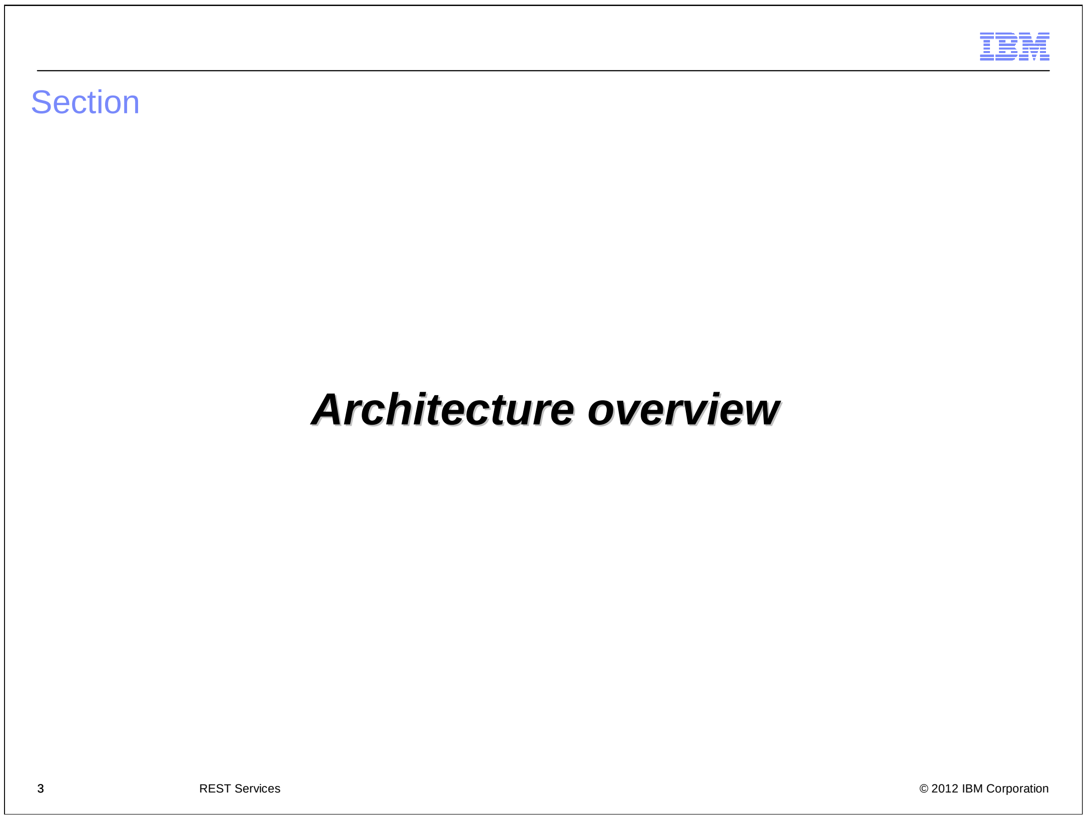
WebSphere Commerce V7 Feature Pack 4

REST services

This presentation provides an introduction to the WebSphere Commerce REST services framework.

Table of contents

This presentation begins with an overview of the REST services solution architecture. Following the architecture overview is a series of examples that demonstrate the concepts covered in the overview. This presentation concludes with some example customization scenarios and problem determination tips.

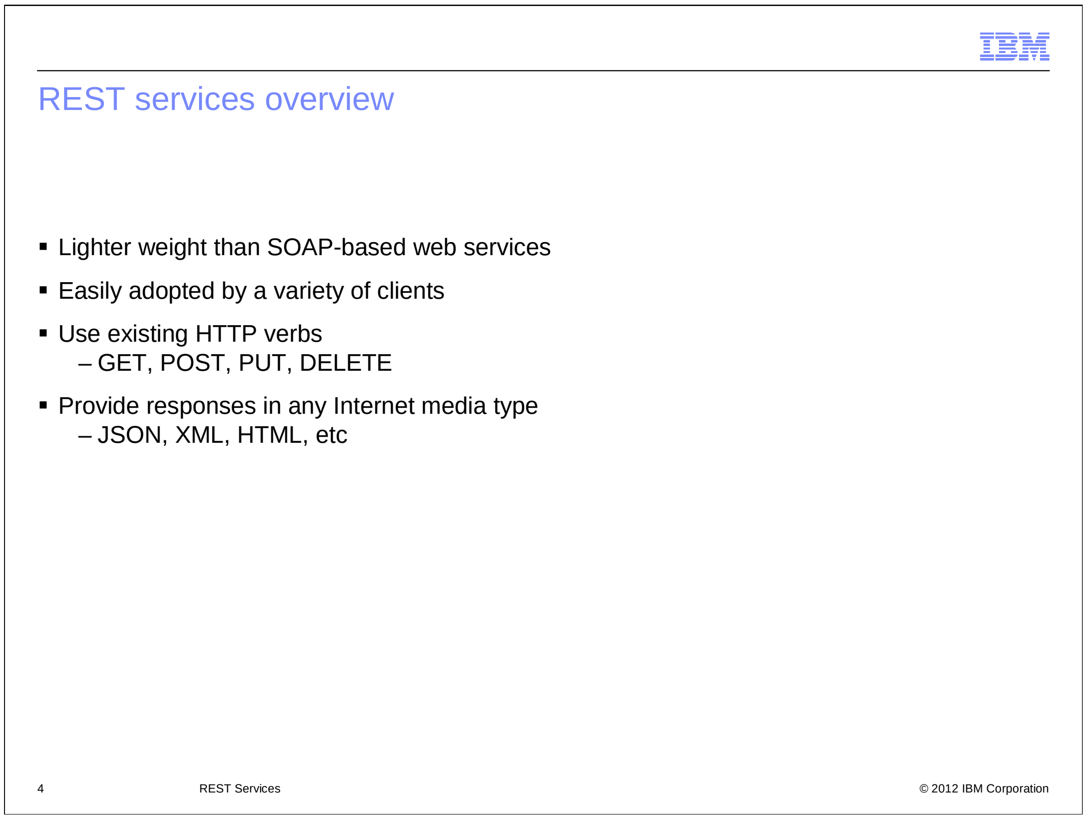


Section

Architecture overview

3 REST Services © 2012 IBM Corporation

This section introduces the REST services framework architecture.

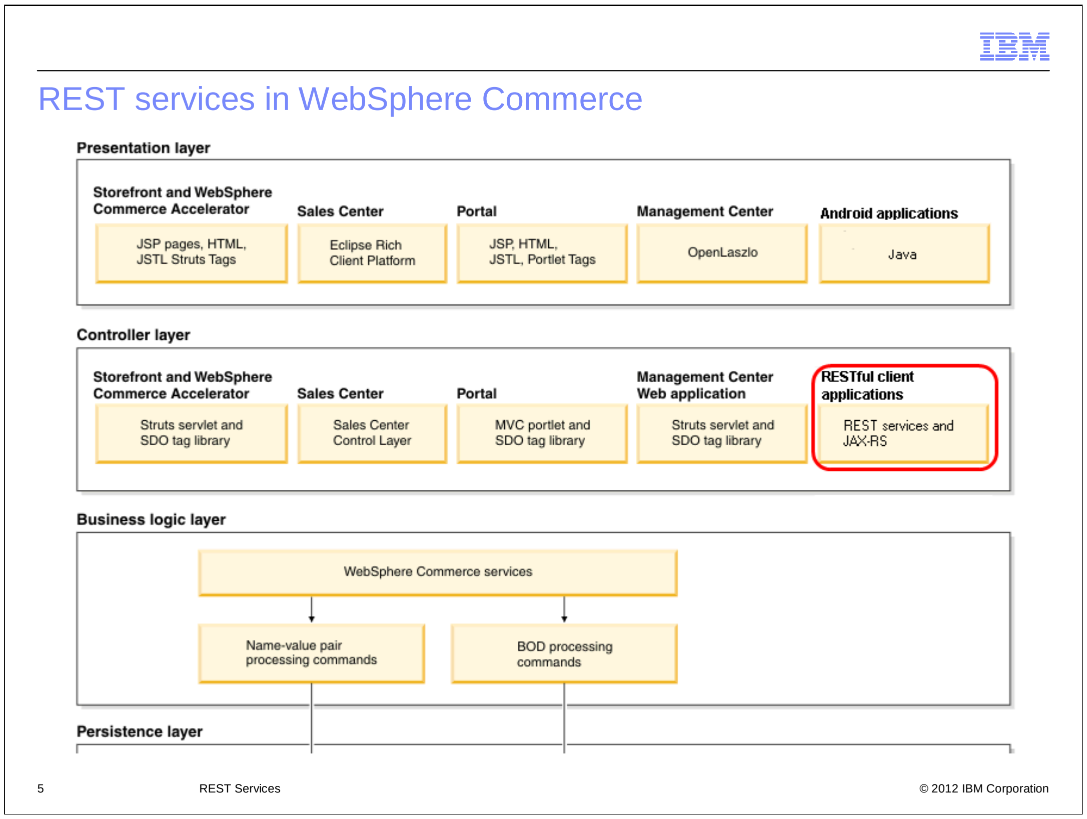


REST services overview

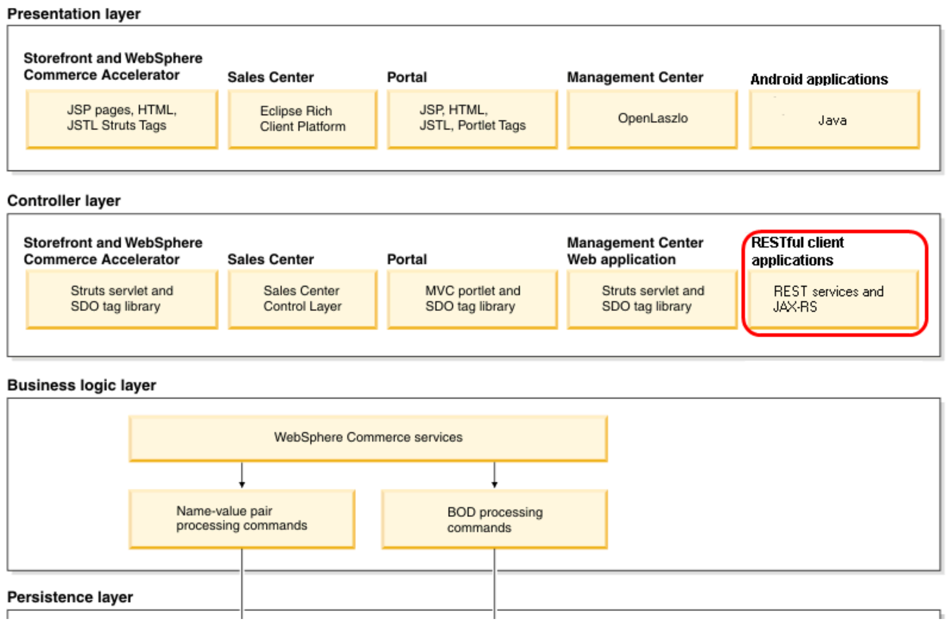
- Lighter weight than SOAP-based web services
- Easily adopted by a variety of clients
- Use existing HTTP verbs
  - GET, POST, PUT, DELETE
- Provide responses in any Internet media type
  - JSON, XML, HTML, etc

4 REST Services © 2012 IBM Corporation

REST services provide an alternate way of implementing a client-server communication model than SOAP-based web services. These lighter weight, simpler services can be easily adopted by a wide variety of clients. Some examples are discussed later in this presentation. Rather than defining an action verb within the service request as SOAP services do, REST services make use of the existing GET, POST, PUT and DELETE verbs supported by HTTP. The response format back to the client can be any internet media type. Common response types include JSON, XML and HTML.



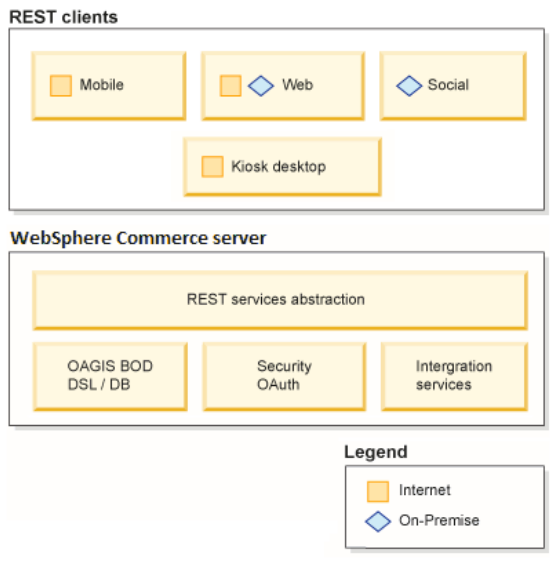
REST services in WebSphere Commerce


Presentation layer
Storefront and WebSphere Commerce Accelerator
JSP pages, HTML, JSTL Struts Tags
Sales Center
Eclipse Rich Client Platform
Portal
JSP, HTML, JSTL, Portlet Tags
Management Center
OpenLaszlo
Android applications
Java
Controller layer
Storefront and WebSphere Commerce Accelerator
Struts servlet and SDO tag library
Sales Center
Sales Center Control Layer
Portal
MVC portlet and SDO tag library
Management Center Web application
Struts servlet and SDO tag library
RESTful client applications
REST services and JAX-RS
Business logic layer
WebSphere Commerce services
Name-value pair processing commands
BOD processing commands
Persistence layer

5 REST Services © 2012 IBM Corporation

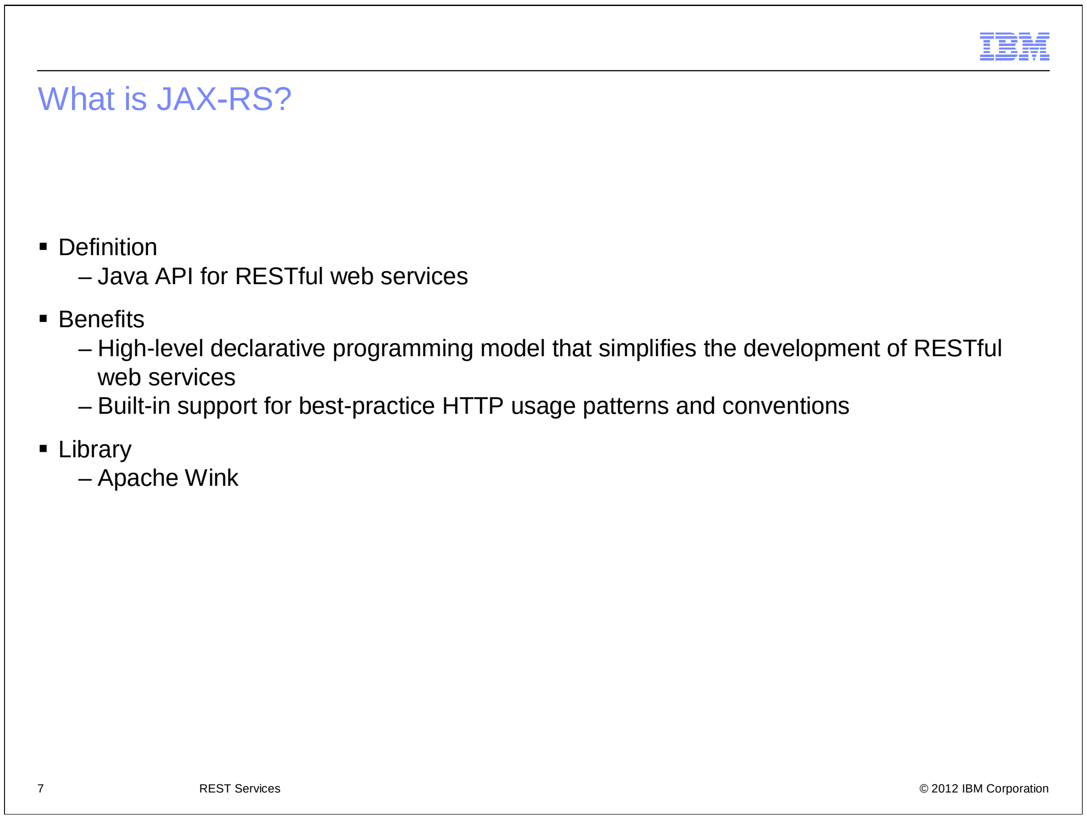
The diagram shows where REST services fit in the overall WebSphere Commerce architecture. REST services act as a controller layer between RESTful client applications, such as the sample Android application, and the WebSphere Commerce business logic layer. The REST service framework makes use of the JAX-RS API to manage incoming resource requests and outgoing responses. The REST services invoke the existing WebSphere Commerce services to process the incoming request.

Sample RESTful applications



Here are several examples of clients that might use REST services to communicate with WebSphere Commerce.

For mobile applications, REST services allow the application to use device-specific native user interfaces, or an embedded web browser for the user experience and REST services for data and updates. Web applications can include traditional storefronts or other websites that provide WebSphere Commerce functionality through REST services. Kiosk applications can use WebSphere Commerce services to bridge in-store shoppers with the online stores and services. Finally, social applications are rendered within social containers such as Facebook. Social applications can extend shopping and customer experiences.

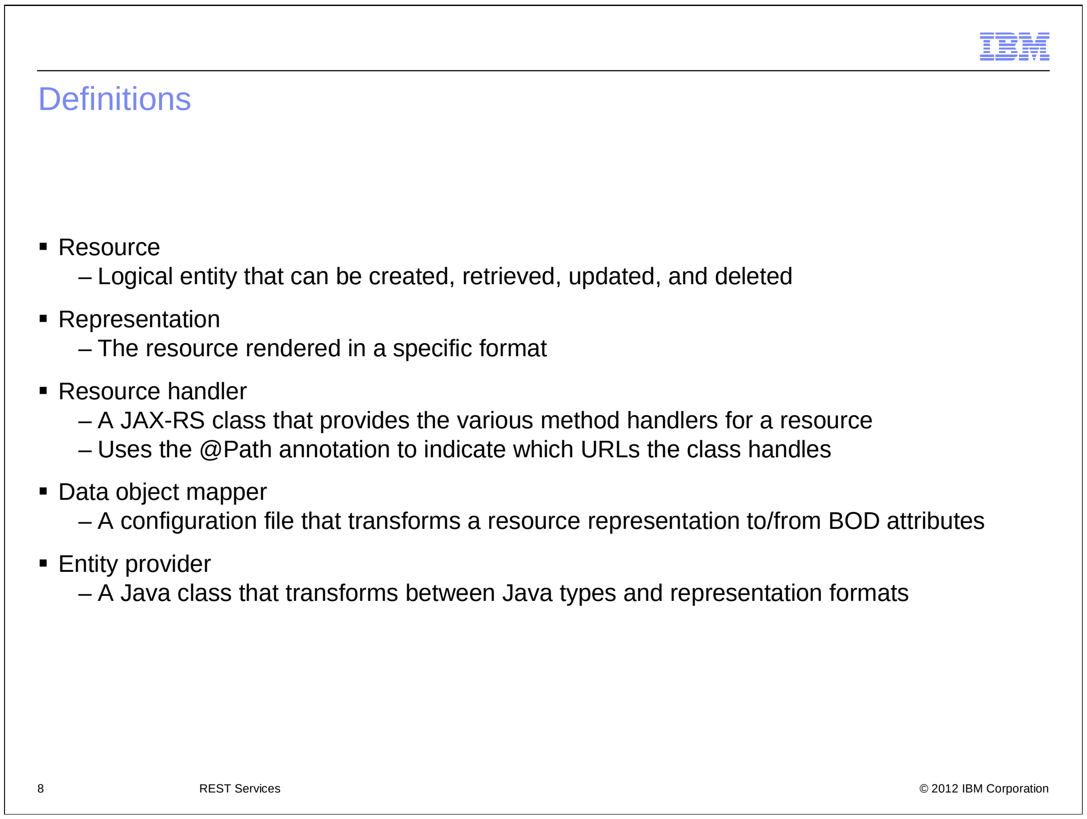


What is JAX-RS?

