RPC adapter within a J2EE Web application entails placing RpcadapterConfig.xml in the WEB-INF/lib directory of the Web application containing the Java Beans that you want to make accessible, then configuring Web.xml accordingly and starting the application server. You can then interact with the RPC adapter for information using a directory of URLs as listed on the next slide:



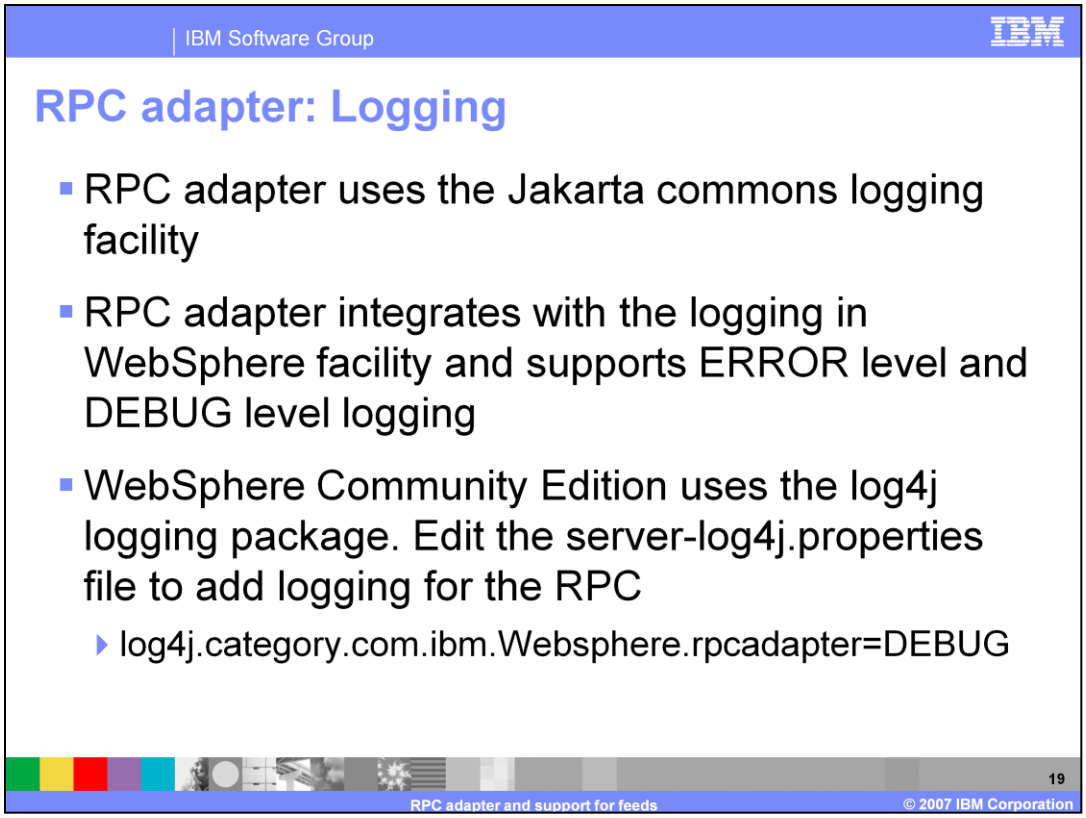
Once the Web.xml has been configured, you can then interact with the RPC adapter using the directory of URLs that is shown here for your reference.



This is an image of RPCadapterConfig.xml. Note the validation class, the POJO name, and the methods filter. This file contains rules for the RPC adapter. Validators are defined by using validators element, and you can specify a set of validators for individual POJO services.



