Oracle Banking Digital Experience

Extensibility Guide
Release 18.3.0.0.0

Part No. F12056-01

December 2018

ORACLE®

Extensibility Guide

December 2018

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

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 Digital Experience Release 18.3.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Licensing Guide

2. Objective and Scope

2.1 Background

OBDX is designed to help banks respond strategically to today's business challenges, while also transforming their business models and processes to reduce operating costs and improve productivity across both front and back offices. It is a one-stop solution for a bank that seeks to leverage Oracle Fusion experience across its core banking operations across its retail and corporate offerings.

OBDX provides a unified yet scalable IT solution for a bank to manage its data and end-to-end business operations with an enriched user experience. It comprises pre-integrated enterprise applications leveraging and relying on the underlying Oracle Technology Stack to help reduce in-house integration and testing efforts.

2.2 Objective

While most product development can be accomplished via highly flexible system parameters and business rules, further competitive differentiation can be achieved via IT configuration & extension support. Time consuming, custom coding to enable region specific, site specific or bank specific customizations can be minimized by offering extension points and customization support which can be implemented by the bank and / or by partners.

Extensibility objective

OBDX when extended & customized by the Bank and / or Partners results in reduced dependence on Oracle. As a result of this, the Bank does not have to align plans with Oracle's release plans for getting certain customizations or product upgrades. The bank has the flexibility to choose and do the customizations themselves or have them done by partners.

One of the key considerations towards enabling extensibility in OBDX has been to ensure that the developed software can respond to future growth. This has been achieved by disciplined software development leading to cleaner dependencies, well defined interfaces and abstractions with corresponding reduction in high cohesion & coupling. Hence, the extensions are kept separate from Core – Bank can take advantage of OBDX Core upgrades as most extensions done for a previous release can sit directly on top of the upgraded version. This reduces testing effort thereby reducing overall costs of planning & taking up an upgrade. This would also improve TTM significantly as the bank enjoys the advantage of getting universal features through upgrades.

The broad guiding principles w.r.t. providing extensibility in OBDX are summarized below:

- Strategic intent for enabling customers and partners to extend the application.
- Internal development uses the same principles for client specific customizations.
- Localization packs.
- Extensions by Oracle Consultants, Oracle Partners, Banks or Bank Partners.
- Extensions through the addition of new functionality or modification of existing functionality.
- Planned focus on this area of the application.
- Standards based.
- Leverage large development pool for standards based technology.
- Developer tool sets provided for as part of JDeveloper and Eclipse for productivity.

2.3 Scope

The scope of this document is to explain the ***customization*** & ***extension*** of OBDX for the following use cases:

- Customizing OBDX application services and implement composite application services
- Adding pre-processing or post processing validations in the application services extension
- Adding Business Logic in pre hook or post hook points in the application services extension
- Altering the product behavior at customizations hooks provided as adapter calls in functional areas that are prone to change and in between modules that can be replaced (e.g. alerts, content management)
- Adding new fields to the OBDX domain model and including it on the corresponding screen.
- Defining the security related access and authorization policies
- Defining different security related rules, validator and processing logics
- Customizing OBDX UI
- Adding a new field or a table on the screen
- Removing fields from the UI

This document would be a useful tool for Oracle Consulting, bank IT and partners for customizing and extending the product.

The document is a developer's extensibility guide and does not intend to work as a replacement of the functional specification which would be the primary resource covering the following:

- OBDX installation & configuration.
- OBDX parameterization as part of implementation.
- Functional solution and product user guide.

Out of scope

The scope of extensibility does not intend to suggest that OBDX is forward compatible.

2.4 Structure

This document is organized into following chapters

- Architecture of Service Tier: Provides overall architecture of the service tier of OBDX platform. This chapter will set the context for further chapters and also will introduce you to various terminologies that you will encounter throughout this document
- Extensible Points in Service Tier: Provides in depth knowledge about various extensible hooks available in the service tier.
- Architecture of GUI Tier: Provides overall architecture of the GUI tier of OBDX platform. This chapter will introduce you to various terminologies that you will encounter for UI extensibility.
- Extensible points in GUI Tier: Provides in depth knowledge about various extensible hooks available in the GUI tier.
- Libraries: Provides a listing of various libraries provided by OBDX out of the box along with their usage
- Workspace Setup: Provides step by step guidelines for setting up Eclipse workspace for extensibility
- Deployment: Provides information in packaging and deployment of the customized code on Weblogic server
- GUI Tier – Workspace Setup: Provides step by step guidelines for setting up workspace for GUI tier extensibility
- GUI Tier – Deployment: Provides information on packaging and deployment of customized GUI code on HTTP server
- Use Cases: This chapter discusses some of the extensibility points covered in earlier chapters with the help of some use cases.

Home

3. Architecture of Service Tier

Let's go through the building blocks of OBDX framework (also known as DIGX framework). To build a REST API, each of these framework components (as mentioned below) needs to be addressed and that's why it becomes important to have a holistic idea about each of them. The arrangement of all of these framework components can be clearly understood in the following diagram:

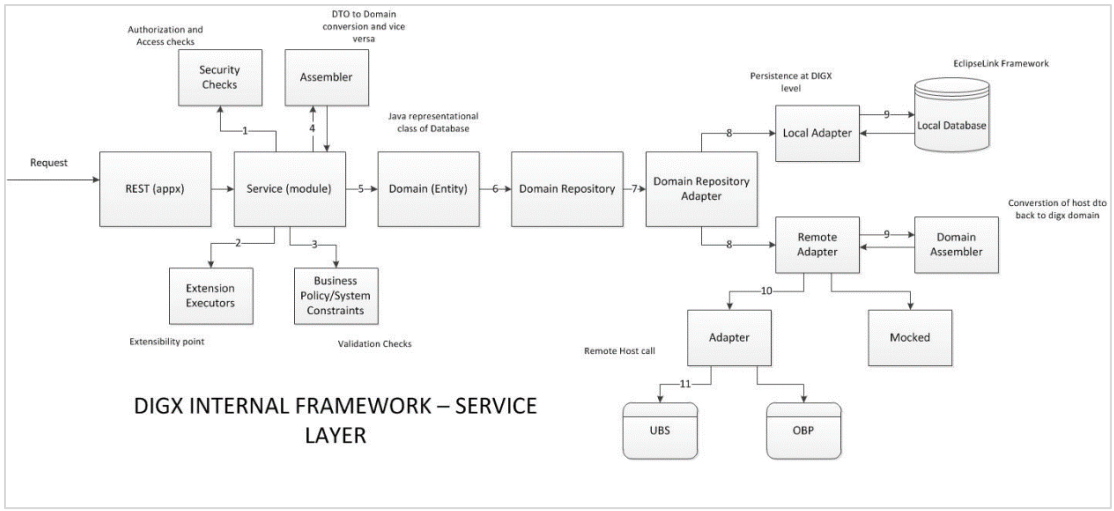


Figure 1 DIGX Service Layer

1. **REST:** The endpoint layer which gets invoked whenever a request URI is called. Also known as the layer which contains REST annotations and path to resources or sub-resources of an application
2. **Service:** Also called as module layer of the framework. Generally, the core modules of DIGX application will have their own service implementation classes responsible for implementing core business logic, validation and security checks
3. **Assemblers:** These are the mapping classes which convert data object containing request or response parameters into domain or database compatible form. These classes help us to get the required domain objects which can be further used in object-relational mapping
4. **Business Policy/ System Constraints:** Before letting the query data read or persisted in the core application, certain business policies need to be validated. This separate layer of constraints check let the application behave as per the policies configured
5. **Domain/Entity:** Represents the Java Object form of Database. This domain layer also contains data to be persisted or query response fetched through Object relational mapping
6. **Domain Repository:** The term 'repository' denotes any data storage component. Each module of the application will have its own repository to manage its CRUD operations and that can be easily done using this component of the DIGX framework
7. **Domain Repository Adapter:** Adapters are the connecting points to some external system and as the name suggests, this part of the framework contacts two kinds of repositories of DIGX application – Local Repository and Remote Repository. Eventually, the configured one out of these two will be invoked

8. **Adapters:** Finally these are the adapter classes that can call either Local Database (DIGX specific tables) or Remote Repository (external system). Remote adapters can further be mocked if required
9. **External System/ Host:** The core banking application such as UBS/FCORE or OBP or any third-party application which operates final banking transactions.

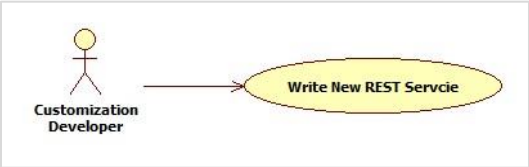
Home

4. Extensible Points in Service Tier

Various extensible points / hooks provided by OBDX framework, are explained in detail in this section

4.1 REST Tier

Customization developer can extend the REST tier by writing new REST services. This new REST service will consume new or existing application service. Please note that it is not possible to customize the REST services provided out of the box. Extensibility in REST tier is limited to writing new services.



References:

Please refer to workspace setup of DTO (xface) and REST service.

Please refer to Use case 1 for steps to write new REST service along with sample code.

4.1.1 Guidelines

- OBDX REST tier follows façade pattern, meaning that it is just an endpoint built on top of application service(s).
- A REST service should not have any business logic. It should consume one or more application services and prepare the response.
- Before coding a new REST service, developer should decide the resource(s) and sub-resources(s) that s/he needs to develop. Based on this, the developer can design required URIs. E.g. A ‘Demand Deposit Account’ is a resource in the system and /accounts/demandDeposit/{accountId} is the REST URI to access it.
- The service should be annotated suitably using JAX-RS annotations.
- The service should wrap its operation in ‘Channel Interaction’.
- The service should use adequate logging.

4.1.2 HTTP Standards

HTTP Methods

OBDX resources support following HTTP methods. New services also should use these methods appropriately.

Method	Purpose
GET	Retrieve / fetch the resource
POST	Create a new resource
PUT	Update / modify an existing resource. The payload is expected to have full resource.
PATCH	Update / modify very small part of an existing resource. The payload is expected to have only the fields to be updated.
DELETE	Delete a resource

HTTP Response Codes

Following HTTP response codes are used. New REST services should return appropriate response code based on result of the operation.

Code	Status	Description
200	OK	Request successfully executed and the response has content
201	Created	Resource successfully created
202	Accepted	Request has been accepted for processing but processing has not been completed
204	No Content	Request successfully executed and the response doesn't have content
304	Not Modified	The resource has not been modified for a conditional GET request

400	Bad Request	The request could not be understood by the server due to malformed syntax
401	Unauthorized	The request requires user authentication, or authorization has been refused for the credential passed in the request
404	Not Found	The requested resource was not found
500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request

4.2 Service Extensions

This extension point should be used when the customization developer needs additional business logic for an application service. This additional logic, which is not available as part of the digital experience product functionality, but could be a client requirement. For these purposes, two hooks are provided in the application code:

Pre-extension hook

This extension point is available in application service before it performs any validations and executes business logic. This hook can be important in the following scenarios:

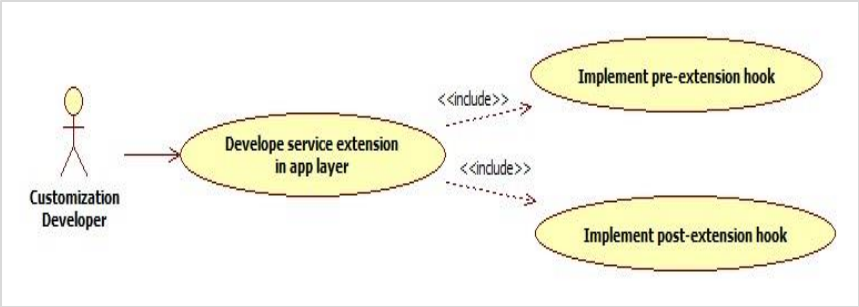
- Additional input validations
- Execution of business logic, which necessarily has to happen before going ahead with normal service execution.

Post-extension hook

This extension point is available in the application service after it has executed business logic. This hook can be important in the following scenarios:

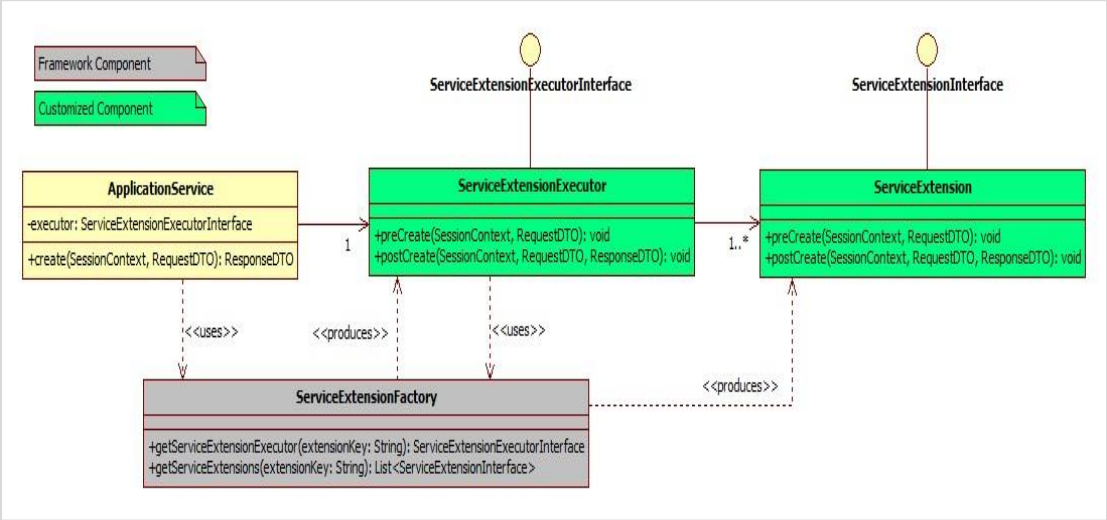
- Output response manipulation
- Custom data logging for subsequent processing or reporting.

Both ‘pre’ and ‘post’ service extensions are available in the application service layer (also known as the ‘app’ layer) of OBDX.



This hook in implemented using service extension executor and service extensions. These components are explained in detail below. Customization developer can use these components suitably based on the requirement.

Below class diagram depicts the relationship between application service, extension executor and extensions. The diagram considers a sample ‘create’ method in application service.



Note: The RequestDTO and ResponseDTO components depicted in above diagram are explained in subsequent sections. For now, note that the RequestDTO contains inputs to the application service method and ResponseDTO contains output generated by the method.

4.2.1 Service Extension Interface

This interface has a pair of pre and post method definitions for each application service method of the present. A service extension class has to implement this interface. The ‘pre’ method is the pre-extension hook as explained before. Similarly the ‘post’ method is the post-extension hook.

Multiple implementations can be defined for a particular service. The service extensions executor invokes all the implementations defined for the particular service both before and after the actual service executes. The signatures of these methods are:

public void pre<Method_Name>(SessionContext, <Method_Parameters>) throws Exception;

public void post<Method_Name>(SessionContext, <Method_Parameters>, ResponseDTO) throws Exception;

Naming Convention

The naming convention of service extension interface is

I<Service_Name>Ext

For example, consider below code sample.

```
public interface ILoanApplicationExt {

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @throws Exception
     */
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
            throws Exception;

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required after
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *            The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *            The instance of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
     *            request.
     *
     * @param loanApplicationResponseDTO
     *            The instance of type {@link LoanApplicationCreateResponseDTO} used for creating loan application
     *            response.
     *
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext,
            LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
            LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception;

    /**
     * Extension point for LoanApplication.update. The method is intended to keep all the extensions required before
     * updating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *            The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationUpdateRequestDTO
     *            The request DTO of type {@link LoanApplicationUpdateRequestDTO} used for updating loan application of
     *            any party.
     *
     * @throws Exception
     */
```

4.2.2 Service Extension Executor Interface

This acts as an interface for the application service to access service extensions. The implementing class creates an instance each of all the extensions defined in the service extensions configuration file. If no extensions are defined for a particular service, the executor creates an instance of the default extension for the service. The executor also has a pair of pre and post methods for each method of the actual service. These methods in turn call the corresponding methods of all the extension classes defined for the service (extension chaining).

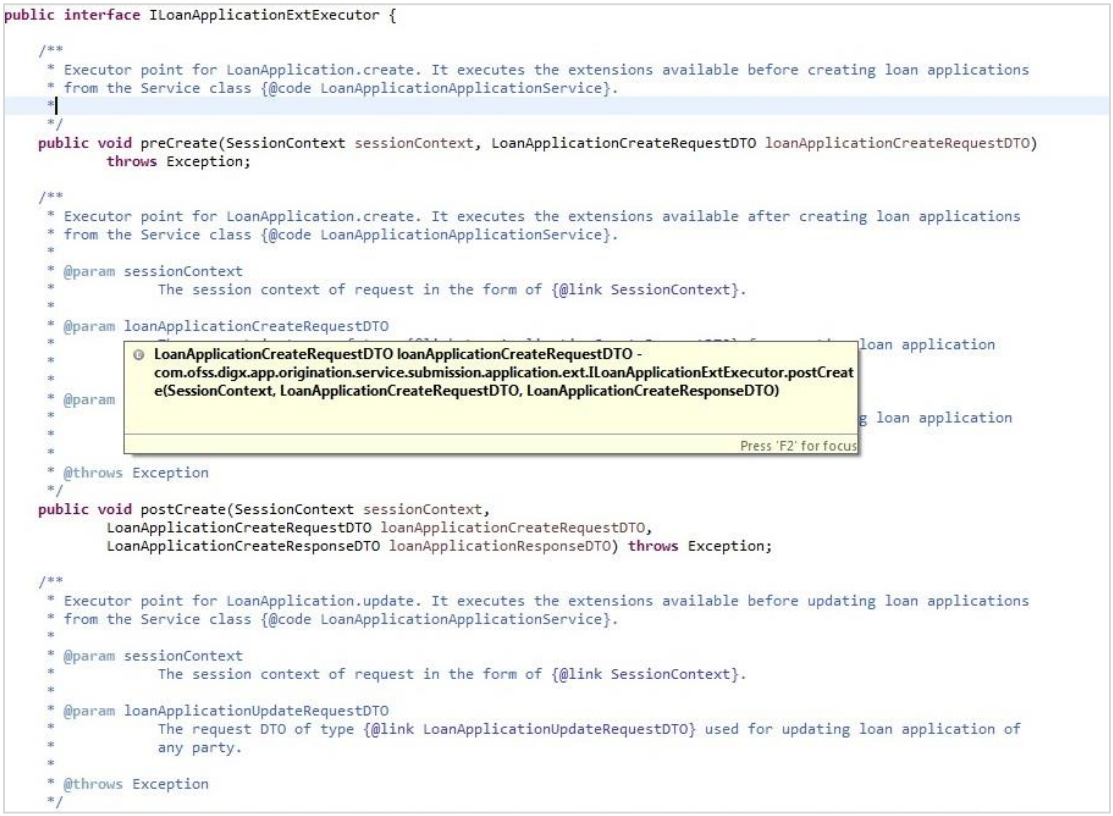
Naming convention

The naming convention for extension executor class is as below:

Interface : I<Service_Name>ExtExecutor

Implementation : <Service_Name>ExtExecutor

For example, consider below code sample:

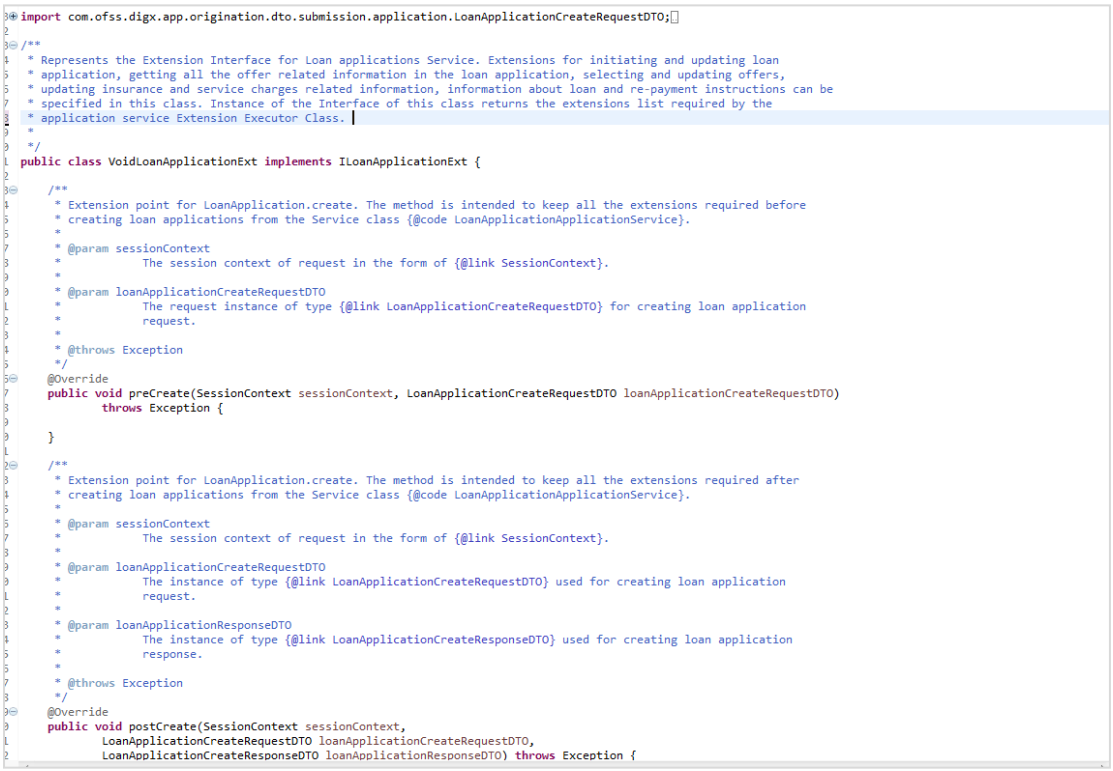


4.2.3 Default Extension (Void Extension)

This class, named as Void<Service_Name>Ext, is provided out of the box for each application service. This class implements the aforementioned service extension interface without any business logic viz. the implemented methods are empty.