- Definition
  - Java API for RESTful web services
- Benefits
  - High-level declarative programming model that simplifies the development of RESTful web services
  - Built-in support for best-practice HTTP usage patterns and conventions
- Library
  - Apache Wink

7 REST Services © 2012 IBM Corporation

JAX-RS, the Java API for RESTful web services, is used to simplify the development of new RESTful web services. JAX-RS runtime handles the processing of incoming client requests and responses from the WebSphere Commerce server. The API also provides support for best practices when developing new services. WebSphere Commerce is using the Apache Wink implementation of JAX-RS.



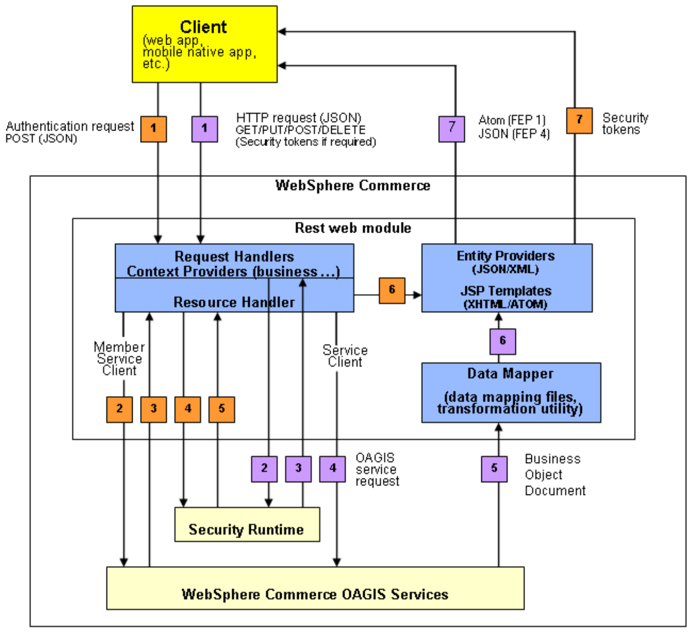
Definitions

- Resource
  - Logical entity that can be created, retrieved, updated, and deleted
- Representation
  - The resource rendered in a specific format
- Resource handler
  - A JAX-RS class that provides the various method handlers for a resource
  - Uses the @Path annotation to indicate which URLs the class handles
- Data object mapper
  - A configuration file that transforms a resource representation to/from BOD attributes
- Entity provider
  - A Java class that transforms between Java types and representation formats

8 REST Services © 2012 IBM Corporation

This slide defines several terms you will see in this presentation. A resource is a logical entity that the client can interact with, such as a shopping cart. A representation is a specific form of the resource, such as JavaScript Object Notation (JSON) object. Incoming REST service requests for a specific resource are routed to a resource handler. This is a Java class that provides method handlers for various types of resource requests, such as create and update. The @Path annotation is used to define which resources a specific class handles. A data object mapper is a configuration file responsible for converting between the Business Object Document (BOD) representation of a resource and its JSON representation. Finally, an entity provider is a Java class that transforms a resource from its server-side representation into the response format expected by the calling service.

RESTful web service framework in Feature Pack 4

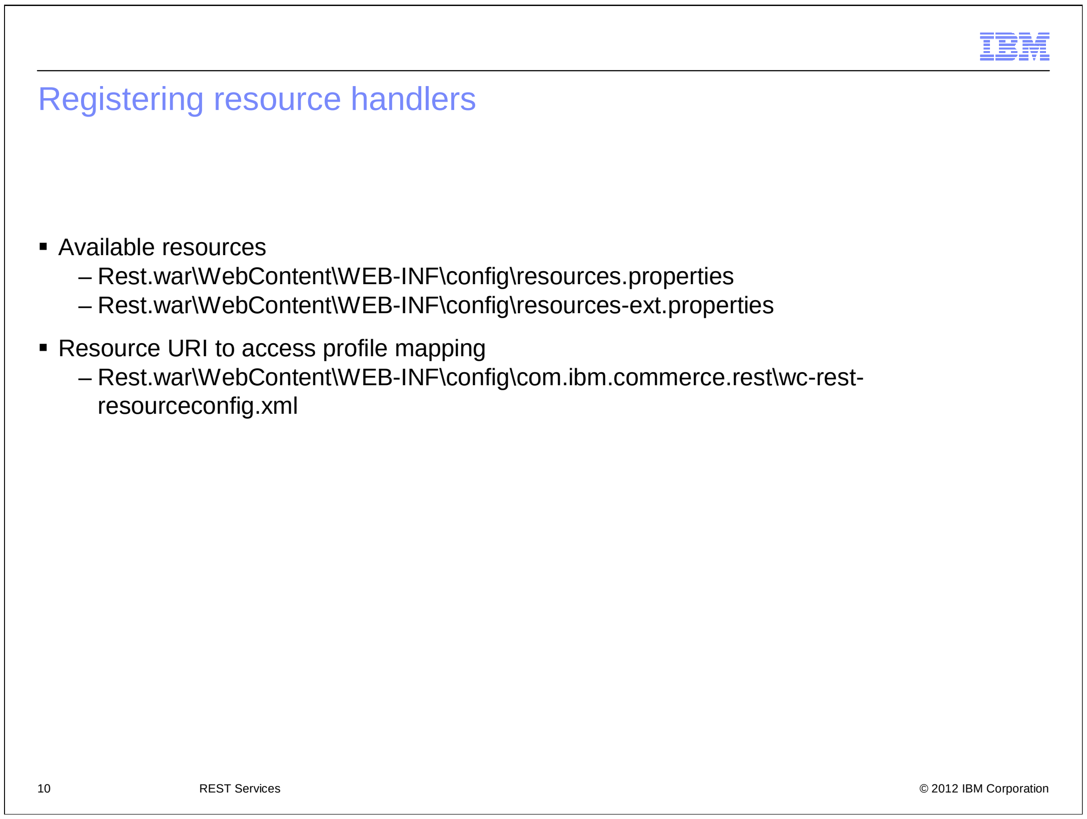


9 REST Services © 2012 IBM Corporation

This diagram shows the interactions between the REST web module and other WebSphere Commerce subsystems when handling authentication requests and regular requests. The authentication flow is marked by the orange numbers and the regular request flow is marked by the purple numbers. The authentication flow is explained first.

In step one, the client initiates an HTTP POST request to authenticate a registered shopper or get a temporary identity for a guest shopper. At step two, the JAX-RS framework invokes the request handler and matches the URL of the request to the appropriate JAX-RS resource handler. The resource handler converts the HTTP request into a WebSphere Commerce OAGIS service request to the member services. The WebSphere Commerce OAGIS service returns the result containing authentication information for the shopper in step three. In steps four and five, the REST layer calls the security runtime to create authentication tokens, WCToken and WCTrustedToken, for future requests. WCToken should be used in both non-secure and secure connections. WCTrustedToken should be used in secure connections. In step six, the JAX-RS resource invokes the entity provider to generate the response. Finally, in step seven, the response is created and returned to the client.

Now, the regular request in the purple numbers is explained. In step one, the client initiates a HTTP request, which can be GET, POST, PUT, or DELETE. If the related noun operation requires authentication, the security tokens must be sent in the HTTP request header. For non-secure requests, only the WCToken should be sent. For secure requests, both the WCToken and WCTrustedToken should be sent. The Apache Wink JAX-RS framework invokes the request handler in step two. The request handler creates the business context and calls the security runtime to verify the authentication tokens. In step three, the security runtime verifies the authentication tokens. Next, in step four, the request handler converts the incoming REST request into a WebSphere Commerce OAGIS service request. The service returns the result in BOD format in step five. In step six, the data mapper configuration file for the response BOD is loaded by the entity provider to convert the response to JSON. Finally, in step seven, the JSON response is returned to the client.



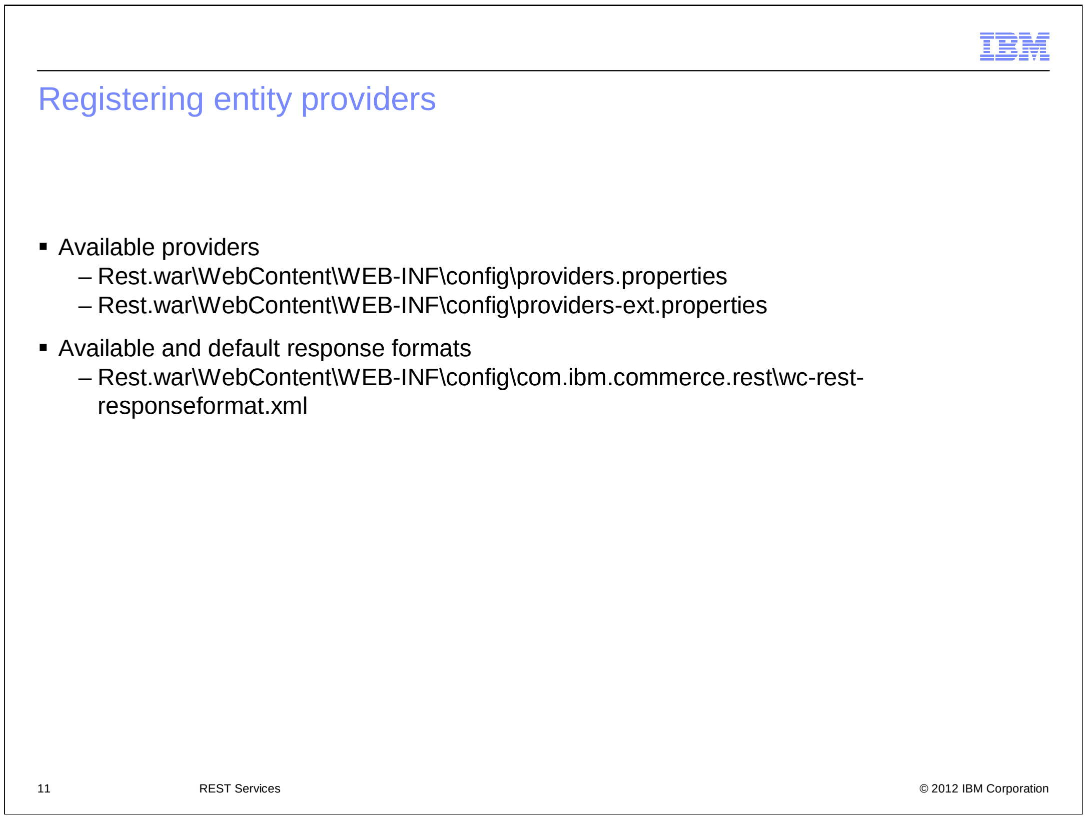
Registering resource handlers

- Available resources
  - Rest.war\WebContent\WEB-INF\config\resources.properties
  - Rest.war\WebContent\WEB-INF\config\resources-ext.properties
- Resource URI to access profile mapping
  - Rest.war\WebContent\WEB-INF\config\com.ibm.commerce.rest\wc-rest-resourceconfig.xml

10 REST Services © 2012 IBM Corporation

The available resource handlers need to be registered so that incoming requests can be directed to them. There are two property files to register resource handlers, one predefined file and one extensions file. When you create or modify a resource handler, you should register the implementation in the extensions file.

In addition to the properties file to register resource handlers, there is also an XML configuration file that maps resource URIs to the access profile to use when making the corresponding WebSphere Commerce service request.

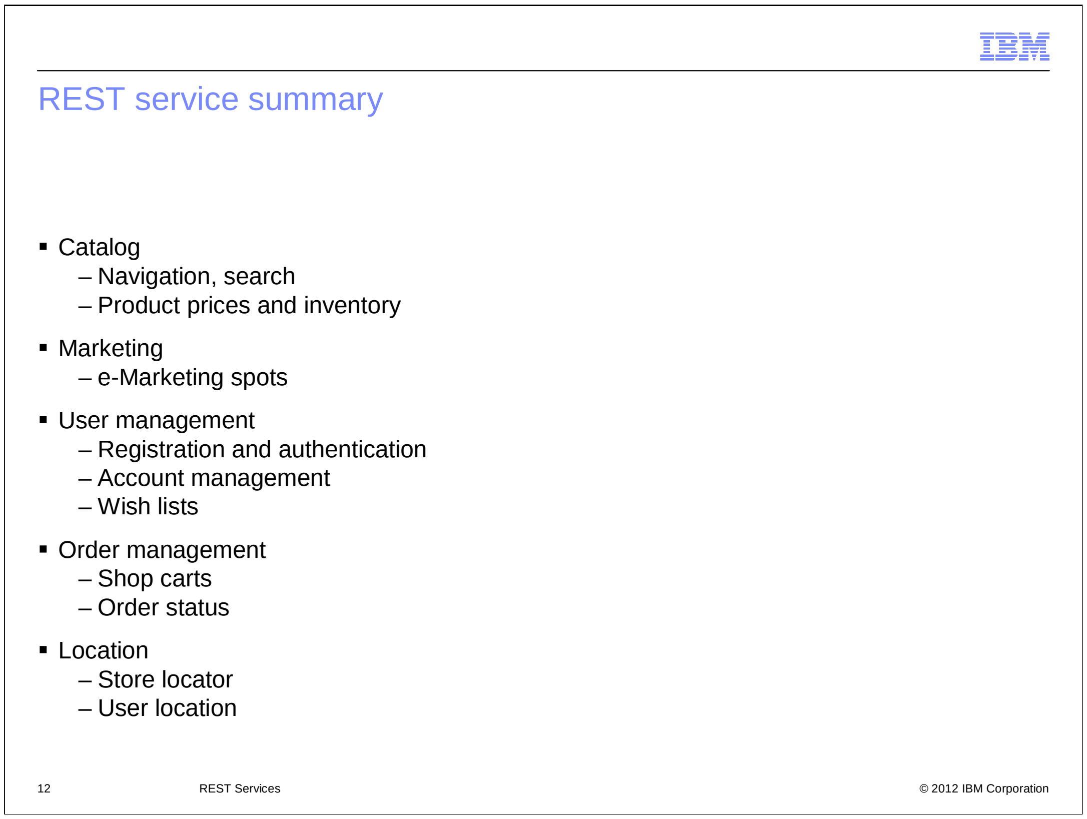


Registering entity providers

- Available providers
  - Rest.war\WebContent\WEB-INF\config\providers.properties
  - Rest.war\WebContent\WEB-INF\config\providers-ext.properties
- Available and default response formats
  - Rest.war\WebContent\WEB-INF\config\com.ibm.commerce.rest\wc-rest-responseformat.xml

11 REST Services © 2012 IBM Corporation

Entity providers also have a pair of property files for registering available implementations. When you create or modify an entity provider, you should register the implementation in the extensions file. For entity providers, there is an additional XML configuration file that maps the supported response formats to their internet media types. This file also defines the default response format.

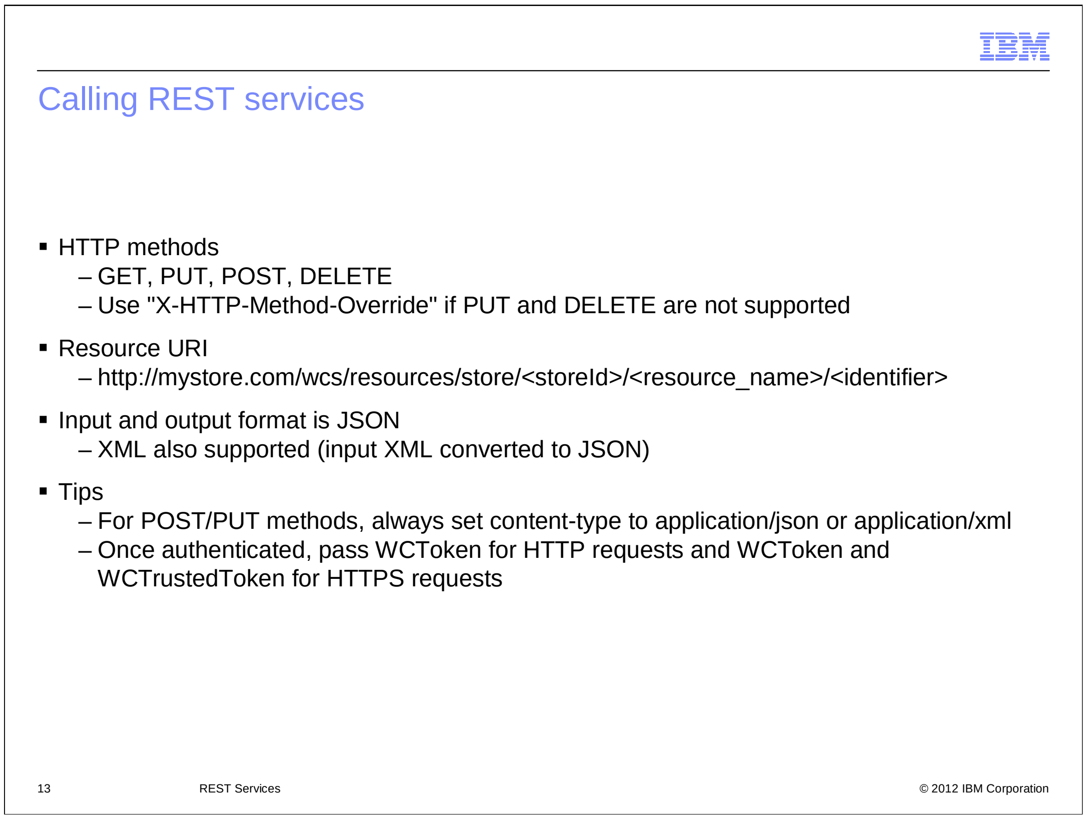


REST service summary

- Catalog
  - Navigation, search
  - Product prices and inventory
- Marketing
  - e-Marketing spots
- User management
  - Registration and authentication
  - Account management
  - Wish lists
- Order management
  - Shop carts
  - Order status
- Location
  - Store locator
  - User location

12 REST Services © 2012 IBM Corporation

This slide summarizes the WebSphere Commerce resources that are accessible through REST services in Feature Pack 4. All services are intended for business-to-consumer use only. More information on individual services is available in the Information Center. A link is provided on the resources page at the end of this presentation.



