Oracle® Banking Digital Experience

Extensibility Guide

Release 25.1.0.0.0
G38590-01
July 2025

Oracle Banking Digital Experience Extensibility Guide, Release 25.1.0.0.0

G38590-01

Contents

Preface

Purpose

This guide is designed to help acquaint you with the Oracle Banking application. This guide provides answers to specific features and procedures that the user need to be aware of the module to function successfully.

Audience

This document is intended for the following audience:

- Customers
- Partners

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=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Critical Patches

Oracle advises customers to get all their security vulnerability information from the Oracle Critical Patch Update Advisory, which is available at Critical Patches, Security Alerts and

Bulletins. All critical patches should be applied in a timely manner to ensure effective security, as strongly recommended by Oracle Software Security Assurance.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Related Resources

For more information on any related features, refer to the following documents:

- Oracle Banking Digital Experience Installation Manuals
- Oracle Banking Digital Experience Licensing Manuals

Screenshot Disclaimer

Personal information used in the interface or documents is dummy and does not exist in the real world. It is only for reference purposes; actual screens that appear in the application may vary based on selected browser, theme, and mobile devices.

Acronyms and Abbreviations

The list of the acronyms and abbreviations used in this guide are as follows:

Table 1 Acronyms and Abbreviations

Abbreviation	Description
OBDX	Oracle Banking Digital Experience

1

Objective and Scope

1.1 Background

This topic provides information on **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.

1.2 Objective

This topic provides information on **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.

1.3 Scope

This topic provides information on **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.

1.4 Structure

This topic provides information on **Structure**.

This document is organized into following chapters:

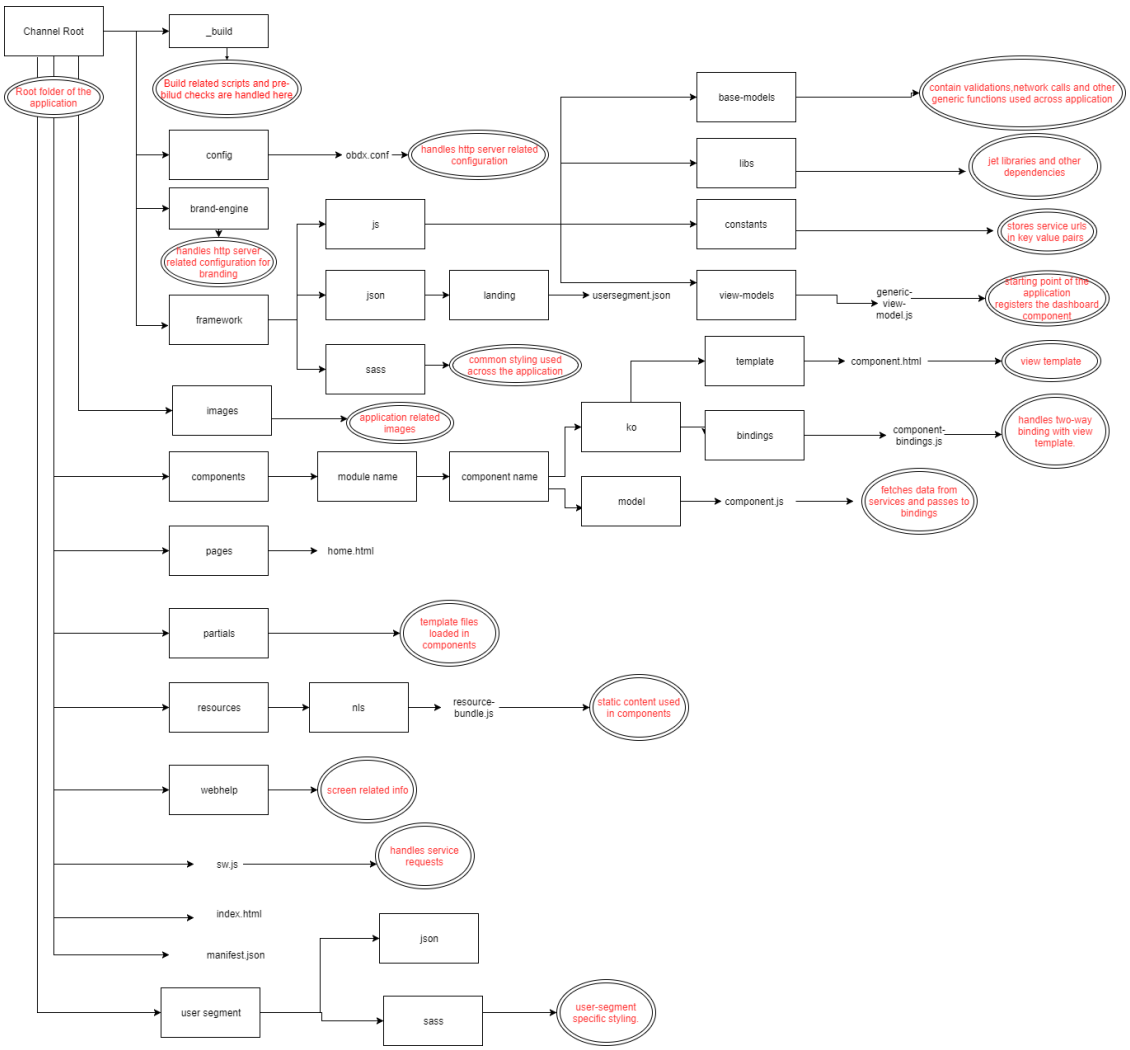
- Architecture of Service Tier: Provides overall architecture of the service tier of OBDX platform. This chapter will set the context for further chapters and also will introduce you to various terminologies that you will encounter throughout this document
- Extensible Points in Service Tier: Provides 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.

2

Architecture of GUI Tier

This topic provides information on **Architecture of GUI Tier**.

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



3
Extensible Points in Service Tier

This topic provides information on **Extensible Points in Service Tier**. Various extensible points / hooks provided by OBDX framework, are explained in detail in this section.

- REST Tier
  This topic provides information on **REST Tier**.
- Service Extensions
  This topic provides information on **Service Extensions**.
- Business Policy
  This topic provides information on **Business Policy**.
- Dictionary
  This topic provides information on **Dictionary**.
- Domain Extensions
  This topic provides information on **Domain Extensions**.
- Error Messages
  This topic provides information on **Error Messages**.
- Adapter Tier
  This topic provides information on **Adapter Tier**.
- Outbound web service extensions
  This topic provides information on **Outbound web service extensions**.
- Security Customizations
  This topic provides information on **Security Customizations**.
- Taxonomy Validations
  This topic provides information on **Taxonomy Validations**. For extensions in taxonomy validations, please refer to **Oracle Banking Digital Experience Taxonomy Configuration Guide**
- Authentication Extensibility
  This topic provides information on **Authentication Extensibility**.
- Miscellaneous
  This topic provides information on **Miscellaneous**.

3.1 REST Tier

This topic provides information on **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.

- Guidelines
  This topic provides information on **Guidelines**.
- HTTP Standards
  This topic provides information on **HTTP Standards**.

3.1.1 Guidelines

This topic provides information on **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.

3.1.2 HTTP Standards

This topic provides information on **HTTP Standards**.

HTTP Methods

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

For more information on fields, refer to the field description table.

Table 3-1 HTTP Methods

Method	Purpose
GET	Retrieve / fetch the resource
POST	Create a new resource

Table 3-1 (Cont.) HTTP Methods

Method	Purpose
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.

For more information on fields, refer to the field description table.

Table 3-2 HTTP Response Codes

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

3.2 Service Extensions

This topic provides information on **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:

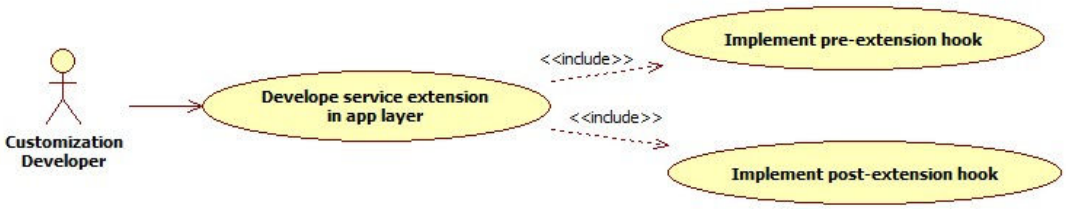
- Additional input validations
- Execution of business logic, which necessarily has to happen before going ahead with normal service execution.

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:

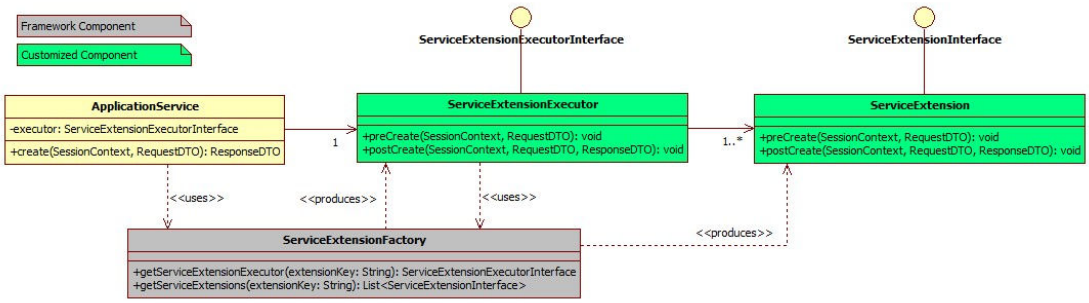
- Output response manipulation
- Custom data logging for subsequent processing or reporting.

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.

- Service Extension Interface
  This topic provides information on **Service Extension Interface**.
- Service Extension Executor Interface
  This topic provides information on **Service Extension Executor Interface**.
- Default Extension (Void Extension)
  This topic provides information on **Default Extension (Void Extension)**.

- Custom Extension
  This topic provides information on **Custom Extension**.
- Service Extension Configurations
  This topic provides information on **Service Extension Configurations**.
- Sequence of events in service extension
  This topic provides information on **Sequence of events in service extension**.

3.2.1 Service Extension Interface

This topic provides information on **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
     */
```

3.2.2 Service Extension Executor Interface

This topic provides information on **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.

```
public interface ILoanApplicationExtExecutor {

    /**
     * Executor point for LoanApplication.create. It executes the extensions available before creating loan applications
     * from the Service class {@code LoanApplicationApplicationService}.
     *
     */
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
            throws Exception;

    /**
     * Executor point for LoanApplication.create. It executes the extensions available 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
     *                                                                              loan application
     *
     *
     * @param
     *                                                                            g loan application
     *
     *
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext,
            LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
            LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception;

    /**
     * Executor point for LoanApplication.update. It executes the extensions available 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
     */
```

LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO - com.ofss.digx.app.origination.service.submission.application.ext.ILoanApplicationExtExecutor.postCreate(SessionContext, LoanApplicationCreateRequestDTO, LoanApplicationCreateResponseDTO)

Press 'F2' for focus

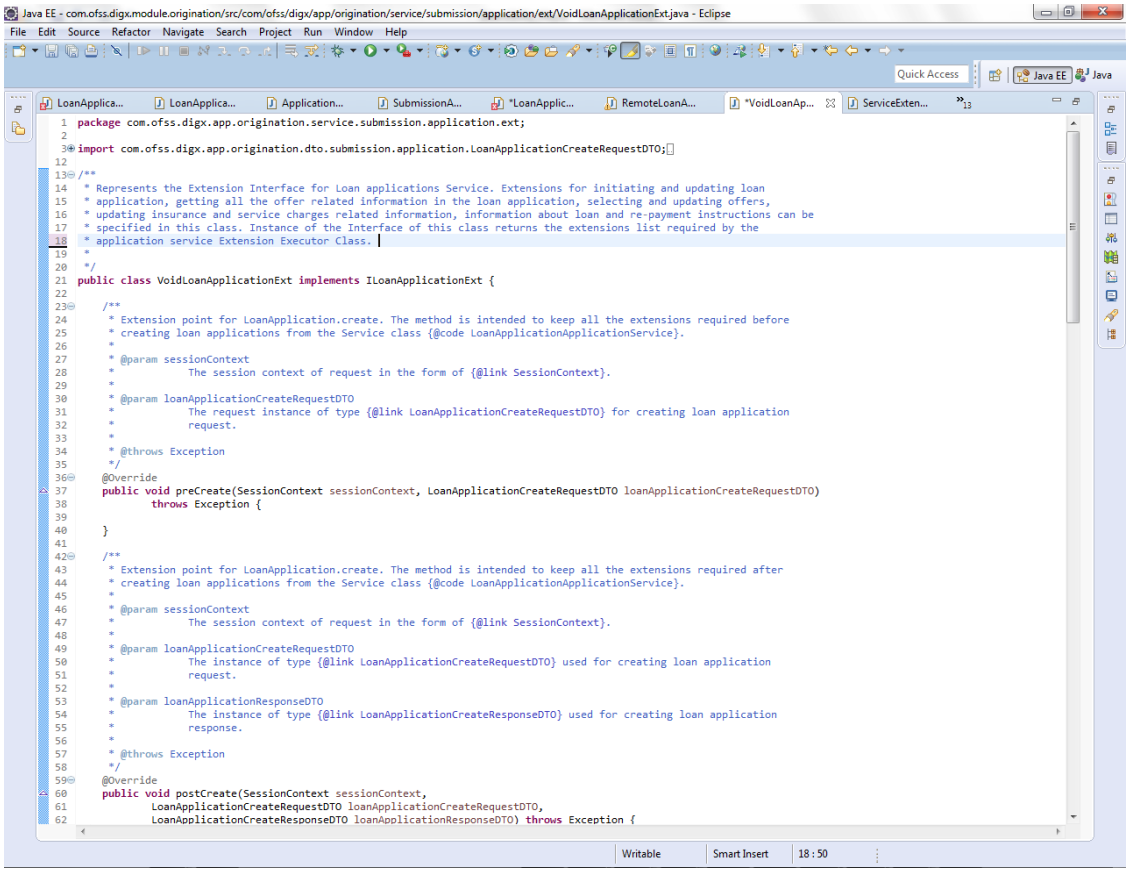
3.2.3 Default Extension (Void Extension)

This topic provides information on **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

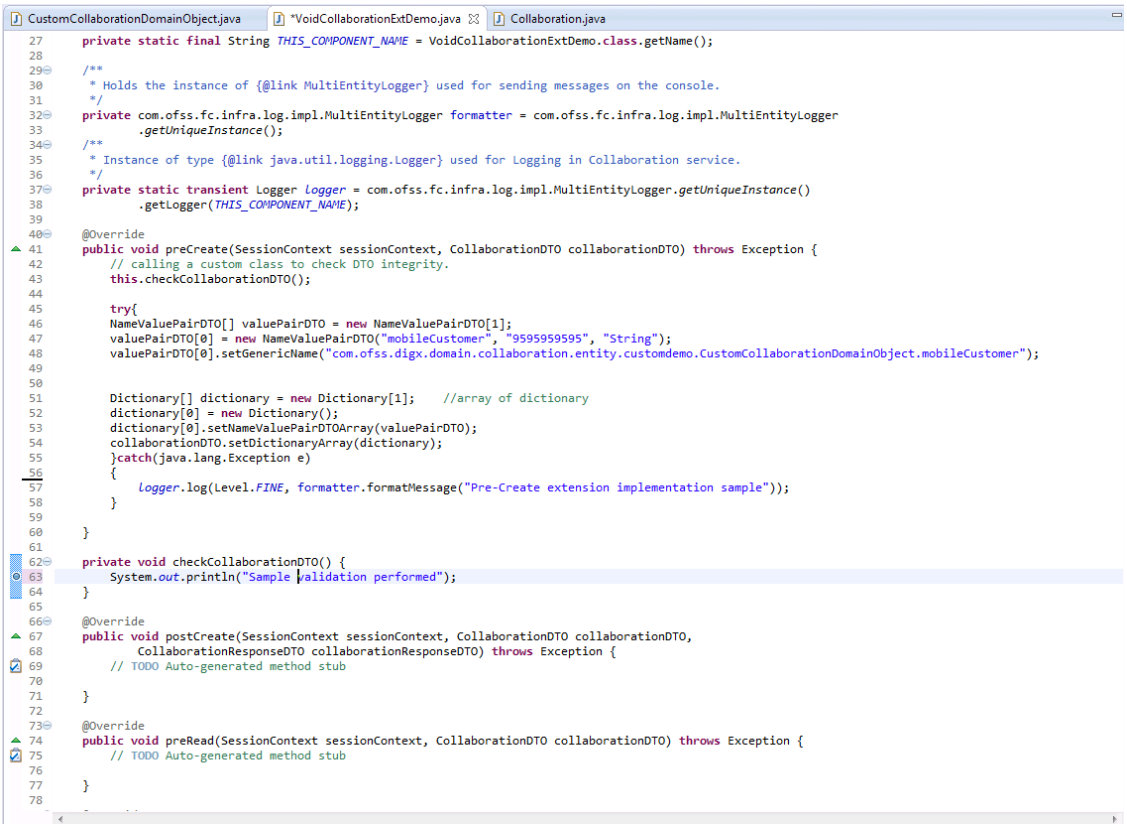



```
1  package com.ofss.digx.app.origination.service.submission.application.ext;
2
3  import com.ofss.digx.app.origination.dto.submission.application.LoanApplicationCreateRequestDTO;
12
13 /**
14  * Represents the Extension Interface for Loan applications Service. Extensions for initiating and updating loan
15  * application, getting all the offer related information in the loan application, selecting and updating offers,
16  * updating insurance and service charges related information, information about loan and re-payment instructions can be
17  * specified in this class. Instance of the Interface of this class returns the extensions list required by the
18  * application service Extension Executor Class.
19  *
20  */
21 public class VoidLoanApplicationExt implements ILoanApplicationExt {
22
23     /**
24      * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
25      * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
26      *
27      * @param sessionContext
28      *            The session context of request in the form of {@link SessionContext}.
29      *
30      * @param loanApplicationCreateRequestDTO
31      *            The request instance of type {@link LoanApplicationCreateRequestDTO} for creating loan application
32      *            request.
33      *
34      * @throws Exception
35      */
36     @Override
37     public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
38             throws Exception {
39
40     }
41
42     /**
43      * Extension point for LoanApplication.create. The method is intended to keep all the extensions required after
44      * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
45      *
46      * @param sessionContext
47      *            The session context of request in the form of {@link SessionContext}.
48      *
49      * @param loanApplicationCreateRequestDTO
50      *            The instance of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
51      *            request.
52      *
53      * @param loanApplicationResponseDTO
54      *            The instance of type {@link LoanApplicationCreateResponseDTO} used for creating loan application
55      *            response.
56      *
57      * @throws Exception
58      */
59     @Override
60     public void postCreate(SessionContext sessionContext,
61             LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
62             LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception {
```

3.2.4 Custom Extension

This topic provides information on **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.



> **Note:**
>
> The concept of ‘Dictionary’ is explained in detail in subsequent section.

3.2.5 Service Extension Configurations

This topic provides information on **Service Extension Configurations**.

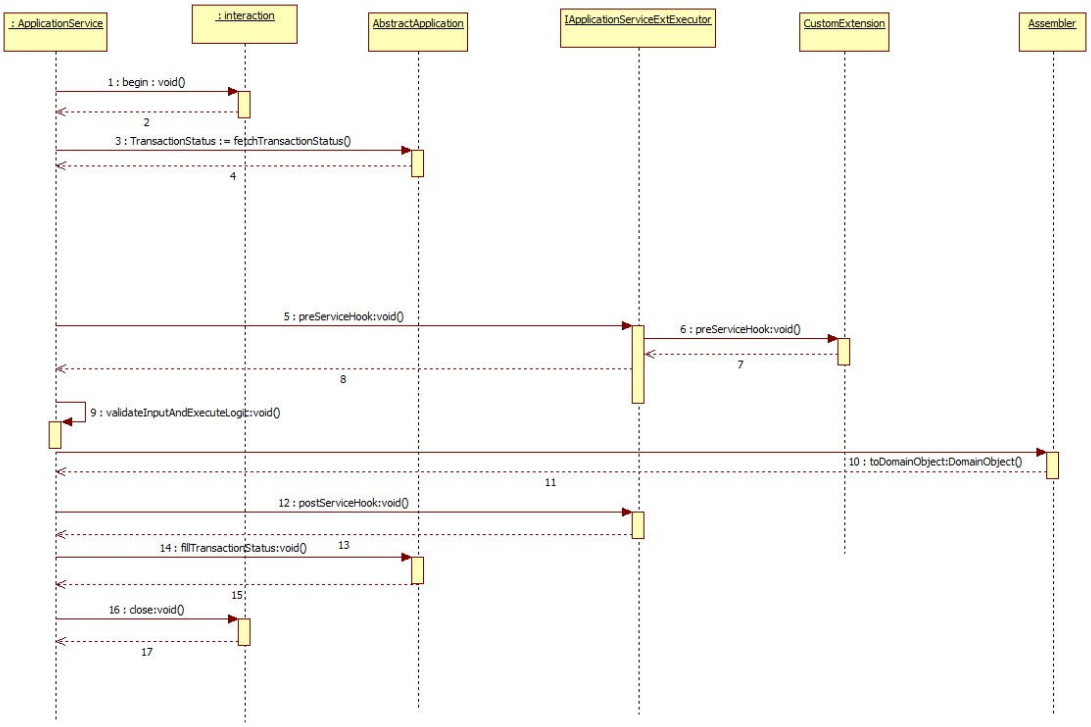
Set the property id and the property values in the `digx_fw_config_all_b` table. The property id will be the fully qualified name of the service and the value will be the fully qualified name of the custom extension created.