The default extension is a useful & convenient mechanism to implement the pre and / or post extension hooks for specific methods of an application service. Instead of implementing the entire interface, one should extend the default extension class and override only required methods with the additional business logic. Product developers do not implement any logic, including product extension logic, inside the default extension classes.

For example:



```
import com.ofss.digx.app.origination.dto.submission.application.LoanApplicationCreateRequestDTO;

/**
 * Represents the Extension Interface for Loan applications Service. Extensions for initiating and updating loan
 * application, getting all the offer related information in the loan application, selecting and updating offers,
 * updating insurance and service charges related information, information about loan and re-payment instructions can be
 * specified in this class. Instance of the Interface of this class returns the extensions list required by the
 * application service Extension Executor Class.
 *
 */
public class VoidLoanApplicationExt implements ILoanApplicationExt {

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *            The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *            The request instance of type {@link LoanApplicationCreateRequestDTO} for creating loan application
     *            request.
     *
     * @throws Exception
     */
    @Override
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
            throws Exception {

    }

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required after
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *            The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *            The instance of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
     *            request.
     *
     * @param loanApplicationResponseDTO
     *            The instance of type {@link LoanApplicationCreateResponseDTO} used for creating loan application
     *            response.
     *
     * @throws Exception
     */
    @Override
    public void postCreate(SessionContext sessionContext,
            LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
            LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception {
```

4.2.4 Custom Extension

Below is an example of customized service extension class that implements methods of application service extension interface. This class contains pre hook and post hook point for the service. The pre method of this customized extension is executed before the actual service method and the post method of this is executed after the service method.

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

/**
 * Holds the instance of {@link MultiEntityLogger} used for sending messages on the console.
 */
private com.ofss.fc.infra.log.impl.MultiEntityLogger formatter = com.ofss.fc.infra.log.impl.MultiEntityLogger
        .getUniqueInstance();
/**
 * Instance of type {@link java.util.logging.Logger} used for Logging in Collaboration service.
 */
private static transient Logger logger = com.ofss.fc.infra.log.impl.MultiEntityLogger.getUniqueInstance()
        .getLogger(THIS_COMPONENT_NAME);

@Override
public void preCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // calling a custom class to check DTO integrity.
    this.checkCollaborationDTO();

    try{
    NameValuePairDTO[] valuePairDTO = new NameValuePairDTO[1];
    valuePairDTO[0] = new NameValuePairDTO("mobileCustomer", "9595959595", "String");
    valuePairDTO[0].setGenericName("com.ofss.digx.domain.collaboration.entity.customdemo.CustomCollaborationDomainObject.mobileCustomer");


    Dictionary[] dictionary = new Dictionary[1];    //array of dictionary
    dictionary[0] = new Dictionary();
    dictionary[0].setNameValuePairDTOArray(valuePairDTO);
    collaborationDTO.setDictionaryArray(dictionary);
    }catch(java.lang.Exception e)
    {
        logger.log(Level.FINE, formatter.formatMessage("Pre-Create extension implementation sample"));
    }

}

private void checkCollaborationDTO() {
    System.out.println("Sample validation performed");
}

@Override
public void postCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO,
        CollaborationResponseDTO collaborationResponseDTO) throws Exception {
    // TODO Auto-generated method stub

}

@Override
public void preRead(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // TODO Auto-generated method stub

}
```

Note: The concept of 'Dictionary' is explained in detail in subsequent section.

4.2.5 Service Extension Configurations

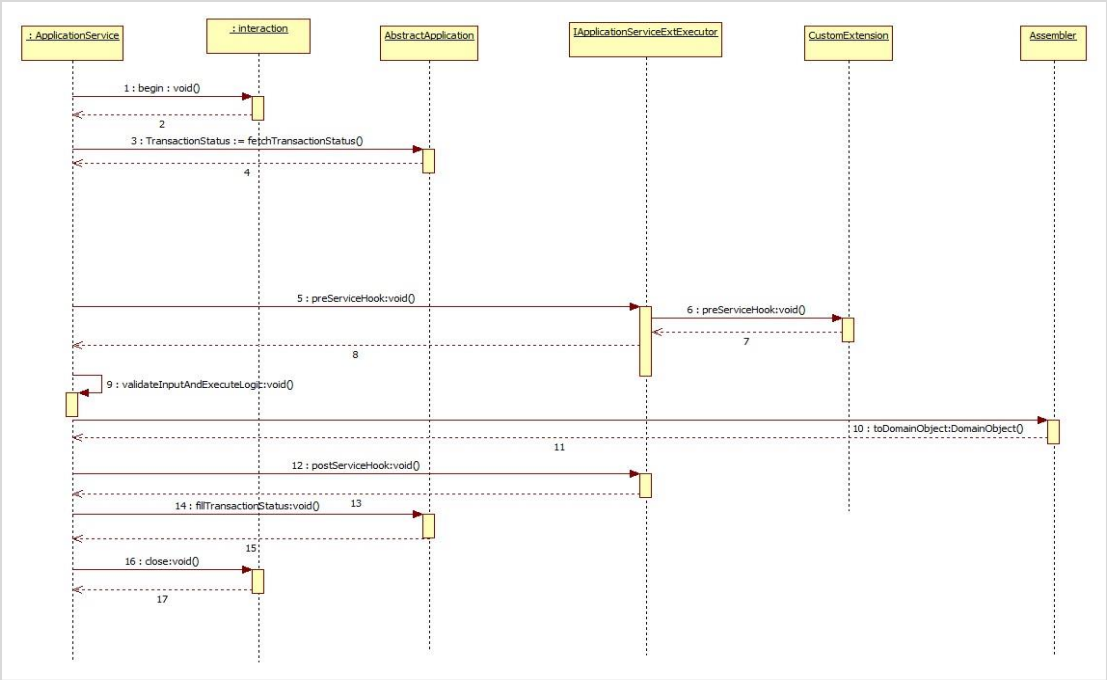
Set the property id and the property values in the digx_fw_config_all_b table. The property id will be the fully qualified name of the service and the value will be the fully qualified name of the custom extension created.

For example:

```
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values ('com.ofss.digx.app.origination.service.submission.applicant.Applicant', 'ServiceExtensionsConfig',
'com.ofss.digx.app.origination.service.submission.application.ext.CustomLoanApplicationExtension', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
```

4.2.6 Sequence of events in service extension

Every application service method has a standard set of framework method calls as shown in the sequence diagram below:



The pre hook is provided after the invocation of *fetchTransactionStatus* call inside the application service. At this step, the current task code is received , any additional manipulation of the input received from the User interface channel can be done in the pre hook. Apart from this additional data coming from the screen specific to client requirements can be handled in the pre hook.

The post hook is provided after the business logic corresponding to the application service invoked has executed and before the successful execution of the entire service is marked in the status object. This ensures that the status marking takes into consideration any execution failures of post hook prior to reporting the result to the calling source. Both, the pre and the post hooks accept the service input parameters as the inputs. The post hook also accepts the Response parameter as the input.

4.3 Business Policy

OBDX supports three types of validations

DTO field validations: These are the field level validations such as syntax check of the input. These validations are achieved by using field level annotations in request DTO. These validations are not available for extension. Below is the list of out of box annotations available

Annotation	Description
@Email	This annotation is used to validate the respective field with email regular- expression. If the field doesn't satisfy the mentioned regular-expression then the respective error code is thrown
@Mandatory	This annotation marks the fields as mandatory. Once marked, if the field is null then respective error-code is thrown Eg. @Mandatory(errorCode = DemandDepositErrorConstants.DDA_MANDATORY_ACCOUNT_ID) private Account accountId;
@Length	This annotation marks the lengths of the fields. Once marked, if the validation is violated then the respective error code is thrown. Eg. @Length(min = 2, max = 20, errorCode = PartyErrorConstants.PI_LENGTH_EXTERNAL_REF_ID)
@NonNegative	This annotation checks that the value is non-negative
@Regex	This annotation checks if the value matches regular expression provided

System Constraints: System performs these checks mandatorily. It is not possible to override or bypass these checks.

Business Policies: These are typically the business validations required to be performed before executing business logic. OBDX framework allows customization developer to override business policies as per the requirement.

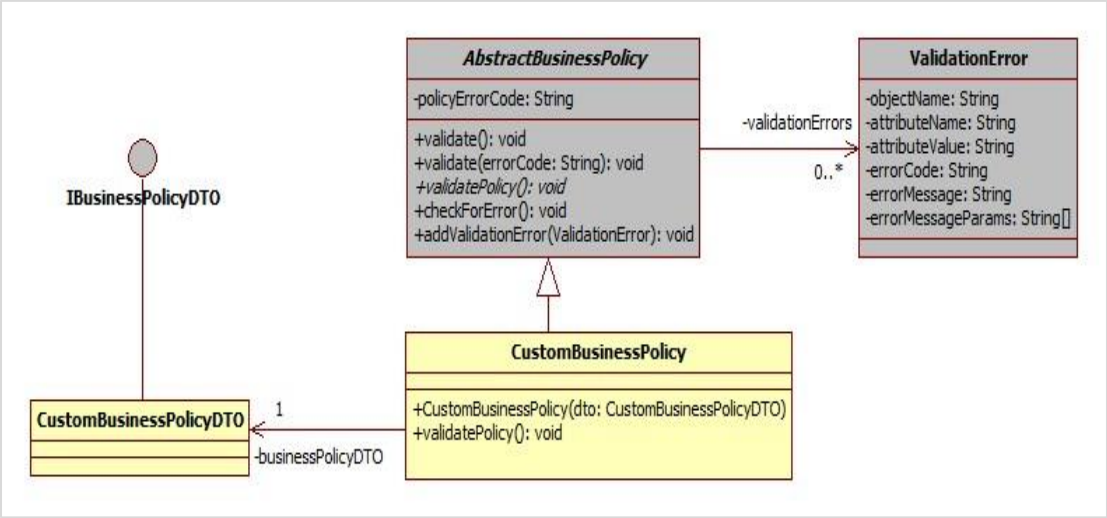
4.3.1 Adding new business policy

Customization developer can add new business policy for new or existing services. Following are the steps to add a new business policy:

1. **Create new business policy DTO.** This DTO is supposed to encapsulate all the input fields upon which validation is to be performed.
2. **Create new business policy.** This class must extend AbstractBusinessPolicy class and implement the validatePolicy() method. This method should have the validation logic and if the validation fails, then it should call addValidationError() method with a new instance of ValidationError as parameter. One of the parameter to the constructor of ValidationError is error code. A new error could be added by following guidelines provided in Error Messages section.
3. **Configure new business policy.** The business policy class created in above step should be configured in digx_fw_config_all_b table with category_id as 'CustomBusinessPolicy' and prop_id as the policy identifier. For example:

```
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values (CREATE_TERM_DEPOSIT_BUSINESS_POLICY ', ' CustomBusinessPolicy ', 'com.ofss.digx.cz.domain.td.entity.policy.CustomBusinessPolicy', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
```

The class diagram for new custom business policy will be like this



4.3.2 Extending existing business policy

OBDX provides out of box business policies for all services. If only a part of the validation is to be modified or a new validation is to be added in addition to the validations that the existing business policy does, then it is possible to extend existing business policy and override existing validation.

Please note that this capability depends on how the original business policy is coded. If the out of box business performs all its validations in validatePolicy() method, then this approach may not be useful. On the other hand, if the out of box business policy has separate individual methods for validations and validatePolicy() method calls these methods one by one, then extension of the business policy is useful.

The steps to be followed as same as mentioned in earlier section, except the difference that the custom business policy class will extend the out of box business policy class and override its methods as per the requirement.

4.4 Dictionary

Dictionary is not an extension point in itself, but it plays an important role in enabling extensibility of domain. Hence, it is worth understanding the 'Dictionary' before proceeding to subsequent sections

Data transfer object (DTO)

Data transfer object (DTO) is a design pattern used to transfer data between an external system and the application service. All the information may be wrapped in a single DTO containing all the details and passed as input request as well as returned as an output response. The client can then invoke accessory (or getter) methods on the DTO to get the individual attribute values from the Transfer Object. All request response classes in OBDX application services are modelled as data transfer objects.

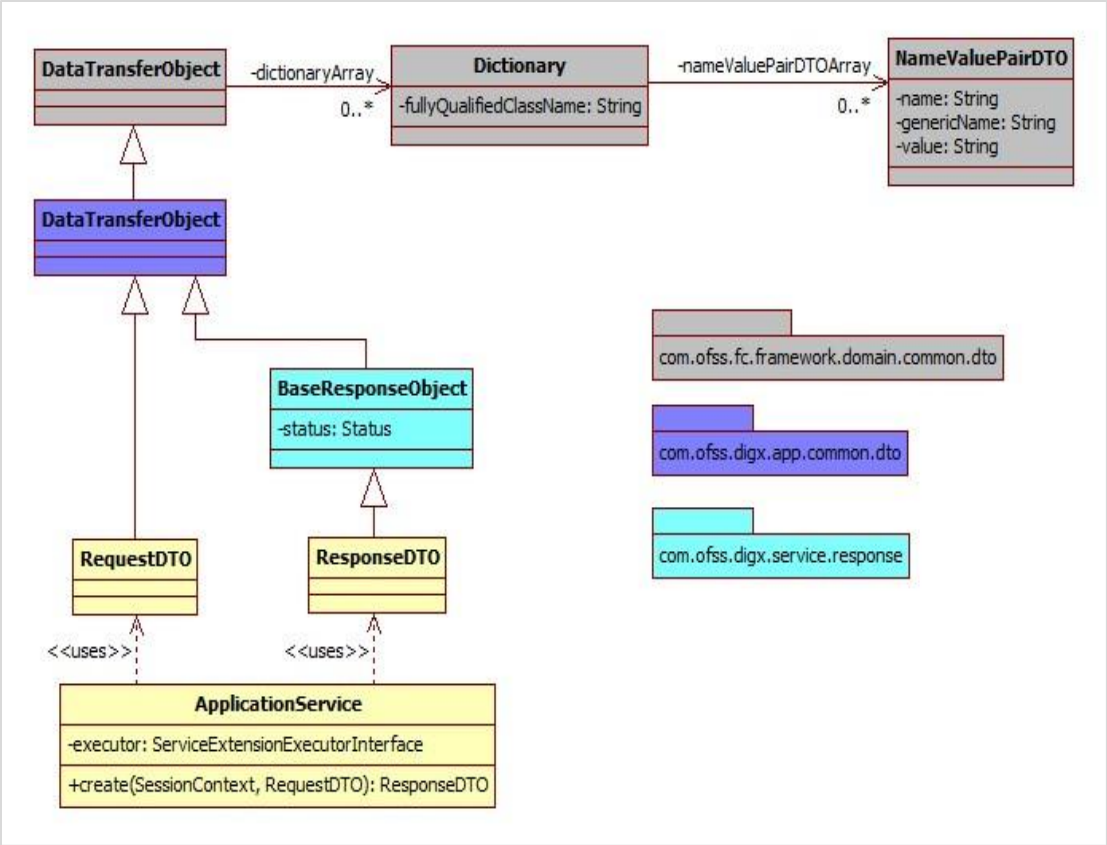
The NameValuePairDTO object contain following fields: (Type of each field is 'String')

1. name : Refers to the name of the custom data field. E.g. mobileCustomer
2. genericName : This field contains the fully qualified name of the custom data field including class hierarchy. E.g. com.ofss.digx.<module_hierarchy>.MobileCustomer
3. value : This field contains the value of the custom data field. E.g. 1111111111

```
public abstract class DataTransferObject extends Validatable implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = -6584908885732656582L;
    /**
     * Subclasses of the Customized AbstractDomainObject corresponding to this
     * AbstractDomainObject<br>
     * are defined with the help of this attribute. This concept can be extended
     * to have joined or<br>
     * union subclass heirarchy in subsequent releases.
     */
    private Dictionary[] dictionaryArray;

    /**
     * Returns data for subclasses of the Customized Domain Object as name value
     * pair data with the<br>
     * name being a fact.
     *
     * @return
     */
    public Dictionary[] getDictionaryArray() {
        return dictionaryArray;
    }

    public void setDictionaryArray(Dictionary[] dictionaryArray) {
        this.dictionaryArray = dictionaryArray;
    }

}
```

Dictionary

All data transfer objects extend a base class ***DataTransferObject*** which holds an array of Dictionary object. The Dictionary encapsulates an array of ***NameValuePairDTO*** which is used to pass data of custom data fields or attributes from the UI layer to the host middleware. Below class diagram shows the relationship between these classes


DataTransferObject
-dictionaryArray
0..*
Dictionary
-fullyQualifiedClassName: String
-nameValuePairDTOArray
0..*
NameValuePairDTO
-name: String
-genericName: String
-value: String
DataTransferObject
BaseResponseObject
-status: Status
RequestDTO
ResponseDTO
<<uses>>
<<uses>>
ApplicationService
-executor: ServiceExtensionExecutorInterface
+create(SessionContext, RequestDTO): ResponseDTO
com.ofss.fc.framework.domain.common.dto
com.ofss.digx.app.common.dto
com.ofss.digx.service.response

Dictionary class looks like

Dictionary.class CustomLoanApplicationExtension.java

```
1  package com.ofss.fc.framework.domain.common.dto;
2
3  import java.io.Serializable;
4
5  import javax.xml.bind.annotation.XmlType;
6
7  @XmlType(namespace="http://dto.common.domain.framework.fc.ofss.com")
8  public class Dictionary  implements Serializable {
9
10     /**
11      *
12      */
13     private static final long serialVersionUID = 640766746819694755L;
14     private NameValuePairDTO[] nameValuePairDTOArray;
15
16     public NameValuePairDTO[] getNameValuePairDTOArray() {
17         return nameValuePairDTOArray;
18     }
19
20     public void setNameValuePairDTOArray(NameValuePairDTO[] nameValuePairDTOArray) {
21         this.nameValuePairDTOArray = nameValuePairDTOArray;
22     }
23
24
25 }
26
```

Following image shows use of dictionary with NameValuePairDTO and added it to the Data Transfer Object.

