Oracle® Banking APIs

Extensibility Guide

Patchset Release 22.2.4.0.0
F99652-01
June 2024

Oracle Banking APIs Extensibility Guide, Patchset Release 22.2.4.0.0

F99652-01

Contents

Preface

Purpose

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

Audience

This document is intended for the following audience:

- Customers
- Partners

Documentation Accessibility

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

Access to Oracle Support

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

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and

the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Conventions

The following text conventions are used in this document:

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

Related Resources

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

- Oracle Banking APIs Installation Manuals

Screenshot Disclaimer

Personal information used in the interface or documents is dummy and does not exist in the real world. It is only for reference purposes.

Acronyms and Abbreviations

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

Table 1 Acronyms and Abbreviations

Abbreviation	Description
OBAPI	Oracle Banking APIs

1

Objective and Scope

1.1 Background

OBAPI 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.

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

1.2 Objective

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

OBAPI 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 OBAPI 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 OBAPI 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 OBAPI are summarized below:

- Strategic intent for enabling customers and partners to extend the application.
- Internal development uses the same principles for client specific customizations.

- Localization packs.
- Extensions by Oracle Consultants, Oracle Partners, Banks or Bank Partners.
- Extensions through the addition of new functionality or modification of existing functionality.
- Planned focus on this area of the application.
- Standards based.
- Leverage large development pool for standards based technology.
- Developer tool sets provided for as part of JDeveloper and Eclipse for productivity.

1.3 Scope

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

- Customizing OBAPI 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 OBAPI 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 OBAPI 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:

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

Out of scope

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

1.4 Structure

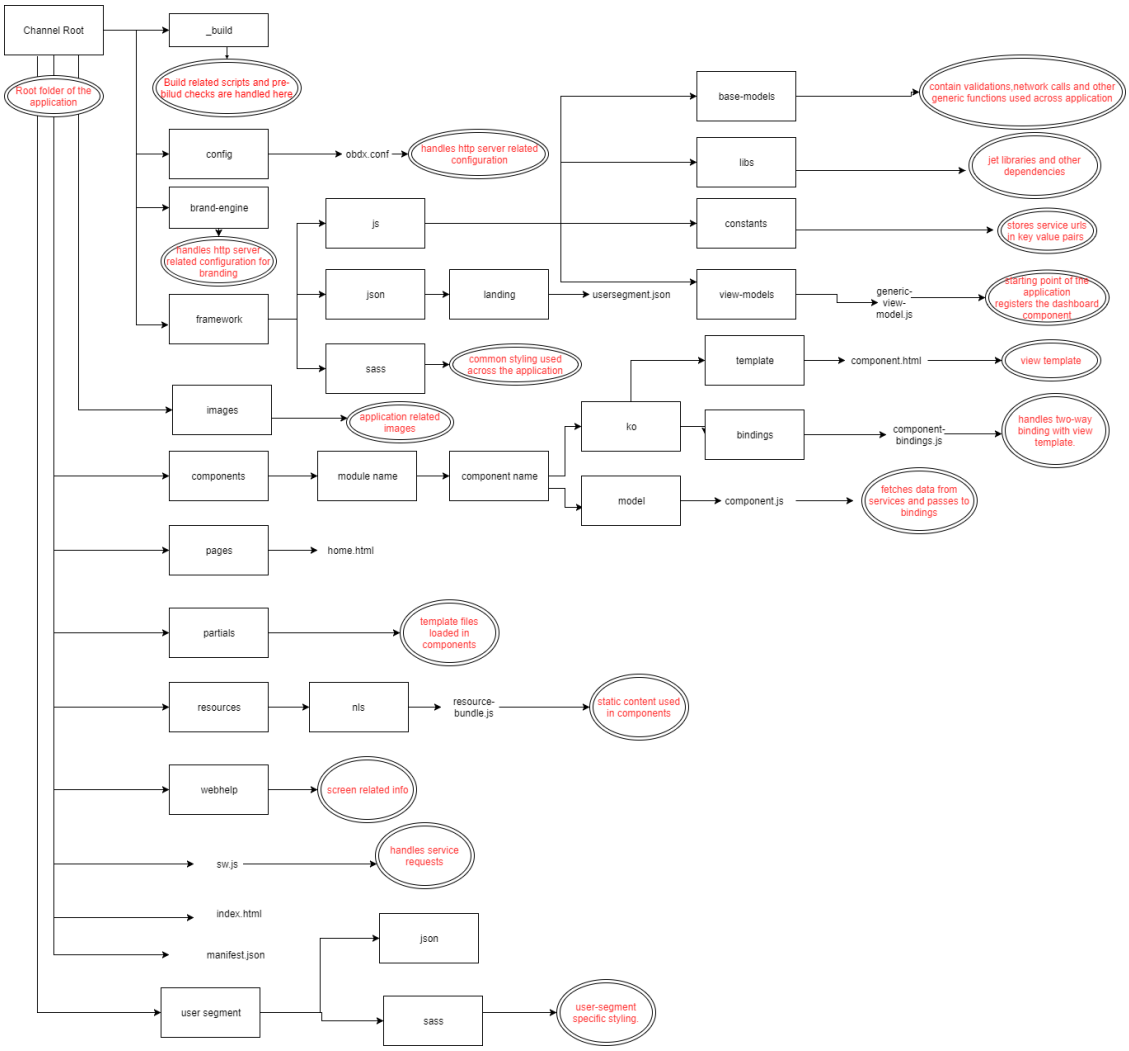
This document is organized into following chapters:

