Technical Design Document

ODIL – DMSapi Solution Overview

OneDealer GmbH

Real Consulting Group

44 Kifisias Avenue,
151 25 Marousi Attikis
Athens, Greece

ODIL – DMSapi Solution Overview

ThinkRIT Ltd

1, Persefonis str,

152 34 Gerakas,

Attica, Greece

1. ODIL – DMSapi Solution Overview

2. ODIL overview

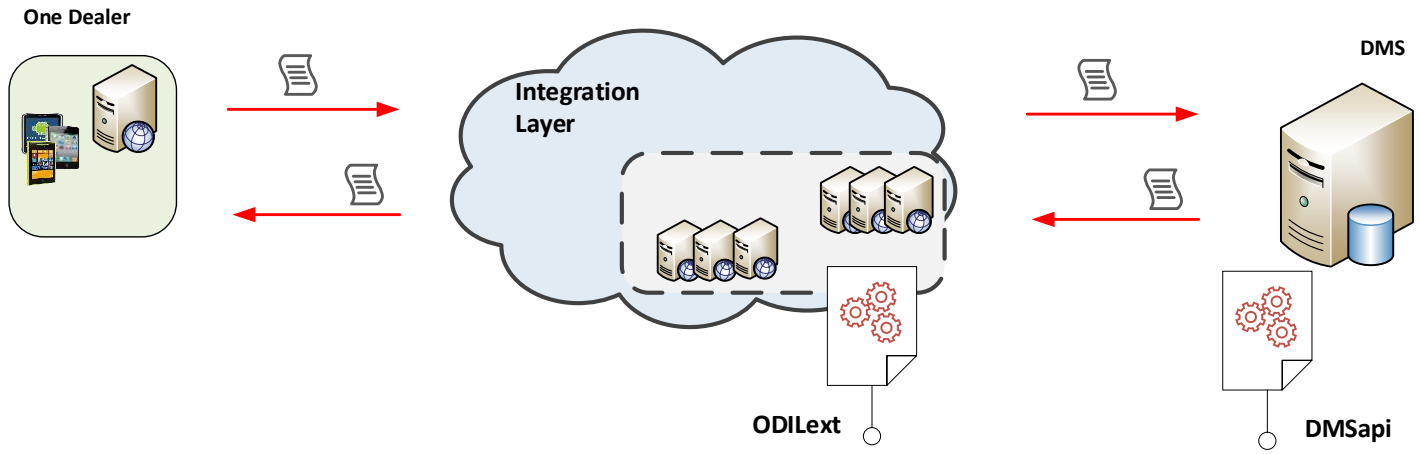


FIGURE 1: ODILext and DMSapi

3. Transformations, Mappings & Lookups

