Lotusphere
Connect2013
Get social. Do business.

BP204 Take a REST and put your data to work with APIs

Craig Schumann | Inner Ring Solutions

Agenda

- Background
- Practical Examples from real life
- Designing guidelines for creating your own APIs
- How to create your own APIs and free your data

House keeping

- IBM Lotus Domino®
- IBM Domino®
- IBM Lotus Notes®
- IBM Notes®
- IBM Lotus Designer®
- IBM Designer®

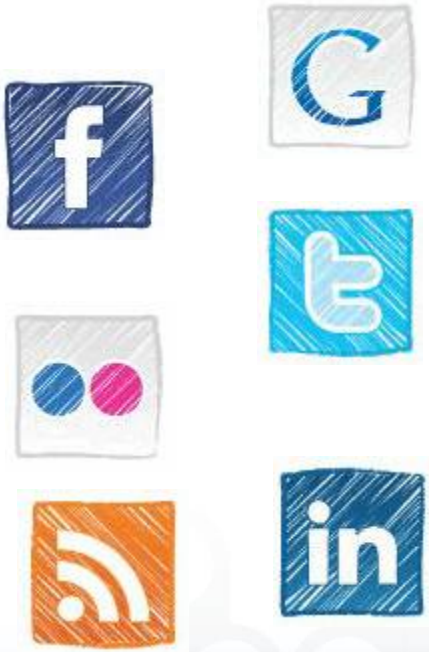
Background

So much data….

- What this talk is about is freeing ***your*** data

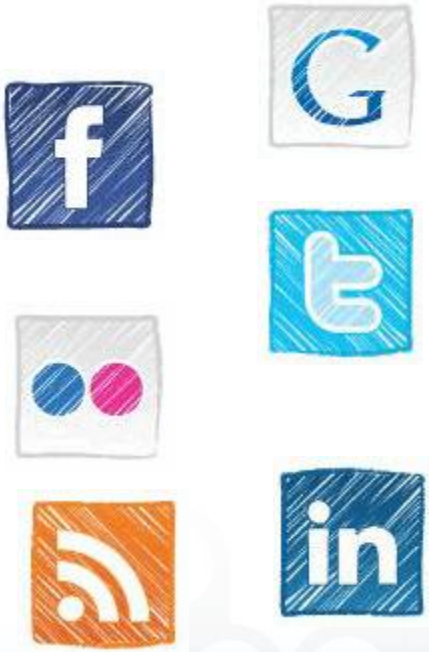
Public data….

- What this talk is about is freeing your data



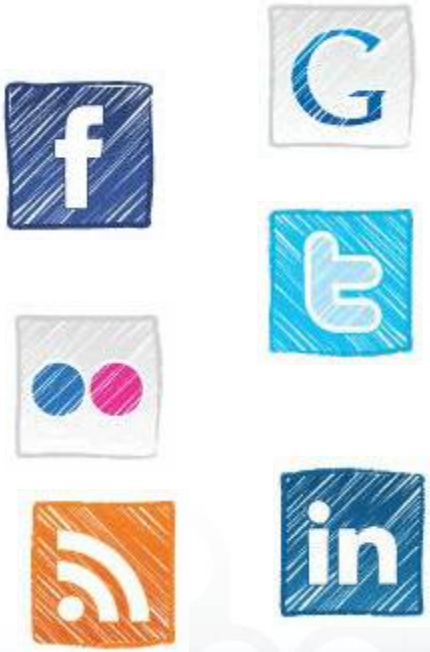
Enterprise data….

- What this talk is about is freeing your data



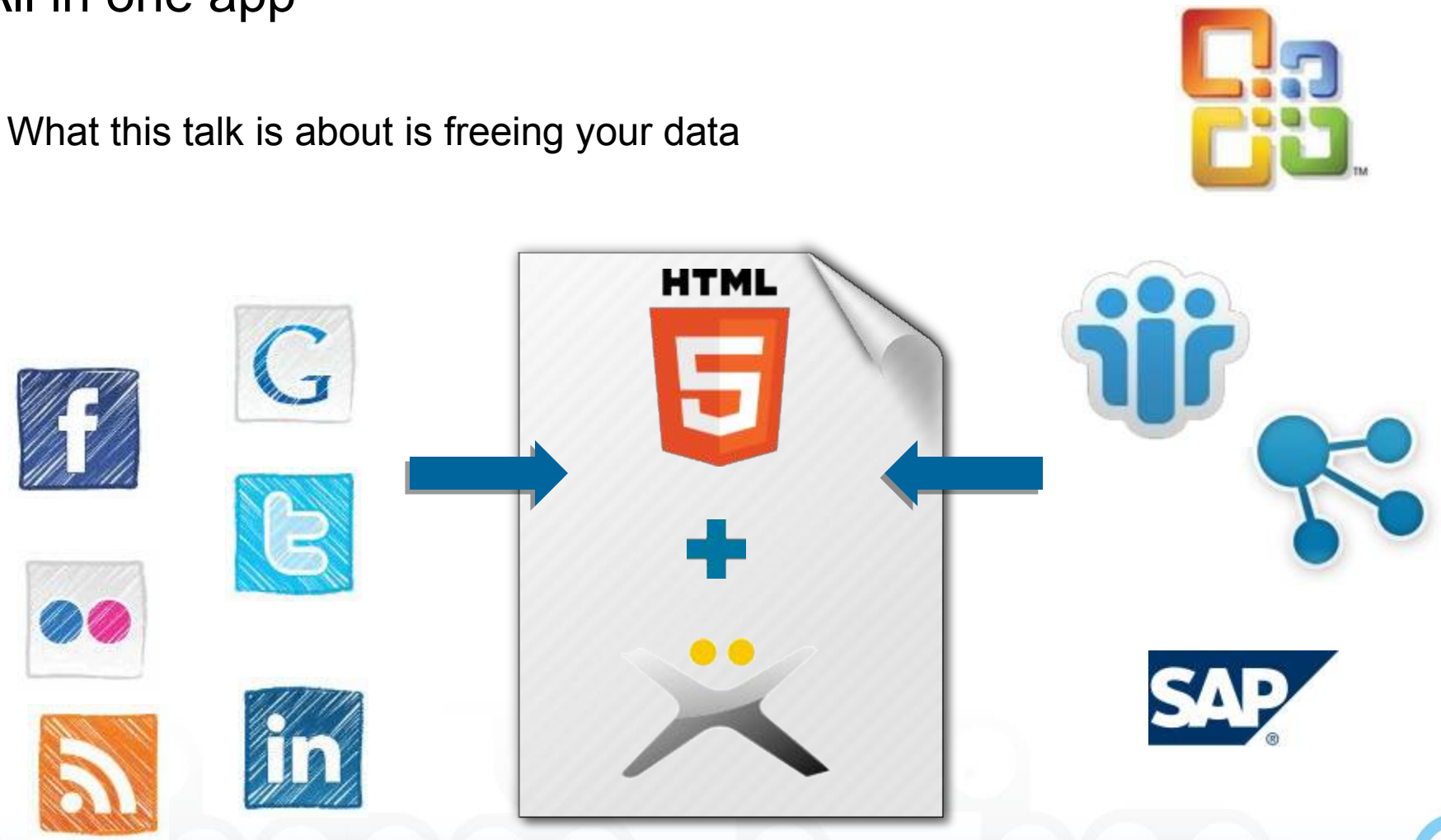
Bring it together…

- What this talk is about is freeing your data



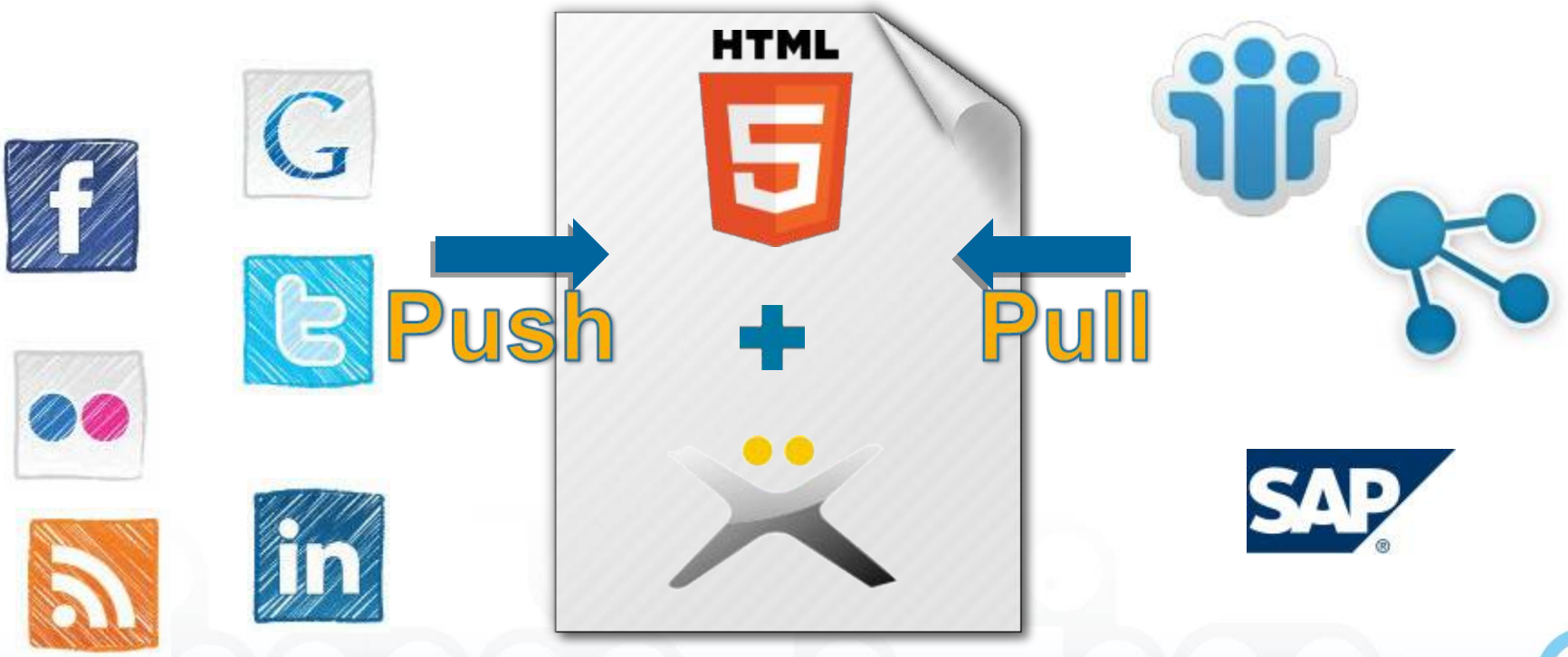
All in one app

- What this talk is about is freeing your data



How about sharing?

