Oracle Endeca for Mobile
Getting Started Guide – Mobile API

Copyright and Disclaimer

Getting Started Guide – Mobile API

By default, the Mobile API is configured to work with the Camera Store dataset. This guide goes through the files that need to be changed to configure the Mobile API for another Dgraph. The shopping cart and checkout are powered by default from a mock API, and this guide also shows you how to change this to work with your commerce system backend.

XML Configuration Files

/WEB-INF/spring/app-config.xml

1. Change the MDEX Engine host and port:

```
<bean id="mdexConfig" class="com.endeca.b2creference.config.MdexConfig">
        <property name="host" value="localhost"/>
        <property name="port" value="13000"/>
</bean>
```

2. Update the SelectedRecordsQueryProcessor:

   Replace “common.id” in the recordSpecName property to the MDEX Engine property that is used for the record spec:

```
<bean class="com.endeca.mobile.services.query.impl.SelectedRecordsQueryProcessor">
        <property name="recordSpecName" value="common.id" />
</bean>
```

3. Update the SearchInterfaceQueryProcessor:

   If you have a localized search interface, that uses the 2 letter language code (i.e. All_en, All_de, All_es, etc), you can use this query processor to select the appropriate search interface automatically. If you have only one search interface, change appendLanguageCode to false, and update searchInterface from “All_” to the search interface for your MDEX Engine:

```
<bean class="com.endeca.mobile.services.query.impl.SearchInterfaceQueryProcessor">
        <property name="appendLanguageCode" value="true" />
        <property name="defaultLanguageCode" value="en" />
        <property name="searchInterface" value="All_" />
</bean>
```

4. Update FieldListConfigurationPreProcessor:

   This query processor provides an important performance optimization for a Dgraph with a large number of properties on each record. If you have wide records in your Dgraph, you should specify only those properties used in the results list.

   You can also comment out the entire <bean> element, and do this configuration later.

```
<bean class="com.endeca.mobile.services.query.impl.FieldListConfigurationPreProcessor">
        <property name="fieldNames">
                <list>
                        <value>product.id</value>
                        <value>product.name</value>
                        <value>product.price</value>
```

```
                <value>product.brand.name</value>
                <value>product.img_url_large</value>
                <value>product.review.avg_rating</value>
                <value>product.review.count</value>
                <value>product.img_url_thumbnail</value>
                <value>product.inventory.count</value>
                <value>product.short_desc</value>
                <value>product.long_desc</value>
                <value>product.brand.img_url</value>
                <value>review.title</value>
                <value>review.rating</value>
                <value>review.timestamp</value>
                <value>review.author.name</value>
                <value>review.full_text</value>
            </list>
        </property>
</bean>
```

5. Comment out the Camera Store specific query processors:

   ```
   <bean class="com.camerastore.mobile.services.query.impl.RecordFilterQueryProcessor"/>
   <bean
   class="com.camerastore.mobile.services.query.impl.RefinementExclusionResultsProcessor"/>
   ```

   You may want to add your own query processors to apply record filters, or post process the results. These processors provide a good reference example. The URLParameterRecordFilterProcessor, for example, is used to apply different record filters based on a request parameter, so that the same controller can be used to provide a web service API for different types of records, such as product records, review records, and store records.

6. Comment out the Camera Store specific detail query processors

   ```
   <bean class="com.camerastore.mobile.services.query.impl.RecordDetailResultsProcessor"/>
   ```

7. Change the brandDimensionId property to the dimension id of the Brand dimension in your Dgraph. If you don't need a Browse by Brand page, don't worry about this configuration.

   You can also specify a list of navStates (space delimited dimension value ids). The refinement results from each navState will be merged to produce the list of brands. This can be used to get around precedence rules, and if you don't need to worry about precedence rules, the N=0 nav state can be used.

   ```
   <bean id="brandConfig" class="com.endeca.mobile.config.BrandConfig">
       <property name="brandDimensionId" value="20001"></property>
       <property name="navStates">
           <list>
               <value>0</value>
           </list>
       </property>
   </bean>
   ```

8. For native iPhone applications in particular, it may be useful to remove HTML tags and entities from properties. This can be done by adding configuration to the metadataService. The abstraction layer handles removing HTML, so this configuration does not affect the mobile web

version of these properties.

```
<bean id="metadataService"
class="com.endeca.b2creference.services.impl.DefaultMetadataService">
        …
        <property name="abstractionMetadataMap">
                <map>
                        <entry key="product.short_desc">
                                <bean
class="com.endeca.b2creference.services.AbstractionMetadata">
                                        <property name="stripHtml" value="true"/>
                                        <property name="preserveNewlines" value="false"/>
                                </bean>
                        </entry>
                        <entry key="product.long_desc">
                                <bean
class="com.endeca.b2creference.services.AbstractionMetadata">
                                        <property name="stripHtml" value="true"/>
                                        <property name="preserveNewlines" value="true"/>
                                </bean>
                        </entry>
                </map>
        </property>
</bean>
```

