AD1238: REST Services in Domino - Key to modern Web Applications

Bernd Hort, assono GmbH - February 1, 2016

Connect 2016

The Premier Social Business and Digital Experience Conference

#ibmconnect

Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Questions and Answers

Any code in this presentation and in the sample database is available under the Apache Software License v2!

Bernd Hort, assono GmbH

- IBM® Notes® Developer/Designer since Notes 3.3
- OpenNTF Contributer: assono Framework 2
  - An OOP LotusScript Framework
- Lately specialized in XPages, web and mobile development
- Working for assono GmbH, Germany
- Blog http://blog.assono.de
- Twitter @BerndHort

REST in a Nutshell

- Representational state transfer (REST) is an architectural style bases on web standards for accessing resources
- RESTful Web Services implements access and modification of resources
- Mostly using HTTP
- Resources may have different representations, e.g. text, xml, json etc.
  - The rest client can ask for specific representation via the HTTP protocol (content negotiation).
- No standard!
- The WSDL equivalent Web Application Description Language (WADL) is mostly optional
  - Hopefully the documentation is good ;-)

REST in a Nutshell – Nouns and Verbs

- Resources are identified via a URI - Nouns
- Actions on resources uses the HTTP methods - Verbs

Resource	*Get*	*Post*	*Put*	*Delete*
Collections of resources http://yourserver/path/sessions	**Lists** all resources	**Create** a new entry in the collection.	**Replace** the entire collection with another collection.	**Delete** the entire collection
Single resource http://yourserver/path/sessions/AD1238	**Retrieve** the details of a given resource	*Normally not implemented*	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	**Delete** the specified resource

Also known as create, read, update and delete (CRUD)

IBM Domino Access Services (DAS)

Connect 2016

The Premier Social Business and Digital Experience Conference

#ibmconnect

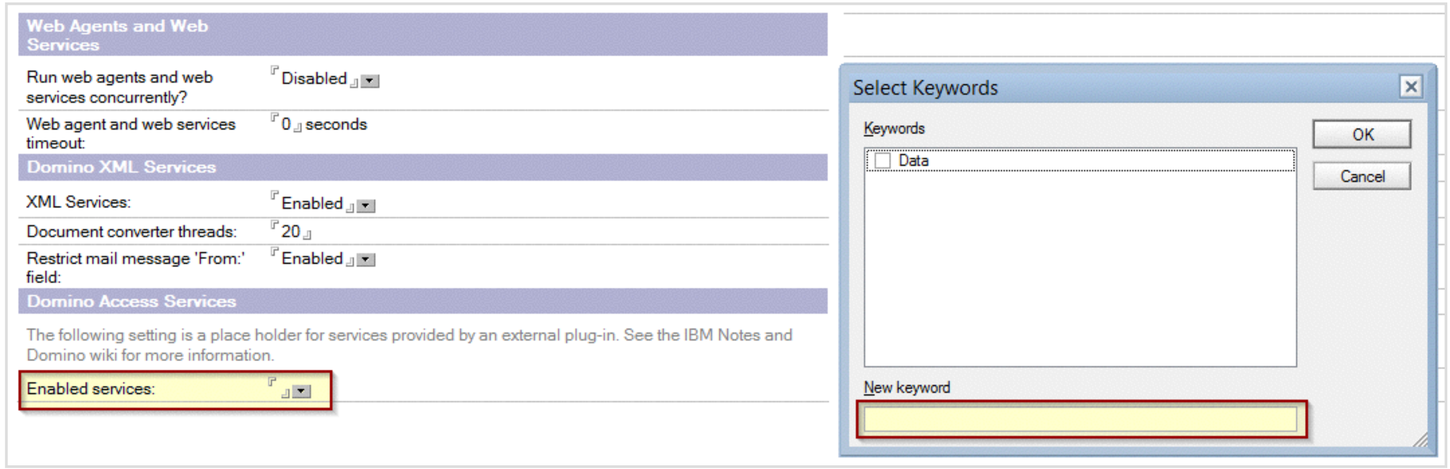
IBM Domino Access Services (DAS)

- IBM Domino Access Services delivers resources via HTTP
- Domino Core Service
  - Resources which are not specific to any other service
    - e.g. Password statistics resource
- Domino Data Service
  - Resources which are a Notes database or part of a Notes database
- Domino Calendar Service
  - Resources to access calendars on a Domino server
- Domino Mail Service
  - Resources to access mail files on a Domino server
  - Part of the Extension Library 9.0.1

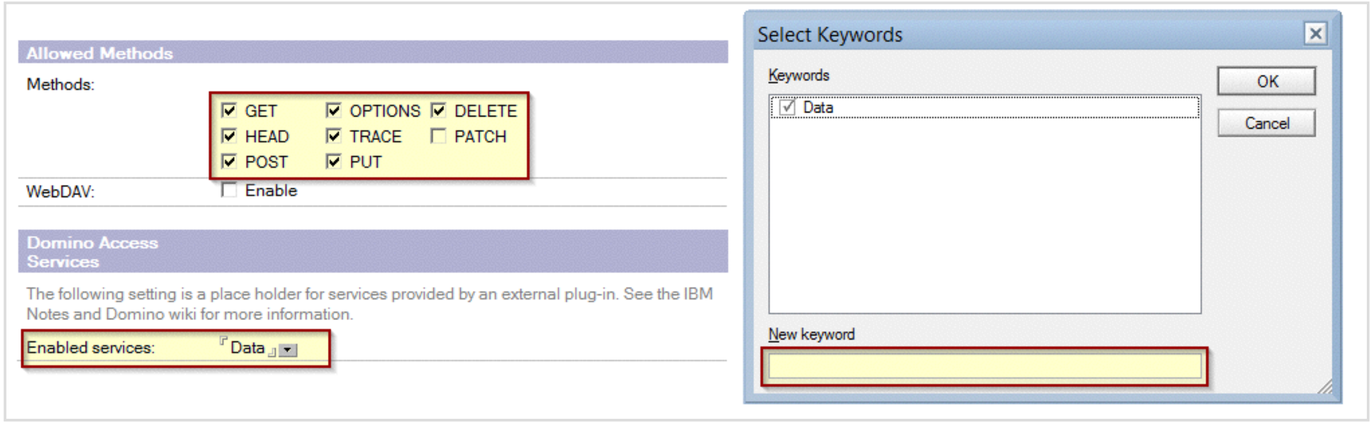
Links to the IBM Notes and Domino Application Development wiki