- What this talk is about is freeing your data



What are APIs

- APIs are a way to get access to data and functionality of your application without providing UI.
- They can be implemented in almost every way imaginable
  - Android apps use public java packages, etc.
  - Developing an iOS application, you use the iOS API via C header files.
  - Web applications also have APIs in the form of Web services, or even just URLS. A permalink is a form of web API.

Bottom line - An API makes it easier for other people or applications to access your data or features.

What are APIs

- APIs have been around a long time, and it gives others access to your system via a public, documented interface.
- Provides public paths to into more proprietary systems.
  - WIN32
  - Notes BE Objects
  - NotesCAPI
  - iOS, Android
- They give you the ability to create new applications on top of existing data / platforms.

This is not to hide all your secret sauce (though it could be)! It is more to keep things simple for your users. To give people access to the good stuff without burdening them with the details of how the back end works.

What is a *WEB API?*

A web **API** is nothing more that an **API** (access to your stuff) that people use over HTTP - AKA web services

Yesterday when you talked about web services, you were talking about SOAP Simple Object access protocol.

Today the trend these days is to build web services around REST (**RE**presentational **S**tate **T**ransfer). REST is *literally* what the wide web is based on.

Web Applications with public APIs

- Tell people what you had for lunch or the product you are about to launch:
  - Connections / Facebook / Twitter (https://dev.twitter.com/docs/api)
- Show your sales for the month:
  - Google Charts (https://developers.google.com/chart/)
- Send a text message from your web app
  - Twillio (http://www.twilio.com/docs/api/rest)
- Find pictures for your presentation?
  - Flickr (http://www.flickr.com/services/api/)
- Blog about your new product
  - WordPress.com REST API (http://developer.wordpress.com/docs/api/)
- Netflix
  - Pick a movie to watch now your app is launched! (http://developer.netflix.com/docs)

Why create an API?

- Break apart your application and the data.

By providing an API:

- You don't have to expose the nitty-gritty of how your app stores data.
- You create a common way to access your data.
- APIs force you to separate your client related code from your data related code. Then by doing this, anyone can create a client. And the client can be anything - mobile, desktop, web, native, anything.
- If you are thinking about creating mobile versions of your applications, having an API to access the data will be huge.

Why create an API?

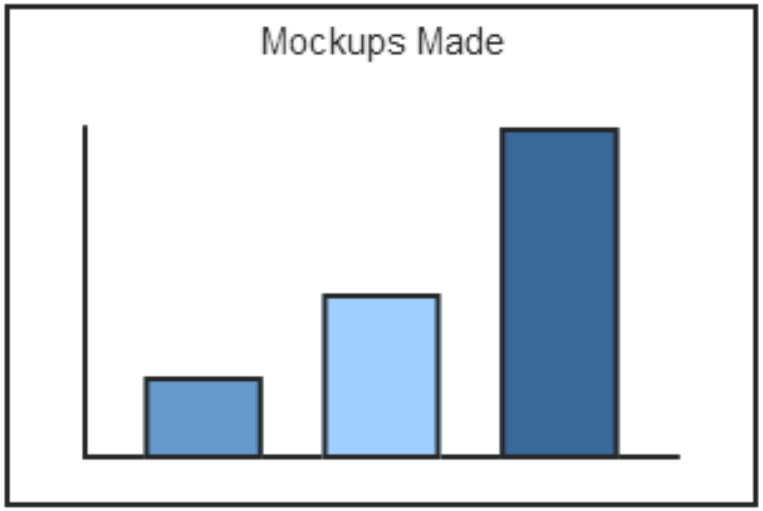
- Don't get caught in the idea that XPages are the only way to access Domino data. API access to your data can free you from a lot of constraints.
- A simple document focused API for example will make it easy to create any sort of mobile app you want. Want to create native iOS app? Great! Want to create a mobile web app? Fantastic! Do you need a desktop application? Right on! In all these cases, the server side has not changed at all. Your client app is merely a wrapper around your API!