```
@Override
public void preCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // calling a custom class to check DTO integrity.
    this.checkCollaborationDTO();

    try{
    NameValuePairDTO[] valuePairDTO = new NameValuePairDTO[1];
    valuePairDTO[0] = new NameValuePairDTO("mobileCustomer", "9595959595", "String");
    valuePairDTO[0].setGenericName("com.ofss.digx.domain.collaboration.entity.customdemo.CustomCollaborationDomainObject.mobileCustomer");


    Dictionary[] dictionary = new Dictionary[1];    //array of dictionary
    dictionary[0] = new Dictionary();
    dictionary[0].setNameValuePairDTOArray(valuePairDTO);
    collaborationDTO.setDictionaryArray(dictionary);
    }catch(java.lang.Exception e)
    {
        logger.log(Level.FINE, formatter.formatMessage("Pre-Create extension implementation sample"));
    }

}
```

4.5 Domain Extensions

The Domain layer is a central layer in designing entities in OBDX. The design philosophy is called domain driven design. In this, the domain object (also referred as 'entity' in OBDX context) is central to the design. The domain captures all attributes of the real time entity that it models.

OBDX provides infrastructure to customize existing domains. It also allows to add new domains.

4.5.1 Custom Domain Objects

OBDX framework (leveraging undelaying OBP infrastructure) provides a standard mechanism to customize the domain objects that are provided out of the box. The Dictionary object plays an important role in this mechanism.

This section describes how consultants or other third parties can extend domain and achieve Extensibility. This provides true domain model extension capabilities by allowing addition of custom data fields to the underlying domain objects.

Translating Dictionary data into custom domain object

If dictionary is added to DTO then it is necessary to get customized domain Object which extends base Domain Object. Method getCustomizedDomainObject in AbstractAssembler is used for the same.

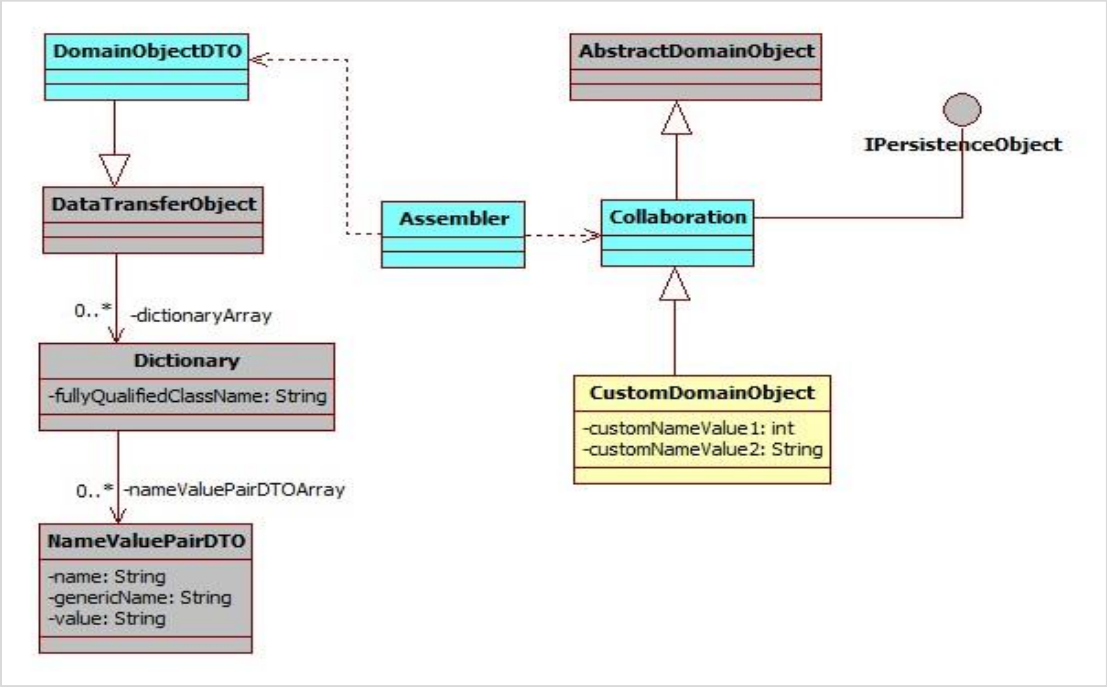
Following image shows call to get Customized domain Object if additional data (Dictionary) is added to the request DTO.

```
public Collaboration toDomainObject(CollaborationDTO collaborationDTO) {

    Collaboration collaboration;
    if (collaborationDTO.getDictionaryArray() != null) {
        try {
            collaboration = (Collaboration) getCustomizedDomainObject(collaborationDTO);
        } catch (java.lang.Exception e) {
            collaboration = new Collaboration();
        }
    } else {
        collaboration = new Collaboration();
    }

    CollaborationKey collaborationKey = new CollaborationKey();
    if (collaborationDTO.getId() != null) {
        collaborationKey.setId(collaborationDTO.getId());
    }
    collaboration.setExternalRefId(collaborationDTO.getExternalRefId());
    collaboration.setAuthorId(collaborationDTO.getAuthorId());
    collaboration.setAuthorName(collaborationDTO.getAuthorName());
    collaboration.setUrl(collaborationDTO.getUrl());
    collaboration.setExpiryDate(collaborationDTO.getExpiryDate());
    collaboration.setNoOfParticipants(collaborationDTO.getNoOfParticipants());
    collaboration.setCollaborationChannel(collaborationDTO.getCollaborationChannel());
    List<Participant> participantsList = new ArrayList<Participant>();
    if (collaborationDTO.getParticipants() != null && !collaborationDTO.getParticipants().isEmpty()) {
        for (ParticipantDTO participantDTO : collaborationDTO.getParticipants()) {
            Participant participant = new Participant();
            ParticipantKey participantKey = new ParticipantKey();
            participant.setKey(participantKey);
            participant.setInternalPartyId(participantDTO.getInternalPartyId());
            participant.setName(participantDTO.getName());
            participant.setPublishingId(participantDTO.getPublishingId());
            participant.setIsPublished(participantDTO.getIsPublished());
            participantsList.add(participant);
        }
        collaboration.setParticipants(participantsList);
    }
    collaboration.setCollaborationKey(collaborationKey);
    return collaboration;
}
```

Writing Custom Domain Object

The custom domain object must extend existing domain object class. Mapping for same should be done in database as Customized Abstract Domain Object Configuration. This class contains additional fields added at UI layer and getter, setter for the same.

Below diagram shows the custom domain object and also depicts the role of Dictionary in mapping additional fields from DTO to this custom domain object.



For Example:

```java
package com.ofss.digx.domain.collaboration.entity.customdemo;

import com.ofss.digx.domain.collaboration.entity.Collaboration;

/**
 * custom domain object to handle the changes ate UI or pre hoot level
 */
public class CustomCollaborationDomainObject extends Collaboration {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    /**
     * store the mobile number to call on.
     */
    private String mobileCustomer;

    /**
     * @return the mobileCustomer
     */
    public String getMobileCustomer() {
        return mobileCustomer;
    }

    /**
     * @param mobileCustomer
     *            the mobileCustomer to set
     */
    public void setMobileCustomer(String mobileCustomer) {
        this.mobileCustomer = mobileCustomer;
    }

}
```

Configure Customized domain object in database

The domain object created needs to be mapped as a custom domain object for the existing domain object. For example:

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values ('com.ofss.digx.domain.origination.entity.submission.lending.application', 'CustomizedAbstractDomainObjectConfig', 'com.ofss.digx.domain.origination.entity.submission.lending.application.ext.Application', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);

Three main columns that need to be fed with new information are.

- CATEGORY_ID : “CustomizedAbstractDomainObjectConfig”
- PROP_VALUE:” CLASS NAME of the class implementing the custom domain object ”
- PROP_ID:” CLASS NAME of the DomainObject”.

ORM Mapping

If this domain needs to be persisted in local database, then you need to create Eclipselink ORM mapping to map fields in the domain to database table. Follow these steps:

- We are going to use Eclipselink JPA inheritance strategy.

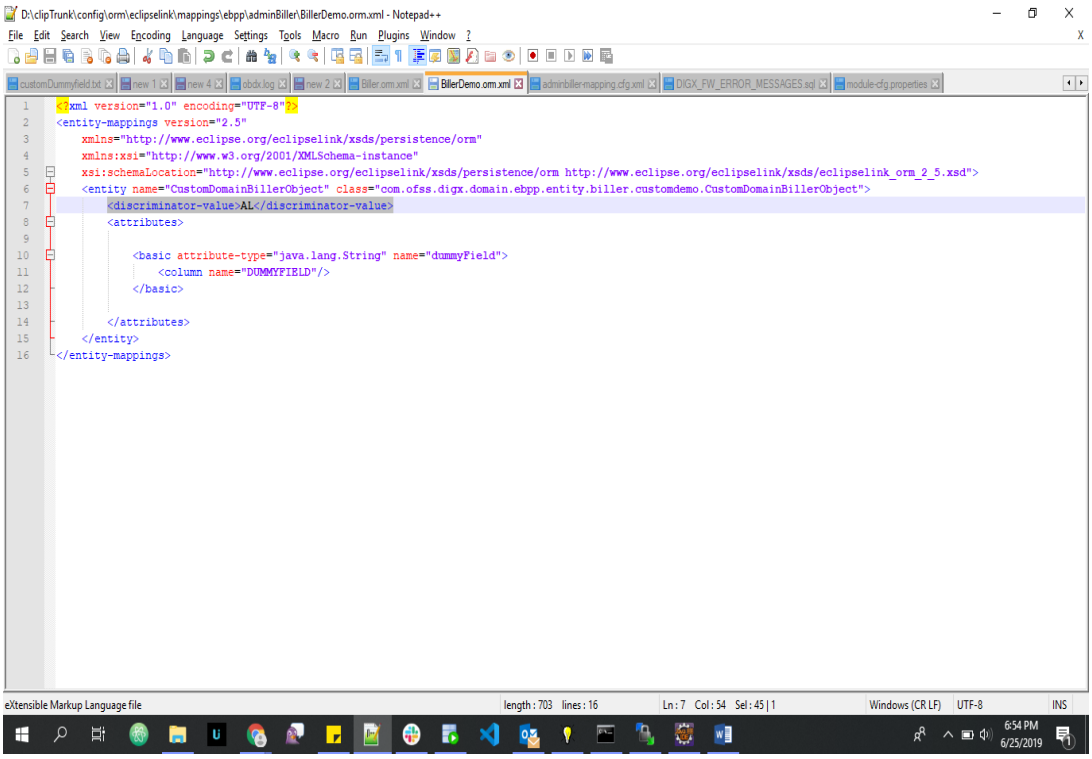
Refer to the below url to implement inheritance with Eclipselink JPA

https://wiki.eclipse.org/EclipseLink/Examples/JPA/Inheritance

You can implement it using either Single Table inheritance or Joined Table inheritance approach.

Here we will see how to implement it with **Single table inheritance**:

Create a new ORM file to handle Customized Domain Object. This ORM file should contain entries **only** for those corresponding new fields, which you want to add.

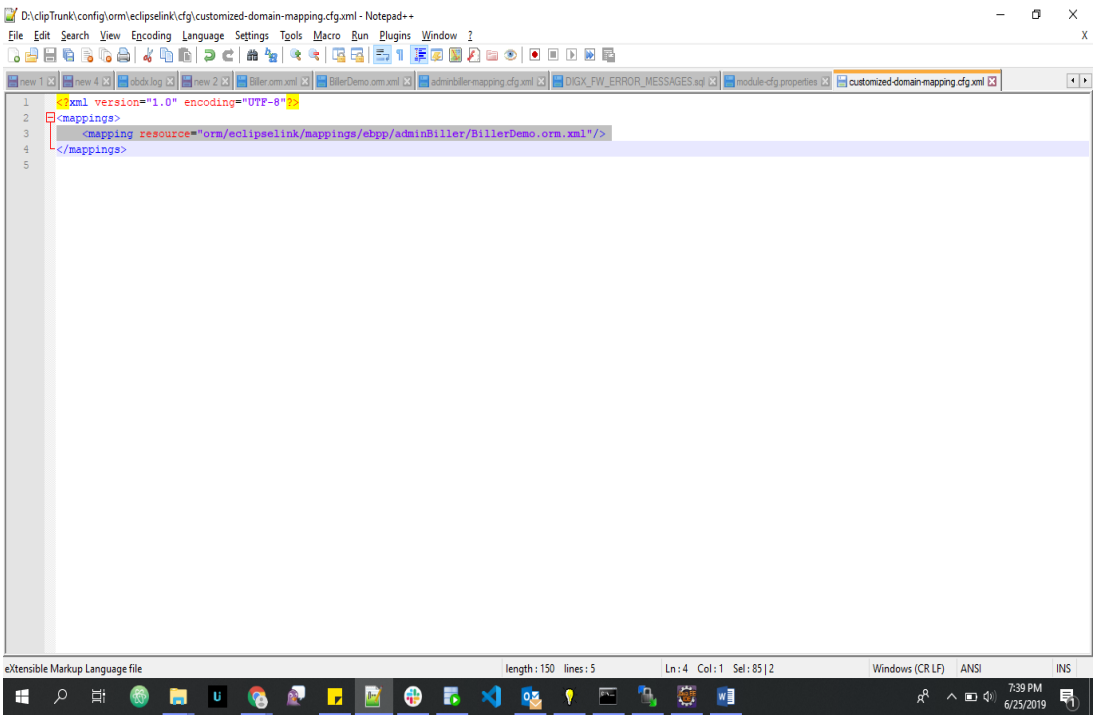


In above screenshot, you can see that BillerDemo.orm.xml is new ORM file to handle Customized Domain object. Also you will have to add discriminator value as shown below

(see highlighted portion in above screenshot)

<discriminator-value>AL</discriminator-value>

- Add this child ORM (newly created BillerDemo.orm.xml) file's entry in a configuration file named **customized-domain-mapping.cfg.xml**



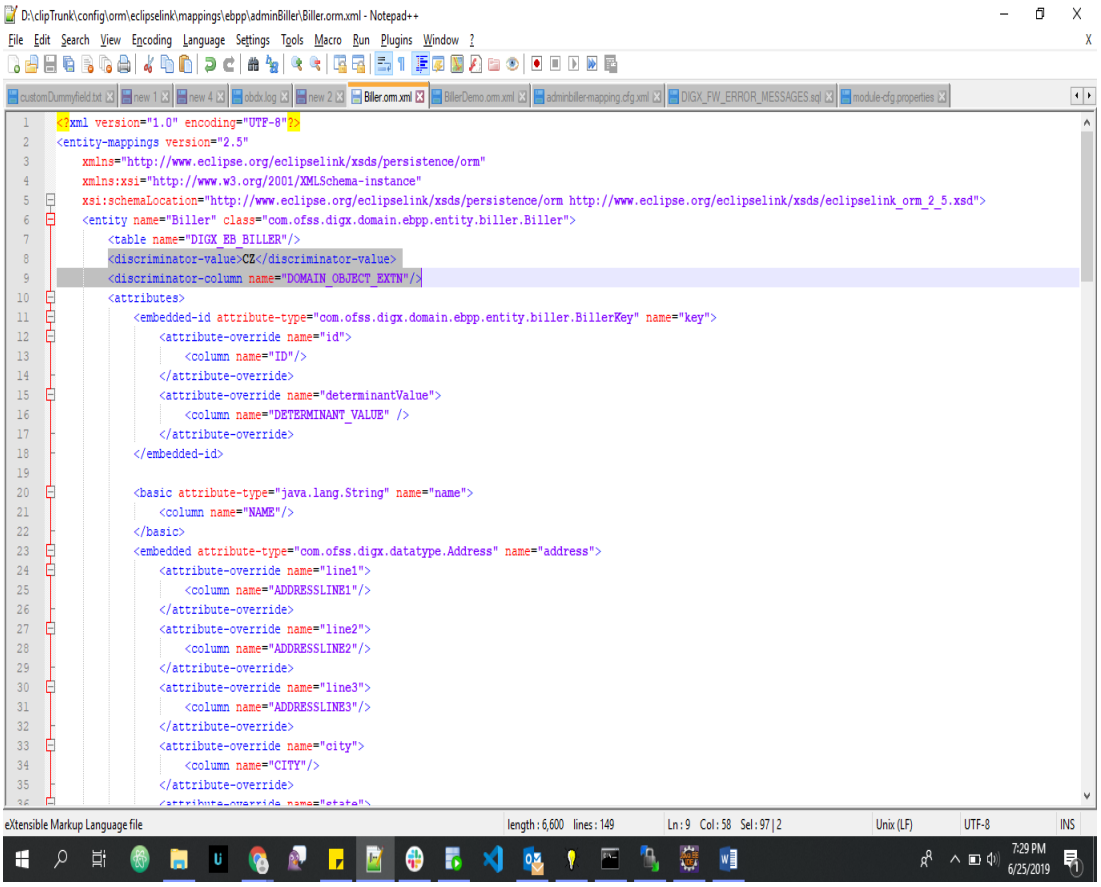
- Add few changes in parent ORM which maps to parent domain (i.e Biller.orm.xml) which is extended by customized domain object.

  Add below properties in your parent ORM as shown

  <discriminator-column discriminator-type="STRING" name="DOMAIN_OBJECT_EXTN"/>

  <discriminator-value>CZ</discriminator-value>

Note: Add this discriminator property in ORM **if and only if it already does not exists**.



- If the custom domain object is extending a domain which already has a JPA inheritance strategy applied directly in the parent domain or in some domain up the hierarchy then you have 2 options

    a. Either follow the same inheritance strategy in the customized child domain as well.

    b. Or define the inheritance strategy in the parent domain's orm mapping xml file on the basis of your requirement.

    The JPA provides three type of Inheritance Strategies:

    c. **SINGLE_TABLE** (Default).

    d. **TABLE_PER_CLASS**

    e. **JOINED**

    Out of these JOINED is the preferred strategy as it does not alter the DB table structure of the parent domain.

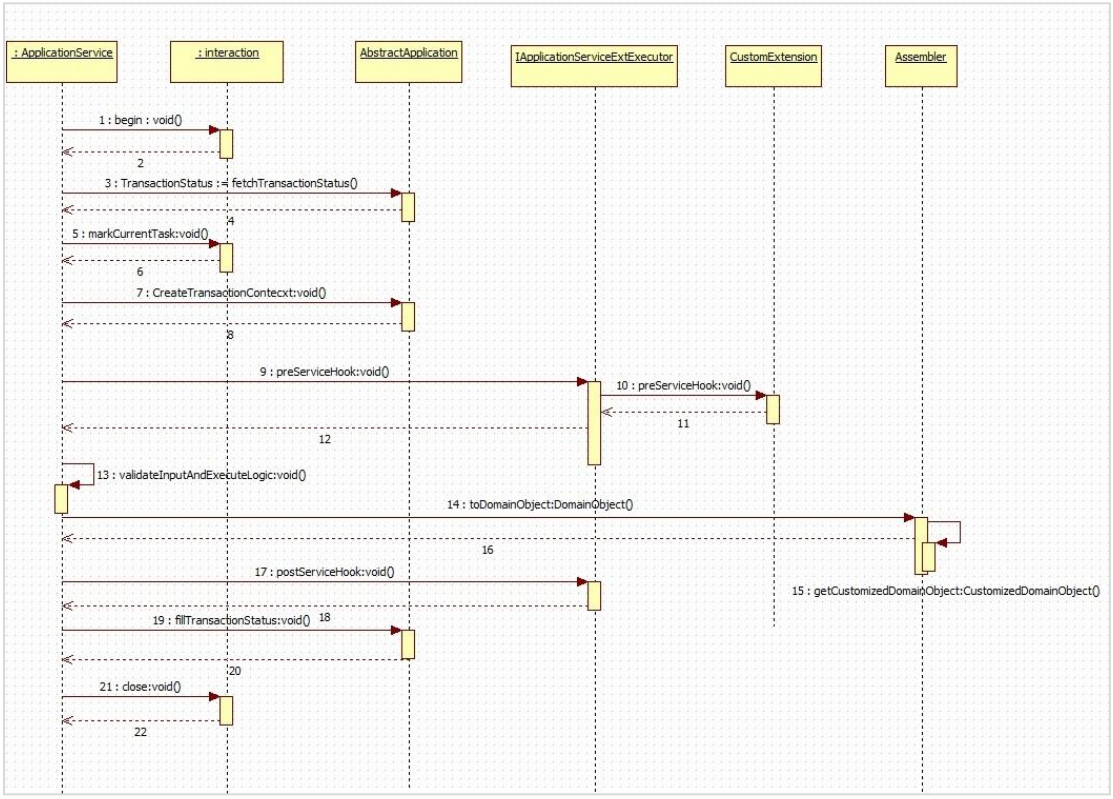
- Add new columns in parent domain **Table** for newly added fields in customized domain object and also add new column as DOMAIN_OBJECT_EXTN (This column is used to distinguish to which class type a database row belongs)

Here Assembler should fetch customized domain object. Following example shows Assembler calls **getCustomizedDomainObject** which returns customized domain object with mapping of **nameValuePairDTOArra**y to this customized domain Object internally.

For example:

