Lightweight REST Approaches to Data Access at Xtime

Adam Galper
CTO, Xtime

August 2015

Goals

- Problem Statement
- Data Access Approaches
- REST Advantages
- Dynamic Data Service (DDS)
- Experience with DDS Implementation at Xtime
- Comparison to ORDS 3.0

Problem Statement

- We may build long-lived systems in a RDBMS, but:
  - application frameworks come and go
  - development languages come and go
  - drivers vary across languages and frameworks
- As we build new applications or rewrite old ones, there is a significant cost in porting a data access layer
- A data access layer should be:
  - lightweight for the developer, requiring only basic knowledge of the data model
  - lightweight for the app, requiring no unnecessary transformations of data
  - lightweight for the database, never allowing untuned requests from the app
  - mostly universal, applicable to at least 80% of data access use cases
  - long-lived, relying on protocols that are independent of languages and app frameworks

Data Access Approaches

xtime

- At Xtime, over a 13 year period, we have built data access layers for executing queries and stored behaviors in Oracle, using:
  - Our own ORM, written in Java and XSLT
  - Hibernate
  - Spring JDBC
  - Spring Batch
  - Our own Python access framework
- All were heavyweight
  - Generating code for classes, interfaces, deployment descriptors
  - Requiring an app-level query language (OQL)
  - Consuming CPU when mapping result sets to transfer objects to access objects
- A few years ago, we hypothesized that a thin REST layer over tables, views, and stored behaviors would address >80% of our use cases

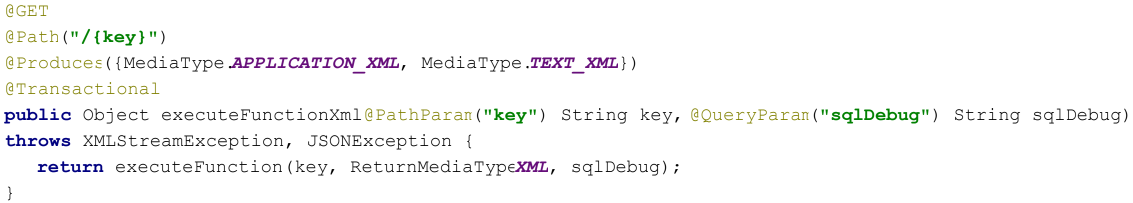
REST Advantages

- REST: REpresentational State Transfer
  - Introduced in 2000 as an architecture for scalable web services
- Uses HTTP protocol and verbs (GET, PUT, POST, DELETE), resulting in simple, consistent, stateless interfaces
- Every development language supports HTTP, so clients do not require a database driver (e.g. OCI, JDBC, or cx_Oracle)
  - A RESTful server exposing interfaces can manage connections/pools
- REST is message format agnostic
  - JSON is the most common format input/output
  - But can also return non-JSON messages (XML, CSV, etc…)
- REST is the most widely used interface for accessing both internal and external services in the enterprise

Dynamic Data Service (DDS)

- We decided to build our own REST-based data access layer, because Oracle REST Data Services (ORDS) seemed too heavyweight at the time
  - Dependent on APEX
  - Difficult to configure
  - Missing features

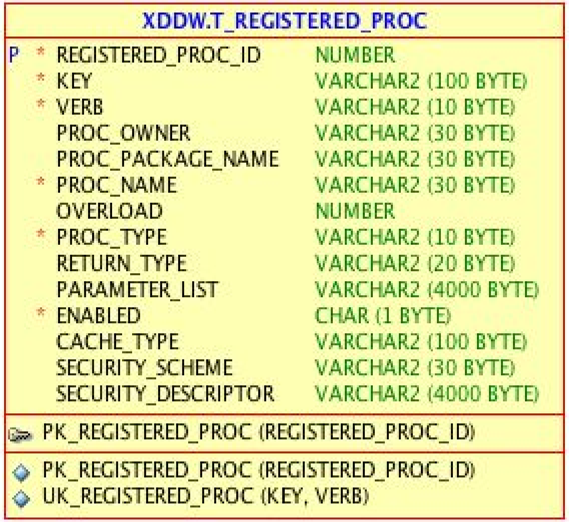
- DDS is a Jersey (JAX-RS) Java program deployed as a WAR in Tomcat clusters
  - DDS reads its endpoint metadata from a table registry and a procedure registry
  - The registries define the access path, security scheme, and various default behaviors for the endpoints
  - DDS listens for incoming JAX-RS requests that match the endpoints, executing the access or update, and converting return data (cursors) into the requested response format



```
@GET
@Path("/{key}")
@Produces({MediaType.APPLICATION_XML, MediaType.TEXT_XML})
@Transactional
public Object executeFunctionXml@PathParam("key") String key, @QueryParam("sqlDebug") String sqlDebug)
throws XMLStreamException, JSONException {
   return executeFunction(key, ReturnMediaTypeXML, sqlDebug);
}
```

