Oracle Banking APIs

Taxonomy Configuration Guide
Release 19.1.0.0.0

Part No. F18559-01

May 2019

Taxonomy Configuration Guide

May 2019

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001
www.oracle.com/financialservices/

Table of Contents

1. **Preface** 4
   - 1.1 Intended Audience 4
   - 1.2 Documentation Accessibility 4
   - 1.3 Access to Oracle Support 4
   - 1.4 Structure 4
   - 1.5 Related Information Sources 4
2. **Revision History** 5
3. **Introduction** 6
4. **Taxonomy Validation Table structure** 7
5. **Taxonomy Validation Process** 9
6. **Data Type definition** 10
7. **Categories of taxonomy validators** 11
8. **Hierarchy to define field validation** 12
9. **Extended validators** 13
10. **Validators for complex data type** 14
11. **Configurations for taxonomy Validation** 15
12. **Key Things to note for Taxonomy Validation** 16
13. **Templates** 17
14. **Configuring Taxonomy Validation in UI** 23
15. **Utility to generate Validators** 24
16. **Manual to create Validators** 26
17. **Glossary** 31

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=accandid=docacc.

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=accandid=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=accandid=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

The subsequent chapters describes following details:

- Purpose
- Configuration / Installation.

1.5 Related Information Sources

For more information on Oracle Banking APIs Release 19.1.0.0.0, refer to the following documents:

- Oracle Banking APIs Installation Manuals

2. Revision History

	Name
Author	APIs Development team
Current Version	1
Date	01-Feb-2019

3. Introduction

Taxonomy validation is used to validate each field of the request object for each service in OBAPI application. This validation pattern for each field can be defined in OBAPI tables. The taxonomy validation can be used for language specific validation of fields. This validation can be configured at system as well as object level.

4. Taxonomy Validation Table structure

DIGX_FW_LOCALE_DATA_TYPE

This table holds the complete list of predefined data types in Out of the box OBAPI application. The default locale used to define data type is ‘en’. Separate entries for data types can be made for required locales. All the taxonomy data is validated against these data types. The validation pattern for specific data type is formed based on the column values of this table which is as follows:

Column Name	**Type**	**Description**
ID	VARCHAR2(100)	Unique identifier of the data type
LOCALE	VARCHAR2(3)	Locale identifier for which validation pattern is required
DESCRIPTION	VARCHAR2(255)	Description of the taxonomy
MINLENGTH	NUMBER	minimum length required for the field
MAXLENGTH	NUMBER	maximum length allowed for the field
PATTERN	VARCHAR2(255)	regex pattern required to be validated(if any)
ERRORCODE	VARCHAR2(255)	the default error code if the pattern validation fails
LENGTHERRORCODE	VARCHAR2(255)	error code thrown if length criteria fails
VALIDATION_CLASS	VARCHAR2(255)	Fully qualified name of the class that needs to be invoked for the validation.

DIGX_FW_TAXONOMY_DATA_TYPE_MAP

This table is used to define taxonomy and map it to the desired data type. The taxonomy can be defined at class level or field level in this table. The validation pattern for the taxonomy can be defined using following columns.

Column name	Type	Description
ID	VARCHAR2(255)	Unique identifier of the field to be validated. This can be only field name, field name in class hierarchy or complex fields parameters
TYPE	VARCHAR2(20)	(FIELD/CLASS/COMPLEX)
DATATYPEID	VARCHAR2(100)	Identifier of the Data Type to be applied on taxonomy
MINLENGTH	NUMBER	minimum length required for the field
MAXLENGTH	NUMBER	maximum length allowed for the field
MANDATORY	VARCHAR2(1)	Is the field value mandatorily required (Values - Y/N)
ERRORCODE	VARCHAR2(255)	the default error code if the pattern validation fails
LENGTHERROCODE	VARCHAR2(255)	error code thrown if length criteria fails
MANDATORYERRORCODE	VARCHAR2(255)	error code thrown if mandatory criteria fails

Note: For all the fields which are common in both the tables, the field value in 'DIGX_FW_TAXONOMY_DATA_TYPE_MAP' will take the precedence.

5. Taxonomy Validation Process

Taxonomy validation is applicable for all the objects that extend Validatable class. All the private member (excluding static) fields are validated using taxonomy validation. The member fields of the class which also extends the Validatable class will be validated recursively. If there is a List of the fields, all the values will be validated in loop.

