# Table of Contents

1. **Preface** .......................................................................................................................... 6
   1.1 Intended Audience ........................................................................................................ 6
   1.2 Documentation Accessibility ...................................................................................... 6
   1.3 Access to Oracle Support ............................................................................................ 6
   1.4 Structure ..................................................................................................................... 6
   1.5 Related Information Sources ...................................................................................... 6

2. **Objective and Scope** .................................................................................................... 7
   2.1 Background .................................................................................................................. 7
   2.2 Objective ................................................................................................................... 7
   2.3 Scope .......................................................................................................................... 8
   2.4 Structure .................................................................................................................... 9

3. **Architecture of Service Tier** ...................................................................................... 10

4. **Extensible Points in Service Tier** ............................................................................. 12
   4.1 REST Tier .................................................................................................................. 12
       4.1.1 Guidelines .......................................................................................................... 12
       4.1.2 HTTP Standards ............................................................................................... 13
   4.2 Service Extensions ..................................................................................................... 14
       4.2.1 Service Extension Interface ............................................................................... 16
       4.2.2 Service Extension Executor Interface ............................................................... 17
       4.2.3 Default Extension (Void Extension) ................................................................. 18
       4.2.4 Custom Extension ............................................................................................. 18
1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

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

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=accandid=info or visit http://www.oracle.com/pls/topic/lookup?ctx=accandid=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

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

The subsequent chapters describe the following details:

- Purpose
- Configuration / Installation

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Release 18.1.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Licensing Guide
2. **Objective and Scope**

2.1 **Background**

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

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

2.2 **Objective**

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

**Extensibility objective**

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

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

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

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

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

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

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

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

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

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

2.4 Structure

This document is organized into following chapters

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

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

![Diagram of DIGX Service Layer](image)

**Figure 1 DIGX Service Layer**

1. **REST**: The endpoint layer which gets invoked whenever a request URI is called. Also known as the layer which contains REST annotations and path to resources or sub-resources of an application.

2. **Service**: Also called as module layer of the framework. Generally, the core modules of DIGX application will have their own service implementation classes responsible for implementing core business logic, validation and security checks.

3. **Assemblers**: These are the mapping classes which convert data object containing request or response parameters into domain or database compatible form. These classes help us to get the required domain objects which can be further used in object-relational mapping.

4. **Business Policy/System Constraints**: Before letting the query data read or persisted in the core application, certain business policies need to be validated. This separate layer of constraints check let the application behave as per the policies configured.

5. **Domain/Entity**: Represents the Java Object form of Database. This domain layer also contains data to be persisted or query response fetched through Object relational mapping.

6. **Domain Repository**: The term ‘repository’ denotes any data storage component. Each module of the application will have its own repository to manage its CRUD operations and that can be easily done using this component of the DIGX framework.

7. **Domain Repository Adapter**: Adapters are the connecting points to some external system and as the name suggests, this part of the framework contacts two kinds of repositories of DIGX application – Local Repository and Remote Repository. Eventually, the configured one out of these two will be invoked.
8. **Adapters**: Finally these are the adapter classes that can call either Local Database (DIGX specific tables) or Remote Repository (external system). Remote adapters can further be mocked if required.

9. **External System/ Host**: The core banking application such as UBS or OBP or any third-party application which operates final banking transactions.
4. **Extensible Points in Service Tier**

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

4.1 **REST Tier**

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

References:

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

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

4.1.1 **Guidelines**

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

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

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

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>Request successfully executed and the response has content</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>Resource successfully created</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
<td>Request has been accepted for processing but processing has not been completed</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
<td>Request successfully executed and the response doesn't have content</td>
</tr>
</tbody>
</table>
### 4.2 Service Extensions

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

**Pre-extension hook**

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

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

**Post-extension hook**

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

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

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

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

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

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

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

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

```java
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.

```java
public interface LoanApplicationExt {

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before creating loan applications from the Service class [Link 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 [Link 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 loanApplicationCreateResponseDTO
     * The instance of type [Link LoanApplicationCreateResponseDTO] used for creating loan application response.
     *
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
                            LoanApplicationCreateResponseDTO loanApplicationCreateResponseDTO) 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 [Link LoanApplicationApplicationService].
     *
     * @param sessionContext
     * The session context of request in the form of [Link SessionContext].
     *
     * @param loanApplicationUpdateRequestDTO
     * The request DTO of type [Link LoanApplicationUpdateRequestDTO] used for updating loan application of any party.
     *
     * @throws Exception
     */
}
```
4.2.2 Service Extension Executor Interface

This acts as an interface for the application service to access service extensions. The implementing class creates an instance 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:

```java
public interface LoanApplicationExtExecutor {
    /**
     * 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
     *      The request DTO of type (@link LoanApplicationCreateRequestDTO) used for creating loan application.
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext,
                            LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
                            LoanApplicationCreateResponseDTO loanApplicationCreateResponseDTO)
        throws Exception;
}
```
4.2.3 Default Extension (Void Extension)

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

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

For example:

```java
import com.sfs1.dlig.app.validation.dto.application.LoanApplicationCreateRequestDTO;

