NOTES & DOMINO
ENTWICKLERCAMP

assono

REST Services in Domino

EntwicklerCamp
13. April 2016

Innovative Software-Lösungen.

www.assono.de

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

Bernd Hort, assono GmbH - February 1, 2016

Connect2016

The Premier Social Business and Digital Experience Conference

#ibmconnect

NOTES & DOMINO ENTWICKLERCAMP

assono

Bernd Hort

Diplom-Informatiker, Universität Hamburg

seit 1995 entwickle ich Lotus Notes Anwendungen

IBM Certified Application Developer, System Administrator & Instructor

Sprecher auf diversen Konferenzen & Lotusphere 2008/IBM Connect 2014/IBM ConnectED 2015/ IBM Connect 2016

bhort@assono.de
http://www.assono.de/blog
040/73 44 28-315

assono
IT-Consulting & Solutions

NOTES & DOMINO ENTWICKLERCAMP

assono

Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

NOTES & DOMINO ENTWICKLERCAMP

assono

Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

REST in a Nutshell

- Representational state transfer (REST) ist ein Programmierstil für den Zugriff auf Ressourcen basierend auf Web Standards
- RESTful Web Services implementieren den Zugriff und die Veränderung von solchen Ressourcen
- Meistens mittels des HTTP-Protokolls
- Ressourcen können unterschiedliche Repräsentationen haben, z.B. text, xml, json etc.
  - Der Rest Client kann über das HTTP Protokoll (content negotiation) nach einer bestimmten Repräsentation fragen.
- Kein Standard!

WADL

- Das WSDL Equivalent
  Web Application Description Language (WADL)
  ist optional
- Aufruf, Parameter und Rückgabewerte müssen der Dokumentation entnommen werden
- Hoffentlich ist die Dokumentation gut ;-)

NOTES & DOMINO ENTWICKLERCAMP

assono

REST in a Nutshell – Nomen und Verben

- Ressourcen werden über eine URI identifiziert - Nomen
- Aktionen auf den Ressourcen werden durch HTTP Methoden bestimmt - Verben

Ressource	Get	Post	Put	Delete
Collection von Ressourcen http://yourserver/path/sessions	**Listed** alle Ressourcen auf	**Erzeugt** ein neuen Eintrag in der Collection	**Tauscht** die gesamte Collection durch eine andere Collection aus.	**Löscht** die gesamte Collection
Einzelne Ressource http://yourserver/path/sessions/BP206	**Zeigt** die Details einer bestimmten Ressource an	Normalerweise nicht implementiert	**Akualisiert** den adressierte Eintrag in der Collection, oder **erzeugt** ihn, wenn er nicht existiert.	**Löscht** die spezifisierte Ressource

Auch bekannt als create, read, update and delete (CRUD)

NOTES & DOMINO ENTWICKLERCAMP

assono

Agenda

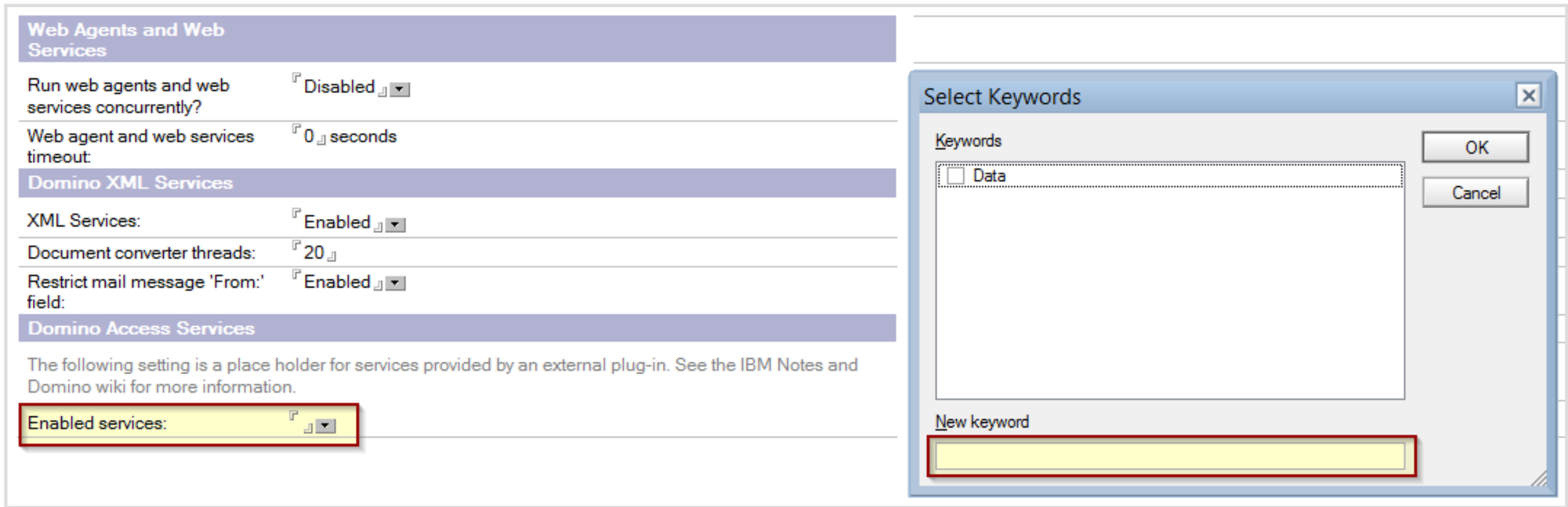
- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

IBM Domino Access Services (DAS)

- IBM Domino Access Services liefert Ressourcen via HTTP
- Domino Core Service
  - Ressourcen, die sich nicht auf andere Services beziehen
    - z.B. Password Statistik Ressource
- Domino Data Service
  - Ressourcen, die Notes Datenbanken oder sich innerhalb von Notes Datenbanken befinden
- Domino Calendar Service
  - Ressourcen, um auf Kalender zuzugreifen
- Domino Mail Service
  - Ressourcen für den Zugriff auf Mail Dateien auf einem Domino server
  - Teil der Extension Library 9.0.1

