Creating a Dynamic REST Service

Session 429 – OE REST, Part 2 of 2

Dustin Grau – dgrau@progress.com

Principal Solutions Consultant

PROGRESS BravePoint

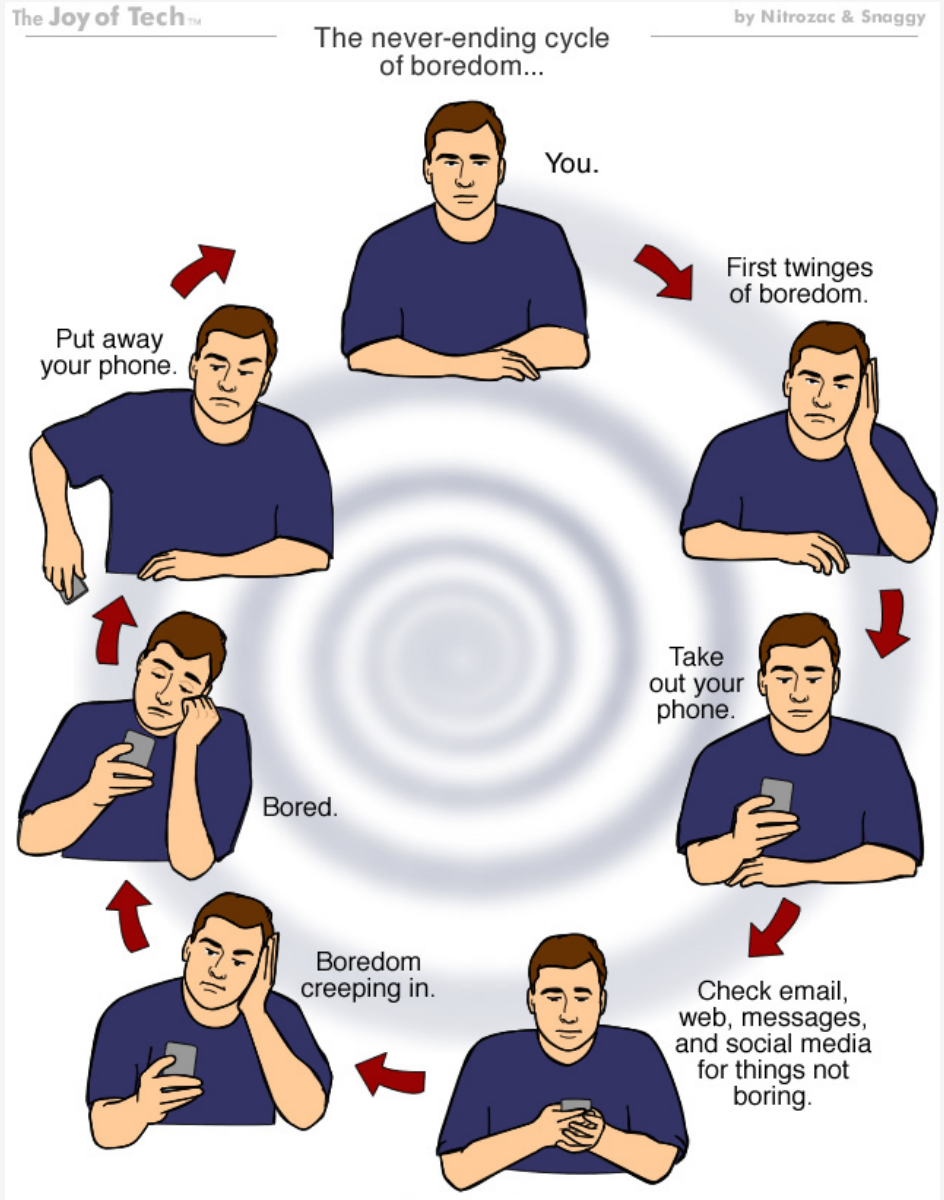
PUG CHALLENGE EXCHANGE AMERICAS

Introductions

PUG CHALLENGE EXCHANGE AMERICAS

Establishing Ground Rules

- Only covering this topic in terms of the “Classic AppServer” approach
  - No discussion of the Pacific AppServer here (Roy Ellis covered this well)
  - We’ll be referencing the “Tomcat in the box” bundled with PDSOE
- There are many types of REST implementations
  - We’ll examine what OpenEdge expects
  - Not covering versioning patterns
  - You can still roll your own
- Definitions of “dynamic”
  - Dynamically created interface with rigid class structure
  - Static interface with a more dynamic class structure
- Need to describe and access data before we can present it

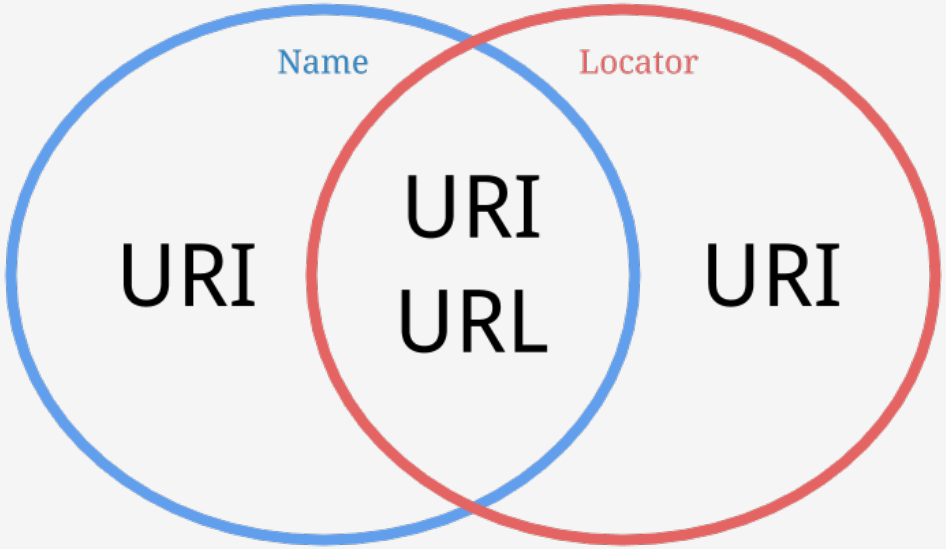

The Joy of Tech™
by Nitrozac & Snaggy
The never-ending cycle of boredom...
You.
First twinges of boredom.
Take out your phone.
Check email, web, messages, and social media for things not boring.
Boredom creeping in.
Bored.
Put away your phone.

The Data Layer

PUG CHALLENGE EXCHANGE AMERICAS

Describing Data

- http://<server>:<port>/<webapp>/rest/<service>/<resource>[/<sub-resource>]