9. The Mobile API provides a place to configure view specific properties from app-config.xml called "viewConfig". Any property in the requestMap will be available as a HttpServletRequest request attribute. This allows JSP renderers or JSON abstraction APIs to leverage this configuration without having to create specific new configuration classes.

```
<!-- Autowired to the BaseController values in the requestMap get added directly to the spring
model so they are available in the view jsps  -->
<bean id="viewConfig" class="com.endeca.b2creference.config.ViewConfig">
        <property name="requestMap">
                <map>
                        <entry key="STATIC_RESOURCE_PATH"
value="/components/mobile_shared" />
                </map>
        </property>
</bean>
```

/WEB-INF/web.xml

If you renamed the mobile webapp to something other than 'mobile', you will need to update the following:

```
<context-param>
        <param-name>webAppRootKey</param-name>
        <param-value>mobile</param-value>
</context-param>
```

/WEB-INF/spring/endeca-uicl-config.xml

If you renamed the mobile webapp to something other than ‘mobile’, you will need to update the following:

```
…
        <entry key="resultsBasePath">
                <value>/mobile/search</value>
        </entry>
…
        <entry key="storesBasePath">
                <value>/mobile/stores</value>
        </entry>
…
        <entry key="refiewsBasePath">
                <value>/mobile/reviews</value>
        </entry>
```

/WEB-INF/messages/attribute_display_names.properties

The attribute_display_name files are used for localization of property names. To add display names to a property, you can edit this file. Below is an example, configured for the Camera Store data.

```
displaynames.product.brand.name = Brand
displaynames.product.review.avg_rating_range = Average Rating
displaynames.product.price_range = Price
displaynames.camera.megapixel_range = Megapixels
displaynames.camera.optical_zoom_range = Optical Zoom
displaynames.product.features = Other Features
```

Implementing Shopping Cart and Checkout

You need to do 2 things to implement the Oracle Endeca for Mobile cart and checkout functionality.

1. Create an instance of the CartAPI (com.endeca.mobile.services.cart.CartAPI) interface.

   These methods all take in the HttpSession object so that they can store transaction information between requests. If the implementation of the CartAPI needs to query an external service over HTTP, the apache HttpClient library can be used to maintain a session between the CartAPI instance and the external service, caching the HttpClient instance in the session. The httpclient-4.1.jar is included in the project by default to support this use case.

   The methods also all throw CartException. The Endeca Mobile applications all store their own User, Cart, and Transaction state (user saved addresses, selected transaction addresses and cards, shipping options, subtotal amounts, etc). For adding, updating and removing saved addresses and credit cards and for selecting addresses, cards, and shipping methods, the Mobile applications will their own state accordingly unless a CartException is thrown. For instance, the if addAddress method does not throw an exception, the mobile applications will update their User state object, adding the Address object passed into the addAddress CartAPI method. This is not the case with the shopping cart items, the order summary, and the shipping methods, which are only updated from what is returned from the CartAPI methods.

2. Configure /WEB-INF/spring/app-config.xml to point to your class. By default, the Endeca API is set up to use a MockCartAPI class, which you can use as a reference or a starting place for your own implementation.

```
<bean id="cartApiService" class="com.endeca.mobile.services.cart.impl.MockCartAPI" />
```

QueryProcessor Reference

RecordFilterQueryPreProcessor

This pre-processor sets the given recordFilter on the query. If the query contains an existing record filter, it creates a new record filter from the union (AND) of the given recordFilter and the existing record filter, and sets this on the query.

Fully Qualified Class Name:
com.endeca.b2creference.services.query.impl.RecordFilterQueryPreProcessor

Property(ies):
recordFilter

Query types supported:
Navigation Query
Aggregate Record Query

Example:

```
<bean
class="com.endeca.b2creference.services.query.impl.RecordFilterQueryPreProces
sor">
      <property name="recordFilter" value=”pBuyable:1” />
</bean>
```

SimpleLoggingPostProcessor

Fully Qualified Class Name:
com.endeca.b2creference.services.query.impl.SimpleLoggingPostProcessor

Property(ies):
logserverHost
logserverPort

Query types supported:
Navigation Query
Aggregate Record Query

Example:

```
<bean
class="com.endeca.b2creference.services.query.impl.RecordFilterQueryPreProces
sor">
      <property name="logserverHost" value=”localhost” />
      <property name="logserverPort" value=”15010” />
</bean>
```

AggregateRecordsPreProcessor

This pre-processor sets the rollup (aggregation) key on the navigation query.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.AggregateRecordsPreProcessor

Property(ies):
aggregationKey

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.AggregateRecordsPreProcessor">
      <property name="aggregationKey" value=”pItemID” />
</bean>
```

FieldListConfigurationPreProcessor

This pre-processor sets the selection (field list) on the navigation query. This is used to limit the properties returned for navigation queries. For mobile, there is usually limited screen space, which naturally limits the amount of properties needed for navigation queries. Typically an MDEX Engine will be configured to show for more properties with navigation queries than a mobile device needs. This pre-processor is used to reduce the properties and dimensions to only the necessary ones, reducing the size of the response that needs to be sent to the mobile device.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.FieldListConfigurationPreProcessor

Property(ies):
fieldNames

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.FieldListConfigurationPreProcess
or">
      <property name="fieldNames">
            <list>
                  <value>product.id</value>
                  <value>product.name</value>
                  <value>product.price</value>
                  <value>product.brand.name</value>
                  <value>product.img_url_large</value>
                  <value>product.review.avg_rating</value>
                  <value>product.review.count</value>
                  <value>product.img_url_thumbnail</value>
                  <value>product.inventory.count</value>
```

```
            </list>
      </property>
</bean>
```