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

public class void.createApplicationExt implements LoanApplicationExt {
    //
    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
     * creating loan applications from the Service class (Loan LoanApplicationApplicationService).
     * @param sessionContext
     *          The session context of request in the form of (SessionContext).
     * @param LoanApplicationCreateRequestDTO
     *          The request instance of type (LoanApplicationCreateRequestDTO) for creating loan application
     *          request.
     * @throws Exception
     */
    @Override
    public void preCreateSessionContext 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 (Loan LoanApplicationApplicationService).
     * @param sessionContext
     *          The session context of request in the form of (SessionContext).
     * @param LoanApplicationCreateRequestDTO
     *          The instance of type (LoanLoanApplicationCreateRequestDTO) used for creating loan application
     *          request.
     * @param LoanApplicationCreateResponseDTO
     *          The instance of type (LoanLoanApplicationCreateResponseDTO) used for creating loan application
     *          response.
     * @throws Exception
     */
    @Override
    public void postCreateSessionContext sessionContext,
            LoanApplicationCreateRequestDTO LoanApplicationCreateRequestDTO,
            LoanApplicationCreateResponseDTO LoanApplicationCreateResponseDTO) throws Exception {
    }
}
```

4.2.4 Custom Extension

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

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

For example:

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

4.2.6 Sequence of events in service extension

Every application service method has a standard set of framework method calls as shown in the sequence diagram below:
The pre hook is provided after the invocation of *fetchTransactionStatus* call inside the application service. At this step, the current task code is received, any additional manipulation of the input received from the User interface channel can be done in the pre hook. Apart from this additional data coming from the screen specific to client requirements can be handled in the pre hook.

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

OBDX supports three types of validations

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

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Email</td>
<td>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</td>
</tr>
<tr>
<td>@Mandatory</td>
<td>This annotation marks the fields as mandatory. Once marked, if the field is null then respective error-code is thrown</td>
</tr>
<tr>
<td></td>
<td>Eg. @Mandatory(errorCode = DemandDepositErrorConstants.DDA_MANDATORY_ACCOUNT_ID)</td>
</tr>
<tr>
<td>private Account accountId;</td>
<td></td>
</tr>
<tr>
<td>@Length</td>
<td>This annotation marks the lengths of the fields. Once marked, if the validation is violated then the respective error code is thrown.</td>
</tr>
<tr>
<td></td>
<td>Eg. @Length(min = 2, max = 20, errorCode = PartyErrorConstants.PI_LENGTH_EXTERNAL_REF_ID)</td>
</tr>
<tr>
<td>@NonNegative</td>
<td>This annotation checks that the value is non-negative</td>
</tr>
<tr>
<td>@Regex</td>
<td>This annotation checks if the value matches regular expression provided</td>
</tr>
</tbody>
</table>

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

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

4.3.1 Adding new business policy

Customization developer can add new business policy for new or existing services. Following are the steps to add a new business policy:
1. **Create new business policy DTO.** This DTO is supposed to encapsulate all the input fields upon which validation is to be performed.

2. **Create new business policy.** This class must extend AbstractBusinessPolicy class and implement the validatePolicy() method. This method should have the validation logic and if the validation fails, then it should call addValidationError() method with a new instance of ValidationError as parameter. One of the parameter to the constructor of ValidationError is error code. A new error could be added by following guidelines provided in Error Messages section.

3. **Configure new business policy.** The business policy class created in above step should be configured in digx-fw-config-all_b table with category_id as 'CustomBusinessPolicy' and prop_id as the policy identifier. For example:

   ```sql
   insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER) values ('CREATE_TERM_DEPOSIT_BUSINESS_POLICY ', 'CustomBusinessPolicy ', 'com.ofss.digx.cz.domain.td.entity.policy.CustomBusinessPolicy', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
   ```

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

   ![Class Diagram](image)

4.3.2 **Extending existing business policy**

OBDX provides out of box business policies for all services. If only a part of the validation is to be modified or a new validation is to be added in addition to the validations that the existing business policy does, then it is possible to extend existing business policy and override existing validation.
Please note that this capability depends on how the original business policy is coded. If the out of box business performs all its validations in validatePolicy() method, then this approach may not be useful. On the other hand, if the out of box business policy has separate individual methods for validations and validatePolicy() method calls these methods one by one, then extension of the business policy is useful.

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

4.4 Dictionary

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

**Data transfer object (DTO)**

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

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

    /*
     * Subclasses of the Customized AbstractDomainObject corresponding to this
     * AbstractDomainObject (or) are defined with the help of this attribute. This concept can be extended
     * to have joined or (or) union subclass hierarchy 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 fact.
     */
    public Dictionary[] getDictionaryArray() {
        return this.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 objects. 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

```
package com.ofss.fc.framework.domain.common.dto;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlType;

@XmlType(namespace="http://dto.common.domain.framework.fc.ofss.com")
public class Dictionary implements Serializable {

  /**
   * private static final long serialVersionUID = 642866746519694755L;
   * private NameValuePairDTO[] nameValuePairDTOArray;
   * public NameValuePairDTO[] getNameValuePairDTOArray() {
   *   return nameValuePairDTOArray;
   * }
   * public void setNameValuePairDTOArray(NameValuePairDTO[] nameValuePairDTOArray) {
   *   this.nameValuePairDTOArray = nameValuePairDTOArray;
   * }
  }
```

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

```
@Override
public void preCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {

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

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

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

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

4.5.1 Custom Domain Objects

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

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

Translating Dictionary data into custom domain object

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

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

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

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

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

/**
 * custom domain object to handle changes at UI or pre-boot level
 */
public class CustomCollaborationDomainObject extends Collaboration{

  /**
   * 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:

```sql
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)
values ('com.ofss.digx.domain.origination.entity.submission.lending.application', 'CustomizedAbstractDomainObjectConfig', 'com.ofss.digx.domain.origination.entity.submission.lending.application.ext.Application', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
```

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

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

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

- Create new ORM file to handle Customized Domain Object. This ORM file should contain entries for all columns in corresponding domain object table.
- 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):

```xml
<extension-meta

  <meta
    xsi:type="xml-meta.ORM.Meta"/>
</extension-meta>
```

```
<entity name="CollaborationDemo" class="com.nsfw.disp.domain.CollaborationDemo" customload="CustomCollaborationDemoObject"/>

<attribute
  attribute-type=""com.nsfw.disp.domain.CollaborationDemo#ID""/>
  <attribute
    attribute-type=""com.nsfw.disp.domain.CollaborationDemo#CollaborationKey""/>

<attribute
  attribute-type=""java.lang.String"" name=""externalRefId""/>
  <attribute
    attribute-type=""java.lang.String"" name=""authorID""/>

<attribute
  attribute-type=""java.lang.String"" name=""authorName""/>
  <attribute
    attribute-type=""java.lang.String"" name=""collaborationType""/>

<attribute
  attribute-type=""java.lang.String"" name=""secureCode""/>
  <attribute
    attribute-type=""java.lang.String"" name=""url""/>

<attribute
  attribute-type=""com.nsfw.fc.datatype.Data"" name=""expiryData""/>

<attribute
  attribute-type=""java.lang.Boolean"" name=""status""/>

<attribute
  attribute-type=""java.util.List"" fetch=""LAZY"
    name=""participantList"" target-entity=""com.nsfw.disp.domain.CollaborationDemo#Participant""/>

<attribute
  attribute-type=""java.util.List"" fetch=""LAZY"
    name=""participantList"" target-entity=""com.nsfw.disp.domain.CollaborationDemo#Participant""/>

<attribute
  attribute-type=""java.util.List"" fetch=""LAZY"
    name=""participantList"" target-entity=""com.nsfw.disp.domain.CollaborationDemo#Participant""/>
```
Mapping configuration xml(persistence.xml):

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:

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

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

```
insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED)
values ('<Fully qualified domain name>', '<Determinant Type>', '<Represented Name>', 'Y');
```

There are four possible determinant types as follows:

- Enterprise (ENT)
- Legal Entity (LGE)
- Market Entity (MKE)
- Business Unit (BNU)
4.5.2 Adding new Domain

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

1. Create new domain class. The new domain class must extend AbstractDomainObject and implement IPersistenceObject.

2. Identify attributes and operations supported by the domain and add them to above domain class accordingly.

3. The domain object will typically have associated DTO that encapsulates same fields as in domain. This DTO will be used in request and responses. An assembler will be used to map fields between domain object and the DTO. Below diagram depicts this relationship.