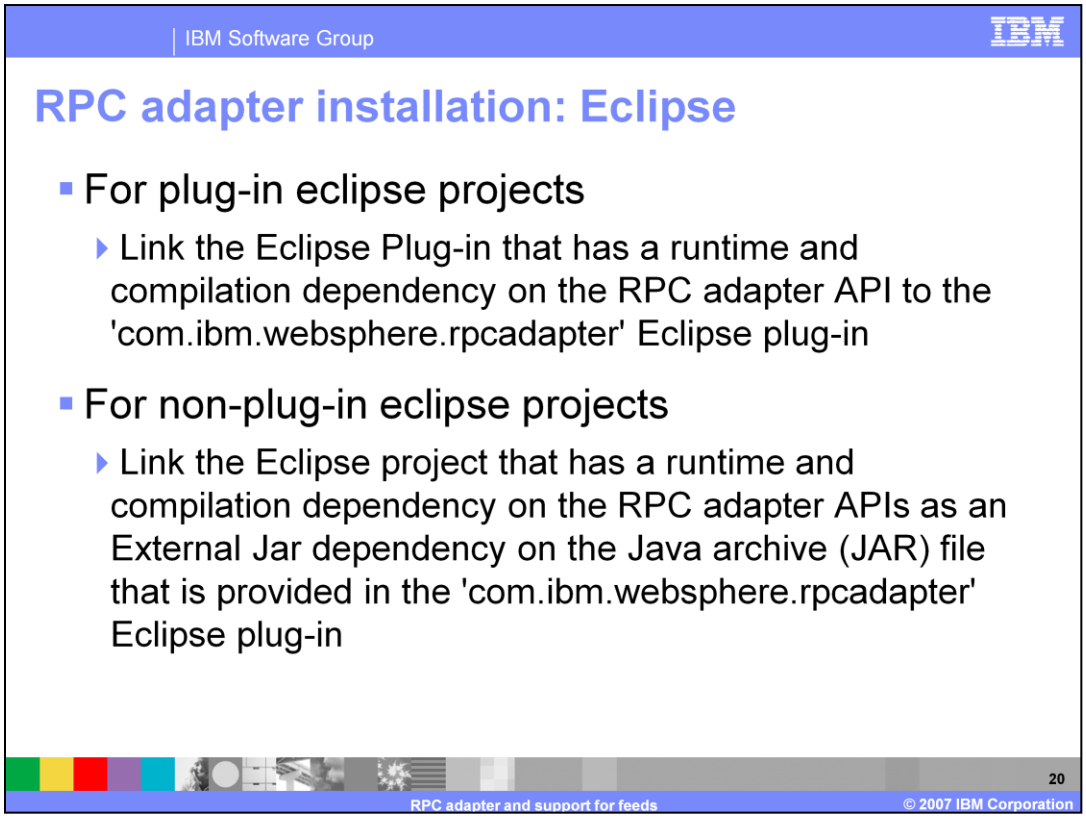
This image is a sample Web.xml deployment descriptor file. It defines several parameters that are used when the Web application is deployed. In this example, note the servlet class name and the URL pattern.



The RPC adapter uses the Jakarta Commons Logging facility. The Jakarta Commons Logging provides a logging framework that can integrate other logging facilities like log4j and java.util.logging. The logging configuration that you use will depend on the application server you are deploying the RPC adapter to.

For WebSphere Application Server version 6.0 and version 6.1, the RPC adapter integrates with the WebSphere logging facility and supports ERROR level and DEBUG level logging. In this scenario, select the com.ibm.websphere.rpcadapter package under the Logging and Tracing panel then select *finest* to enable the DEBUG level logging.

Other Application Servers such as WebSphere Application Server Community Edition use the log4j logging package. To add logging for the RPC adapter, edit the server-log4j.properties file by adding the entry shown here.