How to enable Domino Access Services

- On the Server document
  - Internet Protocols… \ Domino Web Engine



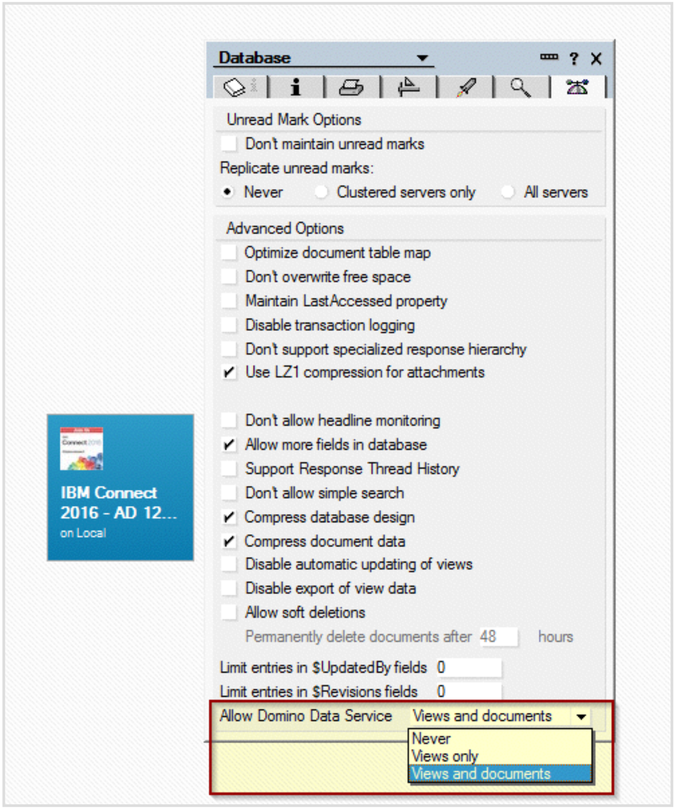
- Or in a Internet Site Web document
  - Configuration



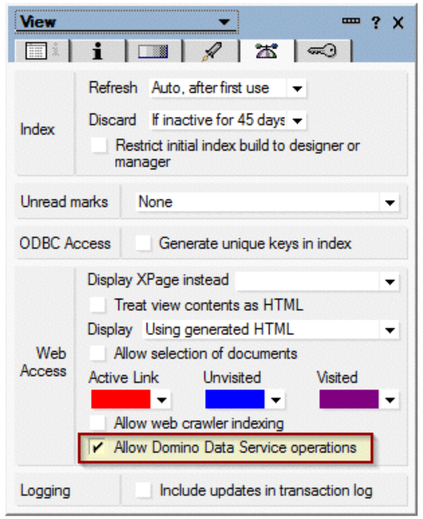
- Use „New keyword“ for additional services like „Calendar“, „Mail“

How to enable Domino Access Services

- In the database properties
  - Default setting „Never“



- In the view properties
  - Not enabled by default



Make Every **Moment** Count

Accessing Domino Data Service

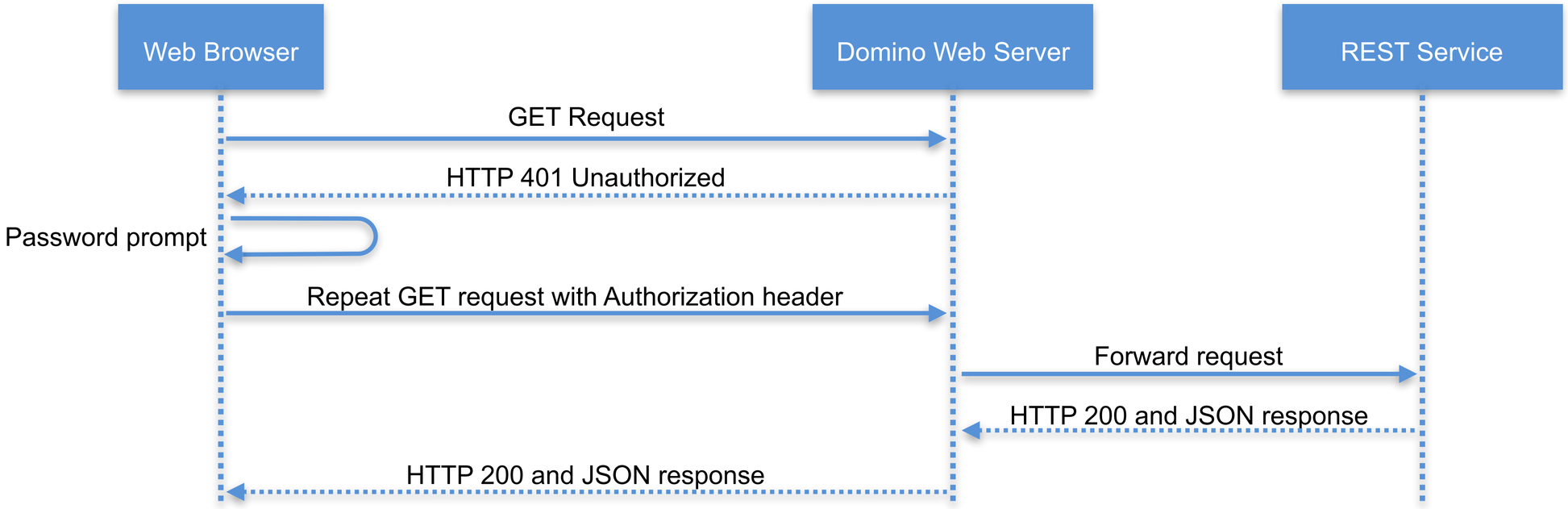
- The URL specifies the database and the kind of resource
- Document resource
  - /{database}/api/data/documents/unid/{docunid}
- Document collection resource
  - /{database}/api/data/documents
- View/folder collection resource
  - /{database}/api/data/collections
- View/folder entries resource
  - /{database}/api/data/collections/name/{name}