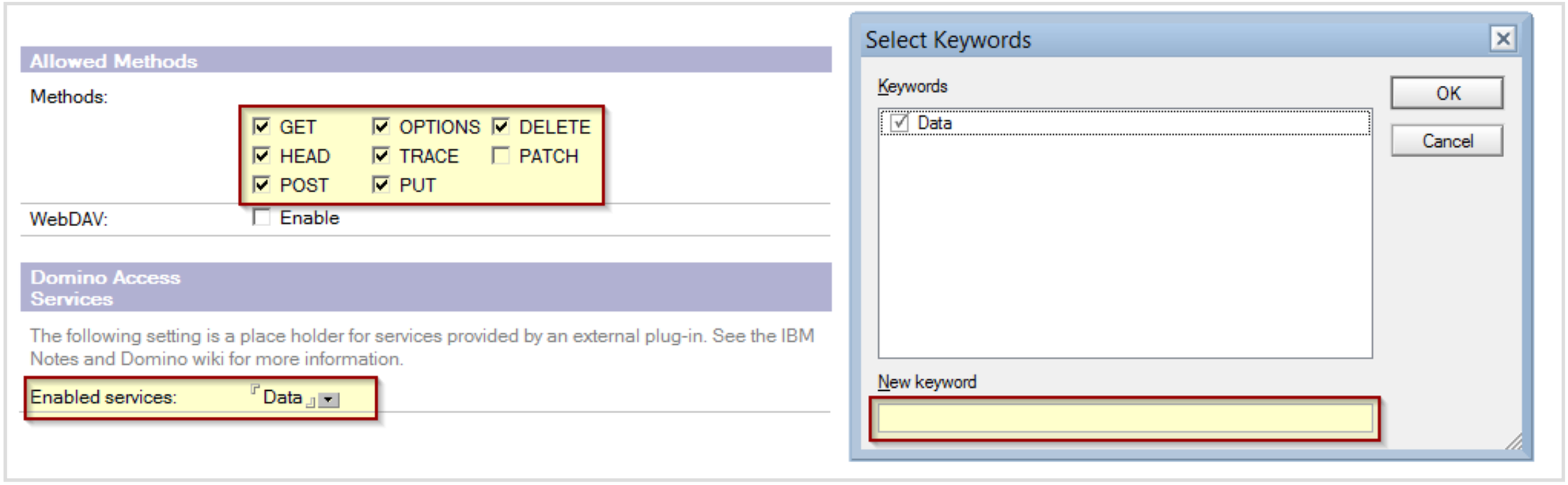
Einrichtung Domino Access Services

- Entweder im Server Dokument
  - Internet Protocols… \ Domino Web Engine



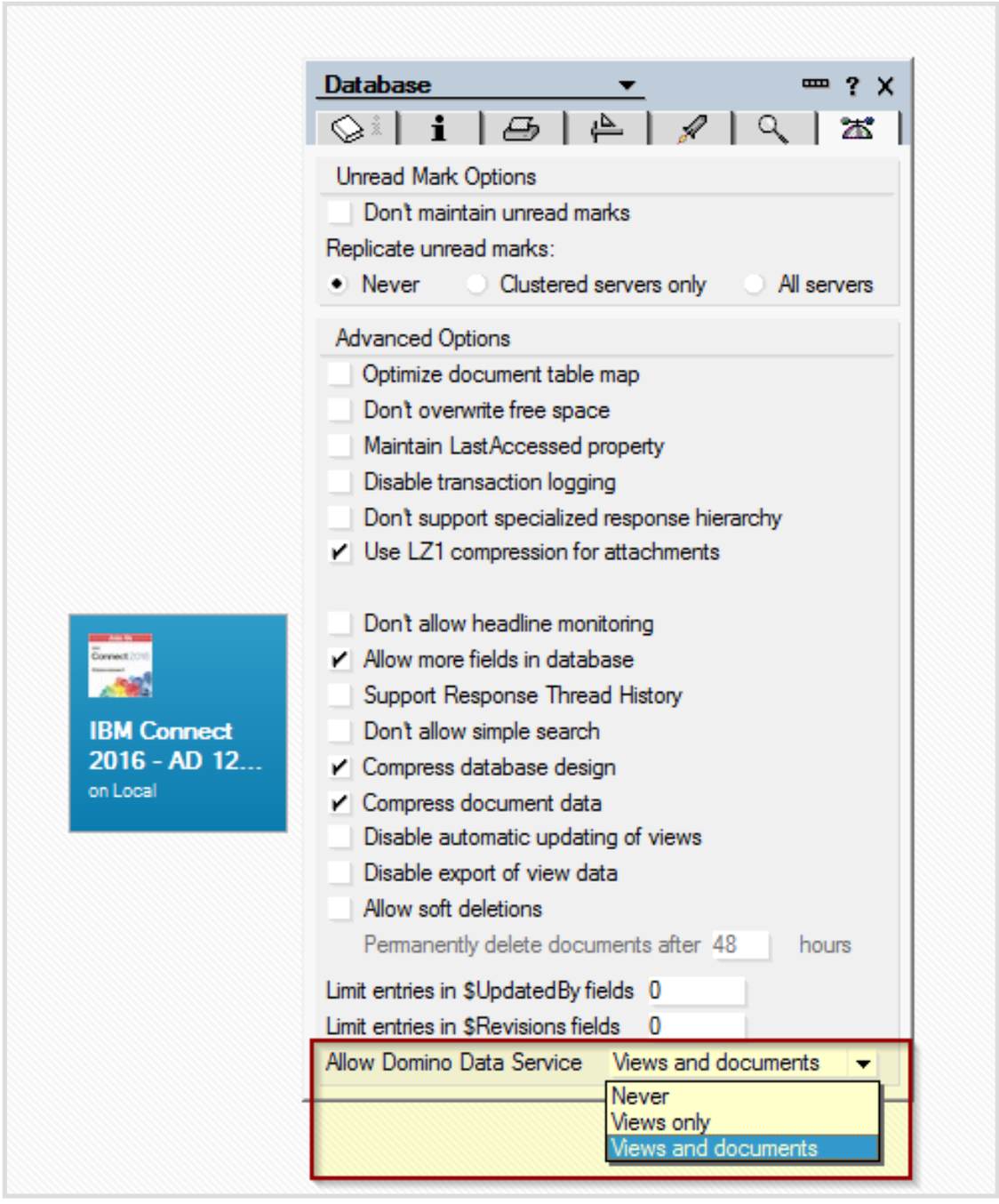
Einrichtung Domino Access Services

- Oder im Internet Site Document
  - Configuration
- Über „New Keyword“ zusätzliche Services aktivieren