```
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);
```

3.2.6 Sequence of events in service extension

This topic provides information on **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.

3.3 Business Policy

This topic provides information on **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

For more information on fields, refer to the field description table.

Table 3-3 Field Level Annotation

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.

- Adding new business policy
  This topic provides information on **Adding new business policy**.
- Extending existing business policy
  This topic provides information on **Extending existing business policy**.

3.3.1 Adding new business policy

This topic provides information on **Adding new business policy**.

Customization developer can add new business policy for new or existing services. System support multiple business policies for a single service.

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. Steps for creating a new business policy class:
   a. BusinessPolicy class must have constructor which accepts one parameter of type IBusinessPolicyDTO.
   b. BusinessPolicy class must also have a default no-args constructor.
   c. BusinessPolicy class must extend com.ofss.fc.framework.domain.policy.AbstractBusinessPolicy.
   d. BusinessPolicy class must implement the validatePolicy() method.

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.

Below are the annotations used while creating a new business policy

For more information on fields, refer to the field description table.

Table 3-4 Annotations

Annotation	Description
@Custom	The @Custom Annotation signifies that the business policy is customization from the vendor, this is mandatory for every new business policy created by the vendor. In any of the Custom business policy if **overrideAll** is set to true, then it will make sure no base business policy will be loaded for all services mentioned in @TargetService of that custom business policy.
@TargetServices	The @TargetServices annotation must include all the @TargetService that the business policy needs to target.
@TargetService	Each TargetService must include a serviceID (String) specifying the service intended for the current Business Policy. It can optionally include @Priority annotation.
@Priority	The Priority annotation is optional and **defaults to a value of 100**. If a different value is desired for a service, then the Priority should be explicitly set.

3. The @Priority annotation is used to give a priority to the business policy for a particular service. The business policies are executed in order of lower to higher priority for a given service. If a new Custom BusinessPolicy is created by giving appropriate priority in the @TargetServices desired order of execution can be achieved. Please note that a @Custom business policy that targets the same service and has the same priority as @Base business policy will override and suppress the @Base business policy.
4. Use of **isPolicyToBeValidated()** method
   In case multiple business policies configured for one service then policy execution can be controlled by overriding isPolicyToBeValided() method in CustomBusinessPolicy class.

   By default, all Business Policies configured service provider configurator file in META-INF/services will be executed as isPolicyToBeValidated() method in AbstractBusinessPolicy will always return true given that @Priority of all businesspolicies are configured properly.To control the business policy validation based on data check, please override method isPolicyToBeValidated() in your BusinessPolicyClass.
5. **Configure new business policy(s)**. To Configure a new Business policy we have to add an entry of the fully-qualified name of the new business policy in META-INF\services\com.ofss.fc.framework.domain.policy.AbstractBusinessPolicy file.

Let us understand how to create Custom business policies with example If we want to create a business policy “CustomBusinessPolicyFirst” that should target the service “com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create”

```
@TargetServices(services = {
      @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create")
    })
@Custom
public class CustomBusinessPolicyFirst extends AbstractBusinessPolicy{
```

Now that we have created a new Business Policy, we have to register it by adding the business policies fully-qualified-name inside the META-INF/services/ com.ofss.fc.framework.domain.policy.AbstractBusinessPolicy

The @Priority annotation stores the priority of the business policy for a particular service, this determines in which order the business policies mapped to a service will execute. The @Priority annotation is optional if only one business policy is mapped to a service, but if multiple business policies are mapped to a single service then the @Priority annotation with unique values is mandatory, if not used as per instructions it can lead to unexpected behaviour. **The default value for @Priority is 100**.

Let us consider that we have to add a new business policy “CustomBusinessPolicySecond” which will target the service “com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create”, now we know that there is already a business policy “CustomBusinessPolicyFirst” mapped to the given service with default priority of 100. Now as per requirement if we want the “CustomBusinessPolicySecond” to be executed before or after “CustomBusinessPolicyFirst” we can assign priority less than 100 or greater than 100 respectively.

```
@TargetServices(services = {
      @TargetService(serviceid = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create",priority = @Priority(value = 200))
      })
@Custom
public class CustomBusinessPolicySecond extends AbstractBusinessPolicy{
```

Now as we have given Priority value higher than “CustomBusinessPolicySecond”, “CustomBusinessPolicyFirst” will execute first.

respective entries in META-INF/services/ com.ofss.fc.framework.domain.policy.AbstractBusinessPolicy must be made as explained above. Suppose if we want “CustomBusinessPolicyFirst” to also target service “com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.update” we can just simply add it as a @TargetService

```
@TargetServices(services = {
      @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create") ,
       @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.update")
 })
@Custom
public class CustomBusinessPolicyFirst extends AbstractBusinessPolicy{
```

Now lets understand how we can override and supress base business policies as per customization requirements. Lets consider these base policies for this example

CreateInternationalPayeeBankDetailsBusinessPolicy

```
@TargetServices(services = { @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create", priority = @Priority(value = 300)) }) ') })
@Base
public class CreateInternationalPayeeBankDetailsBusinessPolicy extends CreateInternationalPayeeBusinessPolicy {
```

CreateInternationalPayeeSwiftBankBusinessPolicy

```
@TargetServices(services = { @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create", priority = @Priority(value = 300)) })
@Base
public class CreateInternationalPayeeBankDetailsBusinessPolicy extends CreateInternationalPayeeBusinessPolicy {
```

CreateInternationalPayeeNationalClearingBankBusinessPolicy

```
@TargetServices(services = { @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create", priority = @Priority(value = 200)) })
@Base
public class CreateInternationalPayeeNationalClearingBankBusinessPolicy extends CreateInternationalPayeeBusinessPolicy {
```

Suppose we want to suppress CreateInternationalPayeeBankDetailsBusinessPolicy for this service com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create , this can be done by creating a separate @Custom business policy having an empty implementation and the target service should be the services that needs to be suppressed , also the priority given should be similar to the one in the @Base business policy for it to work accurately.

@CUSTOM business policies that target the same service as the @Base business policy with the same priority will override and suppress the @Base business policy

For Example:

In The Below example the Business Policy CustomBusinessPolicy will supress **CreateInternationalPayeeBankDetailsBusinessPolicy** for the service "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create"

```
@TargetServices(services = {
      @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create" ,priority = @Priority(value = 300))
 })
@Custom
public class CustomBusinessPolicy extends AbstractBusinessPolicy{
```

Now the business policy CustomBusinessPolicy will be called instead of CreateInternationalPayeeBankDetailsBusinessPolicy , this could be used to suppress the business policies.

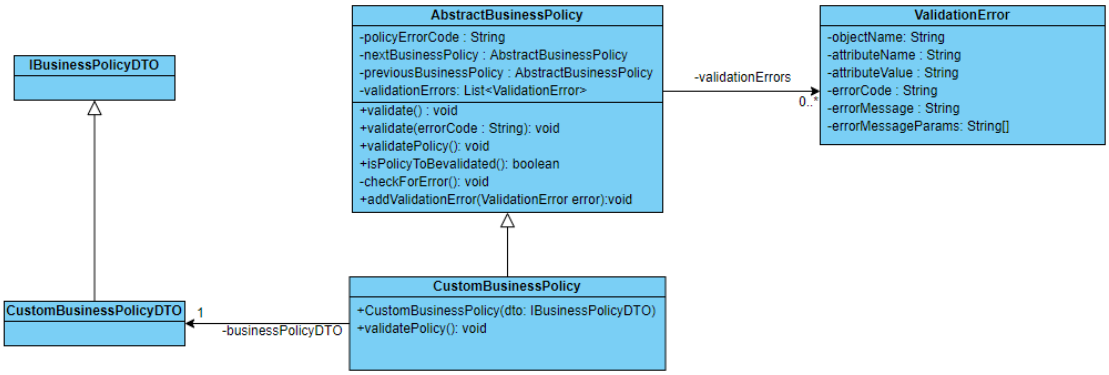
Suppose if we also want to Suppress/Override the existing **CreateInternationalPayeeSwiftBankBusinessPolicy** for service “com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create”**,** then we can simply add it as a @TargetService in **CustomBusinessPolicy**

```
@TargetServices(services = {
      @TargetService(serviceId = "com.ofss.digx.app.payment.service.payee.v3.InternationalPayee.create" ,priority = @Priority(value = 300))
 })
@Custom
public class CustomBusinessPolicy extends AbstractBusinessPolicy{
```

Let us assume there is a situation where there are 4 base business policies A,B,C,D that target services S1 and S2, suppose there is customization requirement to suppress all the base business policies of services S1 and S2, so instead of using the above mentioned method of overriding business policies there is an easier alternative available for this particular use case, we can set the **overrideAll** as true in @Custom, this will enable us to override all the base business policies of the services mentioned in @TargetService annotation.

```
@TargetServices(services = {
        @TargetService(serviceId = "S1"),
        @TargetService(serviceId = "S2")
 })
@Custom(overrideAll = true)
public class CustomBusinessPolicy extends AbstractBusinessPolicy{
```

The class diagram for new custom business policy.



3.3.2 Extending existing business policy

This topic provides information on **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.

As we are creating a new business policy extending the existing business policy, it is also required to note that if the existing business policy needs to be suppressed and new business policy should work for a particular service then steps mentioned in earlier section for suppressing a business policy should be followed.

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.

3.4 Dictionary

This topic provides information on **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)

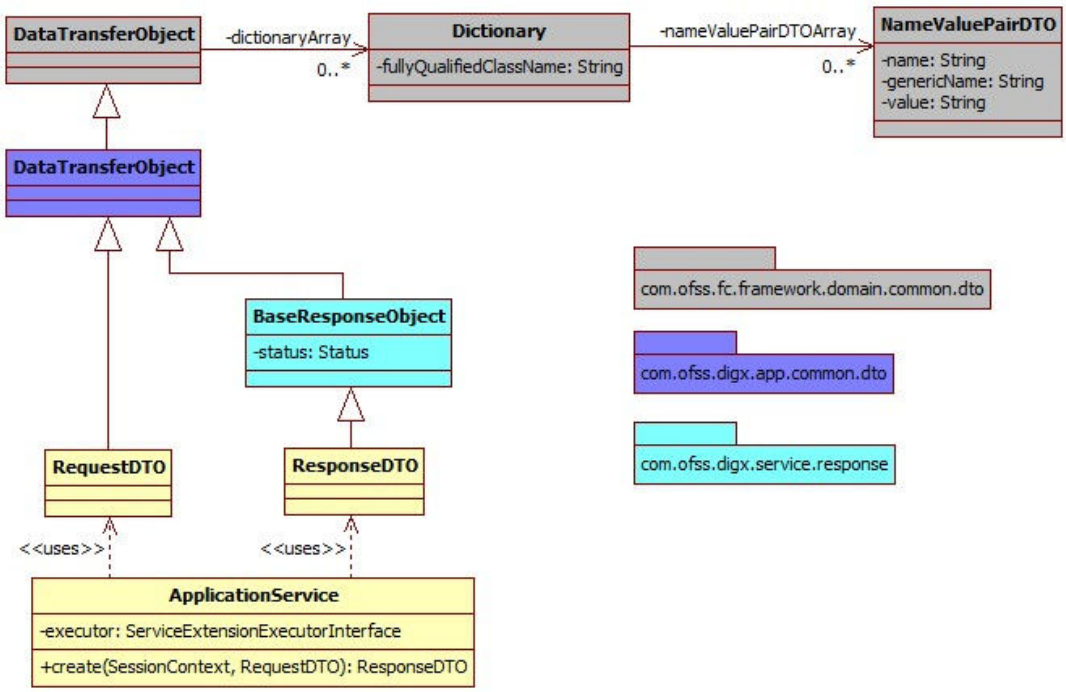
Data transfer object (DTO) is a design pattern used to transfer data between an external system and the application service. All the information may be wrapped in a single DTO containing all the details and passed as input request as well as returned as an output response. The client can then invoke 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.

```
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.

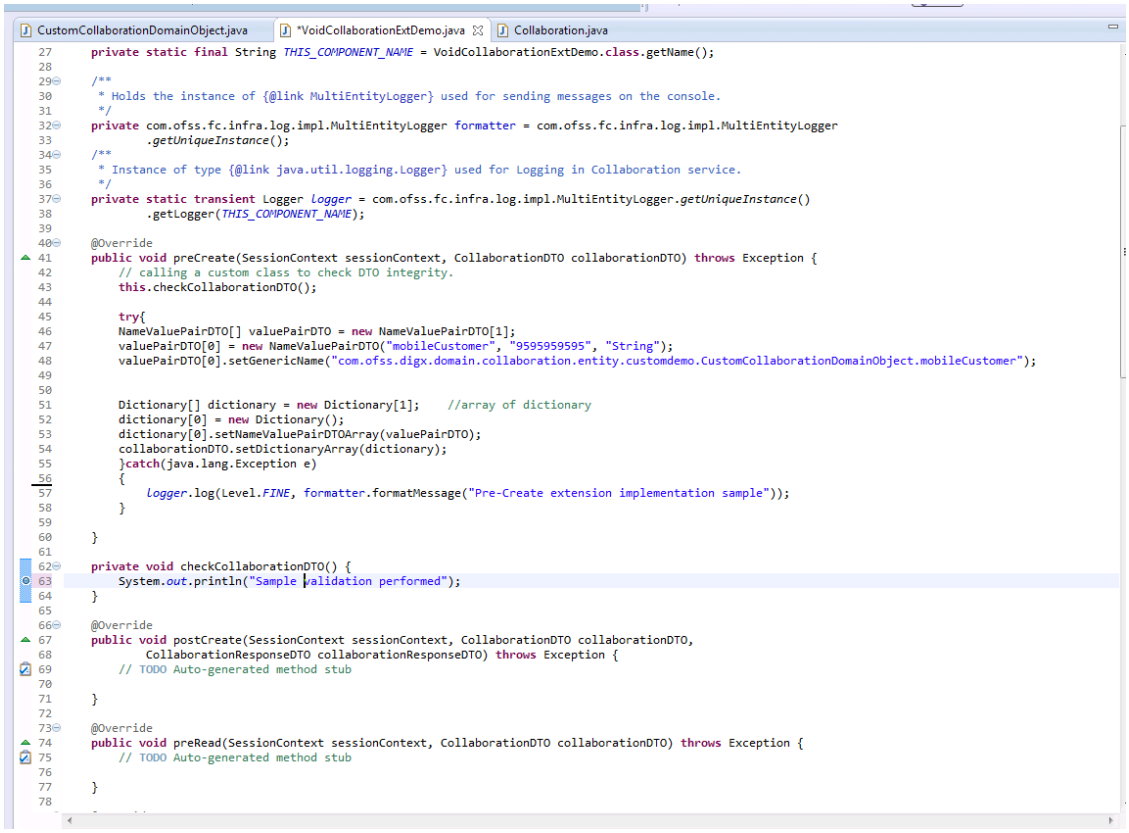


Dictionary class looks like

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.



3.5 Domain Extensions

This topic provides information on **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.

- Custom Domain Objects
  This topic provides information on **Custom Domain Objects**.
- Adding New Domain
  This topic provides information on **Adding New Domain**.

3.5.1 Custom Domain Objects

This topic provides information on **Custom Domain Objects**.

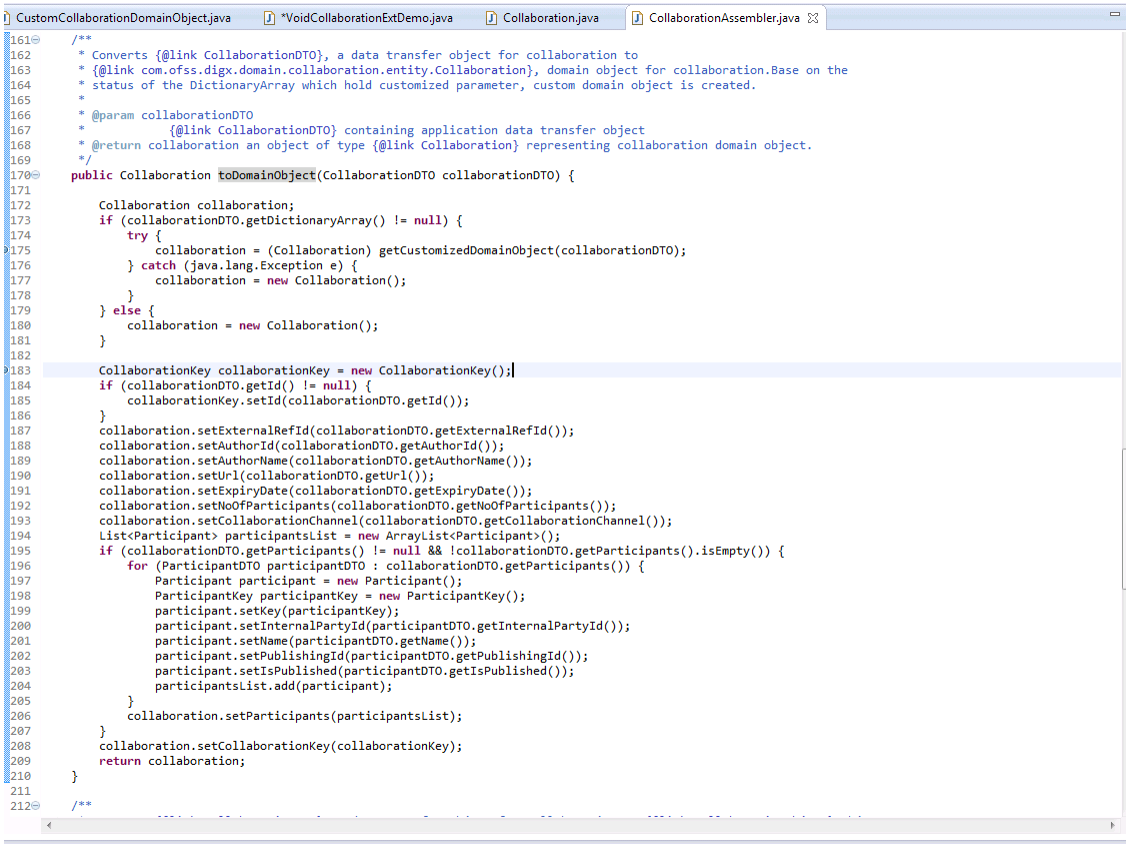
OBDX framework (leveraging undelaying OBP infrastructure) provides a standard mechanism to customize the domain objects that are provided out of the box. The Dictionary object plays an important role in this mechanism.

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.