- Defining a service within our WebApp: /rest/si/
- Plural vs. Singular resource names
- Avoiding verbs in the URI
- Providing a simple resource: /rest/si/customer
- Data returned as an object (JSON)
  - Temp-table = array of objects
  - Dataset = object with properties (temp-tables)



Accessing Data

- http://<server>:<port>/<webapp>/rest/<service>/<resource>[/<sub-resource>]
- GET /rest/si/customer
- GET /rest/si/customer/1
- GET /rest/si/customer?custnum=1
- GET /rest/si/customer?name=Hoops
- GET /rest/si/customer?filter={“name”: “Hoops”}
- GET /rest/si/customer?filter={“ablFilter”: “Name BEGINS ‘Hoops’”}
- GET /rest/si/customer?filter={“field”: “Name”, “operator”: “begins”, “value”: “Hoops”}
- GET /rest/si/customer?filter={“criteria”: [{“field”: “Name”, “operator”: “begins”, “value”: “Hoops”}]}
- POST, PUT, DELETE require JSON as request body

Preparing for Access

- Annotate ABL resources
  - Manually: Define Service Interface
  - Automatic: Use a Mobile project type
- Map REST operations to ABL operations
  - Manually: Add resource(s) in defined service, associate methods with verbs, add parameters
  - Automatic: Provide method annotations within exposed class file
- Test service using a REST client
  - Publish the REST application to Tomcat
  - Use any compatible tool (RESTclient, Postman, etc.)