Examples

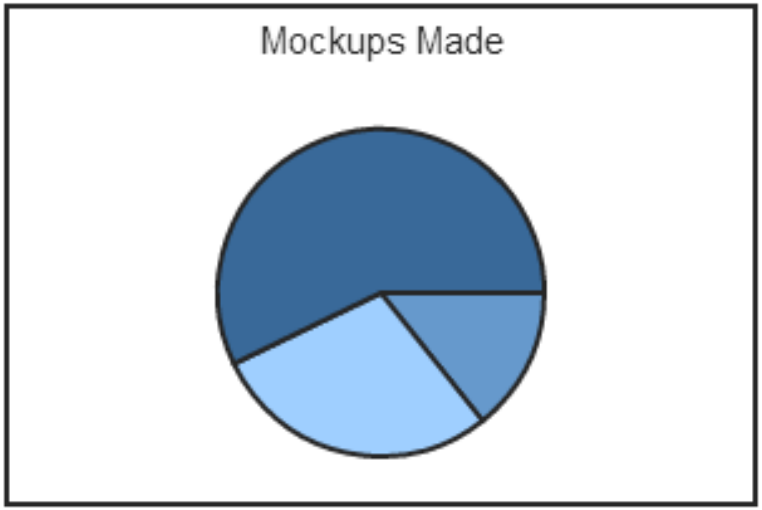
Example - Performance

So lets say I have a web page with this

Total revenue



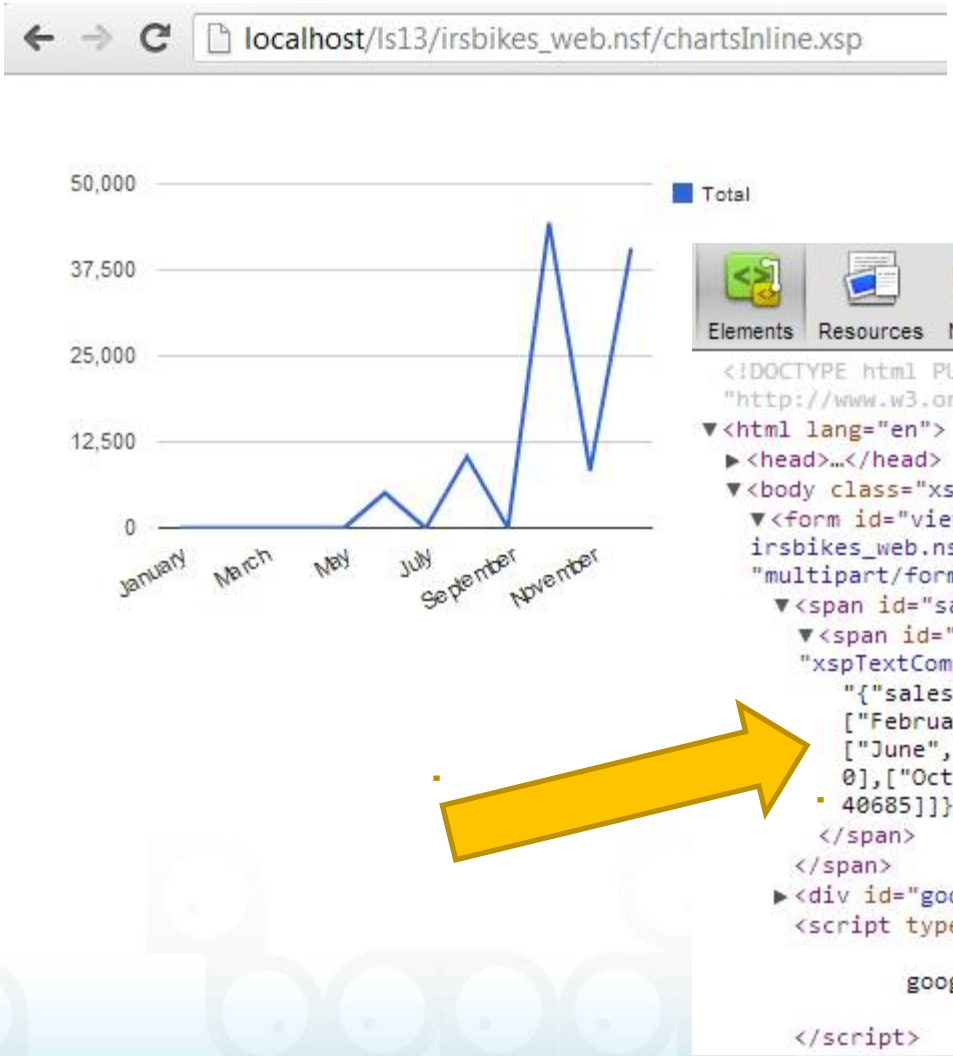
Sales By Department



Embedded in page

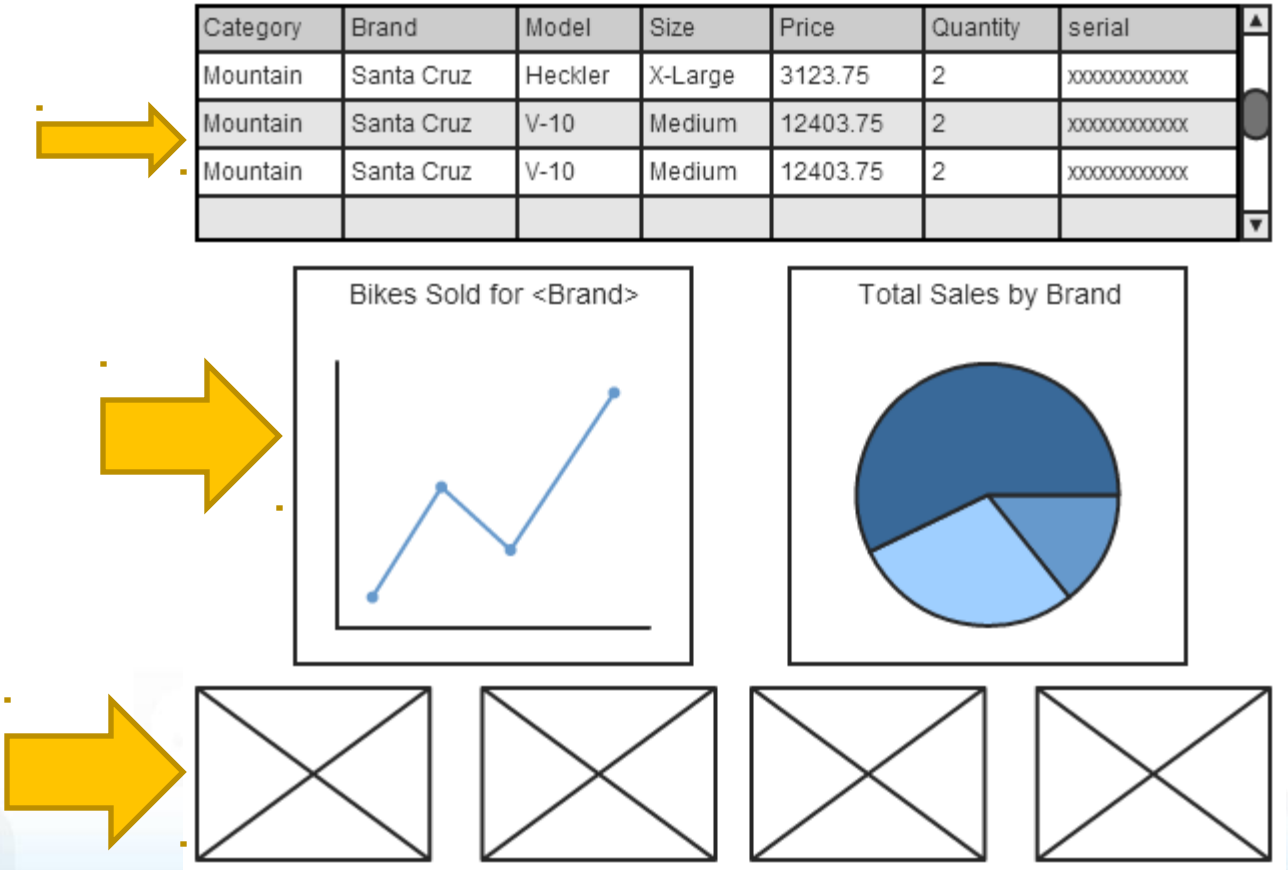
```
<xp:text id='salesData' style="display:none;" escape="true" >
        <xp:this.value><![CDATA[#{javascript:
              var val = "";

              // 12 months of sales numbers
              var mSales = [0,0,0,0,0,0,0,0,0,0,0,0];
              //walk the view, get month of sale and add to list
              var doc = salesVw.getFirstDocument();
              while( null != doc ){
                    var saledate = doc.getItemValue("saledate").elementAt(0);
                    saledate = new Date( saledate.toJavaDate() );
                    month = saledate.getMonth();
                    mSales[month]+=doc.getItemValueInteger("totalprice");
                    doc = salesVw.getNextDocument(doc);
              }

              val +="{\"sales\" : [ ";
              val += "[\"Month\", \"Total\"],";
              val += "[\"January\", " + mSales[0] + "],";
              val += "[\"February\", " + mSales[1] + "],";
              val += "[\"March\", " + mSales[2] + "],";
              val += "[\"April\", " + mSales[3] + "],";
              val += "[\"May\", " + mSales[4] + "],";
              val += "[\"June\", " + mSales[5] + "],";
              val += "[\"July\", " + mSales[6] + "],";
              val += "[\"August\", " + mSales[7] + "],";
              val += "[\"September\", " + mSales[8] + "],";
              val += "[\"October\", " + mSales[9] + "],";
              val += "[\"November\", " + mSales[10] + "],";
              val += "[\"December\", " + mSales[11] + "]";
              val += "]}";

              return val;
        }]]></xp:this.value>
</xp:text>
```

Embedded in page



Working with data

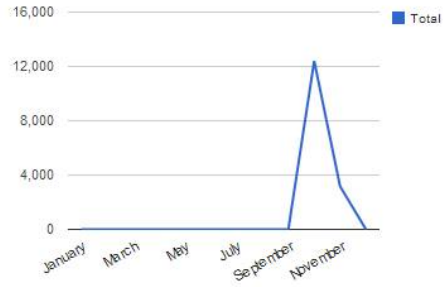
How about this?



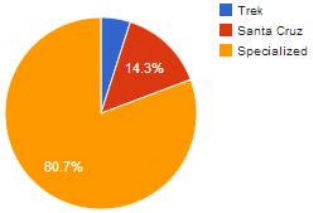
Previous 1 | 2 | 3 Next

Category	Brand	Model	Size	Price	Quantity	serialnumber
Mountain	Santa Cruz	Heckler	X-Large	3123.75	2	4561213843135435
Mountain	Santa Cruz	Tallboy	Small	6748.75	1	65461316876832
Mountain	Santa Cruz	V10	Medium	12403.75	2	494132745737
Mountain	Specialized	Demo	Medium	8250.0	1	a4546846be4687464354
Mountain	Specialized	Demo	Medium	8250.0	1	14357386435787
Mountain	Specialized	Demo	Small	8250.0	2	87983138431384
Mountain	Specialized	Epic 29er	Small	12500.0	1	165431384341387
Mountain	Specialized	Epic 29er	Large	12500.0	1	4156431854135135
Mountain	Specialized	Epic 29er	Medium	12500.0	1	54643218a54f56q
Mountain	Specialized	Stumpjumper FSR	Large	5125.0	1	11684357864321687

Sales for Santa Cruz



Sales by brand



Related Flickr Images

Working with data

- Start with a basic Notes view in XPages
- Add an *onclick* event that ties into a **client side** JavaScript event

Category	Brand	Model	Size	Price
Mountain	Santa Cruz	Heckler	X-Large	3123.75
Mountain	Santa Cruz	Tallboy	Small	6748.75
Mountain	Santa Cruz	V10	Medium	12403.75
Mountain	Specialized	Demo	Medium	8250.0
Mountain	Specialized	Demo	Medium	8250.0
Mountain	Specialized	Demo	Small	8250.0
Mountain	Specialized	Epic 29er	Small	12500.0
Mountain	Specialized	Epic 29er	Large	12500.0
Mountain	Specialized	Epic 29er	Medium	12500.0
Mountain	Specialized	Stumpjumper FSR	Large	5125.0

Working with data

- Start with a basic Notes view in XPages
- Add an ***onclick*** event that ties into a **client side** JavaScript event

```
<xp:viewPanel var="rowData" rows="10" id="viewPanel1"
viewStyleClass="viewPanel" rowClasses="viewPanelRow">
   <xp:this.facets>
          <xp:pager partialRefresh="true" layout="Previous Group Next"
                    xp:key="headerPager" id="pager1" for="viewPanel1">
          </xp:pager>
   </xp:this.facets>
   <xp:this.data>
          <xp:dominoView var="view1"
                    databaseName="ls13\irsbikes_inventory.nsf" viewName="demoView">
          </xp:dominoView>
   </xp:this.data>
   <xp:this.rowAttrs>
          <xp:attr
              value="javascript:loadDocument('#{javascript:rowData.getUniversalID()}')"
              name="onclick">
          </xp:attr>
   </xp:this.rowAttrs>
[.....]
</xp:viewPanel>
```

Working with data

- The ***onclick*** event then triggers 3 things to happen
  - The full document is fetched in JSON format
  - One of the Google charts is updated with the sales for that brand of bike
  - Flickr is searched for images tagged with the brand and model of bike selected

```
function loadDocument(unid){
    // get the document we have selected
    getNotesDocument( unid, "ls13/irsbikes_inventory.nsf", function(doc){
        // get related flickr images
        flickerAPIImages(doc.getItem("brand")+","+doc.getItem("model"));
        // update sales data
        googleChartsAPI(doc.getItem("brand"));
    })
}
```

Working with data

- Step one is to get the document…
- Domino *can* do this for you, but it's better to have a little more control over what you get.

```
function getNotesDocument( docid, db, callback ){
    var url = './apiv1.xsp?f=document';
    url += "&db=" + db;
    url += "&documentId=" + docid;
    $.getJSON(url,function(data){
        var doc = new NotesDocument(data);
        callback(doc);
    });
}
```

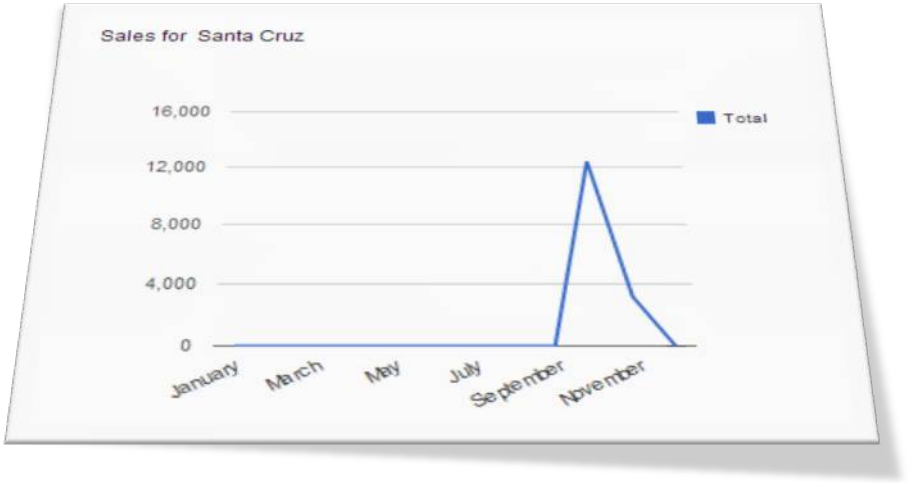
Working with data

- When you control the API, you control the data and how it is represented.

```
function getDocument(){
    var db:NotesDatabase = session.getDatabase("", param['db']);
    var doc:NotesDocument = db.getDocumentByUNID(param['documentId']);

    try{
      var docXML = doc.generateXML();
      var writer = facesContext.getResponseWriter();
      var excon = facesContext.getExternalContext();
      excon.getResponse().setContentType("application/json");
      writer.write( org.json.XML.toJSONObject(docXML).toString());
    }
    catch(e){
        print(e.toString())
    }
}
```

Working with data

- The chart is updated with an API call to get the data needed
- The chart is drawn with the Google charts engine using the data that is returned




```
function googleChartsAPI(brand){
    var url = "./apiv1.xsp?f=sales&k=months";
    if( null != brand && "" != brand ){
        url += "&brand=" + brand;
    }

    $.getJSON(url, function(d) {
      [Google charts code]
    })

}
```

Working with data

- The flicker search is done the same way.