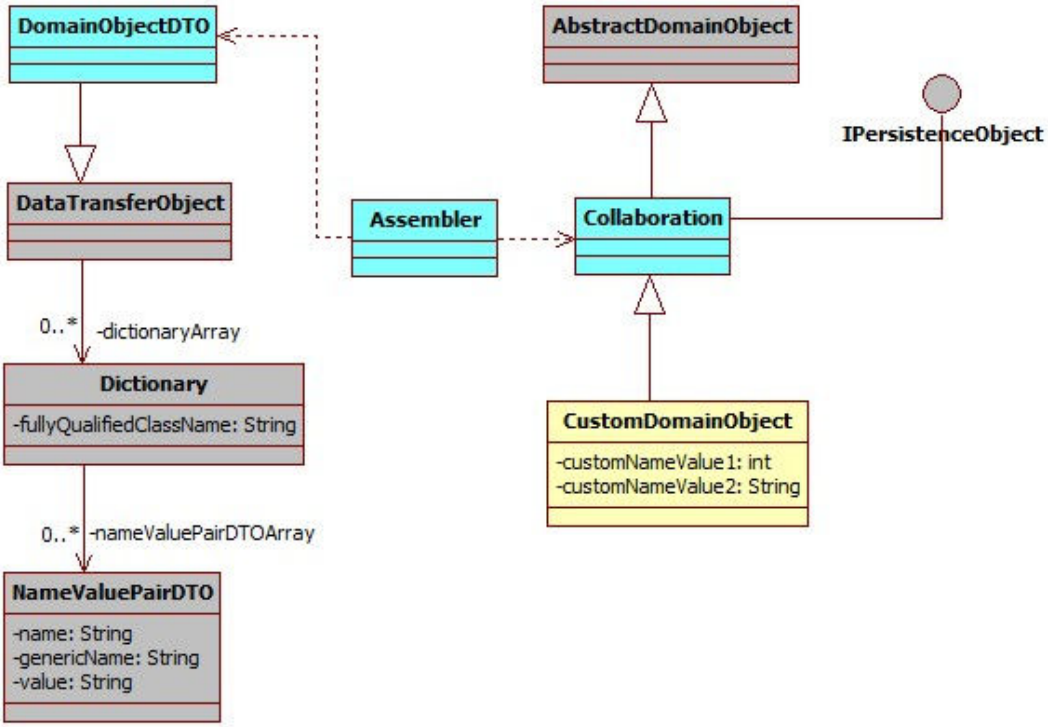



```
161    /**
162     * Converts {@link CollaborationDTO}, a data transfer object for collaboration to
163     * {@link com.ofss.digx.domain.collaboration.entity.Collaboration}, domain object for collaboration.Base on the
164     * status of the DictionaryArray which hold customized parameter, custom domain object is created.
165     *
166     * @param collaborationDTO
167     *            {@link CollaborationDTO} containing application data transfer object
168     * @return collaboration an object of type {@link Collaboration} representing collaboration domain object.
169     */
170    public Collaboration toDomainObject(CollaborationDTO collaborationDTO) {
171
172        Collaboration collaboration;
173        if (collaborationDTO.getDictionaryArray() != null) {
174            try {
175                collaboration = (Collaboration) getCustomizedDomainObject(collaborationDTO);
176            } catch (java.lang.Exception e) {
177                collaboration = new Collaboration();
178            }
179        } else {
180            collaboration = new Collaboration();
181        }
182
183        CollaborationKey collaborationKey = new CollaborationKey();
184        if (collaborationDTO.getId() != null) {
185            collaborationKey.setId(collaborationDTO.getId());
186        }
187        collaboration.setExternalRefId(collaborationDTO.getExternalRefId());
188        collaboration.setAuthorId(collaborationDTO.getAuthorId());
189        collaboration.setAuthorName(collaborationDTO.getAuthorName());
190        collaboration.setUrl(collaborationDTO.getUrl());
191        collaboration.setExpiryDate(collaborationDTO.getExpiryDate());
192        collaboration.setNoOfParticipants(collaborationDTO.getNoOfParticipants());
193        collaboration.setCollaborationChannel(collaborationDTO.getCollaborationChannel());
194        List<Participant> participantsList = new ArrayList<Participant>();
195        if (collaborationDTO.getParticipants() != null && !collaborationDTO.getParticipants().isEmpty()) {
196            for (ParticipantDTO participantDTO : collaborationDTO.getParticipants()) {
197                Participant participant = new Participant();
198                ParticipantKey participantKey = new ParticipantKey();
199                participant.setKey(participantKey);
200                participant.setInternalPartyId(participantDTO.getInternalPartyId());
201                participant.setName(participantDTO.getName());
202                participant.setPublishingId(participantDTO.getPublishingId());
203                participant.setIsPublished(participantDTO.getIsPublished());
204                participantsList.add(participant);
205            }
206            collaboration.setParticipants(participantsList);
207        }
208        collaboration.setCollaborationKey(collaborationKey);
209        return collaboration;
210    }
211
212    /**
```

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:

CustomCollaborationDomainObject.java | *VoidCollaborationExtDemo.java | Collaboration.java | CollaborationAssembler.java

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

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

/**
 * custom domain objet 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.entit
y.submission
.lending.application.ext.Application', 'N', 'asdf', 'asdf', 'asdf', '',
'asdf', '', 'Y', 1);
```

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

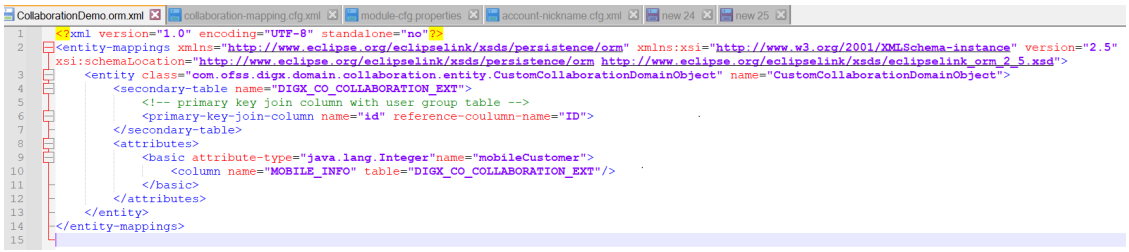
- `CATEGORY_ID` : “CustomizedAbstractDomainObjectConfig”
- `PROP_VALUE`:” CLASS NAME of the class implementing the custom domain object ”
- `PROP_ID`:” CLASS NAME of the DomainObject”.

ORM Mapping

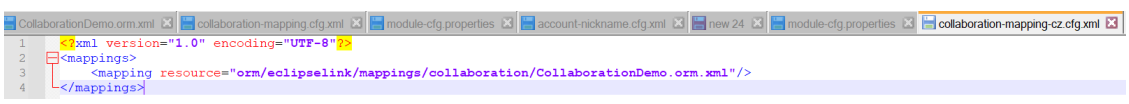
If this domain needs to be persisted in local database, then you need to create Eclipse link ORM mapping to map fields in the domain to database table. Follow these steps:

- Create new ORM file to handle Customized Domain Object.
- This ORM file should contain entries for all custom columns, which are present in the extension domain.
- The extension domain table will be a secondary table, which will have a primary key join column with the base domain.
- Add an entry for this ORM XML in the mapping configuration XML
- Create new table corresponds to newly created Domain Object.

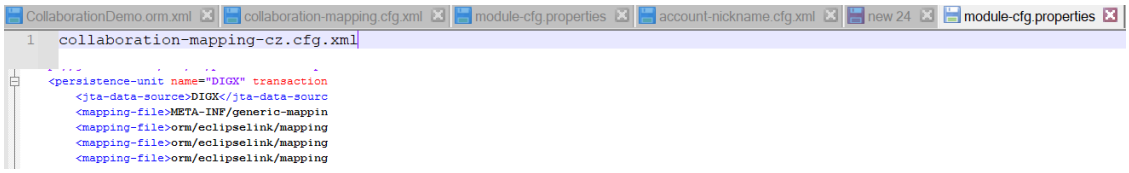
Newly created ORM file will look like (CollaborationDemo.orm.xml):



Now add the newly created mapping ORM entry for the extension domain in your custom jar mapping configuration xml.:

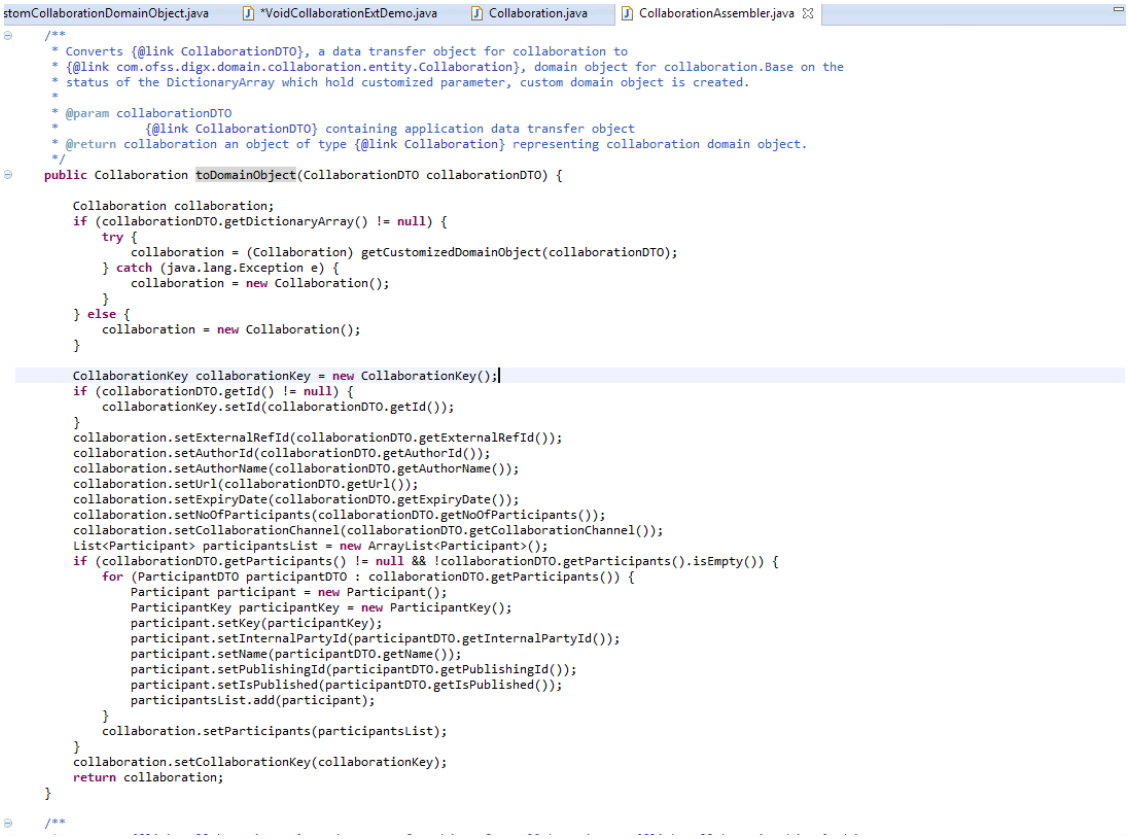


Create/Update “module-cfg.properties” file in your custom jar and add mapping configuration xml file name in it.

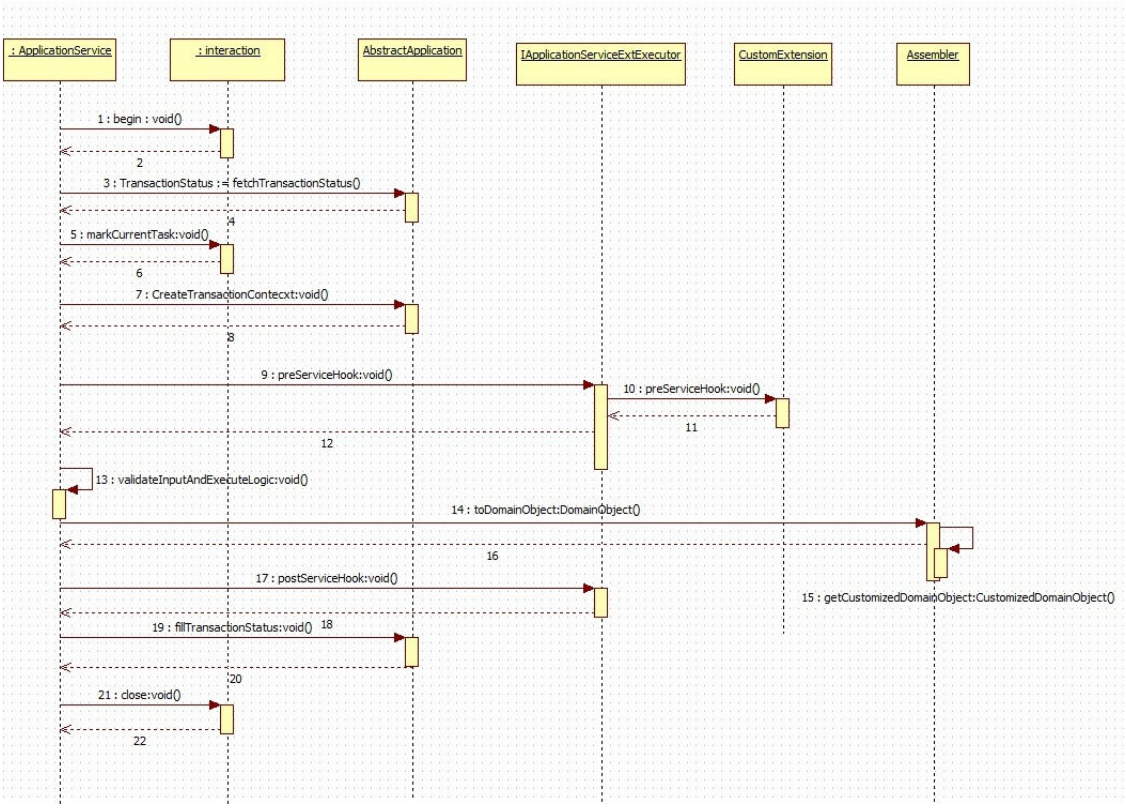


Here Assembler should fetch customized domain object. Following example shows Assembler calls **`getCustomizedDomainObject`** which returns customized domain object with mapping of **`nameValuePairDTOArray`** to this customized domain Object internally.

For example:



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)

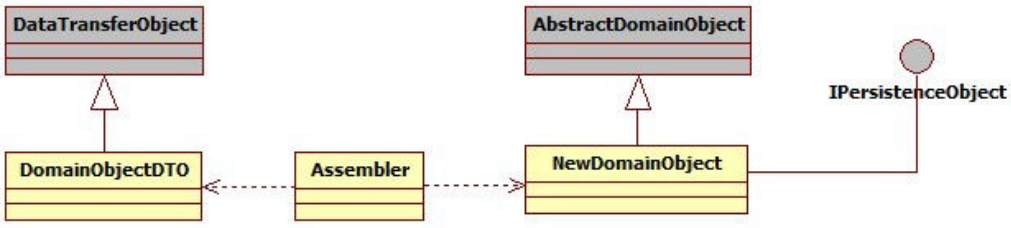
3.5.2 Adding New Domain

This topic provides information on **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');
```

3.6 Error Messages

This topic provides information on **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.

- Adding Error Message
  This topic provides information on **Adding Error Message**.
- Mapping Host Error Code To OBDX Error Code
  This topic provides information on **Mapping Host Error Code To OBDX Error Code**.

3.6.1 Adding Error Message

This topic provides information on **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.

3.6.2 Mapping Host Error Code To OBDX Error Code

This topic provides information on **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.

3.7 Adapter Tier

This topic provides information on **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.

- Service Provider Interface (SPI) Approach
  This topic provides information on **Service Provider Interface (SPI) Approach**.
- Adding a custom adapter
  This topic provides information on **Adding a custom adapter**.
- Host adapter extension to populate pagination informations
  This topic provides information on **Host adapter extension to populate pagination informations**.

3.7.1 Service Provider Interface (SPI) Approach

This topic provides information on **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

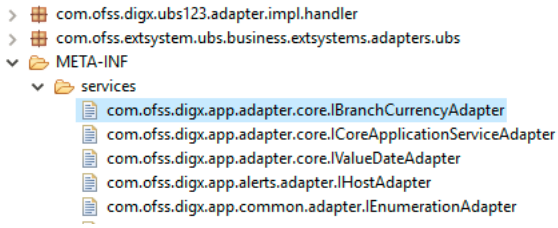
	PROP_ID	CATEGORY_ID	PROP_VALUE	FACTORY_SHIPPED_FLAG	PROP_COMMENTS
1	000	extxfaceadapterconfig	UBS12.4	N	(null)
2	001	extxfaceadapterconfig	OBP2502	N	(null)

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)	ofssuser	26-SEP-17 06.42.09.000000000 AM	ofssuser	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	ofssuser	26-SEP-17 06.42.09.000000000 AM
3	01	extxfaceadapterconfig	UBS12.4,ipm1.0	N	(null)	(null)	ofssuser	20-SEP-17 11.31.25.000000000 AM	ofssuser	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_DESCRI
PTION)
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_TEX
T,CREATED_BY,
CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_N
UMBER,EDITABLE,
CATEGORY_DESCRIPTION)values ('com.ofss.digx.app.loan.adapter.
ILoanAccountAdapter','extxfaceadapterconfig',
'com.ofss.digx.extxface.loan.impl.LoanAccountMockAdapter','N',null,'','ofssuse
r',
sysdate,'ofssuser', sysdate,'Y',1,'N',null);
```

3.7.2 Adding a custom adapter

This topic provides information on **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.

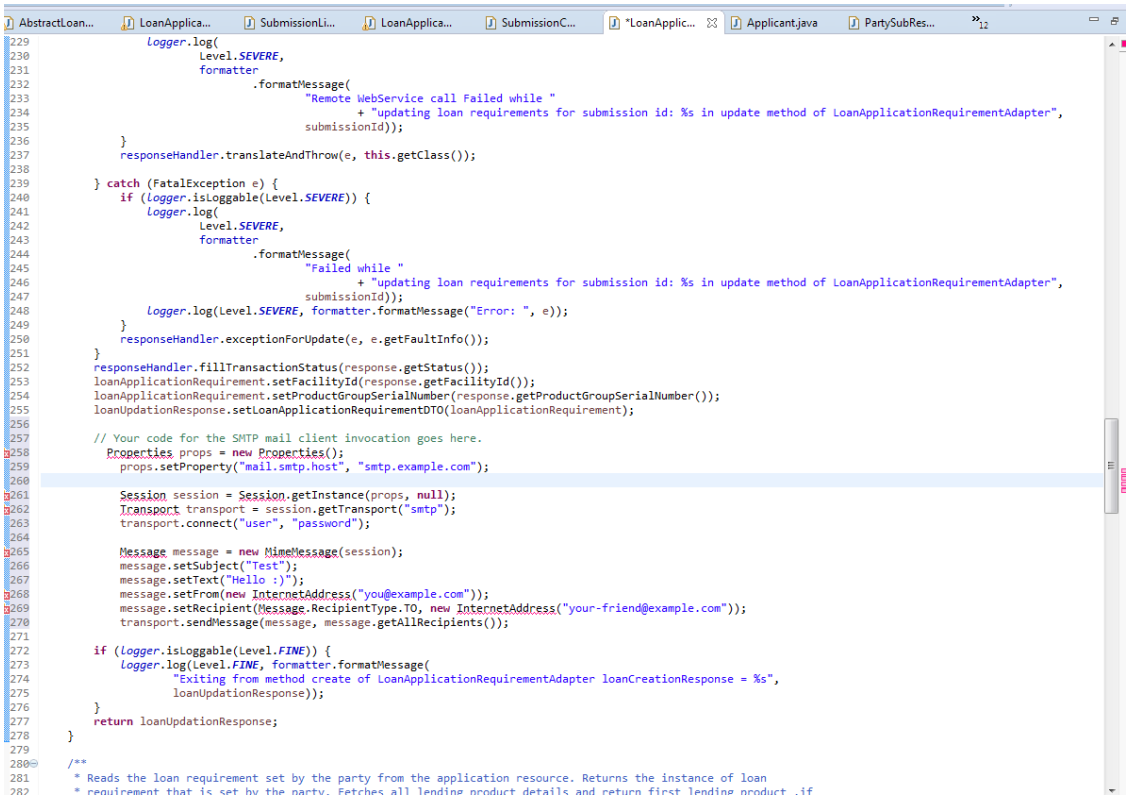
```
/*Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.*/
package com.ofss.fc.app.adapter.party;