Calling REST services

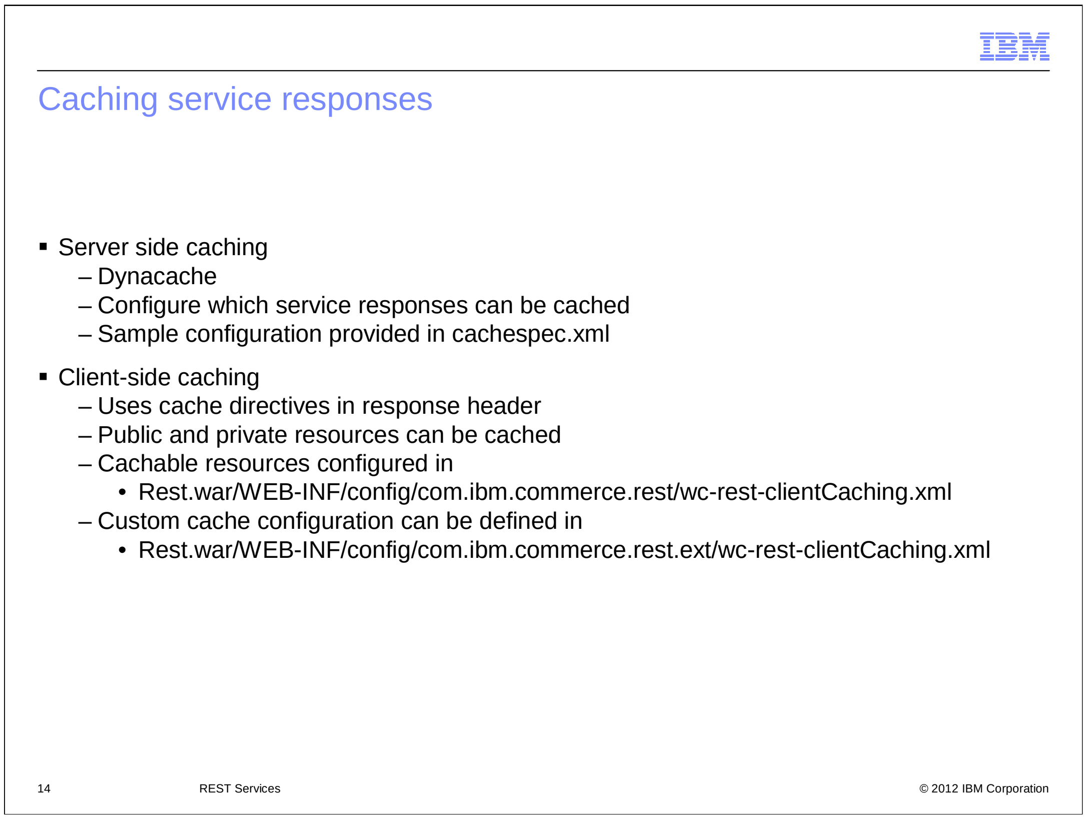
- HTTP methods
  - GET, PUT, POST, DELETE
  - Use "X-HTTP-Method-Override" if PUT and DELETE are not supported
- Resource URI
  - http://mystore.com/wcs/resources/store/<storeId>/<resource_name>/<identifier>
- Input and output format is JSON
  - XML also supported (input XML converted to JSON)
- Tips
  - For POST/PUT methods, always set content-type to application/json or application/xml
  - Once authenticated, pass WCToken for HTTP requests and WCToken and WCTrustedToken for HTTPS requests

13 REST Services © 2012 IBM Corporation

A client calls a REST service by identifying the resource it wants to act on and the action it wants to take. The action is one of GET, PUT, POST or DELETE. If PUT and DELETE are not supported, you can use the “X-HTTP-Method-Override” property in the header to specify the action. The resource to act on is specified by the URI. The URI has a fixed portion “/wcs/resources/store” followed by a client-specified portion. The client specifies the ID of the store for the request, the name of the resource and, if needed, an identifier for the resource.

The default input and output format for WebSphere Commerce REST services is JSON. XML is also supported as both an input and output format but XML input is converted to JSON before the request is processed. A specific XML format is required for the input. The details can be found in the WebSphere Commerce Information Center.

When you are passing input data in a POST or PUT request, make sure you set the content-type attribute in the header to the internet media type you are using. When making requests for an authenticated user, make sure the authentication tokens are included in the header. The WCToken is needed for regular requests and both the WCToken and WCTrustedToken are needed for secure requests.

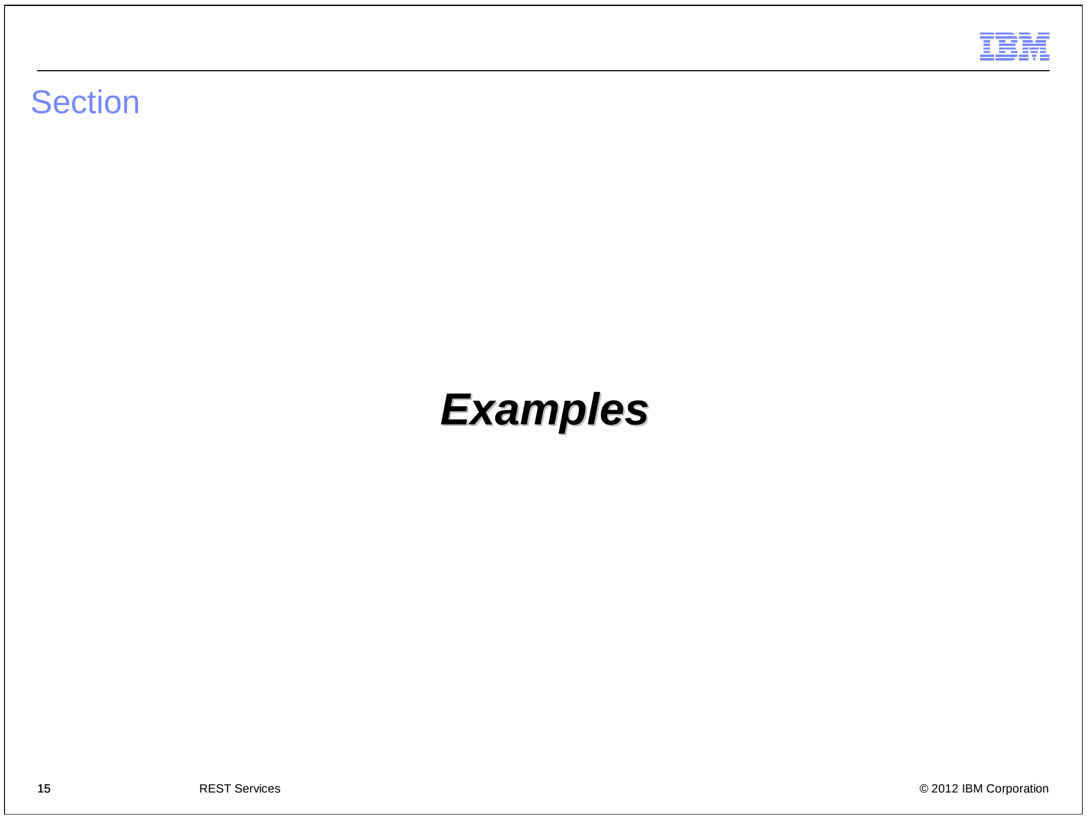


Caching service responses

- Server side caching
  - Dynacache
  - Configure which service responses can be cached
  - Sample configuration provided in cachespec.xml
- Client-side caching
  - Uses cache directives in response header
  - Public and private resources can be cached
  - Cachable resources configured in
    - Rest.war/WEB-INF/config/com.ibm.commerce.rest/wc-rest-clientCaching.xml
  - Custom cache configuration can be defined in
    - Rest.war/WEB-INF/config/com.ibm.commerce.rest.ext/wc-rest-clientCaching.xml

14 REST Services © 2012 IBM Corporation

Caching REST service responses helps to improve the performance of your client application. There are two types of caching available for REST services. Server-side caching is the standard method of caching service responses. This caching method uses dynacache and the cache configuration is defined in the cachespec.xml file. A sample configuration is provided with Feature Pack 4. Client-side caching improves performance even further by reducing calls to the server. By specifying cache directives in the response header, you can identify public or private resources to be cached by the client. WebSphere Commerce only makes use of public resource caching by default. The default client-side caching is defined in the XML configuration file shown on the slide. You can extend or modify the default cache policies by creating a custom file in an extension folder.

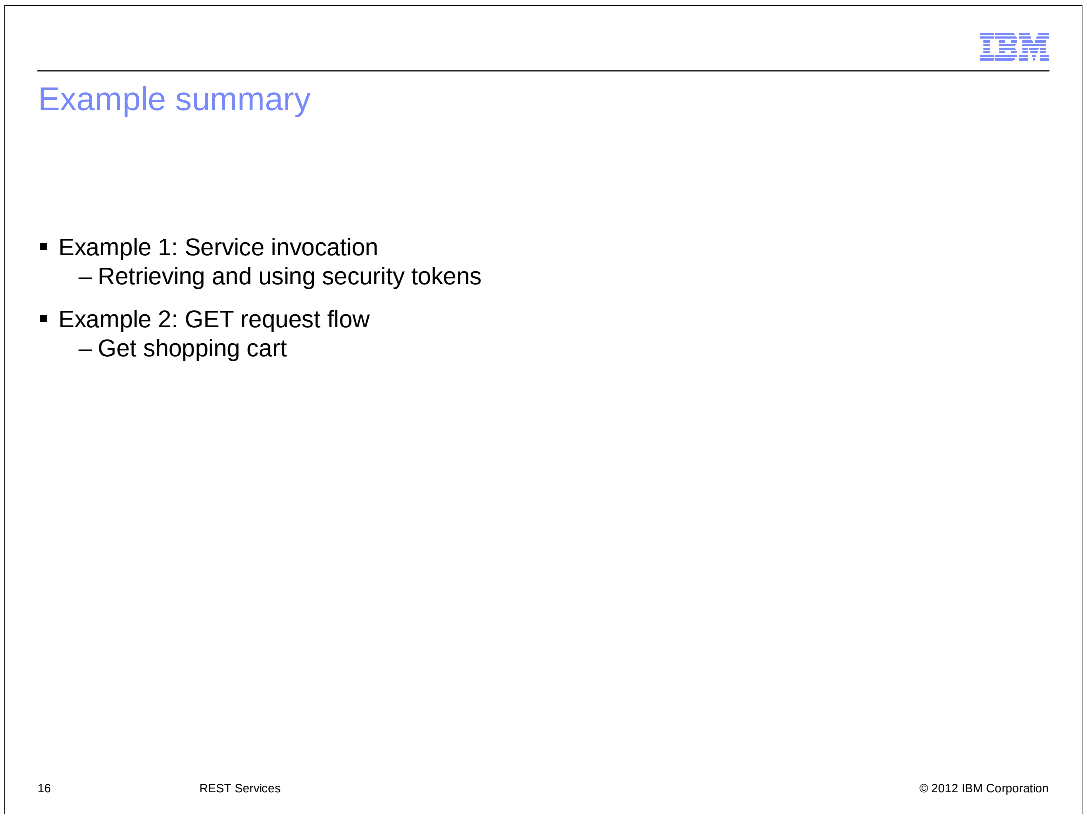


Section

Examples

15 REST Services © 2012 IBM Corporation

This section provides some examples to demonstrate the REST services framework.

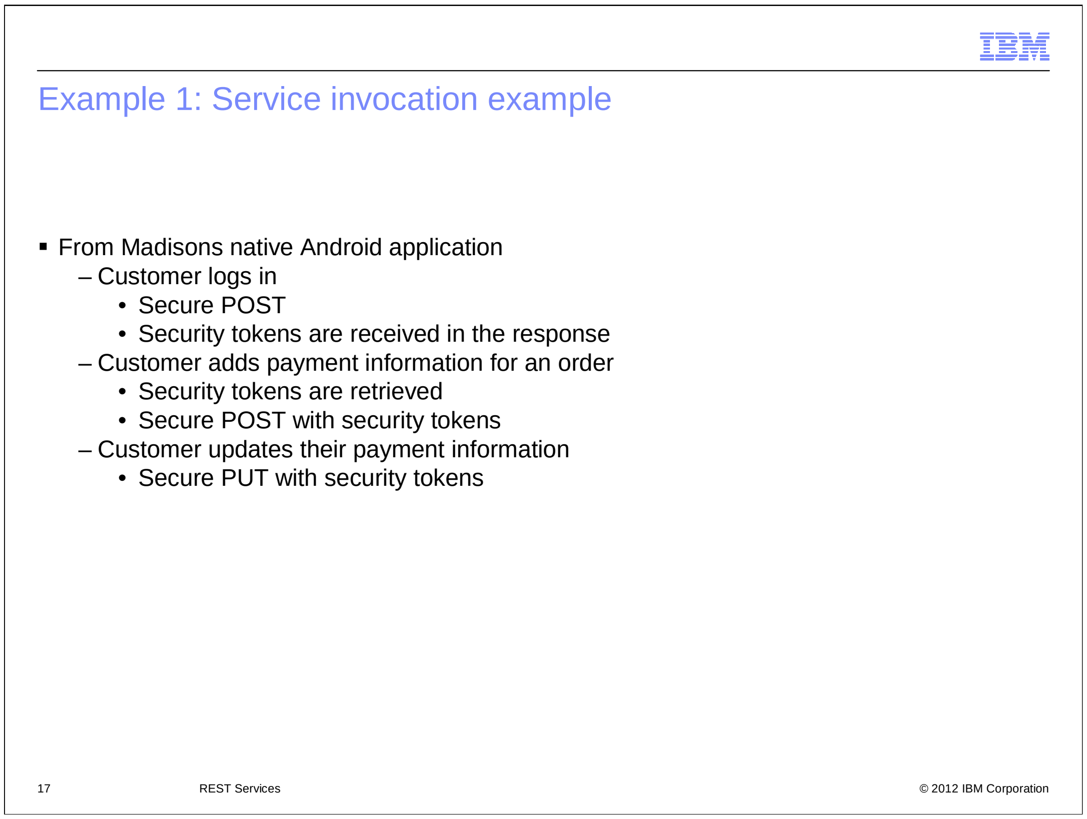


Example summary

- Example 1: Service invocation
  - Retrieving and using security tokens
- Example 2: GET request flow
  - Get shopping cart

16 REST Services © 2012 IBM Corporation

The first example looks at service invocation and working with security tokens. The second example looks at how a GET request is processed on the server side.

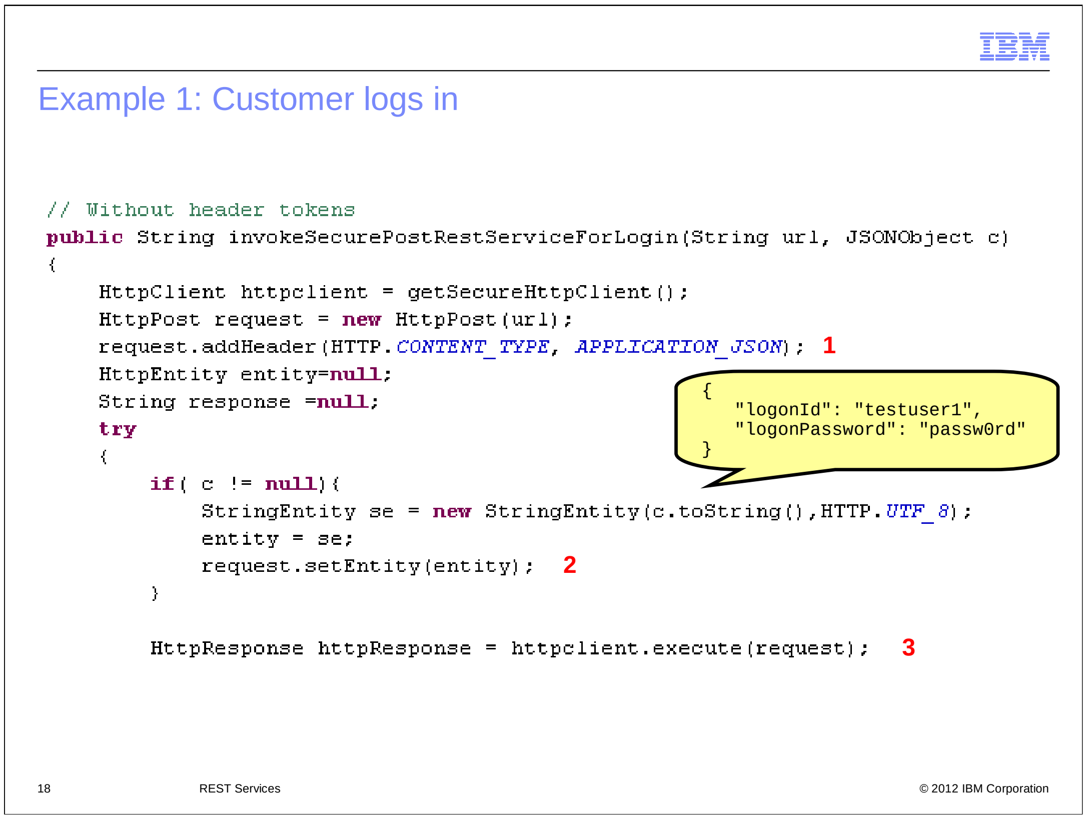


Example 1: Service invocation example

- From Madisons native Android application
  - – Customer logs in
    - • Secure POST
    - • Security tokens are received in the response
  - – Customer adds payment information for an order
    - • Security tokens are retrieved
    - • Secure POST with security tokens
  - – Customer updates their payment information
    - • Secure PUT with security tokens

17 REST Services © 2012 IBM Corporation

The code for the first example comes from the sample Madisons Android application. The next five slides show how different types of requests are created and how security tokens are managed. The example starts with customer authentication through logging in to the store and continues on with payment information requests made by the logged in shopper while checking out an order.



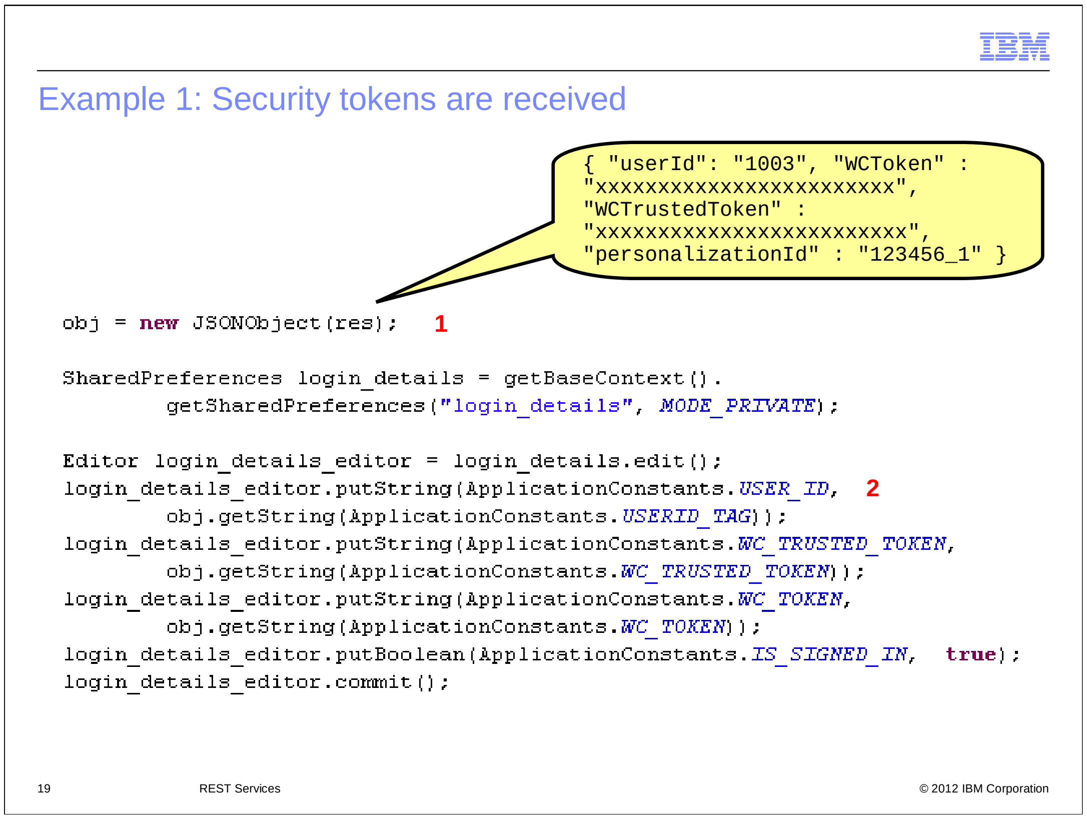
Example 1: Customer logs in

```
// Without header tokens
public String invokeSecurePostRestServiceForLogin(String url, JSONObject c)
{
    HttpClient httpclient = getSecureHttpClient();
    HttpPost request = new HttpPost(url);
    request.addHeader(HTTP.CONTENT_TYPE, APPLICATION_JSON);  1
    HttpEntity entity=null;
    String response =null;
    try
    {
        if( c != null){
            StringEntity se = new StringEntity(c.toString(),HTTP.UTF_8);
            entity = se;
            request.setEntity(entity);   2
        }

        HttpResponse httpResponse = httpclient.execute(request);   3
```

```
{
   "logonId": "testuser1",
   "logonPassword": "passw0rd"
}
```

The method shown here is used to build the REST service request for a shopper to log in. There are no tokens to pass in a login request, only the content type is set in the request header. A JSON object containing the logon ID and password supplied by the shopper is passed as the content of this secure POST request. The last line shown initiates the request and captures the response back.