4. Configure this new domain for appropriate entity level

   insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED) values ('<Fully qualified domain name>', '<Determinant Type>', '<Represented Name>', 'Y');

For example,

   insert into digx_me_entity_determinant_b (DOMAIN_OBJECT_NAME, DETERMINANT_TYPE, REPRESENTED_FIELD, IS_FEATURE_ENABLED) values ('com.ofss.digx.cz.domain.payment.entity.payee.Payee', 'BNU', 'New Payee', 'Y');
4.6 Error Messages

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

4.6.1 Adding Error Message

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

ERROR_CODE column should contain unique value.

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

4.6.2 Mapping Host Error Code To OBDX Error Code

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

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

THIRD_PARTY_ERR_COD column holds the host error code.

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

4.7 Adapter Tier

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

4.7.1 Service Provider Interface (SPI) Approach

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

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

- https://docs.oracle.com/javase/tutorial/ext/basics/spi.html

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.

```java
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:

```java
/**
   * 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.loanapplication.entity.loanapplication.repository.adapter.
   * LoanApplicationRepositoryAdapter
   * #createLoanApplication(com.ofss.digx.domain.loanapplication.entity.loanapplicationLoanApplication)
   */
@Override
public LoanApplication createLoanApplication(LoanApplication object) throws Exception {
    LoanApplicationRequirementAdapter adapter = ExtxfaceAdapterFactory.getInstance().getAdapter(
            LoanApplicationRequirementAdapter.class, "create",
            LoanApplication RequirementAdapter.class.getName()));
    LoanApplicationAssembler assembler = new LoanApplicationAssembler();
    LoanApplicationCreateResponseDTO response = adapter.create(object, key, getSubmissionId(), assembler.fromDomainObject(object));
    return assembler.toDomainObject(response.getLoanApplicationCreateResponseDTO());
}
```

**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

```
<table>
<thead>
<tr>
<th>PROP_ID</th>
<th>CATEGORY_ID</th>
<th>PROP_VALUE</th>
<th>FACTORY_SHIPPED_FLAG</th>
<th>PROP_COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>extxfaceadapterconfig</td>
<td>UBS12.4</td>
<td>N</td>
<td>(null)</td>
</tr>
<tr>
<td>2 001</td>
<td>extxfaceadapterconfig</td>
<td>OBP2502</td>
<td>N</td>
<td>(null)</td>
</tr>
</tbody>
</table>
```

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.

```
> com.ofss.digx.ubs123.adapter.impl.handler
> com.ofss.extsystem.ubs.business.extsystems.adapters.ubs
> META-INF
>   services
>     com.ofss.digx.app.adapter.core.CurrencyServiceAdapter
>     com.ofss.digx.app.adapter.core.CustomerServiceAdapter
>     com.ofss.digx.app.adapter.core.ValueDateAdapter
>     com.ofss.digx.app.alerts.adapter.IHostAdapter
>     com.ofss.digx.app.common.adapter.IEnumerationAdapter
```

**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).

<table>
<thead>
<tr>
<th>DOMAIN_OBJECT_NAME</th>
<th>DETECTIVE_TYPE</th>
<th>REPRESENTED_FIELD</th>
<th>IS_FEATURE_ENABLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.domain.entity.merchant.Merchant</td>
<td>LRE</td>
<td>Merchant Maintenance</td>
<td>Y</td>
</tr>
<tr>
<td>com.ofss.digx.domain.entity.mailbox.ingsubjectmap</td>
<td>LRE</td>
<td>User Group Subject Map</td>
<td>Y</td>
</tr>
<tr>
<td>com.ofss.digx.domain.config.entity.ConfigVarDomains</td>
<td>HNU</td>
<td>ConfigVar Details</td>
<td>Y</td>
</tr>
<tr>
<td>com.ofss.digx.domain.report_entity_Report&amp;AuthRequest</td>
<td>LRE</td>
<td>Report Request</td>
<td>F</td>
</tr>
<tr>
<td>com.ofss.digx.domain.fileUpload_entity.FileIdentifierRegistration</td>
<td>HNU</td>
<td>IT Registration</td>
<td>Y</td>
</tr>
<tr>
<td>com.ofss.digx.domain.fileUpload_entity.FileDetails</td>
<td>HNU</td>
<td>Uploaded File Details</td>
<td>Y</td>
</tr>
</tbody>
</table>

We match determinant type to market entity, then business unit and then legal entity.
On the first match, we derive the external systems using Extensible Points in Service Tier 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.

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 which 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 return 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 Extensible Points in Service Tier Preference against which one can specify the adapter that one want to be overridden by.

E.g.
**4.7.2 Adding a custom adapter**

Please follow below steps for adding a new custom adapter:

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

- The MANIFEST.MF file within this project should have implementation title and implementation version. The implementation title should also capture the phrase 'CZ' to indicate that it is a customized adapter package.

  Implementation-Title: CZUBS

  Implementation-Version: 12.4

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

  Insert into digx_fw_config_all_b (PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMMARY_TEXT,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS,OBJECT_VERSION_NUMBER,EDITABLE,CATEGORY_DESCRIPTION)
  values ('01','extxfaceadapterconfig', 'CZUBS12.4,UBS12.4,ipm1.0','N',null,'','ofssuser',sysdate,'ofssuser', sysdate,'Y',1,'N',null);

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

**Customizing existing adapters (Custom Adapter)**

If an added functionality or replacement functionality is required for an existing adapter or existing method in an adapter, the customization developer has to develop a new adapter and corresponding adapter factory and override the method in a new custom adapter class. The custom adapter would have to override and implement the methods which need changes.
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 `CustomLoanApplicationRequirementAdapterFactory`. As a guideline, the custom adapter should extend the existing adapter and override the methods which needs to be replaced with new functionality.

For Example:
Custom Adapter Configuration

```sql
insert into digx-fw-config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)
values ('IS_LOAN_APPLICATION_REQUIREMNT_ADAPTER_CUSTOM', 'customadapterconfig', 'true', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
```

Mock Adapter

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

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

```java
package com.ofis.digx.app.loan.application.impl.application;

import java.util.logging.Logger;