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:



```
            logger.log(
                    Level.SEVERE,
                    formatter
                            .formatMessage(
                                    "Remote WebService call Failed while "
                                            + "updating loan requirements for submission id: %s in update method of LoanApplicationRequirementAdapter",
                                    submissionId));
            }
            responseHandler.translateAndThrow(e, this.getClass());

        } catch (FatalException e) {
            if (logger.isLoggable(Level.SEVERE)) {
                logger.log(
                        Level.SEVERE,
                        formatter
                                .formatMessage(
                                        "Failed while "
                                                + "updating loan requirements for submission id: %s in update method of LoanApplicationRequirementAdapter",
                                        submissionId));
                logger.log(Level.SEVERE, formatter.formatMessage("Error: ", e));
            }
            responseHandler.exceptionForUpdate(e, e.getFaultInfo());
        }
        responseHandler.fillTransactionStatus(response.getStatus());
        loanApplicationRequirement.setFacilityId(response.getFacilityId());
        loanApplicationRequirement.setProductGroupSerialNumber(response.getProductGroupSerialNumber());
        loanUpdationResponse.setLoanApplicationRequirementDTO(loanApplicationRequirement);

        // Your code for the SMTP mail client invocation goes here.
          Properties props = new Properties();
            props.setProperty("mail.smtp.host", "smtp.example.com");

            Session session = Session.getInstance(props, null);
            Transport transport = session.getTransport("smtp");
            transport.connect("user", "password");

            Message message = new MimeMessage(session);
            message.setSubject("Test");
            message.setText("Hello :)");
            message.setFrom(new InternetAddress("you@example.com"));
            message.setRecipient(Message.RecipientType.TO, new InternetAddress("your-friend@example.com"));
            transport.sendMessage(message, message.getAllRecipients());

        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, formatter.formatMessage(
                    "Exiting from method create of LoanApplicationRequiremntAdapter loanCreationResponse = %s",
                    loanUpdationResponse));
        }
        return loanUpdationResponse;
    }

    /**
     * Reads the loan requirement set by the party from the application resource. Returns the instance of loan
     * requirement that is set by the party. Fetches all lending product details and return first lending product .if
```

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);
```

3.7.3 Host adapter extension to populate pagination informations

This topic provides information on **Host adapter extension to populate pagination informations**.

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.

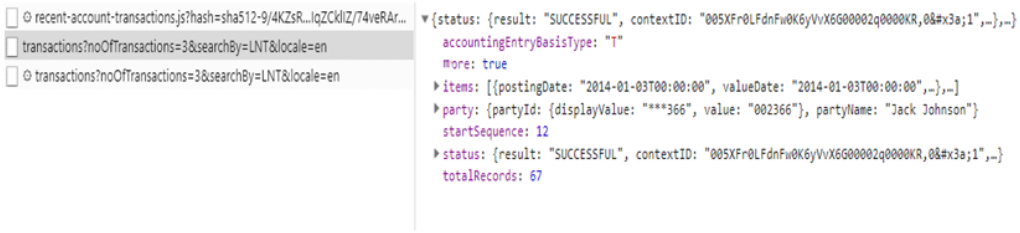
To use the above extension following steps need to be executed.

- The response DTO of service should implement `'com.ofss.digx.app.dto.Ipaginable'` 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:



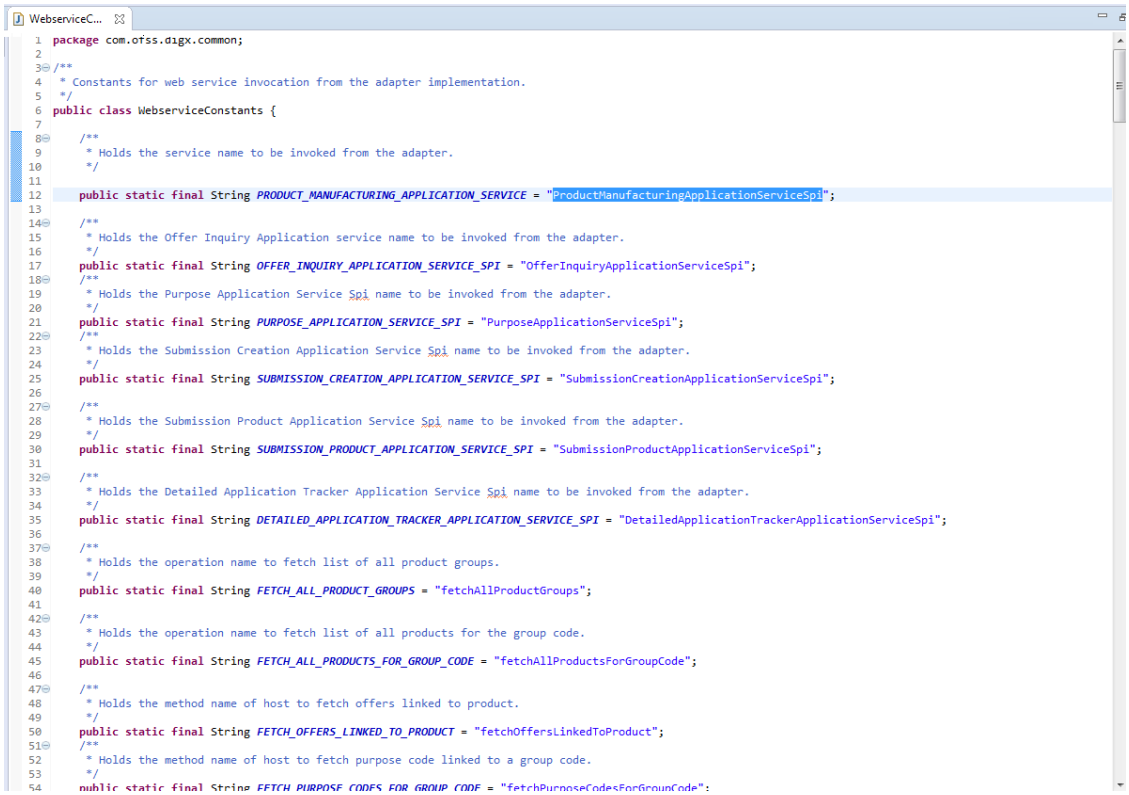
3.8 Outbound web service extensions

This topic provides information on **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:



```
WebserviceC...
1  package com.ofss.digx.common;
2
3  /**
4   * Constants for web service invocation from the adapter implementation.
5   */
6  public class WebserviceConstants {
7
8      /**
9       * Holds the service name to be invoked from the adapter.
10      */
11
12     public static final String PRODUCT_MANUFACTURING_APPLICATION_SERVICE = "ProductManufacturingApplicationServiceSpi";
13
14     /**
15      * Holds the Offer Inquiry Application service name to be invoked from the adapter.
16      */
17     public static final String OFFER_INQUIRY_APPLICATION_SERVICE_SPI = "OfferInquiryApplicationServiceSpi";
18     /**
19      * Holds the Purpose Application Service Spi name to be invoked from the adapter.
20      */
21     public static final String PURPOSE_APPLICATION_SERVICE_SPI = "PurposeApplicationServiceSpi";
22     /**
23      * Holds the Submission Creation Application Service Spi name to be invoked from the adapter.
24      */
25     public static final String SUBMISSION_CREATION_APPLICATION_SERVICE_SPI = "SubmissionCreationApplicationServiceSpi";
26
27     /**
28      * Holds the Submission Product Application Service Spi name to be invoked from the adapter.
29      */
30     public static final String SUBMISSION_PRODUCT_APPLICATION_SERVICE_SPI = "SubmissionProductApplicationServiceSpi";
31
32     /**
33      * Holds the Detailed Application Tracker Application Service Spi name to be invoked from the adapter.
34      */
35     public static final String DETAILED_APPLICATION_TRACKER_APPLICATION_SERVICE_SPI = "DetailedApplicationTrackerApplicationServiceSpi";
36
37     /**
38      * Holds the operation name to fetch list of all product groups.
39      */
40     public static final String FETCH_ALL_PRODUCT_GROUPS = "fetchAllProductGroups";
41
42     /**
43      * Holds the operation name to fetch list of all products for the group code.
44      */
45     public static final String FETCH_ALL_PRODUCTS_FOR_GROUP_CODE = "fetchAllProductsForGroupCode";
46
47     /**
48      * Holds the method name of host to fetch offers linked to product.
49      */
50     public static final String FETCH_OFFERS_LINKED_TO_PRODUCT = "fetchOffersLinkedToProduct";
51     /**
52      * Holds the method name of host to fetch purpose code linked to a group code.
53      */
54     public static final String FETCH_PURPOSE_CODES_FOR_GROUP_CODE = "fetchPurposeCodesForGroupCode";
```

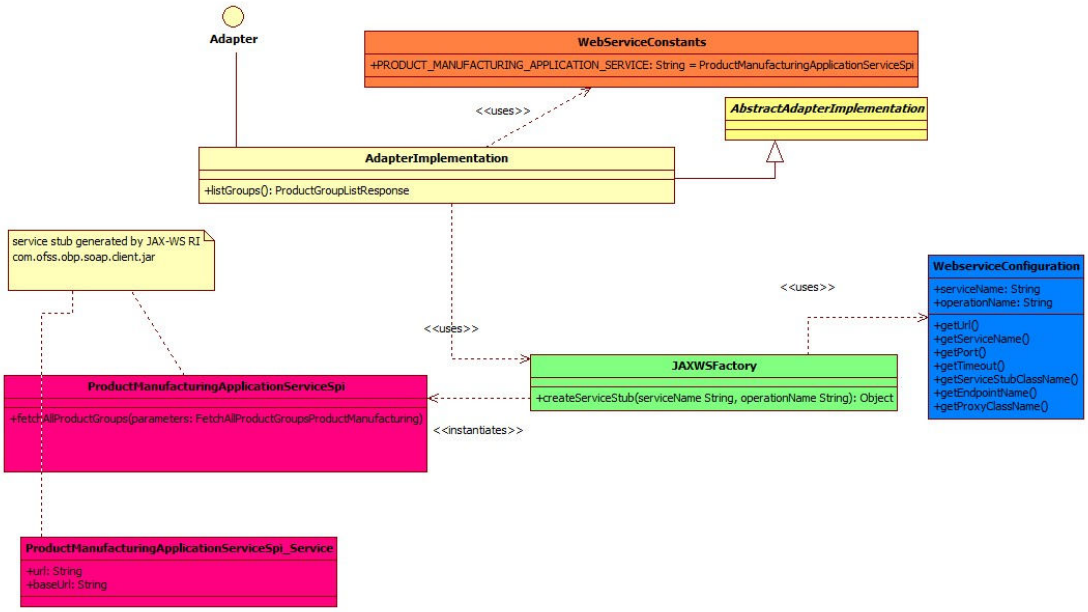
Web service configuration

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



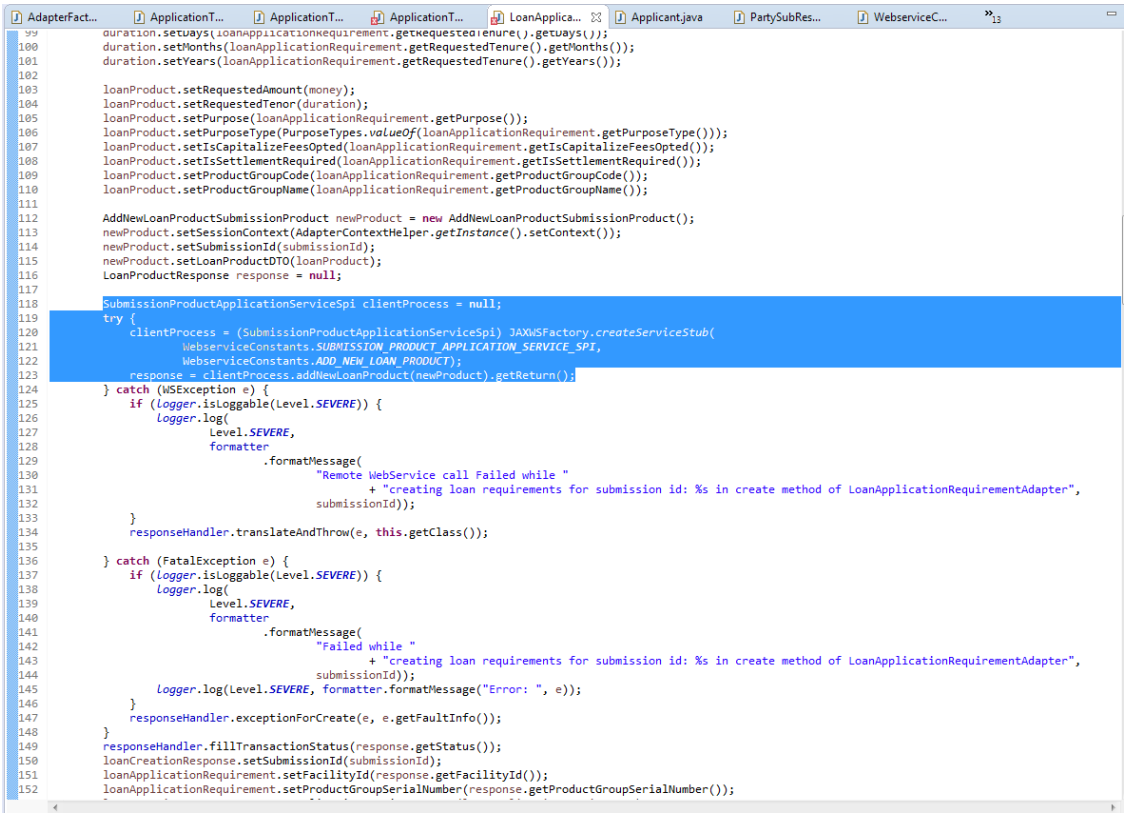
Client Jar

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.

For example:



```
        duration.setDays(loanApplicationRequirement.getRequestedTenure().getDays());
        duration.setMonths(loanApplicationRequirement.getRequestedTenure().getMonths());
        duration.setYears(loanApplicationRequirement.getRequestedTenure().getYears());

        loanProduct.setRequestedAmount(money);
        loanProduct.setRequestedTenor(duration);
        loanProduct.setPurpose(loanApplicationRequirement.getPurpose());
        loanProduct.setPurposeType(PurposeTypes.valueOf(loanApplicationRequirement.getPurposeType()));
        loanProduct.setIsCapitalizeFeesOpted(loanApplicationRequirement.getIsCapitalizeFeesOpted());
        loanProduct.setIsSettlementRequired(loanApplicationRequirement.getIsSettlementRequired());
        loanProduct.setProductGroupCode(loanApplicationRequirement.getProductGroupCode());
        loanProduct.setProductGroupName(loanApplicationRequirement.getProductGroupName());

        AddNewLoanProductSubmissionProduct newProduct = new AddNewLoanProductSubmissionProduct();
        newProduct.setSessionContext(AdapterContextHelper.getInstance().setContext());
        newProduct.setSubmissionId(submissionId);
        newProduct.setLoanProductDTO(loanProduct);
        LoanProductResponse response = null;

        SubmissionProductApplicationServiceSpi clientProcess = null;
        try {
            clientProcess = (SubmissionProductApplicationServiceSpi) JAXWSFactory.createServiceStub(
                    WebserviceConstants.SUBMISSION_PRODUCT_APPLICATION_SERVICE_SPI,
                    WebserviceConstants.ADD_NEW_LOAN_PRODUCT);
            response = clientProcess.addNewLoanProduct(newProduct).getReturn();
        } catch (WSException e) {
            if (logger.isLoggable(Level.SEVERE)) {
                logger.log(
                        Level.SEVERE,
                        formatter
                                .formatMessage(
                                        "Remote WebService call Failed while "
                                                + "creating loan requirements for submission id: %s in create method of LoanApplicationRequirementAdapter",
                                        submissionId));
            }
            responseHandler.translateAndThrow(e, this.getClass());

        } catch (FatalException e) {
            if (logger.isLoggable(Level.SEVERE)) {
                logger.log(
                        Level.SEVERE,
                        formatter
                                .formatMessage(
                                        "Failed while "
                                                + "creating loan requirements for submission id: %s in create method of LoanApplicationRequirementAdapter",
                                        submissionId));
                logger.log(Level.SEVERE, formatter.formatMessage("Error: ", e));
            }
            responseHandler.exceptionForCreate(e, e.getFaultInfo());
        }
        responseHandler.fillTransactionStatus(response.getStatus());
        loanCreationResponse.setSubmissionId(submissionId);
        loanApplicationRequirement.setFacilityId(response.getFacilityId());
        loanApplicationRequirement.setProductGroupSerialNumber(response.getProductGroupSerialNumber());
```

3.9 Security Customizations

This topic provides information on **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.

- Out of box seeding of policies
  This topic provides information on **Out of box seeding of policies**.

3.9.1 Out of box seeding of policies

This topic provides information on **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.

3.10 Taxonomy Validations

This topic provides information on **Taxonomy Validations**. For extensions in taxonomy validations, please refer to **Oracle Banking Digital Experience Taxonomy Configuration Guide**

3.11 Authentication Extensibility

This topic provides information on **Authentication Extensibility**.

OBDX now supports authentication extensibility for users based on enterprise roles. This can be done by following the below steps -

1. Need to write own Java class to implement authentication. Different classes can be used for different enterprise roles.
2. The custom classes must implement **com.ofss.digx.app.sms.handlers.credentials.ICredentialsManager**. Below methods need to be implemented -
   create - This method is to be used to create a user on the external system

   public void create(AbstractUser user) throws Exception;

   update - This method is to be used to update the user on the external system

   public boolean update(User user, boolean isPasswordSystemGenerated) throws Exception;

   verify - This method is to be used to authenticate the user on the external system

   public boolean verify(String name, String newPassword, String currentPassword) throws Exception;
3. The classes' fully qualified names have to be updated in **DIGX_FW_CONFIG_ALL_B** against prop_ids - **credentials_manager_administrator**, **credentials_manager_corporateuser**, **credentials_manager_retailuser**. By default all three currently have **com.ofss.digx.app.sms.handlers.credentials.LocalCredentialsManager** as prop_value.

3.12 Miscellaneous

This topic provides information on **Miscellaneous**.

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

- Task Configurations
  This topic provides information on **Task Configurations**.

3.12.1 Task Configurations

This topic provides information on **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 three parts:

1. **Module Name** : The first 2 alphabets representing the module to which the service in question belongs. e.g 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 three 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.

For more information on fields, refer to the field description table.

Table 3-5 List of aspects supported by OBDX framework

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
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.

Steps to register a task with OBDX:

1. 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.

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');
```

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.

Managing Task Aspects for Custom Requirements

Out of the Box behaviour:

Every DML service in OBDX application is associated with a Task. A Task in obdx can be associated to one or more Task Aspects like Approval, Limits, Two Factor Authentication etc.

Probable Requirement: In a special scenario while invoking a service a financial institution might want to toggle a task aspect for a task.

For such requirements we provide a configuration called taskEvaluatorFactories. This config can be checked in the application using below query.

For such requirements we provide a TaskEvaluatorFactory which is mapped to Tasks based on the “@TargetTask” in“@TargetTasks” annotationAnnotations needed to create a TaskEvaluatorFactory

For more information on fields, refer to the field description table.

Table 3-6 Annotation

Annotation	Description
@Custom	The @Custom Annotation signifies that the business policy is customization from the vendor, this is mandatory for every new business policy created by the vendor.
@TargetTasks	The @TargetTasks annotation must include all the @TargetTask that the business policy needs to target.
@TargetTask	Each TargetTask must include a TaskCode(String) specifying the task intended for the current Business Policy.