The detailed validation process is as follows:

1. The taxonomy validation flow will be called as part of the traditional <DTO_Object>.validate method call in the service class.
2. The taxonomy validation should be applied or not for the respective DTO object is configured in preferences (Explained in 'Configurations' section)
3. All the classes in 'xface' and 'appcore/dto' package which extends Validatable class will have their corresponding validator class loaded in the system in the same package. The validator class name would be <DTO_NAME>Validator.java. (Note: these validator classes need not be written by developers. These files will be generated at build time and loaded in corresponding ear files)
4. All the member fields of the validatable object class will be validated against the corresponding taxonomy data loaded.
5. If the validation fails for a particular field the corresponding error will be added to validation error list. All the validation criteria will be applied to the field at once and all the possible errors codes will collated together.
6. The ValidationError object will contain error code, error message, the fully qualified name of the parent request DTO on which validate method is called and the fieldkey. The fieldkey is the name of the field with its hierarchy w.r.t parent class.
7. If there is no validation data maintained for the field at class level or field level, an exception will be raised.
8. The rest process of service execution will remain same.

6. Data Type definition

All the out of box available data types will be listed in ‘DIGX_FW_LOCALE_DATA_TYPE' table for default locale ('en').

If the data type needs to be redefined for a different locale (e.g. if pattern is required to be changed for different locale for a data type), a specific entry needs to be inserted for that data type for required data type.

If the validation pattern for a particular field is asked for a locale other than 'en' and corresponding entry is available in DIGX_FW_LOCALE_DATA_TYPE table the same will be applied. Otherwise by default validation pattern of 'en' will be used.

Perform following steps, if new data type is to be introduced for the taxonomy validation.

a. Make an entry in ' DIGX_FW_LOCALE_DATA_TYPE' for default locale ('en')

b. Define the validation definition using the columns available in the table (like minlength, maxlength, pattern etc)

c. Provide required error codes and their corresponding entries in 'DIGX_FW_ERROR_MESSAGES'

d. Create a taxonomy validation class for the data type and provide its fully qualified name in the respective column

e. The out of the box taxonomy validation class should reside in 'projects\framework\com.ofss.digx.appcore.dto\src\com\ofss\digx\app\dto\validator\taxonomy'

f. Each taxonomy validation class must implement 'com.ofss.digx.app.dto.validator.ITaxonomyValidator'. Use the overridden 'validate' method to provide logic for validation. It should be defined as singleton class and provide a ‘getInstance’ method for its loading. (Refer templates section for reference)

g. Also each taxonomy validation class must implement the ‘getCategory’ method. The validator category can be ‘NUMBER’, ‘TEXT’, ‘DATE’ or ‘OTHER’

7. Categories of taxonomy validators

The taxonomy has been divided into 4 categories.

1. NUMBER
2. TEXT
3. DATE
4. OTHER

The validation of taxonomy based on length and pattern is done based on above 4 categories. Each taxonomy validator has to define it category by implementing ‘getCategory()’ method.

Methodology to validate length based on category

- For TEXT category the minLength and maxLength columns will refer to the actual length of the string.
- For NUMBER category the minLength and maxLength columns will refer to the minimum and maximum allowed integer value for the input
- For DATE category the minLength and maxLength columns will refer to the minimum offset value (in number of days – positive or negative) from current date in which input date is allowed.
- For OTHER category length validation will not be performed.

8. Hierarchy to define field validation

The field can be defined for validation in one the following ways:

Level	Description	Entry in DIGX_FW_TAXONOMY_DATA_MAP	
		ID	**TYPE**
Field Level	This level is used to validate the taxonomy based on the name of the field. It will be applicable to all the fields with same name in the application irrespective of its data type. This validation can be overridden by a class level entry for specific request DTO	Exact name (case-sensitive) of the private member field of the request DTO class. e.g. payeeID, fromDate, name	FIELD
Class Level	This level is used to validate the taxonomy based on the complete hierarchical field name starting with fully qualified name of the request DTO.	Fully hierarchical name of the field. e.g. com.ofss.digx.app.dto.finlimit.TransactionalLimitDTO.owner.value	CLASS
Complex Field Level	This level is used to validate the fields of complex data type	As defined in the complex data type validator class.	COMPLEX

Note:
1. While defining the taxonomy in ‘DIGX_FW_TAXONOMY_DATA_MAP’, override the parameters of ‘DIGX_FW_LOCALE_DATA_TYPE’ as per the requirement.