- Architecture of Service Tier: Provides overall architecture of the service tier of OBAPI 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 OBAPI 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 OBAPI out of the box along with their usage
- Workspace Setup: Provides step by step guidelines for setting up Eclipse workspace for extensibility
- Deployment: Provides information in packaging and deployment of the customized code on Weblogic server
- GUI Tier: Workspace Setup: Provides step by step guidelines for setting up workspace for GUI tier extensibility
- GUI Tier: Deployment: Provides information on packaging and deployment of customized GUI code on HTTP server
- Use Cases: This chapter discusses some of the extensibility points covered in earlier chapters with the help of some use cases.

2

Architecture of GUI Tier

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



3
Extensible Points in Service Tier

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

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

3.1 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.

3.2 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 API 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:

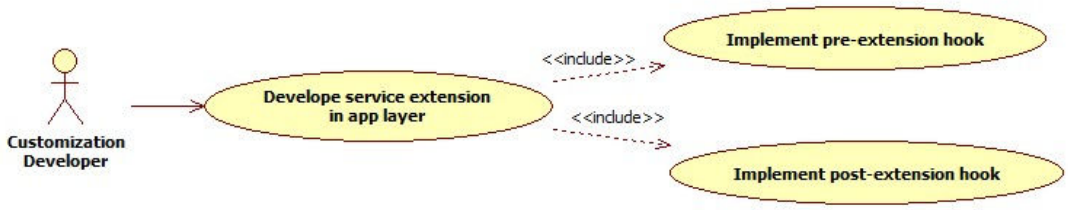
- Additional input validations
- Execution of business logic, which necessarily has to happen before going ahead with normal service execution.

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:

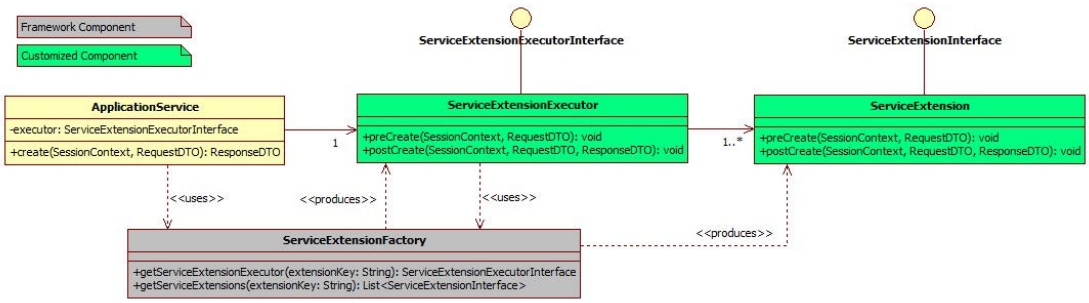
- Output response manipulation
- Custom data logging for subsequent processing or reporting.