1. Every custom TaskEvaluatorFactory must have @Custom annotation signifying it as a customization from vendor.
2. Every TaskEvaluatorFactory must have a no-args default constructor.
3. Every TaskEvaluatorFactory is configured against a TASK_ID’s using @TargetTasks annotation which hold the @TargetTask. If the @Base TaskEvaluatorFactory and @Custom TaskEvaluatorFactor target the same task then for that task the base TaskEvaluatorFactory will be overridden and suppressed by the Custom TaskEvaluatorFactory.
4. The fully qualified name of every new TaskEvaluatorFactory must be added in new line of . META-INF/services/com.ofss.digx.framework

Every TaskEvaluatorfactory class configured here implements the below interface:

```
com.ofss.digx.framework.task.evaluator.ITaskEvaluatorFactory
```

which has the below method declaration

```
public ITaskEvaluator getEvaluator(TaskAspect taskAspect);
```

Inputs: TaskAspect for which the default behaviour is needed to be changed. For example if TaskAspect approvals need to be toggled then only that evaluator can be implemented and its instance can be returned. Rest all TaskAspects can continue using their default evaluators.

Output: Implementation of class ITaskEvaluator explained below.

that means TaskEvaluatorFactory takes a Task Aspect as an input and returns a TaskEvaluator.Every TaskEvaluator is a class that implements the ITaskEvaluator

which has the below method declaration

```
public String evaluateTaskCode(String taskCode,
List<Object> serviceInputs) throws Exception;
```

Inputs: taskCode - the current task code configured in the system as per the service invoked.

serviceInputs - the arguments passed to the first service called from rest. These arguments help in deducing the logic whether the special condition is met or not in which we wish to toggle the TaskAspect.

Output: String that is a new TaskCode(not the one passed as an input) for which we have configured the TaskAspect in a way different than the default taskcode(the one passed as an input to evaluateTaskCode).

Let us consider an example of. A TaskEvaluatorFactory:PeerToPeerPaymentTaskEvaluatorFactory is an implementation of Task Evaluator Factory which will be used for Task **PC_F_CPTP, PC_F_PTP**, this is avalaible in the base product.

```
@TargetTasks(tasks = {@TargetTask(taskId = "PC_F_CPTP"),
                      @TargetTask(taskId = "PC_F_PTP")
})
@Base
public class PeerToPeerPaymentTaskEvaluatorFactory implements ITaskEvaluatorFactory {
```

Overriding existing Task evaluator factory for a particular task

This can be done by just creating a custom class and adding @TargetTask with taskId of the taskfor which the Task Evaluator Factory needs to be overridden , this will suppress the @Base Task evaluator factory implementation The use case for this could be to change the existing functionality of the TaskEvaluator Factory .

Let us create custom Task Evauluator Factory

CustomPeerToPeerPaymentTaskEvaluatorFactory which will override existing PeerToPeerPaymentTaskEvaluatorFactory for taskId PC_F_CPTP.

```
@TargetTasks(tasks = {@TargetTask(taskId = "PC_F_CPTP") })
@Custom
public class CustomPeerToPeerPaymentTaskEvaluatorFactory  implements ITaskEvaluatorFactory {
```

So when TaskEvaluatorFactory is required for taskId “PC_F_CPTP” the CustomPeerToPeerPaymentTaskEvaluatorFactory will be loaded.

Hypothetical Sample Requirement:

OBDX application should not ask for the configured 2nd Factor Authentication in case of payments made for less than a pre-configured amount.

Process:

1. insert a new TaskCode in the application by making an entry in the table digx_cm_task.

2. configure the TaskAspects for this new task such that 2fa is disabled by making appropriate entries in the table digx_cm_task_aspects.
3. Write a TaskEvaluator as mentioned above such that with the help of serviceInputs it figures out that the amount getting transfered in this payment is less than pre-configured amount and hence returns the Task Code created in Step 1. If the amount is greater than the pre-configured amount, then it returns the task code passed as an input.
4. Write a TaskEvaluatorFactory as explained above. This new TaskEvaluatorFactory can extend the preconfigured(default) TaskEvalutorFactory such that for TaskAspects other than TWO_FACTOR_AUTHENTICATION it can return same TaskEvaluator as the preconfigured TaskEvalutorFactory. For TaskAspect TWO_FACTOR_AUTHENTICATION it returns the newly created TaskEvaluator written in Step 3.
5. Register fully qualified name of this TaskEvalutorFactory.in META-INF/services/ com.ofss.digx.framework.task.evaluator.ITaskEvaluatorFactory file.

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:

1. Ensure that the ‘limit’ aspect is configured in `DIGX_CM_TASK_ASPECTS` table and ENABLED column is updated as 'Y' for your task id.
2. Register taskEvaluatorFactory for your task code. Please refer the above steps for registering a taskEvalautorFactory.
   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.
- Every LimitDataEvaluator(AspectDataEvaluator) must have @Custom annotation signifying it as a customization from vendor.
- Every LimitDataEvaluator(AspectDataEvaluator) must have a no-args default constructor
- Every LimitDataEvaluator is configured against TASK_ID's using @TargetTasks annotation which hold the @TargetTask. **If the @Base LimitDataEvaluator and @Custom LimitDataEvaluator target the same task and the supportedAspects are the same then for that task the @Base LimitDataEvaluator will be overridden and suppressed by the @Custom LimitDataEvaluator.**
- The fully qualified name of every new TaskEvaluatorFactory must be added in new line of META-INF\services\com.ofss.digx.framework.evaluator.data.IAspectDataEvaluator file.

1. 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.

TargetEvaluator 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.

2. Register the TargetEvaluator coded in Step 4.

> **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 4. `<<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`.

Steps to Code an approval assembler for the new task. Annotations that are used are as follows:

For more information on fields, refer to the field description table.

Table 3-7 Annotation

Annotation	Description
@Custom	The @Custom Annotation signifies that the business policy is customization from the vendor, this is mandatory for every new business policy created by the vendor.
@TargetServices	The @TargetServices annotation must include all the @TargetService that the business policy needs to target.
@TargetService	Each TargetService must include a serviceID (String) specifying the service intended for the current Business Policy. It can optionally include @Priority annotation.

1. An approval assembler is a class that extends AbstractApprovalAssembler.
2. The new Approval Assembler class should contain @Custom annotation. The @Custom annotation is used to denote that the Approval assembler is customization from vendor.
3. It must also contain @TargetServices which contains @TargetService, mapping the approval assembler to different services The @TargetServices annotation holds different @TargetService that hold serviceID of a service.If a @Custom Assembler targets a service similar to the @Base Assembler then the @Custom assembler will override the @Base assembler for that service.
4. The new Approval Assembler must have a no-args default constructor.

5. For every new Approval assembler anEntry in META-INF\services\com.ofss.digx.framework.domain.transaction.assembler.AbstractApprovalAssembler is mandatory.

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 guidelinesto 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 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.

- 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 a new approval assembler and extend the existing com.ofss.digx.framework.domain.transaction.assembler.GenericDTOTransactionAssembler and map this approval assembler to your required service. This will cause the new assembler to inherit all the methods of the GenericDTOTransactionAssembler.

```
@TargetServices(services = {@TargetService(serviceId = "com.ofss.digx.app.security.service.authentication.maintenance.AuthenticationMaintenance")
})
@Custom
public class CustomApprovalAssembler extends GenericDTOTransactionAssembler <C, Transaction> {
```

It must have no-args default constructor and and entry must be made in META-INF\services\com.ofss.digx.framework.domain.transaction.assembler.AbstractApprovalAssembler

Lets us understand how to override the approval assemblers

Suppose we want to customize the assembler for "com.ofss.digx.app.approval.service.usergroup.UserGroup.create" service which uses UserGroupTransactionAssembler as seen below.

UserGroupTransactionAssembler

```
@TargetServices(services = {@TargetService(serviceId = "com.ofss.digx.app.approval.service.usergroup.UserGroup.create")
                @TargetService(serviceId = "com.ofss.digx.app.approval.service.usergroup.UserGroup.update")})
@Base
public class UserGroupTransactionAssembler extends AbstractApprovalAssembler<UserGroupDTO, Transaction> {
```

Create a new assembler “UserGroupCustomTransactionAssembler” that extends the AbstractApprovalAssembler and give it the @Custom Annotation to indicate it as a customization , also if **@Custom Assembler targets a service which is also a Target of @Base Assembler then for that particular service the @Custom assembler will override the @Base Assembler**

```
@TargetServices(services = {@TargetService(serviceId = "com.ofss.digx.app.approval.service.usergroup.UserGroup.create")
})

@Custom
public class UserGroupCustomTransactionAssembler extends AbstractApprovalAssembler <UserGroupDTO, Transaction> {
```

Also respective entries for the new UserGroupCustomTransactionAssembler needs to be made in META-INF/services/ **com.ofss.digx.framework.domain.transaction.assembler.AbstractApprovalAssembler**

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'`
   Annotations needed to create a AccountRelationshipEvaluator

   For more information on fields, refer to the field description table.

Table 3-8 Annotation

Annotations	Description
@Custom	The @Custom Annotation signifies that the business policy is customization from the vendor, this is mandatory for every new business policy created by the vendor.
@TargetTasks	The @TargetTasks annotation must include all the @TargetTask that the business policy needs to target.
@TargetTask	Each TargetTask must include a TaskCode(String) specifying the task intended for the current Business Policy.

1. Every custom AccountRelationshipEvaluator must have @Custom annotation signifying it as a customization from vendor.
2. Every AccountRelationshipEvaluator must have a no-args default constructor.
3. Every TaskEvaluatorFactory is configured against TASK_ID's using @TargetTasks annotation which hold the @TargetTask. If the @Base AccountRelationshipEvaluator and @Custom AccountRelationshipEvaluator target the same task then for that task the base AccountRelationshipEvaluator will be overridden and suppressed by the Custom AccountRelationshipEvaluator.
4. The fully qualified name of every new **AccountRelationshipEvaluator** must be added in new line of META-INF/services/ com.ofss.digx.app.accountrelationship.evaluator.mapping.IAccountRelationshipDataEvaluator file.

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'.

```
@TargetTasks(tasks = { @TargetTask(taskId = "TD_F_OTD")})
@Custom
public class TDAccountRelationshipEvaluator implements IAccountRelationshipDataEvaluator {
```

Also respective entries of name of the evaluator needs to be made in META-INF/services/com.ofss.digx.app.accountrelationship.evaluator.mapping.IAccountRelationshipDataEvaluator file.

This will register TDAccountRelationshipEvaluator for the task “TD_F_OTD”

Note that if there is a @Base(comes with base product) evaluator that targets the same task “TD_F_OTD” the above @Custom evaluator will override it.

Date Evaluators

They are used to get the date depending on the TaskCode. This could be used in scenarios where a different date evaluator implementation is required for different TaskCode as current date is specific to the transaction being performed.

eg: For transactions(such as international transfer) where third party hosts are used , different date evaluators can be used to get the current date from respective host.

Annotations needed to create a DateEvaluator

For more information on fields, refer to the field description table.

Table 3-9 Annotations

Annotation	Description
@Custom	The @Custom Annotation signifies that the business policy is customization from the vendor, this is mandatory for every new business policy created by the vendor.
@TargetTasks	The @TargetTasks annotation must include all the @TargetTask that the business policy needs to target.
@TargetTask	Each TargetTask must include a TaskCode(String) specifying the task intended for the current Business Policy.

1. Every custom DateEvaluator must have @Custom annotation signifying it as a customization from vendor.
2. Every DateEvaluator must have a no-args default constructor.
3. Every DateEvaluator is configured against TASK_ID’s using @TargetTasks annotation which hold the @TargetTask. If the @Base DateEvaluator and @Custom DateEvaluator target the same task then for that task the base DateEvaluator will be overridden and suppressed by the Custom DateEvaluator.
4. The fully qualified name of every new DateEvaluator must be added in new line of META-INF/services/com.ofss.digx.app.date.evaluator.AbstractDateEvaluator file
5. Every DateEvaluator must extend com.ofss.digx.app.date.evaluator.AbstractDateEvaluator;

4

Extensible Points in Approval

This topic provides information on **Extensible Points in Approval**. This article explains extensible points in Approval framework.

- Adding New Rule Criteria
  This topic provides information on **Adding New Rule Criteria**.

4.1 Adding New Rule Criteria

This topic provides information on **Adding New Rule Criteria**.

Every rule in the system is created against a TaskType. TaskType decides which Rule Criteria are to be associated with a Rule being created. Examples of existing Rule Criteria are Transaction, Account, Amount and Currency.

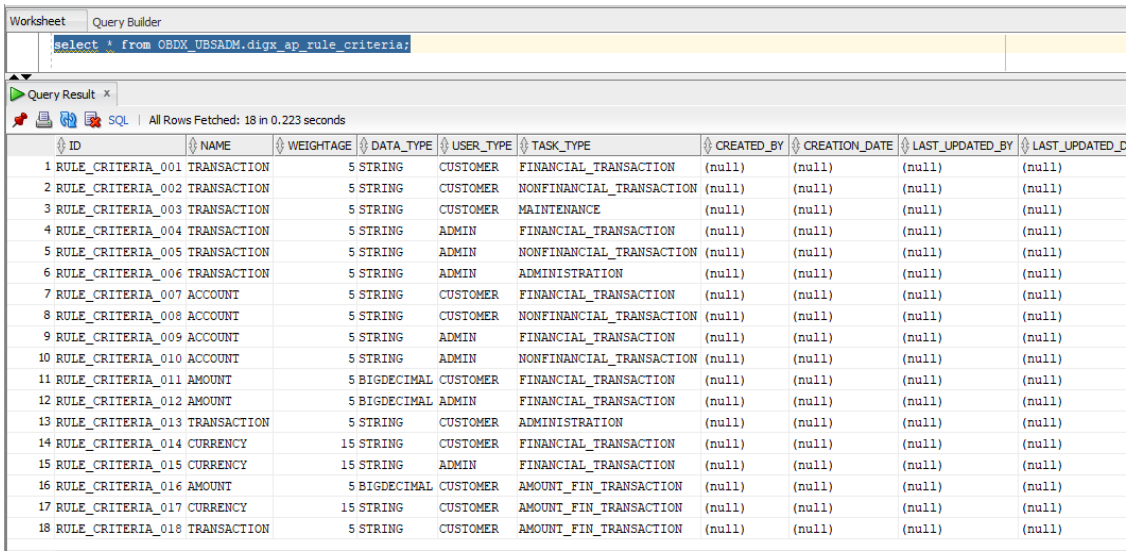
If the existing Rule Criteria does not meet your requirement, then a new Rule Criteria can be extended in the system by following the steps given below:

- Adding New Rule Criteria
  This topic provides information on **Adding New Rule Criteria**.
- Implementing a Rule Criteria Handler
  This topic provides information on **Implementing a Rule Criteria Handler**.
- Registering a Rule Criteria Handler
  This topic provides information on **Registering a Rule Criteria Handler**.

4.1.1 Adding New Rule Criteria

This topic provides information on **Adding New Rule Criteria**.

Add a new rule criteria in the Table `DIGX_AP_RULE_CRITERIA` shown below against the `TASK_TYPE` to which the customized Task belongs:



Worksheet | Query Builder

```
select * from OBDX_UBSADM.digx_ap_rule_criteria;
```

Query Result | SQL | All Rows Fetched: 18 in 0.223 seconds

	ID	NAME	WEIGHTAGE	DATA_TYPE	USER_TYPE	TASK_TYPE	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATED_D
1	RULE_CRITERIA_001	TRANSACTION	5	STRING	CUSTOMER	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
2	RULE_CRITERIA_002	TRANSACTION	5	STRING	CUSTOMER	NONFINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
3	RULE_CRITERIA_003	TRANSACTION	5	STRING	CUSTOMER	MAINTENANCE	(null)	(null)	(null)	(null)
4	RULE_CRITERIA_004	TRANSACTION	5	STRING	ADMIN	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
5	RULE_CRITERIA_005	TRANSACTION	5	STRING	ADMIN	NONFINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
6	RULE_CRITERIA_006	TRANSACTION	5	STRING	ADMIN	ADMINISTRATION	(null)	(null)	(null)	(null)
7	RULE_CRITERIA_007	ACCOUNT	5	STRING	CUSTOMER	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
8	RULE_CRITERIA_008	ACCOUNT	5	STRING	CUSTOMER	NONFINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
9	RULE_CRITERIA_009	ACCOUNT	5	STRING	ADMIN	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
10	RULE_CRITERIA_010	ACCOUNT	5	STRING	ADMIN	NONFINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
11	RULE_CRITERIA_011	AMOUNT	5	BIGDECIMAL	CUSTOMER	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
12	RULE_CRITERIA_012	AMOUNT	5	BIGDECIMAL	ADMIN	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
13	RULE_CRITERIA_013	TRANSACTION	5	STRING	CUSTOMER	ADMINISTRATION	(null)	(null)	(null)	(null)
14	RULE_CRITERIA_014	CURRENCY	15	STRING	CUSTOMER	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
15	RULE_CRITERIA_015	CURRENCY	15	STRING	ADMIN	FINANCIAL_TRANSACTION	(null)	(null)	(null)	(null)
16	RULE_CRITERIA_016	AMOUNT	5	BIGDECIMAL	CUSTOMER	AMOUNT_FIN_TRANSACTION	(null)	(null)	(null)	(null)
17	RULE_CRITERIA_017	CURRENCY	15	STRING	CUSTOMER	AMOUNT_FIN_TRANSACTION	(null)	(null)	(null)	(null)
18	RULE_CRITERIA_018	TRANSACTION	5	STRING	CUSTOMER	AMOUNT_FIN_TRANSACTION	(null)	(null)	(null)	(null)

4.1.2 Implementing a Rule Criteria Handler

This topic provides information on **Implementing a Rule Criteria Handler**.

For the newly created RuleCriteria mentioned in step above , create a `RuleCriteriaHandler` implementation. This class implements the interface named `com.ofss.digx.app.approval.service.rulecriteria.handler.IRuleCriteriaHandler`

Override the methods

- **addRuleCriteriaRelationships :** This method returns the list of RuleRuleCriteriaRelationshipDTO to be added as a part of the newly created RuleCriteria to the rule being created for the TaskType to which the customized task belongs.
- **getRuleCriteriaMultiplierForRule:** returns a multiplier (datatype :double) which gives precedence to a rule over other rule in case both the rules are applicable for a particular instance of a transaction.

> **Note:**
>
> While implementing Rule Criteria Handler make sure that it is implemented in a way that it does not impact existing Tasks in the system belonging to the TaskType against which it is added.

4.1.3 Registering a Rule Criteria Handler

This topic provides information on **Registering a Rule Criteria Handler**.