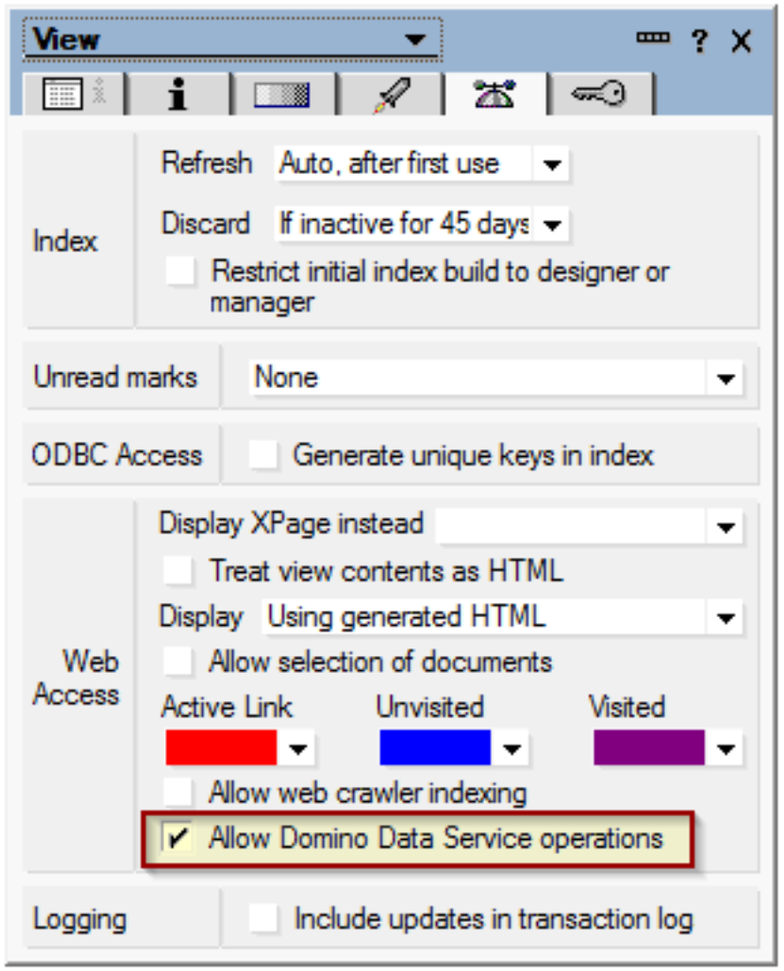
Einrichtung Domino Access Services

- Auf Datenbank-Ebene
  - Allow Domino Data Service
- Auswahl
  - Never
  - Views only
  - Views and Documents
- Vorgabe ist „Never“



Einrichtung Domino Access Services

- Auf Ansichtsebene
  - Allow Domino Data Service operations
- Vorgabe ist nicht aktiviert



Zugriff auf Domino Data Services

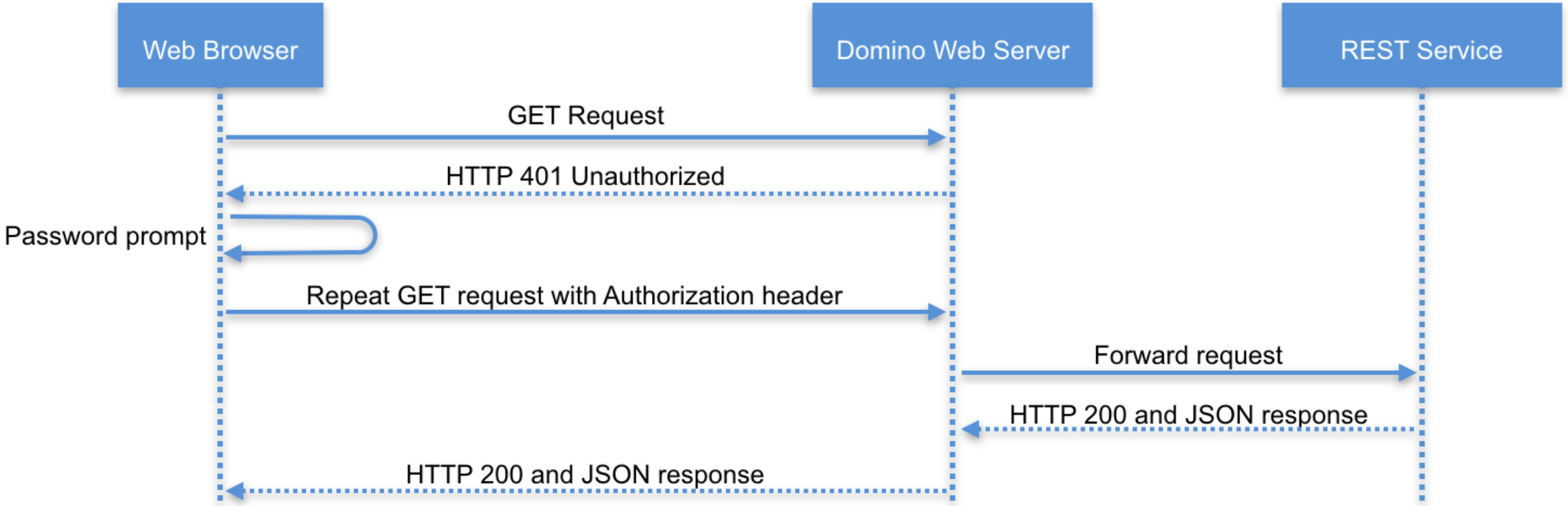
- Die URL spezifiziert die Datenbank und die Art der Ressource
- Dokument Ressource
  - /{database}/api/data/documents/unid/{docunid}
- Dokument Collection Ressource
  - /{database}/api/data/documents
- Ansicht / Ordner Collection Ressource
  - /{database}/api/data/collections
- Ansicht / Ordner Eintrag Ressource
  - /{database}/api/data/collections/name/{name}

Authentifizierung Domino REST Service Requests

- REST Services verwendet HTTP Security Features für die Authentifizierung
- Für Domino unterstützt werden
  - Basic Authentication
  - Session Based Authentication
  - SPNEGO (Single Sign-On für Windows)
  - SAML (Verfügbar seit Domino 9.0)
  - Third-Party Single Sign-On (SSO) Erweiterungen (z.B. CA SiteMinder)
- Guter Artikel im IBM Notes and Domino Application Development wiki
  - Authenticating Domino REST Service Requests

Basic Authentication

- Username und codiertes Passwort werden im Header gesendet
  - Immer HTTPS mit Basic Authentication verwenden
- Authentifizierung Sequenz