Authenticating Domino REST Service Requests

- REST Services uses HTTP security features for authentication
- On Domino this could be
  - Basic Authentication
  - Session Based Authentication
  - SPNEGO (single sign-on with Windows)
  - SAML (available since Domino 9.0)
  - Various third-party single sign-on (SSO) extensions (for example, CA SiteMinder)
- Good article on the IBM Notes and Domino Application Development wiki
  - Authenticating Domino REST Service Requests

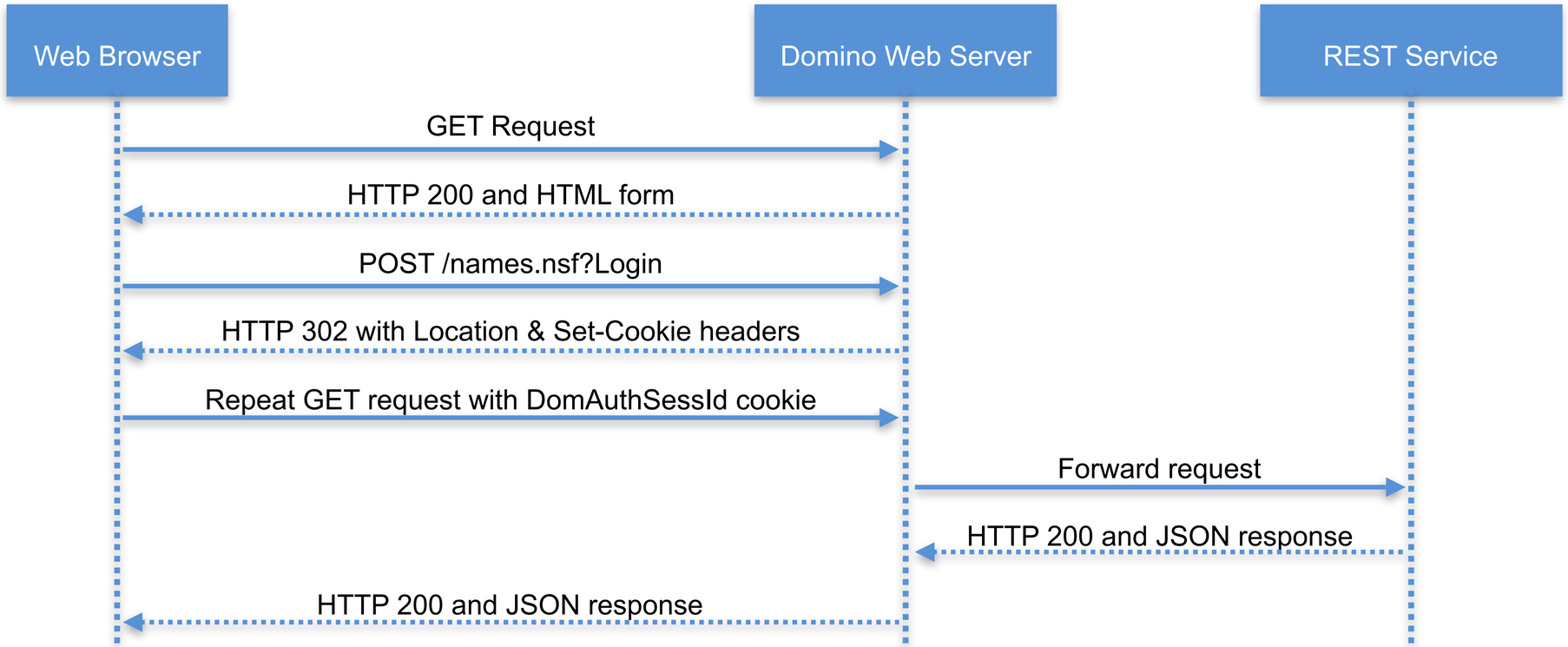
Basic Authentication

- Username and encoded password are send as part of the header
  - Always use HTTPS with basic authentication
- Authentication sequence



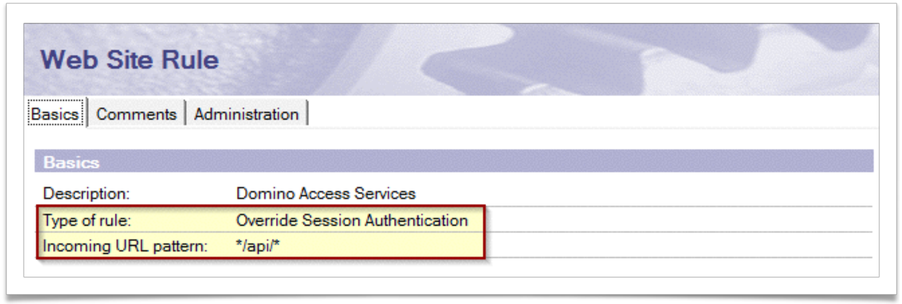
Session Based Authentication

- Session cookie are send with every request
  - Always use HTTPS with basic authentication
- Authentication sequence