2. If the entry for the field already exists in ‘DIGX_FW_TAXONOMY_DATA_MAP’, do not modify it for specific case. Instead make an entry of the field as per class hierarchy.

9. Extended validators

The default DTO validator classes can be extended using customized DTO validators. The customized validator must implement IDTOValidator class. The validation logic should be provided in overridden validate method. The default DTO validation logic can be used by extending the default validator of the DTO class and calling super.validate() followed by customized validation logic. The customized class name should be given in 'DIGX_FW_CONFIG_ALL_B' under category 'ExtValidationConfig'.

The extended validation class should contain a protected constructor and should use singleton pattern with a 'getInstance' method to return the validator object. Refer the templates section for sample extended validator.

10. Validators for complex data type

The complex data type in OBAPI can have specific validators. These validators are responsible for validating the fields in the complex data type. The validation of the fields can also be configured in 'DIGX_FW_TAXONOMY_DATA_MAP' (as per given in section 6). By using this, the field validation can be kept configurable using database entries (refer template section for sample code). The custom validation can also be written in validators specific to complex data type.

11. Configurations for taxonomy Validation

Following are the day-0 configuration properties related to taxonomy. All the properties are maintained in 'DIGX_FW_CONFIG_ALL_B'. For all properties default handling is for 'false'.

PROP_ID	VALUE	CATEGORY_ID	DECSRIPTION
TAXONOMY_VALIDATION_ENABLED	true/false	ValidationConfig	This property indicates whether taxonomy validation is required or annotation based validation should be applied.
CHECK_TAXONOMY_WHITELIST	true/false	ValidationConfig	This property will be used if taxonomy validation is enabled. If this property is true, the DTOs for which taxonomy needs to be enabled should be configured. If false, the DTOs for which taxonomy validation is not required should be configured.
<Fully_qualified_name_of_dto>.EnableTaxonomy	true/false	ValidationConfig	This property will be effective for respective DTO if 'CHECK_TAXONOMY_WHITELIST' is true. If this property is set to true, taxonomy validation will be applied to the DTO. Otherwise it will follow annotation based validation.
<Fully_qualified_name_of_dto>.DisableTaxonomy	true/false	ValidationConfig	This property will be effective for respective DTO if 'CHECK_TAXONOMY_WHITELIST' is false. If this property is set to true, annotation based validation will be applied to the DTO. Otherwise it will follow taxonomy validation.
<Fully_qualified_name_of_dto>	<Fully_qualified_name_of_extended_validator>	ExtValidationConfig	This property is used to specify the extended validator class for a specific DTO.

12. Key Things to note for Taxonomy Validation

1. Taxonomy validation is applied to all non-static private fields.
2. The fields should have appropriate getter method.
   a. For Boolean – the getter method should be 'is<Fieldname>' (e.g. isShared)
   h. For other types – the getter should be 'get<Fieldname>' (e.g. getPartyId)
3. If any field in the DTO is of type of an object which extends Validatable Class, the corresponding validator class will be responsible for its field validation.
4. For fields that do not require any specific data type validation, data type 'FREETEXT' can be mapped.
5. For list type of fields the validation will be done in loop, validating each field as per taxonomy definition.
6. In validationError, the structure of the error object is as follows-

   {

   "objectName": <fully qualified name of the DTO on which validate method is called>,

   "attributeName": < the field in which the validation has failed>,

   "errorCode": <error code>,

   "errorMessage": <locale specific message for the errorCode>

   }

   For list of objects or fields the attribute name will have an index concatenated by '#' in attributeName.

13. Templates

1. **Template to define Extended DTO validator**

- DTO name: com.ofss.digx.app.test.TestDTO
- Default validator: com.ofss.digx.app.test.TestDTOValidator
- Extended DTO validator: com.ofss.cz.app.test.ExtTestDTOValidator

```
package com.ofss.cz.app.finlimit.dto.limitpackage;

import java.util.List;

import com.ofss.digx.app.dto.validator.IDTOValidator;
import com.ofss.fc.app.dto.validation.IValidatable;
import com.ofss.fc.infra.validation.error.ValidationError;

public class ExtTestDTOValidator extends com.ofss.digx.app.test.TestDTOValidator
        implements IDTOValidator {


    protected ExtTestDTOValidator() {
    }

    public static ExtTestDTOValidator getInstance() {
        return ExtTestDTOValidatorHolder.INSTANCE;
    }

    private static class ExtTestDTOValidatorHolder {
        private static final ExtTestDTOValidator INSTANCE = new ExtTestDTOValidator();
    }

    @Override
    public void validateInput(IValidatable validatable, String key, String parentName,
            List<ValidationError> validationErrors) {
        LOGGER.log(Level.SEVERE, FORMATTER.formatMessage(
                "Class : %s, Entering into the customized class and calling digx class ",
THIS_COMPONENT_NAME));
        super.validateInput(validatable, key, parentName, validationErrors);
        // Provide the required addiotion validation for the DTO. Add the required errors in
validationErrors for the failed cases.
    }
}
```