/**
 * Mock adapter represents the intermediate layer required for communicating with third party host system as part of
 * Loan requirement for a party.
 * Following operations are supported by this adapter.
 * 
 * @link LoanApplicationRequirementMockAdapter
 * @link LoanApplicationRequirementMockAdapter.createLoanApplicationRequirement
 * @link LoanApplicationRequirementMockAdapter.createLoanApplicationCreateResponseDTO
 */

public class LoanApplicationRequirementMockAdapter implements LoanApplicationRequirementMockAdapter {

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

  /**
   * Instance of @link MultiEntityLogger.
   */
  private static final MultiEntityLogger LOGGER = MultiEntityLogger.getLogger(ThisComponentName);

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

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

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

Mock Adapter Configuration

```
insert into digx_FW_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)
values ('PARTY_COLLECTION_ADAPTER_MOCKED', 'adapterfactoryconfig', 'true', 'N', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);
```
4.8 Outbound web service extensions

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

Using your own web service constants

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

```java
package com.ofsc.dclgs.common;

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

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

    public static final String OBDX_MANUFACTURING_APPLICATION_SERVICE = "/ProductManufacturingApplicationService.xml";

    public static final String OFFER_INQUIRY_APPLICATION_SERVICE = "/OfferInquiryApplicationService.xml";

    public static final String PURPOSE_APPLICATION_SERVICE = "/PurposeApplicationService.xml";

    public static final String SUBMISSION_CREATION_APPLICATION_SERVICE = "/SubmissionCreationApplicationService.xml";

    public static final String SUBMISSION_PRODUCT_APPLICATION_SERVICE = "/SubmissionProductApplicationService.xml";

    public static final String DETAILED_APPLICATION_TRACKER_APPLICATION_SERVICE = "/DetailedApplicationTrackerApplicationService.xml";

    public static final String FETCH_ALL_PRODUCT_GROUPS = "/fetchAllProductGroups";

    public static final String FETCH_ALL_PRODUCTS_FOR_GROUP_CODE = "/fetchAllProductsForGroupCode";

    public static final String FETCH_OFFERS_LINKED_TO_PRODUCT = "/fetchOffersLinkedToProduct";

    public static final String FETCH_PURPOSE_CODE_LINKED_TO_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 JAXWSFactory to create instances of the desired service stubs. The rest of the custom adapter implementation is the same as mentioned in the section 5.3

For example:
4.9 Security Customizations

OB DX 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.

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

Security Configuration

Method checkAccess() in class AbstractSecureApplication called from ApplicationService checks flag IS_SECURITY_ENABLED. If this flag has value true then it checks for security. Below image depicts same.

```
protected void checkAccess(String serviceId, Object... inputObjects) throws IllegalAccessException {
    if (logger.isDebugEnabled()) {
        logger.log(1, "Access: ", serviceId);
    }
    logServiceCall(serviceId);
    this.currentService = serviceId;
    //This will get cleared automatically in final interaction.close()
    if (StringHelper.isNotNull((String) ThreadAttribute.get(ThreadAttribute.CURRENTLY_EXECUTING_SERVICE))
        ThreadAttribute.set(ThreadAttribute.CURRENTLY_EXECUTING_SERVICE, serviceId);
    }
    boolean propertyValue = false;
    if (securityConstants == null) {
        securityConstants = ConfigurationFactory.getInstance().getConfiguration(SecurityConstants);
    }
    String IS_COLLECTION BATCH = (String) ThreadAttribute.get(ThreadAttribute.IS_COLLECTION BATCH);
    if (IS_SECURITY_ENABLED = false) {
        boolean isCreateWorkItemEnabled = false;
        if (securityConstants == null) {
            securityConstants = ConfigurationFactory.getInstance().getConfiguration(SecurityConstants);
        }
        if (isCreateWorkItemEnabled) {
            if (isCreateWorkItemEnabled) {
                List<String> services = fetchAllOVeriddenServices();
            }
        }
    }
```

Following Query is used to configure IS_SECURITY_ENABLED flag in database.

```
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_SECURITY_ENABLED, 'SecurityConstants', 'true', 'N', '', 'Security Constant', 'ofssuser', '30-JUN-15 04.04.49.000000 PM', 'ofssuser', '30-JUN-15 04.04.49.000000 PM', 'Y', 1);
```

checkAccess() calls method checkAccessInternal(). This method delegates the calls to the appropriate methods during security access check process. First of all, if the service is blacked out for the period in which the user is accessing it, then he will not be allowed to perform the transaction. Access policies are seeded by the product during installation and can be added or
modified by bank administrator. The bank administrator can either permit or deny access for particular user role to a particular service.

checkAccessInternal has call to method assertAccessPolicies.

assertAccessPolicies is responsible to check whether user has access to a particular service or not.

4.9.1 Out of box seeding of policies

When the application is installed, access policies are seeded such that a separate policy is created for each service (resource) and application role combination.

The application is shipped with two CSV files – Admin.csv and Clip.csv containing the policy data and the installer uses a tool called SeedPolicies.jar to seed these policies in database from where it will be used for application.

4.9.2 Administrative console for policy management

The bank administrator can grant or deny access of a user role to any services. Bank administrator can also create/modify/delete policies. Detailed description of these features is provided in the user manual ‘User Manual Oracle Banking Digital Experience Core’ under the heading ‘Role Transaction Mapping’. Below are some terminologies used in further discussion.

Policy Domain: A Policy Domain is a container for multiple policies created in the system.

One policy domain holds group of policies defined for a specific region. Administrator can create, update, delete Policy domain. For example below diagram shows two policy domains namely ADMIN and CLIP created in the system.

Policy: Policy defines mapping between resources/entitlements and application roles/enterprise roles.

Each policy can be designed to PERMIT or DENY access to resources depending on business
regulations. Policy can have multiple resources or multiple application roles. For example below screen shows a policies related to DemandDeposit service.

**Resource**: A resource is fully qualified name of the service. For example below screens the resource `com.ofss.digx.app.dda.service.core.DemandDeposit.listStatements` available in the first policy.

**Application Role**: An application role is logical grouping of one or more enterprise roles. For example, below screen shows that the role ‘Checker’ is added in above policy.
Above setup will ensure that a user with ‘Checker’ role will have access to the com.ofss.digx.app.dda.service.core.DemandDeposit.listStatements service.

If administrator needs to deny this access then the Effect can be changed to ‘Deny’ as shown in the screen below.

So there is no need to delete existing policy, only the access can be denied.

4.9.3 Seeding policies for new services

There are two ways in which policies can be created for newly added services.

Manual policy creation
This approach will usually be used when the number of services added is small. The bank administrator can create new policies and add necessary application roles and resources in this policy. The policy creation screen is shown below for reference. More details can be obtained from the user manual ‘User Manual Oracle Banking Digital Experience Core’.

One this creation is done, then the service added in this policy will be accessible to the user.

Seeding policies using tool

OBDX provides a tool SeedPolicies.jar. This tool can be used if the number of services is large. This tool accepts a CSV file in below format.