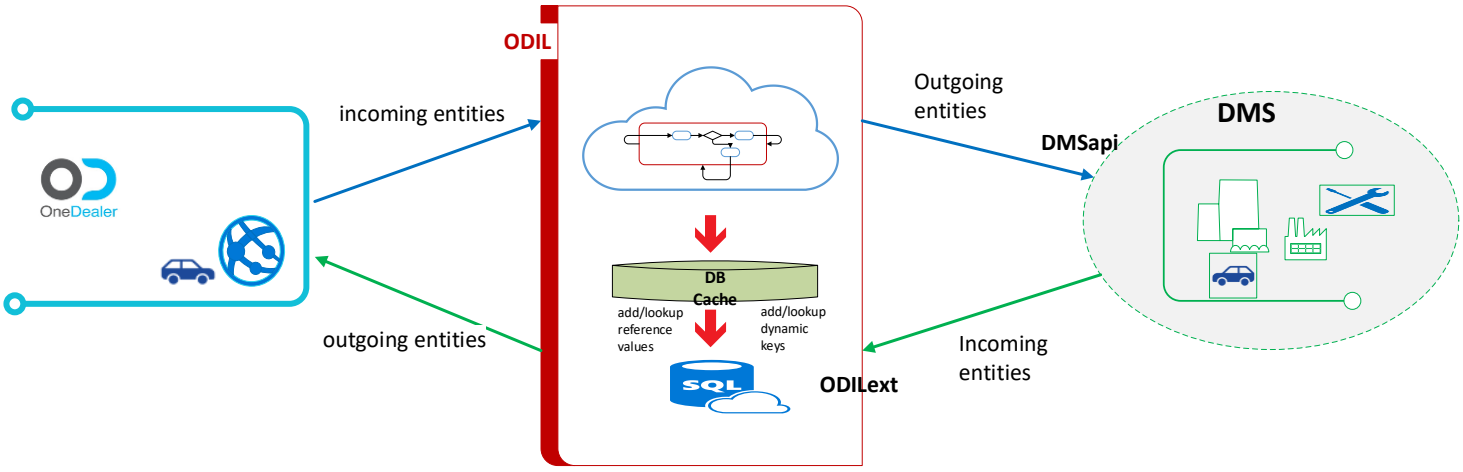


FIGURE 2 – Mappings

3.1. Static & Dynamic Mappings

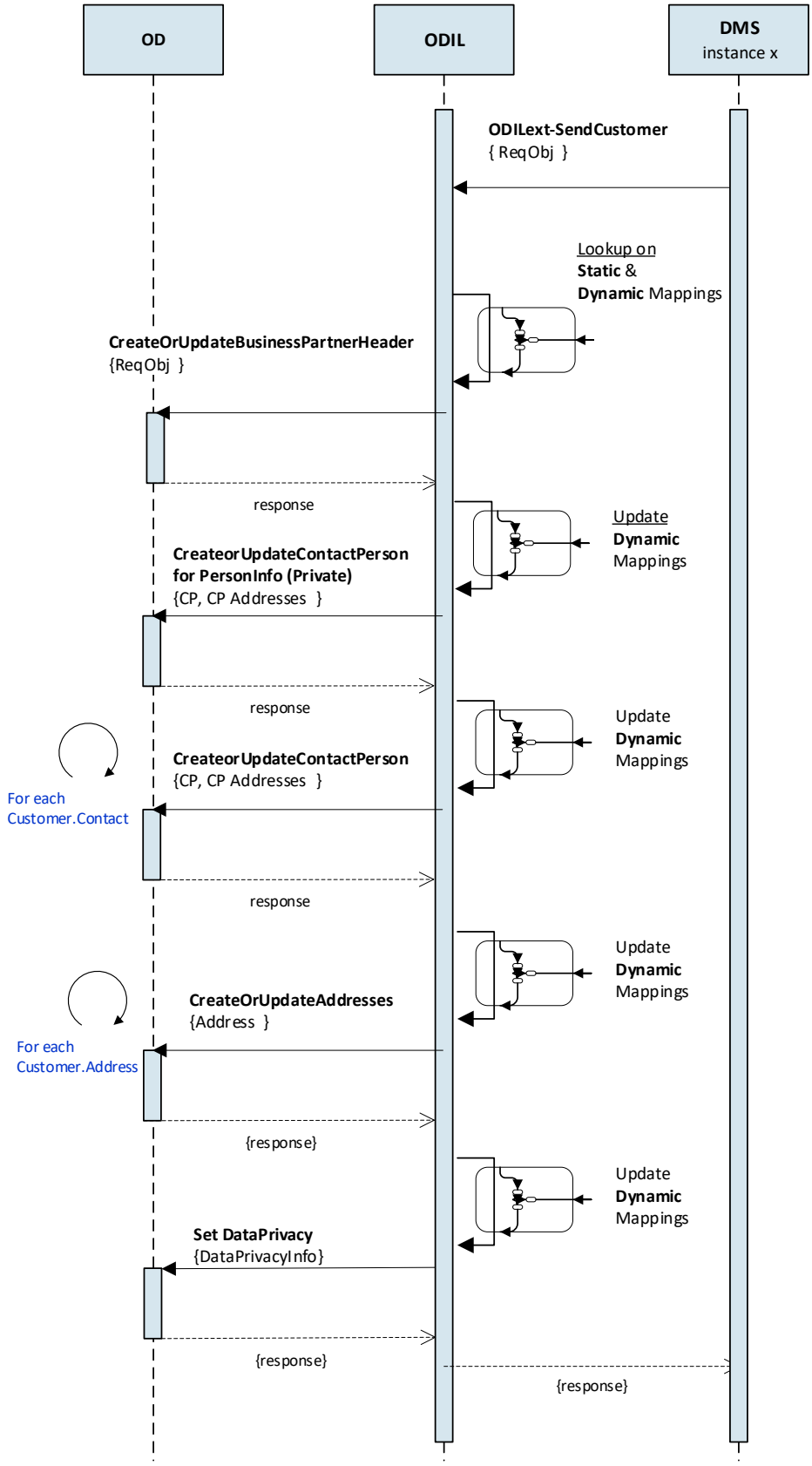


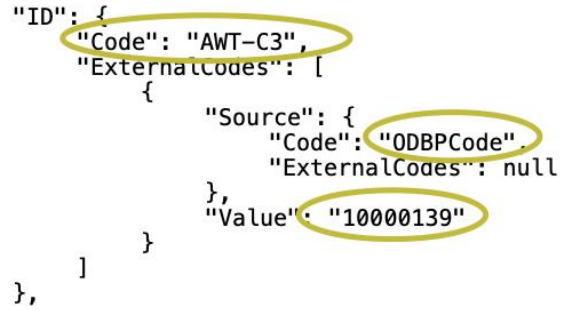
FIGURE 3 - Static & Dynamic Mappings

4. DMSapi – General Guidelines

4.1. Assumptions

- All referenceTypes and IDs must be either *null* or valid Codes (as returned by the Lookups), not ""

- For all entities. all codes are always from the DMS side.