The number one on the slide shows where the request header is set, the number two shows where the content is set for the POST request. An example of the content is shown in the call-out on the slide. The number three shows the REST service being invoked.



Example 1: Security tokens are received

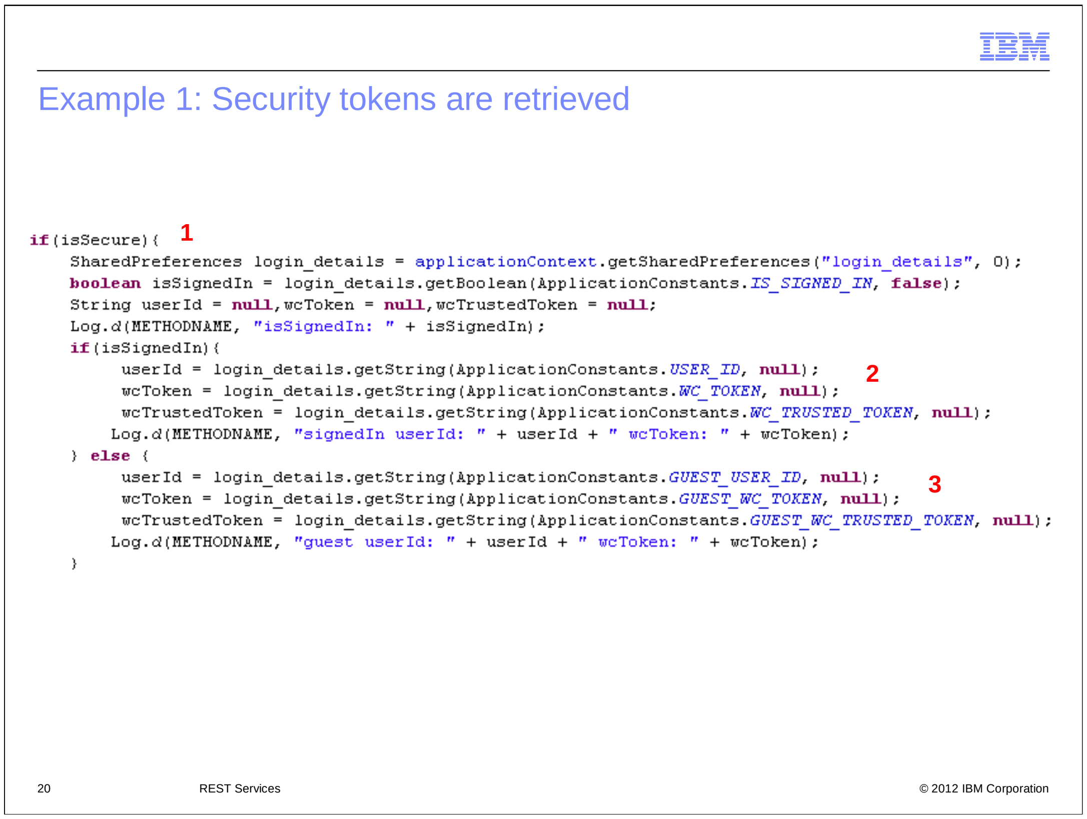
```
{ "userId": "1003", "WCToken" :
"xxxxxxxxxxxxxxxxxxxxxxxxx",
"WCTrustedToken" :
"xxxxxxxxxxxxxxxxxxxxxxxxxx",
"personalizationId" : "123456_1" }
```

```
obj = new JSONObject(res);   1

SharedPreferences login_details = getBaseContext().
        getSharedPreferences("login_details", MODE_PRIVATE);

Editor login_details_editor = login_details.edit();
login_details_editor.putString(ApplicationConstants.USER_ID,   2
        obj.getString(ApplicationConstants.USERID_TAG));
login_details_editor.putString(ApplicationConstants.WC_TRUSTED_TOKEN,
        obj.getString(ApplicationConstants.WC_TRUSTED_TOKEN));
login_details_editor.putString(ApplicationConstants.WC_TOKEN,
        obj.getString(ApplicationConstants.WC_TOKEN));
login_details_editor.putBoolean(ApplicationConstants.IS_SIGNED_IN,  true);
login_details_editor.commit();
```

19 REST Services © 2012 IBM Corporation

The call-out on this slide shows the response to the login request. The login details for this shopper, such as their user ID and authentication tokens, are saved in a shared login details object so they are available for future requests.

The number one on the slide shows the JSON response object and the number two shows the response data being populated in the login details object.



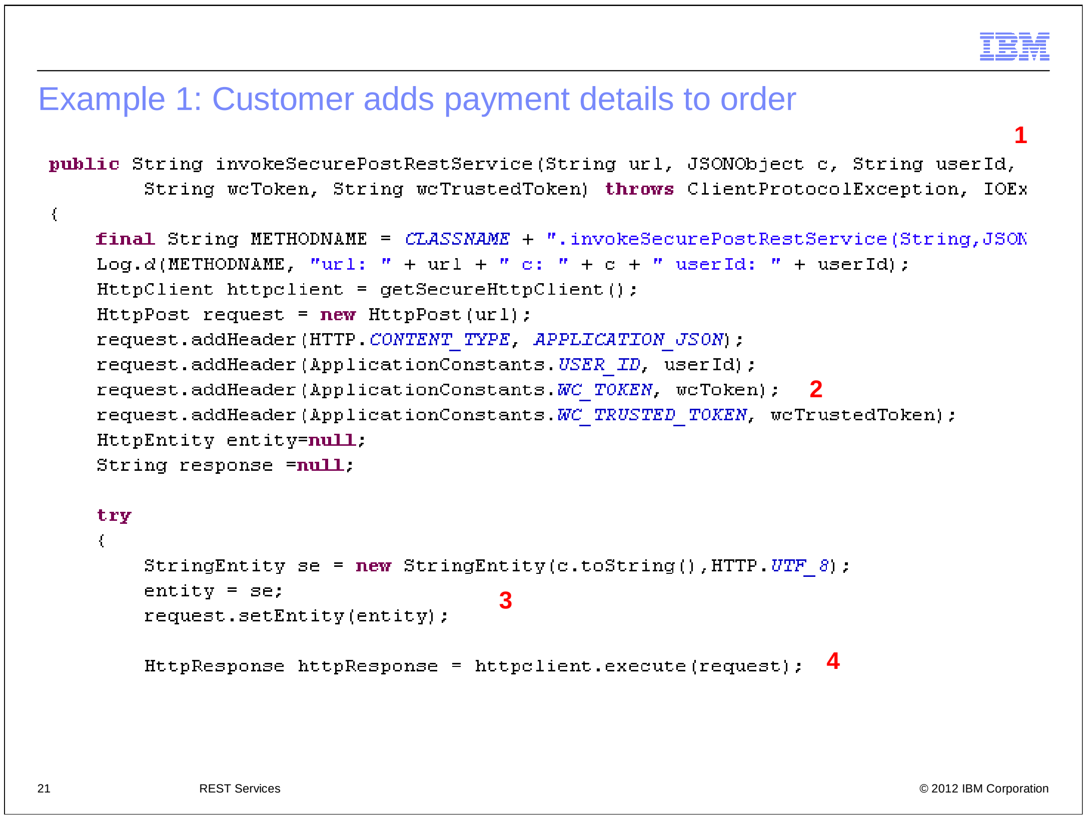
Example 1: Security tokens are retrieved

```
if(isSecure){  1
    SharedPreferences login_details = applicationContext.getSharedPreferences("login_details", 0);
    boolean isSignedIn = login_details.getBoolean(ApplicationConstants.IS_SIGNED_IN, false);
    String userId = null,wcToken = null,wcTrustedToken = null;
    Log.d(METHODNAME, "isSignedIn: " + isSignedIn);
    if(isSignedIn){
         userId = login_details.getString(ApplicationConstants.USER_ID, null);     2
         wcToken = login_details.getString(ApplicationConstants.WC_TOKEN, null);
         wcTrustedToken = login_details.getString(ApplicationConstants.WC_TRUSTED_TOKEN, null);
        Log.d(METHODNAME, "signedIn userId: " + userId + " wcToken: " + wcToken);
    } else {
         userId = login_details.getString(ApplicationConstants.GUEST_USER_ID, null);     3
         wcToken = login_details.getString(ApplicationConstants.GUEST_WC_TOKEN, null);
         wcTrustedToken = login_details.getString(ApplicationConstants.GUEST_WC_TRUSTED_TOKEN, null);
        Log.d(METHODNAME, "guest userId: " + userId + " wcToken: " + wcToken);
    }
```

20 REST Services © 2012 IBM Corporation

When the shopper takes an action in the store, such as adding payment information to their order, the authentication tokens are retrieved from the shared login details object.

The number one on the slide shows that this code snippet is for dealing with secure requests. The number two on the slide shows the user ID and authentication tokens being retrieved for a logged in shopper. For guest shoppers, the guest tokens are retrieved by the code next to the number three.

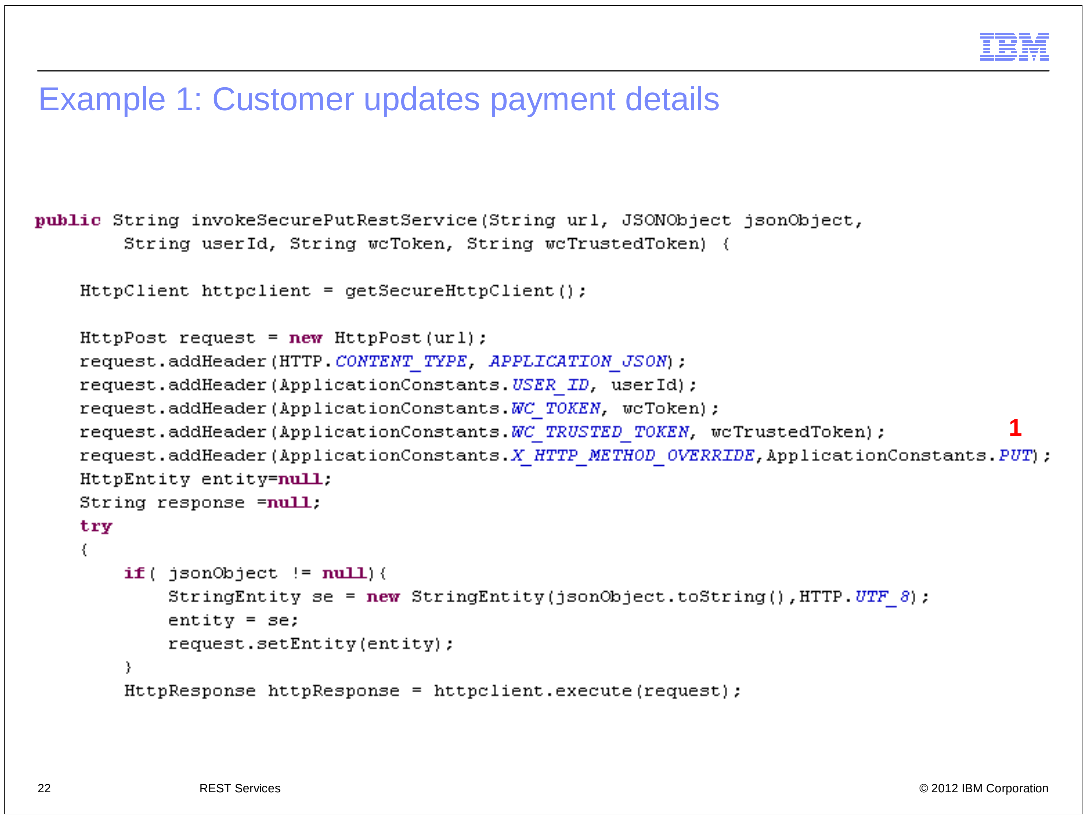

Example 1: Customer adds payment details to order

1

```
public String invokeSecurePostRestService(String url, JSONObject c, String userId,
        String wcToken, String wcTrustedToken) throws ClientProtocolException, IOEx
{
    final String METHODNAME = CLASSNAME + ".invokeSecurePostRestService(String,JSON
    Log.d(METHODNAME, "url: " + url + " c: " + c + " userId: " + userId);
    HttpClient httpclient = getSecureHttpClient();
    HttpPost request = new HttpPost(url);
    request.addHeader(HTTP.CONTENT_TYPE, APPLICATION_JSON);
    request.addHeader(ApplicationConstants.USER_ID, userId);
    request.addHeader(ApplicationConstants.WC_TOKEN, wcToken);   2
    request.addHeader(ApplicationConstants.WC_TRUSTED_TOKEN, wcTrustedToken);
    HttpEntity entity=null;
    String response =null;

    try
    {
        StringEntity se = new StringEntity(c.toString(),HTTP.UTF_8);
        entity = se;                         3
        request.setEntity(entity);

        HttpResponse httpResponse = httpclient.execute(request);   4
```

This method is used to build a secure POST request to add payment details to the order. The user ID and authentication tokens obtained from the shared login details object are added to the request header. The content of the POST request is a JSON object representing the item being added.

The number one shows the information retrieved in the previous example being passed in to this service invocation method. The number two shows where the authentication tokens are added to the request header. Since this is a secure request, both tokens are needed. The number three shows where the payment details are added for the POST request and number four shows the REST service being invoked.



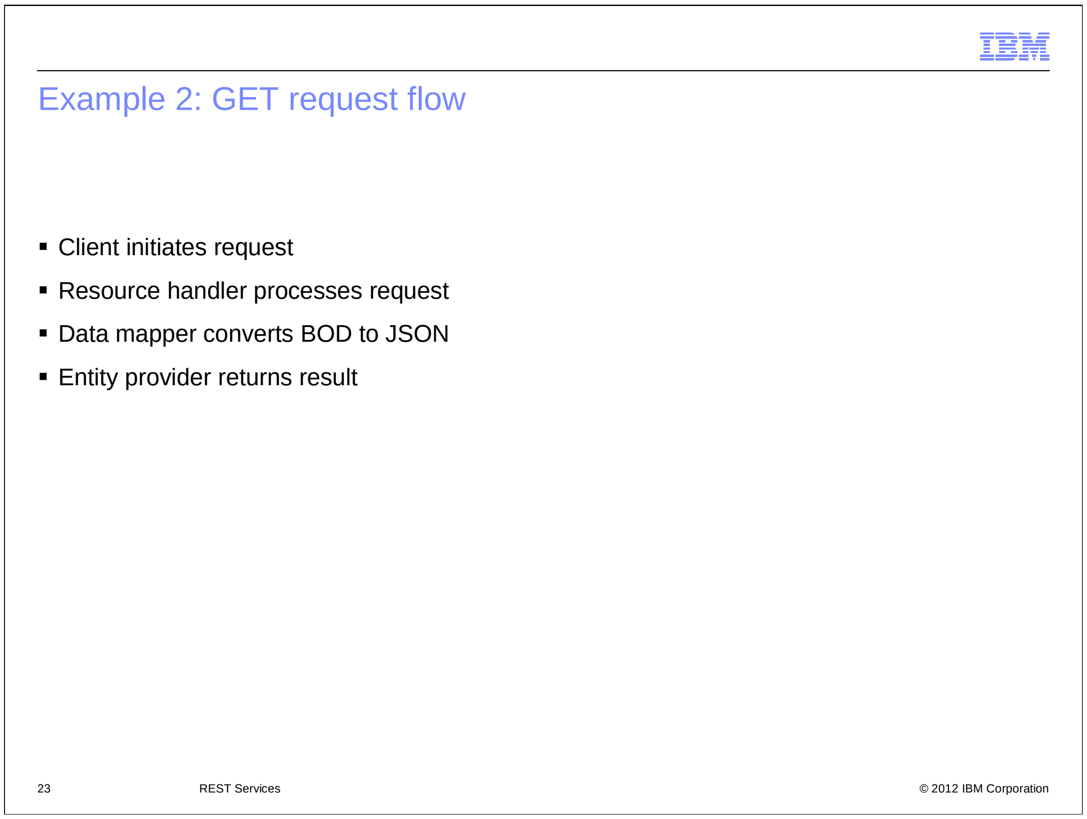
Example 1: Customer updates payment details

```
public String invokeSecurePutRestService(String url, JSONObject jsonObject,
        String userId, String wcToken, String wcTrustedToken) {

    HttpClient httpclient = getSecureHttpClient();

    HttpPost request = new HttpPost(url);
    request.addHeader(HTTP.CONTENT_TYPE, APPLICATION_JSON);
    request.addHeader(ApplicationConstants.USER_ID, userId);
    request.addHeader(ApplicationConstants.WC_TOKEN, wcToken);
    request.addHeader(ApplicationConstants.WC_TRUSTED_TOKEN, wcTrustedToken);
    request.addHeader(ApplicationConstants.X_HTTP_METHOD_OVERRIDE,ApplicationConstants.PUT);
    HttpEntity entity=null;
    String response =null;
    try
    {
        if( jsonObject != null){
            StringEntity se = new StringEntity(jsonObject.toString(),HTTP.UTF_8);
            entity = se;
            request.setEntity(entity);
        }
        HttpResponse httpResponse = httpclient.execute(request);
```

1

22 REST Services © 2012 IBM Corporation

If the shopper later decides to update the payment details for their order, a secure PUT request is created. The creation of the secure PUT request is very similar to the secure POST request. The difference is the extra header attribute for X-HTTP-Method-Override. This extra line is marked with the number one.