```
amonymous_role,com.ofss.digx.app.chatbot.service.ChatbotLogin.login,Service,perform,GRANT
Checker,com.ofss.digx.app.fileupload.service.UploadService.create,Service,perform,GRANT
Viewer,com.ofss.digx.app.fileupload.service.UploadService.canceledFile,Service,view,GRANT
Maker,com.ofss.digx.app.fileupload.service.UploadService.downloadFile,Service,view,GRANT
Checker,com.ofss.digx.app.fileupload.service.UploadService.readRecord,Service,approve,GRANT
```

The first field here is application role name. Second field is the resource name i.e. fully qualified name of the service along with method. Third field will be fixed as ‘Service’. Fourth field indicate
what type of access you want to provide. Possible values are ‘view’, ‘perform’ and ‘approve’. The last option will be fixed as ‘GRANT’.

Once a CSV file is prepared in this format, then the SeedPolicies.jar can be used to upload these grants as policies in the database. This tool creates separate policies for each service and role combination. Command line arguments for this tool are as follows:

Command line arguments:
- CSV file names separated by ‘,’
- PolicyDomainNames separated by ‘,’
- Output log and error file location
- (EnvironmentName)-{driveName}, {username}, {password}, {connectionurl:port:SID}
- (EnvironmentName)-{driveName}, {username}, {password}, {connectionurl:port:SID}

For example:
C:\\OBDX\\Customer.csv,C:\\OBDX\\AuthAdmin.csv CLIP,ADMIN C:\\OBDX_LOGS\\

Dev-oracle.jdbc.OracleDriver,<DEV schema name>,<password>,jdbc:oracle:thin:@<database IP address>:<port>:<SID>

UAT-oracle.jdbc.OracleDriver,<UAT schema name>,<password>,jdbc:oracle:thin:@<database IP address>:<port>:<SID>

4.10 Miscellaneous

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

4.10.1 Task Configurations

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

- Limits
- Approvals
- Two Factor Authentication
- Transaction Blackout
- Working Window
Guidelines for formulating a task code are as follows:

A task code should ideally comprise of 3 parts:

1. **Module Name**: The first 2 alphabets representing the module to which the service in question belongs. eg TD represents Term Deposits module.

2. **Task Type (type of service)**: OBDX supports the following 6 types of services.
   a. **FINANCIAL_TRANSACTION(F)**: Any transaction as a result of which there is a change in the status of the finances of accounts of the participating parties. In general any transaction that involves monetary transfer between parties via their accounts. Few examples include Self transfer, New deposit(Open term deposit), Bill payment etc.
   b. **NONFINANCIAL_TRANSACTION(N)**: Any transaction that pertains to an account but there is no monetary payment or transfer involved in it. For example Cheque book request.
   c. **INQUIRY(I)**: Any read only transaction supported in OBDX that does not manipulate any business domain of the financial institution. For example list debit cards, read loan repayment details, fetch term deposit penalties etc.
   d. **ADMINISTRATION(A)**: Transactions performed by bank admins and corporate admins for a party come under this category. Few examples of such transactions include limit definition, limit package definition, user creation, rule creation and various others.
   e. **MAINTENANCE(M)**: Maintenances done by a party for itself fall under this category. Maintenance transactions performed by a non admin user which does not involve any account or monetary transaction comprise of this transaction type. Example add biller.
   f. **COMMON(C)**: Common transactions include transactions which do not fall under any of the above mentioned categorization. Example login.

So 1 alphabet F,N,I,A,M or C for each of the above mentioned task types respectively forms the second part of the task code.

3. **Abbreviation for service name**: A 3 to 10 lettered abbreviation for the service name. Example OTD for Open Term Deposit.

All the above mentioned 3 parts are delimited by an underscore character. Example: TD_F_OTD where TD represents module name. F represents that its a financial transaction i.e. task type and OTD is the abbreviated form of the transaction(service) name.

Task Aspects:

An 'aspect' of a task is a behavior or feature supported by the task. OBDX framework defines a set of aspects that can be supported by a task in the system. These aspects need to be configured in table DIGX_CM_TASK_ASPECTS. So if a task supports given aspect, then only its entry should be made in this table. If for any task, entry does not exist in this table for given aspect, then system treats it as that aspect is not supported by the task.
Additionally an aspect can be temporarily disabled using the ‘ENABLED’ column of this table. If the ‘ENABLED’ value is set as ‘N’, then system will treat it as this aspect is not supported by the task. Note that if a task is never going to support an aspect, then its entry should not be there in DIGX_CM_TASK_ASPECTS table. The ‘ENABLED’=‘N’ option for disabling aspect should be used only when the task generally supports the aspect but it needs to be disabled for small duration.

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

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

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grace-period</td>
<td>Indicates that the task supports grace period. Grace period is an additional period offered by Approval framework for approving a transaction</td>
</tr>
<tr>
<td>ereceipt</td>
<td>Indicates that the task supports generation of e-receipts</td>
</tr>
<tr>
<td>audit</td>
<td>Indicates that the task supports audit logging</td>
</tr>
<tr>
<td>2fa</td>
<td>Indicates that the task supports two factor authentication</td>
</tr>
<tr>
<td>working-window</td>
<td>Indicates that the task supports working window</td>
</tr>
<tr>
<td>approval</td>
<td>Indicates that the task supports approval</td>
</tr>
<tr>
<td>blackout</td>
<td>Indicates that the task supports blackout</td>
</tr>
<tr>
<td>limit</td>
<td>Indicates that the task supports limit</td>
</tr>
</tbody>
</table>

Steps to register a task with OBDX:

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

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

Extensibility Guide
As evident from the above query example Tasks have a hierarchy. Every task might have a parent task denoted by the task code value held by the PARENT_ID column of DIGX_CM_TASK. In most of the cases its a 3 level hierarchy.

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

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

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

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

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

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

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

Secondly the below query fulfills the requirement mentioned in this step.

insert into DIGX_CM_RESOURCE_TASK_REL (ID, RESOURCE_NAME, TASK_ID, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) values ('1',

Extensible Points in Service Tier

55
The aforesaid procedure enrolls your newly created service as a task in OBDX.

**Limit Configuration**

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

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

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

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

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

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

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