The Rule Criteria Handler implemented in the step above needs to be registered in the system. To register make an entry in the table `DIGX_FW_CONFIG_ALL_B` as shown in the example query 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 ('<<Rule Criteria Name>>', 'RuleCriteriaHandlerConfig','<<Fully
qualified name of the Handler implementation class
created in the step above>>', 'N', 'Specifies the class name of the Handler
for rule criteria type <<Rule Criteria Name>>.',
'Specifies the class name of the Handler for rule criteria type <<Rule
Criteria Name>>.',
'ofssuser', sysdate, 'ofssuser', sysdate, 'A', 1);
```

5

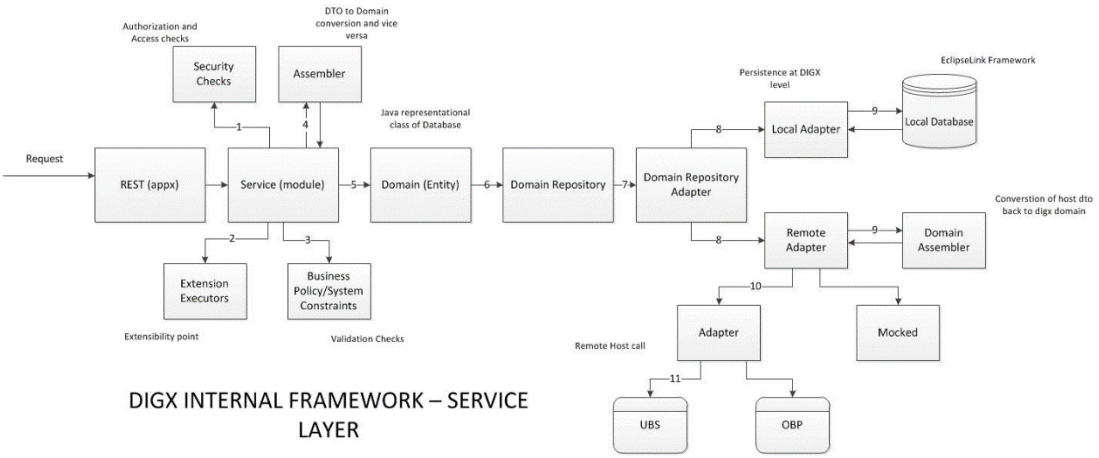
Architecture of Service Tier

This topic provides information on **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:

DIGX Service Layer



DIGX INTERNAL FRAMEWORK – 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).
9. **External System/ Host: The core banking application such as UBS/FCORE or OBP or any third-party application which operates final banking transactions.**

6
Extensible Points in GUI Tier

This topic provides information on **Extensible Points in GUI Tier**. This article provide the guidelines for UI Extensibility.

- Theme and Brand
  This topic provides information on **Theme and Brand**.
- Component Extensibility
  This topic provides information on **Component Extensibility**.
- Calling custom REST service
  This topic provides information on **Calling custom REST service**.

6.1 Theme and Brand

This topic provides information on **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

This topic provides information on **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.
- Adding New And Overriding Existing Components
  This topic provides information on **Adding New And Overriding Existing Components**.
- Add / Modify Validations
  This topic provides information on **Add / Modify Validations**.

6.2.1 Adding New And Overriding Existing Components

This topic provides information on **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.json which is to be entered manually. For example:

Sample JSON for **extension.json**

```
{"components":
[<component1>,<component2>]."partials" :
  ["partial1.html","partial2.html"]}
```

In the same manner you can override the partial templates.

> **Note:**
>
> Out of the box we are providing extension for Internal Account Input Component (inernal-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

This topic provides information on **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

This topic provides information on **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);
},
```

7

Libraries

This topic provides information on **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.

- OBDX Libraries
  This topic provides information on **OBDX Libraries**. This section provides information about various OBDX libraries that are provided out of the box.

7.1 OBDX Libraries

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

- Core/Framework Libraries
  This topic provides information on **Core/Framework Libraries**.
- Common Library
  This topic provides information on **Common Library**.
- Modules
  This topic provides information on **Modules**.
- External System Adapters
  This topic provides information on **External System Adapters**.

7.1.1 Core/Framework Libraries

This topic provides information on **Core/Framework Libraries**.

Provide infrastructure features of OBDX platform. These libraries are packaged in the `digx-shared-libs.war`

For more information on fields, refer to the field description table.

Table 7-1 Libraries

Library	Description
com.ofss.digx.infra.audit	Provides basic infrastructure classes for audit.
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.infra.crypto	Provides cryptography functions such as hash generation, public private key generation and symmetric cryptography provider.
com.ofss.digx.infra.crypto.asymmetric.impl.db	
com.ofss.digx.infra.crypto.asymmetric.impl.keystore	
com.ofss.digx.infra.crypto.asymmetric.impl.remote	

Table 7-1 (Cont.) Libraries

Library	Description
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.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
com.ofss.digx.annotations	Provides various annotations used in OBDX application
com.ofss.digx.appx.core.soap	Provides infrastructure classes for OBDX REST services
com.ofss.digx.core.enumeration.converters	Provides infrastructure classes for defining converter logic for enumeration
com.ofss.digx.framework.scheduler	Provides infrastructure classes for OBDX scheduler
com.ofss.digx.infra.orms	Contains ORM mapping files for framework domains
com.ofss.digx.infra.token.security	Contains infrastructure classes for provide user/subject information using token.

7.1.2 Common Library

This topic provides information on **Common Library**.

Provide common libraries used across all modules of the application. These libraries are packaged in digx-shared-libs.war

Table 7-2 Common Libraries

Library	Description
com.ofss.digx.adapter	Provides interfaces for cross-domain adapters.
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.access.core	Provides core classes for account access processing

7.1.3 Modules

This topic provides information on **Modules**.

Provide functional module available in the application.

- digx-access.war
- digx-account.war
- digx-accountaggregation.war
- digx-accountrelationship.war
- digx-alerts.war
- digx-analytics.war
- digx-approval.war
- digx-associatedparty.war
- digx-audit.war
- digx-berlinaisp.war
- digx-berlinpiisp.war
- digx-berlinpisp.war
- digx-brand.war
- digx-budget.war
- digx-bulkadmin.war
- digx-bulkcms.war
- digx-bulkinvoice.war
- digx-bulkpayment.war
- digx-bulkscf.war
- digx-bulktradefinance.war
- digx-bulkvam.war
- digx-card.war
- digx-chatbot.war
- digx-cms.war
- digx-collaboration.war
- digx-common.war
- digx-config.war
- digx-content.war
- digx-creditfacility.war
- digx-cutoff.war
- digx-dda.war
- digx-ebpp.war
- digx-feedback.war
- digx-finlimit.war
- digx-forexdeal.war
- digx-goal.war
- digx-insight.war
- digx-invoice.war

- digx-liquiditymanagement.war
- digx-loan.war
- digx-loanapplication.war
- digx-location.war
- digx-login.war
- digx-me.war
- digx-mobile.war
- digx-nlp.war
- digx-oauth.war
- digx-obc.war
- digx-origination.war
- digx-party.war
- digx-payment.war
- digx-pm.war
- digx-processmanagement.war
- digx-report.war
- digx-rewards.war
- digx-scf.war
- digx-security.war
- digx-sms.war
- digx-smsbanking.war
- digx-social.war
- digx-spendanalysis.war
- digx-sr.war
- digx-td.war
- digx-tradefinance.war
- digx-ukaisp.war
- digx-ukcbpii.war
- digx-ukpisp.war
- digx-user.war
- digx-vam.war
- digx-wallet.war
- digx-wm.war

7.1.4 External System Adapters

This topic provides information on **External System Adapters**.

These are packaged into module specific wars.

For more information on fields, refer to the field description table.

Table 7-3 External System Adapters

Library	Description
com.ofss.digx.<module_name>.extxface	Provides all external interfaces
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

8

Digx Scheduler Application

This topic provides information on **Digx Scheduler Application**. This section describes how to create custom schedulers in OBDX.

- Create New Scheduler Class
  This topic provides information on **Create New Scheduler Class**.
- Configure Scheduler Class
  This topic provides information on **Configure Scheduler Class**.

8.1 Create New Scheduler Class

This topic provides information on **Create New Scheduler Class**.

Follow the steps given below while creating new scheduler:

1. **Implement the class with org.quartz.Job, java.io.Serializable.**
   Example

   ```
   public class ReportSchedulerImpl implements Serializable, Job {}
   ```

2. **Define the required business logic in the overridden method execute(JobExecutionContext) required for scheduling.**
   Example

   ```
   @Overridepublic void execute
   (JobExecutionContext paramJobExecutionContext) throws JobExecutionException
   {// business logic required for scheduling}
   ```

3. **Get the SessionContext and AccessPoint objects from the method parameter before calling the business logic (if any). Set both the objects in the thread attributes.**
   Example

   ```
   SessionContext sessionContext = (SessionContext)
   paramJobExecutionContext.getJobDetail().getJobDataMap().get("sessionContext
   ");
   AccessPointDTO accessPoint = (AccessPointDTO)
   paramJobExecutionContext.getJobDetail().getJobDataMap().get("accessPoint");
   com.ofss.digx.infra.thread.ThreadAttribute.set(com.ofss.digx.infra.thread.T
   hreadAttribute.ACCESS_POINT,
   accessPoint);ThreadAttribute.set(ThreadAttribute.SESSION_CONTEXT,
   sessionContext);
   ```

4. **Call the respective service class (if any) for business logic.**
   Example

   ```
   try {
           ReportRequest service = new
           ReportRequest();service.executeScheduled(sessionContext);
               }
   ```

```
        catch (Exception e)
        {
            logger.log(Level.SEVERE, "Error occurred while executing
ReportSchedulerImpl
            class at : " + new java.util.Date(), e);
        }
        catch (java.lang.Exception e)
        {
            logger.log(Level.SEVERE, "Error occurred while executing
ReportSchedulerImpl
            class at : " + new java.util.Date(), e);
        }
```

8.2 Configure Scheduler Class

This topic provides information on **Configure Scheduler Class**.

Configure the newly created scheduler class in "DIGX_CM_TIMER" table as per the following script.

Example:

```
Insert into digx_cm_timer
(TIMER_ID,TIMER_CLASS,SECONDS,MINUTE,HOUR,DAY_OF_WEEK,DAY_OF_MONTH,MONTH,YEAR,
IS_ENABLED,IS_PESISTENT,
JVM_ID,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_VERSI
ON_NUMBER)
values
('ReportSchedulerTimer','com.ofss.digx.scheduler.report.ReportSchedulerImpl','
0',
'*/
15','*',null,null,null,null,'Y','N','1','ofssuser',sysdate,'ofssuser',sysdate,
1);
```

9
Consistent UI Download

9.1 Implement IPaginable and add XmlRootElement annotation on Response Object

This topic provides information on **Implement IPaginable and add XmlRootElement annotation on Response Object**

To enable UI Download on a service, you should implement the IPaginable Interface and add the XmlRootElement annotation as shown below. The XmlRootElement's name property should be 'root', and you need to implement all the methods in the IPaginable Interface.

```
import com.ofss.digx.app.dto.IPaginable;
import com.ofss.digx.service.response.BaseResponseObject;

import javax.xml.bind.annotation.XmlRootElement;
import java.util.List;

@XmlRootElement(name = "root")
public class BrandManagementListResponseDTO extends BaseResponseObject implements IPaginable<BrandDTO> {

    /**
     * serialVersionUID
     */
    private static final long serialVersionUID = 212372340387896408L;

    /**
     * Represents object of type {@link BrandDTO}
     */
    private List<BrandDTO> brandDTOs;

/**
 * Represents object of type {@link BrandDTO}
 */
private List<BrandDTO> items;

/**
 * Represents object of type {@link Boolean}
 */
private Boolean more;

/**
 * Represents object of type {@link Integer}
 */
private Integer totalRecords;

/**
 * Represents object of type {@link Integer}
 */
private Integer startSequence;

/**
 * Returns brandDTOs.
 *
 * @return brandDTOs in the form of {@link BrandDTO}.
 */



}

/**
 *
 * @return items in the form of {@link List<BrandDTO>}
 */
@Override
public List<BrandDTO> getItems() {
    return items;
}

/**
 *
 * @param items in the form of {@link List<BrandDTO>}
 */
@Override
public void setItems(List<BrandDTO> items) {
    this.items = items;
}

/**
 *
 * @return more in the form of {@link Boolean}
 */
@Override
public Boolean hasMore() {
    return more;
}

/**
 *
 * @param more in the form of {@link Boolean}
 */
@Override
public void setMore(Boolean more) {
```

9.2 Add configurations in the Metadata Tables

This topic provides information on **Add configurations in the Metadata Tables**.

The report generation system relies on the following metadata tables

1. DIGX_CM_TABLE_METADATA
   Stores information about each table.

   For more information on fields, refer to the field description table.

Table 9-1 Stores Information - Field Description

Property	Description
TABLE_CODE	Unique identifier for each table.
SUPPORTED_DOWNLOAD_TYPES	Media types supported for download. Supported values are ‘pdf’ and ‘csv’.
PAGINATION_TYPE	The type of pagination supported. Supported values are ‘S’ and ‘V’. Static (‘S’) refers to a one time fetching of all records. Virtual (‘V’) refers to virtual fetching of records.
ACTION_COMPONENT	The path of the UI component present in channel folder for which gets loaded on click of a row.
TABLE_HEADER	Comma Separated Values for Report and UI Screen Headers. Please note headings are NLS supported. The file name should be <TABLE_CODE>.properties and maintain at location “config/resources/nls/tablemetadata” with the keys and values. Example: BrandManagement, ManageBrand Here the BrandMangement header key will be used for reports and ManageBrand will be used for UI screen. Incase the second value is missing. The UI screen won’t show the header. Example: BrandManagement
TABLE_HEADER	The heading to show on the table. Please note headings are NLS supported. The file name should be <TABLE_CODE>.properties and maintain at location “config/resources/nls/tablemetadata” with the keys and values.
ROW_ID	Unique identifier for each record in a table.
SERIAL_NUMBER_REQUIRED	Flag to enable serial numbers on the user interface. Supported values are ‘Y’ to enable and ‘N’ to disable.
MAX_COLUMNS	Property to limit the number of columns a PDF can show. Default is 6 which can be changed by updating this property.

Example

```
Insert into DIGX_CM_TABLE_METADATA

(TABLE_CODE,SUPPORTED_DOWNLOAD_TYPES,PAGINATION_TYPE,ACTION_COMPONENT,TABLE
_HEADER,ROW_ID,
SERIAL_NUMBER_REQUIRED,MAX_COLUMNS)
        values ('ManageBrandBrand','csv,pdf','S','theme-config/review-
theme',
'brand,brand','brandId',null,4);
```

	TABLE_CODE	SUPPORTED_DOWNLOAD_TYPES	PAGINATION_TYPE	ACTION_COMPONENT	TABLE_HEADER	ROW_ID	SERIAL_NUMBER_REQUIRED
1	ManageBrandBrand	csv,pdf	S	theme-config/review-theme	brand	brandId	(null)

2. DIGX_CM_COLUMN_METADATA

   Stores information about columns available for a given table.

   For more information on fields, refer to the field description table.

Table 9-2 Stores Information - Field Description

Property	Description
TABLE_METADATA_ID	Unique identifier for each table. Many to one relationship to DIGX_CM_TABLE_METADATA table and TABLE_CODE column.
NAME	The name of the column with NLS support. Maintain the file with the name "<TABLE_CODE>.properties" at the location "config/resources/nls/tablemetadata" along with the corresponding keys and values. Avoid creating duplicate files, as this file already contains the TABLE_HEADER for the DIGX_CM_TABLE_METADATA table.
COMPONENT_ID	Custom component created for user interface. Used to add custom formatting for specific columns. Default value is 'null'.
DATATYPE	The supported datatypes are String, Number, Date, Currency and Complex. Similar to COMPONENT_ID, which is purely use for UI rendering; Datatypes is for report generation.
PATH	For value fetching, use the data path. The root path of a record is represented by the dot operator ('.'). Use the root path if the entire data object is required. Alternatively, use specific JSON paths when only specific values are required, example "Person.name", here we read name from the Person object.
FIXED	To view column on some condition, Supported values are 'Y' to enable and 'N' to disable.
SORTABLE	Flag to enable serial numbers on the user interface. Supported values are 'Y' to enable and 'N' to disable.
DOWNLOADABLE	The column support for download. Supported values are 'Y' to enable and 'N' to disable.
MIN_WIDTH	The minimum width of the column.
MAX_WIDTH	The maximum width of the column.
SEQUENCE_NO	The position of the column in the table.
LENGTH	The width of the column. The sum of all column lengths for a table code should be 100 to avoid overflow and underflow of table content. If not mentioned framework will auto size the widths.

Example

```
Insert into DIGX_CM_TABLE_METADATA

(ID,TABLE_METADATA_ID,NAME,COMPONENT_ID,DATATYPE,PATH,FIXED,SORTABLE,DOWNLO
ADABLE,SEQUENCE_NO,LENGTH)
        values

('ManageBrandBrandthemeName','ManageBrandBrand','themeName',null,'String','
brandName','Y','Y','Y',1,null);Insert into DIGX_CM_TABLE_METADATA

(ID,TABLE_METADATA_ID,NAME,COMPONENT_ID,DATATYPE,PATH,FIXED,SORTABLE,DOWNLO
ADABLE,SEQUENCE_NO,LENGTH)
        values
```

```
('ManageBrandBrandthemeDesc','ManageBrandBrand','themeDesc',null,'String','
brandDescription','N','Y','Y',2,null);Insert into DIGX_CM_TABLE_METADATA

(ID,TABLE_METADATA_ID,NAME,COMPONENT_ID,DATATYPE,PATH,FIXED,SORTABLE,DOWNLO
ADABLE,SEQUENCE_NO,LENGTH)
        values

('ManageBrandBranddateCreated','ManageBrandBrand','dateCreated','formattedD
ate','Date','creationDate','Y','Y','Y',3,40);Insert into
DIGX_CM_TABLE_METADATA

(ID,TABLE_METADATA_ID,NAME,COMPONENT_ID,DATATYPE,PATH,FIXED,SORTABLE,DOWNLO
ADABLE,SEQUENCE_NO,LENGTH)
        values
        ('ManageBrandBrandactions','ManageBrandBrand','actions','theme-
config/theme-actions','String','brandId','N','Y','Y',4,null);
```

	ID	TABLE_METADATA_ID	NAME	COMPONENT_ID	DATATYPE	PATH	FI...	S...	DO...	MIN...	MAX...	SEQUENCE_NO
1	ManageBrandBrandthemeName	ManageBrandBrand	themeName	(null)	String	brandName	Y	Y	Y	(null)	(null)	1
2	ManageBrandBrandthemeDesc	ManageBrandBrand	themeDesc	(null)	String	brandDescription	N	Y	Y	(null)	(null)	2
3	ManageBrandBranddateCreated	ManageBrandBrand	dateCreated	formattedDate	Date	creationDate	Y	Y	Y	(null)	(null)	3
4	ManageBrandBrandactions	ManageBrandBrand	actions	theme-config/theme-actions	String	brandId	N	Y	Y	(null)	(null)	4

9.3 Custom Datatypes for Report Download

This topic provides information on **Custom Datatypes for Report Download**.

The framework supports various data types, including String, Number, Date, and Complex. For any unsupported data type, the framework looks for corresponding XSL templates to handle report generation.

To create your own custom data types, follow these steps:

1. Identify the data type string to for using in the DIGX_CM_COLUMN_METADATA table. For example, 'CustomDateType' can be a string used to create special handling for dates. Alphanumeric combinations like 'CustomDateType#1' for additional variations, where each type corresponds to its own set of templates.
2. Create a custom template at the following location:

```
config\resources\com\ofss\digx\framework\list\universal\templates
      <?xml version="1.0" encoding="UTF-8"?>
      <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
      <xsl:template name="CustomDateType">
                 <xsl:param name = "data"   />
                <fo:block>
                <!-- Add handling here     --
>
                <!-- <xsl:value-of     select="$data/calendarDayOfWeek" />
-->
             </fo:block>
      </xsl:template>
      </xsl:stylesheet>
```

The above is a sample template for your reference. We save it as CustomDateType.xsl at the given location. Each template has a data parameter as input, which contains the data provided based on the path specified in the maintenance. The above template selects the 'calendarDayOfWeek' value and displays it in the PDF from the available data.

3. Import the template in config\resources\com\ofss\digx\framework\list\universal\loader.xsl and add the selection criteria.

```
<?xml version="1.0"
      encoding="UTF-8"?>
      <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
      <!-- Import template -->
      <xsl:include
      href="resources/com/ofss/digx/framework/list/universal/templates/
CustomDateType.xsl"/>
      <xsl:template name="loader">           <xsl:param name =
"dataType" />           <xsl:param name = "data" />
<xsl:choose>              <!-- Add selection critria here and call
template
      -->
                  <xsl:when test="$dataType
      = 'CustomDateType'">                  <xsl:call-template
name="CustomDateType"
      select="$data"/>              </xsl:when>              <!--
default handling -->              <xsl:otherwise>
<fo:block>                      <xsl:value-of select="$data"
      />                  </fo:block>              </
xsl:otherwise>           </xsl:choose>
      </xsl:template>
      </xsl:stylesheet>
```

4. Steps for CSV templates:
   The steps remain the same as mentioned above, with the difference being the storage location of templates and the loader file. The templates are at `'config\resources\com\ofss\digx\framework\list\universal\csv\templates', and the loader file should be 'config\resources\com\ofss\digx\framework\list\universal\csv\loader.xsl'.`

9.4 Adding content before and after table in PDF Reports

This topic provides information on **Adding content before and after table in PDF Reports**.