  - DDS provides built-in features for managing the registries, caching results, paging results, and applying the appropriate security scheme (e.g. Spring Security)
  - The core of DDS was ready for testing after a two week development effort
  - DDS is not intended for use outside of Xtime

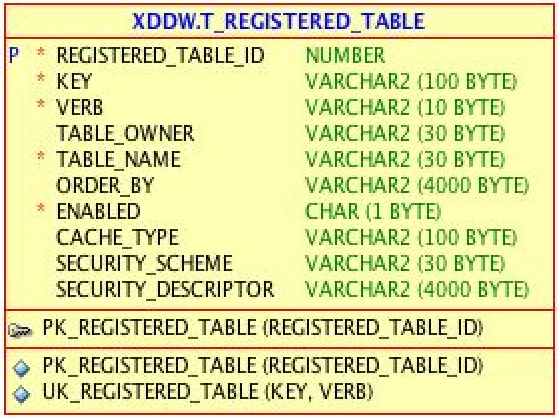
Configuration with DDS



XDDW.T_REGISTERED_PROC			
P	*	REGISTERED_PROC_ID	NUMBER
	*	KEY	VARCHAR2 (100 BYTE)
	*	VERB	VARCHAR2 (10 BYTE)
		PROC_OWNER	VARCHAR2 (30 BYTE)
		PROC_PACKAGE_NAME	VARCHAR2 (30 BYTE)
	*	PROC_NAME	VARCHAR2 (30 BYTE)
		OVERLOAD	NUMBER
	*	PROC_TYPE	VARCHAR2 (10 BYTE)
		RETURN_TYPE	VARCHAR2 (20 BYTE)
		PARAMETER_LIST	VARCHAR2 (4000 BYTE)
	*	ENABLED	CHAR (1 BYTE)
		CACHE_TYPE	VARCHAR2 (100 BYTE)
		SECURITY_SCHEME	VARCHAR2 (30 BYTE)
		SECURITY_DESCRIPTOR	VARCHAR2 (4000 BYTE)

PK_REGISTERED_PROC (REGISTERED_PROC_ID)

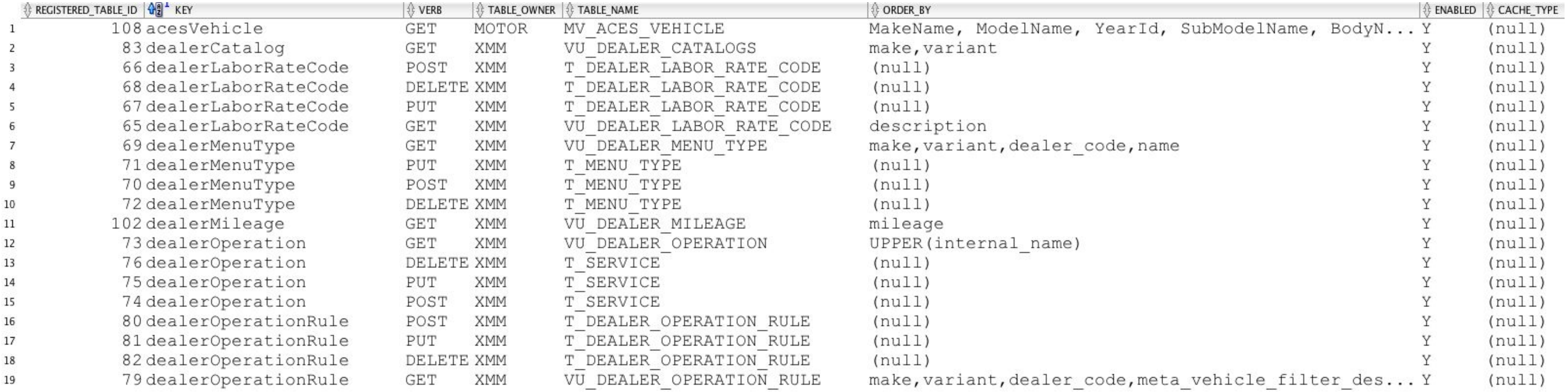
◆ PK_REGISTERED_PROC (REGISTERED_PROC_ID)
◆ UK_REGISTERED_PROC (KEY, VERB)