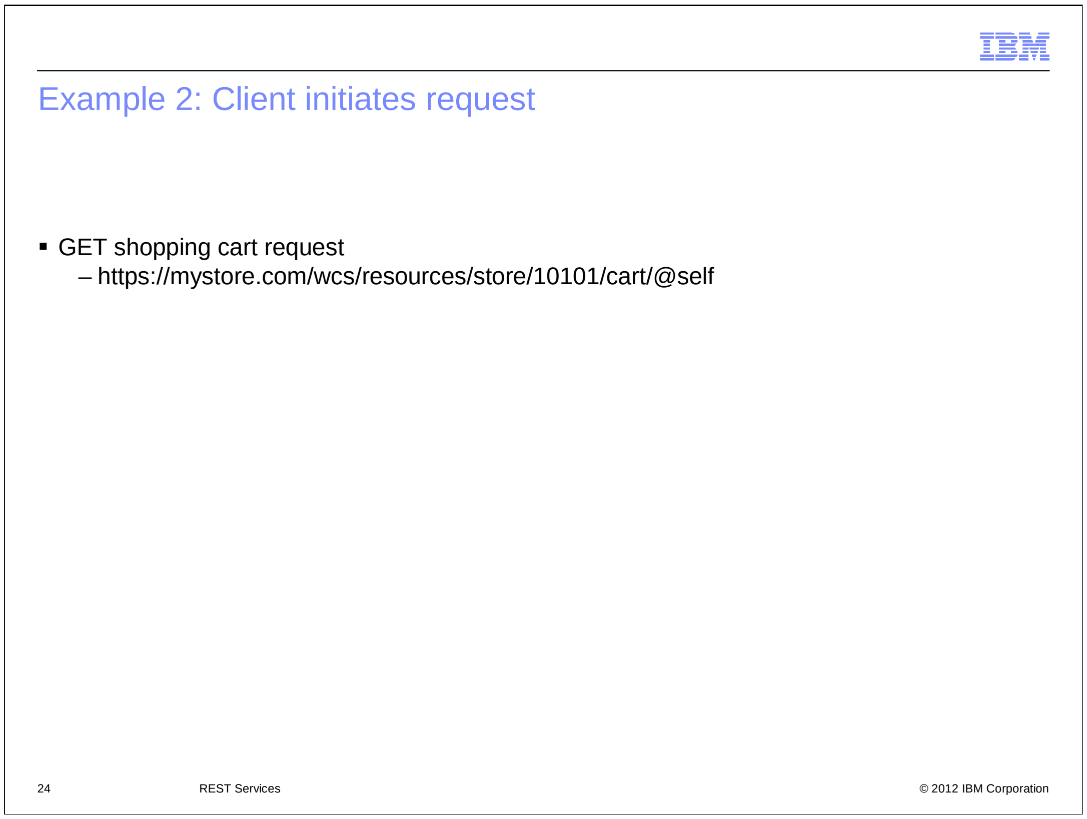


Example 2: GET request flow

- Client initiates request
- Resource handler processes request
- Data mapper converts BOD to JSON
- Entity provider returns result

23 REST Services © 2012 IBM Corporation

The second example follows the processing of a GET request from invocation to the response being returned.

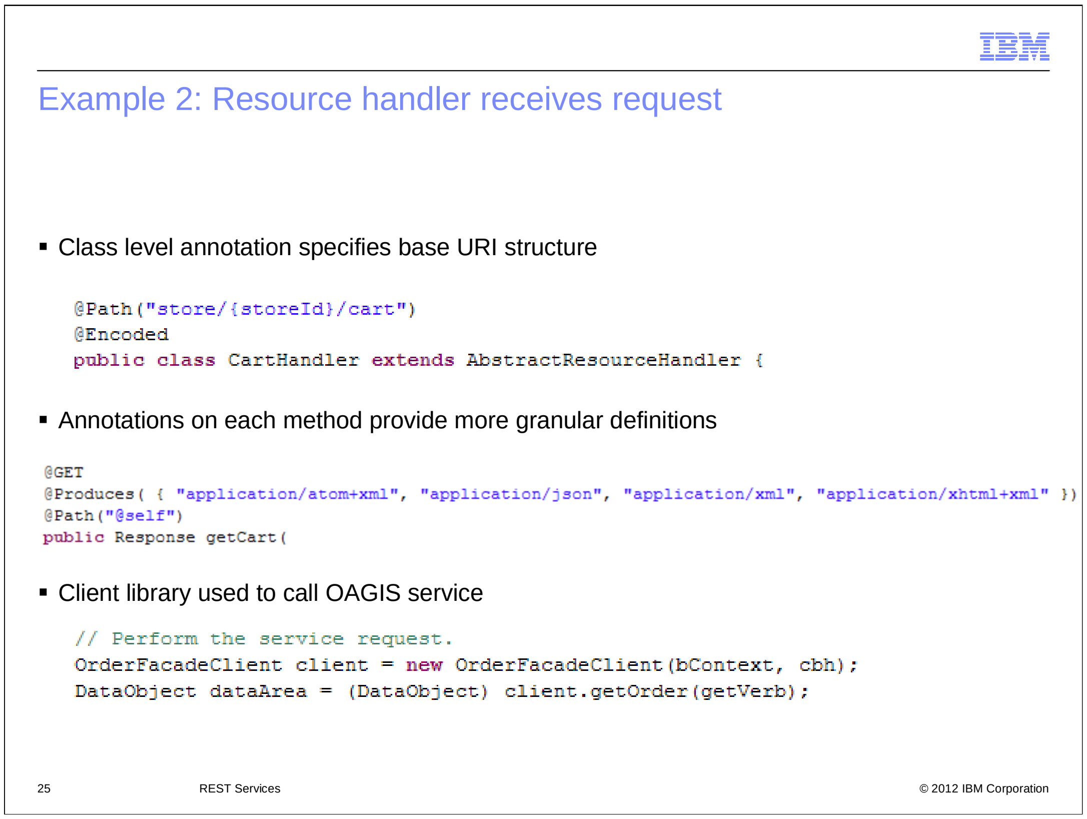


Example 2: Client initiates request

- GET shopping cart request
  - https://mystore.com/wcs/resources/store/10101/cart/@self

24 REST Services © 2012 IBM Corporation

This is the request from the client. It is a request to get the shopping cart for the current shopper. Authentication tokens identifying the shopper are included in the request header.



Example 2: Resource handler receives request

- Class level annotation specifies base URI structure

```
@Path("store/{storeId}/cart")
@Encoded
public class CartHandler extends AbstractResourceHandler {
```

- Annotations on each method provide more granular definitions

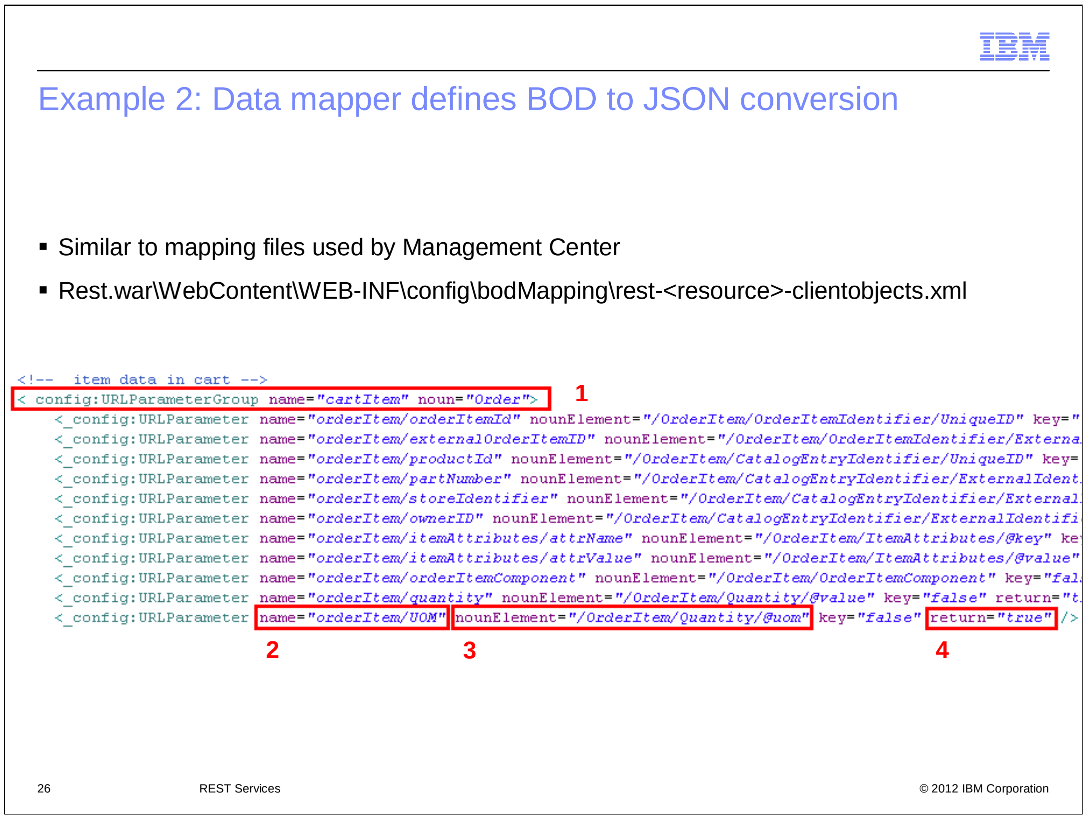
```
@GET
@Produces( { "application/atom+xml", "application/json", "application/xml", "application/xhtml+xml" })
@Path("@self")
public Response getCart(
```

- Client library used to call OAGIS service

```
// Perform the service request.
OrderFacadeClient client = new OrderFacadeClient(bContext, cbh);
DataObject dataArea = (DataObject) client.getOrder(getVerb);
```

25 REST Services © 2012 IBM Corporation

The URI in the request is matched to a corresponding resource handler in WebSphere Commerce. Resource handlers identify which URL’s they support using the @Path annotation. Within the resource handler class, there are methods that process specific types of requests for the resource such GET requests and POST requests. The second code snippet you see here is the method header for the method that will process the GET cart request. In order to get the shopper’s cart details, the resource handler method uses the OrderFacadeClient to call the OAGIS service. The cart details are returned in the response BOD.



Example 2: Data mapper defines BOD to JSON conversion

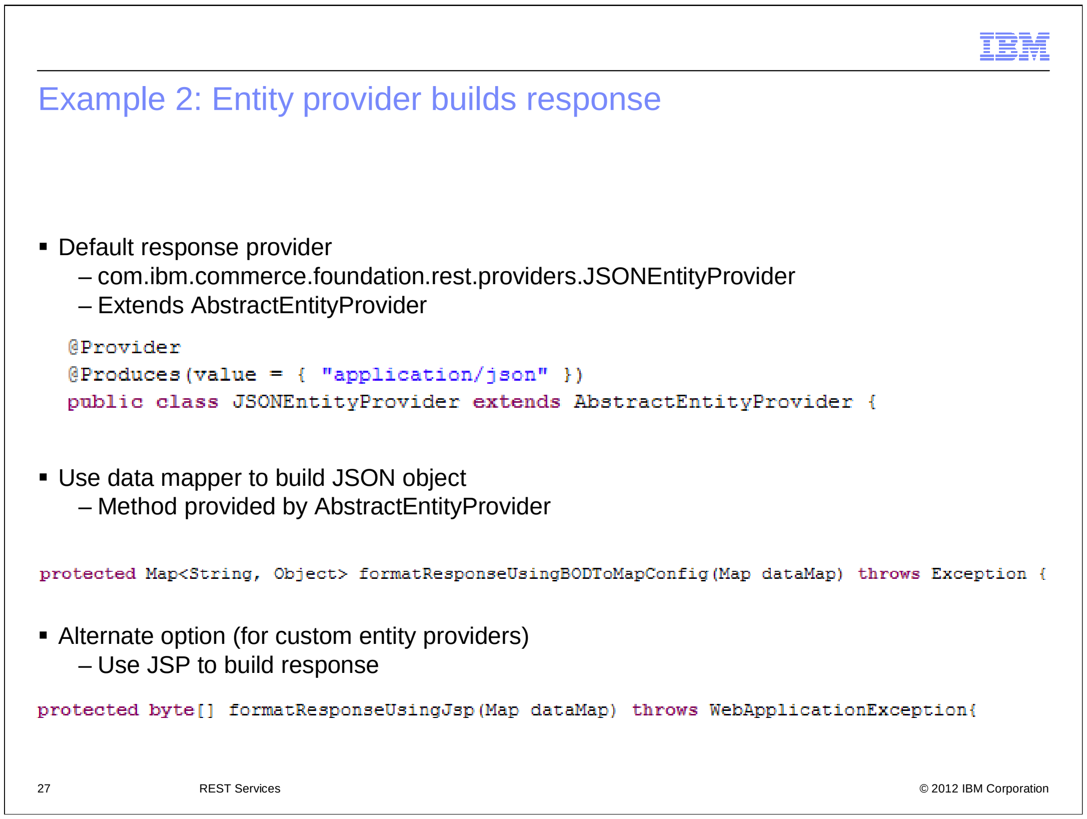
- Similar to mapping files used by Management Center
- Rest.war\WebContent\WEB-INF\config\bodMapping\rest-<resource>-clientobjects.xml

```
<!--  item data in cart -->
< config:URLParameterGroup name="cartItem" noun="Order">
    <_config:URLParameter name="orderItem/orderItemId" nounElement="/OrderItem/OrderItemIdentifier/UniqueID" key="
    <_config:URLParameter name="orderItem/externalOrderItemID" nounElement="/OrderItem/OrderItemIdentifier/Externa
    <_config:URLParameter name="orderItem/productId" nounElement="/OrderItem/CatalogEntryIdentifier/UniqueID" key=
    <_config:URLParameter name="orderItem/partNumber" nounElement="/OrderItem/CatalogEntryIdentifier/ExternalIdent
    <_config:URLParameter name="orderItem/storeIdentifier" nounElement="/OrderItem/CatalogEntryIdentifier/External
    <_config:URLParameter name="orderItem/ownerID" nounElement="/OrderItem/CatalogEntryIdentifier/ExternalIdentifi
    <_config:URLParameter name="orderItem/itemAttributes/attrName" nounElement="/OrderItem/ItemAttributes/@key" ke
    <_config:URLParameter name="orderItem/itemAttributes/attrValue" nounElement="/OrderItem/ItemAttributes/@value"
    <_config:URLParameter name="orderItem/orderItemComponent" nounElement="/OrderItem/OrderItemComponent" key="fal
    <_config:URLParameter name="orderItem/quantity" nounElement="/OrderItem/Quantity/@value" key="false" return="t
    <_config:URLParameter name="orderItem/UOM" nounElement="/OrderItem/Quantity/@uom" key="false" return="true" />
```

26 REST Services © 2012 IBM Corporation

Since the cart details are returned to the resource handler in BOD format but the REST service is expected to return JSON format some data mapping is needed to convert one form to the other. This mapping is very similar to the mapping in Management Center that is used to convert BOD format to the internal Management Center object format. Each resource type has it's own mapping file. The code snippet here shows a portion of the cart resource mapping file. Each element in the BOD is mapped to an element in the JSON object. The return attribute specifies whether to include the data in the REST service response. These same mapping files are also used to convert JSON to BOD format for POST and PUT requests.

The number one shows the logical name for this part of the mapping. The numbers two and three show the REST representation and BOD representation of the data. The number four shows the return attribute. Setting this attribute to true means the data should be returned in the REST response. False means the data should not be included.



Example 2: Entity provider builds response

- Default response provider
  - com.ibm.commerce.foundation.rest.providers.JSONEntityProvider
  - Extends AbstractEntityProvider

```
@Provider
@Produces(value = { "application/json" })
public class JSONEntityProvider extends AbstractEntityProvider {
```

- Use data mapper to build JSON object
  - Method provided by AbstractEntityProvider

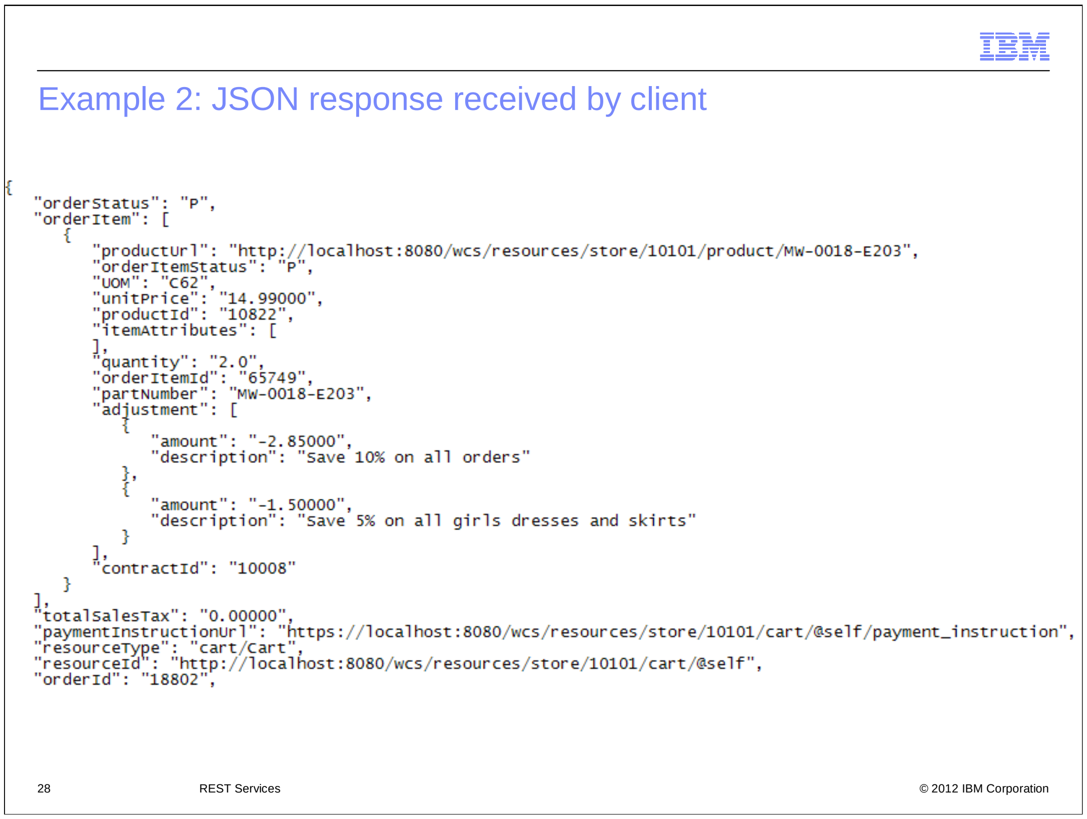
```
protected Map<String, Object> formatResponseUsingBODToMapConfig(Map dataMap) throws Exception {
```

- Alternate option (for custom entity providers)
  - Use JSP to build response

```
protected byte[] formatResponseUsingJsp(Map dataMap) throws WebApplicationException{
```

27 REST Services © 2012 IBM Corporation

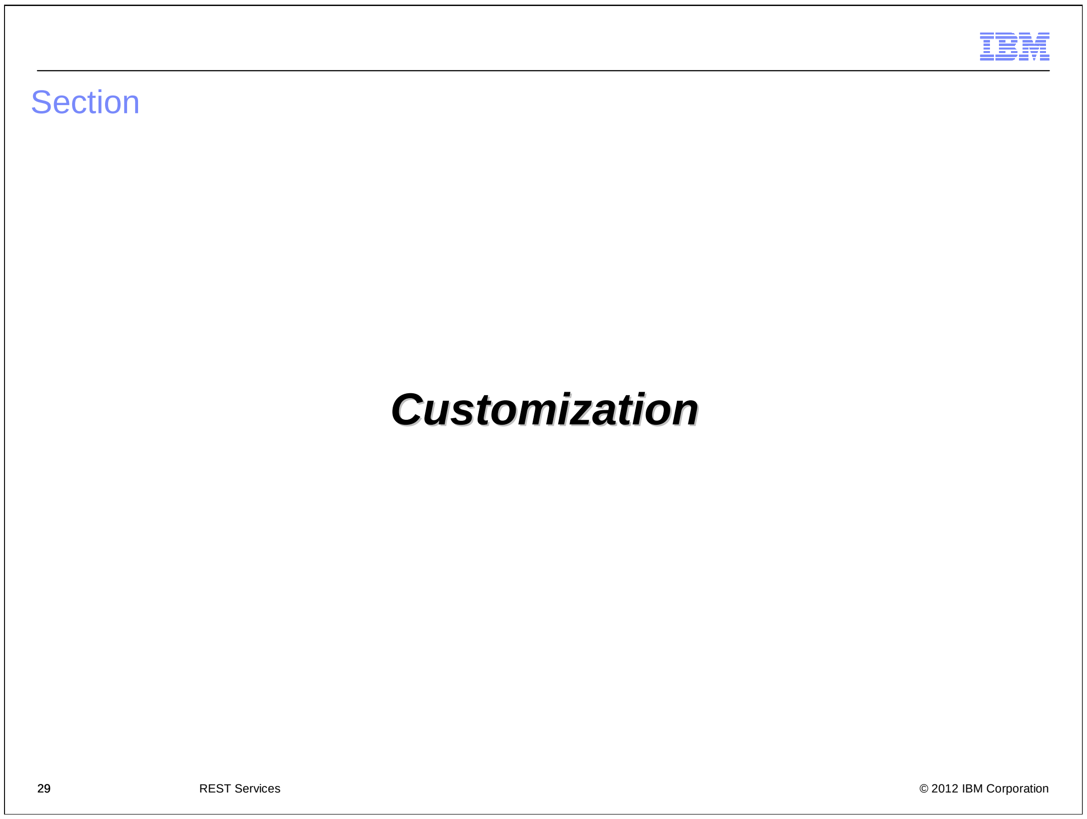
The response formatting is handled by the JSON entity provider which is a subclass of the abstract entity provider. It uses a method provided by the abstract entity provider that converts the SDO to a JSON object using the mapping files. The JSON object is then returned as the service response. For custom entity providers, the abstract entity provider also provides a JSP formatting option.