Issues with Session Based Authentication

- A Service Consumer has to check for the HTTP Return Code and the content type to check for missing authentication
- For Non-Web-Browser Service Consumer the handling of the DomAuthSessId cookie means additional coding
- Solution: Web Site Rule – Override Session Authentication
  - Forces all requests for a given URL pattern to use Basic Authentication
  - URL pattern: */api/*



HTTP Return Codes

- List from Wikipedia:
- 1xx Informational Responses
- 2xx Successful Responses
- 3xx Redirection Messages
- 4xx Client Error Responses
- 5xx Server Error Responses

HTTP Return Codes – 2xx Successful Responses

- 200 OK
  - Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.
- 201 Created
  - The request has been fulfilled and resulted in a new resource being created.
- 204 No Content
  - The server successfully processed the request, but is not returning any content.

HTTP Return Codes – 3xx Redirection Messages

- 301 Moved Permanently
  - This and all future requests should be directed to the given URI.
- 302 Found
  - This response code means that URI of requested resource has been changed temporarily. New changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

HTTP Return Codes – 4xx Client Error Responses

- 400 Bad Request
  - "The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).“
    - This return code should be used if no other code fits.
- 401 Unauthorized
  - Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.
- 403 Forbidden
  - The request was a valid request, but the server is refusing to respond to it. Unlike a 401 Unauthorized response, authenticating will make no difference.

HTTP Return Codes – 4xx Client Error Responses

- 404 Not Found
  - The requested resource could not be found but may be available again in the future.
- 409 Conflict
  - Indicates that the request could not be processed because of conflict in the request, such as an edit conflict in the case of multiple updates.

HTTP Return Codes – 5xx Server Error Responses

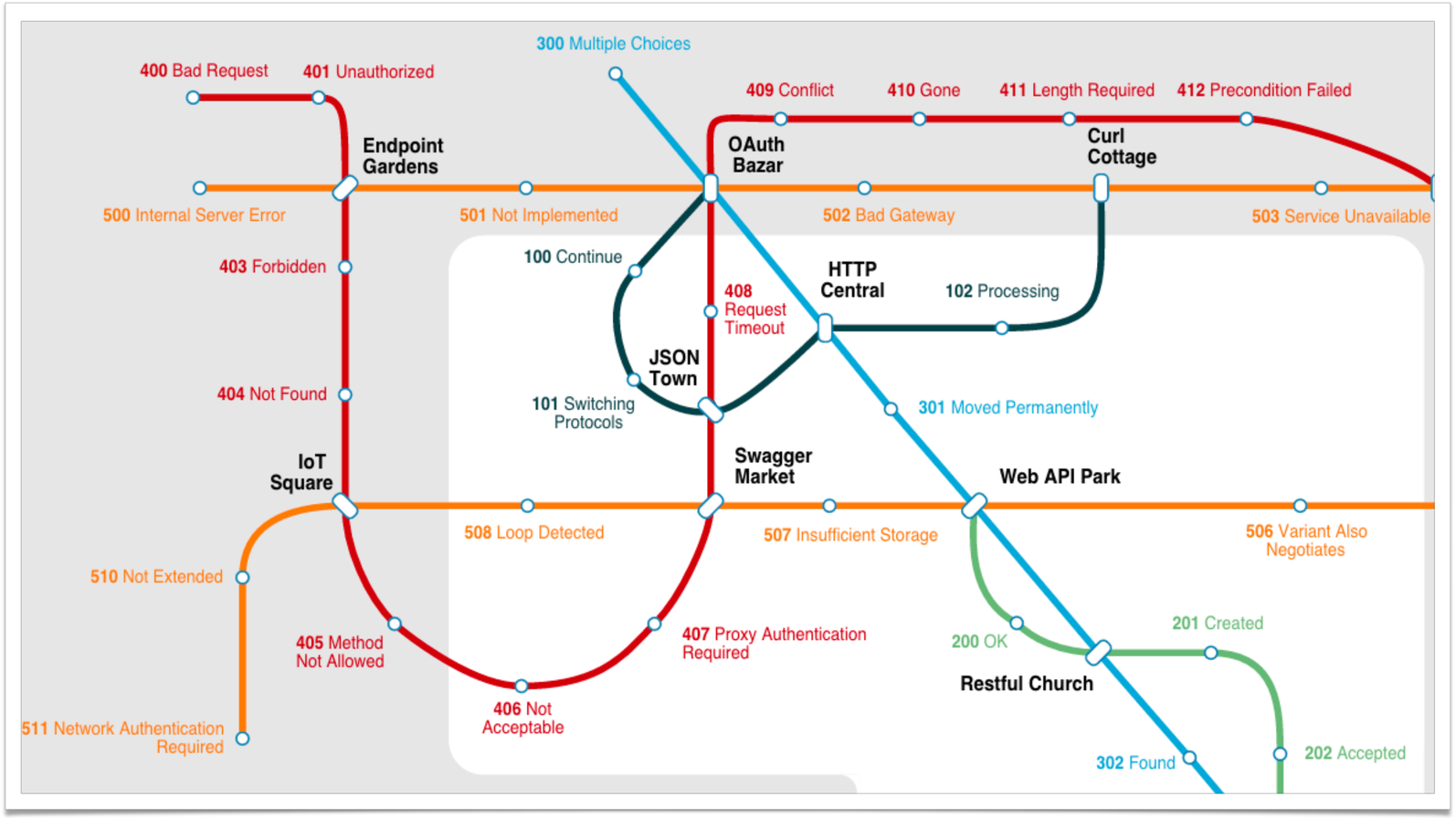
- 500 Internal Server Error
  - A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.
- 501 Not Implemented
  - The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

Can you rely on HTTP Return Codes?