2. Template to define taxonomy data type validator

Data Type: TESTTYPE

Validator: TestTypeValidator

```
package com.ofss.digx.app.dto.validator.taxonomy;

import java.util.List;
import com.ofss.digx.app.dto.validator.ITaxonomyValidator;
import com.ofss.digx.app.dto.validator.ValidationData;
import com.ofss.fc.infra.validation.error.ValidationError;

public class TestTypeValidator implements ITaxonomyValidator {

    private TestTypeValidator() {
    }
    private static class TestTypeValidatorHolder {
        private static final TestTypeValidator INSTANCE = new TestTypeValidator();
    }
    /**
     * @return unique instance of {@link TestTypeValidator}
     */
    public static TestTypeValidator getInstance() {
        return TestTypeValidatorHolder.INSTANCE;
    }
    @Override
    public void validate(ValidationData data, Object val, String parentName, String fieldKey,
            List<ValidationError> validationErrors) {
        if (val != null) {
            if (val instanceof <Type of the object>) {
                < Type of the object > value = (<Type of the object >) val;
                if (<validation_case>) {
                    validationErrors.add(new ValidationError(parentName, fieldKey, null,
data.getErrorCode(), null));
                }
            } else {
                validationErrors.add(new ValidationError(parentName, fieldKey, null,
"DIGX_INVALID_VALUE_TYPE", null));
            }
        }
    }
}
```