GeoFilterDistancePropertyPostProcessor

This post-processor makes it easier to access the kilometers_to_* and miles_to_* properties. It removes the kilometers_to_ and miles_to_ properties that the MDEX Engine returns when a geo filter or geo sort is applied, and adds 2 new properties that are the name of the geo filter property with ".distance_mi" or ".distance_km" appended.

For example, for the sort Ns=store.geocode(42.3621088,-71.0810343), the MDEX Engine returns the properties:

kilometers_to_store.geocode(42.3621088,-71.0810343) = 31.000662
miles_to_store.geocode(42.3621088,-71.0810343) = 19.262912

For the range filter, Nf=store.geocode|GCLT 42.3621088,-71.0810343 50, the MDEX Engine returns the properties:

kilometers_to_store.geocode|GCLT 42.3621088,-71.0810343 50 = 31.000662
miles_to_store.geocode|GCLT 42.3621088,-71.0810343 50 = 19.262912

This post-processor will remove these properties and create the following properties:

store.geocode.distance_km = 31.000662
store.geocode.distance_mi = 19.262912

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.GeoFilterDistancePropertyPostProcessor

Property(ies):
<NONE>

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.GeoFilterDistancePropertyPostPro
cessor" />
```

GeoLookupPreProcessor

Where a user's device cannot provide a gecode for geospatial search, this pre-processor uses the Google Maps Geocoding service to convert the text from a query string parameter into a geocode. If the text can be converted into a geocode, this pre-processor adds a geocode range filter (GCLT) to the navigation query, with a radius equal to the value of the geoFilterRadiusKm property. A geocode cache is maintained, the location of which is configured with the cacheFileResource property.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.GeoLookupPreProcessor

Property(ies):
locationParameter
geocodeProp
geoFilterRadiusKm
cacheFileResource

Query types supported:
Navigation Query

Example:

```
<bean class="com.endeca.mobile.services.query.impl.GeoLookupPreProcessor">
     <property name="locationParameter" value="location" />
     <property name="geocodeProp" value="store.geocode" />
     <property name="geoFilterRadiusKm" value="50.0" />
     <property name="cacheFileResource" value="/WEB-INF/zipgeocache.txt" />
</bean>
```

HiddenDimensionsPostProcessor

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.HiddenDimensionsPostProcessor

Property(ies):
<NONE>

Query types supported:
Navigation Query

Example:

MobileLoggingPostProcessor

This post-processor extends the SimpleLoggingPostProcessor, adding a MOBILE_PROFILE string to the log entry, based on the value of the Mp request parameter.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.MobileLoggingPostProcessor

Property(ies):
logserverHost
logserverPort

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.MobileLoggingPostProcessor">
      <property name="logserverHost" value=”localhost” />
      <property name="logserverPort" value=”15010” />
</bean>
```

SearchInterfaceQueryProcessor

This pre-processor adds the given searchInterface to searches on the Navigation Query if they don't contain a search interface. If appendLanguageCode is set to true, it will try to resolve the locale, and append the language code to the end of the searchInterface.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.SearchInterfaceQueryProcessor

Property(ies):
searchInterface
appendLanguageCode
defaultLanguageCode

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.SearchInterfaceQueryProcessor">
     <property name="appendLanguageCode" value="true" />
     <property name="defaultLanguageCode" value="en" />
     <property name="searchInterface" value="All_" />
</bean>
```

SelectedRecordsQueryProcessor

This query processor queries enables the selected records parameter (Rsel) to work properly. It builds a record filter based on the value of this parameter and adds it to the navigation query.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.SelectedRecordsQueryProcessor

Property(ies):
recordSpecName

Query types supported:
Navigation Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.SelectedRecordsQueryProcessor">
      <property name="recordSpecName" value="common.id" />
</bean>
```

URLParameterRecordFilterProcessor

This pre-processor creates and adds to the query, a new record filter from the concatenation of the value of the recordFilterPrefix property and the value of query string parameter configured by the urlParameter property. If the query contains an existing record filter, it creates a record filter from the union (AND) of the new recordFilter and the existing record filter, and sets this on the query. The recordFilterPrefix and the value of the urlParameter must not be null. If either are null, no record filter will be created.

Fully Qualified Class Name:
com.endeca.mobile.services.query.impl.URLParameterRecordFilterProcessor

Property(ies):
recordFilterPrefix
urlParameter

Query types supported:
Navigation Query
Aggregate Record Query

Example:

```
<bean
class="com.endeca.mobile.services.query.impl.URLParameterRecordFilterProcesso
r">
     <property name="recordFilterPrefix" value="review.product.id:" />
     <property name="urlParameter" value="product_id" />
</bean>
```