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



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

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



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

3.3 Business Policy

OBAPI supports three types of validations

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

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

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

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

3.4 Dictionary

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

Data transfer object (DTO)

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

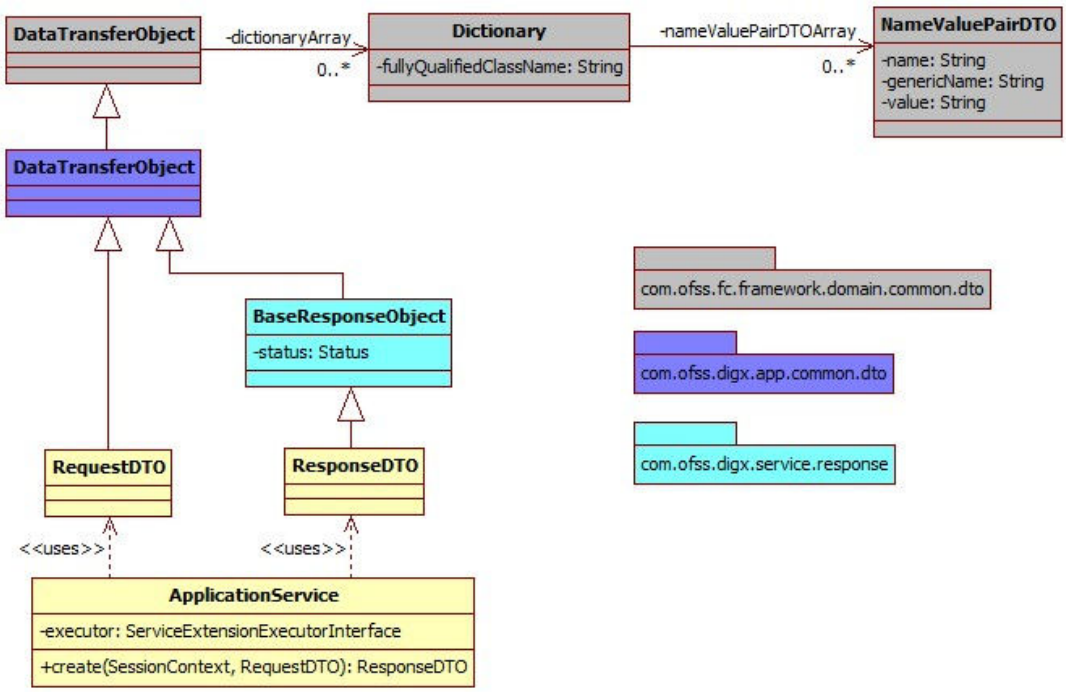
individual attribute values from the Transfer Object. All request response classes in OBAPI application services are modelled as data transfer objects.