3. Template to define validator for custom data type

```
package com.ofss.digx.app.dto.validator.taxonomy;

import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.prefs.Preferences;

import com.ofss.digx.app.dto.validator.ITaxonomyValidator;
import com.ofss.digx.app.dto.validator.TaxonomyHandler;
import com.ofss.digx.app.dto.validator.TaxonomyValidatorFactory;
import com.ofss.digx.app.dto.validator.ValidationData;
import com.ofss.digx.enumeration.validator.TaxonomyCategory;
import com.ofss.fc.datatype.PostalAddress;
import com.ofss.fc.infra.config.ConfigurationFactory;
import com.ofss.fc.infra.log.impl.MultiEntityLogger;
import com.ofss.fc.infra.validation.error.ValidationError;

/**
 * Validation class for complex type @AmountRange
 */
public class PostalAddressValidator implements ITaxonomyValidator {

   private static final String THIS_COMPONENT_NAME =
PostalAddressValidator.class.getName();

   private static final Logger LOGGER =
MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);

   private static final MultiEntityLogger FORMATTER =
MultiEntityLogger.getUniqueInstance();

   /**
    * Constant to hold validation configuration preference name
    */
   private static final String VALIDATION_CONFIGURATION = "ValidationConfig";

   /**
    * Fully qualified name of the default length validator class
    */
   private static final String LENGTH_VALIDATOR = "LENGTH_VALIDATOR";

   /**
    * Fully qualified name of the default length validator class
    */
   private static final String DEFAULT_LENGTH_VALIDATOR =
"com.ofss.digx.app.dto.validator.taxonomy.LengthValidator";

   /**
    * private Constructor
    */
   private PostalAddressValidator() {

   }
```

```
    private static class PostalAddressValidatorHolder {
        private static final PostalAddressValidator INSTANCE = new PostalAddressValidator();
    }

    /**
     * @return unique instance of {@link PostalAddressValidator}
     */
    public static PostalAddressValidator getInstance() {
        return PostalAddressValidatorHolder.INSTANCE;
    }

    /**
     *
     */
    @Override
    public void validate(ValidationData data, Object val, String parentName, String fieldKey,
            List<ValidationError> validationErrors) {
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.log(Level.FINE, FORMATTER.formatMessage(
                    "Complex Data Type validation inside Class : %s, for Field : %s",
THIS_COMPONENT_NAME, fieldKey));
        }
        PostalAddress v = (PostalAddress) val;


validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne1." + data.getLocale()),
                v.getLine1(), parentName, fieldKey + ".line1", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne2." + data.getLocale()),
                v.getLine2(), parentName, fieldKey + ".line2", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne3." + data.getLocale()),
                v.getLine3(), parentName, fieldKey + ".line3", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne4." + data.getLocale()),
                v.getLine4(), parentName, fieldKey + ".line4", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne5." + data.getLocale()),
                v.getLine5(), parentName, fieldKey + ".line5", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne6." + data.getLocale()),
                v.getLine6(), parentName, fieldKey + ".line6", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne7." + data.getLocale()),
                v.getLine7(), parentName, fieldKey + ".line7", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne8." + data.getLocale()),
            v.getLine8(), parentName, fieldKey + ".line8", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne9." + data.getLocale()),
            v.getLine9(), parentName, fieldKey + ".line9", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne10." + data.getLocale()),
            v.getLine10(), parentName, fieldKey + ".line10", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne11." + data.getLocale()),
            v.getLine11(), parentName, fieldKey + ".line11", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.li
ne12." + data.getLocale()),
            v.getLine12(), parentName, fieldKey + ".line12", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.c
ity." + data.getLocale()),
            v.getCity(), parentName, fieldKey + ".city", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.s
tate." + data.getLocale()),
            v.getState(), parentName, fieldKey + ".state", validationErrors);

validateField(TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.c
ountry." + data.getLocale()),
            v.getCountry(), parentName, fieldKey + ".country", validationErrors);
      validateField(

TaxonomyHandler.getInstance().getValidationMap().get("POSTALADDRESS.postalCode." +
data.getLocale()),
            v.getPostalCode(), parentName, fieldKey + ".postalCode", validationErrors);

   }

   /**
    * Validate individual field of the complex data type {@link PostalAddress}
    *
    * @param data
    * @param value
    * @param parentName
    * @param key
    * @param validationErrors
    */
   private void validateField(ValidationData data, String value, String parentName, String
key,
         List<ValidationError> validationErrors) {

      Preferences validationFactoryConfigurator = ConfigurationFactory.getInstance()
            .getConfigurations(VALIDATION_CONFIGURATION);

      ITaxonomyValidator lengthValidator = TaxonomyValidatorFactory.getInstance()
            .getValidator(validationFactoryConfigurator.get(LENGTH_VALIDATOR,
DEFAULT_LENGTH_VALIDATOR));
```

```
        if (value == null) {
            if (data.isMandatory()) {
                validationErrors.add(new ValidationError(parentName, key, null,
data.getMandatoryErrorCode(), null));
            }
        } else {
            if (data.getMinLength() != null || data.getMaxLength() != null) {
                lengthValidator.validate(data, value, parentName, key, validationErrors);
            }

            ITaxonomyValidator validator =
TaxonomyValidatorFactory.getInstance().getValidator(data.getClassName());
            if (validator != null) {
                validator.validate(data, value, parentName, key, validationErrors);
            }
        }

    }

    @Override
    public final TaxonomyCategory getCategory() {
        return TaxonomyCategory.OTHER;
    }
}
```

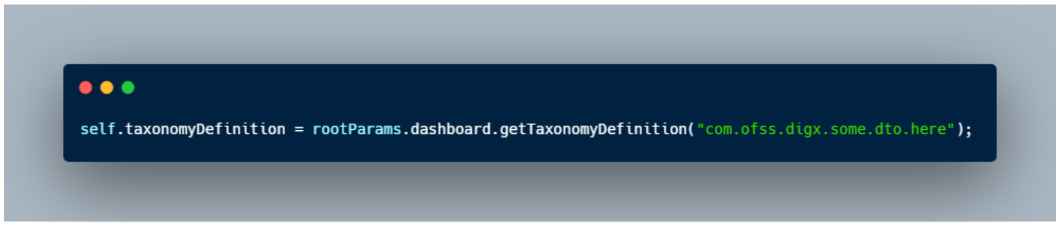
14. Configuring Taxonomy Validation in UI

To integrate the taxonomy based validation in UI components following steps are required for migration from earlier UI based validations.

View Model

Create **taxonomyDefinition** instance by calling **getTaxonomyDefinition** method from dashboard.
It accepts the name of the DTO as the only argument.