XDDW.T_REGISTERED_TABLE			
P	*	REGISTERED_TABLE_ID	NUMBER
	*	KEY	VARCHAR2 (100 BYTE)
	*	VERB	VARCHAR2 (10 BYTE)
		TABLE_OWNER	VARCHAR2 (30 BYTE)
	*	TABLE_NAME	VARCHAR2 (30 BYTE)
		ORDER_BY	VARCHAR2 (4000 BYTE)
	*	ENABLED	CHAR (1 BYTE)
		CACHE_TYPE	VARCHAR2 (100 BYTE)
		SECURITY_SCHEME	VARCHAR2 (30 BYTE)
		SECURITY_DESCRIPTOR	VARCHAR2 (4000 BYTE)

PK_REGISTERED_TABLE (REGISTERED_TABLE_ID)

◆ PK_REGISTERED_TABLE (REGISTERED_TABLE_ID)
◆ UK_REGISTERED_TABLE (KEY, VERB)



REGISTERED_TABLE_ID	KEY	VERB	TABLE_OWNER	TABLE_NAME	ORDER_BY	ENABLED	CACHE_TYPE
108	acesVehicle	GET	MOTOR	MV_ACES_VEHICLE	MakeName, ModelName, YearId, SubModelName, BodyN...	Y	(null)
83	dealerCatalog	GET	XMM	VU_DEALER_CATALOGS	make,variant	Y	(null)
66	dealerLaborRateCode	POST	XMM	T_DEALER_LABOR_RATE_CODE	(null)	Y	(null)
68	dealerLaborRateCode	DELETE	XMM	T_DEALER_LABOR_RATE_CODE	(null)	Y	(null)
67	dealerLaborRateCode	PUT	XMM	T_DEALER_LABOR_RATE_CODE	(null)	Y	(null)
65	dealerLaborRateCode	GET	XMM	VU_DEALER_LABOR_RATE_CODE	description	Y	(null)
69	dealerMenuType	GET	XMM	VU_DEALER_MENU_TYPE	make,variant,dealer_code,name	Y	(null)
71	dealerMenuType	PUT	XMM	T_MENU_TYPE	(null)	Y	(null)
70	dealerMenuType	POST	XMM	T_MENU_TYPE	(null)	Y	(null)
72	dealerMenuType	DELETE	XMM	T_MENU_TYPE	(null)	Y	(null)
102	dealerMileage	GET	XMM	VU_DEALER_MILEAGE	mileage	Y	(null)
73	dealerOperation	GET	XMM	VU_DEALER_OPERATION	UPPER(internal_name)	Y	(null)
76	dealerOperation	DELETE	XMM	T_SERVICE	(null)	Y	(null)
75	dealerOperation	PUT	XMM	T_SERVICE	(null)	Y	(null)
74	dealerOperation	POST	XMM	T_SERVICE	(null)	Y	(null)
80	dealerOperationRule	POST	XMM	T_DEALER_OPERATION_RULE	(null)	Y	(null)
81	dealerOperationRule	PUT	XMM	T_DEALER_OPERATION_RULE	(null)	Y	(null)
82	dealerOperationRule	DELETE	XMM	T_DEALER_OPERATION_RULE	(null)	Y	(null)
79	dealerOperationRule	GET	XMM	VU_DEALER_OPERATION_RULE	make,variant,dealer_code,meta_vehicle_filter_des...	Y	(null)