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

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

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

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

}
```

Dictionary

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

Below class diagram shows the relationship between these classes.

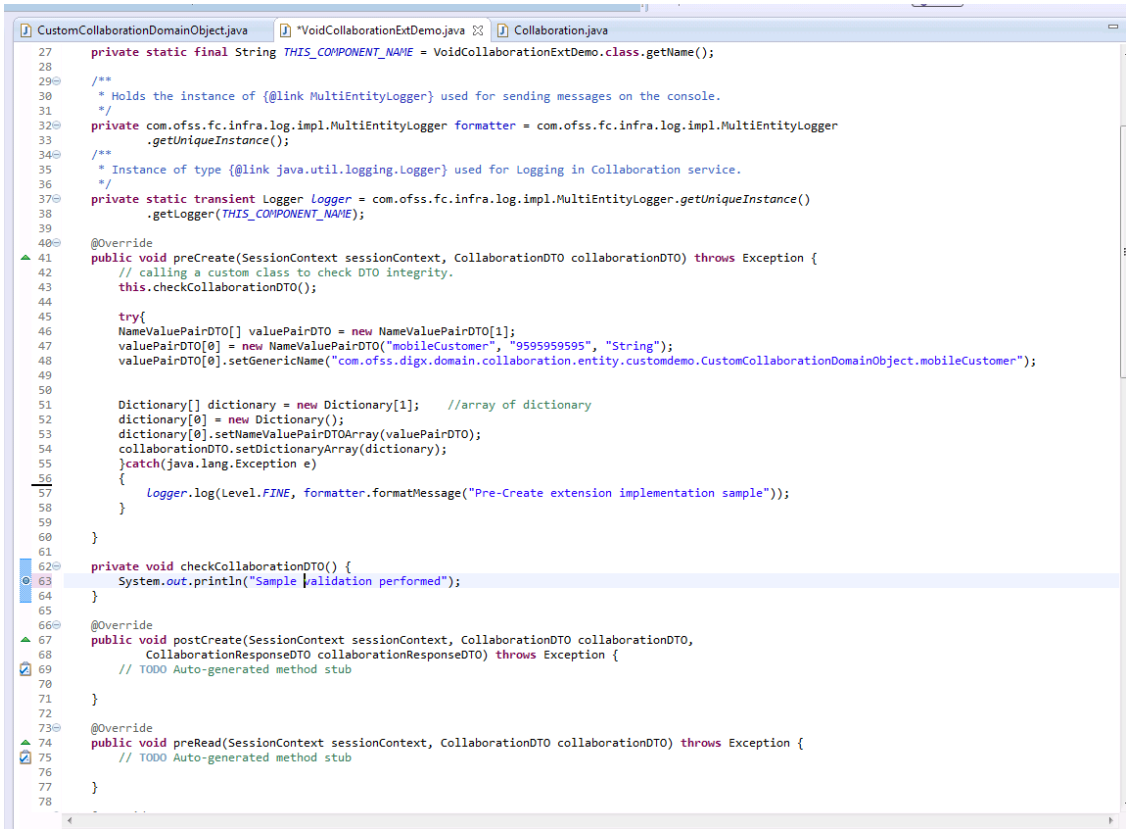


Dictionary class looks like

Dictionary.class CustomLoanApplicationExtension.java

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

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



3.5 Domain Extensions

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

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

3.6 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.

3.7 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