Working with data

- The flicker search is done the same way…

```
function flickerAPIImages(search){
/*
 * Flickr Photo API
 * http://www.flickr.com/services/api/flickr.photos.search.html
 */

   search = search.replace(/,/g,"+");
   //
   var URL = "http://api.flickr.com/services/rest/?method=flickr.photos.search";
   url += "&api_key=<your api key goes here>";
   url += "&tags=cycling%2C+mountain&text="+search+"+%20bikes";
   url += "&per_page=5&format=json&nojsoncallback=1";
   $.getJSON( URL, function(d) {
          var source   = $("#entry-template").html();
          var template = Handlebars.compile(source);
          var context = d.photos;
          var html     = template(context);
          $('#imageInner').html(html);
   })
}
```

Working with data

- XPages - server side is not the place to do it
  - Rendering a page should not wait for a response from some other site.
  - Things get messy quick with a mix of server side and client side code updating the page
- In general don't fill your page with tons of *data*
  - Valuable data is trapped and mixed in with the rest of the UI
  - There is a lot of different *kinds of data* here, from multiple sources.
- Partial refresh is not always the answer either.
  - Hard to chain things together so one value affects many others
  - could render the whole page

Data trapped in the UI???

- What happens when the client isn't always a browser?
- Other clients might not want HTML

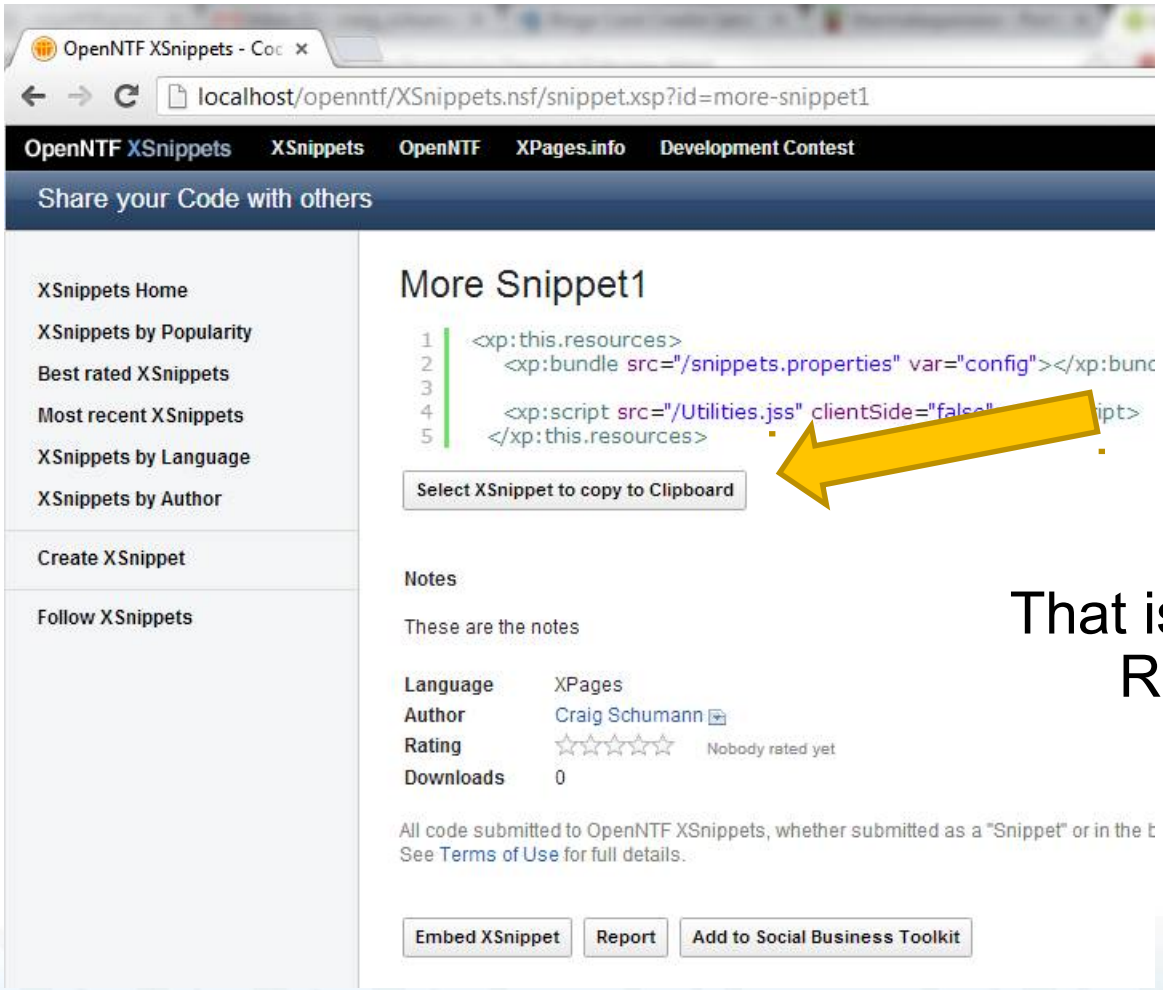
- Example - XSnippets

Example - XSnippets

- XSnippets – OpenNTF.org project
  - http://openntf.org/XSnippets.nsf/home.xsp
  - Template can be downloaded and customized at http://xsnippets.openntf.org/

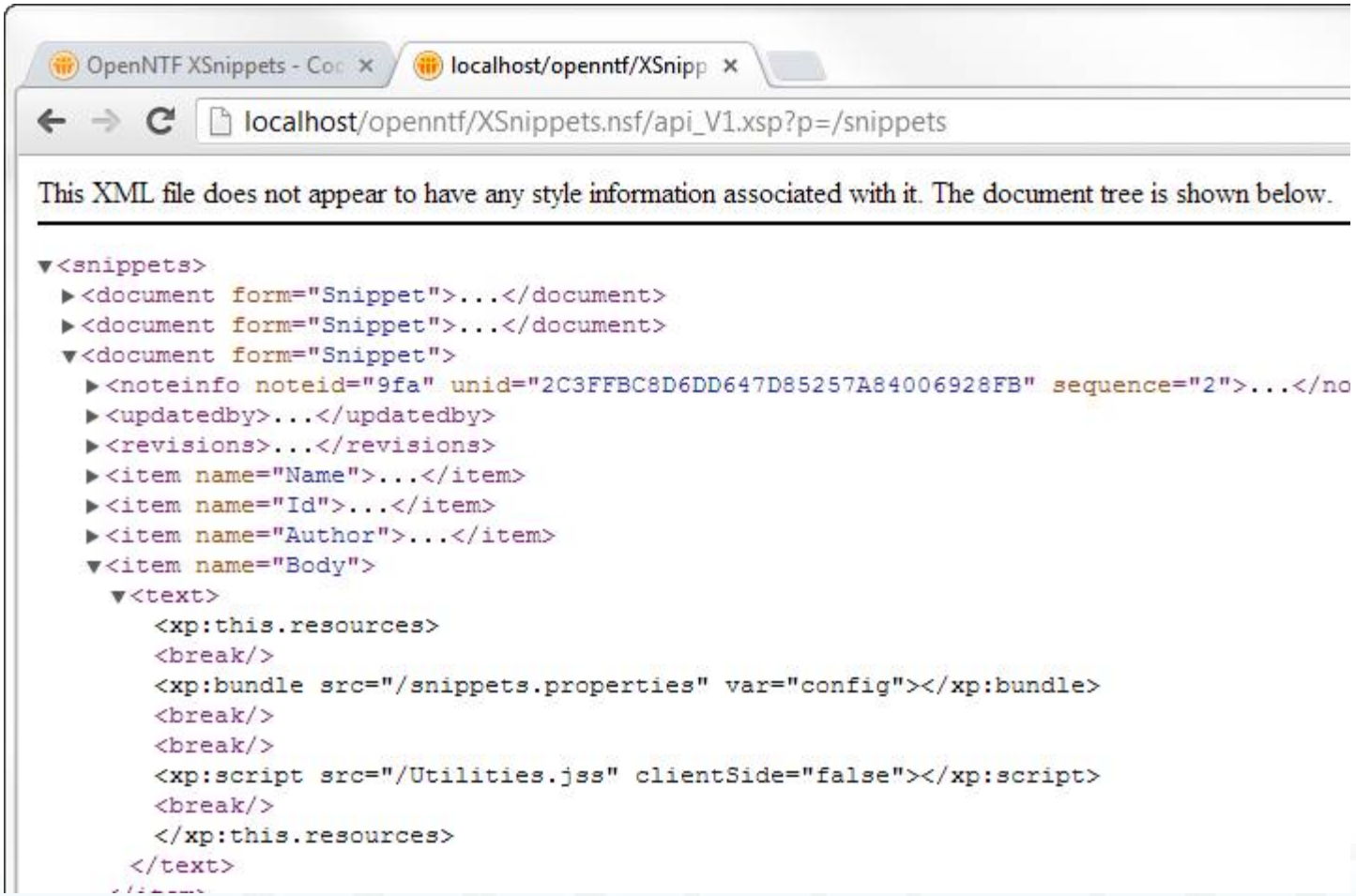
- Currently there is only a web interface