DDS Table Operations

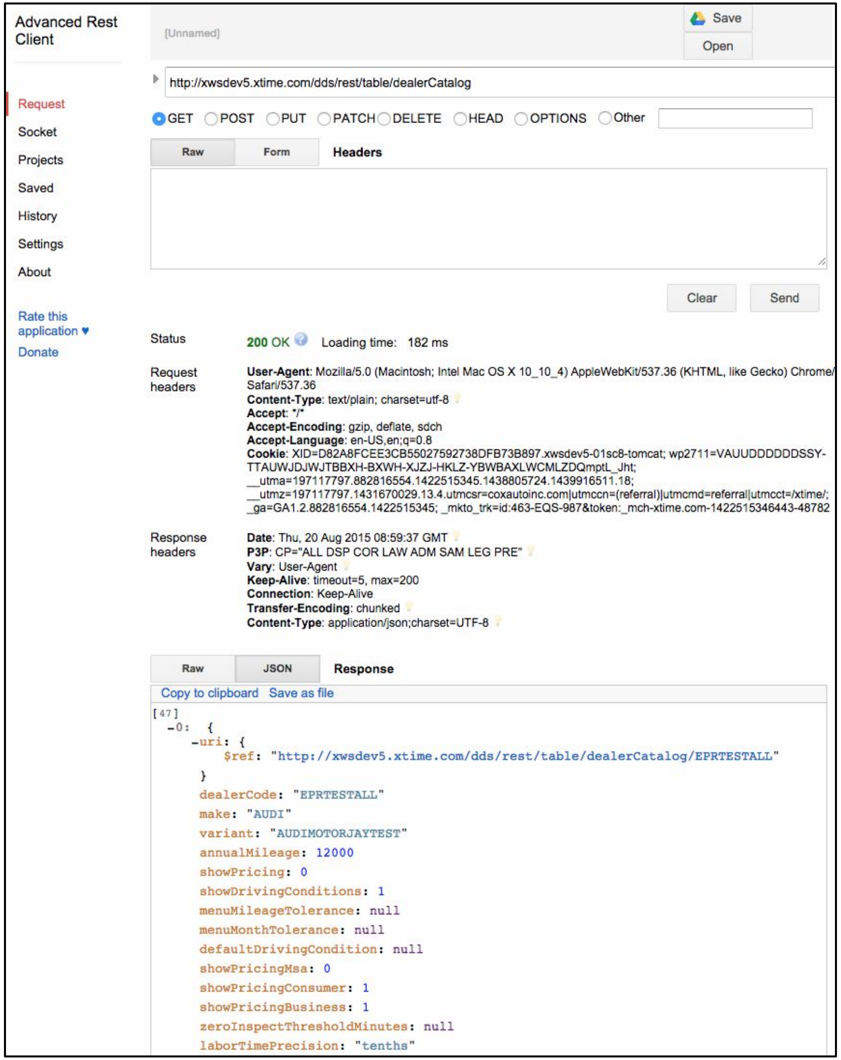
CRUD via standard REST Operations

GET /rest/table/[key]	Select all rows
GET /rest/table/[key]/[id]	Select a single row
GET /rest/table/[key]?<query_params>	Select rows using (optional) query params
POST /rest/table/[key]	Insert a row, provided in JSON
PUT /rest/table/[key]/[id]	Update a row, attribute subset provided in JSON
DELETE /rest/table/[key]/[id]	Delete a single row

Guidelines:

1. Tables must have a single column primary key which must be the first column in the table.
2. Transactions cannot span DDS service calls.
3. If a sequence of steps must commit or rollback as a unit, use a single procedural service instead of a sequence of CRUD services.