Example 2: JSON response received by client

```
{
   "orderStatus": "P",
   "orderItem": [
      {
         "productUrl": "http://localhost:8080/wcs/resources/store/10101/product/MW-0018-E203",
         "orderItemStatus": "P",
         "UOM": "C62",
         "unitPrice": "14.99000",
         "productId": "10822",
         "itemAttributes": [
         ],
         "quantity": "2.0",
         "orderItemId": "65749",
         "partNumber": "MW-0018-E203",
         "adjustment": [
            {
               "amount": "-2.85000",
               "description": "Save 10% on all orders"
            },
            {
               "amount": "-1.50000",
               "description": "Save 5% on all girls dresses and skirts"
            }
         ],
         "contractId": "10008"
      }
   ],
   "totalSalesTax": "0.00000",
   "paymentInstructionUrl": "https://localhost:8080/wcs/resources/store/10101/cart/@self/payment_instruction",
   "resourceType": "cart/Cart",
   "resourceId": "http://localhost:8080/wcs/resources/store/10101/cart/@self",
   "orderId": "18802",
```

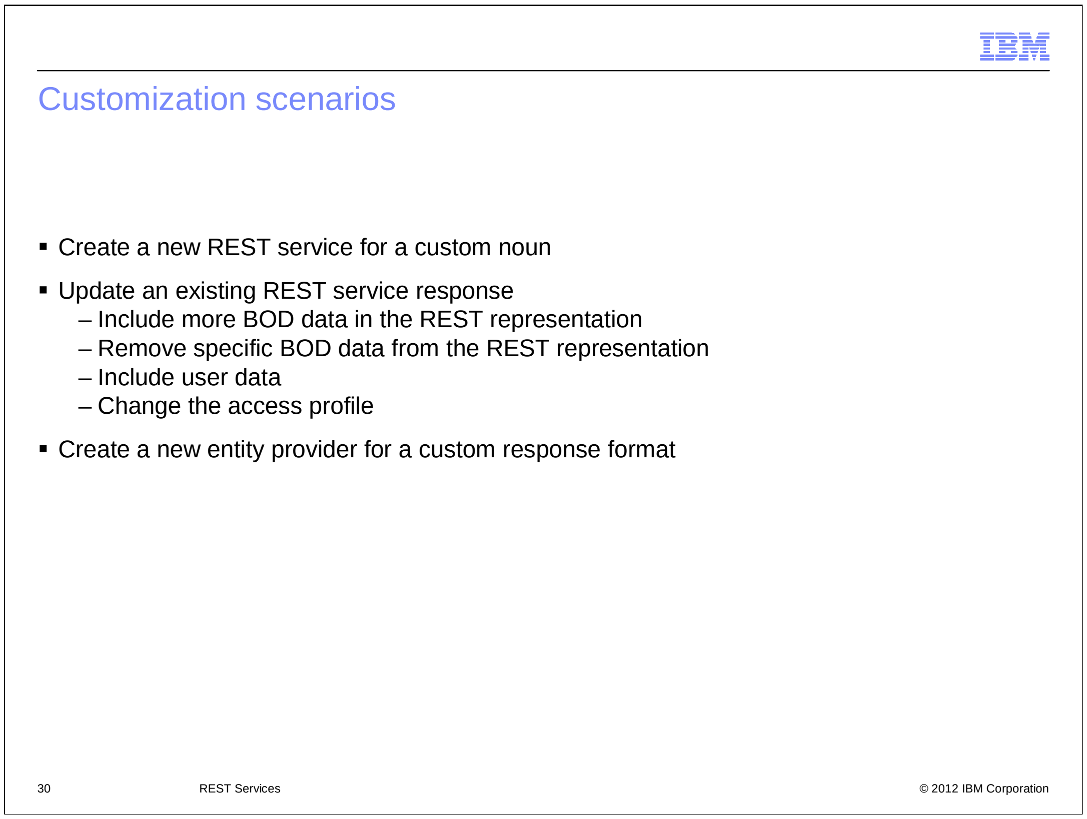
Here you see a portion of the shopping cart details in JSON format. This response is then parsed by the client and the shopping cart is displayed.



Section

Customization

This section introduces the REST services customization scenarios.



Customization scenarios

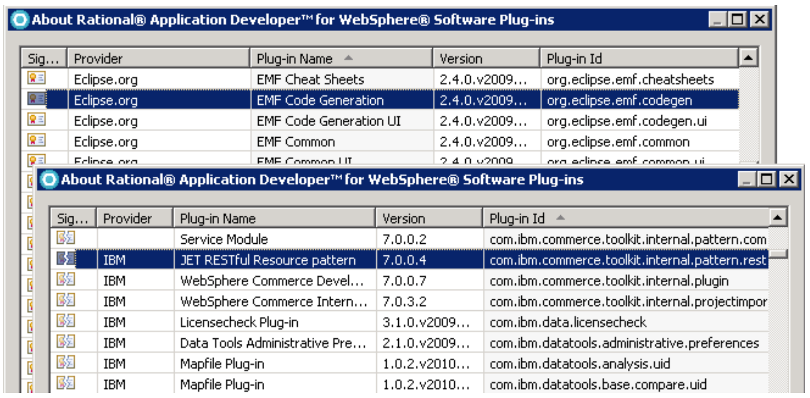
- Create a new REST service for a custom noun
- Update an existing REST service response
  - Include more BOD data in the REST representation
  - Remove specific BOD data from the REST representation
  - Include user data
  - Change the access profile
- Create a new entity provider for a custom response format

30 REST Services © 2012 IBM Corporation

There are three main areas of customization for the REST service framework. The first is to create a new REST service for a new or customized noun. The second is to modify an existing REST service response either by adding or removing data. The third is to create a new entity provider to support a custom response format.

Code generation tool

- Code generation is provided for creating and customizing REST services
- WebSphere Commerce Developer prerequisites
  - EMF Code Generation plug-in is installed
  - JET RESTful Resource pattern plug-in is installed


About Rational® Application Developer™ for WebSphere® Software Plug-ins

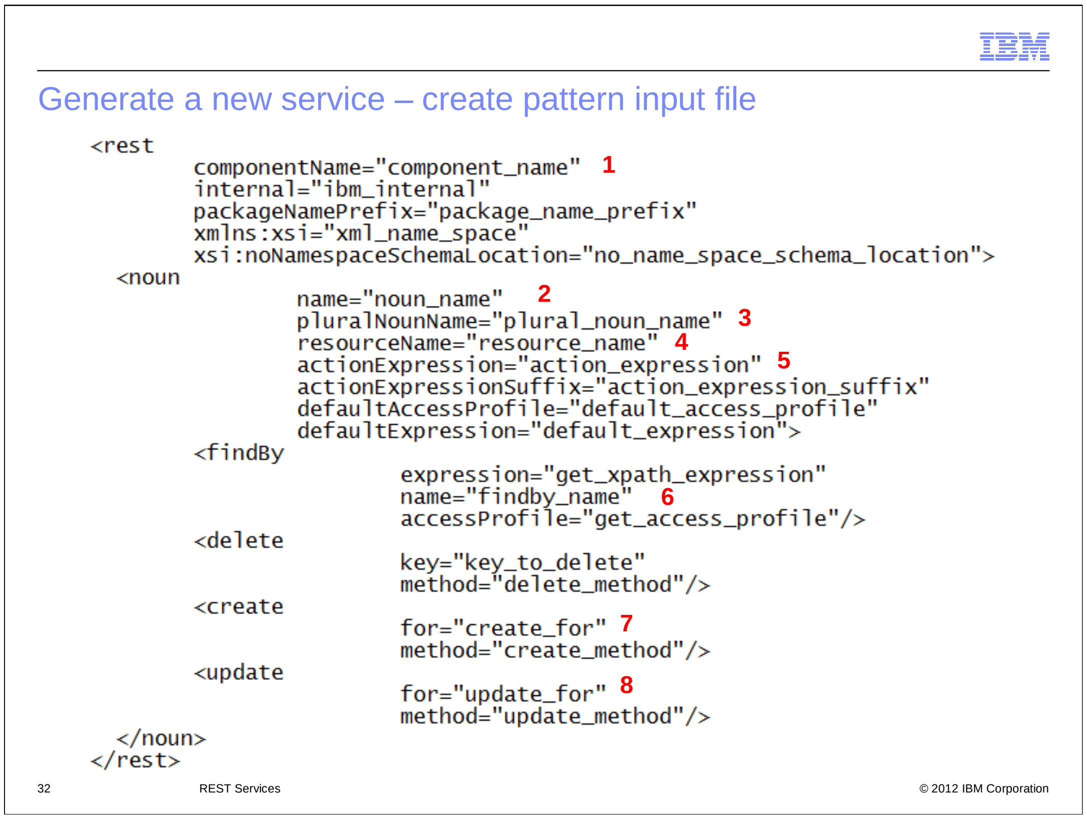
Sig…	Provider	Plug-in Name	Version	Plug-in Id
	Eclipse.org	EMF Cheat Sheets	2.4.0.v2009…	org.eclipse.emf.cheatsheets
	Eclipse.org	EMF Code Generation	2.4.0.v2009…	org.eclipse.emf.codegen
	Eclipse.org	EMF Code Generation UI	2.4.0.v2009…	org.eclipse.emf.codegen.ui
	Eclipse.org	EMF Common	2.4.0.v2009…	org.eclipse.emf.common
	Eclipse.org	EMF Common UI	2.4.0.v2009	org.eclipse.emf.common.ui

About Rational® Application Developer™ for WebSphere® Software Plug-ins

Sig…	Provider	Plug-in Name	Version	Plug-in Id
		Service Module	7.0.0.2	com.ibm.commerce.toolkit.internal.pattern.com
	IBM	JET RESTful Resource pattern	7.0.0.4	com.ibm.commerce.toolkit.internal.pattern.rest
	IBM	WebSphere Commerce Devel…	7.0.0.7	com.ibm.commerce.toolkit.internal.plugin
	IBM	WebSphere Commerce Intern…	7.0.3.2	com.ibm.commerce.toolkit.internal.projectimpor
	IBM	Licensecheck Plug-in	3.1.0.v2009…	com.ibm.data.licensecheck
	IBM	Data Tools Administrative Pre…	2.1.0.v2009…	com.ibm.datatools.administrative.preferences
	IBM	Mapfile Plug-in	1.0.2.v2010…	com.ibm.datatools.analysis.uid
	IBM	Mapfile Plug-in	1.0.2.v2010…	com.ibm.datatools.base.compare.uid

31 REST Services © 2012 IBM Corporation

Creating a new service is done using the JET code generation capability in WebSphere Commerce Developer. You need to have the plug-ins shown here installed in your development environment to generate code. You can find more information on installing plug-ins in WebSphere Commerce Information Center.



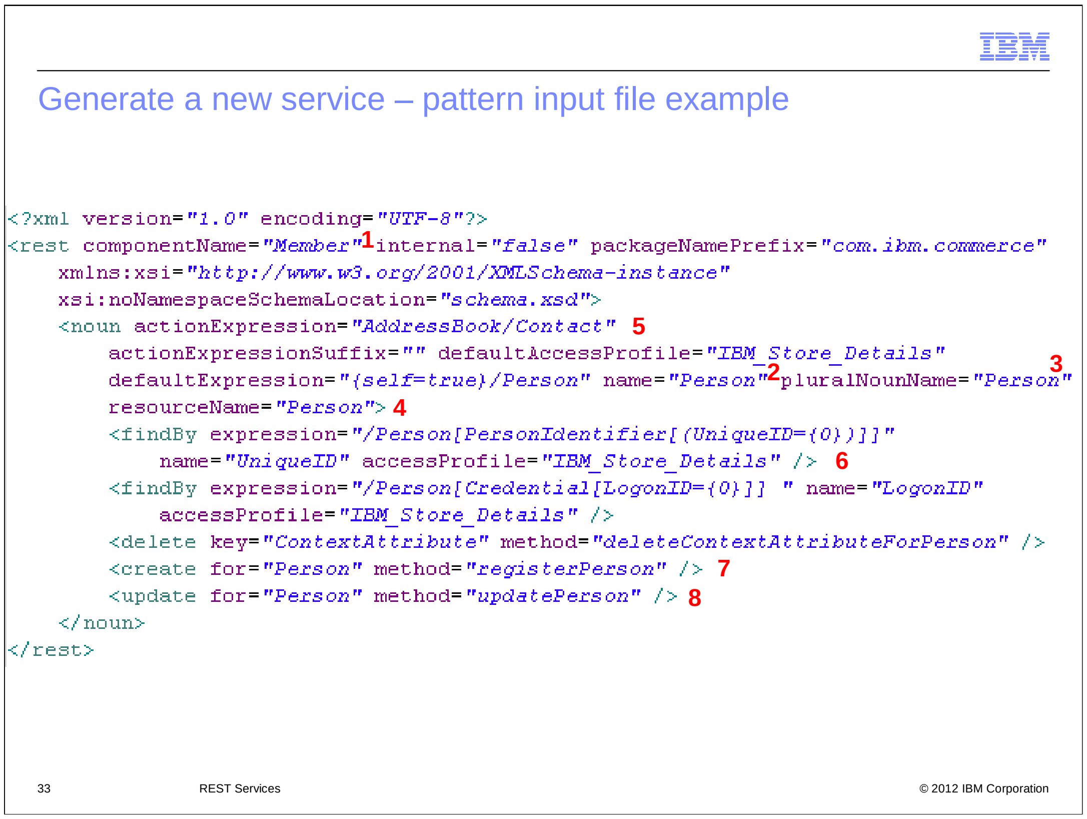
Generate a new service – create pattern input file

```
<rest
        componentName="component_name"  1
        internal="ibm_internal"
        packageNamePrefix="package_name_prefix"
        xmlns:xsi="xml_name_space"
        xsi:noNamespaceSchemaLocation="no_name_space_schema_location">
  <noun
                name="noun_name"   2
                pluralNounName="plural_noun_name"  3
                resourceName="resource_name"  4
                actionExpression="action_expression"  5
                actionExpressionSuffix="action_expression_suffix"
                defaultAccessProfile="default_access_profile"
                defaultExpression="default_expression">
        <findBy
                        expression="get_xpath_expression"
                        name="findby_name"   6
                        accessProfile="get_access_profile"/>
        <delete
                        key="key_to_delete"
                        method="delete_method"/>
        <create
                        for="create_for"  7
                        method="create_method"/>
        <update
                        for="update_for"  8
                        method="update_method"/>
  </noun>
</rest>
```

32 REST Services © 2012 IBM Corporation

The first step in code generation is creating a pattern file as the input to the code generator. This slide shows the template for the XML pattern file.

The number one is the component name of the web service that you want to create a REST service for. Numbers two and three are the name of the noun within the component that is modified through the REST service. In Feature Pack 4, the convention is to use the singular version of the noun name so both of these attributes can be set to the same value. The number four is the name of the REST resource you want to create. Number five is the action expression for the noun or noun part that corresponds to the resource. The numbers six, seven and eight are the client library methods to call for the different resource actions. Six is the findBy method name used for GET requests. Seven is the creation method used for POST requests and eight is the update method used for PUT requests.



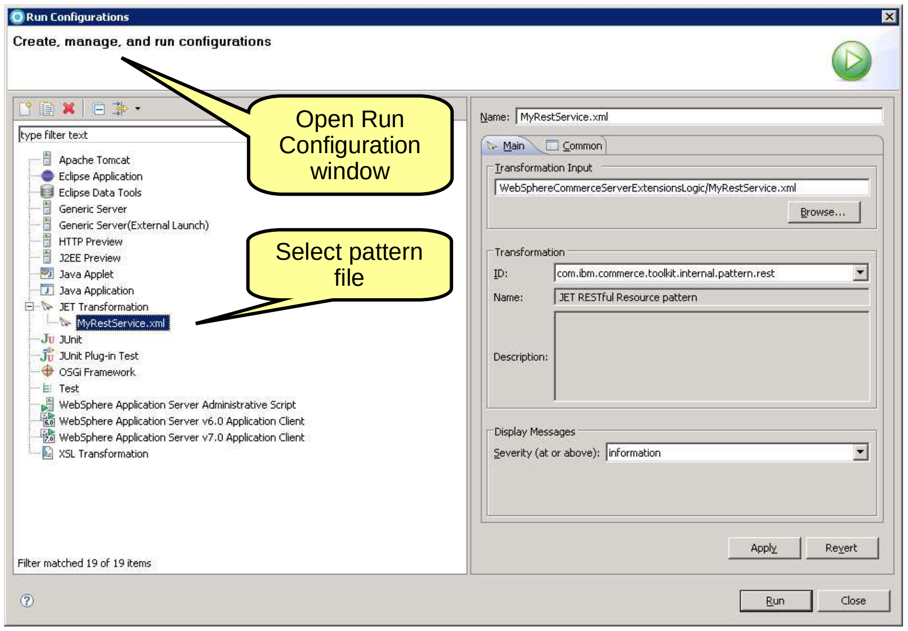
Generate a new service – pattern input file example

```
<?xml version="1.0" encoding="UTF-8"?>
<rest componentName="Member"1internal="false" packageNamePrefix="com.ibm.commerce"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="schema.xsd">
    <noun actionExpression="AddressBook/Contact" 5
        actionExpressionSuffix="" defaultAccessProfile="IBM_Store_Details"         3
        defaultExpression="{self=true}/Person" name="Person"2pluralNounName="Person"
        resourceName="Person"> 4
        <findBy expression="/Person[PersonIdentifier[(UniqueID={0})]]"
            name="UniqueID" accessProfile="IBM_Store_Details" /> 6
        <findBy expression="/Person[Credential[LogonID={0}]] " name="LogonID"
            accessProfile="IBM_Store_Details" />
        <delete key="ContextAttribute" method="deleteContextAttributeForPerson" />
        <create for="Person" method="registerPerson" /> 7
        <update for="Person" method="updatePerson" /> 8
    </noun>
</rest>
```

33 REST Services © 2012 IBM Corporation

Here is an example of a complete pattern file for the Person noun. The numbers highlight the sections in the pattern described on the previous slide.