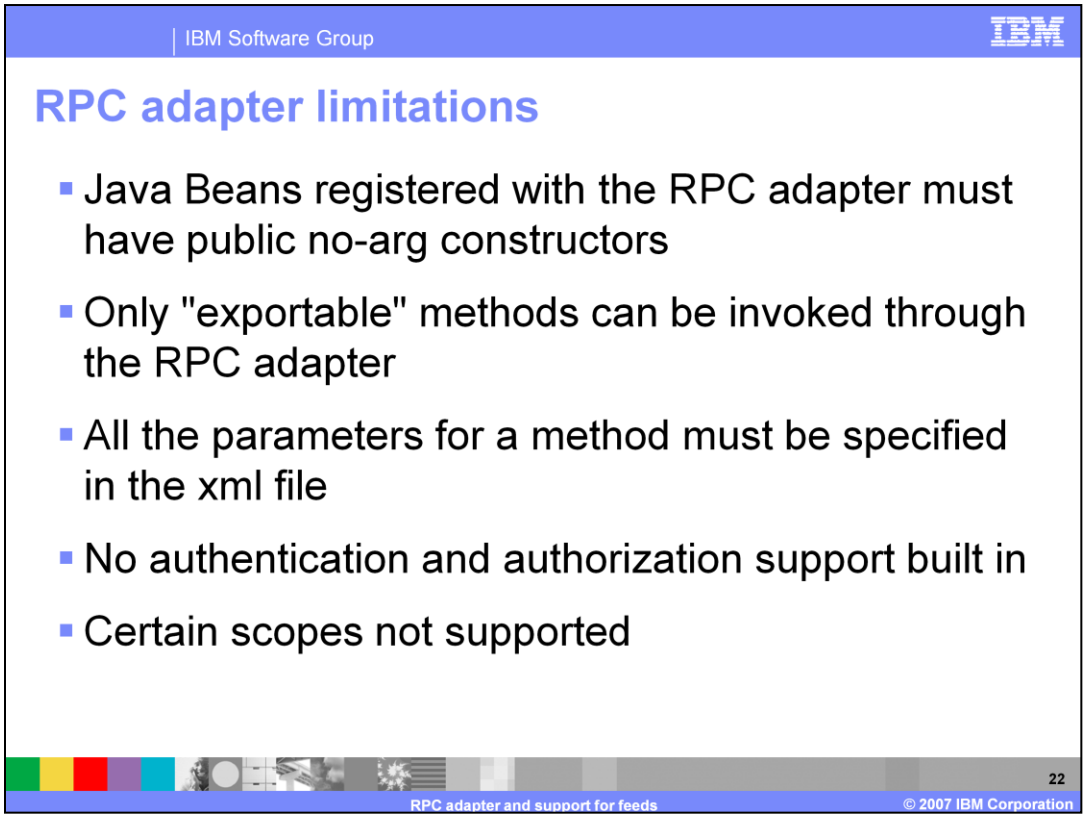
For Eclipse plug-ins, link the Eclipse plug-in that has a runtime and compilation dependency on the RPC adapter API to the 'com.ibm.websphere.rpcadapter' Eclipse plug-in.

For Eclipse non-plug-in projects, link the Eclipse project that has a runtime and compilation dependency on the RPC adapter APIs as an External Jar dependency on the JAR file. The JAR file is provided in the 'com.ibm.websphere.rpcadapter' Eclipse plug-in.