Therefore,
  - ID.Code is DMS-side id,

whereas
  - ID.ExternalCodes is external to the DMS
    It is a placeholder where other systems ids may be found (e.g. OneDealer or OEM ids), in case they are useful to the DMs to store them, although it is not required to do it so.

    For every record in the ExternalCodes list ,
      - Source.Code is some identification that corresponds to the external source
      - Value holds the other system id.

- OD → backend communication
  - When sending a new entity from OD to DMS, the ID.Code is null for a newly created entity. This is an indicator that the entity is a new record, and you should return back the newly assigned ID from your system.

  - When updating an entity i.e. sending an existing one, the ID.Code has the previously returned value by the considered backend (DMS)

  - In all cases, ID.ExternalCodes collection will hold information at least for the OD internal Code.

- Backend → OD communication
  - When sending a new entity, from DMS to OD, the ID.Code must be the existing DMS internal identification code.

4.2. {baseURL}/session/create

This method returns a session for the requested ODIL api (ApiCode) in conjunction to the provided user credentials.

If its not used, e.g. basic authentication per call, it should return any string, as shown in the example provided.

4.3. {baseURL}/dms/lookups/?type=

DMSapi {url}/dms/Lookups method is required to be exposed by every DMS instance in order to return the codes used internally in the DMS and their associated descriptions, for every entity that is required to be mapped through the OD integration layer (ODIL).

Currently used types:

Make, (Model), (Carline), Company, Branch, Location,
VehicleType, ExteriorColor, VehicleBodyStyle, VehiclePriceList, FuelType, InteriorColor, InteriorType, GearsType, VehicleStatusClassification, TireSize, NumberOfDoors, EmissionsSticker, EmissionClass, Transmission, OptionFamily, OptionCategory, OptionSourceType, VehicleStatus, CylindersDisplacement,
MaritalStatus, PaymentTermsCountry, Currency, VATGroup, Language, Occupation, CustomerClassification, Title, Gender, PaymentType, DealDocumentStatus ,InquiryStatus, InquirySource, InquiryChannel, User

This function is crucial for the OD configuration. For each type, OD retrieves the available codes and associates them to the internally used codes. The generated mappings are saved in ODIL for lookup reference through the API method calls.

```
[
 {
  "Description":"string",
  "ID": {  "Code": "string"
```

```
      ,"ExternalCodes": [        ← Currently out of scope, can be safely omitted.
        {
          "Source": {
            "Code": "string",
            "ExternalCodes": [
              {}
            ]
          },
          "Value": "string"
        }
      ]
```

```
  }
 }
]
```

LookupRef types

Whenever such values are used (LookupRef), then the following rules take place during the transportation of the entities between the two systems (OD and DMS), in either direction.

- The *Code* property is replaced by the corresponding value found in the previously stored mapping in the ODIL, held separately for every single backend (DMS instance).
  If the value cannot be found, null is used.
- The *Description* property (if given) is always forwarded as is. It is up to the OD whether to store the given description, or the internal one (given the mapped code) or a combination of them

Note:

A workaround for *lookup types* that are not supported, is that this method returns hardcoded values; That is to say, to always return at least one value per type e.g. {“Not Available”, “N/A”}.

In this way, the OD user interface can perform the necessary mapping with a default or a non-available internal code, to keep consistency.

As a result, if the “N/A” code is given instead of *null* to a corresponding entity field, the default associated mapping will take place and be forwarded to the OD as defined. In the reverse direction, when the default OD code is encountered, it will be transformed to the “N/A” code instead of null.

Model tree

- master data will be published by DMS and uploaded to the OD through the corresponding ODILext method.
  - MakeCode will be mapped between OD and the backends.
  - CarlineCode (i.e. Family Code) and Model Codes will be used by OD as is (no value mapping takes place)

DealerId, CompanyId, BranchId, LocationId

- DealerId is used by the Incadea API as the BackendId; it is required to call the Incadea API, always in conjunction to the LocationId.
  - However CompanyId and BranchId are also required by the OD, so they should also be provided to the ODILext interface, e.g. for the Inquiry Creation/Update