```sql
```

4. Code a LimitDataEvaluator for the task. LimitDataEvaluator is a class that implements ILimitDataEvaluator interface present in com.ofss.digx.finlimit.core jar. This is a functional interface with a single method having signature as shown below:

```java
/**
 * provide (@link LimitData) of currently executing task.
 */
```
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 need to be enabled. For instance consider the service to open a term deposit. Signature of the service is as shown below.

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

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

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

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

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

- **Register the LimitDataEvaluator coded in Step 3.**

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

```
Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID,PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) values (<<task code>>, 'limitDataEvaluator', <<limitDataEvaluator>>, 'N', 'Limit data evaluator for <<service name>> service', null, 'ofssuser', sysdate,  'ofssuser', sysdate, 'A', 1);
```
In the above query <<task code>> is the task code for the service, <<limitDataEvaluator>> is the fully qualified name of the class coded in Step 3. <<service name>> is a descriptive name for the service.

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

---

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

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

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

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

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

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

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

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

Register the TargetEvaluator coded in Step 5.

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

Payee limits are Periodic and Transactional limits applied on a Payee.
This needs an insert in **DIGX_FL_TARGET_EVALUATOR** table as shown below:

```
Insert into DIGX_FL_TARGET_EVALUATOR (TASK_CODE, TARGET_TYPE, EVALUATOR, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS, OBJECT_VERSION_NUMBER) values ('<<task code>>', 'PAYEE', '<<TargetEvaluator>>', null, 'target evaluator for <<service name>> service', 'ofssuser', sysdate, 'ofssuser', sysdate, 'Y', 1);
```

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

The aforesaid procedure enables limits for a task in OBDX.

**Approval Configuration**

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

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

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

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

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

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

```java
protected T toDomainObject(D requestDTO, T transaction) 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 term deposit. Signature of the service is as shown below.

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

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

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

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

  transaction = new AmountAccountTransaction();

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

  transaction = new AccountTransaction();

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

  transaction = new PartyTransaction();

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

  transaction = new PartyTransaction();

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

  transaction = new Transaction();

- **Call to AbstractApprovalAssembler**:

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

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

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

If the newly created task is of type `ADMINISTRATION` and the maintenance is not specific to a party then the approval assembler to be registered against your service is `om.ofss.digx.framework.domain.transaction.assembler.GenericDTOTransactionAssembler`.

2 Factor Authentication Maintenance is a fine example of such transactions. The service id for this transaction is `com.ofss.digx.app.security.service.authentication.maintenance.AuthenticationMaintenance.create`.

The below query fulfills the requirement of this step:

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

In all other cases where you have implemented a custom approval assembler as per the guidelines in step 2, the fully qualified class name of that approval assembler will be registered against your service. The below query fulfills the requirement of this step:
Insert into DIGX_FW_CONFIG_ALL_B
(PROP_ID,
CATEGORY_ID,
PROP_VALUE,
FACTORY_SHIPPED_FLAG,
PROP_COMMENTS,
SUMMARY_TEXT,
CREATED_BY,
CREATION_DATE,
LAST_UPDATED_BY,
LAST_UPDATED_DATE,
OBJECT_STATUS,
OBJECT_VERSION_NUMBER)
values
(<<service id>>,  
'approval_assembler',  
<<ApprovalAssembler>>,  
'N',  
'assembler class for conversion from DataTransferObject to Transaction domain',  
'assembler class for conversion from DataTransferObject to Transaction domain',  
'ofssuser',  
sysdate,  
'ofssuser',  
sysdate,  
'A',  
1);

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

The aforesaid procedure enables approvals for a task in OBDX.
5. **Architecture of GUI Tier**

Below diagram shows structure of the UI artifacts and some of the important artifacts are explained subsequently.
6. **Extensible Points in GUI Tier**

This article provides the guidelines for UI Extensibility.

6.1 **Theme and Brand**

- SCSS Variables are available for the modifications. You can change the variables available in `variable.scss` located at `<USER_SEGMENT>/scss/_base` folder. In this file, you can override the variable values which are present in `framework/scss/_base/_variable.scss`. We are not allowing changes in the framework file. Same functionality you can achieve by the Branding.
- We are not allowing adding new styles in the core UI.
- For the Images you are free to do modifications.

6.2 **Component Extensibility**

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

6.2.1 **Adding New and Overriding Existing Components**

If you want to add a new component, place that component in `<CHANNEL_ROOT_PATH>/extensions/components`. It follows the same structure which is present in the components folder. Same thing is applicable for the existing components. If you want to change anything, then copy that component and place it in `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 remains the 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`, it will take precedence over the `<CHANNEL_ROOT_PATH>/components`. For it, we maintain the list of components available in extensions in `<CHANNEL_ROOT_PATH>/extensions/extension.json`. In this file, the user should provide the list of all the components. For example:
If newly created component needs the styles put related style in `<CHANNEL_ROOT_PATH>/scss/_<componentName>.scss` and import this file in `<CHANNEL_ROOT_PATH>/scss/_extension.scss`.

### 6.2.2 Add / Modify Validations

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

In this file Implementer can add or override validations.

For Example: The pattern to be updated can be found corresponding to the input field from the respective HTML. If you need to change the pattern which validate mobile number. Add updated pattern in this file as below.

```javascript
define(["oj10n!resources/nls/obdx-locale"], function(locale) {
    "use strict";
    var Locale = {
        validations: {
            MOBILE_NO: [{
                type: "RegExp",
                options: {
                    pattern: /^\+\d{1,3}[\- ]?\d{10}$",
                    messageDetail: locale.messages.MOBILE_NO
                }
            }
        }
    }
    return Locale;
});
```
6.3 Calling custom REST service

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

For example see the snippet below:

```javascript
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);
};
```
6.4 **Dashboard and JSON context extension**

For every application you can override following three properties

- **User Type**: Type of user
- **Context**: folder location from which json will be picked
- **Dashboard**: The dashboard to be loaded for a particular Application Role.

To evaluate these parameters there are three respective functions and one array in `<CHANNEL_ROOT_PATH>/extensions/override/extensions.js` file

**customRoles** Array,

**Description**,  
Add any custom Application Roles created in this array as string values.

**evaluateUserType** function,

**Arguments** ( roles: [] )

roles[] (array) will contain all the roles mapped to the user

defaultSegment (string) will contain system evaluated default segment

**Return Value** (String),
Should return string representing what is the segment for respective component (possible values: ANON | CORPADMIN | RETAIL | CORP | ADMIN)

**Description**,  
If you want to change user type for your application, you can do so by implementing this function and return required user type for your application.