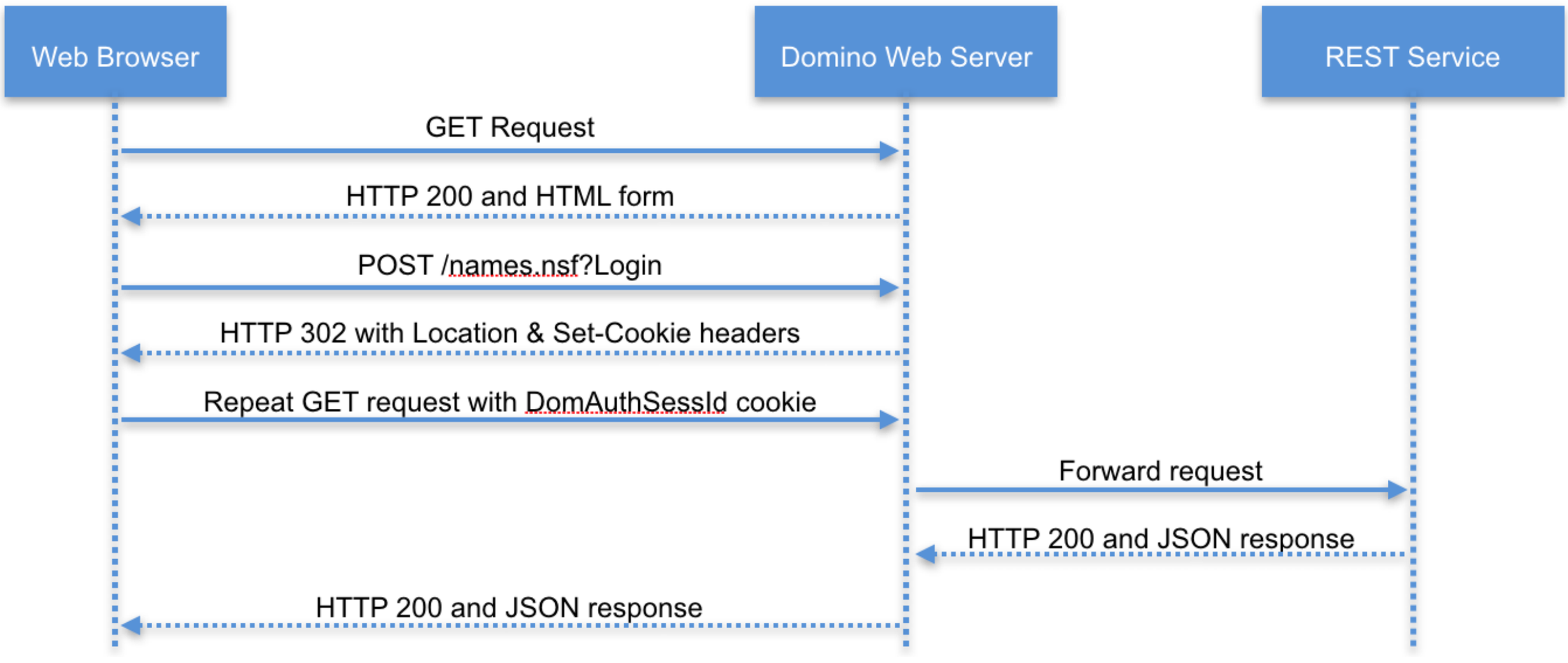


NOTES & DOMINO ENTWICKLERCAMP

assono

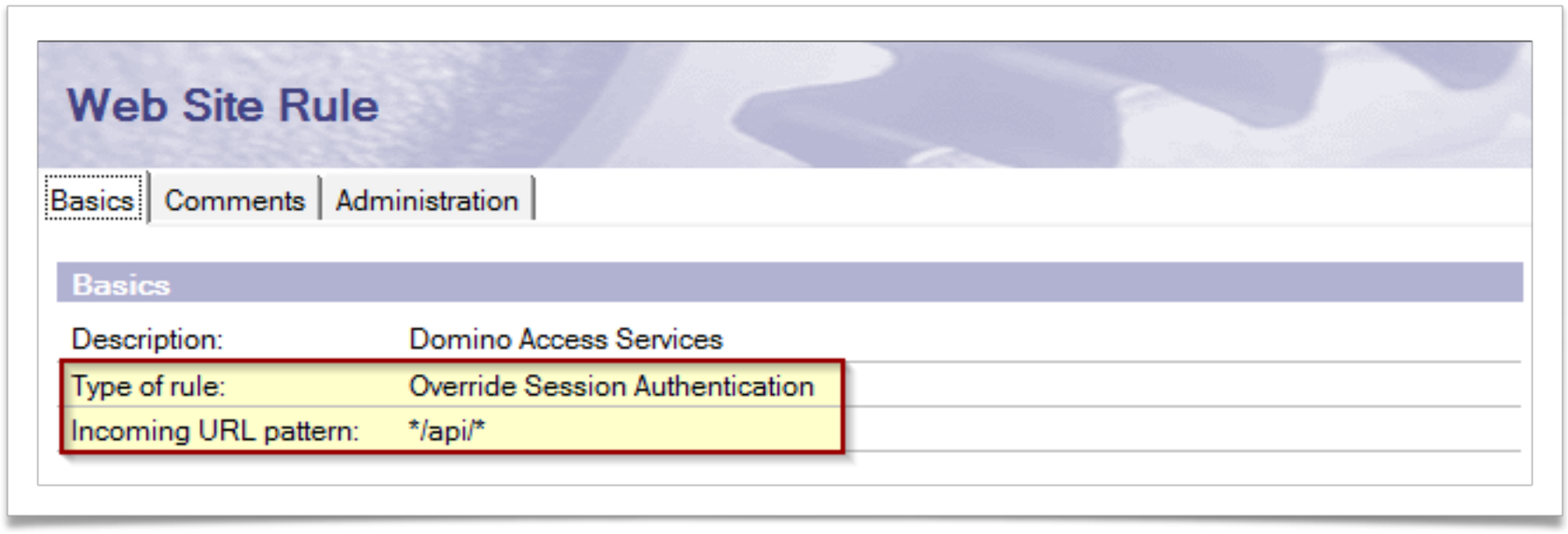
Session Based Authentication

- Session Cookie wird mit jedem Request gesendet
  - Immer HTTPS mit Session based Authentication verwenden
- Authentifizierung Sequenz



Probleme mit Session Based Authentication

- Ein Service Consumer muss den HTTP Return Code und den Content Typ bzw. den Content überprüfen, um die fehlende Authentifizierung feststellen
- Für Nicht-Web-Browser Service Consumer bedeutet die Verwaltung des DomAuthSessId Cookie zusätzliche Programmierung
- Lösung: Web Site Rule – Override Session Authentication
- Erzwingt für alle Requests eines definierten URL Musters die Basic Authentication
- URL pattern: */api/*



HTTP Return Codes

- Liste von 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 für erfolgreiche HTTP Requests. Die konkrete Response hängt von der verwendeten Methode ab. Bei einem GET Request wird die angeforderte Ressource zurückgeliefert. Bei einem POST Request wird entweder die erzeugte Ressource oder eine Beschreibung des Ergebnisses der Aktion zurückgeliefert.
- 201 Created
  - Der Request wurde verarbeitet und eine neue Ressource wurde angelegt.
- 204 No Content
  - Der Server hat die Anfrage erfolgreich verarbeitet, aber es gibt keinen Inhalt der zurückgeliefert werden könnte.

HTTP Return Codes – 3xx Redirection Messages

- 301 Moved Permanently
  - Diese und alle zukünftigen Requests sollten auf die neue URI geändert werden.
- 302 Found
  - Dieser Response Code bedeutet das URI der angeforderten Ressource temporär geändert wurde. Erneute Anpassungen an der URI können in Zukunft möglich sein. Aus diesem Grund sollten Clients bei zukünftigen Anfragen die ursprüngliche URI verwenden.

HTTP Return Codes – 4xx Client Error Responses

- 400 Bad Request
  - Der Server kann oder will die Anfrage nicht bearbeiten aufgrund etwas, was als Client Fehler eingeschätzt wird. (z.B. ungültige Request Syntax, fehlende Informationen, etc).
  - Dieser Return Code sollte verwendet werden, wenn kein spezifischer Code passt.
- 401 Unauthorized
  - Vergleichbar mit 403 Forbidden
  - Wird insbesondere verwendet, wenn Authentifizierung notwendig ist, der Authentifizierungsversuch gescheitert ist oder bisher noch keine Credentials übergeben wurden.
- 403 Forbidden
  - Die Anfrage ist an und für sich gültig, aber Server verweigert die Ressource. Authentifizierung ist sinnlos.