```
public Collaboration toDomainObject(CollaborationDTO collaborationDTO) {

    Collaboration collaboration;
    if (collaborationDTO.getDictionaryArray() != null) {
        try {
            collaboration = (Collaboration) getCustomizedDomainObject(collaborationDTO);
        } catch (java.lang.Exception e) {
            collaboration = new Collaboration();
        }
    } else {
        collaboration = new Collaboration();
    }

    CollaborationKey collaborationKey = new CollaborationKey();
    if (collaborationDTO.getId() != null) {
        collaborationKey.setId(collaborationDTO.getId());
    }
    collaboration.setExternalRefId(collaborationDTO.getExternalRefId());
    collaboration.setAuthorId(collaborationDTO.getAuthorId());
    collaboration.setAuthorName(collaborationDTO.getAuthorName());
    collaboration.setUrl(collaborationDTO.getUrl());
    collaboration.setExpiryDate(collaborationDTO.getExpiryDate());
    collaboration.setNoOfParticipants(collaborationDTO.getNoOfParticipants());
    collaboration.setCollaborationChannel(collaborationDTO.getCollaborationChannel());
    List<Participant> participantsList = new ArrayList<Participant>();
    if (collaborationDTO.getParticipants() != null && !collaborationDTO.getParticipants().isEmpty()) {
        for (ParticipantDTO participantDTO : collaborationDTO.getParticipants()) {
            Participant participant = new Participant();
            ParticipantKey participantKey = new ParticipantKey();
            participant.setKey(participantKey);
            participant.setInternalPartyId(participantDTO.getInternalPartyId());
            participant.setName(participantDTO.getName());
            participant.setPublishingId(participantDTO.getPublishingId());
            participant.setIsPublished(participantDTO.getIsPublished());
            participantsList.add(participant);
        }
        collaboration.setParticipants(participantsList);
    }
    collaboration.setCollaborationKey(collaborationKey);
    return collaboration;
}
```

Sequence Diagram



Configuring this custom domain object at appropriate entity level

insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED)

values ('<Fully qualified domain name>', '<Determinant Type>', '<Represented Name>', 'Y');

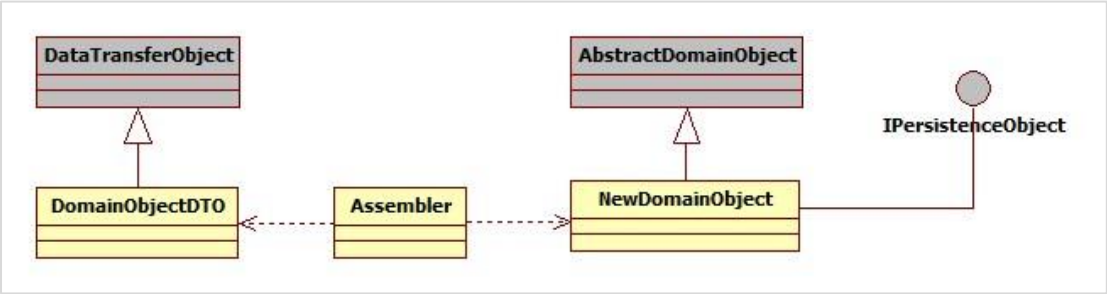
There are four possible determinant types as follows:

- Enterprise (ENT)
- Legal Entity (LGE)
- Market Entity (MKE)
- Business Unit (BNU)

4.5.2 Adding new Domain

The customization developer can add new domain. Below are the steps to add a new domain.

1. Create new domain class. The new domain class must extend AbstractDomainObject and implement IPersistenceObject
2. Identify attributes and operations supported by the domain and add them to above domain class accordingly
3. The domain object will typically have associated DTO that encapsulates same fields as in domain. This DTO will be used in request and responses. An assembler will be used to map fields between domain object and the DTO. Below diagram depicts this relationship.



4. Configure this new domain for appropriate entity level

insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED)

values ('<Fully qualified domain name>', '<Determinant Type>', '<Represented Name>', 'Y');

For example,

insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED)

values (' com.ofss.digx.cz.domain.payment.entity.payee.Payee', 'BNU', 'New Payee', 'Y');

4.6 Error Messages

If an API fails, It returns an error code and an error message which briefly specifies the failure reason of the API call. Error message is returned from service to convey the cause of transaction failure.

4.6.1 Adding Error Message

Error codes with their error messages are stored in DIGX_FW_ERROR_MESSAGES table. One can add a new error message in the table with a unique error code.

ERROR_CODE column should contain unique value.

ERROR_MESSAGE column contains the error message which need to be added.

4.6.2 Mapping Host Error Code To OBDX Error Code

When a transaction fails in host, it provides an error code in response to the failed transaction. This error code provided by the host could be mapped with OBDX error code to provide a user friendly error message.

This host error code and OBDX error code mapping is done in DIGX_FW_ERR_COD_MAP table.

THIRD_PARTY_ERR_COD column holds the host error code.

LOCAL_ERR_COD column holds OBDX error code which must be present in DIGX_FW_ERROR_MESSAGES table from where error message will be picked.

4.7 Adapter Tier

An adapter, by definition, helps the interfacing or integrating components adapt. In software it represents a coding discipline that helps two different modules or systems to communicate with each other and helps the consuming side adapt to any incompatibility of the invoked interface work together. Incompatibility could be in the form of input data elements which the consumer does not have and hence might require defaulting or the invoked interface might be a third party interface with a different message format requiring message translation. Such functions, which do not form part of the consumer functionality, can be implemented in the adapter layer.

4.7.1 Service Provider Interface (SPI) Approach

This section provides information about the SPI approach and how adapters are packaged and derived at runtime based on current entity and domain under consideration.

Service Provider Interface (SPI) is an API intended to be implemented or extended by a third party. It can be used to enable framework extension and replaceable components.

- https://docs.oracle.com/javase/tutorial/ext/basics/spi.html
- http://www.developer.com/java/article.php/3848881/Service-Provider-Interface-Creating-Extensible-Java-Applications.htm

All the external facing adapters will be loaded using SPI.

Benefits of SPI

- No database entries are required.
- No need of adapter factories.
- Can add adapters at run-time.
- Provides the list of available implementations from which we can use the best suited one.

In this approach adapter is selected using the following call.

ExtxfaceAdapterFactory.getInstance().getAdapter(Interface.class, "method", DeterminantType);

Here,

- ‘Interface.class’ is object of interface implemented by the host (external system) adapter.
- ‘Method’ is name of method which we are intended to call of that adapter.
- DeterminantType is determinant type of the domain from which this call is made.

Sample code is as follows:

```
/**
 * Used to insert/persist LoanApplication Object into database
 *
 * @param object
 *            LoanApplication Object to be persisted
 * @return LoanApplication object that was created.
 * @throws com.ofss.digx.infra.exceptions.Exception
 *
 * @see com.ofss.digx.domain.origination.entity.submission.lending.application.repository.adapter.
 *      ILoanApplicationRepositoryAdapter
 *      #createLoanApplication(com.ofss.digx.domain.origination.entity.submission.lending.application.LoanApplication)
 */
@Override
public LoanApplication createLoanApplication(LoanApplication object) throws Exception {
    ILoanApplicationRequirementAdapter adapter = ExtxfaceAdapterFactory.getInstance().getAdapter(
            ILoanApplicationRequirementAdapter.class, "create",
            DeterminantResolver.getInstance().getDeterminantTypeForObject(LoanApplication.class.getName()));
    LoanApplicationAssembler assembler = new LoanApplicationAssembler();
    LoanApplicationCreateResponseDTO response = adapter.create(object.getKey().getSubmissionId(), assembler.fromDomainObject(object));
    return assembler.toDomainObject(response.getLoanApplicationRequirementDTO());
}
```

Adapter configuration:

For adapter configurations, the preference **ExtxfaceAdapterPreference** is used. This preference contains Entity as key and External System (Host Name + Version) as value. So we can use select external systems (Hosts) on the basis of entity. E.g. For entity 000 we want to use UBS 12.4 and for entity 001 use OBP 2502 then the entries will be

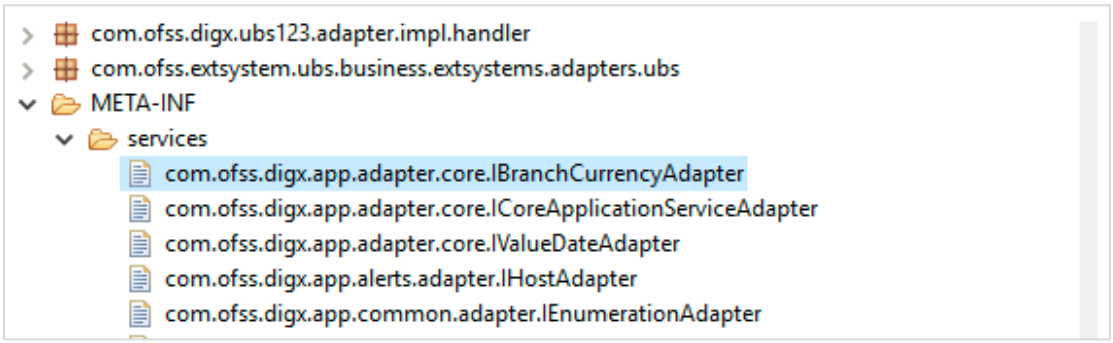
	PROP_ID	CATEGORY_ID	PROP_VALUE	FACTORY_SHIPPED_FLAG	PROP_COMMENTS
1	000	extxfaceadapterconfig	UBS12.4	N	(null)
2	001	extxfaceadapterconfig	OBP2502	N	(null)

We can also give multiple External System separated by comma “,” for an entity, and then adapter will get selected on the basis sequences of external systems given in value.

E.g. if the value is UBS12.4,BI1.0 then first implementation is searched in UBS 12.4 jar if is not found then it will look in BI1.0 jar.

Adapter Registration:

After adding adapter java file in project it need to be register as provider. To register your service provider, create a provider configuration file, which is stored in the META-INF/services directory of the project. The name of the configuration file is the fully qualified class name of the service provider(interface implemented by adapter), and file content which is fully qualified name of the adapter class.



How will system derive adapter?

In the external system interface implementation project like (com.ofss.digx.extxface.ubs124.impl), inside **src/META-INF** folder, we will have a **MANIFEST.MF** file inside which we will define the following attributes:-

Implementation-Title: UBS

Implementation-Version: 12.4

It will tell us that the adapters are for external system UBS 12.4. While adding a new interface implementation project, we need to create MANIFEST.MF file too, defining implementation title and version.

While calling an adapter, we provide three parameters 1. Interface class name 2.method name 3.determinant type(for particular domain class).

Determinant type for particular domain class (digx_me_entity_determinant_b).

	DOMAIN_OBJECT_NAME	DETERMINANT_TYPE	REPRESENTED_FIELD	IS_FEATURE_ENABLED
1	com.ofss.digx.domain.payment.entity.merchant.Merchant	LGE	Merchant Maintenance	Y
2	com.ofss.digx.domain.collaboration.entity.mailbox.usergroupsubjectmap	LGE	User Group Subject Map	Y
3	com.ofss.digx.domain.config.entity.ConfigVarBDomain	BNU	ConfigVarB Details	Y
4	com.ofss.digx.domain.report.entity.ReportAdhocRequest	LGE	Report Request	Y
5	com.ofss.digx.domain.fileupload.entity.FileIdentifierRegistration	BNU	FI Registration	Y
6	com.ofss.digx.domain.fileupload.entity.FileDetails	BNU	Uploaded File Details	Y

We match determinant type to market entity, then business unit and then legal entity.

On the first match, we derive the external systems using ExtxfaceAdapterPreference explained above. Then we derive external systems corresponding to others(lower order ones). Thus we have a list(list 1) of external systems in order.

For example, if 1st match is market entity. Then we will have external systems corresponding to entries for market entity, then business unit and finally legal entity if entries are found.(in order).

If 1st match is business unit, then we will have external systems corresponding to entries for business unit and legal entity if found(in order).

Here in the diagram above, for domain class ConfigVarBDomain, determinant_type is BNU(business unit). lets suppose corresponding determinant value is 000.

	PROP_ID	CATEGORY_ID	PROP_VALUE	FACTORY_SHIPPED_FLAG	PROP_COMMENTS	SUMMARY...	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATED_DATE
1	000	extxfaceadapterconfig	UBS12.3,ipm1.0	N	(null)	(null)	ofsuser	26-SEP-17 06.42.09.000000000 AM	ofsuser	26-SEP-17 06.42.09.000000000 AM
2	001	extxfaceadapterconfig	TP1.0	N	(null)	(null)	Filter...	26-SEP-17 06.42.09.000000000 AM	ofsuser	26-SEP-17 06.42.09.000000000 AM
3	01	extxfaceadapterconfig	UBS12.4,ipm1.0	N	(null)	(null)	ofsuser	20-SEP-17 11.31.25.000000000 AM	ofsuser	20-SEP-17 11.31.25.000000000 AM

Now, for prop_id=000, it will fetch extsystems as UBS12.3,ipm1.0.

Now for legal entity(LGE), lets suppose corresponding determinant value is 001. so it will fetch external system as TP1.0.

So we have external system list (list 1) as {UBS12.3,imp1.0,TP1.0};

Also If none matches, we derive external system corresponding to enterprise. for eg. for enterprise, lets suppose corresponding determinant value as 01. so external system list(list 1) will be {UBS12.4,ipm1.0}.

How the adapters are loaded:

Now we will load all those adapter classes, that will implement the interface whcih we get as first parameter. Now we will maintain another list or map (list 2) of external systems to adapter, that

we will resolve from all those adapter classes. (How will system know that a adapter belongs to which external system or host?).

We will iterate through list 1(list of external systems that we got from preference entry) in order. When we find the first matching external system in list 2, we will return the corresponding adapter.

For example, we iterate through list 1 : {UBS12.3,imp1.0,TP1.0}. it will first find if loaded adapter class contains adapter that belongs to external system UBS12.3. then it will retun that adapter. if not found, it will search if any loaded adapter class belongs to imp1.0. if found it will return that adapter. if not, then it will similarly go for TP1.0.

How to override an adapter?

One can enter (interface class name + "."+ method name or only interface class name) in ExtxfaceAdapterPreference against which one can specify the adapter that one want to be overriden by.

E.g.

Insert into digx_fw_config_all_b
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEXT,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)

values (<Fully qualified adapter interface name>,'extxfaceadapterconfig', <Fully qualified adapter implementation name>,'N',null,'','ofssuser', sysdate,'ofssuser', sysdate,'Y',1,'N',null);

sample:-

Insert into digx_fw_config_all_b
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEXT,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)

values ('com.ofss.digx.app.loan.adapter. ILoanAccountAdapter','extxfaceadapterconfig',
'com.ofss.digx.extxface.loan.impl.LoanAccountMockAdapter','N',null,'','ofssuser',
sysdate,'ofssuser', sysdate,'Y',1,'N',null);

4.7.2 Adding a custom adapter

Please follow below steps for adding a new custom adapter:

- Create a new project for customized adapter interfaces. Typically, there will be only one customized adapter interfaces project. The name of the project should have the phrase ‘cz’ indicating that it is customized version. For example, com.ofss.digx.cz.extxface
- Please refer to the ‘Workspace Setup’ section and its ‘Adapter Interfaces’ subsection for details.

- Add required adapter interfaces in this project
- Create another new project for customized adapter implementation classes. Typically, one project will need to be created per entity, however if the core banking host is same for different entities, then one project can be used for multiple entities. This decision should be taken based on implementation scenario. If you are interfacing with any other external system apart from core banking system (e.g. content management system), then separate project should be created for adapters interfacing with such systems.
- Please refer to the 'Workspace Setup' section and its 'Adapter Implementation' subsection for details.
- Name of the project should be having the phrase 'cz' indicating that it is part of the customization. The name should also include external system name and version. This will bring clarity about contents of the project by looking at the name. The same name will be used for the JAR packaged out of this project. For example, name of the project for customized adapters for UBS 12.4 will be com.ofss.digx.cz.extxface.ubs124.impl
- The MANIFEST.MF file within this project should have implementation title and implementation version. The implementation title should also capture the phrase 'CZ' to indicate that it is a customized adapter package.

  Implementation-Title: CZUBS

  Implementation-Version: 12.4

- Write required adapter implementation classes that implement appropriate adapter interface
- Create folder 'META-INF/services' under the 'src' folder.
- Create a file under this 'services' folder with the name as fully qualified name of the adapter interface.
- In this file, write the fully qualified name of the adapter implementation class
- Package the adapter interface in JAR
- Package the adapter implementation project(s) in JAR(s)
- Configure the adapter implementation package in digx_fw_config_all_b. The prop_value should have comma separated external system IDs. For example,