DDS Table Examples



http://xwsdev5.xtime.com/dds/rest/table/dealerCatalog

● GET ○ POST ○ PUT ○ PATCH ○ DELETE ○ HEAD ○ OPTIONS ○ Other

Status: 200 OK Loading time: 182 ms

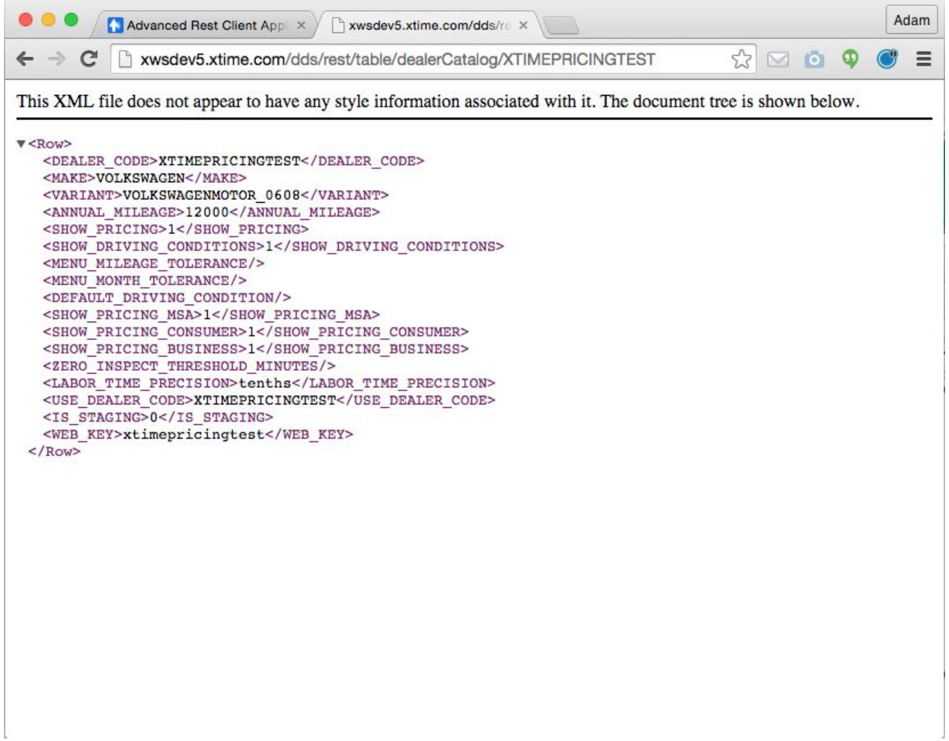
Request headers

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
Safari/537.36
Content-Type: text/plain; charset=utf-8
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: XID=D82A8FCEE3CB55027592738DFB73B897.xwsdev5-01sc8-tomcat; wp2711=VAUUDDDDDDSSY-TTAUWJDJWJTBBXH-BXWH-XJZJ-HKLZ-YBWBAXLWCMLZDQmptL_Jht; __utma=197117797.882816554.1422515345.1438805724.1439916511.18; __utmz=197117797.1431670029.13.4.utmcsr=coxautoinc.com|utmccn=(referral)|utmcmd=referral|utmcct=/xtime/; _ga=GA1.2.882816554.1422515345; _mkto_trk=id:463-EQS-987&token:_mch-xtime.com-1422515346443-48782

Response headers

Date: Thu, 20 Aug 2015 08:59:37 GMT
P3P: CP="ALL DSP COR LAW ADM SAM LEG PRE"
Vary: User-Agent
Keep-Alive: timeout=5, max=200
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/json;charset=UTF-8

```
[47]
 -0:  {
    -uri: {
        $ref: "http://xwsdev5.xtime.com/dds/rest/table/dealerCatalog/EPRTESTALL"
     }
     dealerCode: "EPRTESTALL"
     make: "AUDI"
     variant: "AUDIMOTORJAYTEST"
     annualMileage: 12000
     showPricing: 0
     showDrivingConditions: 1
     menuMileageTolerance: null
     menuMonthTolerance: null
     defaultDrivingCondition: null
     showPricingMsa: 0
     showPricingConsumer: 1
     showPricingBusiness: 1
     zeroInspectThresholdMinutes: null
     laborTimePrecision: "tenths"
```