1. Create a template with slots at location “config\resources\uidownload\templates\pdf“ The file should be named with tableCode example ManageBrandBrand.xsl where ManageBrandBrand is tablecode.

   Use the below starter template,

   <?xml version="1.0" encoding="UTF-8"?>

   <xsl:stylesheet

   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

   xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0" >

<xsl:include href="resources/com/ofss/digx/framework/list/universal/utils/ui-download.xsl" />

<xsl:template match="/">

<xsl:call-template name="ui-download">

<xsl:with-param name="data" select="." />

</xsl:call-template>

</xsl:template>

<xsl:template name="top-slot"></xsl:template>

<xsl:template name="bottom-slot"></xsl:template>

</xsl:stylesheet>

2. Now new content can be added to the top-slot and bottom-slot templates, example
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0" >

<xsl:include href="resources/com/ofss/digx/framework/list/universal/utils/ui-download.xsl" />

<xsl:template match="/">

<xsl:call-template name="ui-download">

<xsl:with-param name="data" select="." />

</xsl:call-template>

</xsl:template>

<xsl:template name="top-slot">

<xsl:param name="data" />

<fo:block>

<xsl:value-of select="$data/status/apiType" />

</fo:block>

</xsl:template>

<xsl:template name="bottom-slot">

<xsl:param name="data" />

<fo:block>

<xsl:value-of select="$data/status/apiType" />

</fo:block>

</xsl:template>

</xsl:stylesheet>

3. The complete response object can be accessed using the $data param, excluding the items.
{

"status": {

"result": "SUCCESSFUL",
"contextID": "0063eZOykwSAHReEtbToWH00E9EP000CXx",
"message": {
"type": "INFO"
},
"apiType": "brand"
},
"brandDTOs": []
}

10

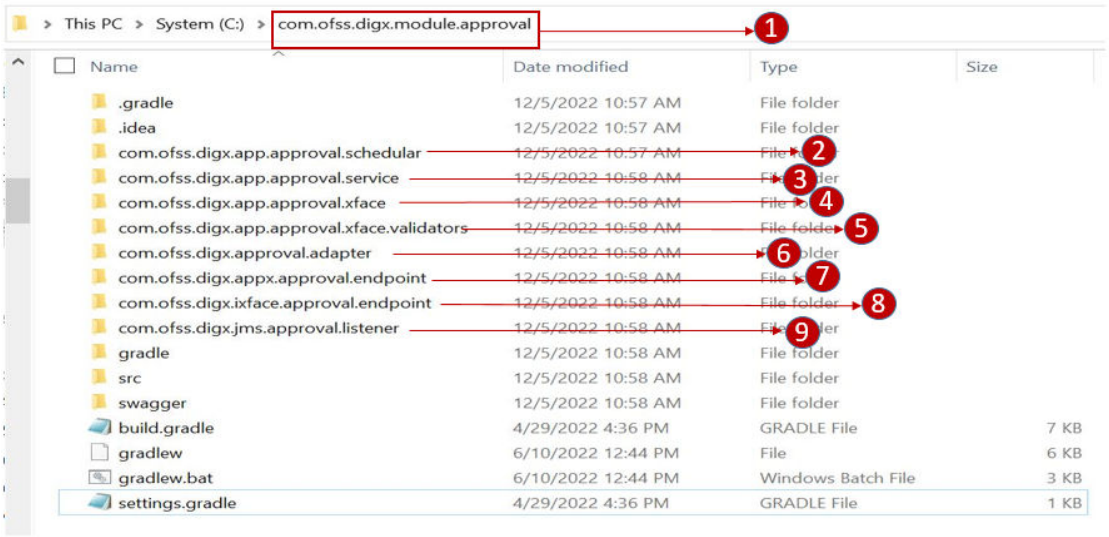
Package and Deploy Customisations

10.1 Base product packaging

This topic provides information on **Base product packaging**.

Before we look at how to package service extensions we need to understand the packaging of the base product.

Below we showcase project structure of an OBDX base module. We take approvals as an example.



1. Main-module-project.
2. Sub-project containing all the schedulers required by the module.
3. Sub-project containing all the services comprising the module. (Majority extensions fall under this subproject).
4. Sub-project containing all the Data Transfer Objects used in other sub-projects
5. Sub-project containing validators used to validate Data Transfer Objects facilitating input to OBDX.
6. Sub-project containing classes used to make cross module calls.
7. Sub-project exposing endpoints to UI for end user to interact with OBDX.

8. Subproject exposing endpoints used by other module's services to consume the services of this module via REST.
9. Sub-project containing JMS listeners required for the functioning of this module.

Each of the above listed subprojects are gradle projects which are then built into their respective jars. In case of the example shown above the jars artifacts resulting from the build are as given below.

1. com.ofss.digx.app.<<moduleName>>.scheduler.jar eg. com.ofss.digx.app.approval.scheduler.jar
2. com.ofss.digx.app.<<moduleName>>.service.jar eg. com.ofss.digx.app.approval.service.jar
3. com.ofss.digx.app.<<moduleName>>.xface.jar eg. com.ofss.digx.app.approval.xface.jar
4. com.ofss.digx.app.<<moduleName>>.xface.validators.jar eg. com.ofss.digx.app.approval.xface.validators.jar
5. com.ofss.digx.app.<<moduleName>>.adapter.jar eg. com.ofss.digx.app.approval.adapter.jar
6. com.ofss.digx.appx.<<moduleName>>.endpoint.jar eg. com.ofss.digx.appx.approval.endpoint.jar
7. com.ofss.digx.ixface.<<moduleName>>.endpoint.jar eg. com.ofss.digx.ixface.approval.endpoint.jar
8. com.ofss.digx.jms.<<moduleName>>.listener.jar eg. com.ofss.digx.jms.approval.listener.jar

10.2 Customisation packaging

This topic provides information on **Customisation packaging**.

Customizations or extensions can be broadly classified into 2 as mentioned below

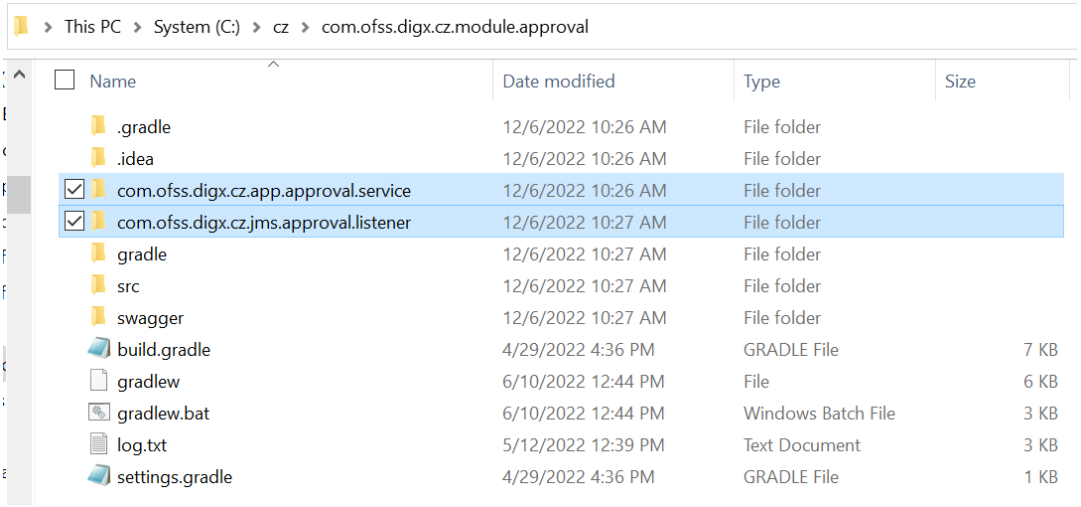
- Customizations in existing service layer without the need to expose a new customized REST endpoint
  This topic provides information on **Customizations in existing service layer without the need to expose a new customized REST endpoint**.
- Customizations to add new war
  This topic provides information on **Customizations to add new war**.

10.2.1 Customizations in existing service layer without the need to expose a new customized REST endpoint

This topic provides information on **Customizations in existing service layer without the need to expose a new customized REST endpoint**.

1. **Building custom classes into customised jars:-**
   The majority customizations that fall into this category for example Pre-Post hooks, domain and adapter extensions are done on artifacts present in the service jar mentioned in the previous section namely com.ofss.digx.app.<<moduleName>>.service.jar. So the corresponding extensions should be packaged in a jar named com.ofss.digx.cz.app.<<moduleName>>.service.jar

> **Note:**
>
> Similarily in case required artifacts related to extension classes get packaged into corresponding cz jars as mentioned above. For example if for a requirement we need to add a custom listener to a module say approval, the artifacts related to these listeners are packaged in a jar named com.ofss.digx.cz.jms.approval.listener.jar. This is depicted in the image below.



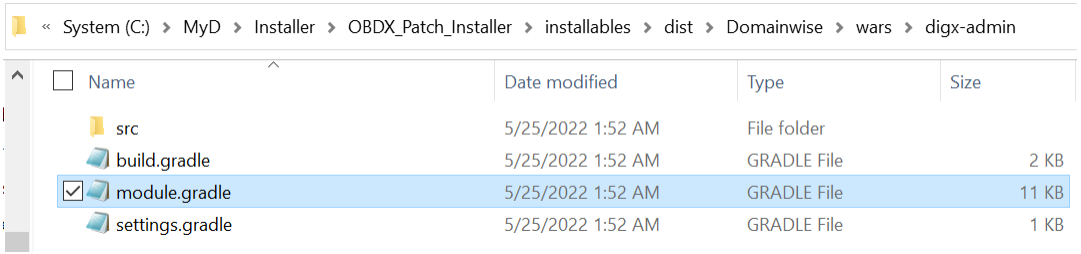
2. **Adding customised jars as dependencies in build scripts:-**
These custom jars can then be added to the war of the domain using the gradle scripts provided in the installer as demonstrated below:

   The patch set installer has the following folder structure

   `OBDX_Patch_Installer\installables\dist\Domainwise\wars`

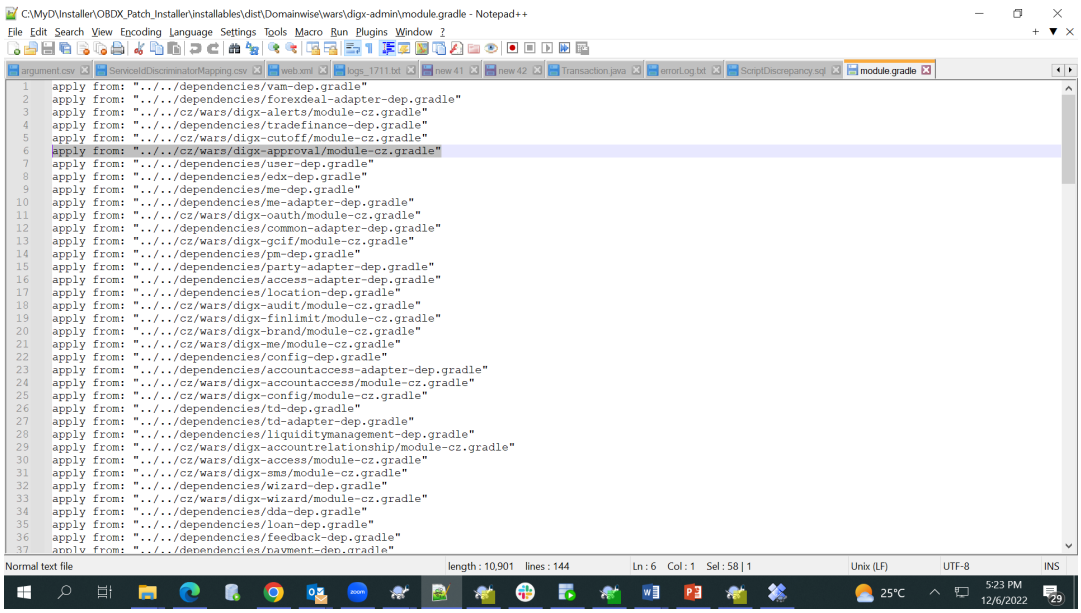
   Taking ahead the current customization example we will refer module approval packaged within domain digx-admin. Please refer the below mentioned file for module approval. (As module approval is packaged in the domain named digx-admin)

   `OBDX_Patch_Installer\installables\dist\Domainwise\wars\digx-admin\module.gradle`



   There is a line in the above file as shown below:

   apply from: `"../../cz/wars/digx-approval/module-cz.gradle"`



The highlighted line above refers to the file present inside the installer at the location given below.

`OBDX_Patch_Installer\installables\dist\Domainwise\cz\wars\digx-approval\module-cz.gradle`

So after customizations are done in a new jar say com.ofss.digx.cz.app.approval.service.jar, this jar can be specified in this (module-cz.gradle) file above as a dependency. Since dependencies in gradle are specified in group:artifact:version format, we can specify the dependency of this customized jar as below:

warLibs "com.ofss.digx.cz.module.approval:com.ofss.digx.cz.app.account.service:$libs_digxVersion"

3. a. **Place custom jars in the folder such that it gets picked by the gradle script and is packaged within the domain war:-**
   So that the above specified dependency of the customized jar gets resolved we need to place it in the folder structure as per **group:artifact:version** format.The repository defined for our base and customized product jars is

   `OBDX_Patch_Installer\installables\gradle-repo`

   Since in the above example**group** isas mentioned below

   com.ofss.digx.cz.module.approval

   So wewillcreateafolderstructure**`\com\ofss\digx\cz\module\approval`**`insideOBDX_Patch_Installer\installables\gradle-repo`

   Now coming to**artefact**

   com.ofss.digx.cz.app.app roval.service

   For this we will create a folder named **`/com.ofss.digx.cz.app.approval.service`** inside the above mentioned folder.

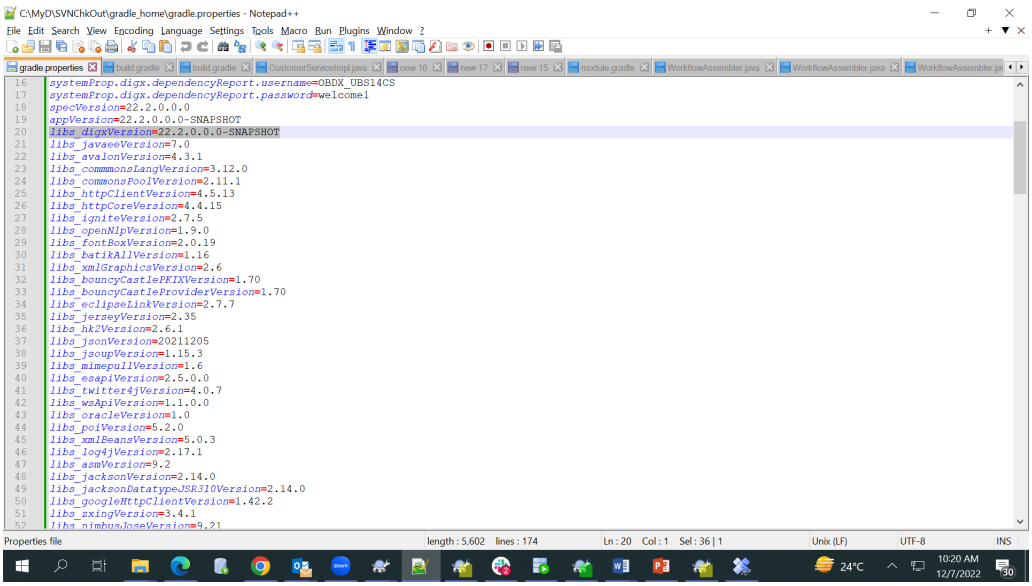
   Finally the **version** is

$libs_digxVersion

This version is a variable. The value of this variable is defined in a file

`OBDX_Patch_Installer\core\config\gradle.properties.`

If the value of the variable is as shown below



Create a folder named **22.2.0.0.0-SNAPSHOT** inside the folder created for artefact above.

Consequently the final folder structure should be as below

`OBDX_Patch_Installer\installables\gradle-repo\com\ofss\digx\cz\module\approval\com.ofss.digx.cz.app.approval.service\22.2.0.0.0-SNAPSHOT`

Place your customised jar inside the above folder such that it gets picked by the gradle script and packaged inside the digx-admin war

10.2.2 Customizations to add new war

This topic provides information on **Customizations to add new war**.

1. Create module specific folder in `dist\cz\wars` (typically 'digx-cz-<<ModuleName>>')
2. Ensure all the artifacts like src, build.gradle, settings.gradle, module.gradle of modules are present.
3. Provide all the dependency, like other module jars and third party jars in module.gradle. The libraries which are part of digx-shared-lib should not be included here.
4. Once the dependencies are included, build the war using gradle build command. It will generate the module war in `wars\digx-cz-<<ModuleName>>\build\libs` folder.
5. Ensure the generated war has all the necessary components and deploy the same as an application on the server. Also make sure that the module name is correctly present in application.properties with following property name.

6. `spring.application.name=digx-cz-<ModuleName>`

11

Messaging System Integration for OBDX

11.1 Overview

This topic provides information on **Overview**.

OBDX now supports Apache Kafka as a messaging system in addition to JMS. Kafka provides high throughput, scalability, and fault tolerance, making it an excellent choice for event-driven architectures. OBDX will work with either JMS or Kafka but not both simultaneously. This section provides details on integrating Kafka and extending its functionalities and supporting existing and any new JMS implementations.

> **Note:**
>
> The steps in this document for **Kafka integration, producer, and consumer creation** should be followed **only if Kafka is enabled**.To **enable Kafka**, refer to the section **Enabling Kafka in OBDX** of the document **Oracle Banking Digital Experience Installation Guide**.

11.2 Kafka

This topic provides information on **Kafka**.

This section describes how to enable Kafka, implement Kafka producers and consumers and override Kafka configurations.

11.2.1 New Topic Creation With Producer and Consumer

This topic provides information on **New Topic Creation With Producer and Consumer**.

1. Producing Kafka Events

In the application that produces events or messages, **com.ofss.digx.infra.events.MessageProducerUtility** class should be used for producing data to Kafka topics.

MessageProducerUtility :

For more information on fields, refer to the field description table.

Table 11-1 MessageProducerUtility

Modifier and type	Method and description
Boolean	**sendMessage**(**String** message, **String** topic) produces message with provided data to the given topic. **message**- specifies message to be sent. It is represented as a string. **topic**- name of the topic to which the message will be sent.
Boolean	**sendMessage**(**Object** key, **String** message, **Class<K>** keyClass, **String** topic) produces message with specified key and data to the given topic. **key** - The key object associated with the message. This key is used for partitioning or routing the message within the topic (can be String, Integer or null). **message**- specifies message to be sent. It is represented as a string. **keyClass**- The class type of the key object. This helps in serializing or processing the key appropriately. **topic**- name of the topic to which the message will be sent.
Boolean	**sendObjectMessage**(**Object** message ,**Class<T>** eventClass, **String** topic) produces message with provided data to the given topic. **message**- specifies value to be used in the message. Can be Avro object or normal POJO. **eventClass** - The class type of the message that is being sent to the topic. This helps in serializing or processing the message appropriately. Must be an Avro class if using Avro. If format is JSON, then it will be class instance of the POJO. **topic**- name of the topic to which the message will be sent.
Boolean	**sendObjectMessage**(Object key, Object message, Class<K> keyClass ,Class<T> eventClass, String topic) produces message with specified key and data to the given topic. **key** - The key object associated with the message. This key is used for partitioning or routing the message within the topic (can be String, Integer or null). **message**- specifies value to be used in the message. Can be Avro object or normal POJO. **keyClass**- The class type of the key object. This helps in serializing or processing the key appropriately. **eventClass** - The class type of the message that is being sent to the topic. This helps in serializing or processing the message appropriately. Must be an Avro class if using Avro. If format is JSON, then it will be class instance of the POJO. **topic**- name of the topic to which the message will be sent.

Sample Producer code:

JMSutility and TopicUtility code to be replaced by below snippet.