PDSOE Annotations

- Drives creation of static catalog within a Mobile project
  - Service
    - Resource
      - Schema
      - Operations
        - Params
- Immediately precede the item they describe
- Can alter the object described
  - Change exposed path in REST service
  - For INVOKE's, change verb used (default: PUT)

Class Header Annotations

@program FILE(name="CustomerBE.cls", module="AppServer").

@openapi.openedge.export FILE(type="REST", executionMode="singleton", useReturnValue="false", writeDataSetBeforeImage="false").

@progress.service.resource FILE(name="CustomerBE", URI="/customer", schemaName="dsCustomer",

schemaFile="MyProject/AppServer/Sports/Business/customerbe.i").

class Sports.Business.CustomerBE inherits BusinessEntity:

{Sports/Business/customerbe.i}

…

Per-method Annotations

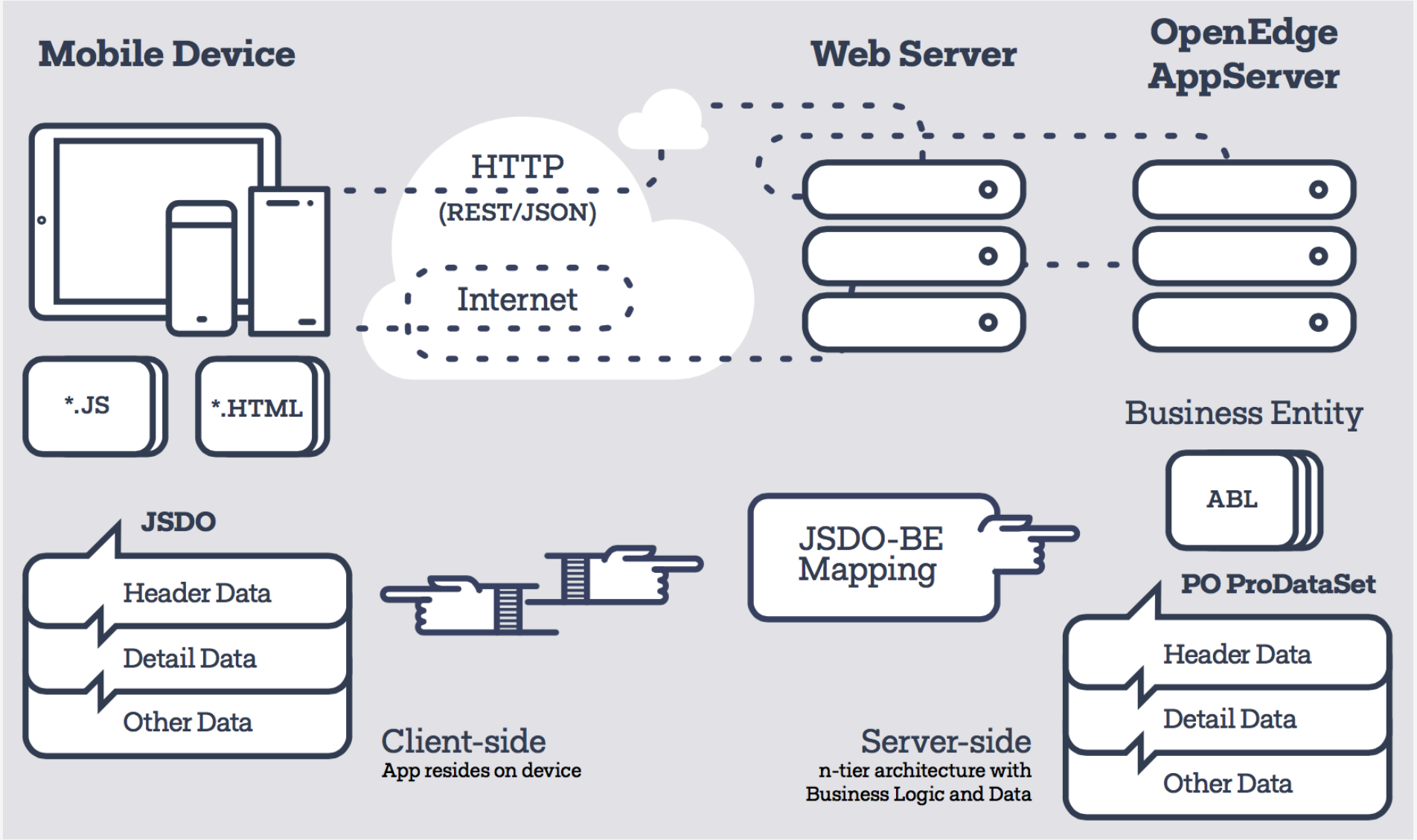
```
@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
@progress.service.resourceMapping(type="REST", operation="invoke", URI="/count?filter=~{filter~}", alias="",
mediaType="application/json").
method public void count ( input filter as character, output numRecs as integer ):
…

@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="true").
@progress.service.resourceMapping(type="REST", operation="read", URI="?filter=~{filter~}", alias="",
mediaType="application/json").
@openapi.openedge.method.property (name="mappingType", value="JFP").
@openapi.openedge.method.property (name="capabilities", value="ablFilter,top,skip,id,orderBy").
method public void ReadCustomerBE ( input filter as character, output dataset dsCustomer ):
…
```