XSnippets – browser client

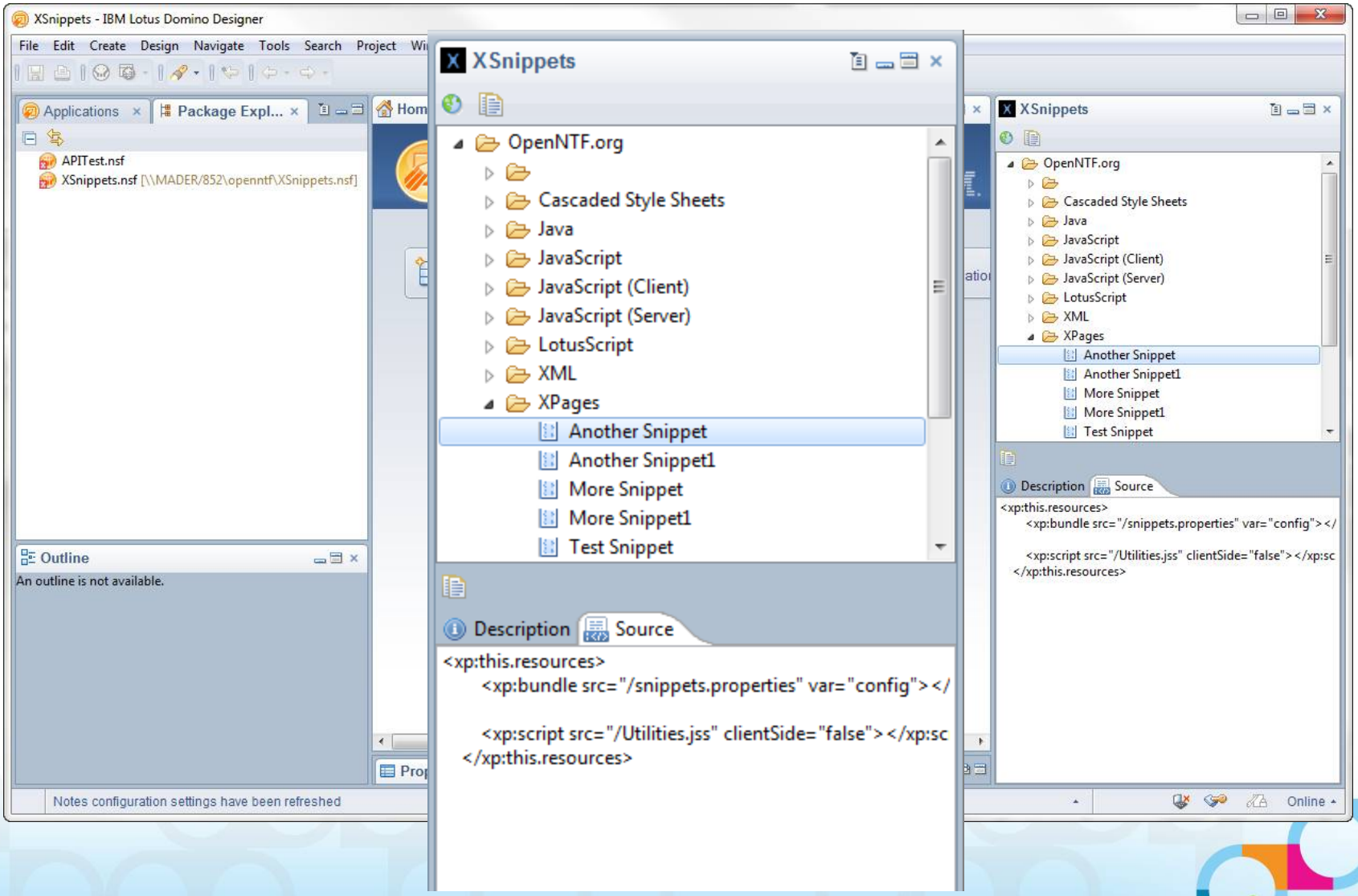


That is enough Right?

XSnippets API



XSnippets – Designer client!



Design Guidelines

Options

- **SOAP**
  - ‘Classic’ Web services, Strong "object model"
  - Message based / transactional
  - Hard to do simple things like getting a list of things
  - been there done that....
- **REST**
  - resources / Object model based
  - resources can be anything you can create a URI for.
  - path oriented
    - database/collection (like a view)
    - database/collection/entity (like a document)

REST

REST is not a standard. REST is a style, a way to build a web application.

That being said, REST does make use of standards. HTTP, XML, URLs

REST works with resources as collections and objects. A resource can be anything you are interested in. For example:

www.example.com/customers/IBM

Would be a request for the IBM customer resource. A web server can respond with a *representation* of the resource IBM.html for example, an XPage customer.xsp, or just an XML or JSON document with the IBM specific data.

Notice there is no “VERB” in the resource URI….

REST and URLs

The key difference with a RESTfull service and a SOAP service is the way you use the HTTP *verbs.*

SOAP encourages the method to define the verb - getCustomers.

REST uses the HTTP methods* to define the operation

GET /customers

* http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

REST - *GET*

You have probably head about HTTP GET. GET is the most basic action a client can do. With GET the client is requesting a read only copy of a resource.

REST - *POST*

- You are also probably familiar with POST. This the most common thing to do when sending data back to the server - submitting a form for example.
- POST is should be used when **creating or updating** resources when the resulting URI is not known.
- RESTfull services will also make use of PUT and DELETE, and this is where a true RESTfull service will diverge a bit from the common use of POST.

REST - *PUT*

- PUT is used when you are updating an ***existing*** resource.
  - PUT is used when you know the path to the resource.
  - PUT can create resources if one does not already exist

- POST vs PUT
  - POST in form submission because you don't know what the URL to the new resource is going to be.
  - PUT requires the URL to resource to be **known**
  - HTML **FORMs** only support *GET* and *POST*

REST - *DELETE*

DELETE does what it says on the tin - deletes a resource.

REST *verbs*

- Client side JavaScript with Dojo

```
dojo.xhrPut ({
    url: '/api_v1.xsp',
    putData: 'data to send to server',
    handleAs: 'text',
    error: function(result) {
        // Whoops
    }
});
```

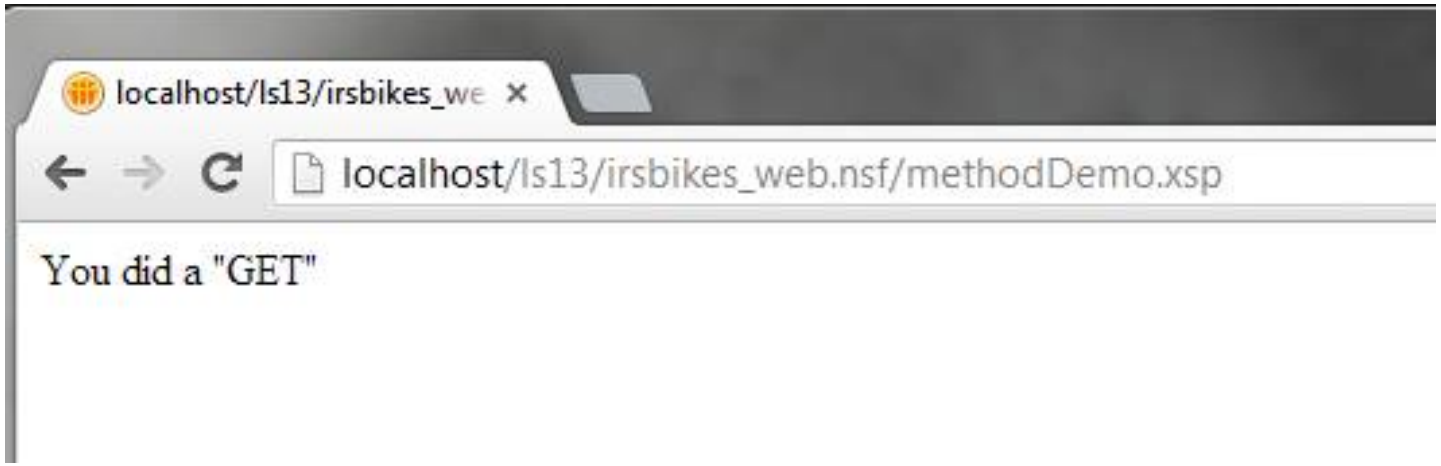
- Java

```
URL url;
try {
    url = new URL(uri);
    HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("PUT");
}
```

REST *verbs* – XPages example

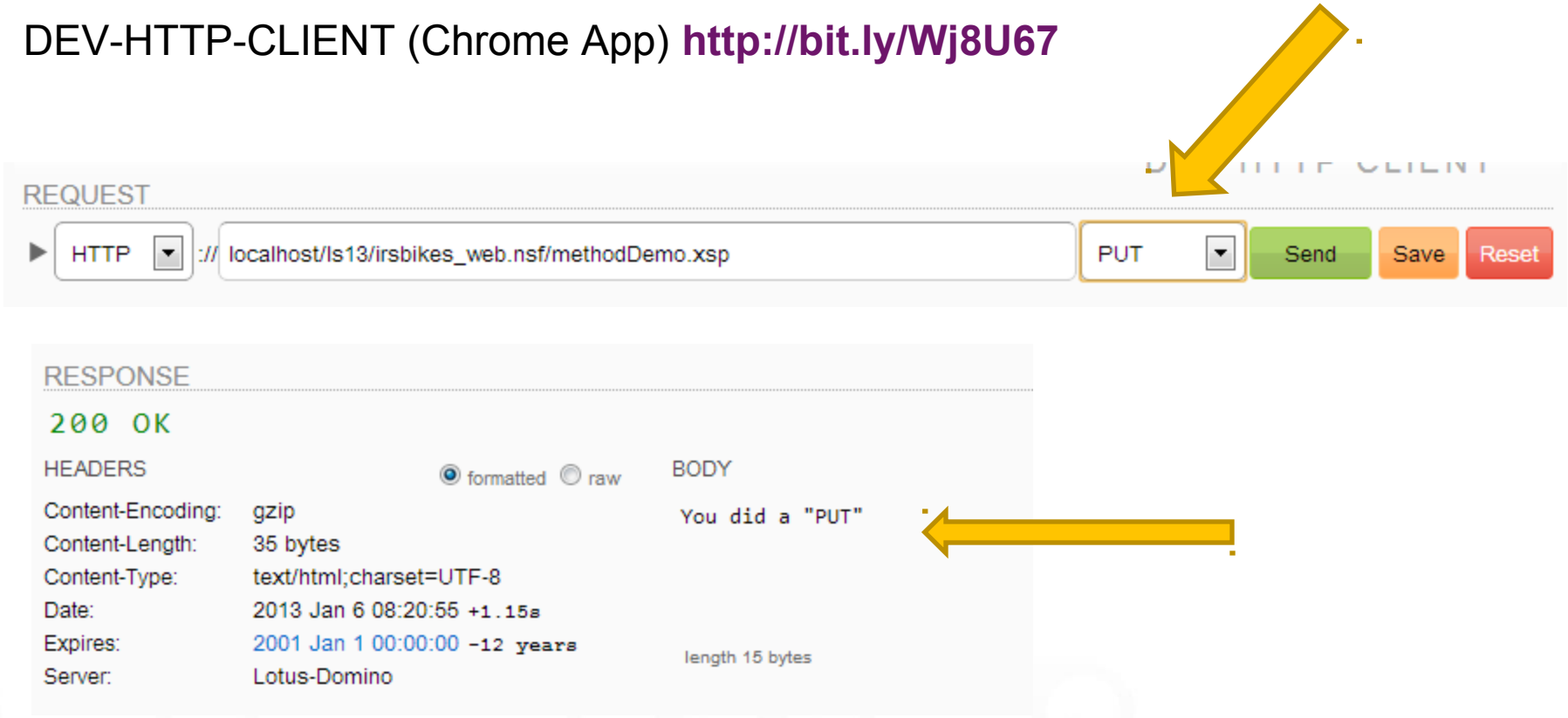
```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" rendered='false'>

  <xp:this.afterRenderResponse><![CDATA[#{javascript:

        var request = facesContext.getExternalContext().getRequest();
        var writer = facesContext.getResponseWriter();

        var method = request.getMethod();
        if( method === "GET" ){
           writer.write("You did a \"GET\"");
        }
        else if( method === "POST" ){
           writer.write("You did a \"POST\"");
        }
        else if( method === "PUT" ){
           writer.write("You did a \"PUT\"");
        }
        else if( method === "DELETE" ){
           writer.write("You did a \"DELETE\"");
        }

   }]]></xp:this.afterRenderResponse>

</xp:view>
```

REST *verbs* – XPages example



REST *verbs* – XPages example

DEV-HTTP-CLIENT (Chrome App) **http://bit.ly/Wj8U67**



REST - resources

- REST uses HTTP to provide the VERB to the resource
- Since the verb (get,post,etc) is part of the request, you shouldn't use verbs in the URL to the resource, only **nouns**.
- If your data has Objects like customers, projects, meetings, etc. the REST API should use the *same* **object names**

www.example.com/**customers**?id=IBM

vs

www.example.com/getCustomer?customer=IBM

REST

URLs should point to logical resources, not physical.

www.example.com/customers/IBM

vs

www.example.com/customers/IBM.html

These kinds of URLs are not easy in Domino. Luckily REST is a **design principle**, and not a standard!
You can ***still*** design a RESTfull api with Domino.