  ```
  Insert into digx_fw_config_all_b
  (PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEXT,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)

  values ('01','extxfaceadapterconfig', 'CZUBS12.4,UBS12.4,ipm1.0','N',null,'','ofssuser',
  sysdate,'ofssuser', sysdate,'Y',1,'N',null);
  ```

- Package all customized adapters in obdx.cz.extsystem.domain.ear and deploy it as a library

Customizing existing adapters (Custom Adapter)

If an added functionality or replacement functionality is required for an existing adapter or existing method in an adapter, the customization developer has to develop a new adapter and corresponding adapter factory and override the method in a new custom adapter class. The custom adapter would have to override and implement the methods which need changes.

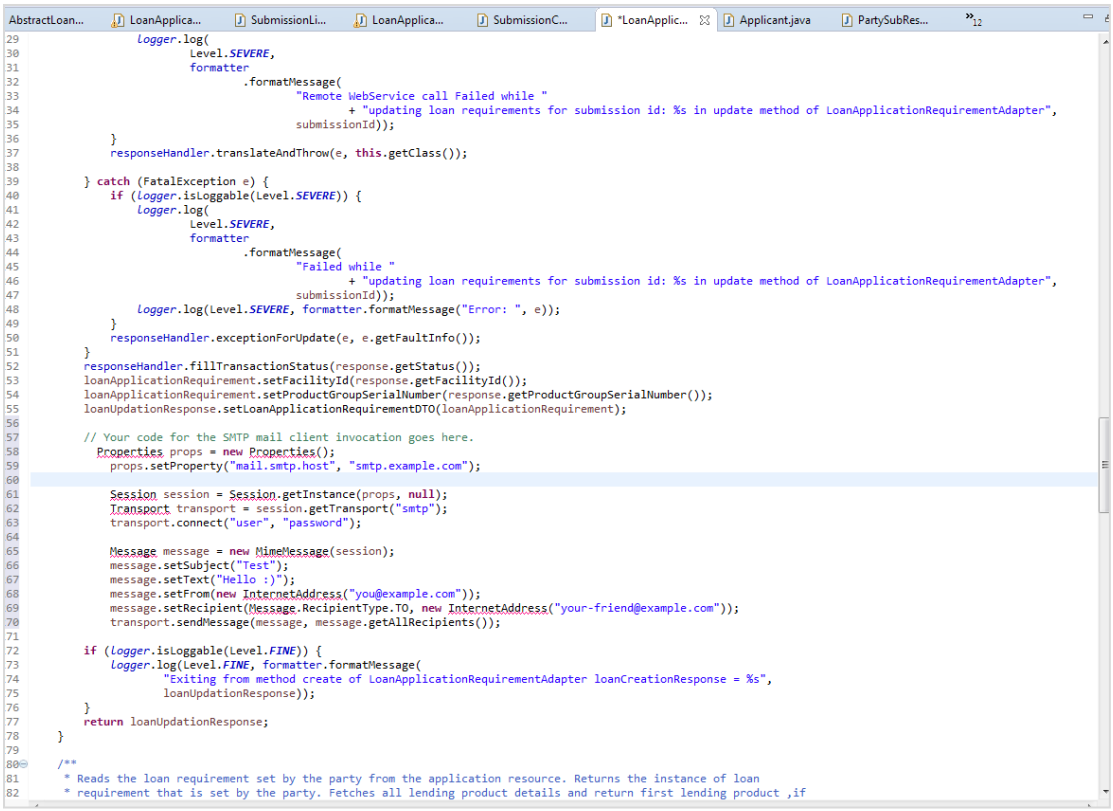
```
Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
package com.ofss.fc.app.adapter.party;

import java.util.List;

import com.ofss.fc.app.context.SessionContext;
import com.ofss.fc.app.loan.dto.LoanBalanceInquiryResponse;
import com.ofss.fc.app.loan.dto.account.unified.inquiry.LoanAccountUnifiedInquiryResponse;
import com.ofss.fc.app.loan.dto.disbursement.loghistory.LoanDisbursementLogHistoryResponse;
import com.ofss.fc.app.party.dto.LoanAccountAttributesDTO;
import com.ofss.fc.app.party.dto.relation.account.preferences.ChannelFacilitiesDTO;
import com.ofss.fc.enumeration.loan.LoanAccountStatusType;
import com.ofss.fc.framework.context.ApplicationContext;
import com.ofss.fc.infra.exception.FatalException;

/**
 * This class serves as the adapter class for the Party and loans Integration mainly for the Single Party View<br>
 *
 * @author VallabhM
 */
public interface ICustomerLoansAdapter {

    /**
     * Service to return all the loans Account information to be displayed in SPIV<br>
     *
     * @param applicationContext
     * @param partyId
     * @return LoanAccountAttributesDTO
     */
    public abstract LoanAccountAttributesDTO[] fetchLoanAccountsInformation(ApplicationContext applicationContext, String partyId);

    public abstract void fetchLNInformation(String accountId);

    /**
     * Maintain primary Account Holder ID for Loan Account.
     *
     * @param accountId
     *            ,newpartyId
     */
    public void primaryAccountHolderIDMaintenance(String accountId, String newPartyId);

    /**
     * @param loanAccounts
     * @param LoanAccountStatusType
     */
    public abstract void modifyStatusOfLoanAccounts(List<String> loanAccounts, LoanAccountStatusType LoanAccountStatusType);

    /**
     * Method to fetch Outstanding, RPA and unclear balances for a given loan account
     * @param accountId
     * @throws FatalException
     *
     */
    public abstract LoanBalanceInquiryResponse inquireLoanBalance(ApplicationContext applicationContext, String accountId) throws FatalException;

```

Custom Adapter Example

We take the example of LoanApplicationRequirementAdapter. For example the requirement is to send an email alert when the requirements of a particular loan application are updated. The OBDX application by default does not provide any integration with an SMTP/Email server. The additional interfacing with the gateway can be done in the custom adapter. . The following steps would have to be followed for implementation of a custom LoanApplicationRequirementAdapter.

Develop a *CustomLoanApplicationRequirementAdapter* and *Custom* LoanApplicationRequirementAdapterFactory. As a guideline, the custom adapter should extend the existing adapter and override the methods which needs to be replaced with new functionality.

For Example:



Custom Adapter Configuration

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values (‘IS_LOAN_APPLICATION_REQUIREMNT_ADAPTER_CUSTOM', ‘customadapterconfig’, 'true', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);

Mock Adapter

Mock adapter represents the intermediate layer required for communicating with third party host system. operations supported by Mock adapter are create, fetch, etc. Mock Adapter is basically responsible for generating required response from the core banking system for a given request.

Mock Adapter Example

Here it is responsible for communicating with third party host system as part of Loan requirement for a party.

LoanApplicationRequirementMockAdapter.java

```java
package com.ofss.digx.app.origination.adapter.impl.submission.application;

import java.util.logging.Level;

/**
 * Mock adapter represents the intermediate layer required for communicating with third party host system as part of
 * Loan requirement for a party.<br/>
 * Following operations are supported by this adapter.
 * <ol>
 * <li>Create loan application based on submission Identifier and loan details</li>
 * <li>Fetch loan application based on submission Identifier and facility Identifier</li>
 * </ol>
 */
public class LoanApplicationRequirementMockAdapter implements ILoanApplicationRequirementAdapter {

    /**
     * The component name for this class.
     */
    private static final String THIS_COMPONENT_NAME = LoanApplicationRequirementMockAdapter.class.getName();

    /**
     * Instance of {@link MultiEntityLogger}.
     */
    private static final MultiEntityLogger formatter = MultiEntityLogger.getUniqueInstance();

    /**
     * Instance of {@link java.util.logging.Logger} to support multi-entity wide logging.
     */
    private static final Logger logger = formatter.getLogger(THIS_COMPONENT_NAME);

    /**
     * The Simple name for this class.
     */
    private static final String THIS_SIMPLE_NAME = LoanApplicationRequirementMockAdapter.class.getSimpleName();

    /**
     * Creates the loan requirements of a party. An application for the loan requirement is created using submission
     * identifier and {@link LoanApplicationRequirementDTO} containing requested Amount, requested Tenor, list of
     * variants, etc. Returns the response containing information of successful or unsuccessful creation of the loan
     * requirement.
     *
     * @param submissionId
     *            unique identifier of the submitted application
     * @param loanApplicationRequirement
     *            {@link LoanApplicationRequirementDTO} representing the loan requirement submitted by the applicant
     * @return {@link LoanApplicationCreateResponseDTO} referring the response generated post creation of the loan
     *         requirement.
     * @throws Exception
     *             if the core banking system is not able to create a loan requirement based on details provided.
     */
    @Override
    public LoanApplicationCreateResponseDTO create(String submissionId,
            LoanApplicationRequirementDTO loanApplicationRequirement) throws Exception {
```

Mock Adapter Configuration

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values ('PARTY_COLLECTION_ADAPTER_MOCKED', 'adapterfactoryconfig', 'true', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);

4.7.3 Host adapter extension to populate pagination information

This extension feature helps developer to provide information regarding pagination from the host system. This will be typically used in inquiry transactions where large number of records is expected in response. To display such large data, pagination approach is used in user interface to display limited number of records at a time. Based on user action the subsequent records are fetched. The pagination information provided by this extension can be used in UI layer to display pagination response as per developer's requirement. The supported extension parameters are –

- *more* : a Boolean field to represent if any more data is available in response
- *totalRecords* : an Integer containing total number of records for the respective query
- *startSequence* : an Integer which can typically contain the sequence number of the first record in the next pagination records list.

To use the above extension following steps need to be executed.

- The response DTO of service should implement *'com.ofss.digx.app.dto.IpaginableI'* interface and should override all the methods of this interface.
- Add following snippet in respective *extxface* adapter after calling '*HostAdapterManager.processRequest(hostRequest)*'.

```
Map<Class<?>, PaginationResponseInfo> paginationInfo = new HashMap<>();
if (ThreadAttribute.get(ThreadAttribute.PAGINATION_INFO_MAP) != null
        && ThreadAttribute.get(ThreadAttribute.PAGINATION_INFO_MAP) instanceof HashMap) {
    paginationInfo = (Map<Class<?>, PaginationResponseInfo>) ThreadAttribute
            .get(ThreadAttribute.PAGINATION_INFO_MAP);
}

PaginationResponseInfo paginationValueDTO = new PaginationResponseInfo();
paginationValueDTO.setMore(hostResponse.hasMore);
paginationValueDTO.setTotalRecords(hostResponse.totalRecords);
paginationValueDTO.setStartSequence(hostResponse.startSequence);

paginationInfo.put(DemandDepositFinancialStatementItemDTO.class, paginationValueDTO);

ThreadAttribute.set(ThreadAttribute.PAGINATION_INFO_MAP, paginationInfo);
```

The host specific adapter should return values for 'hasMore', 'totalRecords', 'startSequence' in order to set the same in the Thread attribute.

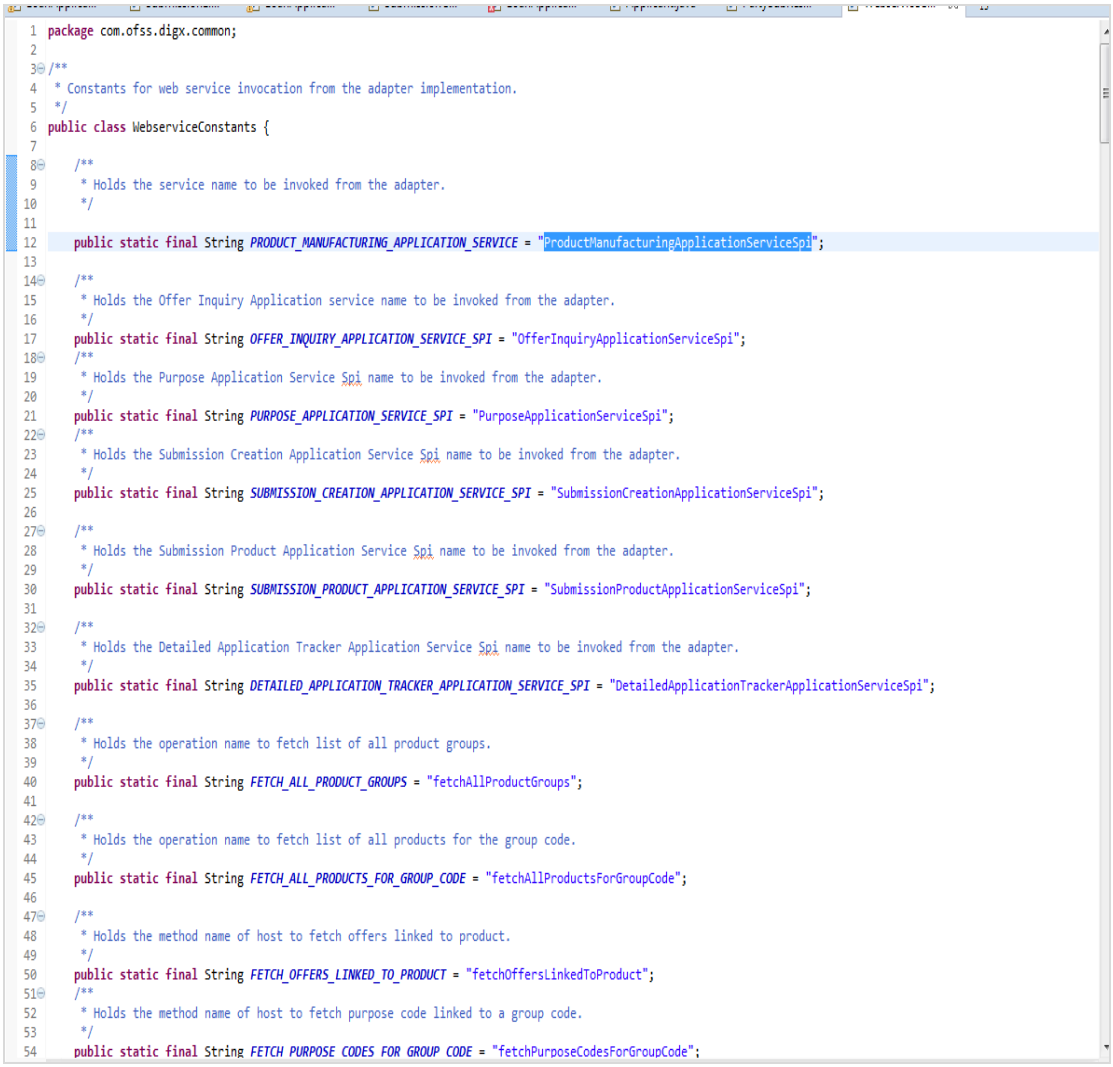
- The extension parameters set in the thread attribute will be available in the REST response as follows:

4.8 Outbound web service extensions

The outbound webservice configurations are set of properties defined to invoke services from the host. The host is the core bank system where the business logic for core banking facilities is written and contains the corresponding services to access that data. The existing OBDX application has an Adapter layer which directly interacts with the host. There are extension endpoints available for configuring a different host in the adapter layer. Following steps need to be followed:

Using your own web service constants

The web service constants will change depending on the WSDL specification provided by the host system. An Example WebServiceConstants file is shown below:



```java
package com.ofss.digx.common;

/**
 * Constants for web service invocation from the adapter implementation.
 */
public class WebserviceConstants {

    /**
     * Holds the service name to be invoked from the adapter.
     */

    public static final String PRODUCT_MANUFACTURING_APPLICATION_SERVICE = "ProductManufacturingApplicationServiceSpi";

    /**
     * Holds the Offer Inquiry Application service name to be invoked from the adapter.
     */
    public static final String OFFER_INQUIRY_APPLICATION_SERVICE_SPI = "OfferInquiryApplicationServiceSpi";
    /**
     * Holds the Purpose Application Service Spi name to be invoked from the adapter.
     */
    public static final String PURPOSE_APPLICATION_SERVICE_SPI = "PurposeApplicationServiceSpi";
    /**
     * Holds the Submission Creation Application Service Spi name to be invoked from the adapter.
     */
    public static final String SUBMISSION_CREATION_APPLICATION_SERVICE_SPI = "SubmissionCreationApplicationServiceSpi";

    /**
     * Holds the Submission Product Application Service Spi name to be invoked from the adapter.
     */
    public static final String SUBMISSION_PRODUCT_APPLICATION_SERVICE_SPI = "SubmissionProductApplicationServiceSpi";

    /**
     * Holds the Detailed Application Tracker Application Service Spi name to be invoked from the adapter.
     */
    public static final String DETAILED_APPLICATION_TRACKER_APPLICATION_SERVICE_SPI = "DetailedApplicationTrackerApplicationServiceSpi";

    /**
     * Holds the operation name to fetch list of all product groups.
     */
    public static final String FETCH_ALL_PRODUCT_GROUPS = "fetchAllProductGroups";

    /**
     * Holds the operation name to fetch list of all products for the group code.
     */
    public static final String FETCH_ALL_PRODUCTS_FOR_GROUP_CODE = "fetchAllProductsForGroupCode";