Generate a new service – generate code from pattern file



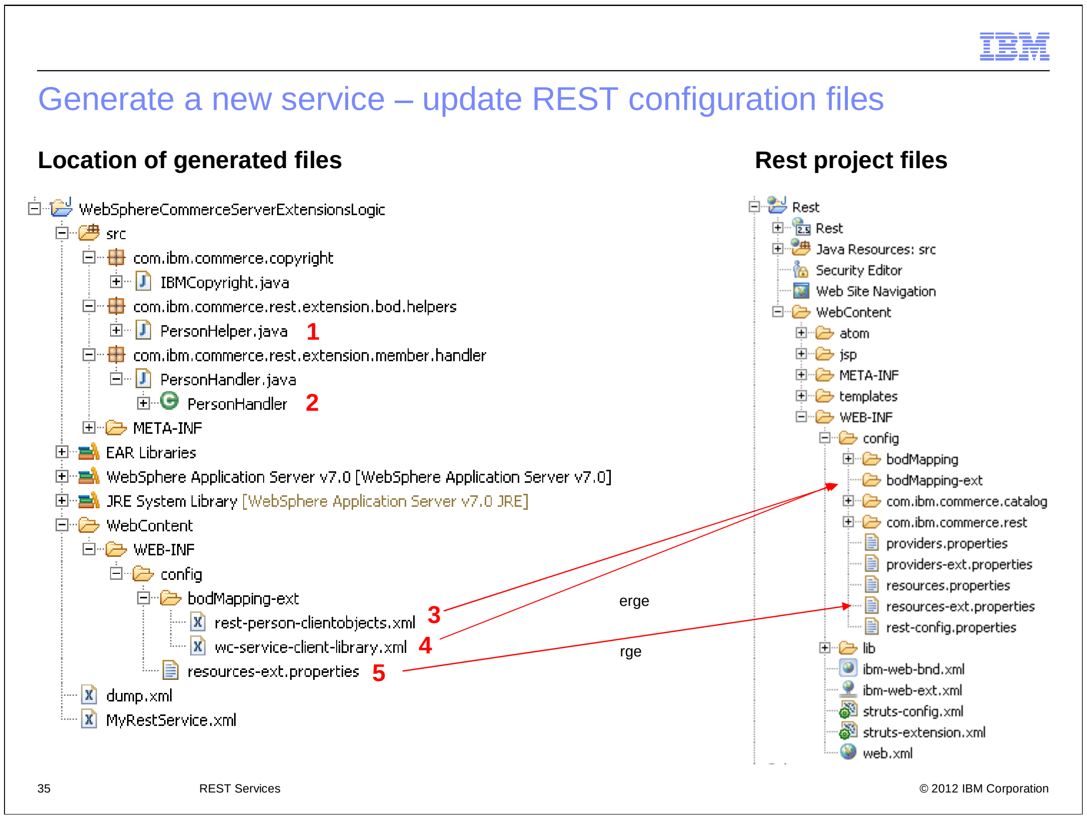
Name: MyRestService.xml

Transformation Input: WebSphereCommerceServerExtensionsLogic/MyRestService.xml

Transformation
ID: com.ibm.commerce.toolkit.internal.pattern.rest
Name: JET RESTful Resource pattern
Description:

Display Messages
Severity (at or above): information

Once your pattern file is ready, launch the Run Configuration window in WebSphere Commerce Developer and select your pattern file. Run the code generation. The details of the code generation process can be found in the WebSphere Commerce Information Center.



Generate a new service – update REST configuration files

Location of generated files

```
WebSphereCommerceServerExtensionsLogic
  src
    com.ibm.commerce.copyright
      IBMCopyright.java
    com.ibm.commerce.rest.extension.bod.helpers
      PersonHelper.java  1
    com.ibm.commerce.rest.extension.member.handler
      PersonHandler.java
        PersonHandler  2
    META-INF
  EAR Libraries
  WebSphere Application Server v7.0 [WebSphere Application Server v7.0]
  JRE System Library [WebSphere Application Server v7.0 JRE]
  WebContent
    WEB-INF
      config
        bodMapping-ext
          rest-person-clientobjects.xml  3
          wc-service-client-library.xml  4
        resources-ext.properties  5
  dump.xml
  MyRestService.xml
```

erge

rge

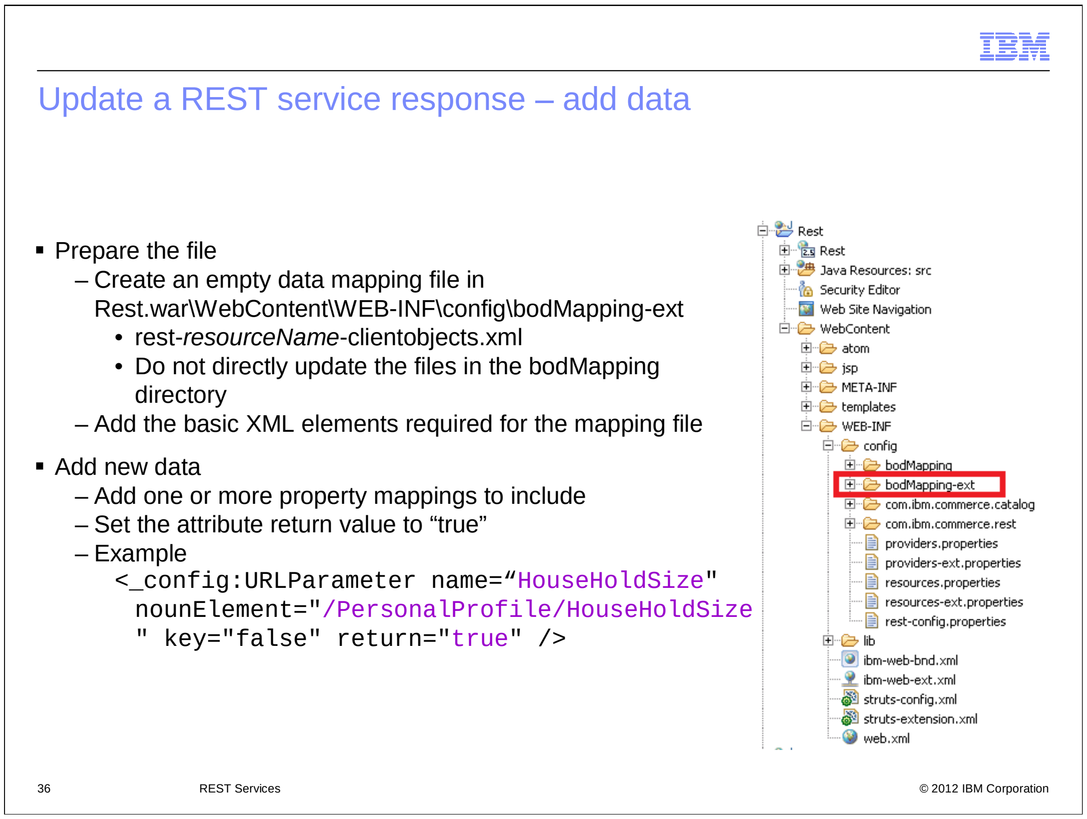
Rest project files

```
Rest
  Rest
  Java Resources: src
  Security Editor
  Web Site Navigation
  WebContent
    atom
    jsp
    META-INF
    templates
    WEB-INF
      config
        bodMapping
        bodMapping-ext
        com.ibm.commerce.catalog
        com.ibm.commerce.rest
        providers.properties
        providers-ext.properties
        resources.properties
        resources-ext.properties
        rest-config.properties
      lib
      ibm-web-bnd.xml
      ibm-web-ext.xml
      struts-config.xml
      struts-extension.xml
      web.xml
```

35 REST Services © 2012 IBM Corporation

The left side of this screen shows the files generated using the sample Person pattern file.

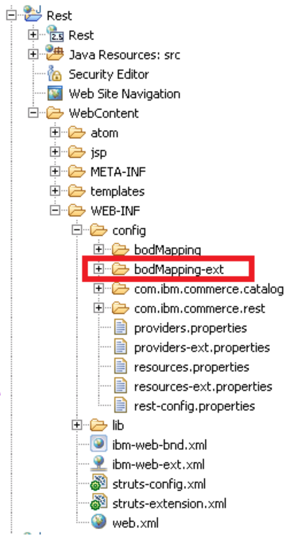
The numbers one and two indicate the new resource handler and a helper file for calling the corresponding OAGIS service. Numbers three, four and five indicate configuration files including the mapping file and the resource handler definition file. These files need to be merged with the existing files in the Rest WAR as shown on the slide.



Update a REST service response – add data

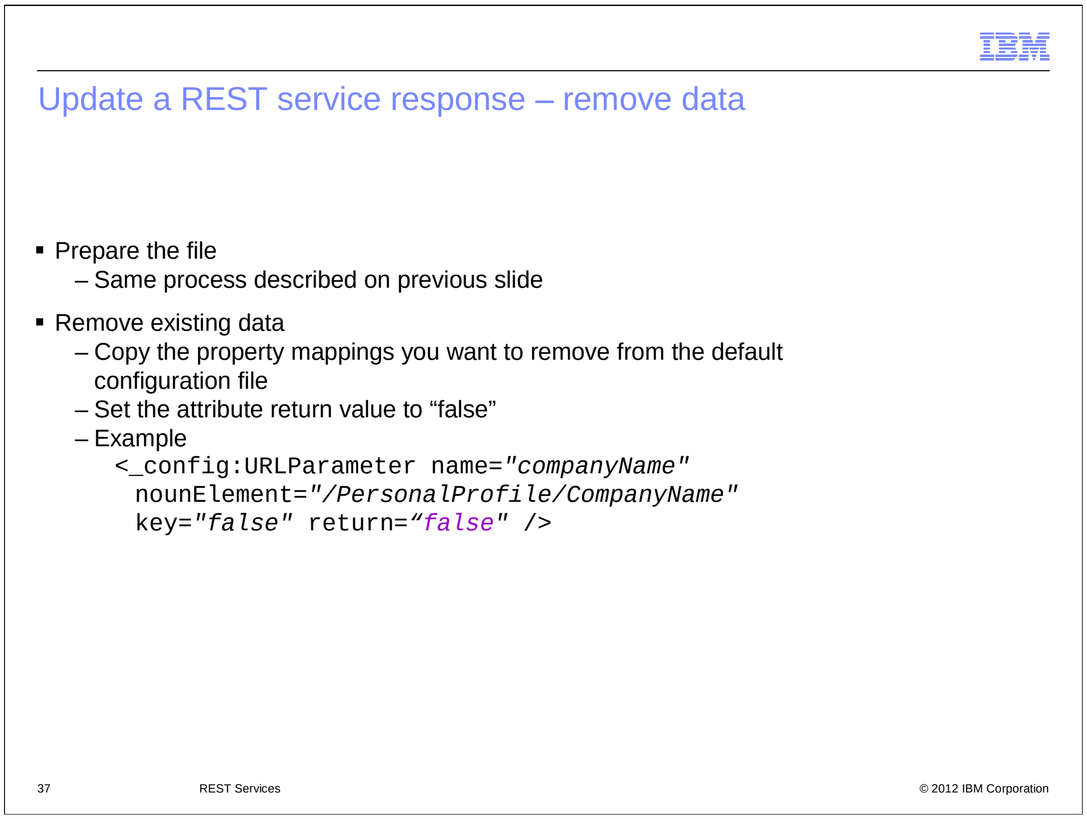
- Prepare the file
  - Create an empty data mapping file in Rest.war\WebContent\WEB-INF\config\bodMapping-ext
    - rest-*resourceName*-clientobjects.xml
    - Do not directly update the files in the bodMapping directory
  - Add the basic XML elements required for the mapping file
- Add new data
  - Add one or more property mappings to include
  - Set the attribute return value to "true"
  - Example

```
<_config:URLParameter name="HouseHoldSize"
  nounElement="/PersonalProfile/HouseHoldSize
  " key="false" return="true" />
```



36 REST Services © 2012 IBM Corporation

The second main area of customization is modifying the response of an existing REST service. Whether you are adding or removing data, the first step is to create an extension mapping file. In this file you only need the elements you want to change. To add new data, you add one or more property mappings and set the return attribute for each to true.



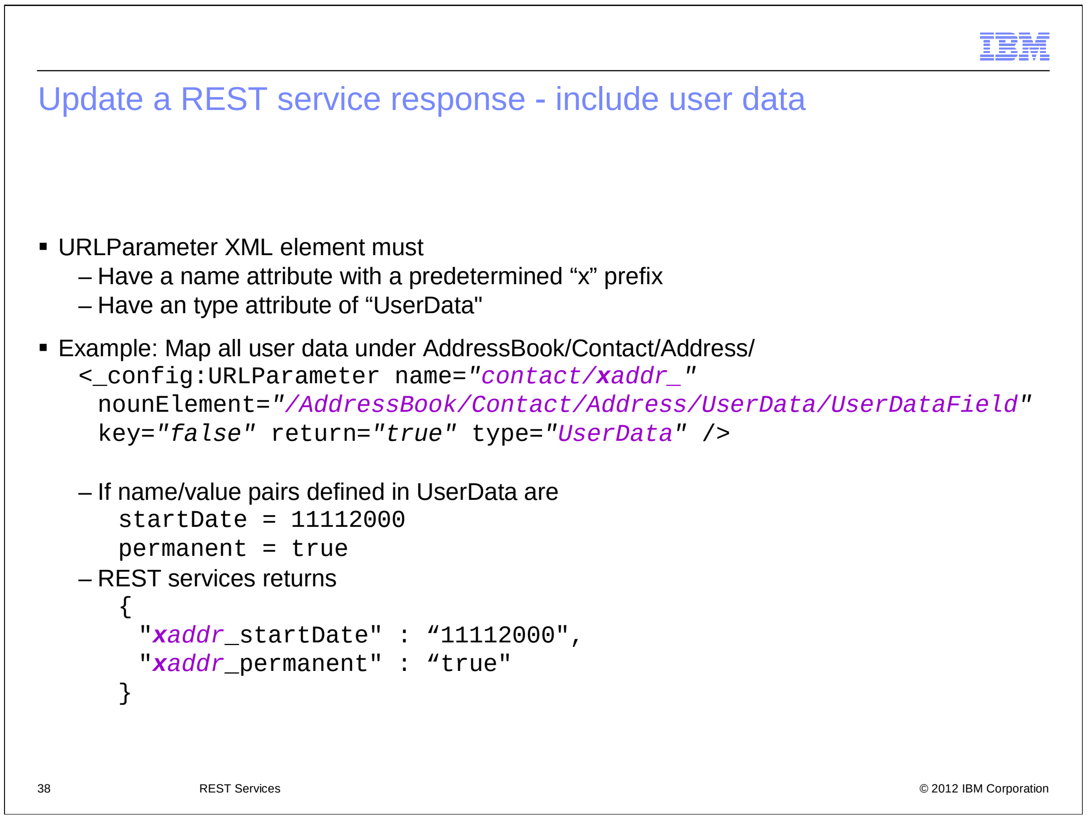
Update a REST service response – remove data

- Prepare the file
  - Same process described on previous slide
- Remove existing data
  - Copy the property mappings you want to remove from the default configuration file
  - Set the attribute return value to “false”
  - Example

```
<_config:URLParameter name="companyName"
  nounElement="/PersonalProfile/CompanyName"
  key="false" return=“false" />
```

37 REST Services © 2012 IBM Corporation

To remove data from the REST service response, you follow the same steps as the previous slide including adding each property to be removed into the extension mapping file. You then set the return attribute to false and each element specified are ignored when building the JSON response.



Update a REST service response - include user data

- URLParameter XML element must
  - Have a name attribute with a predetermined “x” prefix
  - Have an type attribute of “UserData"
- Example: Map all user data under AddressBook/Contact/Address/

```
<_config:URLParameter name="contact/xaddr_"
  nounElement="/AddressBook/Contact/Address/UserData/UserDataField"
  key="false" return="true" type="UserData" />
```

  - If name/value pairs defined in UserData are

```
startDate = 11112000
permanent = true
```

  - REST services returns

```
{
 "xaddr_startDate" : “11112000",
 "xaddr_permanent" : “true"
}
```

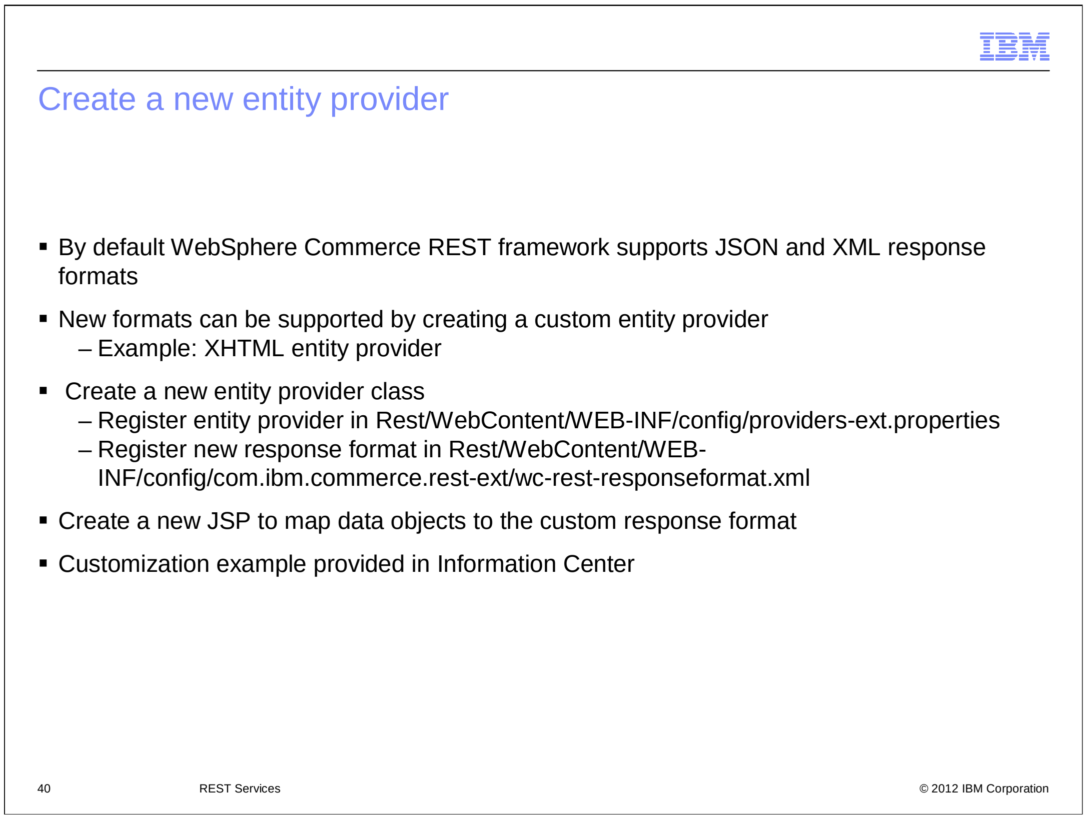
38 REST Services © 2012 IBM Corporation

Another way to update a REST service response is to include user data. The existing WebSphere Commerce REST services return the user data fields for their corresponding nouns. If you generate a new REST service, you need to populate the mapping file including any user data contained in your custom noun. A code snippet for adding user data to your mapping file is shown on the slide. Notice that the type attribute indicates this is a user data field. The remaining code examples show sample user data name value pairs and the resulting JSON format for the data.

Update a REST service response - change the access profile

39 REST Services © 2012 IBM Corporation

If you have created a custom access profile or search profile to return extra data, you need to configure the REST service to use the new profiles. The first step is to create an extension resource configuration file to define the new access profile mappings. You should not change the default file directly. Once the extension file has been created, copy over the GetUri elements that have changes in their access profiles. If you are adding new mappings, you can use those in the default file as a template. Set the accessProfile attribute to the name of your new access profile. If there is an associated search profile that has been customized, specify the new search profile name as well. Once this change is complete, your REST service will use the updated access profile mapping and the custom data is returned in the BOD response. If necessary, add additional data to the BOD mapping file as described in the previous slides.