HTTP Return Codes – 4xx Client Error Responses

- 404 Not Found
  - Die angefragte Ressource konnte nicht gefunden werden. Sie könnte aber in der Zukunft wieder verfügbar sein.
- 409 Conflict
  - Die Anfrage konnte aufgrund eines Konflikts nicht verarbeitet werden. Dieses könnte zum Beispiel beim parallelen Updates der Ressource auftreten.

HTTP Return Codes – 5xx Server Error Responses

- 500 Internal Server Error
  - Diese generische Fehlermeldung wird zurückgeliefert wenn eine unerwartete Fehlersituation aufgetreten ist und keine spezifischer Meldung möglich ist.
- 501 Not Implemented
  - Die im Request verwendete Methode wird vom Server nicht unterstützt und wird dem entsprechend nicht verarbeitet.
  - Die einzigen Methoden die immer unterstützt werden müssen sind GET und HEAD. Dem entsprechend kann dieser Fehlercode bei solchen Anfragen nicht erscheinen.

Sind HTTP Return Codes verlässlich?

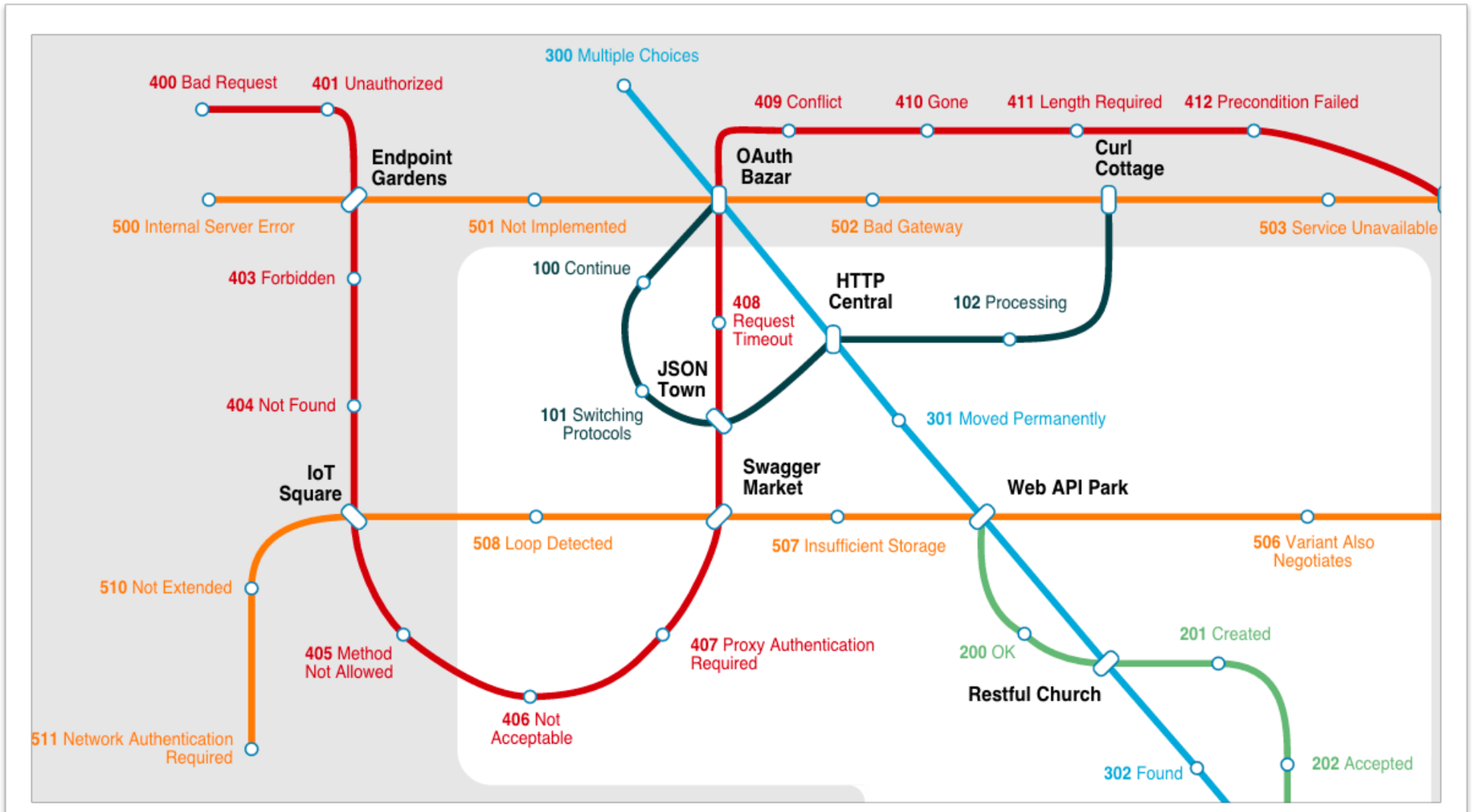
- Kurze Antwort: Nein!
  - Für einige Domino REST Services ist es schwierig, überhaupt etwas anderes zurück zu liefern als „200 Ok“.
  - Facebook beantwortet auch jede Anfrage mit „200 Ok“ ;-)
- Lange Antwort: ein HTTP Return Code ist nicht immer ausreichend
  - Einfach nur „400 Bad Request“ bei einem fehlenden Pflichtfeld hilft dem Anwender nicht dabei herauszufinden, welches Feld fehlt.
  - Da es keinen Standard bei REST gibt, muss der Entwickler einen Weg finden, mitzuteilen warum die Anfrage „bad“ ist.
    - Und es dann auch gut zu dokumentieren.

NOTES & DOMINO ENTWICKLERCAMP

assono

Welchen Return Code sollte ich verwenden?

- Empfohlener Artikel
  „Choosing an HTTP Status Code — Stop Making It Hard“
  von Michael Kropat
- Oder diese Karte verwenden ;-)



Ein Dokument mit Domino Data Services erzeugen

- Verwenden der Dokument Collection Ressource
  - /{database}/api/data/documents
- POST mit JSON-Body
- Rückgabewert „201 Created“ und im Header „Location“ mit der URI des neu angelegten Dokumentes
- Unterstützte Query Parameters
  - form – der Maskenname
  - computewithform – boolean
  - parentid – die UNID des Elterndokumentes bei Antwortdokumenten
- Eingabevalidierung basierend auf der Formelsprache!