RESTful data

RESTful APIs typically don't deal in HTML. HTML is UI, Your REST API shouldn't be worrying about or injecting **UI**

- Don’t include HTML in your RESTful data
- Strictly speaking, the format for data can be anything, but it’s usually:
  - XML
  - JSON/JSONP (AKA XML with curly braces).

REST and errors

Use error codes in the response

- 2xx == all good
- 4xx == client messed something up
- 5xx == server messed something up

RESTful errors

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" rendered='false'>

  <xp:this.afterRenderResponse><![CDATA[#{javascript:
  var externalContext = facesContext.getExternalContext();
  var response = externalContext.getResponse();
  var request = externalContext.getRequest();

  if( !param.containsKey("p") ){
    response.setStatus(400);
    response.setContentType("application/json");
    var writer = facesContext.getResponseWriter();
    writer.write("Missing method call parameter 'p'");
    return;
  }

  var s = new org.openntf.api.snippet();
  s.service( request, response);
  }]]>
</xp:this.afterRenderResponse>
</xp:view>
```

API Versioning

You must version your API!

Versioning your API means that you don't need to update all your clients immediately, if at all, when you add or change your API**.

Make the version part of the resource, not a parameter. Make it explicit.

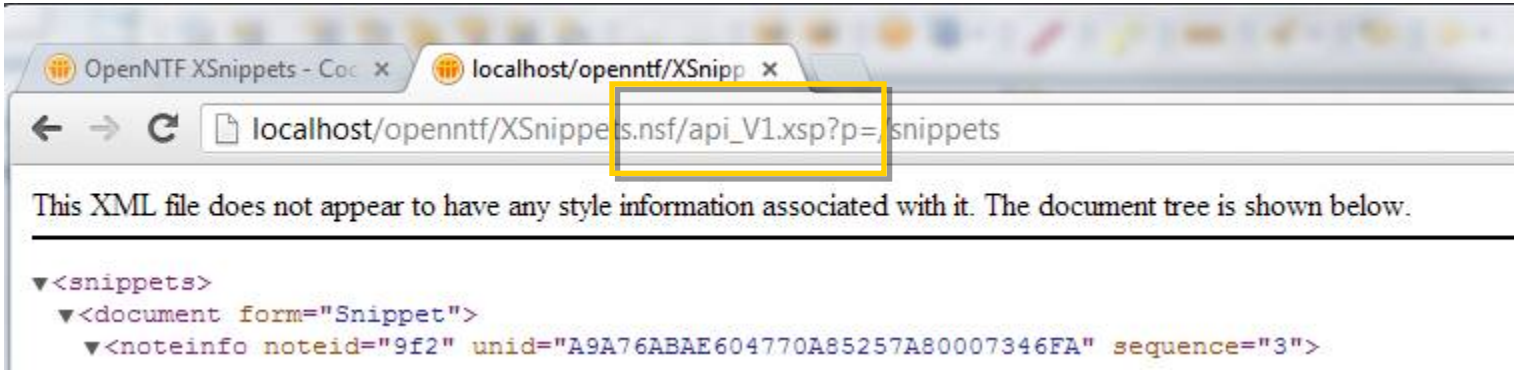
www.example.com/**v1**/customers/IBM

vs

www.example.com/customers/IBM?v=1

****You will change it, so just get it out of the way and design it into V1**

API Versioning



API Authentication

A RESTfull application should also be ***stateless***. So this means that you should not be storing session information server side. This is Domino so you don't have much control over this, but in general you shouldn't be using *appScope* or *sessionScope* variables to store anything.

- **Public / No authorization needed**
- **HTTP Authentication**
  - Server handles the login
  - All HTTP requests must contain the login info via Authorization header
  - Database ACL is set to require authentication (Anonymous=No Access)
- **Token Authentication**
  - You handle the login
  - an API key is generated for client access. That key is used as the login credentials. Not really security, but limits access to the data to a known set of users and gives application control over who has access without having to manage the ACL
  - Database ACL is more open

Get your own!

Old school

- &ReadViewEntries
- Supported since R6
- Gives you a dump of a view in XML or JSON format

- Pros
  - Built in so performance is good
  - No coding needed
  - XML is a little more useful than the JSON version
- Cons
  - No control over the output
  - Does not support JSONP
  - There are a few bugs that can cause the JSON output to be invalid
  - There is no equiv for docs so everything you need must be in the view
  - Not really an API..

Old School

- demoView?readviewentries

```
▼<viewentries timestamp="20130106T122615,36Z" toplevelentries="22">
  ▼<viewentry position="1" unid="6783AB7D38FACE2A85257AD9005BDC35" noteid="906" siblings="22">
    ▼<entrydata columnnumber="0" name="category">
       <text>Mountain</text>
     </entrydata>
    ▼<entrydata columnnumber="1" name="brand">
       <text>Santa Cruz</text>
     </entrydata>
    ▼<entrydata columnnumber="2" name="model">
       <text>Heckler</text>
     </entrydata>
    ▼<entrydata columnnumber="3" name="Size">
       <text>X-Large</text>
     </entrydata>
    ▼<entrydata columnnumber="4" name="price">
       <number>3123.75</number>
     </entrydata>
    ▼<entrydata columnnumber="5" name="quantity">
       <text>2</text>
     </entrydata>
    ▼<entrydata columnnumber="6" name="serialnumber">
       <text>4561213843135435</text>
     </entrydata>
  </viewentry>
  ▼<viewentry position="2" unid="DD41ED147028B86285257AD9005BE3A1" noteid="93A" siblings="22">
    ▼<entrydata columnnumber="0" name="category">
```

Old School

- demoView?readviewentries&outputformat=json


```
{
"@timestamp": "20130106T122907,95Z",
"@toplevelentries": "22",
"viewentry": [
{
"@position": "1",
"@unid": "6783AB7D38FACE2A85257AD9005BDC35",
"@noteid": "906",
"@siblings": "22",
"entrydata": [
{
"@columnnumber": "0",
"@name": "category",
"text": {
"0": "Mountain"
}
},
{
"@columnnumber": "1",
"@name": "brand",
"text": {
"0": "Santa Cruz"
}
},
{
"@columnnumber": "2",
"@name": "model",
"text": {
"0": "Heckler"
```

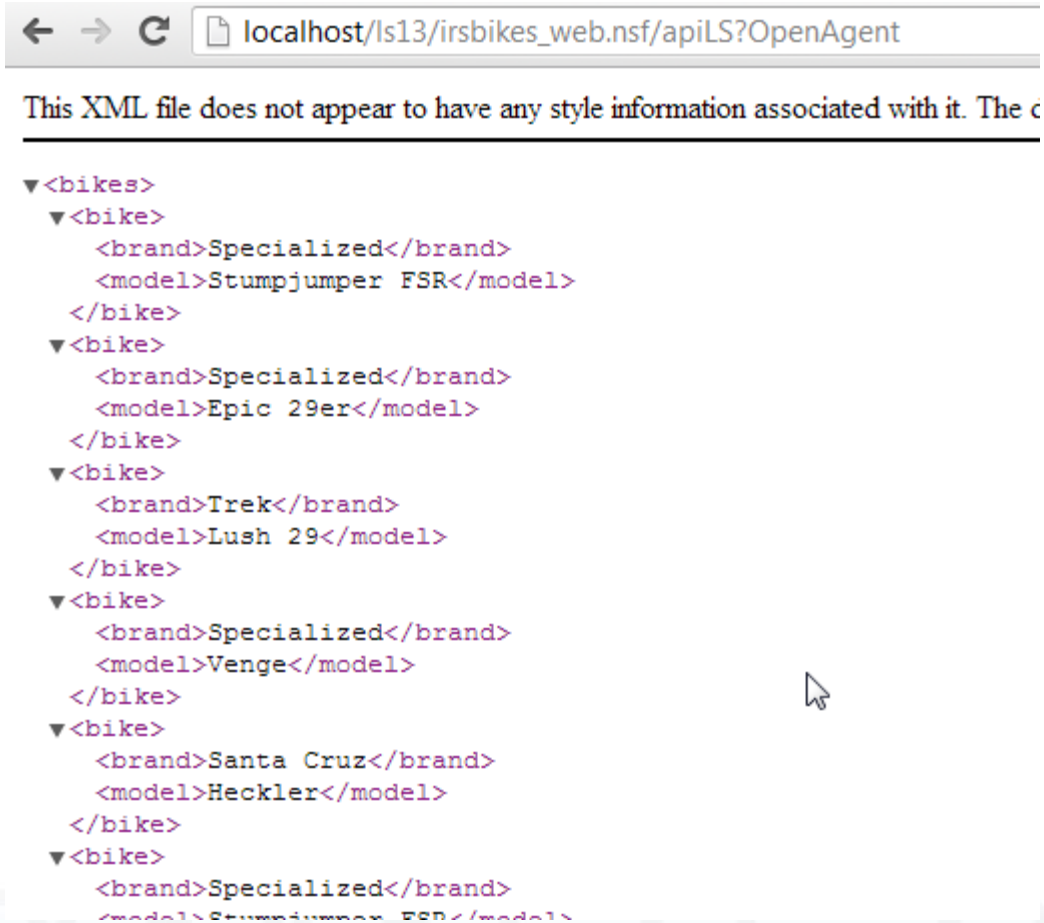
POA (Plain old Agents)

- These don’t require anything special and can do all the same things we talked about previously.
- Can use Java or LotusScript.

POA (Plain old Agents)