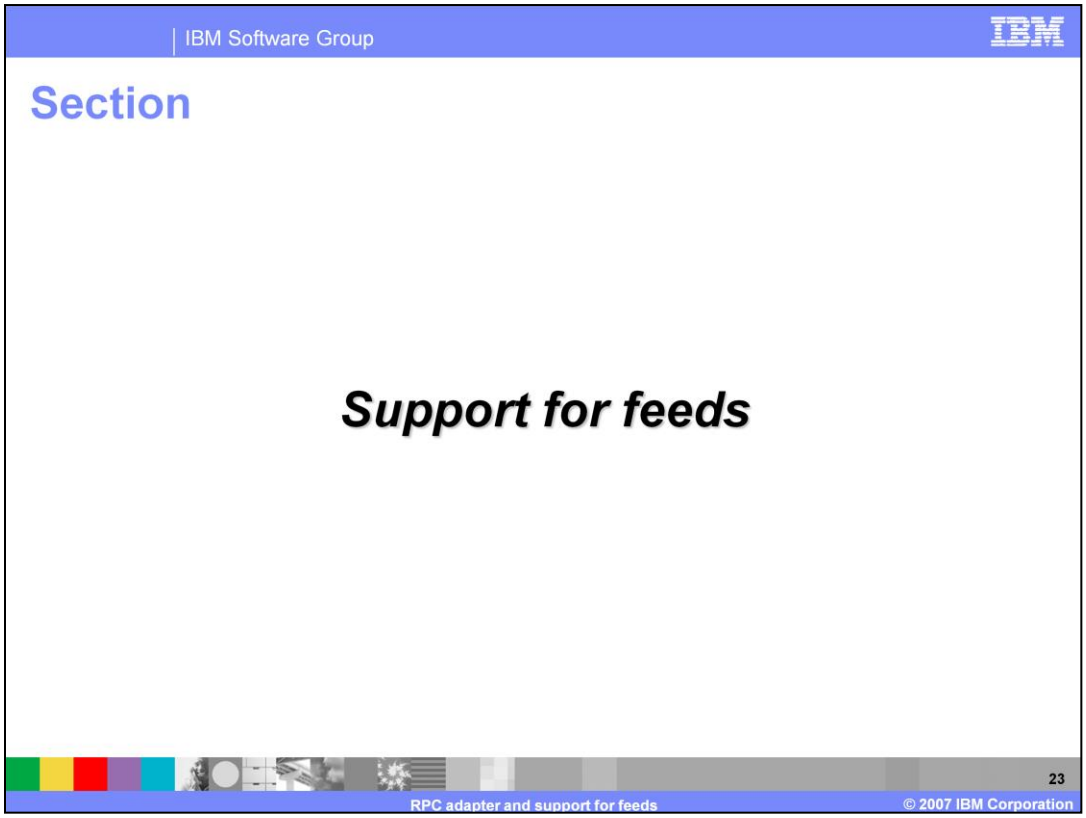
RPC adapter benefits

- Provides a lightweight Web endpoint which can expose methods of Java objects (EJB, PoJo, Web service proxies)
- Can easily be invoked from Ajax applications using JSON or XML formats
- Supports HTTP GET/POST mapping for methods
- Enabled through simple configuration options without rewriting the original Java objects, EJB or Web services

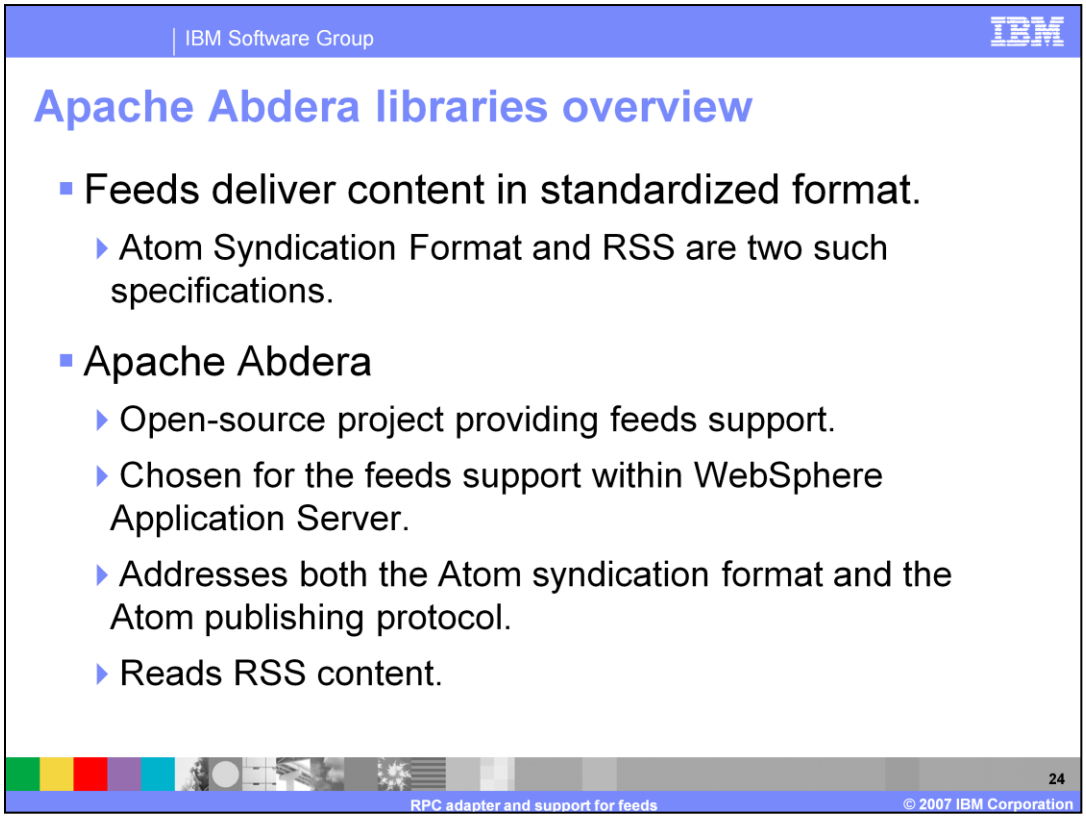
The benefits of the RPC adapter are therefore to provide a lightweight Web endpoint which can expose methods of Java objects (EJB, PoJo, Web service proxies). RPC adapter can easily be invoked from Ajax applications using JSON or XML formats and it supports HTTP GET/POST mapping for methods. Additionally, it is enabled through simple configuration options without rewriting the original Java objects, EJB or Web services