**evaluateContext** function,

**Arguments** ( segment, roles )

roles[] (array) will contain all the roles mapped to the user

segment (string) will contain evaluated user segment

**Return Value** (String)
Should return string representing what is the context for respective component

(Possible values: index | retail | corporate | admin | corp-admin or any custom value from CHANNEL_ROOT)

**Description.**

This function will set context for your components to fetch json artifacts from correct folder. Return possible value and then json will be fetch from respective location. Using this you can specify menu for your context. For example, if you have a new application role as SpecialMaker, you need to return fully qualified name from CHANNEL_ROOT, as “extensions/SpecialMaker”.

Please note that all the JSON lookup will change for the application. For this example above, you would copy corporate/json to extensions/SpecialMaker/json.

Also, new location for menu.json will be “extensions/SpecialMaker/json/menu.json”

**evaluateDashboard** function,

*Arguments* (roles, rolesList)

- role (String) will contain the first application role matched from customRole array.
- rolesList (Array[]) will contain all the application roles assigned to a user.

*Return Value* (String)

Should return string representing what is the dashboard required.

Possible string values are (grouped by enterprise role):

- **Retail:** “dashboard”, “term-deposits”, “demand-deposits”, “loans”, “trends”, “cards”
- **Corporate:** “maker”, “approver”, “viewer”, “term-deposits”, “demand-deposits”, “loans”
- **Admin:** “approver”, “maker”, “checker”, “dashboard”, “configurations”
- **Corporate Admin:** “approver”, “checker”, “maker”, “dashboard”
Description,

This function will set dashboard for your custom application role to be one of the shipped dashboards.

Note:

• Core functionality of Framework Elements like (header, dashboard, menu etc.) are not available for the modification. You can customize menu options.

• If any component is present in <CHANNEL_ROOT_PATH>/extensions/components will take precedence over the <CHANNEL_ROOT_PATH>/components.

• All components available under component folder are available for the extension.
7. Libraries

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

7.1 OBDX Libraries

This section provides information about various OBDX libraries that are provided out of the box. These libraries are packaged in the enterprise application obdx.app.domain.ear.

7.1.1 Core/Framework Libraries

Provide infrastructure features of OBDX platform.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.infra</td>
<td>Provides basic infrastructure classes.</td>
</tr>
<tr>
<td>com.ofss.digx.infra.audit</td>
<td>Provides basic infrastructure classes for audit.</td>
</tr>
<tr>
<td>com.ofss.digx.infra.crypto</td>
<td>Provides cryptography functions such as hash generation, public private key generation and symmetric cryptography provider.</td>
</tr>
<tr>
<td>com.ofss.digx.framework.domain</td>
<td>Provides base classes for entities, assemblers, repositories etc.</td>
</tr>
<tr>
<td>com.ofss.digx.framework.rest</td>
<td>Provides infrastructure classes for OBDX REST services</td>
</tr>
<tr>
<td>com.ofss.digx.framework.fileupload</td>
<td>Provides infrastructure classes for bulk file processing</td>
</tr>
<tr>
<td>com.ofss.digx.appx.core.rest</td>
<td>Provides infrastructure classes for OBDX REST services</td>
</tr>
<tr>
<td>com.ofss.digx.common</td>
<td>Provides complex data types used in OBDX application</td>
</tr>
<tr>
<td>com.ofss.digx.datatype</td>
<td>Provides all enumerations</td>
</tr>
<tr>
<td>com.ofss.digx.enumeration</td>
<td>Provides base classes for application services,</td>
</tr>
</tbody>
</table>
### Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Interaction classes etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.cloud.extension</td>
<td>??</td>
</tr>
<tr>
<td>com.ofss.digx.fileupload.templates</td>
<td>Provides file upload handlers, entities and DTOs</td>
</tr>
<tr>
<td>com.ofss.digx.finlimit.core</td>
<td>Provides core classes for financial limits processing</td>
</tr>
</tbody>
</table>

#### 7.1.2 DTOs

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.app.xface</td>
<td>Provides all request, response or plain DTOs used in services</td>
</tr>
<tr>
<td>com.ofss.digx.appcore.dto</td>
<td>Provides DTOs used in infrastructure services</td>
</tr>
</tbody>
</table>

#### 7.1.3 Modules

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.module.access</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.account</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.alerts</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.approval</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.audit</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.brand</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.budget</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.card</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.chatbot</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.collaboration</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.common</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.config</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.content</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.dda</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.extension</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.fileupload</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.finlimit</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.goal</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.loan</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.location</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.me</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.mobile</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.origination</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.party</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.payment</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.pm</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.report</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.security</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.sms</td>
<td></td>
</tr>
<tr>
<td>com.ofss.digx.module.spendanalysis</td>
<td></td>
</tr>
</tbody>
</table>
### 7.1.4 Endpoints

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.appx.service.rest</td>
<td>Provides REST endpoint classes for all modules</td>
</tr>
</tbody>
</table>

### 7.1.5 Adapters

These are packaged into obdx.extsystem.domain.ear

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ofss.digx.extxfinterface</td>
<td>Provides all external interfaces</td>
</tr>
<tr>
<td>com.ofss.digx.extxfinterface.sms.dbAuthenticator</td>
<td>??</td>
</tr>
<tr>
<td>com.ofss.digx.extxfinterface.&lt;Host Name&gt;.impl</td>
<td>Provides adapter implementations of the external interfaces for particular host</td>
</tr>
<tr>
<td>com.ofss.&lt;Host Name&gt;.soap.client</td>
<td>Provides stubs used for communicating with host</td>
</tr>
</tbody>
</table>
8. **Workspace Setup**

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

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

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

![Project structure](image)

**Figure 2 Project structure**

8.1 **Basic Setup**

Create a new workspace in Eclipse and launch that workspace.

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

<table>
<thead>
<tr>
<th>Classpath Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGX_LIB</td>
<td>Refers to the base location of OBDX libraries</td>
</tr>
<tr>
<td>OBP_LIB</td>
<td>Refers to the base location of OBP libraries</td>
</tr>
<tr>
<td>EXT_LIB</td>
<td>Refers to the base location of external libraries</td>
</tr>
</tbody>
</table>

The screen will look like this.
8.2 DTO (xface) project

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

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

Also, every Request DTO classes will have a separate Canonicalizer class responsible for reducing possibly encoded string parameters into its simplest form. This is therefore a very important part that should not be missed while writing any DTO classes. Similarly, for the Response DTO classes, we will have to write a separate Encoder class responsible for encoding the string back to its original form for HTML pages. The naming convention for Canonicalizer and Encoder classes will be - <RequestDTO_classname>Canonicalizer and <ResponseDTO_classname>Encoder respectively.