```
Sub Initialize
    Dim s As New NotesSession

    Dim db As New NotesDatabase("", "ls13/irsbikes_inventory.nsf")

    Dim docs As NotesDocumentCollection
    Set docs = db.AllDocuments
    Dim d As NotesDocument
    Set d = docs.getfirstdocument
    Print "Content-Type:application/xml"
    Print |<?xml version="1.0" encoding="UTF-8"?>|
    Print |<bikes>|
    while Not d Is Nothing
        Print "<bike>"
        Print "<brand>" + d.brand(0) + "</brand>"
        Print "<model>" + d.model(0) + "</model>"
        Print "</bike>"
        Set d = docs.Getnextdocument(d)
    Wend
    Print |</bikes>|
End Sub
```

POA (Plain old Agents)



POA (Plain old Agents)

```
public class JavaAgent extends AgentBase {

    public void NotesMain() {

      try {
          Session session = getSession();
          AgentContext agentContext = session.getAgentContext();

          Database db = session.getDatabase("", "ls13/irsbikes_inventory.nsf");
          DocumentCollection docs = db.getAllDocuments();
          Document doc = docs.getFirstDocument();

          java.io.Writer sw = new java.io.StringWriter();
          PrintWriter ao = getAgentOutput();
          ao.println("Content-type: text/xml");
          ao.println("<bikes>");
          while(null!=doc){
             // write the document out
             doc.generateXML(sw);
             doc = docs.getNextDocument(doc);
          }
          sw.flush();
          ao.print(sw.toString());
          ao.println("</bikes>");

      } catch(Exception e) {
          e.printStackTrace();
       }
    }
}
```

Domino and REST

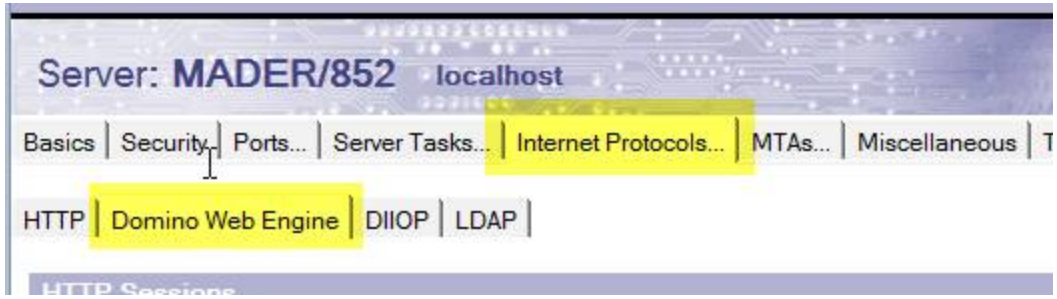
- Introduced in 8.5.3

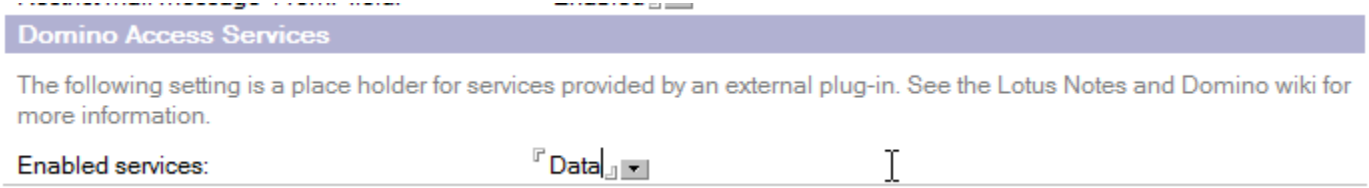
- 4 built in options
  - Domino Data Service
  - XPages REST Services Control
  - Custom Database Servlet
  - Custom Wink Servlet

Domino and REST

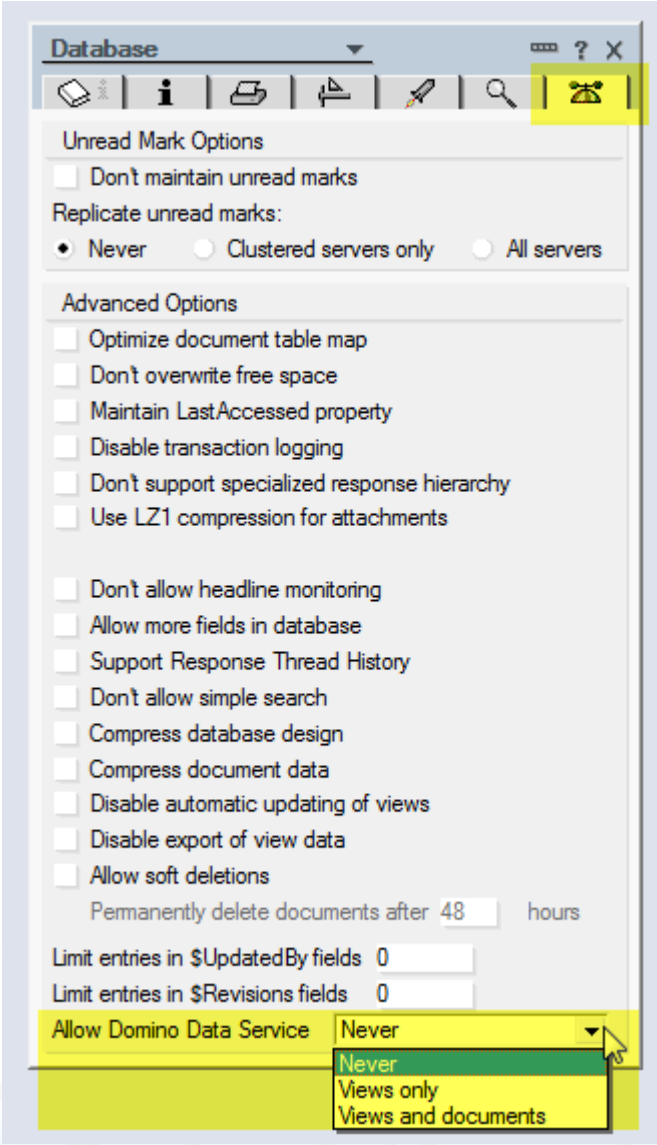
- Pros
  - Built in so support and performance is good
  - No coding needed
  - Very granular
- Cons
  - DDS Needs to be enabled on the server (admins need to be involved)
  - Very granular – database, view properties
  - URLs are not app specific. This is Domino's API = views and documents
  - No coding – can't really expose logical application objects
  - Not documented very well (yet)

Domino and REST – Domino Data Services

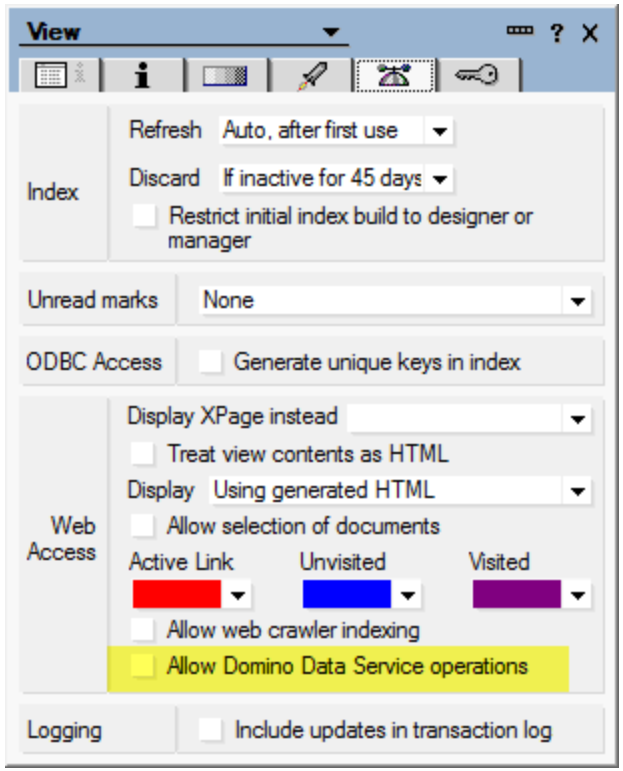




Domino and REST



Domino and REST – Domino Data Services



Domino and REST – Domino Data Services

- http://{host}/**api/data**


```
[
 {
    "@title":"Activity Trends (MADER)",
    "@filepath":"activity.nsf",
    "@replicaid":"852578220B751506",
    "@template":"StdActivityTrendsDatabase",
    "@href":"http:\/\/localhost:80\/activity.nsf\/API\/data\/collections"
 },
 {
    "@title":"Administration Requests",
    "@filepath":"admin4.nsf",
    "@replicaid":"8525782203751506",
    "@template":"StdR4AdminRequests",
    "@href":"http:\/\/localhost:80\/admin4.nsf\/API\/data\/collections"
 },
 ……
]
```

Domino and REST – Domino Data Services

- http://{host}/{path}**/api/data/collections**