There a few RPC adapter limitations listed here. Java Beans registered with the RPC adapter must have public no-arg constructors. The RPC adapter will instantiate the Java Beans, then invoke the specified method in response to RPC requests. Also, only "exportable" methods can be invoked through the RPC adapter. An exportable method is a public method that can be invoked by reflection from string input parameters. Specifically, this allows only methods with no arguments or arguments that are primitives (for example. boolean or int), objects that can be instantiated from a String or arrays of either. Also when you specify a parameter for a particular method in the xml file, you need to specify all the parameters for that method in the xml file. Additionally, there is no authentication and authorization support built in. Last but not least, certain scopes like 'Request', 'Session', 'Application' and so on are not supported.



This section covers support for feeds.

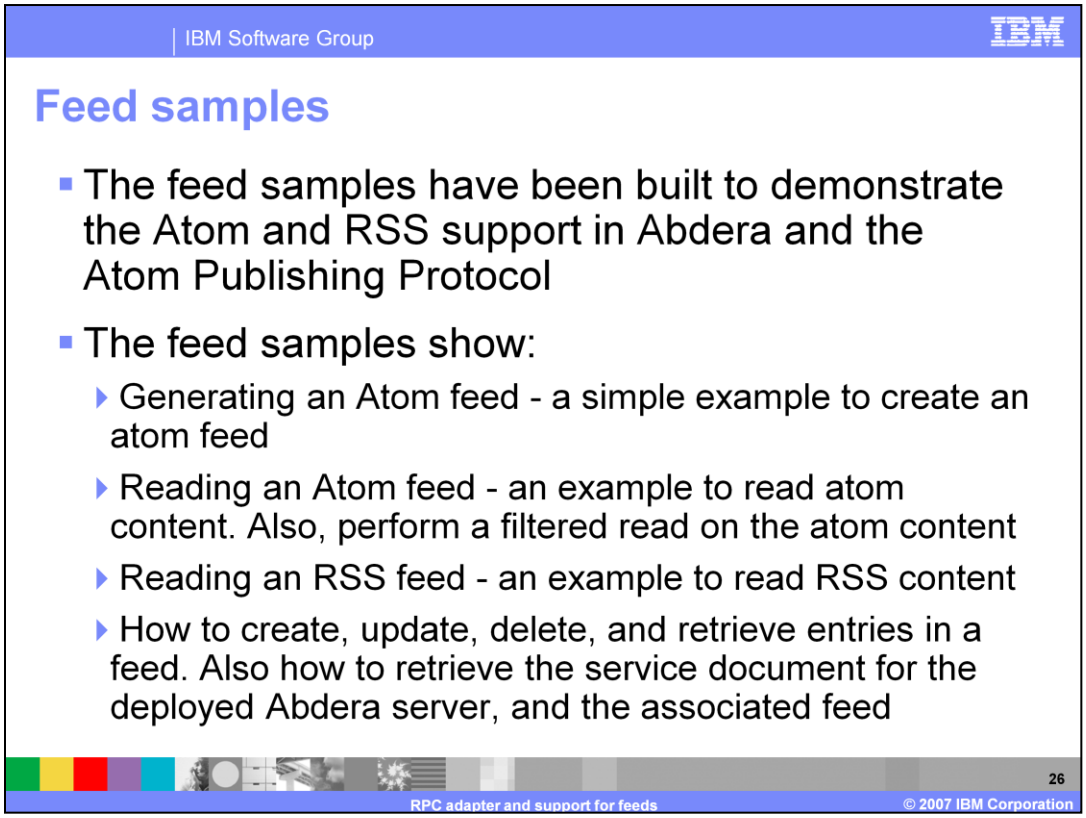


Feeds are a mechanism to deliver content in a standardized specification format. Atom Syndication Format and RSS are two such specifications. Apache Abdera is an open-source project that provides the feed support in WebSphere Application Server. Abdera supports the Atom syndication format, the Atom publishing protocol, and reading RSS content.