Project dependencies for xface project:
Not needed.

Classpath variables:
Extend the following classpath variables to the jars found in OBDX_Installer:
8.3 REST endpoint

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

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

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

**Project dependencies for REST project:**

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

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

**Classpath variables:**

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

<table>
<thead>
<tr>
<th>Classpath name</th>
<th>Jars to extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGX_LIB</td>
<td>com.ofss.digx.appcore.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.appcore.dto.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.common.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.datatype.jar</td>
</tr>
<tr>
<td>OBP_LIB</td>
<td>com.ofss.fc.appcore.dto.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.infra.jar</td>
</tr>
</tbody>
</table>
8.4 Module

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

1. **com.ofss.digx.app.<module_name>.service**: Add the Service Interface and Implementation class in this package. The name of Service class should be same as the name of the REST class created in the REST project. For instance, this package will have classes named `IAccount.java` and `Account.java` which are same as the REST class name for Account. This service class extends `AbstractApplication` of the DIGX framework.

2. **com.ofss.digx.app.<module_name>.service.ext**: Contains classes for extensions and executors. **Each** Service classes will have their own extension points. Refer mock workspace for more detail.

3. **com.ofss.digx.app.<module_name>.assembler**: Create `<module_name>Assembler` class inside this package. All Assembler classes extends `AbstractAssembler`.

4. **com.ofss.digx.domain.<module_name>.entity**: This package should include Entity class for the module. The name of entity class to be created should be same as REST as well as Service class names. For instance, it will have Account.java entity class for Account service and REST classes. Also known as Domain classes, they extend `AbstractDomainObject` taken from OBP libraries.

5. **com.ofss.digx.domain.<module_name>.entity_assembler**: Add a domain assembler class with the name `<module_name>DomainAsmber` in this package.
6. **com.ofss.digx.domain.<module_name>.entity.policy**: Add the business policy classes in this package to ensure the validation of business constraints added in these classes. Refer workspace attached with this document for more detail. Classes in this project are again one of the must-haves as far as enforcement of any system validation is concerned.

7. **com.ofss.digx.domain.<module_name>.entity.repository**: Contains repository class (<module_name>Repository.java) which invokes Repository adapter classes described in the next point. This class extends **AbstractDomainObjectRepository** of DIGX framework.

8. **com.ofss.digx.domain.account.entity.repository.adapter**: Add repository adapter interfaces, Local and Remote Repository Adapter classes in this project. If you are writing for the Account service, the naming convention of these classes should be I<module_name>RepositoryAdapter, Local<module_name>RepositoryAdapter, Remote<module_name>RepositoryAdapter respectively.

With this ends the package structure for service classes. The implementation of this project takes maximum time and involves majority of the DIGX service layer handling. It is therefore a very crucial part to look for while developing a REST API in DIGX.
Project dependencies for module project:
Add the following projects in the Java build path of this project:
*com.ofss.digx.cz.adapter, com.ofss.digx.cz.app.xface*

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

<table>
<thead>
<tr>
<th>Classpath variable name</th>
<th>Jars to extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGX_LIB</td>
<td>com.ofss.digx.framework.domain.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.infra.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.appcore.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.common.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.datatype.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.adapter.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.module.alerts.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.module.approval.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.module.party.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.enumeration.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.module.common.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.digx.appcore.dto.jar</td>
</tr>
<tr>
<td>OBP_LIB</td>
<td>com.ofss.fc.framework.domain.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.enumeration.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.datatype.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.obp.patch.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.infra.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.appcore.dto.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.appcore.jar</td>
</tr>
<tr>
<td></td>
<td>com.ofss.fc.framework.dto.jar</td>
</tr>
</tbody>
</table>
8.5 Adapter

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

**Project dependencies for adapter project:**

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

**com.ofss.digx.cz.app.xface**

**Classpath variables:**

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

<table>
<thead>
<tr>
<th>Classpath name</th>
<th>Jars to extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGX_LIB</td>
<td>com.ofss.digx.infra.jar</td>
</tr>
</tbody>
</table>

8.6 Adapter Impl

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

1. **<module_name>AdapterFactory.java**: Factory class to generate Adapter instances for every getAdapter request call. Returns either mock adapter or adapter to call host interface

2. **<module_name>Adapter.java**: A very essential Adapter class for a specific module which is entitled to call external host system

3. **<module_name>MockAdapter.java**: In case a call to host system needs to be skipped and local mocked data needs to be fetched, this adapter class can be used

**Project dependencies for adapter impl project:**

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

**com.ofss.digx.cz.adapter**

**com.ofss.digx.cz.app.xface**

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

**Classpath variables:**

Extend the following classpath variables to the jars found in OBDX_Installer:
Classpath name | Jars to extend
---|---
DIGX_LIB | com.ofss.digx.appcore.dto.jar
| com.ofss.digx.infra.jar
| com.ofss.digx.adapter.jar
| com.ofss.digx.datatype.jar

OBP_LIB | com.ofss.fc.framework.dto.jar

**Database Scripts to be added:**

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

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

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

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

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

Package customized code in library EAR:

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

To generate an EAR in eclipse, we need to create an Enterprise Application Project and include the required project/s during the creation. I am adding screenshot for creating `com.ofss.digx.cz.appx.service.rest.ear`, similar changes can be made for second EAR as well.

Steps to generate EAR in Eclipse:

Create an Enterprise Application Project (EAP) from the “New” option. Give details as mentioned in the Figure 3 and click on Next button.

![Figure 3 Creating new EAR Application Project](image-url)
Select all the projects created above. Do not check the 'Generate application.xml deployment descriptor' option as this EAR will be deployed as a library.

![New EAR Application Project]

Click on Finish to create the EAR.

**Inject customized REST into obdx.app.rest.idm.ear:**

- Open `com.ofss.digx.appx.service.rest.war`
- Create `customization.resources` file under `WEB-INF/classes`
- Add an entry of newly created REST class in this file
- Add a new servlet entry in `web.xml` with URL mapping `/cz/v1/*`
- Re-package this web application
• In META-INF/weblogic-application.xml, add library reference to library ear

Deploying application in Weblogic:
The two EARs created in the previous step should be deployed in the existing deployment setup. Please deploy the obdx.app.rest.idm.ear as an application and obdx.app.cz.domain.ear as library.

Test the application:
Once the application is up, please go to the deployments section of the Weblogic Server. In the control option, you’ll find the option to test the application. Just to verify, check whether the context-root of the custom application is changed to digx/cz. The request URL for testing this application will be –s

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