NOTES & DOMINO ENTWICKLERCAMP

assono

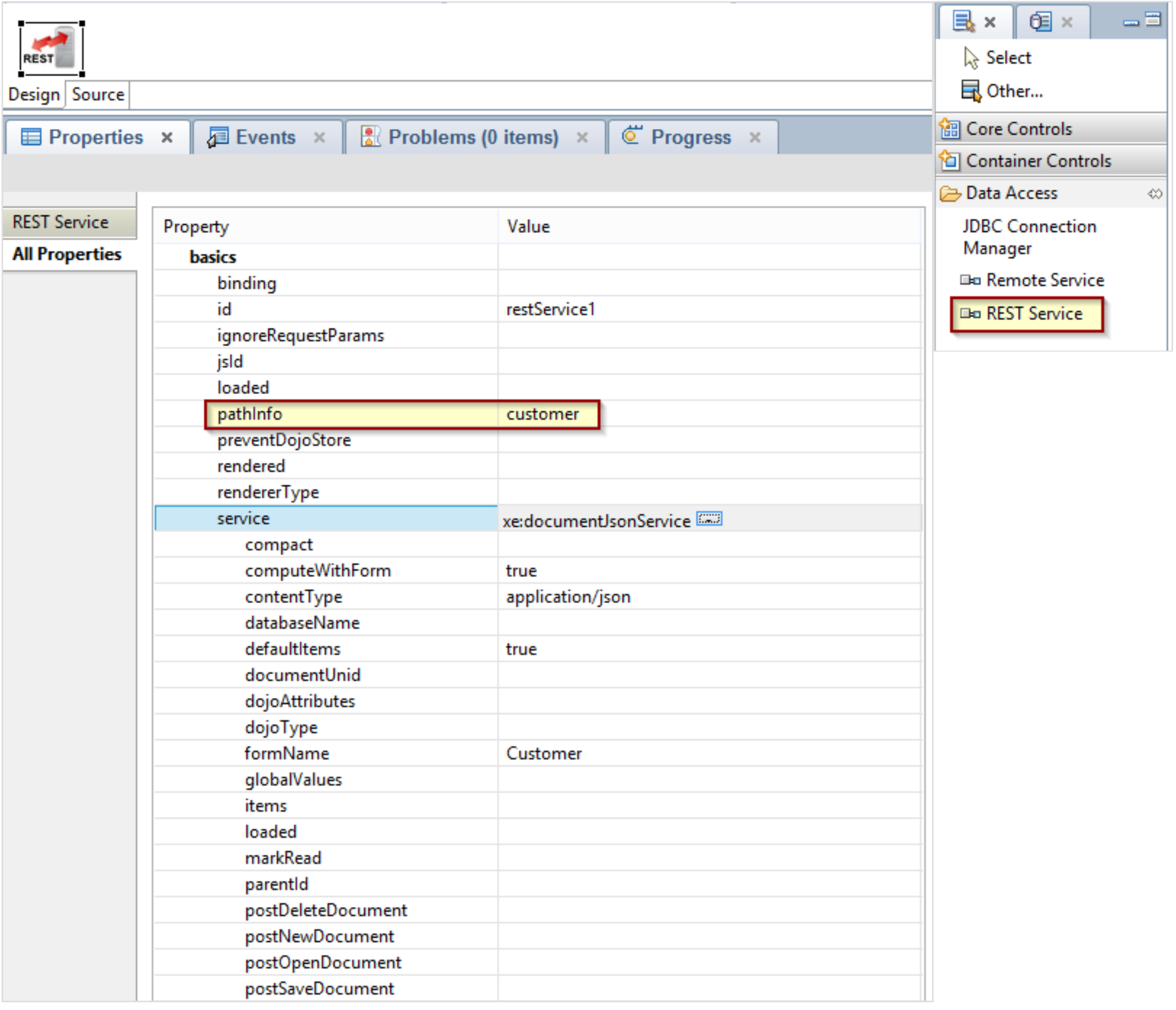
Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

Extension Library: REST Service Control

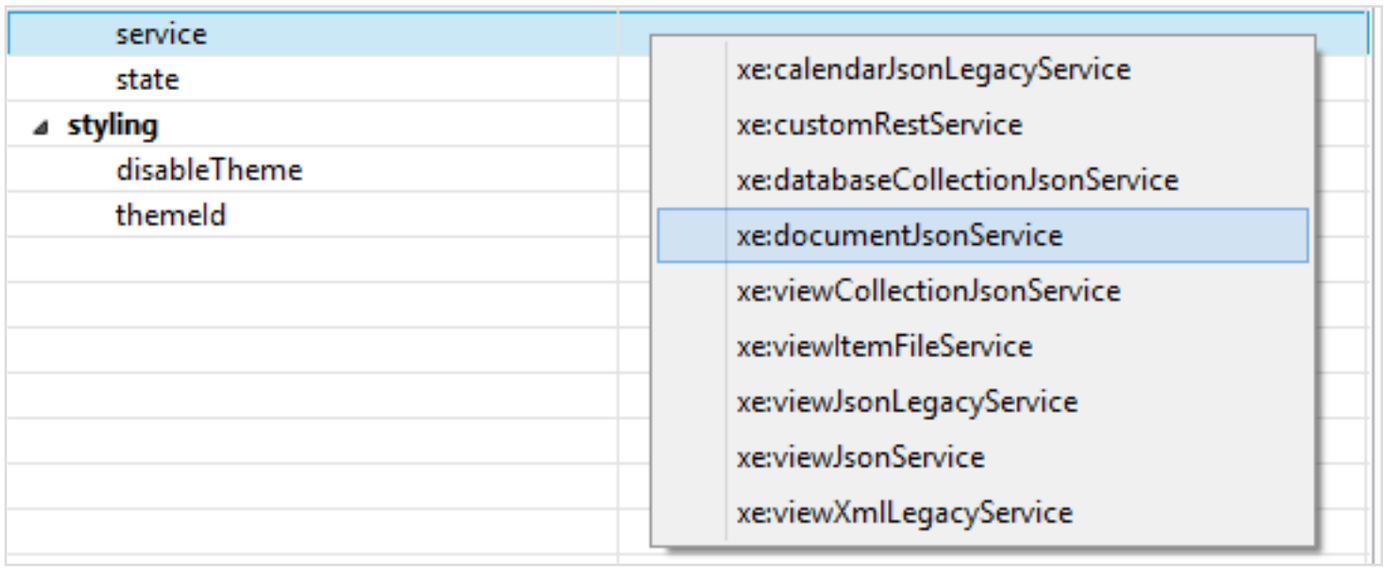
- Das REST Service Control arbeitet auf Datenbankebene
- Domino Data Services (DAS) muss nicht aktiviert sein
- Das Control per Drag & Drop auf eine XPage ziehen
- Den Service auswählen
- Die Eigenschaft pathInfo die URL
  - {database}/{xpage-name}.xsp/{path}

NOTES & DOMINO ENTWICKLERCAMP



Extension Library: REST Service Control

- Auswahl der Services
- calendarJsonLegacyService
- customRestService
- databaseCollectionJsonService
- documentJsonService
- viewItemFileService
- viewJsonLegacyService
- viewJsonService
- viewXmlLegacyService



Ein Dokument mit dem REST Service Control erzeugen

- Verwenden des Service xe:documentJsonService
  - /{database}/{xpage-name}.xsp/{path}
- POST mit JSON-Body
- Rückgabewert „201 Created“ und im Header „Location“ mit der URI des neu angelegten Dokumentes
- Unterstützte Query Parameters
  - form – der Maskenname
  - computewithform – boolean
  - parentid – die UNID des Elterndokumentes bei Antwortdokumenten
- Eingabevalidierung basierend auf der Formelsprache!