- Short Answer: No
  - For some of the Domino REST Services it is difficult to return anything else then „200 Ok“
  - Facebook also only returns „200 Ok“ ;-)
- Long Answer: Sometimes a HTTP return code is not sufficient
  - Just returning an „400 Bad Request“ if the user did not fill out a mandatory field give the user no clue which field is missing.
  - Since there is no standard defined, the developer has to find a way of telling why the request is „bad“.
    - And hopefully document it well.

Which HTTP Return Code should I use?

- Recommended blog article „Choosing an HTTP Status Code — Stop Making It Hard“ from Michael Kropat
- Or use the map



Make Every Moment Count

Creating a Document with Domino Data Services

- Use the Document collection resource
  - /{database}/api/data/documents
- POST with JSON-Body
- Returns a „201 Created“ and a Location pointing to the new created document
- Supported Query Parameters
  - form – the form name
  - computewithform – boolean
  - parentid – the id of a parent document in case of a response document
- **Input validation based on formula language!**

Extension Library: Rest Service Control

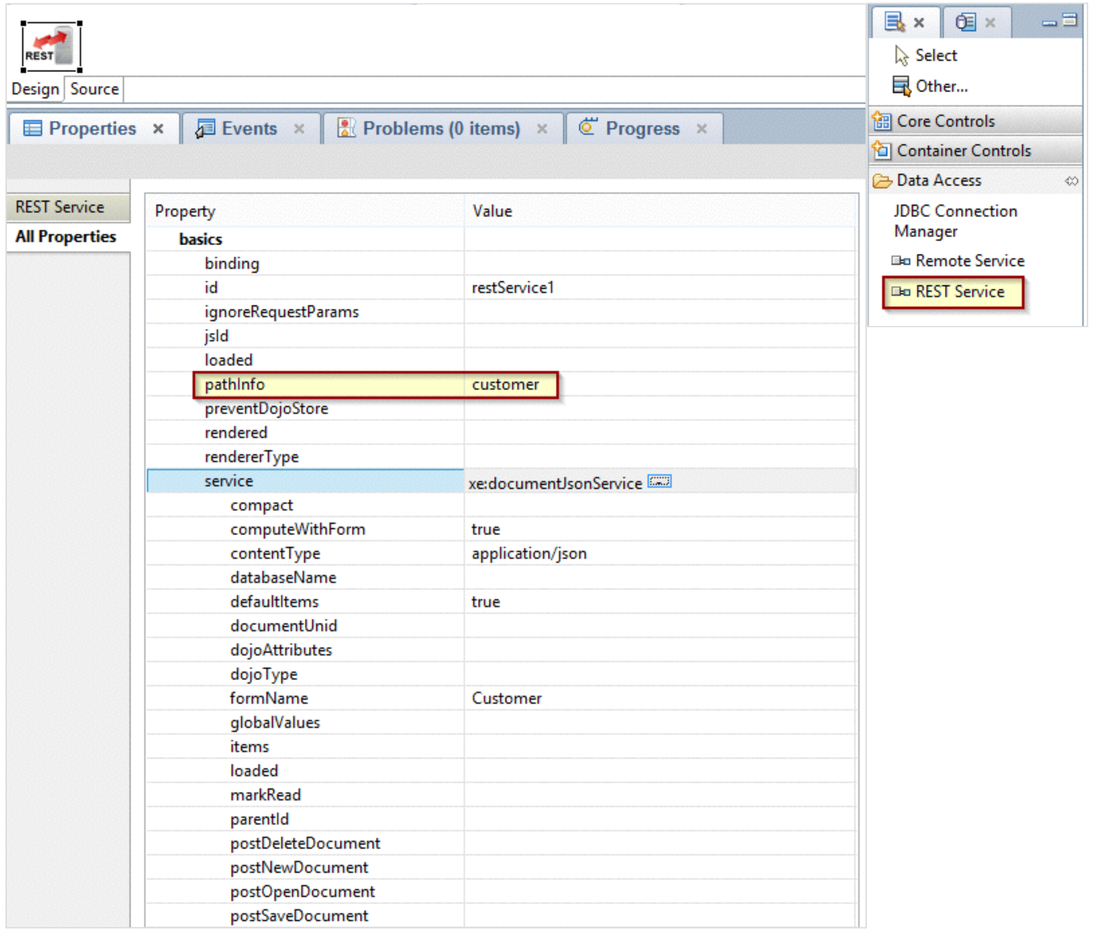
Connect 2016

The Premier Social Business and Digital Experience Conference

#ibmconnect

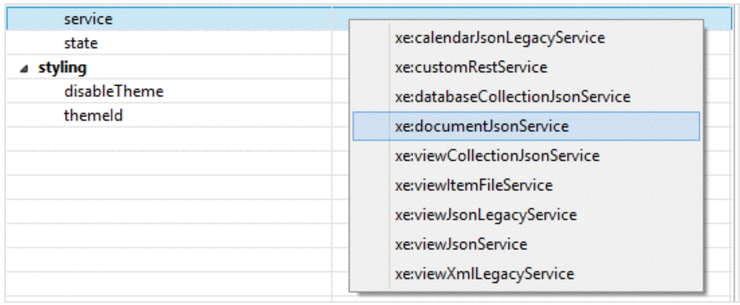
Extension Library - REST Service Control