xwsdev5.xtime.com/dds/rest/table/dealerCatalog/XTIMEPRICINGTEST

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Row>
   <DEALER_CODE>XTIMEPRICINGTEST</DEALER_CODE>
   <MAKE>VOLKSWAGEN</MAKE>
   <VARIANT>VOLKSWAGENMOTOR_0608</VARIANT>
   <ANNUAL_MILEAGE>12000</ANNUAL_MILEAGE>
   <SHOW_PRICING>1</SHOW_PRICING>
   <SHOW_DRIVING_CONDITIONS>1</SHOW_DRIVING_CONDITIONS>
   <MENU_MILEAGE_TOLERANCE/>
   <MENU_MONTH_TOLERANCE/>
   <DEFAULT_DRIVING_CONDITION/>
   <SHOW_PRICING_MSA>1</SHOW_PRICING_MSA>
   <SHOW_PRICING_CONSUMER>1</SHOW_PRICING_CONSUMER>
   <SHOW_PRICING_BUSINESS>1</SHOW_PRICING_BUSINESS>
   <ZERO_INSPECT_THRESHOLD_MINUTES/>
   <LABOR_TIME_PRECISION>tenths</LABOR_TIME_PRECISION>
   <USE_DEALER_CODE>XTIMEPRICINGTEST</USE_DEALER_CODE>
   <IS_STAGING>0</IS_STAGING>
   <WEB_KEY>xtimepricingtest</WEB_KEY>
 </Row>
```

DDS Procedure Operations

GET /rest/proc/[key]?<query-params> POST /rest/proc/[key]?<query-params>	Execute function with params mapped to procedure arguments

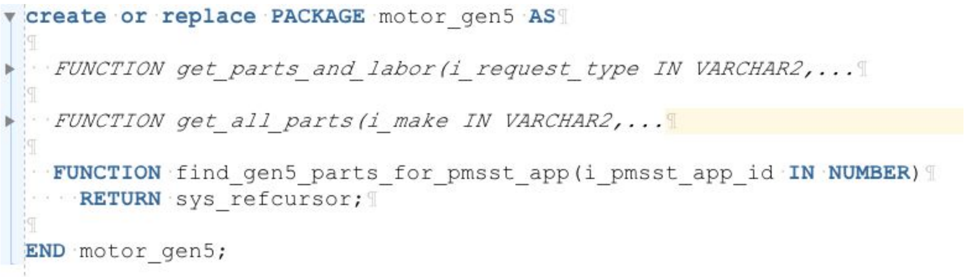
Guidelines:

1. Functions generally return a REF CURSOR, which is converted into JSON, XML, or CSV:

```
> curl 'http://<hostname>/rest/proc/dealerinfo?dealerCode=02037&make=SCION'
[
        {
        "annualMileage": 12000,
        "authContextId": 4013530356,
        "dealerCode": "02037",
        "dealerName": "ALEXANDER TOYOTA YUMA",
...
```

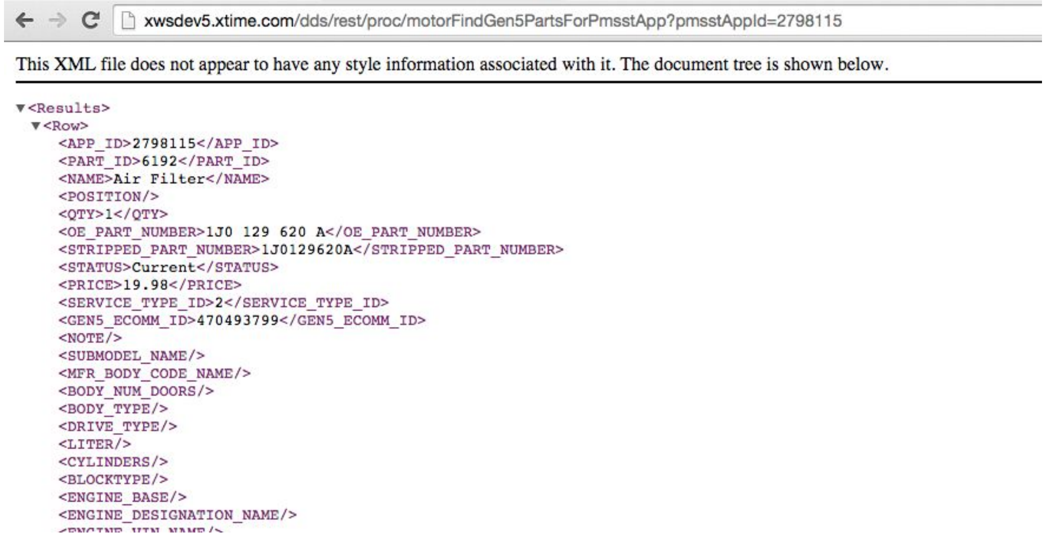
2. Parameters do not need to be in order; a default naming convention (e.g. “dealerCode” => “I_DEALER_CODE”) is used to map to the actual procedure arguments in the right order.
3. If desired, a custom set of parameter names can be specified (in procedure defined order) using the registry’s parameter_list column. This allows a call of this form: `<hostname>/rest/proc/dealerinfoCustom?d=02037&m=SCIO N`
4. Results can be automatically cached (e.g. by TTL) by designating the cache type for the procedure in the registry.
5. In addition to returning JSON and XML, procedures (regardless of return type) can be requested to return data in csv format, using `-H 'Accept: text/csv'`

DDS Procedure Example