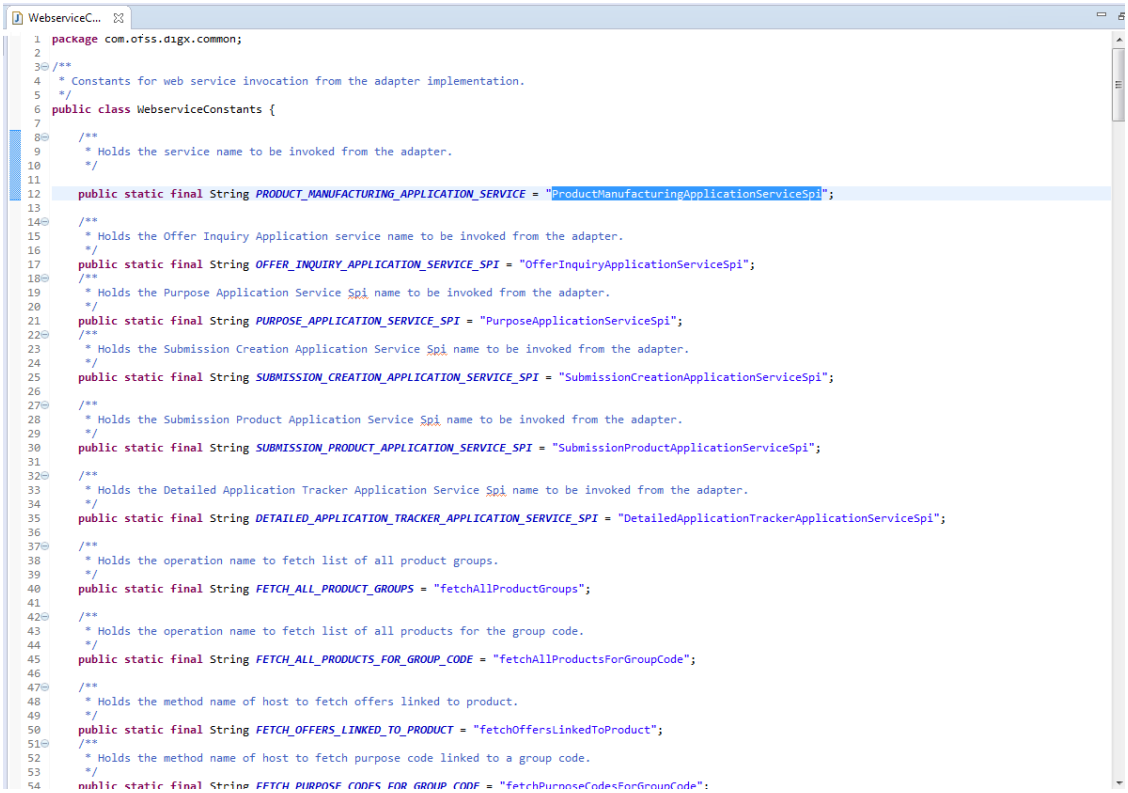
a different message format requiring message translation. Such functions, which do not form part of the consumer functionality, can be implemented in the adapter layer.

3.8 Outbound web service extensions

The outbound webservice configurations are set of properties defined to invoke services from the host. The host is the core bank system where the business logic for core banking facilities is written and contains the corresponding services to access that data. The existing OBAPI 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:



Web service configuration

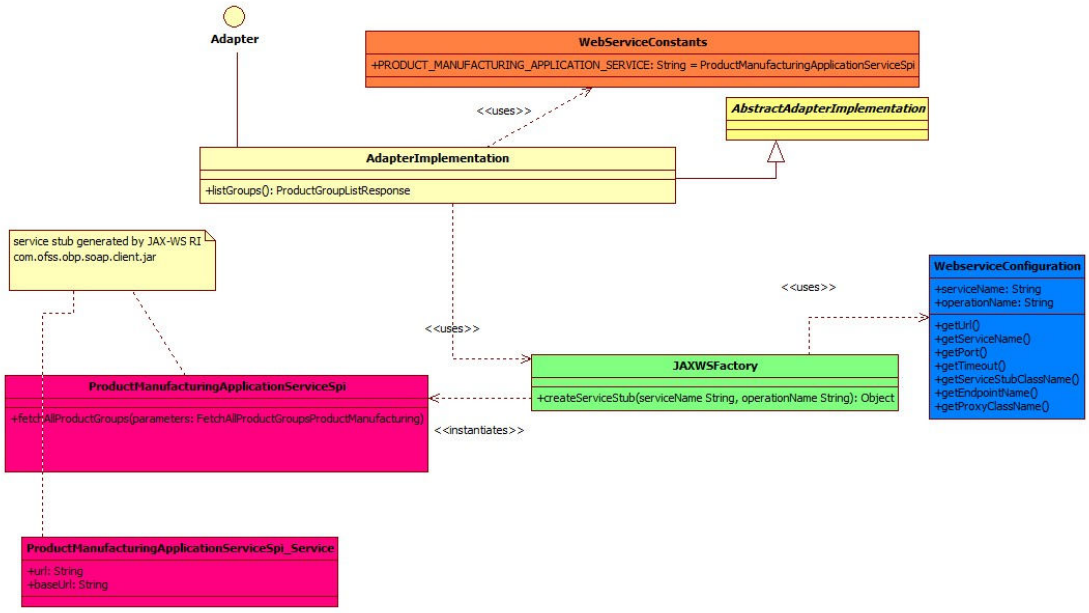
digx_fw_config_out_ws_cfg_b. Holds the entries for the host service endpoints.

For Example:

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