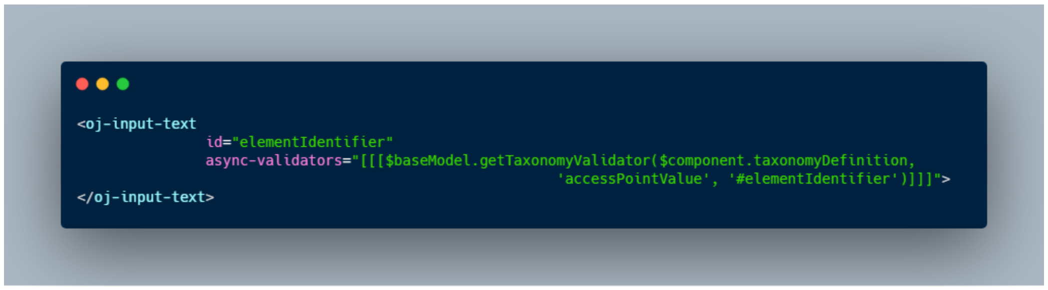
A REST call to fetch taxonomy details will be performed for each invocation of **getTaxonomyDefinition**.



```
self.taxonomyDefinition = rootParams.dashboard.getTaxonomyDefinition("com.ofss.digx.some.dto.here");
```

HTML

getTaxonomyValidator function accepts **taxonomyDefinition** created in the View Model as first argument, and idField to search within **taxonomyDefinition** as second argument, and element selector as the third argument.



```
<oj-input-text
               id="elementIdentifier"
               async-validators="[[[$baseModel.getTaxonomyValidator($component.taxonomyDefinition,
                                                   'accessPointValue', '#elementIdentifier')]]]">
</oj-input-text>
```

15. Utility to generate Validators

Once the taxonomy validation is enabled for a request DTO, it is mandatory to have a taxonomy validator for the same in the application. A utility has been provided to generate the validators for such request DTOs. This section describes the steps to be followed to generate the validators using this utility.

Prerequisites:

1. The machine should have JDK version 1.7 or above installed
2. The project having the custom request DTOs must be ready
3. Supporting jars as mentioned below
   a. com.ofss.digx.appcore.dto.jar – (available in 'obapi.app.framework.ear')
   b. com.ofss.fc.infra.jar – (available in 'obapi.app.core.domain.ear')
   c. com.ofss.fc.appcore.dto.jar – (available in 'obapi.app.core.domain.ear')
   d. Any other jars required for compilation of project containing custom request DTOs

Running the Utility:

This section explains step by step process to generate the validators using the utility.

Step 1: Copy the following artifacts from the <installer> to desired directory. This directory will be referred as <UTIL_DIR>.

1. validator_gen.bat (for Windows) OR validator_gen.sh (for Linux)
2. lib folder – (contains pre-built jars required for utility)

Add the supporting jars as mentioned in the prerequisites (point no 3) in the lib folder.

Step 2: Go to < UTIL_DIR > and open command prompt / terminal.

Execute following command:

a. For Linux
   ./validator_gen.sh <BIN_DIR> <UTIL_DIR> <PROJECT_DIR> <PACKAGE_NAME>
b. For Windows
   ./validator_gen.bat <BIN_DIR> <UTIL_DIR> <PROJECT_DIR> <PACKAGE_NAME>

Where,

<BIN_DIR>: Path where JDK bin is located (e.g. C:\Program Files\Java\jdk1.8.0_171\bin) (use " if the path contains spaces)

<PROJECT_DIR>: path where request DTO project is located

<PACKAGE_NAME>: name of the package where the source of request DTOs is located

Consider following example

If the customized request DTOs are located in 'D:\workspaces\trunk\com.ofss.digx.cz.app.xface\src \com…'

Then, < PROJECT_DIR > = 'D:\workspaces\trunk' and < PACKAGE_NAME > = 'com.ofss.digx.cz.app.xface'

Step 3: As a result of utility, a jar containing the compiled classes of validator files will be created in <UTIL_DIR>. Copy the jar file and paste in the EAR file where the jar of customized DTO classes is located. Deploy the EAR file in the application and restart the server.

Note:
As the utility runs, it creates the source folder of the validator classes in <PROJECT_DIR>. By default, this folder is deleted once the utility execution is completed. This source folder can be retained by following way.