Example when using Avro data format

```
MessageProducerUtility.getInstance().sendObjectMessage(policyMapDTO,
PolicyMap.class, POLICIES_TOPIC);
```

Example when using Byte array data format

```
MessageProducerUtility.getInstance().sendObjectMessage(policyMapDTO,
byte[].class, POLICIES_TOPIC);
```

Example when using String data format

```
MessageProducerUtility.getInstance().sendMessage(policyMapDTO,
POLICIES_TOPIC);
```

Add below dependencies in build.gradle of the gradle project from where you are producing or publishing the Kafka message.

```
implementation
"com.ofss.digx.infra:com.ofss.digx.infra.events:$libs_digxVersion"
```

2. Consuming Kafka Events

For implementing consumers, below steps need to be performed.

1. **Creating Consumer project**
   - Create a new jar for kafka consumer for your module
     Add this in the module's **settings.gradle.**

     Add the below dependencies in build.gradle.

     ```
     implementation "org.slf4j:slf4j-api:$libs_slf4jVersion" implementation
             "org.apache.avro:avro:$libs_avroVersion" implementation

     "com.ofss.digx.infra:com.ofss.digx.infra.events:$libs_digxVersion"
     ```

2. **Extending Consumer Classes**
   Implement own consumers by extending one of the provided consumer classes:
   - **com.ofss.digx.infra.events.kafka.consumer.StringConsumer**
   - **com.ofss.digx.infra.events.kafka.consumer.AvroConsumer**
   - **com.ofss.digx.infra.events.kafka.consumer.ByteArrayConsumer.**
     The choice of class depends on the data type present in the Kafka message.

     AvroConsumer: Extend this class if the data to be consumed is of Avro type.

     StringConsumer: Extend this class if the data to be consumed is of String type.

     ByteArrayConsumer: Extend this class if the data to be consumed is of byte array type.

     All consumer classes - StringConsumer, ByteArrayConsumer and AvroConsumer are generic classes represented as AbstractConsumer<K, T, V>, where:

     – K: The type of the key. It can be String, Integer, or null.

- T: The type of the message sent by the topic.
- V: The type of object to which the message is converted for processing.

3. **Override Methods**
   topicName(): Specify the name of the topic the consumer should listen to. Returns String.

   consumerGroup(): Specify the consumer group name. Returns String. The consumer groupname in each consumer should be different in case there are multiple consumers for the same producer.

   enableSeparateConsumerGroupsPerServer(): When true, each instance of the consumer on each server creates its own consumer group. When false, all instances of this consumer across all servers share the same consumer group. Default is false if not overriden.

   ifFilteredConsumer(): Return true if :

   - The consumer is part of a shared library used in multiple WARs.
   - The **Kafka event** should only be processed **if** a particular filter criteria sent by the producer is **supported** by the consumer's application.

   In simple terms, this ensures that a consumer processes **only relevant events** based on the filter.

   If **ifFilteredConsumer()** is set to return true, you need to pass filter in headers at producer side while sending Kafka event.

   Sample producer code:

   ```
   Map<String, String> headers = new HashMap<>();headers.put("API_TYPE",
   detailDTO.getApiType());
   messageProducerUtility.sendObjectMessageWithFilter(null, detailDTO,
   String.class, byte[].class,
   MULITPLE_TRANSACTION_SERVICE_INVOCATION_QUEUE, detailDTO.getApiType(),
   headers);
   ```

   Inside the implementation of **IMessageProcessor** called from your consumer, override the method process(K key, V data, Map<String, String> headers)

   From the headers, you can fetch the filter criteria and evaluate the further processing logic.

   run(): Responsible for initiating the message consumption process. Within the run method, callthe consume method with an instance of **IMessageProcessor** to handle the processing of each consumed message.

4. **Consumer Group Size Configuration**

   - **Purpose :** Defines the number of consumer instances within a **consumer group** and is useful for **scaling** when multiple **partitions** are configured for a topic.
   - **Storage :** Existing consumer group size configurations are maintained in the `PROP_VALUE`column of the table `DIGX_FW_CONFIG_ALL_B`.
   - **Naming pattern :** `<CONSUMER_GROUP_NAME>_CONSUMER_GROUP_SIZE`

     Example : `'PoliciesTopicGroup_CONSUMER_GROUP_SIZE'`
   - **Adding a New Consumer Group Entry :** If a bank or consulting firm increases their topic partition count and wants to scale their consumers accordingly, they should add a configuration entry following the existing pattern. If not added, default will be 1.

Example SQL Insert Statement:

```
Insert into DIGX_FW_CONFIG_ALL_B

(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMA
RY_TEXT,

CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS
,OBJECT_VERSION_NUMBER,
        EDITABLE,CATEGORY_DESCRIPTION)
          values
('PoliciesTopicGroup_CONSUMER_GROUP_SIZE','KAFKA_CONFIG','1','N',null,'c
onsumer
          group size for
PoliciesTopic','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y',
        'PoliciesTopicGroup_CONSUMER_GROUP_SIZE');Consumer group size
as per
```

- **Consumer group size as per “enableSeparateConsumerGroupsPerServer” flag :**
  Scenario 1: enableSeparateConsumerGroupsPerServer = true
  - Each **server instance** will create its own **consumer group**.
  - **Max consumers per group = Number of topic partitions**.
  - Example :

    ```
    Total partitions = 10
    Managed servers = 2
    Max consumers in a group = 10
    Recommended consumer group size = Up to 10 per server
    ```

    Scenario 2: enableSeparateConsumerGroupsPerServer = false
    * All instances of a particular consumer belong to the same consumer group.
    * The number of consumers per server should be calculated as
      Total Partitions ÷ Number of Managed Servers
    * Example :

      ```
      Total partitions = 10
      Managed servers = 2
      Max consumers in a group = 10
      Recommended consumer group size = Up to 5 per server
      ```

      > **Note:**
      >
      > The consumer group size should not exceed the partition count of the topic.

5. **Creating SPI Entry for Consumer**
   A file named **com.ofss.digx.infra.events.kafka.consumer.IConsumer** should be created in **resources/META-INF/services** in **com.ofss.digx.cz.kafka.{module}.consumer** and the entry of the consumer class has to be provided in this file.

3. Implementing Event Processing Logic

- **com.ofss.digx.infra.events.processor.IMessageProcessor**

  This interface is designed to support both JMS and Kafka. Implementing this interface provides a common business logic layer to ensure maintainability, code reusability and consistent processing approach across messaging systems.

- Write a class implementing **com.ofss.digx.infra.events.processor.IMessageProcessor** in your Gradle project. Inside this class, override the **process** method and write the message or event processing logic. This class has to be invoked from the Kafka consumer and JMS listener classes. Make sure the project's JAR file is a part of the class-path of the application where the consumer is defined.
  IMessageProcessor<K,V>

  For more information on fields, refer to the field description table.

Table 11-2 IMessageProcessor<K,V>

Modifier and type	Method and description
Void	**process**(**K** key, **V** data) processes messages from listener (JMS) or consumer (Kafka) key - The key object associated with the message data - The data to be processed.
Void	**default process(K** key**, V** data**, Map<String, String>** headers**)**processes messages from listener (JMS) or consumer (Kafka) Default method. Provides event headers. key - The key object associated with the message data - The data to be processed. headers - Event headers associated with every message

Example class extending **ByteArrayConsumer** and using **IMessageProcessor** implementation which will be used to consume data from Kafka topic.

Sample Kafka byte array consumer code

```
package com.ofss.digx.kafka.sms.consumer.authorization.policy; import
com.ofss.digx.app.sms.dto.authorization.policy.PolicyMapDTO;import

com.ofss.digx.app.sms.processors.authorization.policy.PoliciesMessageProces
sor;import com.ofss.digx.infra.events.kafka.consumer.AvroConsumer;
import org.slf4j.Logger;import org.slf4j.LoggerFactory; public class
PoliciesTopicConsumer extends ByteArrayConsumer<String, byte[],
PolicyMapDTO>
          {
private static final String
          THIS_COMPONENT_NAME = PoliciesTopicConsumer.class.getName();
private static final Logger logger =
LoggerFactory.getLogger(THIS_COMPONENT_NAME);
private static final String POLICIES_TOPIC = "PoliciesTopic";
private static final String POLICIES_TOPIC_GROUP =
"PoliciesTopicGroup";
public PoliciesTopicConsumer() throws Exception
{
super(String.class, byte[].class,
          PolicyMapDTO.class);
}
```

```
@Override public String topicName()
        {
return POLICIES_TOPIC;
}
@Override public String consumerGroup()
        {
return POLICIES_TOPIC_GROUP;
}
@Override public boolean
          enableSeparateConsumerGroupsPerServer()
{
return true;
}
@Override public void run()
{
logger.info("Entering into
run method of {}", THIS_COMPONENT_NAME);
consume(new PoliciesMessageProcessor());
logger.info("Exiting from run method of {}", THIS_COMPONENT_NAME);
}
}
```

Example class implementing IMessageProcessor<K,V>

```
package com.ofss.digx.app.sms.processors.authorization.policy;
import com.ofss.digx.app.sms.dto.authorization.policy.PolicyMapDTO;
import
com.ofss.digx.app.sms.service.authorization.provider.RoleTransactionAccessS
ervice;
import com.ofss.digx.infra.exceptions.Exception;
import com.ofss.digx.infra.events.processor.IMessageProcessor;import
org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class PoliciesMessageProcessor implements IMessageProcessor<String,
PolicyMapDTO>
{
/**     * Stores the entity name represented by this {@code Class} object
as a {@code String}     */
private static final String THIS_COMPONENT_NAME =
PoliciesMessageProcessor.class.getName();
private static final transient Logger logger =
LoggerFactory.getLogger(THIS_COMPONENT_NAME);
@Override    public void process(String key, PolicyMapDTO data)
{
try {
if (!data.getValues().isEmpty())
{
RoleTransactionAccessService cacheLoader =
RoleTransactionAccessService.getInstance(null);
cacheLoader.updateResourceCache(data.getValues());
}
}
catch (Exception e)
{
logger.error("Exception encountered while invoking process method of
```

```
{}",
THIS_COMPONENT_NAME, e);
}
}
}
```

11.2.2 Kafka Producer/Consumer Configurations

This topic provides information on **Kafka Producer/Consumer Configurations**.

OBDX provides some default configurations for Kafka for Broker settings, partitioning and replication, consumer group size, etc. maintained in the table **DIGX_FW_CONFIG_ALL_B** with `category_id` **KAFKA_CONFIG**. The configurations by default will be applicable to all producers and consumers.

1. Generic Configurations

Below mentioned are the generic configurations **(PROP_ID)** and their default values **(PROP_VALUE)**. These can be overridden if required, by updating them in the table.

For more information on fields, refer to the field description table.

Table 11-3 (PROP_ID) and their default values

PROP_ID	Default PROP_VALUE	Description
bootstrap.servers	localhost:8080	Specifies the Kafka broker(s) that consumers and producers should connect to.
enable.auto.commit	true	Determines whether the consumer's offset is automatically committed.
auto.commit.interval.ms	5000	The frequency (in milliseconds) at which the consumer commits offsets when auto-commit is enabled.
auto.offset.reset	latest	Defines the behavior when a consumer starts reading from a topic. Options: earliest: Read from the beginning of the log. latest: Read only new messages.
CONSUMER_POLL_TIMEOUT_MS	2000	The maximum time (in milliseconds) a consumer waits for records when polling from Kafka.

Apart from the above mentioned properties, any other producer and consumer configuration provided by Kafka can also be overridden by adding the respective entry in the table **DIGX_FW_CONFIG_ALL_B**.

2. Changing Topic Level Configurations

Bank can also override any producer and consumer configuration for a particular topic with their custom values instead of the default ones, by adding an entry in the table **DIGX_FW_CONFIG_ALL_B** in the column **PROP_ID** with the pattern `TOPIC_NAME@CONFIGURATION`.

For example,

```
Insert into DIGX_FW_CONFIG_ALL_B
```

```
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEX
T,
CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJEC
T_VERSION_NUMBER,
EDITABLE,CATEGORY_DESCRIPTION)
          values('structure-
createdAndAuthorized@auto.offset.reset','KAFKA_CONFIG','latest','N',null,'Auto
          offset reset','ofssuser'sysdate,'ofssuser',sysdate,'A',1,'Y', 'Auto
offset reset
          ');
```

11.3 JMS

This topic provides information on **JMS**.

JMSUtility and **TopicUtility** is now deprecated.**MessageProducerUtility**should be used for producing messages to JMS destinations. **MessageProducerUtility** requires entries of the JMS destinations to be present in the table **`DIGX_FW_DESTINATION_METADATA`**.

- For any new as well as existing customized JMS queues or topics, Please ensure to add entries in this table with relevant metadata for topic/queue maintenance. The table has three columns:
  - `DESTINATION`: The name of the topic or queue.
  - `CONNECTION_FACTORY`: The connection factory used.
  - `DESTINATION_TYPE`: The type (e.g., topic or queue).

  For example,

  ```
  INSERT INTO DIGX_FW_DESTINATION_METADATA values
  ('PoliciesTopic','POLICIESQCF', 'TOPIC');
  ```

- Add below dependency to build.gradle of the JMS listener project.

  ```
  implementation
          "com.ofss.digx.infra:com.ofss.digx.infra.events:$libs_digxVersion"
  ```

- For message processing logic, create a class implementing **com.ofss.digx.infra.events.processor.IMessageProcessor** interface, override the **process** method and call this method from the JMS listener class. Refer **Section3: Implementing Event Processing Logic**of **New Topic Creation With Producer and Consumer** chapter for the same.
- **Sample code for JMS listener**

  ```
  package com.ofss.digx.jms.sms.listener.authorization.policy;
  import com.ofss.common.platform.server.ServerPlatformUtils;
  import com.ofss.digx.app.sms.dto.authorization.policy.PolicyMapDTO;
  import
  com.ofss.digx.app.sms.processors.authorization.policy.PoliciesMessageProces
  sor;
  import com.ofss.digx.infra.jms.listener.IJMSTopicListener;
  import org.slf4j.Logger;import org.slf4j.LoggerFactory;
  import javax.jms.JMSException;import javax.jms.Message;
  import javax.jms.ObjectMessage;
  ```

```
import java.io.Serializable;
public class PoliciesTopicListener implements IJMSTopicListener
{
/**     * Stores the name of the entity(class)represented by this {@code
Class} object * as a {@code String}     */
private static final String THIS_COMPONENT_NAME =
PoliciesTopicListener.class.getName();
private static final transient Logger logger =
LoggerFactory.getLogger(THIS_COMPONENT_NAME);
/**     * Property which stores the topic JNDI name.     */
private static final String POLICIES_TOPIC = "PoliciesTopic";
/**     * Property which stores the topic connection factory JNDI
name.     */
private static final String POLICIES_QCF = "POLICIESQCF";
private final PoliciesMessageProcessor processor = new
PoliciesMessageProcessor();
@Override    public String getConnectionFactoryName()
          {
return getJNDIName(POLICIES_QCF);
}
@Override public String getTopicName()
        {
return getJNDIName(POLICIES_TOPIC);
}
@Override public void onMessage(Message message)
          {
logger.debug("Entered into onMessage() of policy topic listener in class
{} ",
        THIS_COMPONENT_NAME);
 Serializable obj = null;
if (message instanceof ObjectMessage)
{
ObjectMessage objMessage = (ObjectMessage)message;
try {
obj = objMessage.getObject();
if (obj instanceof PolicyMapDTO)
{
          @SuppressWarnings("unchecked")
          PolicyMapDTO applicationRoles = (PolicyMapDTO)
obj;
          processor.process(null, applicationRoles);
        }
}
catch (JMSException |
          ClassCastException e)
{
          logger.error("Exception encountered while invoking the service
{}
in onMessage",
          THIS_COMPONENT_NAME, e);
}
catch(java.lang.Exception e)
{
          logger.error("Exception encountered while invoking the service
{}
in onMessage",
```

```
            THIS_COMPONENT_NAME, e);
    }
     }
    }
    private String getJNDIName(String name)
             {
    return
    ServerPlatformUtils.detectServerPlatform().platformJNDIName(name);
    }
    }
```

11.4 Consuming New External Kafka Events

This topic provides information on **Consuming New External Kafka Events**.

- For use cases where any new external Kafka topic needs to be listened to, a new consumer class can be created by following the steps outlined in the **Section: 2. Consuming Kafka Events** of **Kafka** chapter of this document. Additionally, this class must implement the **com.ofss.digx.infra.events.kafka.consumer.IKafkaConsumable** interface.
- For this Topic, new entries must be added in the table **DIGX_FW_CONFIG_ALL_B** as mentioned in the document **Oracle Banking Digital Experience Installation Guide** in the section **OBDX Pre-defined External Kafka Topic Configurations**. Apart from these, any other Producer and Consumer properties defined by Kafka can also be added for the Topic in this table.

For Example,

```
Insert into DIGX_FW_CONFIG_ALL_B

(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEX
T,CREATED_BY,CREATION_DATE,
LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE
,CATEGORY_DESCRIPTION)
          values
('externalSystemAlertMessage@bootstrap.servers','KAFKA_CONFIG',
'ofss-
mum-645.snbomprshared1.gbucdsint02bom.oraclevcn.com:9092','N',null,'Kafka
props','ofssuser',sysdate,
'ofssuser',sysdate,'A',1,'Y','Kafka props');
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
('externalSystemAlertMessage@sasl.jaas.config','KAFKA_CONFIG','org.apache.kafk
a.common.security.scram.ScramLoginModule
          required username="obedx" password="obedx-secret";','N',null,'Kafka
props','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y','Kafka props');
Insert into DIGX_FW_CONFIG_ALL_B
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEX
T,CREATED_BY,CREATION_DATE,
LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE
,CATEGORY_DESCRIPTION)
```

```
values('externalSystemAlertMessage@sasl.mechanism','KAFKA_CONFIG','SCRAM-
SHA-256','N',null,
'Kafka props','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y','Kafka props');
Insert into DIGX_FW_CONFIG_ALL_B
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEX
T,
CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJEC
T_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)
          values
('externalSystemAlertMessage@security.protocol','KAFKA_CONFIG','SASL_SSL','N',
null,
'Kafka props','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y','Kafka props');
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('externalSystemAlertMessage@ssl.truststore.location','KAFKA_CONFIG','/
scratch/app/domain/obdx_domain/KafkaServerKeystore.jks','N',null,'Kafka
          props','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y','Kafka
props');Insert into DIGX_FW_CONFIG_ALL_B

(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEX
T,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,
OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)
          values
('externalSystemAlertMessage@ssl.truststore.password','KAFKA_CONFIG','orcl@123
','N',null,'Kafka
props','ofssuser',sysdate,'ofssuser',sysdate,'A',1,'Y','Kafka props');
```

Sample consumer implementation for external topic:

```
package com.ofss.digx.app.kafka.origination.consumer;
import
com.ofss.digx.app.origination.processors.ApplicationOnSubmitEventMessageProces
sor;
import com.ofss.digx.infra.events.kafka.consumer.IKafkaConsumable;
import com.ofss.digx.infra.events.kafka.consumer.StringConsumer;
import com.ofss.digx.infra.exceptions.Exception;import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class OriginationEventTopicConsumer extends StringConsumer<String,
String, String> implements IKafkaConsumable
{
 private String targetUnit;
public OriginationEventTopicConsumer()
throws Exception
{
super(String.class, String.class, String.class);
}
 private static final String THIS_COMPONENT_NAME =
OriginationEventTopicConsumer.class.getName();
 private static final Logger logger =
LoggerFactory.getLogger(THIS_COMPONENT_NAME);
private static final String OR_OBDX_TOPIC = "externalSystemAlertMessage";
 private static final String OR_GROUP_ID = "obdx-obo-consumer";
```

```
 @Override public String consumerGroup()
{
 return OR_GROUP_ID;
}     @Override public String topicName()
{
 return OR_OBDX_TOPIC;
}     @Override public boolean enableSeparateConsumerGroupsPerServer()
{
 return false;
}     @Override public void run()
{        logger.info("Entering into run method of {}",
THIS_COMPONENT_NAME);
 consume(new ApplicationOnSubmitEventMessageProcessor());
 logger.info("Exiting from run method of {}", THIS_COMPONENT_NAME);
 }
}
```

Index

M

N

O

P

R

S

T