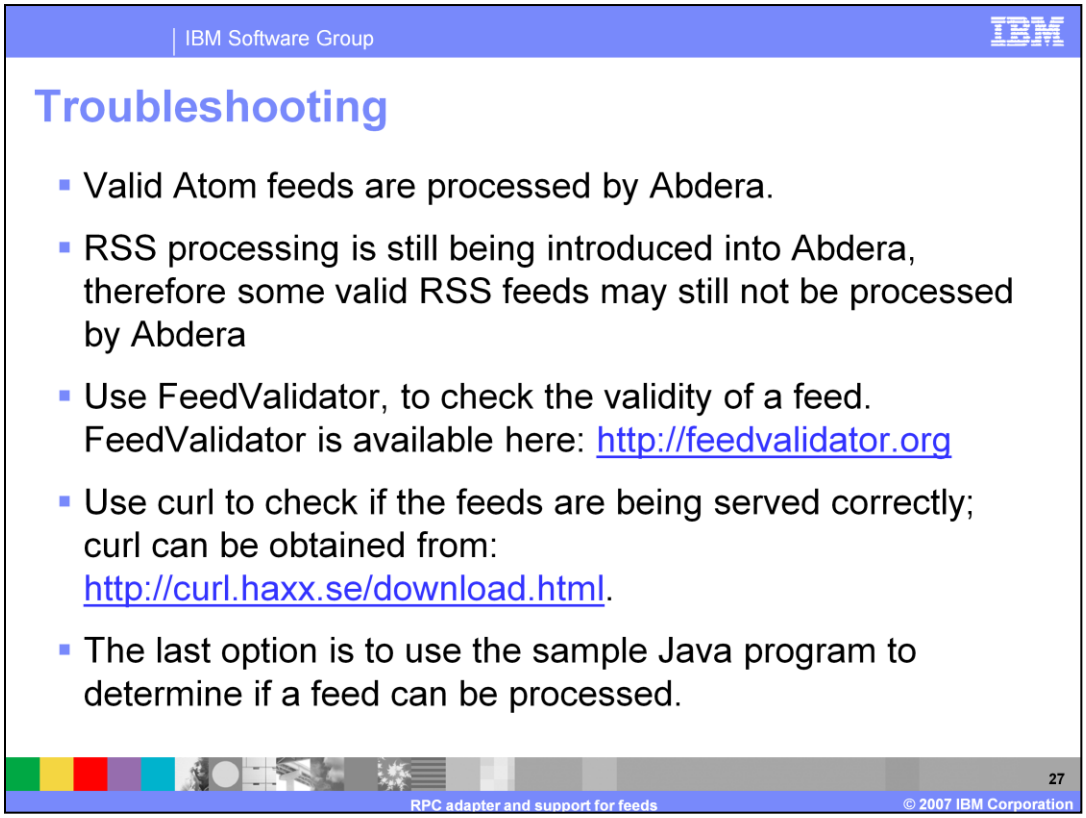
Support for feeds

- The IBM support for feeds uses apache Abdera libraries
- Feed support primarily consists of:
  - Feeds support libraries: Java archive (JAR) files for apache Abdera and the required XML Stax parser support
  - A sample application that demonstrates the use of the feeds support libraries