```
[
 {
    "@title":"By Type",
    "@folder":false,
    "@private":false,
    "@modified":"2012-12-19T20:27:13Z",
    "@unid":"0AC52D85954AB7E785257AD9005D6E35",
    "@href":"http:\/\/localhost:80\/ls13\/irsbikes_inventory.nsf\/API\/data\/collections\/unid\/0AC52D85954AB7E785257AD9005D6E35"
 },
 {
    "@title":"By Size",
    "@folder":false,
    "@private":false,
    "@modified":"2012-12-19T20:27:13Z",
    "@unid":"66ED263FEEC2CF7485257AD9005D3342",
    "@href":"http:\/\/localhost:80\/ls13\/irsbikes_inventory.nsf\/API\/data\/collections\/unid\/66ED263FEEC2CF7485257AD9005D3342"
 }, ……
]
```

Domino and REST – Domino Data Services

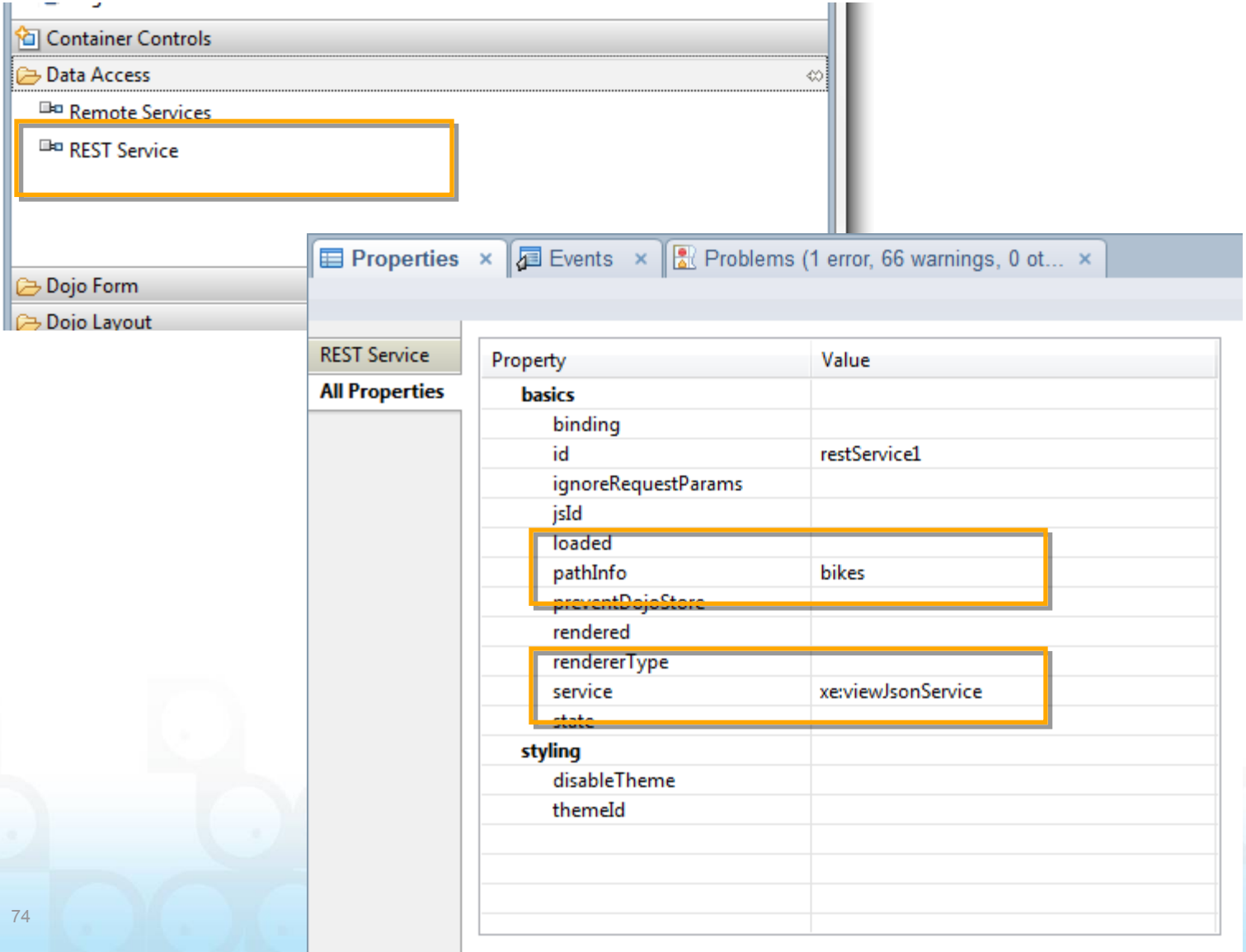
- http://{host}/{path}**/api/data/collections/name/{view name}**
- http://{host}/{path}**/api/data/collections/unid/{view unid}**


```
[
 {
    "@href":"http:\/\/localhost:80\/ls13\/irsbikes_inventory.nsf\/API\/data\/collections\/name\/demoView\/unid\/6783AB7D38FACE2A85257AD9005BDC35",
    "@link":
    {
       "rel":"document",
       "href":"http:\/\/localhost:80\/ls13\/irsbikes_inventory.nsf\/API\/data\/documents\/unid\/6783AB7D38FACE2A85257AD9005BDC35"
    },
    "@entryid":"1-6783AB7D38FACE2A85257AD9005BDC35",
    "@unid":"6783AB7D38FACE2A85257AD9005BDC35",
    "@noteid":"906",
    "@position":"1",
    "@siblings":22,
    "@form":"inventory",
    ….
 },
…..
]
```

Domino and REST – REST Services Control

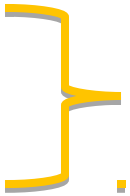
- The REST Services Custom Control comes with the Extension Library.
- No Coding!
- Add to an XPage and set the ‘pathInfo’ paramter.
- Different types of resources can be retrieved
  - databaseCollectionJsonService - Read the list of databases on the server.
  - viewCollectionJsonService - Read the list of views and folders in a database.
  - viewJsonService - Read the entries in a view or folder.
  - documentJsonService - Create, read, update and delete documents.
- Adds to the path for the XPage for specific resources

Domino and REST – REST Services Control



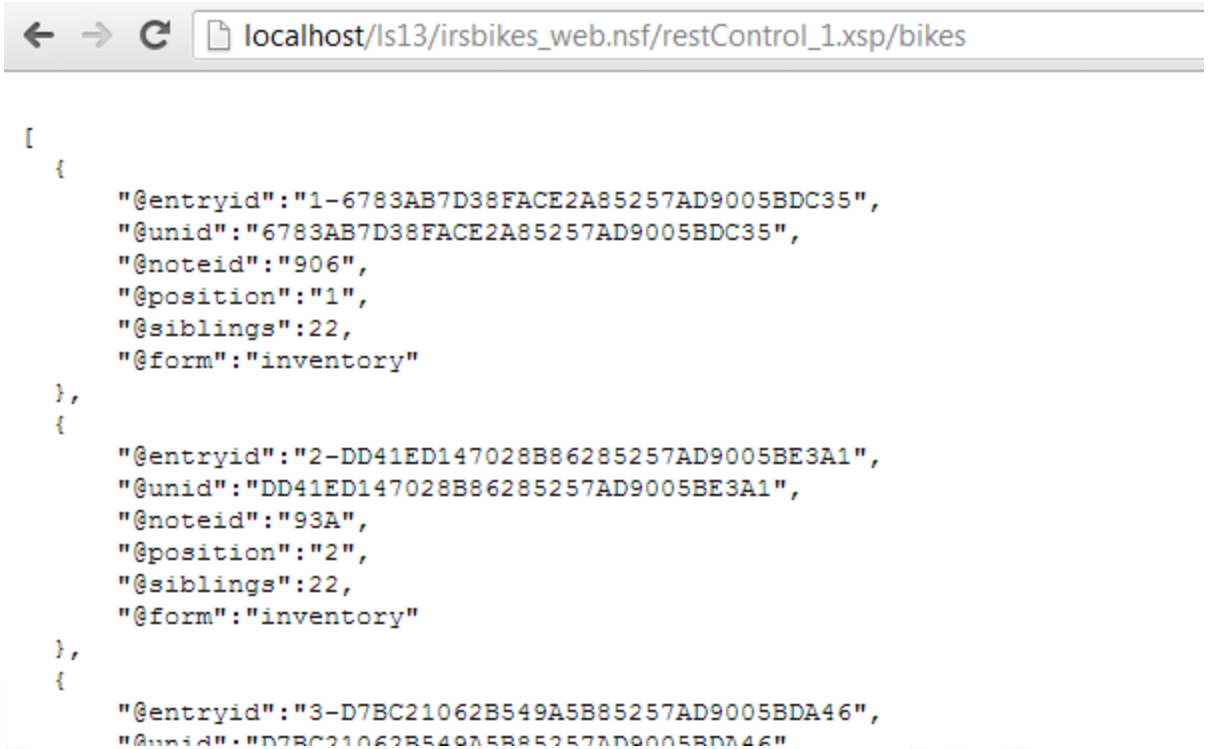
Domino and REST – REST Services Control

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core"
         xmlns:xe="http://www.ibm.com/xsp/coreex">


   <xe:restService id="restService1" pathInfo="bikes">
        <xe:this.service>
           <xe:viewJsonService
                 databaseName="ls13/irsbikes_inventory.nsf"
                 viewName="demoView">
           </xe:viewJsonService>
        </xe:this.service>
    </xe:restService>


</xp:view>
```



Domino and REST – REST Services Control

http://localhost/ls13/irsbikes_web.nsf/restControl.xsp/**bikes**



Domino and REST – Domino Data Services

- More on OpenNTF
  - Niklas Heidloff - http://www.openntf.org/blogs/openntf.nsf/d6plinks/NHEF-8JGGPE
    - Documentation
    - Examples
    - Video

“XAgents”

- XPages with *rendered=‘**false**’*
- Can write your API in Java or Server Side JavaScript
- Keeps you in the same frame of mind as the rest of your XPages development.

“XAgents”

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" rendered='false'>

<xp:this.afterRenderResponse><![CDATA[#{javascript:
    var externalContext = facesContext.getExternalContext();
    var response = externalContext.getResponse();
    var request = externalContext.getRequest();

    if( !param.containsKey("p") ){
      response.setStatus(400);
      response.setContentType("application/json");
      var writer = facesContext.getResponseWriter();
      writer.write("{ \"code\":400, \"text\":\"Bad Parameter\",
      \"message\":\"Missing method call parameter 'p'\" }");
      return;
    }

    var s = new org.openntf.api.snippet();
    s.service( request, response);
}]]>
</xp:this.afterRenderResponse>
</xp:view>
```

“Xagents” + JAVA = Happy

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" rendered='false'>

<xp:this.afterRenderResponse><![CDATA[#{javascript:
    var externalContext = facesContext.getExternalContext();
    var response = externalContext.getResponse();
    var request = externalContext.getRequest();

    if( !param.containsKey("p") ){
      response.setStatus(400);
      response.setContentType("application/json");
      var writer = facesContext.getResponseWriter();
      writer.write("{ \"code\":400, \"text\":\"Bad Parameter\",
                \"message\":\"Missing method call parameter
    'p'\" }");
      return;
    }

    var s = new org.openntf.api.snippet();
    s.service( request, response);
}]]>
</xp:this.afterRenderResponse>
</xp:view>
```

Demo Time – Custom App!

- Inner Ring Bike shop.
  - Using Notes for inventory and sales
  - Wants to add a web interface, and also add some pizzazz with graphs and photos.

For more…

- Presentation download
  - http://www.innerringsolutions.com/Connect2013/bp204.pdf
- Demo database download
  - http://www.innerringsolutions.com/Connect2013/bp204_demo.zip

Thanks!!

Craig Schumann

Inner Ring Solutions, LLC
http://blog.innerringsolutions.com

@ZOPEFF

www.linkedin.com/in/craigschumann/

The End…

Legal disclaimer