Abstracting Access via the JSDO

- JSDO: JavaScript Data Object
- Catalog-driven (descriptions in JSON format)
- Automatically generated via PDSOE annotations
  - Only in a Mobile project only, currently
  - Must create manually in a REST Service project
- Can manually map input/output parameters
  - Works on procedures or classes
  - Must adjust/re-map if parameters change
- Similar purpose to libraries like BreezeJS

JSDO Overview



Static Annotation Considerations

- In my own experiences, at this moment…YMMV
- Where it works perfectly:
  - Single environment/developer
  - Few changes to objects after mapping
  - Small or limited number of resources
  - Using a mobile project with annotations
- Where it gets tricky:
  - Collaborative environments
  - Frequent changes to mapped objects
  - Large number of exposed objects

Authentication via JSDO

- Only certain security models are supported at present (anonymous, user, form)
- We will use the Form-OERealm security model
  - POST's j_username and j_password to j_spring_security_check
- Requires creation of a JSDOSession instance (w/ or w/o credentials)
- Must add a catalog and create JSDO instance against a resource
- What about anonymous/public resources?

JSDO Resource Operations

- CRUD
  - fill(), add(), assign(), remove()
  - saveChanges()
- Submit(11.5)
  - saveChanges(true)
- find(), findById()
- getData()
- subscribe(), unsubscribe()
- Invoke methods
  - REST method == JSDO method
  - Must pass an object as parameter

Your Own Dynamic Implementation

- How to structure of data packets and URI's
  - Use same format as JSDO expects
  - Filters, datasets, errors, etc.
- Use of classes or procedures?
  - Could read internal-procedure signatures to get input/output values
  - No class reflection yet, but you could implement a known interface!
- Need to generate a JSDO-compatible catalog
  - Uses custom annotation methods to create an internal registry
  - Read registry and produce a proper catalog in JSON format
- Execute REST request, apply parameters as needed
- All will be handled by the Spark toolkit (release TBD)

Quick Notes on Security

- Secure the connection between Spring framework and authenticating AppServer
  - Use a pre-generated CP token, set in realmTokenFile property in appSecurity XML
  - Ensures the only the request from an authorized endpoint will be requesting data
- There should be a CP token available on every AppServer request
  - Even anonymous users will get a token: SessionID will be 0, ROLE_ANONYMOUS
  - Authenticated users will have an actual SessionID available and list of roles
- Before establishing the request's CP token, set to a no-access (dummy) token
  - Can be pre-generated and stored as a file on disk
  - Prevents any DB access from a stale/previous request
  - Remember to also do this after your request completes!