    /**
     * Holds the method name of host to fetch offers linked to product.
     */
    public static final String FETCH_OFFERS_LINKED_TO_PRODUCT = "fetchOffersLinkedToProduct";
    /**
     * Holds the method name of host to fetch purpose code linked to a group code.
     */
    public static final String FETCH_PURPOSE_CODES_FOR_GROUP_CODE = "fetchPurposeCodesForGroupCode";
```

Web service configuration

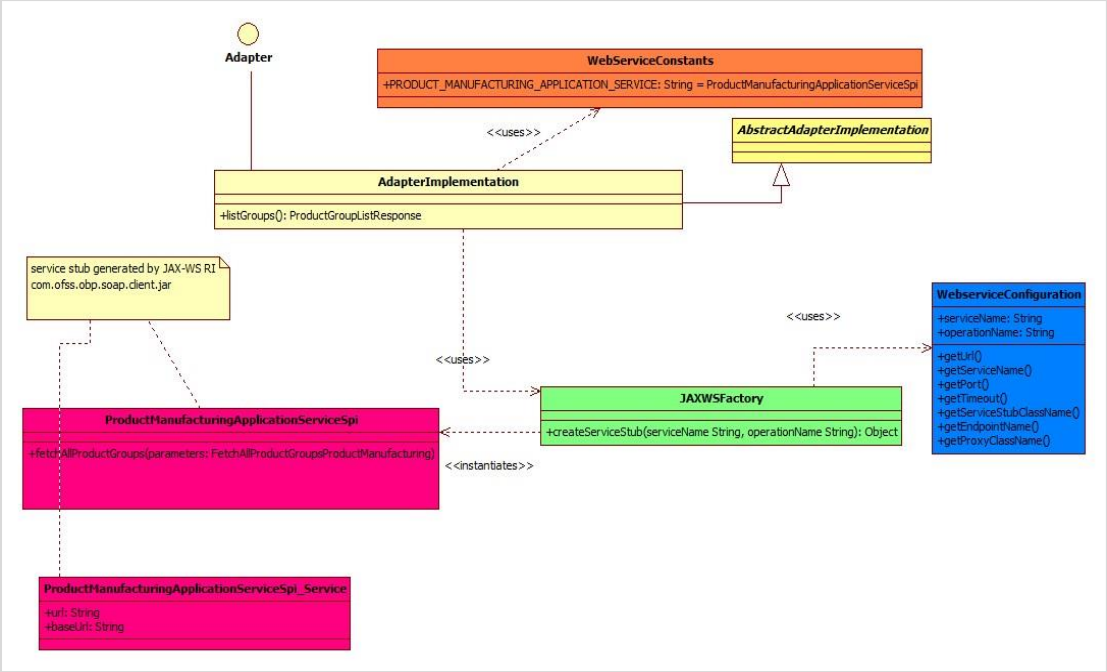
digx_fw_config_out_ws_cfg_b. Holds the entries for the host service endpoints.

For Example:

insert into digx_fw_config_out_ws_cfg_b (SERVICE_ID, PROCESS, URL, ENDPOINT_URL, NAMESPACE, TIME_OUT, SERVICE, STUB_CLASS, SECURITY_POLICY, ENDPOINT_NAME, STUB_SERVICE, HTTP_BASIC_AUTH_CONNECTOR, HTTP_BASIC_AUTH_REALM, PROXY_CLASS_NAME, IP, PORT, USERNAME, PASSWORD, CREATED_BY, LAST_UPDATED_BY, CREATION_DATE, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER, ANONYMOUS_SECURITY_POLICY, ANONYMOUS_SECURITY_KEY_NAME)

values ('inquireApplication', 'BaseApplicationServiceSpi',
'http://ofss310406.in.oracle.com:8001/com.ofss.fc.webservice/services/origination/BaseApplicationServiceSpi?wsdl', '',
'http://application.core.service.origination.appx.fc.ofss.com/BaseApplicationServiceSpi',
1200000, 'BaseApplicationServiceSpi', '', '', 'BaseApplicationServiceSpiPort',
'com.ofss.fc.appx.origination.service.core.application.baseapplicationservice

Class Diagram



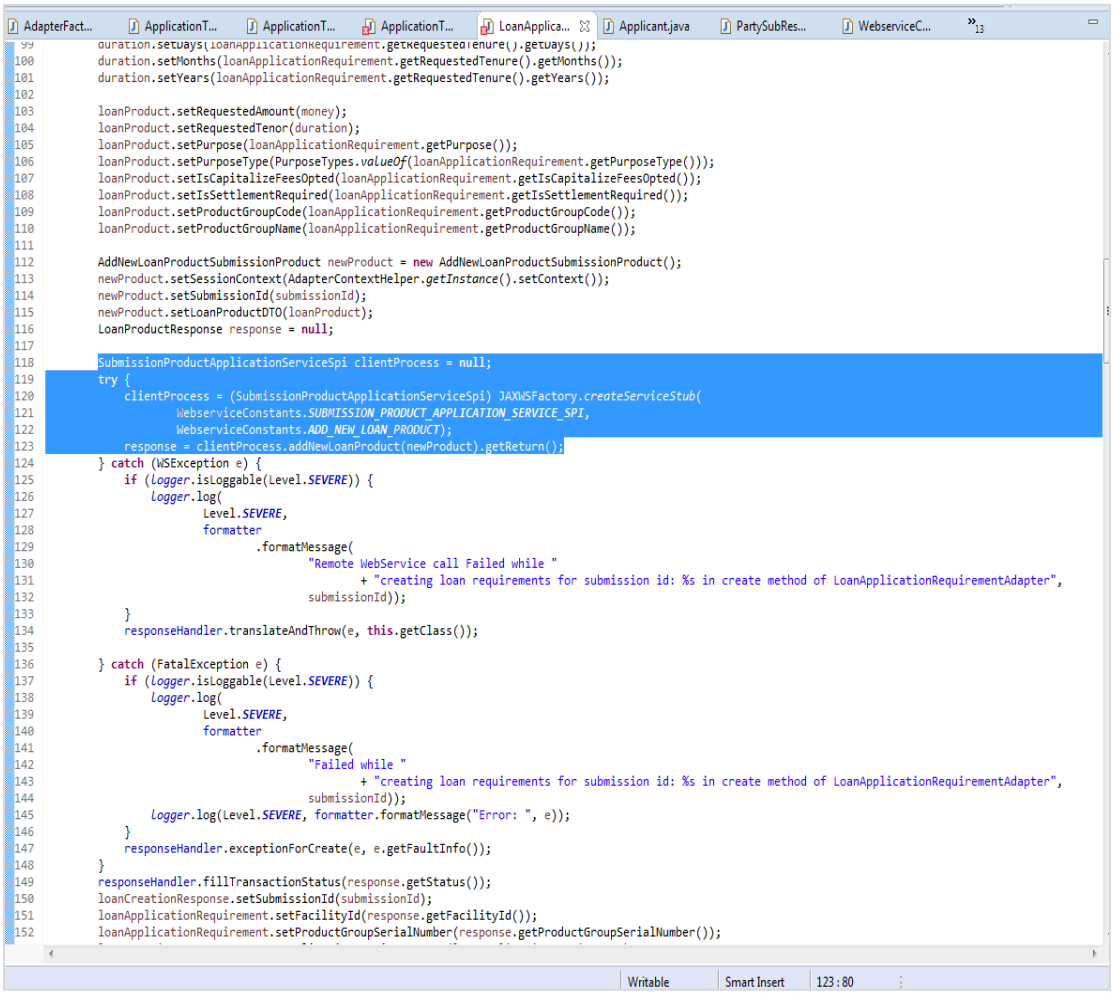
Client Jar

Generate the corresponding service stubs from the WSDL specifications using The JAX-WS RI tool. Package the generated code as a jar and include it in the Adapter implementation.

Custom Adapter

Lastly create a custom adapter to handle the changes made in the host configurations. The custom adapter will be using the JAXWSFacotry to create instances of the desired service stubs. The rest of the custom adapter implementation is the same as mentioned in the section 5.3

For example:



4.9 Security Customizations

OBDX comprising of several modules has to interface with various systems in an enterprise to transfer/share data which is generated during business activity that takes place during teller operations or processing. While managing the transactions that are within OBDX, it is needed to consider security & identity management and the uniform way in which these services need to be consumed by all applications in the enterprise.

OBDX provides a mechanism for creating permissions and role based authorization model that controls access of the user to OBDX services.

4.9.1 Out of box seeding of policies

When the application is installed, access policies are seeded for Day 0 configuration and access point definition by default.

The application is shipped with a CSV file – Day0Policy.csv, the policy data to be seeded by default.

4.10 Miscellaneous

This section lists some other features in OBDX platform that can be extended

4.10.1 Task Configurations

Task Registration:

Every new service to be integrated as a part of **OBDX** needs to provide a task code. This task code is required while integrating the
service with various infrastructural aspects applicable to the service. Few examples of infrastructural aspects or cross cutting
concerns provided out of the box with **OBDX** are:

- **Limits**
- **Approvals**
- **Two Factor Authentication**
- **Transaction Blackout**
- **Working Window**
- **Account Relationship**

Guidelines for formulating a task code are as follows:

A task code should ideally comprise of 3 parts:

1. **Module Name** : The first 2 alphabets representing the module to which the service in question belongs. eg TD represents Term Deposits
   module.
2. **Task Type(type of service) :** OBDX supports the following 6 types of services.

a. **FINANCIAL_TRANSACTION(F) :** Any transaction as a result of which there is a change in the status of the finances of accounts of
the participating parties. In general any transaction that involves monetary transfer between parties via their accounts. Few
examples include Self transfer, New deposit(Open term deposit), Bill payment etc.

b. **NONFINANCIAL_TRANSACTION(N) :** Any transaction that pertains to an account but there is no monetary payment or transfer involved
in it. For example Cheque book request.

c. **INQUIRY(I) :** Any read only transaction supported in OBDX that does not manipulate any business domain of the financial
institution. For example list debit cards, read loan repayment details, fetch term deposit penalties etc.

d. **ADMINISTRATION(A) :** Transactions performed by bank admins and corporate admins for a party come under this category. Few examples of
such transactions include limit definition, limit package definition, user creation, rule creation and various others.

e. **MAINTENANCE(M) :** Maintenances done by a party for itself fall under this category. Maintenance transactions performed by a
non admin user which does not involve any account or monetary transaction comprise of this transaction type. Example add biller.

f. **COMMON(C)** : Common transactions include transactions which do not fall under any of the above mentioned categorization. Example login.
So 1 alphbet F,N,I,A,M or C for each of the above mentioned task types respectively forms the second part of the task code.

3. **Abbreviation for service name :** A 3 to 10 lettered abbreviation for the service name. Example OTD for Open Term Deposit.
All the above mentioned 3 parts are delimited by an underscore character.
Example : TD_F_OTD where TD represents module name. F represents that its a financial transaction i.e. task type and OTD is the abbreviated
form of the transaction(service) name.

Task Aspects:

An ‘aspect’ of a task is a behavior or feature supported by the task. OBDX framework defines a set of aspects that can be supported by a task in the system. These aspects need to be configured in table DIGX_CM_TASK_ASPECTS. So if a task supports given aspect, then only its entry should be made in this table. If for any task, entry does not exist in this table for given aspect, then system treats it as that aspect is not supported by the task.

Additionally an aspect can be temporarily disabled using the ‘ENABLED’ column of this table. If the ‘ENABLED’ value is set as ‘N’, then system will treat it as this aspect is not supported by the task. Note that if a task is never going to support an aspect, then its entry should not be there in DIGX_CM_TASK_ASPECTS table. The ‘ENABLED’=’N’ option for disabling aspect should be used only when the task generally supports the aspect but it needs to be disabled for small duration.

Note that just having an entry in this table does not imply that the feature will be enabled for the task. The entry in this table only tells that system that the task supports this feature. Individual feature might need further configurations for them to work properly.

List of aspects supported by OBDX framework is listed below. Please note that aspects are not extensible – in other words it is not possible to add new aspects as part of customization.

Aspect	Description
grace-period	Indicates that the task supports grace period. Grace period is an additional period offered by Approval framework for approving a transaction Note: Grace Period will be applicable for the transactions with due date only.
ereceipt	Indicates that the task supports generation of e-receipts
audit	Indicates that the task supports audit logging
2fa	Indicates that the task supports two factor authentication
working-window	Indicates that the task supports working window
approval	Indicates that the task supports approval
blackout	Indicates that the task supports blackout
limit	Indicates that the task supports limit
Account Relationship	Indicates that the task supports account relationship check

Steps to register a task with OBDX:

The task code needs to be configured in the database table DIGX_CM_TASK. For example if we consider Open Term Deposit then the below
query fulfills the requirement mentioned in this step.

Insert into DIGX_CM_TASK (ID, NAME, PARENT_ID, EXECUTABLE, TASK_TYPE, MODULE_TYPE, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER) values ('TD_F_OTD', 'New Deposit', 'TD_F', 'Y',
'FINANCIAL_TRANSACTION', 'TD', 'ofssuser', sysdate, 'ofssuser', sysdate, null, 1);

As evident from the above query example Tasks have a hierarchy. Every task might have a

parent task denoted by the task code value held
by the PARENT_ID column of DIGX_CM_TASK. In most of the cases its a 3 level hierarchy.

- Leaf level tasks to which services are mapped at the lowest level
- Task representing the module to which the service belongs at the mid level
- Task representing the task type at the root level

For instance consider the task code AP_N_CUG which represents the Usergroup creation service under module approvals(AP). So the
PARENT_ID column of task AP_N_CUG(leaf level task) has task code as AP(mid level task). If we look at the entry for task code
AP(mid level task) then the value in the PARENT_ID column of DIGX_CM_TASK has MT(root level task) which is the task code representing
task type ADMINISTRATION. The leaf level task has 'Y' as the value in its EXECUTABLE column. The mid level and root level tasks have 'N' as the value in its EXECUTABLE column.

Step 2 – Configure aspects supported by the task. For example, if above task supports blackout, approval and working window, then following entries should be made.

```
Insert into DIGX_CM_TASK_ASPECTS (TASK_ID,ASPECT,ENABLED)
values ('TD_F_OTD','approval','Y');

Insert into DIGX_CM_TASK_ASPECTS (TASK_ID,ASPECT,ENABLED)
values ('TD_F_OTD','working-window','Y');

Insert into DIGX_CM_TASK_ASPECTS (TASK_ID,ASPECT,ENABLED)
values ('TD_F_OTD','blackout','Y');
```

Step 3 - Register the newly created service against this task.
For this step firstly, you need to get the service id for your service(transaction). Service id is the fully qualified name of the
class appended by the dot character (.) and the method name. For example taking open term deposit into consideration, the business
logic for the service is encapsulated in the method named create of the service
class com.ofss.digx.app.td.service.account.core.TermDeposit.
Hence the service id is derived as
: com.ofss.digx.app.td.service.account.core.TermDeposit.create
Secondly the below query fulfills the requirement mentioned in this step.

```
insert into DIGX_CM_RESOURCE_TASK_REL (ID, RESOURCE_NAME, TASK_ID,
CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE,
OBJECT_STATUS, OBJECT_VERSION_NUMBER) values ('1',
'com.ofss.digx.app.td.service.account.core.TermDeposit.create', 'TD_F_OTD', 'ofssuser',
sysdate, 'ofssuser', sysdate, null,1);
```

The aforesaid procedure enrolls your newly created service as a task in OBDX.

Limit Configuration

The below procedure describes the steps required to enable Limits for a newly developed service.

A prerequisite to this configuration is that this newly developed service should be registered as a task in OBDX. Refer "Task Registration" section for further details. **The types of Limits supported by the system are:**

- Periodic Limit(Cumulative) : Limits that get reset after the expiration of a period. Example Daily-limits.
- Duration Limit(Cooling Period) : Limits that get applicable after the occurrence of an event, for instance payee creation, and
  then are applicable for the specified duration after commencement of the event.
- Transaction Limit : Limits applicable to each invocation of a transaction. Holds minimum and maximum amount that can be transacted in a single transaction invocation.

Limits are applicable to targets. The types of targets supported by OBDX are Task and Payee.

- Task : Any service developed as a part of OBDX and registered as a task as mentioned in earlier sections
- Payee : A payee resource created via Payee creation transaction in OBDX.

To enable limits for a service, rather for a task mapped to the service to be precise, we need to follow the below mentioned steps:

- Ensure that the 'limit' aspect is configured in DIGX_CM_TASK_ASPECTS table and ENABLED column is updated as 'Y' for your task id.
- **Step 2.** Register taskEvaluatorFactory for your task code.
  This needs an insert in **DIGX_FW_CONFIG_ALL_B** table under the category_id '**taskEvaluatorFactories**' as shown below

Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) values (<<taskcode>>, 'taskEvaluatorFactories', 'com.ofss.digx.framework.task.evaluator.DefaultTaskEvaluatorFactory', 'N', null, 'Task Evaluator Factory for Mixed FT', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y', 1);

4. Code a LimitDataEvaluator for the task. LimitDataEvaluator is a class that extends AbstractLimitDataEvaluator class present in com.ofss.digx.finlimit.core.jar. This class is an abstract class which has only 1 abstract method having signature as shown below:

```
/**
 * provide {@link AbstractAspectData} of currently executing task.
 *
 * @param serviceInputs
 *          the service inputs
```

```
 * @return {@link AbstractAspectData} required for limit utilization and validation
 * @throws Exception
 */
public T evaluate(List<Object> serviceParameters) throws Exception
```

This method receives a List<Object> as an input. This list has all the arguments that were passed to the newly coded service for which limits needs to be enabled. For instance, consider the service to open a termed deposit. Signature of the service is as shown below.

public TermDepositAccountResponseDTO create(SessionContext sessionContext, TermDepositAccountDTO termDepositAccountDTO) throws Exception

In this case when the LimitDataEvaluator coded for open term deposit task i.e. TD_F_OTD is invoked by the OBDX framework, the serviceInputs argument of evaluate method will contain 2 objects in the list namely SessionContext and TermDepositAccountDTO. The return type of evaluate method is LimitData. The state of a LimitData object comprises of three variables:

- **CurrencyAmount :** an Object of type CurrencyAmount which represents the monetary amount involved in the ongoing transaction along with the currency in the transfer or payment is made.
- **payee** : An object of type PayeeDTO. Needs to be populated in case a payee is involved in the transaction.
- **limitTypesToBeValidated** : A list of LimitTypes. For all unexceptional practical purposes this needs to be populated as shown below:

```
limitTypesToBeValidated = new
ArrayList<LimitType>(Arrays.asList(LimitType.PERIODIC,LimitType.DURATION,LimitType.TRANSACTION)
);
```

These 3 fields in case applicable needs to be derived from the argument serviceInputs and populated in the returned LimitData object.

- Register the LimitDataEvaluator coded in Step 3.

This needs an insert in **DIGX_FW_CONFIG_ALL_B** table under the category_id '**limitDataEvaluator**' as shown below

```
Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID,PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE,
LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER)
values (<<task code>>, 'limitDataEvaluator', <<limitDataEvaluator>>, 'N',
'Limit data evaluator for <<service name>> service', null, 'ofssuser', sysdate,  'ofssuser',
sysdate, 'A', 1);
```

In the above query *<<task code>>* is the task code for the service, *<limitDataEvaluator>>* is the fully qualified name of the class coded in Step 3. *<<service name>>* is a descriptive name for the service.

- **Step 5.** Code a TargetEvaluator for your task.

Note: This step is needed only if your task requires limits involving Payees. Example Duration Limits and payee limits.

Payee limits are Periodic and Transactional limits applied on a Payee.

TargetEvaluator is a class that implements ITargetEvaluator interface. ITargetEvaluator is a functional interface that has only 1 method as shown below :