- The REST Service Control works on a database level
- Domino Data Services (DAS) does not need to be enabled
- Drag the Control on your XPage
- Specify the Service
- The pathInfo property determines the URL
  - {database}/{xpage-name}.xsp/{path}



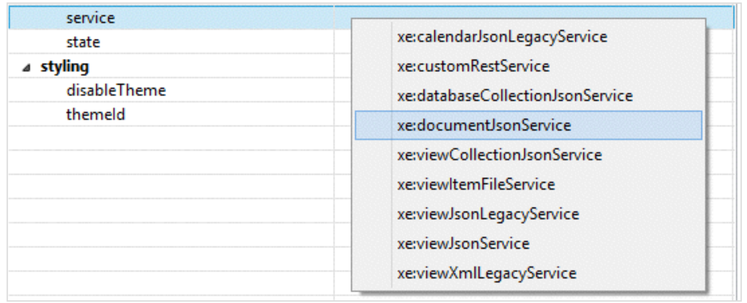
Extension Library - REST Services Control

- Selection of Services
  - calendarJsonLegacyService
  - customRestService
  - databaseCollectionJsonService
  - documentJsonService
  - viewItemFileService
  - viewJsonLegacyService
  - viewJsonService
  - viewXmlLegacyService



Creating a Document with the REST Services Control

- Use Service xe:documentJsonService
  - /{database}/{xpage-name}.xsp/{path}
- POST with JSON-Body
- Returns a „201 Created“ and a Location pointing to the new created document
- Supported Query Parameters
  - form – the form name
  - computewithform – boolean
  - parentid – the id of a parent document in case of a response document
- **Input validation based on formula language!**



Make Every **Moment** Count

Input validation - xe:documentJsonService

- Property computeWithForm=„true“ checks Input validation formula
- Returns „400 Bad Request“ in case
- The messages from the input validation are buried in a Java Stack Trace :-(
  - Hard to extract
- If event queryNewDocument returns „false“, also „400 Bad request“ is returned

Buried failed input validation message


```
{
    "code": 400,
    "text": "Bad Request",
    "message": "Error creating document.",
    "type": "text",
    "data": "com.ibm.domino.services.ServiceException: Error creating document.
        at
com.ibm.domino.services.rest.das.document.RestDocumentJsonService.createDocument(RestDocume
ntJsonService.java:495)
        at
com.ibm.domino.services.rest.das.document.RestDocumentJsonService.renderServiceJSONUpdate(R
estDocumentJsonService.java:413)
        at
com.ibm.domino.services.rest.das.document.RestDocumentJsonService.renderService(RestDocumen
tJsonService.java:166)
        at
com.ibm.designer.runtime.domino.adapter.ComponentModule$AdapterInvoker.invokeServlet(Compon
entModule.java:853)
        at
com.ibm.designer.runtime.domino.adapter.ComponentModule$ServletInvoker.doService(ComponentM
odule.java:796)
        at
com.ibm.designer.runtime.domino.adapter.ComponentModule.doService(ComponentModule.java:565)
        at
com.ibm.domino.xsp.module.nsf.NSFComponentModule.doService(NSFComponentModule.java:1319)
        at com.ibm.domino.xsp.module.nsf.NSFService.doServiceInternal(NSFService.java:662)
        at com.ibm.domino.xsp.module.nsf.NSFService.doService(NSFService.java:482)
        at
com.ibm.designer.runtime.domino.adapter.LCDEnvironment.doService(LCDEnvironment.java:357)
        at
com.ibm.designer.runtime.domino.adapter.LCDEnvironment.service(LCDEnvironment.java:313)
        at
com.ibm.domino.xsp.bridge.http.engine.XspCmdManager.service(XspCmdManager.java:272)
Caused by: NotesException: Validation failed.
Please provide the customer name.
Please provide the customer id.
Remember to fill out your evaluation.
        at lotus.domino.local.Document.computeWithForm(Unknown Source)
        at
com.ibm.domino.services.rest.das.document.RestDocumentJsonService.createDocument(RestDocume
ntJsonService.java:479)
        ... 24 more
"
}
```

Thoughts on Data Driven REST Services

- Domino Access Services(DAS) and most of the services from the REST Service Control give direct access to the data (documents and views).
- Exposes the internal structure of your application
- Limits the possibilities for optimization
- Your REST service acts as a public API
- Think about the services you want to offer

Extension Library - xe:customRestService

- Full control over the REST service
- Direct implementation of methods
  - doDelete
  - doGet
  - doPost
  - doPut
- Or Custom Service Bean

Extension Library - xe:customRestService

- Methods need to return the response

```
<xe:restService
     id="restService1"
     pathInfo="project">
     <xe:this.service>
       <xe:customRestService
         contentType="application/json"
         requestVar="postData"
         requestContentType="application/json">
         <xe:this.doGet><![CDATA[#{javascript:var returnObject = {};
…
return toJson(returnObject);
}]]></xe:this.doGet>
         <xe:this.doDelete><![CDATA[#{javascript:return toJson({status:"error", message:"The
methode 'DELETE' is not implemented"})}]]></xe:this.doDelete>
         <xe:this.doPost><![CDATA[#{javascript:var project = postData;
…

return toJson(returnObject);}]]></xe:this.doPost>
       </xe:customRestService>
     </xe:this.service>
   </xe:restService>
```