Eingabevalidierung - xe:documentJsonService

- Eigenschaft computeWithForm=„true“ überprüft die Eingabevalidierungsformeln
- Rückgabewert „400 Bad Request“
- Die Nachrichten von der Eingabevalidierung ist im Java Stack Trace versteckt :-(
- Sehr schwer zu parsen
- Wenn im queryNewDocument Event „false“ zurück gegeben wird, wird ebenfalls „400 Bad request“ zurück gegeben wird.

```
{
    "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
"
}
```

Gedanken zu Data Driven REST Services

- Domino Access Services(DAS) und die meisten anderen Services des REST Service Controls geben direkten Zugriff auf die Daten (Dokumente und Ansichten).
- Die interne Struktur der Anwendung wird sichtbar.
- Ihr REST Service definiert eine öffentliche API
- Werden direkt die Daten sichtbar gemacht, schränkt es die Möglichkeit für eine spätere Optimierung aus.

- Denken Sie darüber nach, welchen Service Sie wirklich anbieten wollen.

Extension Library - xe:customRestService

- Volle Kontrolle über den REST Service
- Direkte Implementierung der Methoden
  - doDelete
  - doGet
  - doPost
  - doPut
- Oder ein Custom Service Bean

Extension Library - xe:customRestService

```
<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 - 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

- Der Pfad, der Content-Type und der Klassenname werden spezifiziert

```
<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>
```

NOTES & DOMINO ENTWICKLERCAMP

assono

Extension Library - Custom Service Bean

- Die Klasse muss vom CustomServiceBean erben

```
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);
        }

    }
```

NOTES & DOMINO ENTWICKLERCAMP

assono

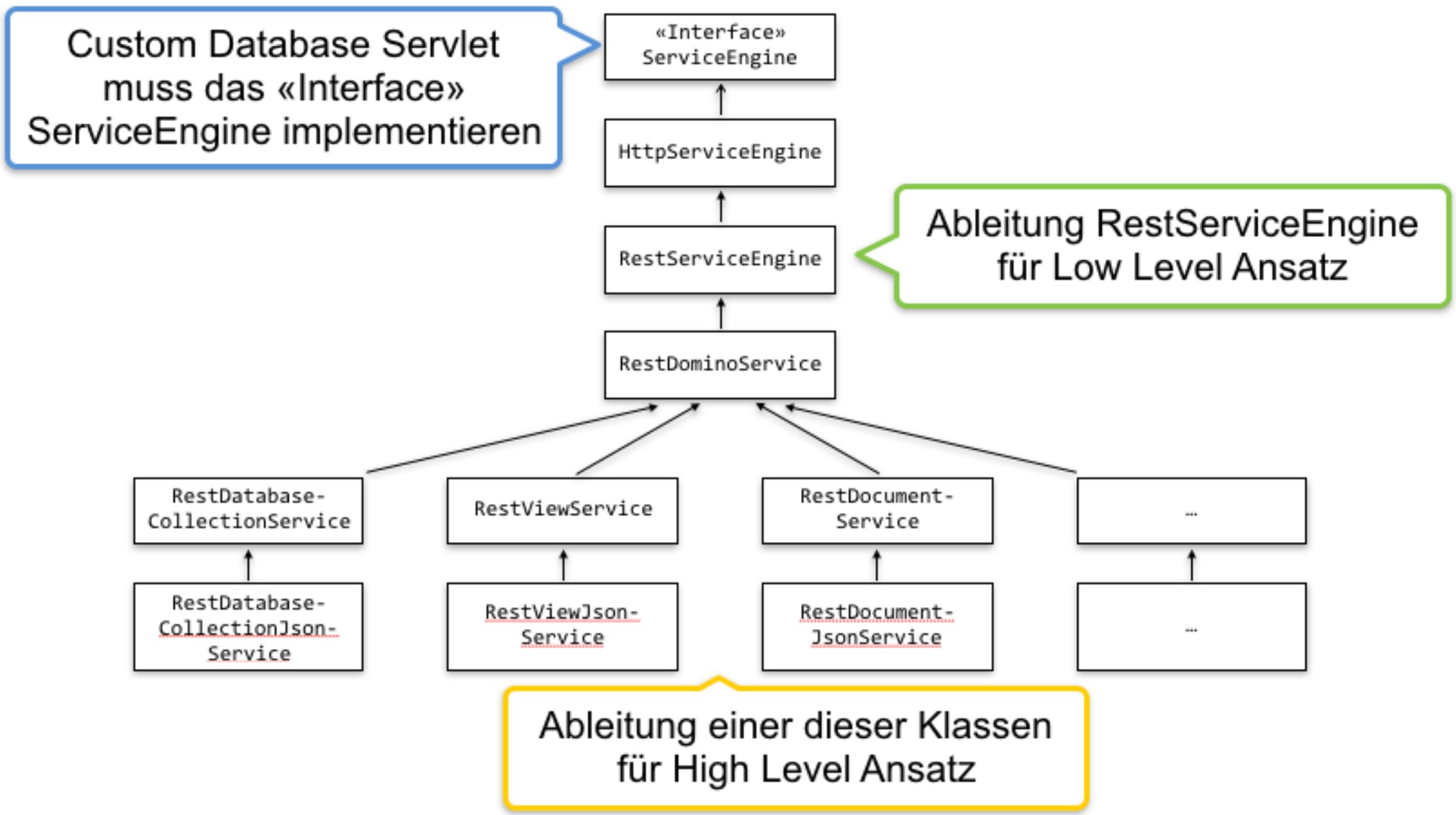
Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

Custom Database Servlet

- Das Custom Database Servlet ist unabhängig von einer XPage
- Low Level Ansatz:
  Verwendung der übergebenen Objekte HttpServletRequest und HttpServletResponse
  - Auswerten der verwendeten HTTP Methode aus dem Request
  - Die Rückantwort erzeugen und an die HttpServletResponse übergeben
- High Level Ansatz:
  Ableitung eines bestehenden REST Domino Services

Custom Database Servlet



Custom Database Servlet

- Überwacht jeden Request der angegebenen URL
  - /{database}/xsp/services/{path}
- Das Servlet muss das Java «Interface» ServiceEngine implementieren
  - com.ibm.domino.services.ServiceEngine
- Benötigt eine ServletFactory und eine ServiceFactory
- Die ServiceFactory erzeugt das Servlet
- Die ServletFactory erzeugt die ServiceFactory

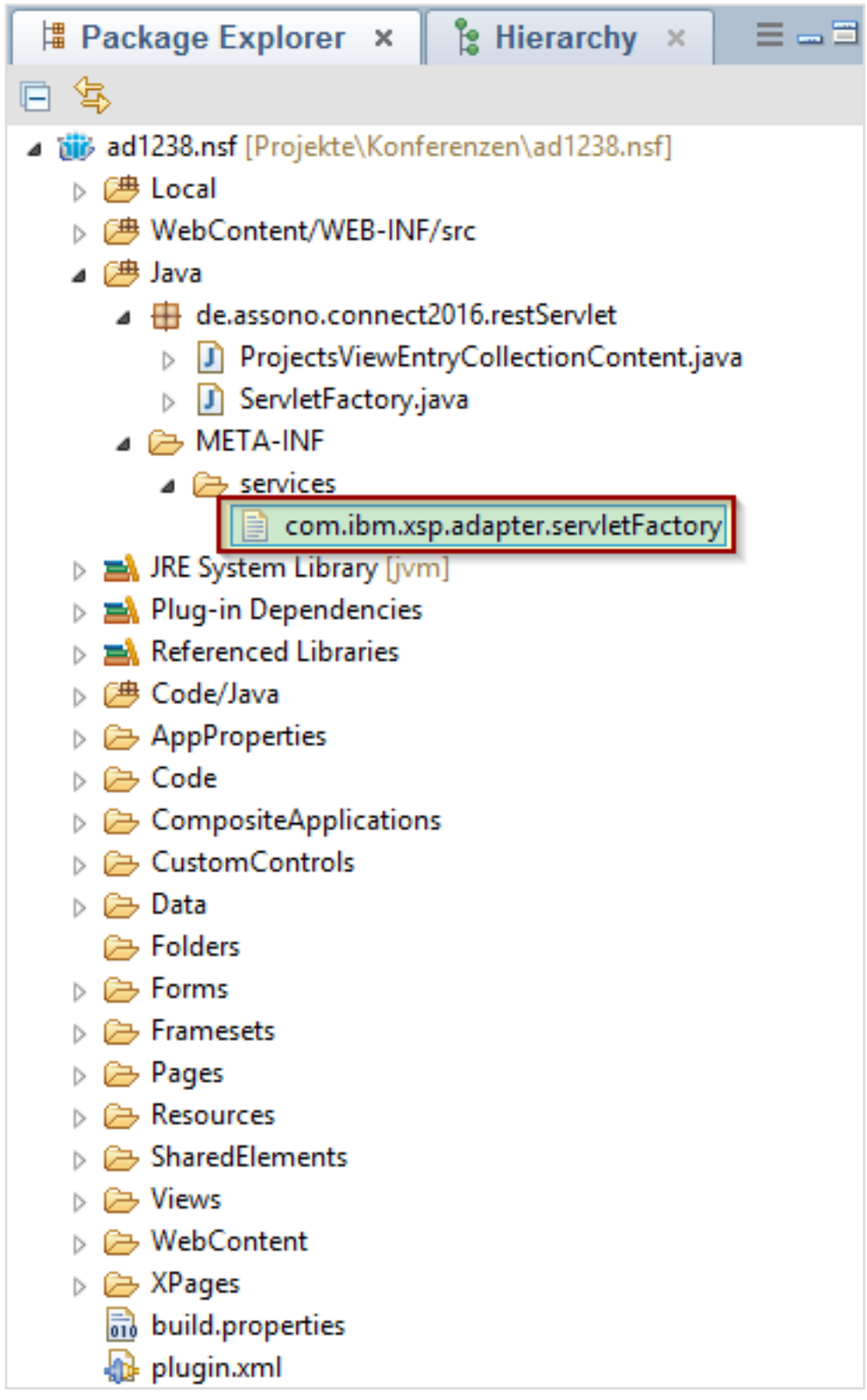
Die Servlet Factory

- Die Servlet Factory wird über eine Textdatei registriert
  - Der Dateiname muss com.ibm.xsp.adapter.servletFactory lauten.
  - In der Textdatei steht der qualifizierte Java Klassenname der Servlet Factory
  - Die Textdatei muss sich im Ordner META-INF/services/ befinden.
  - Der Ordner muss sichtbar für den Java Build Path sein.
- Ableitung der Klasse DefaultServletFactory
  - com.ibm.xsp.extlib.services.servlet.DefaultServletFactory

NOTES & DOMINO ENTWICKLERCAMP

assono

Die Servlet Factory