```
create or replace PACKAGE motor_gen5 AS

  FUNCTION get_parts_and_labor(i_request_type IN VARCHAR2,...

  FUNCTION get_all_parts(i_make IN VARCHAR2,...

  FUNCTION find_gen5_parts_for_pmsst_app(i_pmsst_app_id IN NUMBER)
     RETURN sys_refcursor;

END motor_gen5;
```

Functions above returns a sys_refcursor for a multi-table join, result displayed below.



xwsdev5.xtime.com/dds/rest/proc/motorFindGen5PartsForPmsstApp?pmsstAppId=2798115

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Results>
  ▼<Row>
     <APP_ID>2798115</APP_ID>
     <PART_ID>6192</PART_ID>
     <NAME>Air Filter</NAME>
     <POSITION/>
     <QTY>1</QTY>
     <OE_PART_NUMBER>1J0 129 620 A</OE_PART_NUMBER>
     <STRIPPED_PART_NUMBER>1J0129620A</STRIPPED_PART_NUMBER>
     <STATUS>Current</STATUS>
     <PRICE>19.98</PRICE>
     <SERVICE_TYPE_ID>2</SERVICE_TYPE_ID>
     <GEN5_ECOMM_ID>470493799</GEN5_ECOMM_ID>
     <NOTE/>
     <SUBMODEL_NAME/>
     <MFR_BODY_CODE_NAME/>
     <BODY_NUM_DOORS/>
     <BODY_TYPE/>
     <DRIVE_TYPE/>
     <LITER/>
     <CYLINDERS/>
     <BLOCKTYPE/>
     <ENGINE_BASE/>
     <ENGINE_DESIGNATION_NAME/>
     <ENGINE_VIN_NAME/>
```

Experience with DDS

In two years of use:

1. Two old applications have been completely retrofitted with DDS
2. Two new applications have been built with DDS as the data access layer.
3. Two existing applications have been extended with features supported by DDS.
4. Over 600 endpoints registered, including 80+ tables
5. Table and Proc DDS invocations handle >90% of the new applications' database requests
6. Xtime DDS clients are written in Javascript, Flex, Python, and Java using standard HTTP invocation features of these languages.
7. DB connection pool management is simplified, concentrated in a DDS cluster
8. Clients contain no traces of SQL or references to named DB objects
9. Apps using DDS are built faster, require less code, shorter test cycles
10. DDS itself changes infrequently

Comparison to ORDS 3.0

ORDS 3.0 released May 2015, includes:

1. Features for “auto-enablement” of tables ⇒ equivalent to our Table Operations
2. Easy to write JSON filters allow query predicates and sorts to be specified in a query-by-example format.
3. For more complex operations, REST calls are mapped to SQL and PL/SQL routines you can write which return data in JSON and other formats.

In addition, ORDS is now independent of APEX and is easier to install/configure.

Finally, ORDS 3.0 has support for:

1. OAuth 2.0 security standard
2. Oracle NoSQL REST Access
3. Oracle 12c JSON Document Store