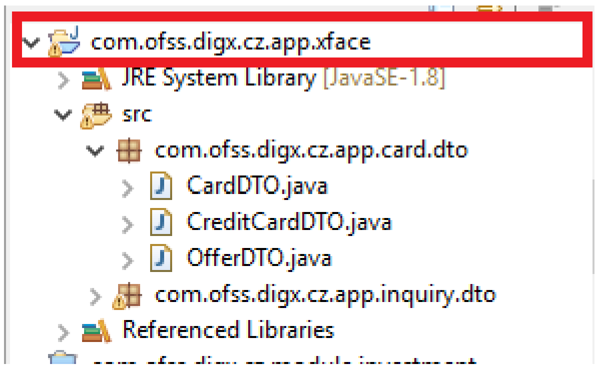
Open ‘validator_gen.bat’ or ‘validator_gen.sh’ in notepad and set the value of variable ‘isValidatorSourceRequired’ to Y. Run the utility again. The source will available in <PROJECT_DIR>.

16. Manual to create Validators

If the utility to generate validator is not available, following steps can be referred to create the request DTO validator manually. This can be created in the same workspace as that of request DTO.

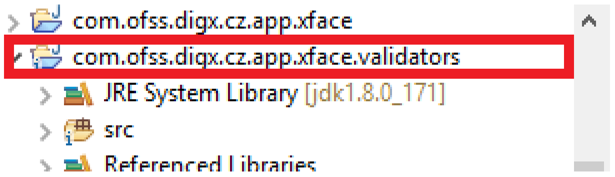
Project Creation for validators

1. Import the project containing customized DTO in eclipse workspace



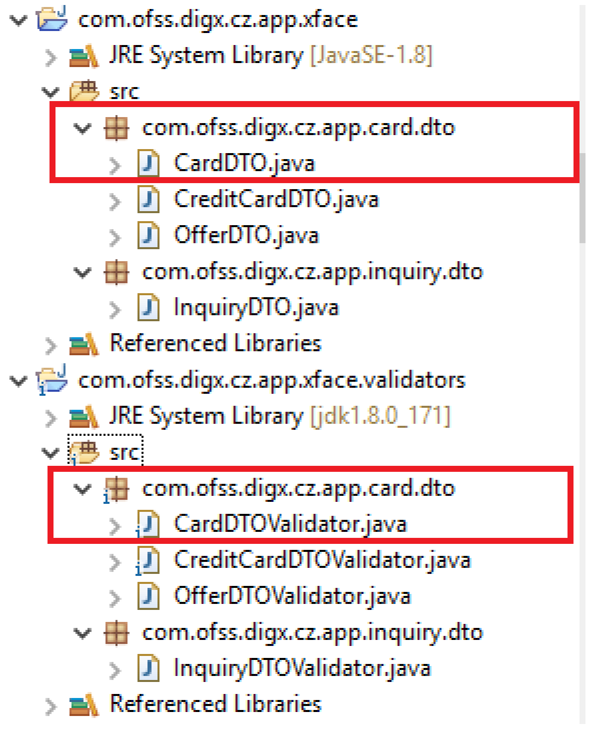
```
com.ofss.digx.cz.app.xface
  JRE System Library [JavaSE-1.8]
  src
    com.ofss.digx.cz.app.card.dto
      CardDTO.java
      CreditCardDTO.java
      OfferDTO.java
    com.ofss.digx.cz.app.inquiry.dto
  Referenced Libraries
```

2. Create project in the workspace for valiadator classes. Name the project by appending '.validators' to the name of project containing DTOs



```
com.ofss.digx.cz.app.xface
com.ofss.digx.cz.app.xface.validators
  JRE System Library [jdk1.8.0_171]
  src
  Referenced Libraries
```

3. Create the validator class for each of the DTO. Maintain the same package structure as that of DTO project while creating the validator classes. To name the validator, use the DTO name appended with prefix ‘Validator’.


com.ofss.digx.cz.app.xface
JRE System Library [JavaSE-1.8]
src
com.ofss.digx.cz.app.card.dto
CardDTO.java
CreditCardDTO.java
OfferDTO.java
com.ofss.digx.cz.app.inquiry.dto
InquiryDTO.java
Referenced Libraries
com.ofss.digx.cz.app.xface.validators
JRE System Library [jdk1.8.0_171]
src
com.ofss.digx.cz.app.card.dto
CardDTOValidator.java
CreditCardDTOValidator.java
OfferDTOValidator.java
com.ofss.digx.cz.app.inquiry.dto
InquiryDTOValidator.java
Referenced Libraries

Validator file creation

Use following steps to create validator class:

In this document, we will use CardDTO, OfferDTO, CreditCardDTO as example to create validators. CardDTO extends DataTransferObject and CreditCardDTO extends CardDTO.

For these DTOs, CardDTOValidator, OfferDTOValidator and CreditCardDTOValidator will be created.