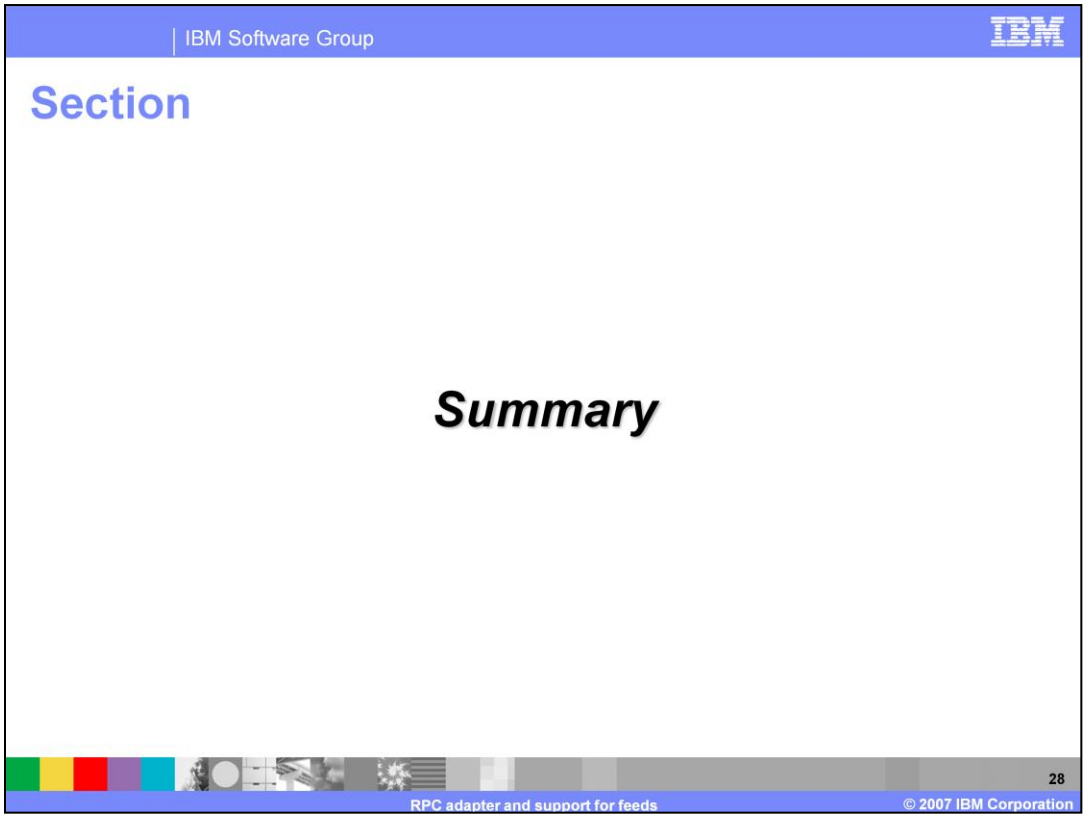
The IBM support for feeds uses the Apache Abdera libraries. The feature pack also includes a sample application that demonstrates the use of the feed support libraries.



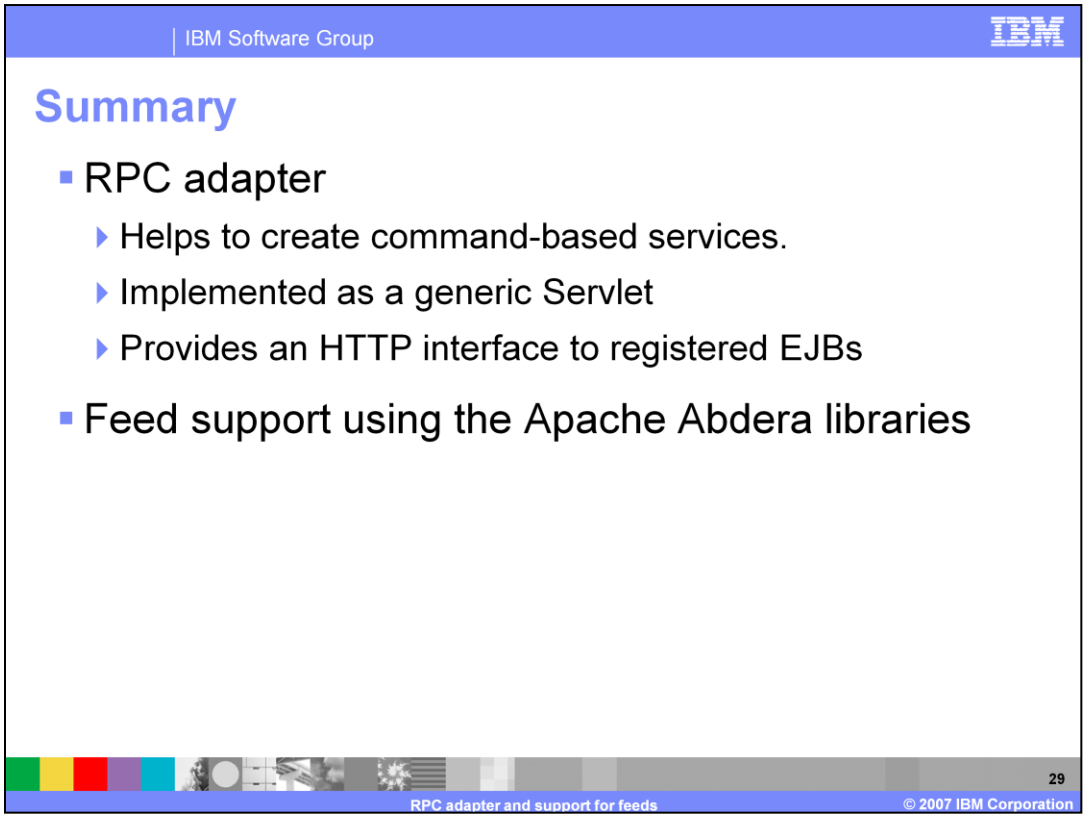
The feed samples have been built to demonstrate the Atom and RSS support in Abdera, and the Atom Publishing Protocol support. With the samples, you can see how to create an Atom feed and use filtered or unfiltered reads of Atom content. The “Atom Publishing Protocol” sample shows how to create, update, delete, and retrieve entries in a feed. You also have the option to retrieve the service document for the deployed Abdera server and the associated feed