```
OBJECT_STATUS,
OBJECT_VERSION_NUMBER, ANONYMOUS_SECURITY_POLICY,ANONYMOUS_SECURITY_KEY_NAME)
values ('inquireApplication','BaseApplicationServiceSpi',

'
 http://ofss310406.in.oracle.com:8001/com.ofss.fc.webservice/services/origination/BaseApplicationServiceSpi?wsdl'
'

'','http://application.core.service.origination.appx.fc.ofss.com/
BaseApplicationServiceSpi',
1200000, 'BaseApplicationServiceSpi', '', '', 'BaseApplicationServiceSpiPort',
'com.ofss.fc.appx.origination.service.core.application.baseapplicationservice
```

Class Diagram



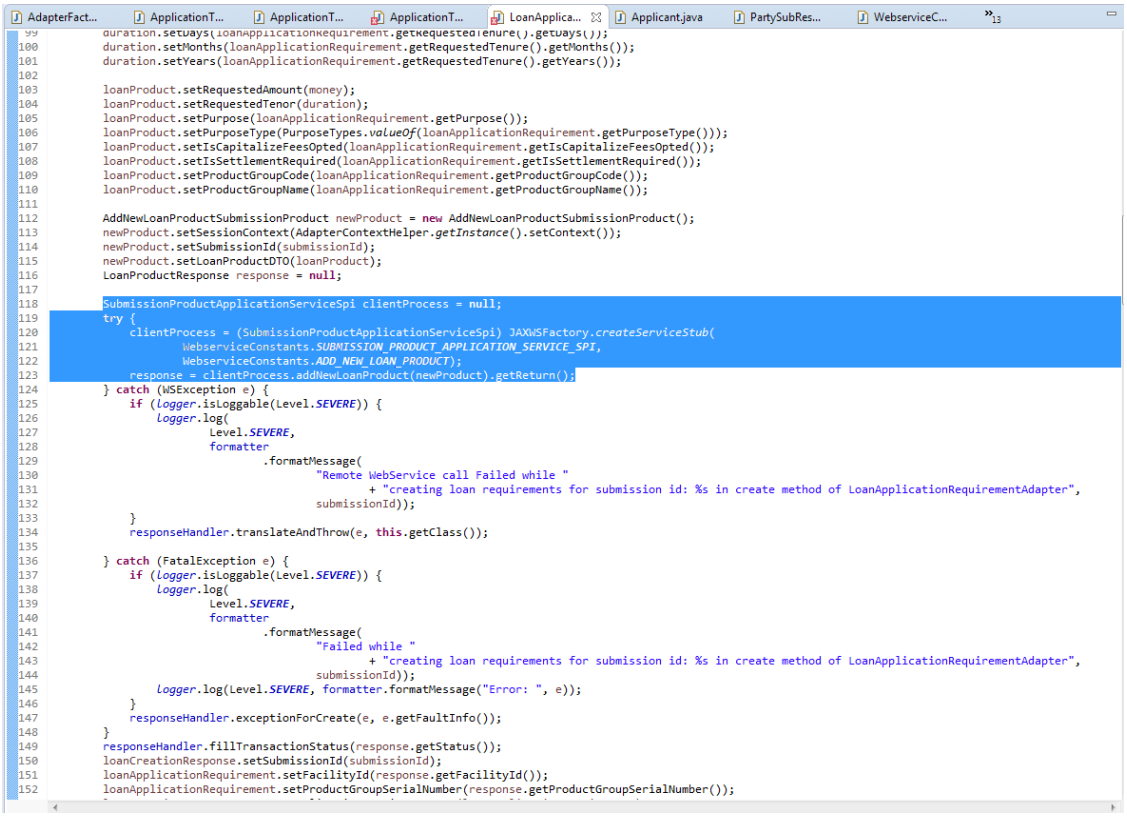
Client Jar

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

Custom Adapter

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

For example:



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

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

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

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

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

3.9 Security Customizations

OBAPI 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 OBAPI, 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.

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

3.10 Taxonomy Validations

For extensions in taxonomy validations, please refer to **Oracle Banking APIs Taxonomy Configuration Guide**

3.11 Authentication Extensibility

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

1. Need to write own Java class to implement authentication. Different classes can be used for different enterprise roles.
2. The custom classes must implement **com.ofss.digx.app.sms.handlers.credentials.ICredentialsManager**. Below methods need to be implemented -

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

public void create(AbstractUser user) throws Exception;

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

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

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

public boolean verify(String name, String newPassword, String currentPassword) throws Exception;

3. The classes' fully qualified names have to be updated in **DIGX_FW_CONFIG_ALL_B** against prop_ids - **credentials_manager_administrator**, **credentials_manager_corporateuser**, **credentials_manager_retailuser**. By default all three currently have **com.ofss.digx.app.sms.handlers.credentials.LocalCredentialsManager** as prop_value.

3.12 Miscellaneous

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

4

Extensible Points in Approval

This article explains extensible points in Approval framework.

- Adding New Rule Criteria

4.1 Adding New Rule Criteria

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

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

- Adding New Rule Criteria
- Implementing a Rule Criteria Handler
- Registering a Rule Criteria Handler

4.1.1 Adding New Rule Criteria

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

Worksheet | Query Builder

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

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

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

4.1.2 Implementing a Rule Criteria Handler

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

Override the methods

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

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

4.1.3 Registering a Rule Criteria Handler

The Rule Criteria Handler implemented in the step above needs to be registered in the system. To register make an entry in the table `DIGX_FW_CONFIG_ALL_B` as shown in the example query below.

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

values ('<<Rule Criteria Name>>', 'RuleCriteriaHandlerConfig','<<Fully
qualified name of the Handler implementation class
created in the step above>>', 'N', 'Specifies the class name of the Handler
for rule criteria type <<Rule Criteria Name>>.',
'Specifies the class name of the Handler for rule criteria type <<Rule
Criteria Name>>.',
'ofssuser', sysdate, 'ofssuser', sysdate, 'A', 1);
```