1. Each validator class must extend a parent DTO validator class.

If Request DTO extends DataTransferObject then the validator class should extend ‘com.ofss.digx.app.dto.validator.AbstractDTOValidator’.

E.g. public class CardDTOValidator extends AbstractDTOValidator implements IDTOValidator {

If request DTO extends another request DTO then the validator class should extends the validator class of parent DTO.

E.g. public class CreditCardDTOValidator extends CardDTOValidator implements IDTOValidator {

2. Each validator class must implement the interface ‘`com.ofss.digx.app.dto.validator.IDTOValidator`’

Each validator should be a singleton class and must contain ‘getInstance’ method to return validator class instance. The class can be made singleton by any method. One of the method is given in following snippet.

```
protected CreditCardDTOValidator() {
}

/**
 * @return instance of {@link CreditCardDTOValidator}.
 */
public static CreditCardDTOValidator getInstance() {
   return CreditCardDTOValidatorHolder.INSTANCE;
}

/**
 * The class holds the instance of {@link CreditCardDTOValidator}.
 */
private static class CreditCardDTOValidatorHolder {
   /**
    * private instance variable {@value new CreditCardDTOValidator()}.
    */
   private static final CreditCardDTOValidator INSTANCE = new CreditCardDTOValidator();
}
```

3. Validator class should override 'validateInput' with signature as given below:

```
@Override
   public void validateInput(IValidatable validatable, String key, String parentName,
         List<ValidationError> validationErrors) {
```

4. In 'validateInput' method: include following content in the beginning-
   a.
      ```
      String parent = parentName != null ? parentName : validatable.getClass().getName();
      String fullKey = (key != null ? (key + ".") : "");
      super.validateInput(validatable, key, parent, validationErrors);
      ```

      (The super.validateInput validates the fields of parent class by calling parent validator's 'validateInput' method.
   b. 'validatable' parameter of 'validateInput' should be casted to DTO which we want to validate. For example, in case of CardDTOValidator, CardDTO is to be validated. Therefore, below code should be used for casting

CardDTO cardDTO = (CardDTO) validatable;

c. Create a HashMap of String as key and Object as value as in below snippet

```
Map<String,Object> fieldsMap = new HashMap<String,Object>();
```

Add all fields of DTO in hashmap one by one.

'fullKey' appended with particular fieldname should be inserted as key of hashmap and corresponding field as value of hashmap .

For eg in case of CardDTO below snippet should be added to CardValidator,

```
Map<String,Object> fieldsMap = new HashMap<String,Object>();
fieldsMap.put(fullKey + "id",cardDTO.getId());
fieldsMap.put(fullKey + "cvv",cardDTO.getCvv());
fieldsMap.put(fullKey + "active",cardDTO.isActive());
```

d. If some DTO have other DTO in its attribute, then to validate attribute level DTO, corresponding DTOs validator should be called.

For e.g. CardDTO contains a field cardProductDTO of type CardProductDTO. So CardValidator should have following snippet to validate field of type CardProductDTO.

```
if(cardDTO.getCardProductDTO() != null) {

DTOValidatorFactory.getInstance().getValidator(cardDTO.getCardProductDTO()).validateInput(cardDTO.getCardProductDTO(), fullKey + "cardProductDTO", parent, validationErrors);
}
fieldsMap.put(fullKey + "cardProductDTO",cardDTO.getCardProductDTO());
```

e. If some DTO contains list of objects, include following snippet for its validation.

```
if(cardDTO.getOffers() != null && !cardDTO.getOffers().isEmpty()) {
    int dtoVarIndex = 0;
    for(OfferDTO dtoVar : cardDTO.getOffers()) {
      DTOValidatorFactory.getInstance().getValidator(dtoVar).validateInput(dtoVar,
fullKey + "offers#" + dtoVarIndex, parent, validationErrors);
      dtoVarIndex++;
    }
  }
fieldsMap.put(fullKey + "offers",cardDTO.getOffers());
```

f. Finally add following method to validate parameters.

```
DefaultDTOValidator.validateParams(fieldsMap,parent,validationErrors
);
```

17. Glossary

Validatable Class	com.ofss.fc.app.dto.validation.Validatable
xface	Project – ‘core\middleware\projects\common\com.ofss.digx.app.xface’
appcore/dto	Project – ‘core\middleware\projects\framework\com.ofss.digx.appcore.dto’
ValidationError Class	com.ofss.fc.infra.validation.error.ValidationError