Create a new entity provider

- By default WebSphere Commerce REST framework supports JSON and XML response formats
- New formats can be supported by creating a custom entity provider
  - Example: XHTML entity provider
- Create a new entity provider class
  - Register entity provider in Rest/WebContent/WEB-INF/config/providers-ext.properties
  - Register new response format in Rest/WebContent/WEB-INF/config/com.ibm.commerce.rest-ext/wc-rest-responseformat.xml
- Create a new JSP to map data objects to the custom response format
- Customization example provided in Information Center

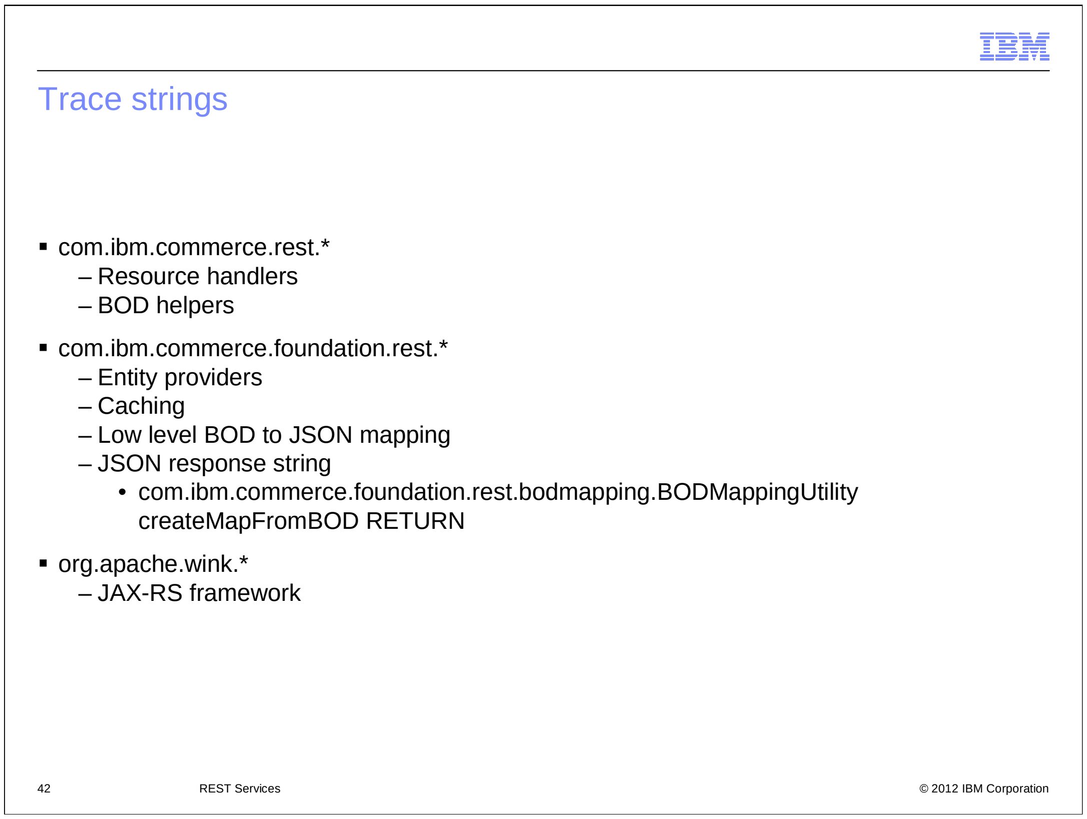
40 REST Services © 2012 IBM Corporation

The final customization scenario is creating a new entity provider. The WebSphere Commerce REST framework supports JSON and XML response formats. If your client requires a different format, you need to create a new entity provider. There is no code generation for creating an entity provider. You extend the AbstractEntityProvider class and provide your custom mapping in one or more JSP files. To make the new entity provider available, the new provider class and the new response format need to be registered in the files shown on the slide. Detailed steps and sample code for this customization are provided in the Information Center.

Section

Problem determination

This section provides some problem determination tips.



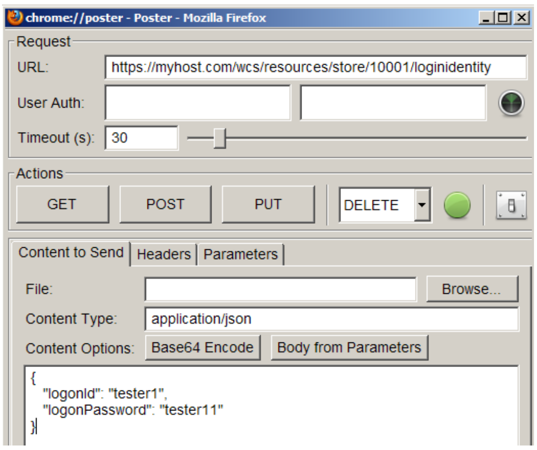
Trace strings

- com.ibm.commerce.rest.*
  - Resource handlers
  - BOD helpers
- com.ibm.commerce.foundation.rest.*
  - Entity providers
  - Caching
  - Low level BOD to JSON mapping
  - JSON response string
    - com.ibm.commerce.foundation.rest.bodmapping.BODMappingUtility createMapFromBOD RETURN
- org.apache.wink.*
  - JAX-RS framework

42 REST Services © 2012 IBM Corporation

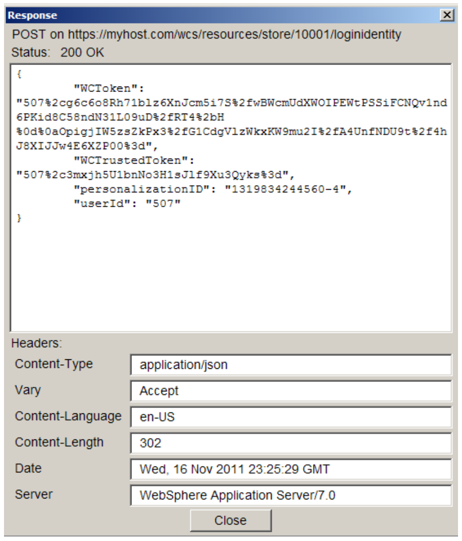
There are several different trace strings that you can use to help debug problems with REST services. The different trace options and the part of the framework they support are shown on the slide.

Testing REST services



URL: https://myhost.com/wcs/resources/store/10001/loginidentity
User Auth:
Timeout (s): 30
File:
Content Type: application/json
Content Options:

```
{
  "logonId": "tester1",
  "logonPassword": "tester11"
}
```



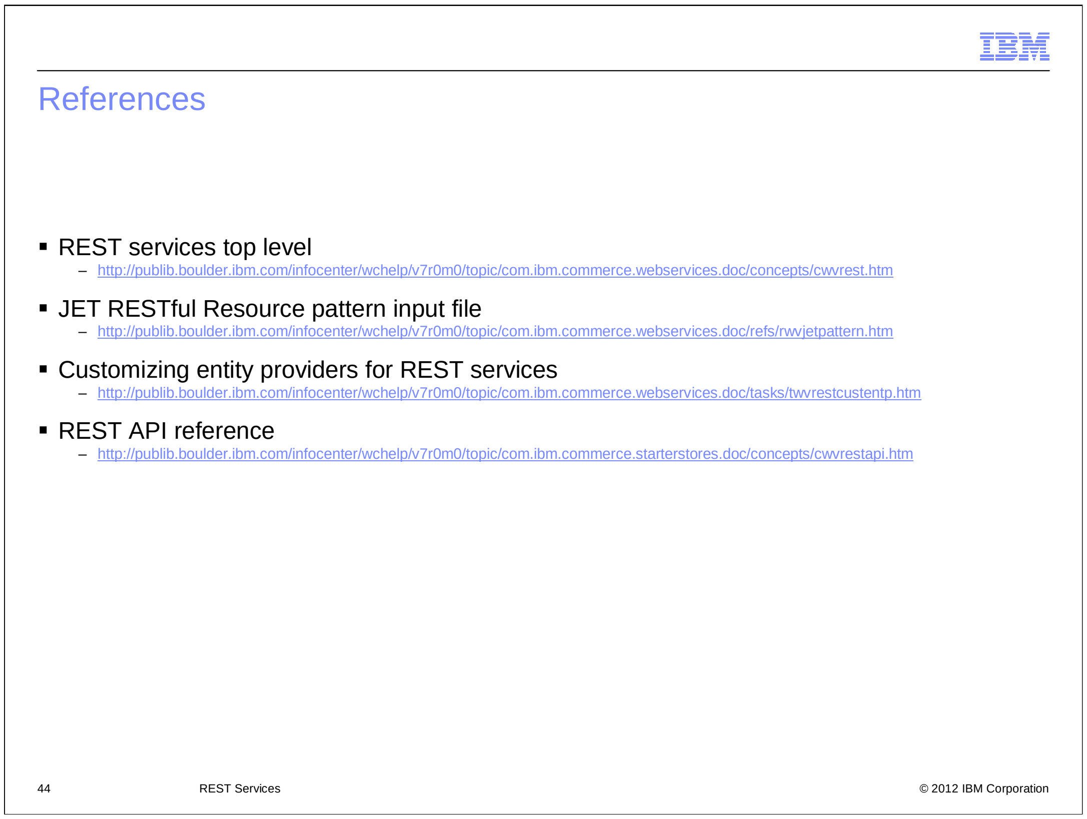
POST on https://myhost.com/wcs/resources/store/10001/loginidentity
Status: 200 OK

```
{
        "WCToken":
"507%2cg6c6o8Rh71blz6XnJcm5i7S%2fwBWcmUdXWOIPEWtPSSiFCNQv1nd
6PKid8C58ndN31L09uD%2fRT4%2bH
%0d%0aOpigjIW5zsZkPx3%2fG1CdgVlzWkxKW9mu2I%2fA4UnfNDU9t%2f4h
J8XIJJw4E6XZP00%3d",
        "WCTrustedToken":
"507%2c3mxjh5U1bnNo3H1sJlf9Xu3Qyks%3d",
        "personalizationID": "1319834244560-4",
        "userId": "507"
}
```

Content-Type	application/json
Vary	Accept
Content-Language	en-US
Content-Length	302
Date	Wed, 16 Nov 2011 23:25:29 GMT
Server	WebSphere Application Server/7.0

- “Poster” add-on for Firefox
- Issue services requests and examine the response

The Poster add-on for Firefox is helpful for testing REST services. You can manually enter a service request including JSON content and then invoke the service. The JSON response is returned along with the response headers.

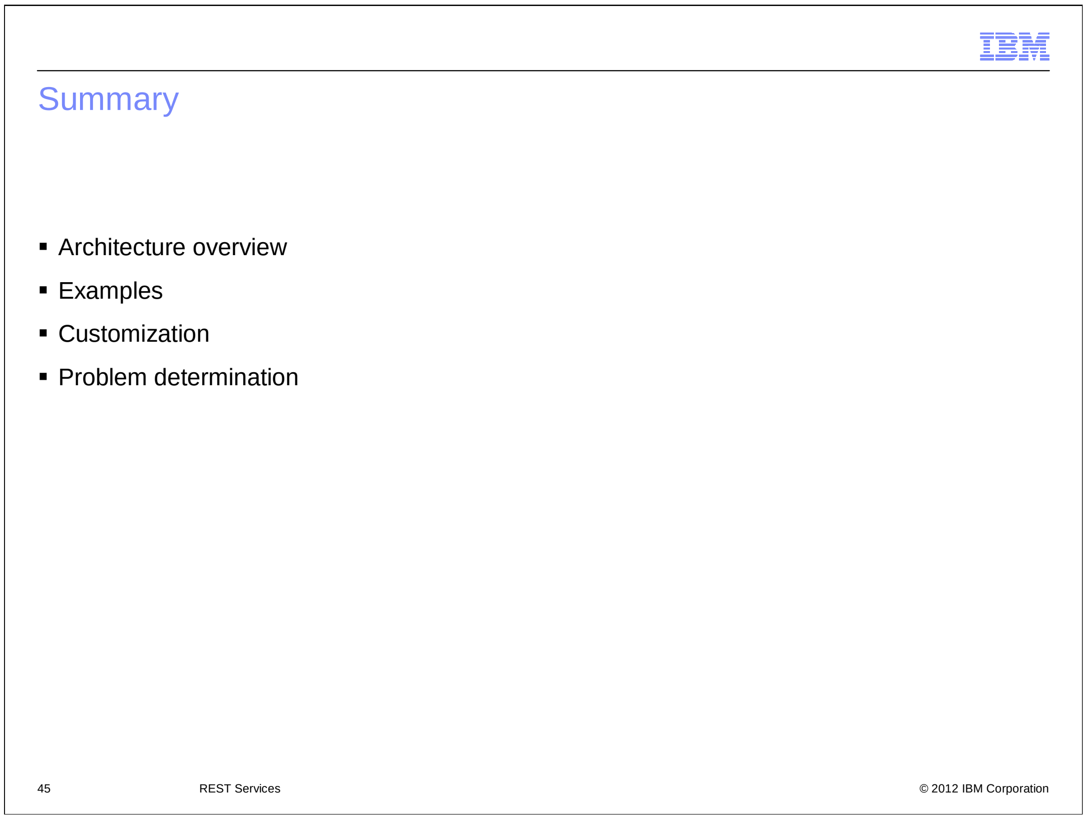


References

- REST services top level
  - http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.webservices.doc/concepts/cwvrest.htm
- JET RESTful Resource pattern input file
  - http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.webservices.doc/refs/rwvjetpattern.htm
- Customizing entity providers for REST services
  - http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.webservices.doc/tasks/twvrestcustentp.htm
- REST API reference
  - http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.starterstores.doc/concepts/cwvrestapi.htm

44 REST Services © 2012 IBM Corporation

This slide contains some references for further reading on this topic.

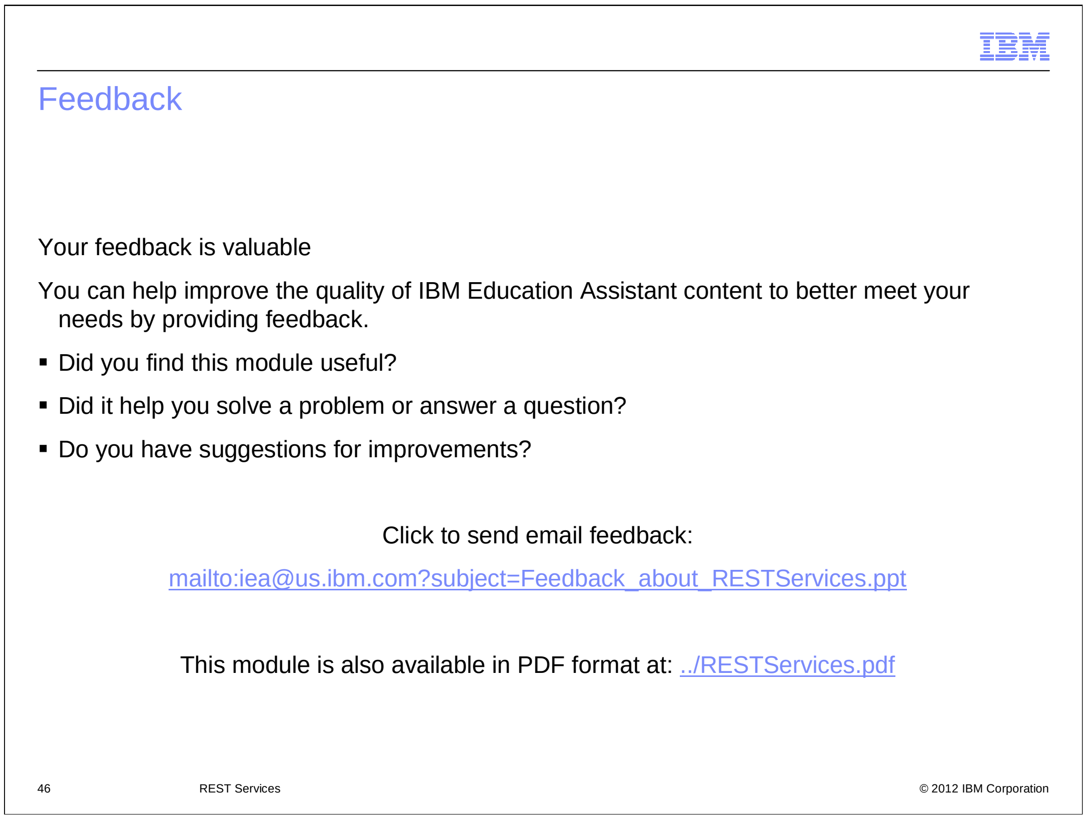


Summary

- Architecture overview
- Examples
- Customization
- Problem determination

45 REST Services © 2012 IBM Corporation

This presentation began with an overview of the REST services solution architecture. Following the architecture overview was a series of examples that demonstrated the concepts covered in the overview. This presentation concluded with some example customization scenarios and problem determination tips.



Feedback

Your feedback is valuable

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

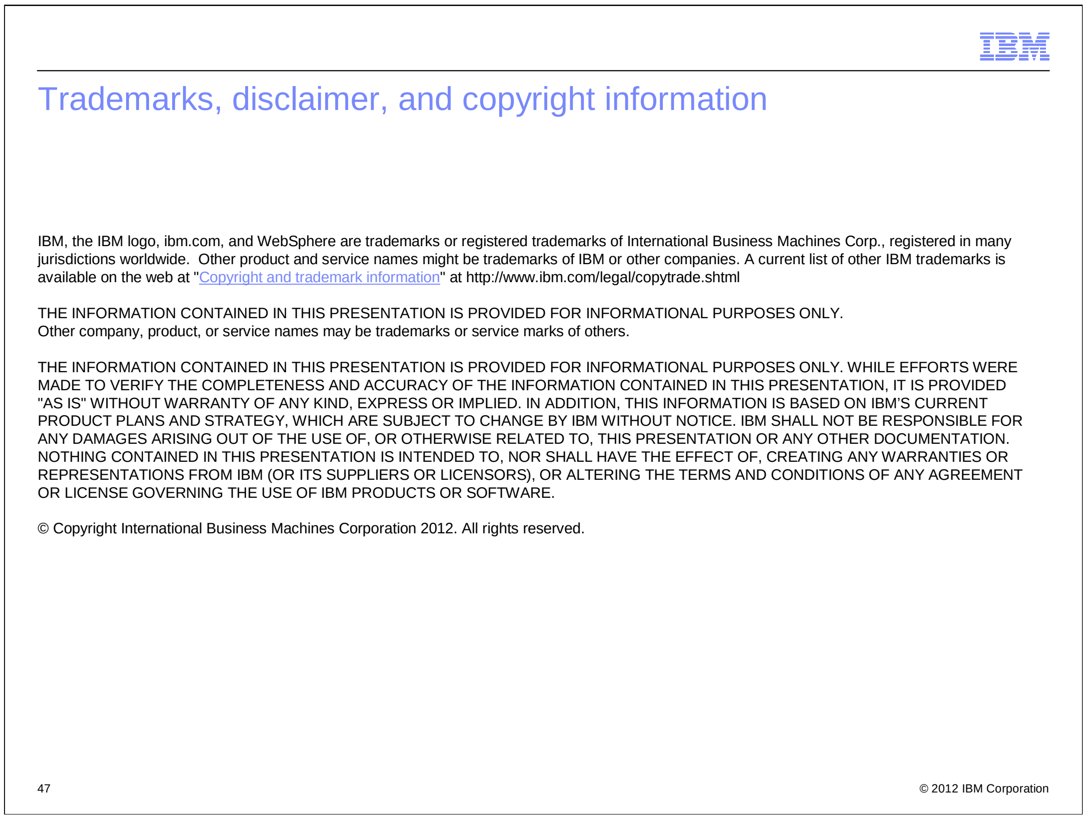
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

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

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

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



Trademarks, disclaimer, and copyright information