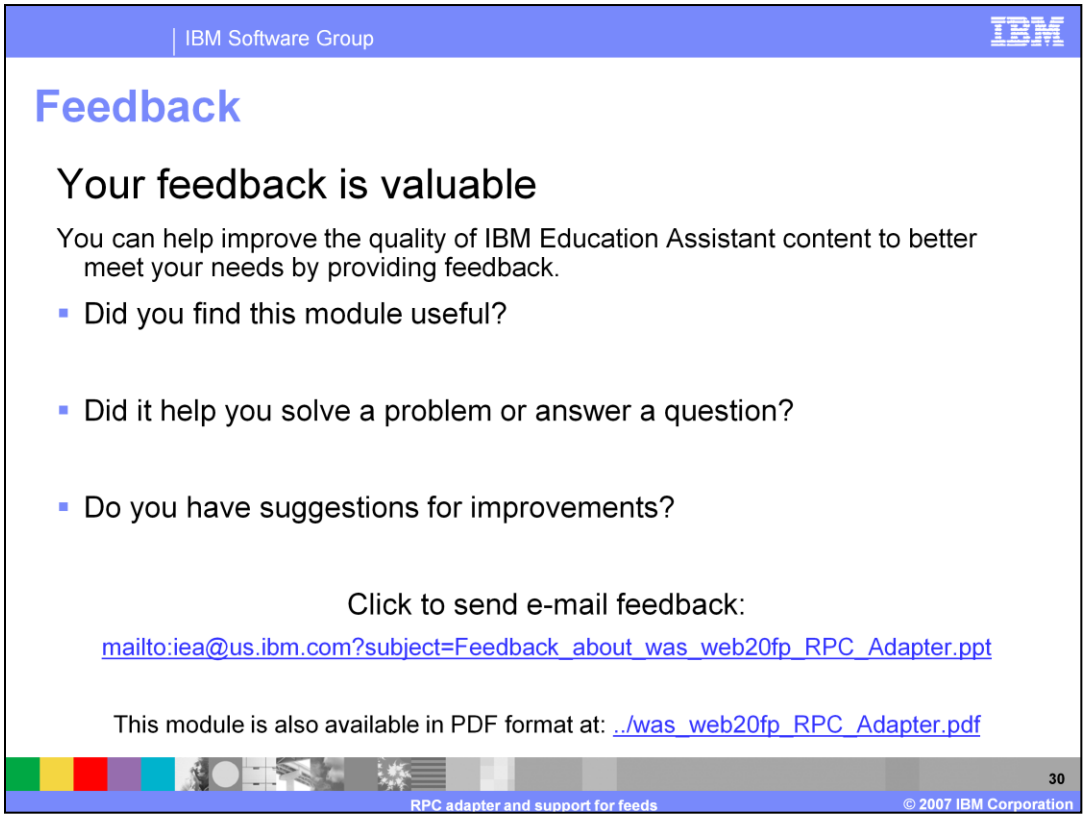
Valid Atom feeds are processed by Abdera, but RSS processing is still being introduced into Abdera, so some valid RSS feeds may not be processed. You can check the validity of a feed with the FeedValidator from the Web address listed here, or you can use curl to check if the feeds are being served correctly. The last option is to use the sample Java program provided to see if a feed can be processed.



The next section is a summary.



RPC adapter is designed to help developers create command-based services quickly and easily in a manner that complements programming styles for Ajax applications and other lightweight clients. Implemented as a generic Servlet, the RPC adapter provides an HTTP interface to registered EJBs. Finally, the feature pack uses the Apache Abdera libraries to support Atom and RSS Web feeds.



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

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

EJB, Enterprise JavaBeans, J2EE, Java, JavaScript, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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.

Information is provided "AS IS" without warranty of any kind. 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.

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.