Extension Library - xe:customRestService

- Sample code for doGet

```
var returnObject = {};
var projectId = @Right(facesContext.getExternalContext().getRequestPathInfo(),"/project/");
if (projectId.length == 0){
   returnObject.status = "fail";
   returnObject.data = {projectId: "The projectId could not be empty."};
   return (returnObject);
}

var lookupView:NotesView = database.getView("ProjectsByProjectId");
var projectDoc:NotesDocument = lookupView.getDocumentByKey(projectId, true);

returnObject.status = "success";
if (projectDoc==null){
   returnObject.data = null;
   returnObject.message = "Could not find a project with the projectid '" + projectId + "'";
} else {
   returnObject.data = {};
   returnObject.data.projectId = projectDoc.getItemValueString("Projectid");
   returnObject.data.project = projectDoc.getItemValueString("Project");
   returnObject.data.customerId = projectDoc.getItemValueString("CustomerId");
   returnObject.data.customer = projectDoc.getItemValueString("Customer");
}

return toJson(returnObject);
```

Extension Library - Custom Service Bean

- Specify the path and the content type
- Name the bean class

```
<xe:restService
  id="restService2"
  pathInfo="customer"
  ignoreRequestParams="false"
  state="false"
  preventDojoStore="true">
  <xe:this.service>
    <xe:customRestService
      contentType="application/json"
      serviceBean="de.assono.connect2016.CustomerRESTServiceBean">

    </xe:customRestService>
  </xe:this.service>
</xe:restService>
```

Extension Library - Custom Service Bean

- The class needs to extend CustomServiceBean

```
public class CustomerRESTServiceBean extends CustomServiceBean {
   …
   @Override
   public void renderService(CustomService service, RestServiceEngine engine)
         throws ServiceException {

      try {
         HttpServletRequest request = engine.getHttpRequest();
         HttpServletResponse response = engine.getHttpResponse();
         response.setHeader("Content-Type",
               "application/json; charset=UTF-8");

         String method = request.getMethod();
         if (method.equals("GET")) {
            this.doGet(request, response);
         } else if (method.equals("POST")) {
            this.doPost(request, response);
         } else if (method.equals("PUT")) {
            this.doPut(request, response);
         } else if (method.equals("DELETE")) {
            this.doDelete(request, response);
         }

      } catch (Exception e) {
         throw new RuntimeException(e);
      }

   }
```

Custom Database Servlet

Connect 2016

The Premier Social Business and Digital Experience Conference

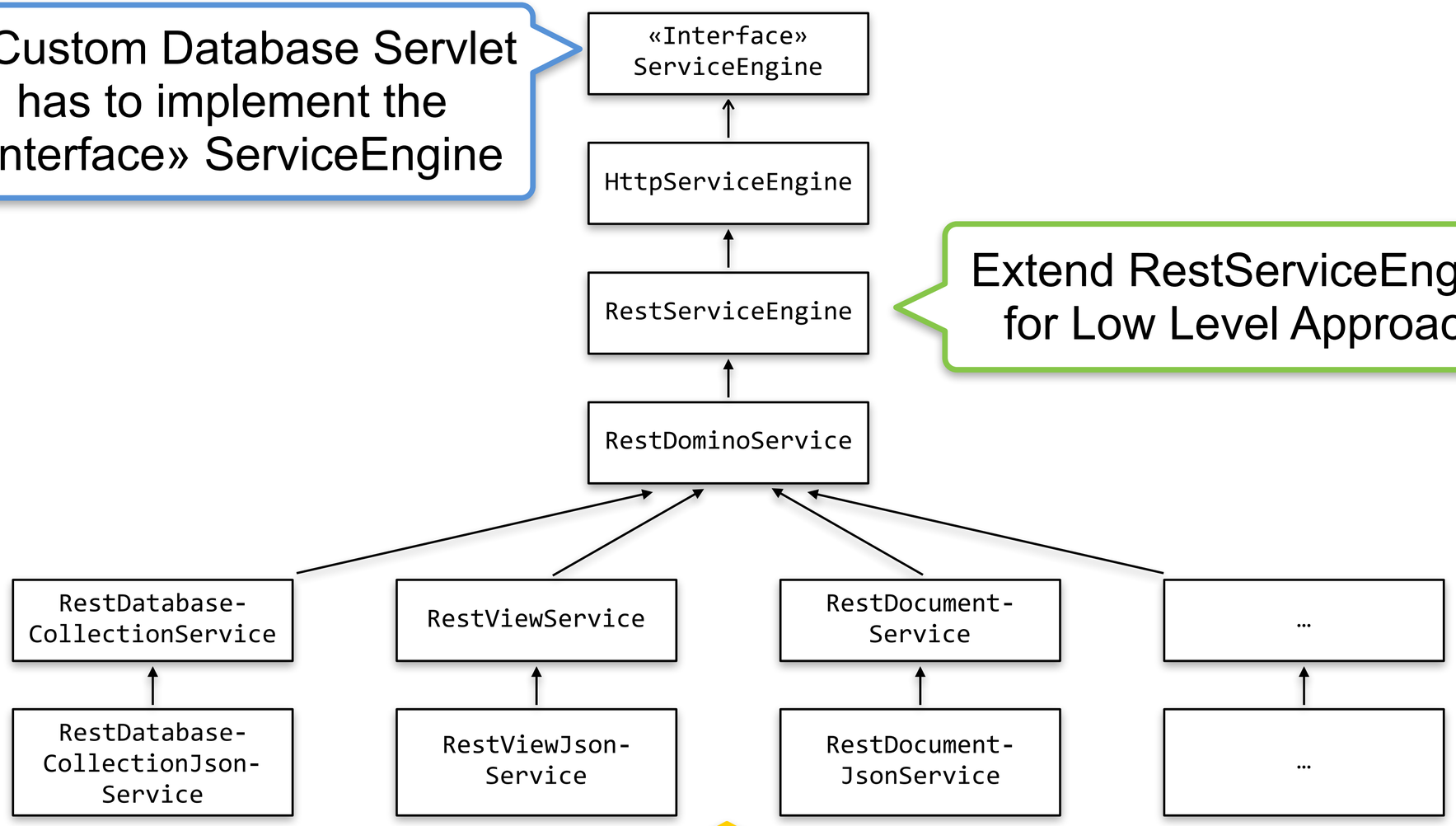
#ibmconnect

Custom Database Servlet

- The Custom Database Servlet operates independent of a XPage
- Low level approach:
  Utilizes passed HttpServletRequest and HttpServletResponse
  - Get the called HTTP method from the request
  - Build the content and pass it to the HttpServletResponse
- High level approach:
  Extend some of the existing REST Domino Services