```
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;
    }
}
```

Spezifiziert den Pfad

Die abgeleitete Klasse für die Generierung der Inhalte

Agenda

- REST in a Nutshell
- IBM Domino Access Services (DAS)
- Extension Library: REST Service Control
- Custom Database Servlet
- Custom Wink Servlet
- Fragen und Antworten

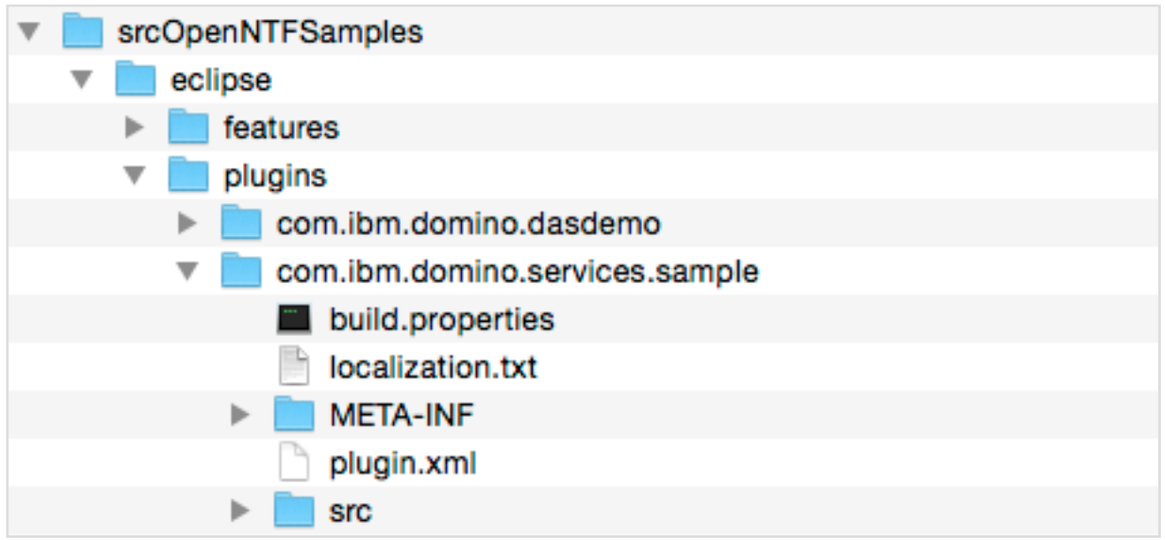
Custom Wink Servlet

- Basiert auf Apache Wink
- REST Service unabhängig von einer Notes Datenbank
- Vergleichbar mit den Domino Core Service
- Installiert als ein OSGi plugin
- Implementiert Java API for RESTful Services (JAX-RS)

Custom Wink Servlet

- Beispiel im OpenNTF Extension Library Download
  - srcOpenNTFSamples

openntf



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

NOTES & DOMINO ENTWICKLERCAMP

assono

Fragen?

jetzt stellen – oder später:

bhort@assono.de

http://www.assono.de/blog

040/73 44 28-315

assono
IT-Consulting & Solutions

Folien & Beispieldatenbank unter
http://www.assono.de/blog/d6plinks/EntwicklerCamp-2016-REST