```
/**
* Evaluates the Target details for the given evaluated task code and service inputs in the form of
* {@link TargetDTO}.
*
* @param evaluatedTaskCode
* the given evaluated task code
* @param serviceInputs
* inputs of the service using this evaluator
* @return target details of the target for this task code and service inputs in the form of {@link TargetDTO}.
* @throws Exception
* exception while evaluating {@link TargetDTO}
*/
public TargetDTO evaluate(String evaluatedTaskCode, List<Object> serviceInputs) throws Exception;
```

This method accepts the task code and serviceInputs in case something needs to be derived from the arguments passed to the
service.

It returns a TargetDTO. TargetDTO has an id, name, value and TargetTypeDTO. TargetType tells whether the target is of type task or payee. If the TargetType is TASK then the variable value of TargetDTO holds the task code for the service.

If the TargetType is PAYEE then the variable value of TargetDTO holds the payeeId of the payee involved in the service.

As this step is required only for limits pertaining to payees so TargetType will be PAYEE and targetDTO's value will be payeeId.

Register the TargetEvaluator coded in Step 5.

Note: This step is needed only if your task requires limits involving Payees. Example Duration Limits and payee limts.

Payee limits are Periodic and Transactional limits applied on a Payee.

This needs an insert in **DIGX_FL_TARGET_EVALUATOR** table as shown below:

Insert into DIGX_FL_TARGET_EVALUATOR (TASK_CODE, TARGET_TYPE, EVALUATOR, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) values (<<task code>>, 'PAYEE', <<TargetEvaluator>>, null,

'target evaluator for <<service name>> service', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y',
1);

In the above query <<*task code*>> is the task code for the service, <<*TargetEvaluator*>> is the fully qualified name of the
class coded in Step 5. <<*service name*>> is a descriptive name for the service.

The aforesaid procedure enables limits for a task in OBDX.

Approval Configuration

The below procedure describes the steps required to enable Approvals for a newly developed service.

A prequisite to this configuration is that this newly developed service should be registered as a task in OBDX. Refer "Task Registration" section for further details.

To enable approvals for a service, rather for a task mapped to the service to be precise, we need to follow the below mentioned steps:

- Ensure that the 'approval' aspect is configured in DIGX_CM_TASK_ASPECTS table and ENABLED column is set to 'Y' for your task id.
- Note : If the newly created task is of type ADMINISTRATION and the maintenance is not specific to a party then this step is not
  required. Examples of such transaction are 2 Factor Authentication maintenance, limit maintenance and limit package maintenance.
  Tasks of type ADMINISTRATION which are specific to a party like Rule management tasks, workflow management tasks etc require this step.
  Tasks of type
  FINANCIAL_TRANSACTION,NONFINANCIAL_TRANSACTION,MAINTENANCE,INQUIRY and COMMON require this step.

  Code an approval assembler for the new task. An approval assembler is a class that extends AbstractApprovalAssembler.

There are 4 methods in abstract approval assembler out of which the one with the below signature:

public abstract T toDomainObject(D requestDTO) throws Exception;
will encapsulate the logic required to populate Transaction domain which is used by approvals framework.

Rest of the methods need to be overridden with empty or null implementations.
As evident from the signature quoted above this method accepts a requestDTO(an object that **IS A** DataTransferObject) and a transaction(an object that **IS A** Transaction).
requestDTO is the same DataTransferObject that was passed to your newly created service. For instance consider the service to open a termed deposit. Signature of the service is as shown below.

public TermDepositAccountResponseDTO create(SessionContext sessionContext,
TermDepositAccountDTO termDepositAccountDTO) throws Exception

In this case when the ApprovalAssembler coded for open term deposit task i.e. TD_F_OTD is invoked by the OBDX framework, the requestDTO argument of toDomainObject method will be the same as termDepositAccountDTO.

This method populates the transaction object on the basis of the requestDTO and returns the transaction domain. The guidelines
to override this method are as follows:-

- **Instantiation:**
  The transaction object passed will be null and needs to be instantiated. If the task type of the newly created service is FINANCIAL_TRANSACTION then the transaction needs to be instantiated as an object of AmountAccountTransaction.

  transaction = new AmountAccountTransaction();

  If the task type of the newly created service is NONFINANCIAL_TRANSACTION then the transaction needs to be instantiated as an
  object of AccountTransaction.

  transaction = new AccountTransaction();

  If the task type of the newly created service is MAINTENANCE then the transaction needs to be instantiated as an object of PartyTransaction.

  transaction = new PartyTransaction();

  If the task is of type ADMINISTRATION and the maintenance is specific to a party then the transaction needs to be instantiated as an object of PartyTransaction.

  transaction = new PartyTransaction();

  If the task is of type ADMINISTRATION and the maintenance is not specific to a party then the transaction needs to be instantiated
  as an object of Transaction.

  transaction = new Transaction();

- **Call** to **AbstractApprovalAssembler :**

  Call
  transaction = super.toDomainObject(requestDTO, transaction);

  This populates the generic state of transaction domain which does not change with the task for which approvals is being configured.
  c. Populate the state of the transaction domain which is specific to the task for which approvals is being configured. Cast the requestDTO to the type being accepted by the service. For example cast it to TermDepositAccountDTO as per the aforesaid example.

Use this DTO to populate the service specific state of the transaction domain like amount, account etc.

- **Step 3.** Register an approval assembler for your service or task.
  To register an approval assembler for your service an entry needs to be made in the database table *DIGX_FW_CONFIG_ALL*_B with
  the value of column *CATEGORY_ID* as 'approval_assembler'.

  If the newly created task is of type *ADMINISTRATION* and the maintenance is not specific to a party then the approval assembler to be registered against your service is om.ofss.digx.framework.domain.transaction.assembler.GenericDTOTransactionAssembler
  2 Factor Authentication Maintenance is a fine example of such transactions. The service id for this transaction is com.ofss.digx.app.security.service.authentication.maintenance.
  AuthenticationMaintenance.

  create
  The below query fulfills the requirement of this step:

```
Insert into DIGX_FW_CONFIG_ALL_B
(PROP_ID,
CATEGORY_ID,
PROP_VALUE,
FACTORY_SHIPPED_FLAG,
PROP_COMMENTS,
SUMMARY_TEXT,
CREATED_BY,
CREATION_DATE,
LAST_UPDATED_BY,
LAST_UPDATED_DATE,
OBJECT_STATUS,
OBJECT_VERSION_NUMBER)
values
('com.ofss.digx.app.security.service.authentication.maintenance.AuthenticationMaintenance.create',
'approval_assembler',
'com.ofss.digx.framework.domain.transaction.assembler.GenericDTOTransactionAssembler',
'N',
'assembler class for conversion from UserSegmentTFAMaintenanceDTO to Transaction domain',
'assembler class for conversion from UserSegmentTFAMaintenanceDTO to Transaction domain',
'ofssuser',
sysdate,
'ofssuser',
sysdate,
'A',
1);
```

  In all other cases where you have implemented a custom approval assembler as per the guidelines in step 2, the fully qualified
  class name of that approval assembler will be registered against your service. The below query fulfills the requirement of
  this step:

```
Insert into DIGX_FW_CONFIG_ALL_B
(PROP_ID,
CATEGORY_ID,
PROP_VALUE,
```

```
FACTORY_SHIPPED_FLAG,
PROP_COMMENTS,
SUMMARY_TEXT,
CREATED_BY,
CREATION_DATE,
LAST_UPDATED_BY,
LAST_UPDATED_DATE,
OBJECT_STATUS,
OBJECT_VERSION_NUMBER)
values
(<<service id>>,
'approval_assembler',
<<ApprovalAssembler>>,
'N',
'assembler class for conversion from DataTransferObject to Transaction domain',
'assembler class for conversion from DataTransferObject to Transaction domain',
'ofssuser',
sysdate,
'ofssuser',
sysdate,
'A',
1);
```

In the above query <<service id>> is the fully qualified name of the class appended by the dot character (.) and the method name. <<ApprovalAssembler>> denotes the fully qualified class name of the approval assembler coded in Step 2.

The aforesaid procedure enables approvals for a task in OBDX.

Account Relationship

Using this aspect, one can control accounts for a transaction.

1. Account Number List Filtration

To filter the account list based on Account Relationship configuration, task code should be provided in REST call in following manner

../digx/v1/accounts/demandDeposit?taskCode=TD_F_OTD

Above REST will return only allowed accounts for ‘New Deposit’ transaction.

2. Account Number Validation

Here we validate account number(s) using Account Relationship Configuration.

Following changes need to be done to achieve this

Evaluator class – If ‘Account Relationship Check’ is enabled for a transaction, then application looks for registered evaluator class. This class is used to identify account number(s) from

incoming request object and converts it into input which is required for account relationship checking.

Evaluator class should implement interface 'com.ofss.digx.app.accountrelationship.evaluator.mapping.IAccountRelationshipDataEvaluator'

Example - 'com.ofss.digx.app.td.evaluator.accountrelationship.TDAccountRelationshipEvaluator' is an evaluator class which is used for 'New Deposit' transaction.

Inside 'evaluate' method of this class, account number from request object 'com.ofss.digx.app.td.dto.account.TermDepositAccountDTO' is being get converted into list of 'com.ofss.digx.app.party.dto.relation.account.PartyToAccountRelationshipDTO'.

Following database entry has been made to register evaluator class '*com.ofss.digx.app.td.evaluator.accountrelationship.TDAccountRelationshipEvaluator*' with task code 'TD_F_OTD' of 'New Deposit' transaction.

Insert into DIGX_FW_CONFIG_ALL_B

(PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER, EDITABLE,

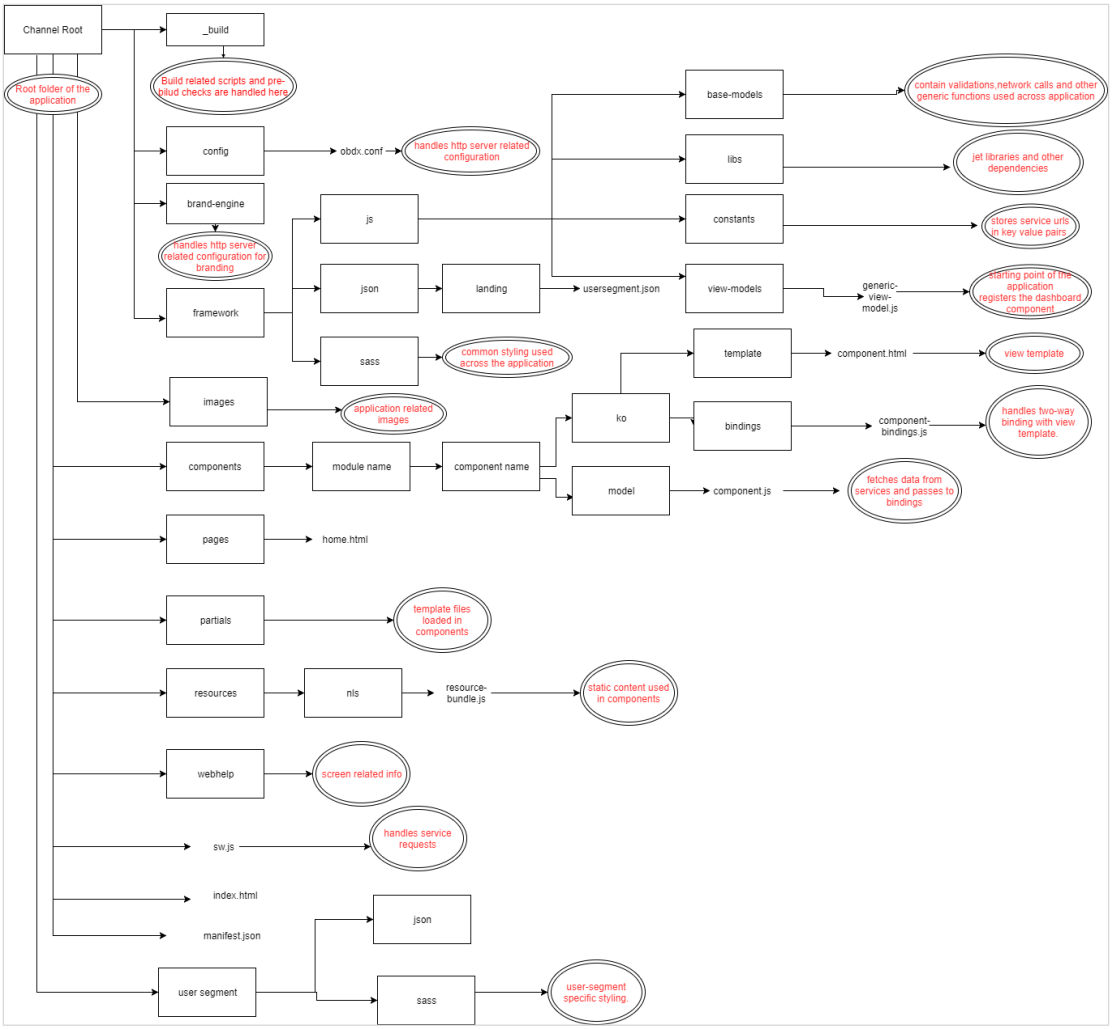
CATEGORY_DESCRIPTION) values

('TD_F_OTD', 'accountRelationshipEvaluator', 'com.ofss.digx.app.td.evaluator.accountrelationship.TDAccountRelationshipEvaluator', 'N', 'Account Relationship evaluator for TD', null, 'ofssuser', sysdate, 'ofssuser', sysdate, 'A', 1, 'N', null);

Home

5. Architecture of GUI Tier

Below diagram shows structure of the UI artifacts and some of the important artifacts are explained subsequently.



Home

6. Extensible Points in GUI Tier

This article provide the guidelines for UI Extensibility.

6.1 Theme and Brand

- CSS Custom Properties are available for modifications. You can change the variables by creating a new CSS file which has updated value of CSS custom properties. Make sure that file is imported after the main.css file.. Same functionality you can achieve by Branding. It is recommended that implementer should use Branding functionality.
- We are not allowing adding new styles in the core UI.
- For the Images you are free to do modifications.

6.2 Component Extensibility

- Framework Elements like (header, dashboard, menu etc.) are not available for the modification and customization.
- All components available under component folder are available for the extension.

6.2.1 Adding New and Overriding Existing Components

If you want to add new component place that component in <CHANNEL_ROOT_PATH>/extensions/components. It follow the same structure, which is present in components folder. Same thing is applicable for the existing components. If you want to change anything, then copy that component and place it extensions/components folder with the same structure.

If resource bundle needs to change for that component place related resource bundle in <CHANNEL_ROOT_PATH>/extensions/resources location. Structure remain same for <CHANNEL_ROOT_PATH>/resources and <CHANNEL_ROOT_PATH>/extensions/resources folder. Make sure that you updated the resource bundle path in your component.

If any component is present in <CHANNEL_ROOT_PATH>/extensions/components will take precedence over the <CHANNEL_ROOT_PATH>/components. For it we maintaining the list of components available in extensions in <CHANNEL_ROOT_PATH>/extensions/extension.js which is to be entered manually. For example:

Sample JS for **extension.js**

return {

"components": [<component1>,<component2>],

“partials” : [“partial1.html”,”partial2.html”],

“resources”: [<resource1>,<resource2>]

}

In the Same manner, you can override the partial templates.

Note: Out of the box, we are providing extension for Internal Account Input Component (internal-account-input). This extension need to be implemented in scenario where the bank account number do not have branch code prefixed in the account.

6.2.2 Add / Modify Validations

All the validation available in the application are maintained in **<CHANNEL_ROOT_PATH>/framework/js/base-models/validations/obdx-locale.js.** Implementer can override and add new validations in the application without changing this file. An extension hook is given at :

For OBDX 18.1 at <CHANNEL_ROOT_PATH>/extensions/validations/obdx-locale.js

From OBDX 18.2 onwards <CHANNEL_ROOT_PATH>extensions\override\obdx-locale.js

In this file Implementer can add or override validations.

For Example: If you need to change the pattern which validate Mobile Number. Add updated pattern in this file as below.

```
define([
    "ojL10n!resources/nls/obdx-locale"
], function(locale) {
    "use strict";
    var Locale = {
        validations: {
            MOBILE_NO: [{
                type: "regExp",
                options: {
                    pattern: "^(\\+\\d{1,3}[- ]?)?\\d{10}$",
                    messageDetail: locale.messages.MOBILE_NO
                }
            }]
        }
    }
    return Locale;
});
```

6.3 Calling custom REST service

In implementation if any new services are written by implementer it has been directed to change the context root for new REST to digx/cz/v1. For supporting it from the UI, implementer has to pass cz/v1 in the version field of the AJAX setting from his model.

For example see the snippet below:

```
fetchDetails = function(urlParams, deferred) {
  var options = {
    url: urlParams,
    version: "cz/v1",
    success: function(data) {
      deferred.resolve(data);
    },
    error: function(data) {
      deferred.reject(data);
    }
  };
  baseService.fetch(options);
},
```

Home

7. Libraries

OBDX has bundled its platform features and capabilities in various libraries based on logical separation of features. This section provides a list of such libraries along with their purpose

7.1 OBDX Libraries

This section provides information about various OBDX libraries that are provided out of the box.

7.1.1 Core/Framework Libraries

Provide infrastructure features of OBDX platform. These libraries are packaged in the enterprise application obdx.app.framework.ear

Library	Description
com.ofss.digx.infra	Provides basic infrastructure classes.
com.ofss.digx.infra.audit	Provides basic infrastructure classes for audit.
com.ofss.digx.infra.crypto	Provides cryptography functions such as hash generation, public private key generation and symmetric cryptography provider.
com.ofss.digx.infra.crypto.impl.jar	Provides default implementations of cryptography functions such as hash generation, public private key generation and symmetric cryptography provider.
com.ofss.digx.framework.domain	Provides base classes for entities, assemblers, repositories etc.
com.ofss.digx.framework.rest	Provides classes for calling host REST services.
com.ofss.digx.framework.adapter	Provides adapter interfaces for cross-domain invocation required for the framework.
com.ofss.digx.appx.core.rest	Provides infrastructure classes for OBDX REST services

com.ofss.digx.datatype	Provides complex data types used in OBDX application
com.ofss.digx.core.enumeration	Provides enumerations required for the core framework of the application.
com.ofss.digx.app.groovy.whitelist	??
com.ofss.digx.appcore	Provides base classes for application services, Interaction classes etc.
com.ofss.digx.security.core	Provides two factor authentication related core classes.
com.ofss.digx.appcore.dto	Provides DTOs used in infrastructure services

7.1.2 Common Librarie

Provide common libraries used across all modules of the application. These libraries are packaged in the enterprise application obdx.app.domain.ear

Library	Description
com.ofss.digx.accountrelationship.core	Provides classes for account relationship evaluators.
com.ofss.digx.adapter	Provides interfaces for cross-domain adapters.
com.ofss.digx.app.xface	Provides all request, response or plain DTOs used in services
com.ofss.digx.cloud.extension	Provides extension executers for groovy.
com.ofss.digx.common	Provides all constants and utilities to be used across the application.
com.ofss.digx.enumeration	Provides all enumerations.
com.ofss.digx.extxface	Provides adapters for interaction with external applications.

com.ofss.digx.finlimit.core	Provides core classes for financial limits processing
com.ofss.digx.security.provider	Provides core classes for two factor authentication.

7.1.3 Modules

Provide functional module available in the application. These libraries are packaged in the enterprise application obdx.app.domain.ear

Library	Description
com.ofss.digx.module.access	
com.ofss.digx.module.account	
com.ofss.digx.module.alerts	
com.ofss.digx.module.approval	
com.ofss.digx.module.audit	
com.ofss.digx.module.brand	
com.ofss.digx.module.budget	
com.ofss.digx.module.card	
com.ofss.digx.module.chatbot	
com.ofss.digx.module.collaboration	
com.ofss.digx.module.common	
com.ofss.digx.module.config	
com.ofss.digx.module.content	
com.ofss.digx.module.dda	
com.ofss.digx.module.extension	

com.ofss.digx.module.ebpp	
com.ofss.digx.module.forexdeal	
com.ofss.digx.module.fileupload	
com.ofss.digx.module.finlimit	
com.ofss.digx.module.goal	
com.ofss.digx.module.loan	
com.ofss.digx.module.location	
com.ofss.digx.module.me	
com.ofss.digx.module.mobile	
com.ofss.digx.module.origination	
com.ofss.digx.module.party	
com.ofss.digx.module.payment	
com.ofss.digx.module.pm	
com.ofss.digx.module.report	
com.ofss.digx.module.security	
com.ofss.digx.module.sms	
com.ofss.digx.module.spendanalysis	
com.ofss.digx.module.sr	
com.ofss.digx.module.td	
com.ofss.digx.module.tradefinance	
com.ofss.digx.module.user	
com.ofss.digx.module.wallet	

7.1.4 Endpoints

Provide functional module available in the application. These libraries are packaged in the enterprise application obdx.app.domain.ear

Library	Description
com.ofss.digx.appx.service.rest	Provides REST endpoint classes for all modules

7.1.5 External System Adapters

These are packaged into obdx.extsystem.domain.ear

Library	Description
com.ofss.digx.extxface	Provides all external interfaces
com.ofss.digx.extxface.sms.dbAuthenticator	Provides all the implementations for OBDX database authenticator
com.ofss.digx.extxface.<Host Name>.impl	Provides adapter implementations of the external interfaces for particular host
com.ofss.<Host Name>.soap.client	Provides stubs used for communicating with host

Home

8. Workspace Setup

In order to implement the DIGX architecture, we will create separate projects for different framework components in Eclipse (with JDK 8)

Why separate projects? : Ensures high extensibility and loose coupling between different components of the system. Also, in later stages, sustenance becomes easier and it helps developers also to effectively maintain the ever-growing code.

Moving on, let's create the following projects as shown in the Figure 2:

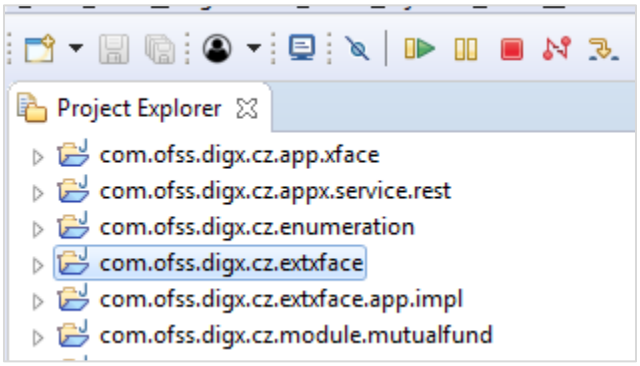


Figure 2 **Project structure**

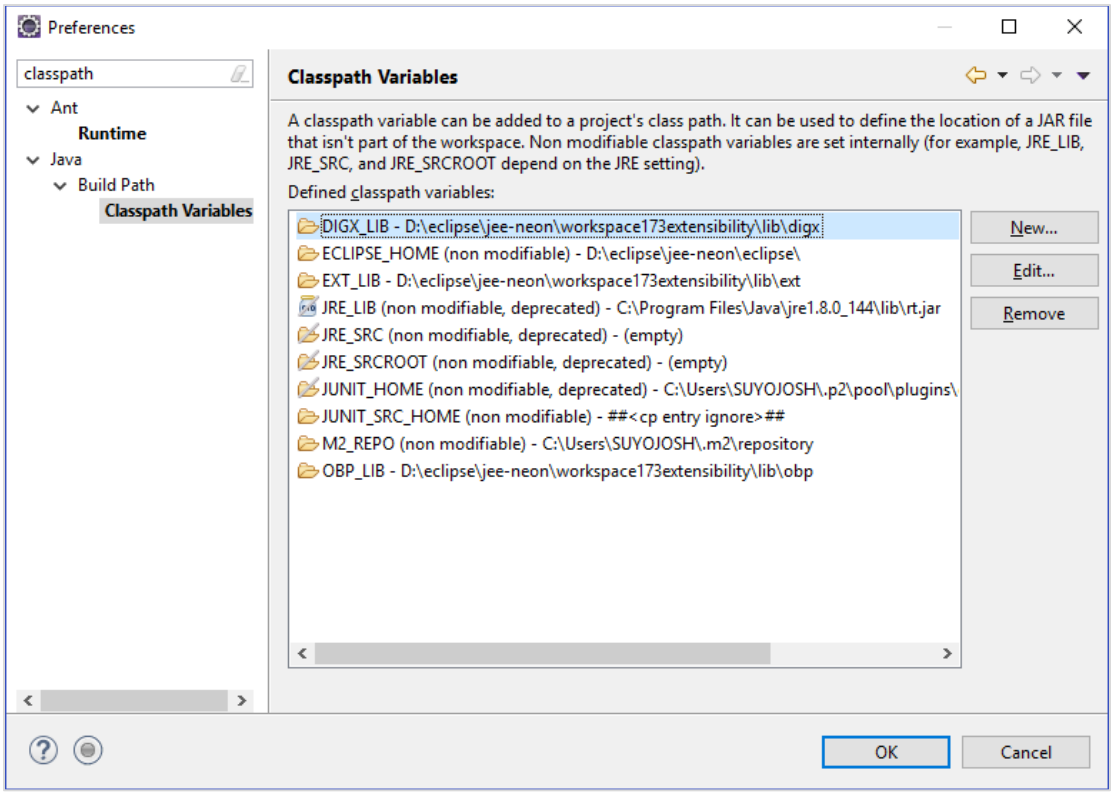
8.1 Basic Setup

Create a new workspace in Eclipse and launch that workspace.

Go to Window → Preferences. Go to Java → Build Path → Classpath Variables. Set following variables there

Classpath Variable	Description
DIGX_LIB	Refers to the base location of OBDX libraries
OBP_LIB	Refers to the base location of OBP libraries
EXT_LIB	Refers to the base location of external libraries

The screen will look like this.



8.2 DTO (xface) project

Create a Java Project with the project name '**com.ofss.digx.cz.app.xface'**. Now create a package in that project with the name **'com.ofss.digx.cz.app.<module_name>.dto' .**

This package consists of Data Transfer Object classes, also referred as Plain Old Java Objects or POJO. All the request as well as response DTO classes are created under this project. The Request DTO classes in this project extend **DataTransferObject** present in OBP libraries whereas the Response DTO classes extend **BaseResponseObject**.

Project dependencies for xface project:

Not needed.

Classpath variables:

Extend the following classpath variables to the jars found in OBDX_Installer:

Classpath name	Jars to extend
DIGX_LIB	com.ofss.digx.app.xface.jar com.ofss.digx.appcore.dto.jar

	com.ofss.digx.infra.jar
OBP_LIB	com.ofss.fc.appcore.dto.jar com.ofss.fc.infra.jar

8.3 REST endpoint

Create a Java Project (Note: No need to create Dynamic Web Project from 18.1 onwards) with the project name '**com.ofss.digx.cz.appx.service.rest**'. Now create a package in that project with the name **'com.ofss.digx.cz.appx.<module_name>'** .

This package contains the endpoint class to take requests and send responses back to server. The REST classes usually contain JAX-RS annotations and URIs which help them to locate themselves whenever a request is made through a REST call.

All the classes in this project extend **AbstractRESTApplication** and also have a default interface class prefixed with 'I' before the name of the corresponding REST implementation class. For instance, Account REST class will have IAccount Interface class in the same the package of this project.

Project dependencies for REST project:

Add the following projects (to be created later) in the Java build path of this project:

com.ofss.digx.cz.app.xface, com.ofss.digx.cz.module.<module_name>

Classpath variables:

Create the following classpath variables (left column) and extend them to the jars (names on the right side column). **Note** that these libraries can be found in the OBDX Installer folder and the exact location of each jar can be found in the section 8 (last part) of this document.

Classpath name	Jars to extend
DIGX_LIB	com.ofss.digx.appcore.jar com.ofss.digx.appcore.dto.jar com.ofss.digx.common.jar com.ofss.digx.datatype.jar com.ofss.digx.enumeration.jar com.ofss.digx.infra.audit.jar com.ofss.digx.infra.jar com.ofss.digx.infra.crypto.jar com.ofss.digx.app.xface.jar com.ofss.digx.appx.core.rest.jar com.ofss.digx.module.<Module Name>.jar (Based on which module

	service you are going to consume)
EXT_LIB	All jersey2 libraries found in OBDX installer folder
OBP_LIB	com.ofss.fc.appcore.dto.jar com.ofss.fc.appcore.jar com.ofss.fc.enumeration.jar com.ofss.fc.infra.jar

8.4 Module

Create a new Java project with this name. This project contains the vital business logic, extension points, constraints, security checks like authorization and access control. The following packages need to be created inside this project:

1. **com.ofss.digx.cz.app.<module_name>.service :** Add the Service Interface and Implementation class in this package. The name of Service class should be same as the name of the REST class created in the REST project. For instance, this package will have classes named IAccount.java and Account.java which are same as the REST class name for Account. This service class extends **AbstractApplication** of the DIGX framework.
2. **com.ofss.digx.cz.app.<module_name>.service.ext:** Contains classes for extensions and executors. **Each** Service classes will have their own extension points. Refer mock workspace for more detail.
3. **com.ofss.digx.cz.app.<module_name>.assembler:** Create <module_name>Assembler class inside this **package**. All Assembler classes extends **AbstractAssembler** .
4. **com.ofss.digx.cz.domain.<module_name>.entity:** This package should include Entity class for the **module**. The name of entity class to be created should be same as REST as well as Service class names. For instance, it will have Account.java entity class for Account service and REST classes. Also known as Domain classes, they extend **AbstractDomainObject** taken from OBP libraries.
5. **com.ofss.digx.cz.domain.<module_name>.entity.assembler:** Add a domain assembler class with the name <**module**_name>DomainAssembler in this package.
6. **com.ofss.digx.cz.domain.<module_name>.entity.policy:** Add the business policy classes in this package to ensure the validation of business constraints added in these classes. Refer workspace attached **with** this document for more detail. Classes in this project are again one of the must-haves as far as enforcement of any system validation is concerned.
7. **com.ofss.digx.cz.domain.<module_name>.entity.repository:** Contains repository class (<module_name>Repository.java) which invokes Repository adapter classes described in the next point. This class extends **AbstractDomainObjectRepository** of DIGX framework.
8. **com.ofss.digx.cz.domain.account.entity.repository.adapter:** Add repository adapter interfaces, Local and Remote Repository Adapter classes in this project. If you are writing for the Account service, the naming convention of these classes should be I<module_name>RepositoryAdapter, Local<module_name>RepositoryAdapter, Remote<module_name>RepositoryAdapter respectively.

With this ends the package structure for service classes. The implementation of this project takes maximum time and involves majority of the DIGX service layer handling. It is therefore a very crucial part to look for while developing a REST API in DIGX.

Project dependencies for module project:

Add the following projects in the Java build path of this project:

com.ofss.digx.cz.adapter, com.ofss.digx.cz.app.xface

Classpath variables:

Extend the following classpath variables to the jars found in OBDX_Installer:

Classpath variable name	Jars to extend
DIGX_LIB	com.ofss.digx.framework.domain.jar com.ofss.digx.infra.jar com.ofss.digx.appcore.jar com.ofss.digx.common.jar com.ofss.digx.datatype.jar com.ofss.digx.adapter.jar com.ofss.digx.module.alerts.jar com.ofss.digx.module.approval.jar com.ofss.digx.module.party.jar com.ofss.digx.enumeration.jar com.ofss.digx.module.common.jar com.ofss.digx.appcore.dto.jar
OBP_LIB	com.ofss.fc.framework.domain.jar com.ofss.fc.enumeration.jar com.ofss.fc.datatype.jar com.ofss.obp.patch.jar com.ofss.fc.infra.jar com.ofss.fc.appcore.dto.jar com.ofss.fc.appcore.jar com.ofss.fc.framework.dto.jar

8.5 Adapter

Create a Java Project with name **com.ofss.digx.cz.extxface** which contains all the Adapter Interfaces in this project. This project is required if you are creating new adapter interfaces. Within this project create a package with the name **com.ofss.digx.cz.extxface.<module_name>.adapter.** Now include the adapter interface for the adapter implementation class for your module. For example, in case of Account module, name of the interface created should be IAccountAdapter.

Project dependencies for adapter project:

Add the following projects in the Java build path of this project:

com.ofss.digx.cz.app.xface

Classpath variables:

Extend the following classpath variables to the jars found in OBDX_Installer:

Classpath name	Jars to extend
DIGX_LIB	com.ofss.digx.infra.jar

8.6 Adapter Impl

Create a project with name **com.ofss.digx.cz.extxface.<Host Name>.impl** This project will contain implementation classes for all the adapter interfaces created in the **com.ofss.digx.cz.adapter** project for required host. Create a package named **com.ofss.digx.cz.extxface.<module_name>.adapter.impl** and add the following classes:

1. **<module_name>AdapterFactory.java :** Factory class to generate Adapter instances for every getAdapter request call. Returns either mock adapter or adapter to call host interface
2. **<module_name>Adapter.java:** A very essential Adapter class for a specific module which is entitled to call external host system
3. **<module_name>MockAdapter.java:** In case a call to host system needs to be skipped and local mocked data needs to be fetched, this adapter class can be used

Project dependencies for adapter impl project:

Add the following projects in the Java build path of this project:

com.ofss.digx.cz.adapter

com.ofss.digx.cz.app.xface

com.ofss.digx.cz.module.<module_name>

Classpath variables:

Extend the following classpath variables to the jars found in OBDX_Installer:

Classpath name	Jars to extend
DIGX_LIB	com.ofss.digx.appcore.dto.jar com.ofss.digx.infra.jar com.ofss.digx.adapter.jar com.ofss.digx.datatype.jar
OBP_LIB	com.ofss.fc.framework.dto.jar

Database Scripts to be added:

There are few places where we decide which classes to be invoked in runtime. These are the possible database configurations done in an ideal case. Please add the following entries in the DIGX_FW_CONFIG_ALL_B table: (Account Service taken as an example and in accordance with the workspace example)

```
INSERT INTO digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) VALUES ('ACCOUNT_CZ_REPOSITORY_ADAPTER', 'repositoryadapterconfig', 'com.ofss.digx.domain.account.entity.repository.adapter.RemoteAccountRepositoryAdapter', 'N', null, 'Adapter repository adapter class', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y', 1);

INSERT INTO digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) VALUES ('ACCOUNT_CZ_ADAPTER_FACTORY', 'adapterfactoryconfig', 'com.ofss.digx.app.account.adapter.impl.AccountAdapterFactory', 'N', null, 'adapter factory class', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y', 1);

INSERT INTO digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) VALUES ('ACCOUNT_CZ_ADAPTER_MOCKED', 'adapterfactoryconfig', 'false', 'N', null, 'Flag to decide to go to Mocked adapter or Remote', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y', 1);
```

8.7 SOAP client

This is a project that will contain stubs that will be used to invoke Host services. This can contain stubs generated by importing the WSDL published by host. This is an optional project and depends on the kind of host integration pattern followed.

Home

9. Deployment

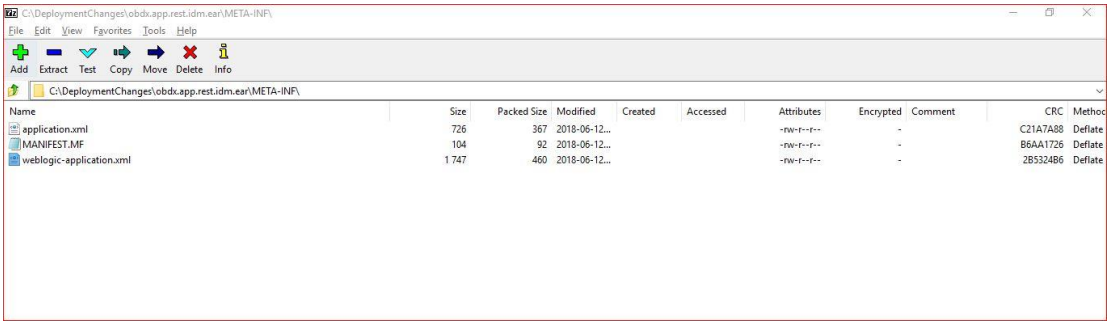
Package customized code in library EAR:

Once all the classes are created and implemented, generate the required ear deployments. The following two EARs need to be created: **com.ofss.digx.cz.app.extsystem.ear** and **obdx.cz.app..domain.ear**.

To generate an EAR in eclipse, we need to create an Enterprise Application Project and include the required project/s during the creation.The REST IDM (“obdx.app.rest.idm.ear”) already deployed on the server must be modified in order to get the CZ EARs deployment working.

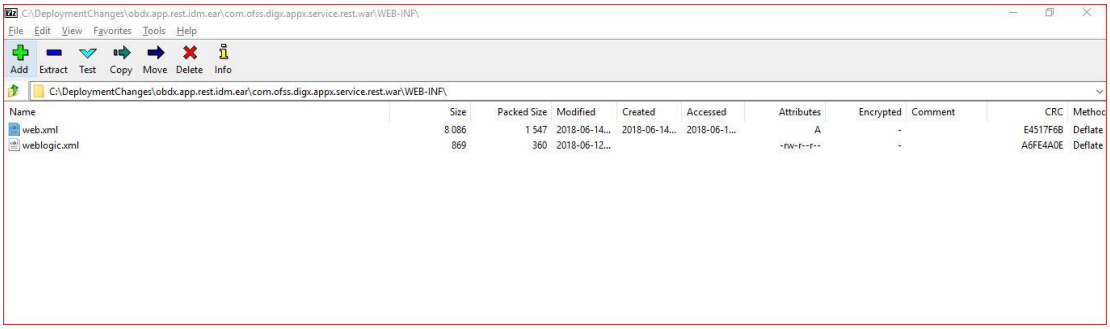
For the purpose two modifications must be done:

1. Add the CZ EARs in the weblogic-application.xml file located at obdx.app.rest.idm.ear/META-INF/ weblogic-application.xml. Changes are depicted below:




```
        <wls:library-name>obdx.extsystem.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.app.core.patch</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.app.core.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.app.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.thirdparty.app.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.cz.app.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.cz.extsystem.domain</wls:library-name>
    </wls:library-ref>
    <wls:library-ref>
        <wls:library-name>obdx.cz.thirdparty.app.domain</wls:library-name>
    </wls:library-ref>

    <wls:prefer-application-packages>
        <package-name>org.eclipse.persistence.*</package-name>
    </wls:prefer-application-packages>
</wls:weblogic-application>
```

2.Add CZ Servlet and the servlet mapping in the web.xml file located at obdx.app.rest.idm.ear/com.ofss.digx.appx.service.rest.war/WEB-INF/web.xml. The changes are depicted below:



web.xml file already has so many servlets, do not change them, just add the CZ Servlet.

```
    <!-- CZServlet -->
  <servlet>
      <description>Custom domain servlet</description>
      <servlet-name>CZServlet</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
      <!--Register resources and providers under my.package. -->
      <init-param>
          <param-name>javax.ws.rs.Application</param-name>
          <param-value>com.ofss.digx.appx.Application</param-value>
      </init-param>
      <!-- Register my custom provider -->
      <init-param>
          <param-name>jersey.config.server.provider.classnames</param-name>
          <param-value>com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider;org.glassfish.jersey.jackson.JacksonFeature;o
      </init-param>
      <init-param>
          <param-name>obdx.rest.resource.filename</param-name>
          <param-value>cz.resources</param-value>
      </init-param>
      <init-param>
          <param-name>openApi.configuration.prettyPrint</param-name>
          <param-value>true</param-value>
      </init-param>
      <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>CZServlet</servlet-name>
      <url-pattern>/v1/cz/*</url-pattern>
  </servlet-mapping>
```

In the above CZ servlet the “Register my custom provider” part of init-params tag is depicted below:

```
- Register my custom provider -->
it-param>
 <param-name>jersey.config.server.provider.classnames</param-name>
 <param-value>com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider;org.glassfish.jersey.jackson.JacksonFeature;org.glassfish.jersey.media.multipart.MultiPartFeature</param-value>
nit-param>
```

Deploying application in Weblogic:

The two EARs created should be deployed in the existing deployment setup. Please deploy the com.ofss.digx.app.rest.idm.ear as an application and **com.ofss.digx.cz.app.extsystem.ear** and **obdx.app.cz.domain.ear** as library.

Test the application:

Once the application is up, please go to the deployments section of the Weblogic Server. In the control option, you’ll find the option to test the application. Just to verify, check whether the

context-root of the custom application is changed to digx/cz. The request URL for testing this application will be –s

http://<hostname>:<port>/digx/cz/v1/application.wadl

Home