The UI Layer

PUG CHALLENGE EXCHANGE AMERICAS

Code Sample: JSDO Creation

```
var serviceURI = "http://oemobiledemo.progress.com/MobilityDemoService”;
var catalogURI = serviceURI + "/static/mobile/MobilityDemoService.json";
var custJSDO= null; // create instance later, uses a dsCustomer dataset.
var jsdosession = new progress.data.JSDOSession({serviceURI: serviceURI,
                                                 catalogURIs: catalogURI});
var promise = jsdosession.login("", "");
promise.done(function(jsdosession, result, info) {
    var catReq = jsdosession.addCatalog(catalogURI);
    catReq.done(function(jsdosession, result, details) {
        custJSDO = new progress.data.JSDO({name: ‘CustomerBE'}); // name of REST resource
        custJSDO.fill().done(onAfterFillCustomers); // callback method to run when done
    });
)};
```

Code Sample: JSDO Creation

```
function onAfterFillCustomers(jsdo, success, request) {
    jsdo.dsCustomer.foreach(function(customer) {
        // write out some of the customer data to the page
        document.write(customer.data.CustNum + ' ' + customer.data.Name + '<br>');
    }
});
```

Code Sample: Kendo Datasource

```
var customerDS = new kendo.data.DataSource({
    transport: {
        jsdo: 'CustomerBE', // matches name of resource in catalog
        tableRef: 'ttCustomer' // required if dataset contains more than 1 table
    },
    error: function (e) {
        console.log('Error: ', e);
    }
});
$("#grid").kendoGrid({
    dataSource: customerDS,
    …
});
```

Advanced JSDO Features

- Invoke methods appear as a method on a JSDO instance
  - To call a method “count” use myJSDO.count({ });
  - Currently requires an object to be passed, even if empty
- New mappingType and capabilities features
  - Built-in type “JFP”: JSON Filter Pattern
  - Converts Kendo criteria to properly-grouped ABL “where” phrase (on client side)
  - Capabilities describes just that (can skip rows, return X records, sort columns, etc.)
  - Provide transport property “countFnName” with name of “count” method
- Adding a new plugin (mapper) to JSDO
  - progress.data.PluginManager.addPlugin(“PluginName”, { … });
  - Currently only supports a requestMapping() method
  - We have requested a responseMapping() method

Real-World Questions

- Logical table vs. database table
  - When fields don’t match (custom schema)
  - Transforming JSON-unsafe fields (dashes)
  - Representing non-normalized structures (extents)
- Managing multi-table datasets
  - Read vs. write operations
  - Sending/handling errors
- Other error-handling needs
  - All-or-nothing or commit only valid records?
  - How to manager errors with end-user?

Thank You!

- “Inside the JSDO: Mobile and Web”
  - Edsel Garcia, Progress Exchange 2014
- View demo pages (change #’s at end)
  - http://oemobiledemo.progress.com/jsdo/example014.html
- Whitepaper: “Using the JSDO with KendoUI”
  - http://www.telerik.com/campaigns/kendo-ui/using-the-jsdo-with-kendo-ui
- Download examples of JSDO v4.0 with KendoUI
  - https://community.progress.com/community_groups/openedge_development/m/documents/2020.aspx
- JSDO available on GitHub
  - https://github.com/CloudDataObject/JSDO
- All slides will be available after the conference!

PUG
CHALLENGE
EXCHANGE
AMERICAS