Custom Database Servlet

A Custom Database Servlet has to implement the «Interface» ServiceEngine



Extend RestServiceEngine for Low Level Approach

Extend any of this classes for High Level Approach

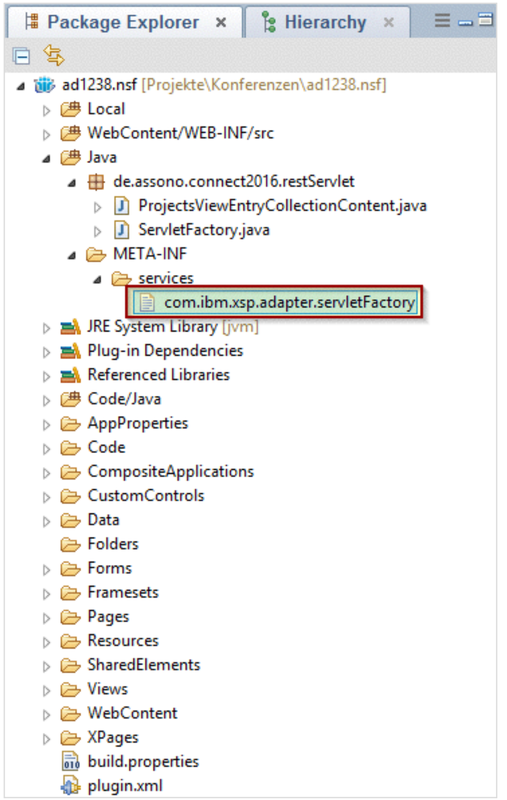
Make Every **Moment** Count

Custom Database Servlet

- Listens to any request to the URL
  - /{database}/xsp/services/{path}
- The Servlet has to implement the Java «Interface» ServiceEngine
  - com.ibm.domino.services.ServiceEngine
- Needs a ServletFactory and a ServiceFactory to be created
- The ServiceFactory creates the Servlet
- The ServletFactory creates the ServiceFactory

The Servlet Factory

- The Servlet Factory is registered via a text file
  - The name has to be com.ibm.xsp.adapter.servletFactory
  - Inside is the qualified Java class name of the Servlet Factory
  - Must resist in a folder META-INF/services/
  - The folder must be visible to the Java Build Path
- Extends the DefaultServletFactory
  - com.ibm.xsp.extlib.services.servlet.DefaultServletFactory



Servlet Factory Sample

```
public class ServletFactory extends DefaultServletFactory {

    public ServletFactory() {
        super("services", "Extension Library Services Servlet", createFactory());
    }


    private static ServiceFactory createFactory() {
        DefaultServiceFactory factory = new DefaultServiceFactory();

        factory.addFactory("projects", new ServiceFactory() {

            public ServiceEngine createEngine(HttpServletRequest httpRequest,
                    HttpServletResponse httpResponse) throws ServletException {

                DefaultViewParameters p = new DefaultViewParameters();
                p.setViewName("ProjectsByProjectId");
                p.setGlobalValues(DefaultViewParameters.GLOBAL_ALL);

                p.setDefaultColumns(true);
                // Set the default parameters
                p.setStart(0);
                p.setCount(9999);

                return new RestViewJsonService(httpRequest, httpResponse, p,
                        new JsonContentFactory() {
                            @Override
                            public JsonViewEntryCollectionContent createViewEntryCollectionContent(
                                    View view, RestViewService service) {
                                return new ProjectsViewEntryCollectionContent(
                                        view, service);
                            }
                        });
            }
        });

        return factory;
    }
}
```

Specifies the path

Custom class for generating the content

Make Every **Moment** Count

Custom Wink Servlet

Connect 2016

The Premier Social Business and Digital Experience Conference

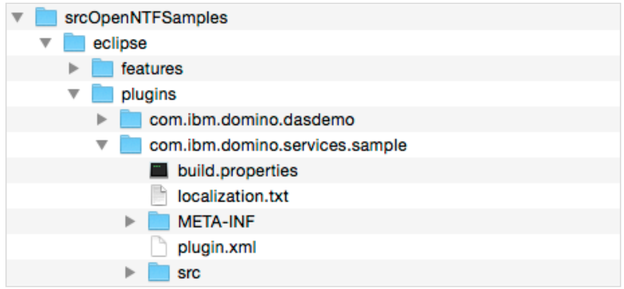
#ibmconnect

Custom Wink Servlet

- Based on Apache Wink
- REST Service independent of a Notes Database
  - Similar to the Domino Core Service
- Installed as a OSGi plugin
- Implements Java API for RESTful Services (JAX-RS)

Custom Wink Servlet

- Sample as part of the OpenNTF Extension Library download
  - srcOpenNTFSamples



- Recommended Articles
  - Paul Withers – From XPages to Web App: Part One – The Application
  - Toby Samples – JAX-RS or THE way to do REST in Domino

Thank you

- Get the latest version of this presentation and the sample database from http://www.assono.de/blog/d6plinks/ibmconnect2016-ad1238

Connect 2016

The Premier Social Business and Digital Experience Conference

#ibmconnect

Acknowledgements and Disclaimers

Availability. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Acknowledgements and Disclaimers cont.