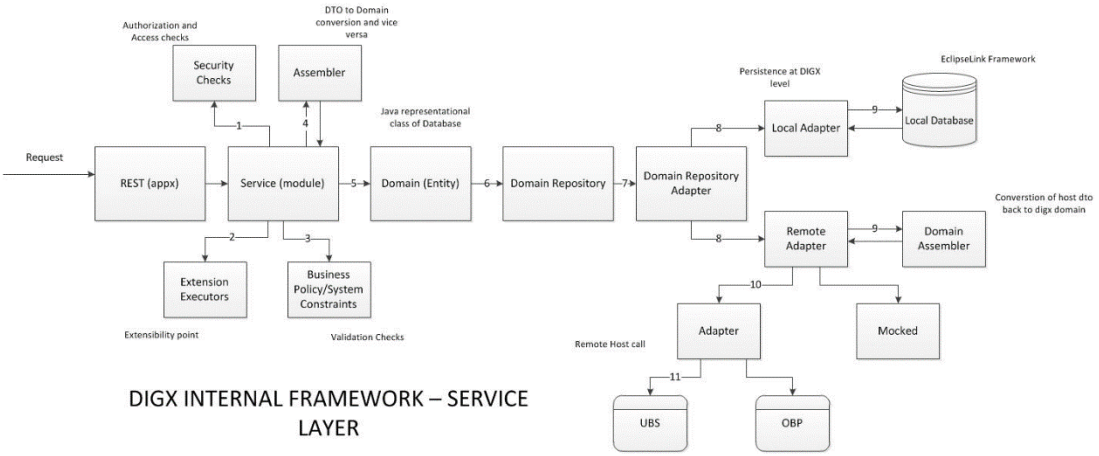
5

Architecture of Service Tier

Let’s go through the building blocks of OBAPI framework (also known as DIGX framework). To build a REST API, each of these framework components (as mentioned below) needs to be addressed and that’s why it becomes important to have a holistic idea about each of them.

The arrangement of all of these framework components can be clearly understood in the following diagram:

DIGX Service Layer



DIGX INTERNAL FRAMEWORK – SERVICE LAYER

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

repositories of DIGX application – Local Repository and Remote Repository. Eventually, the configured one out of these two will be invoked

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

6
Extensible Points in GUI Tier

This article provide the guidelines for UI Extensibility.

6.1 Theme and Brand

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

6.2 Component Extensibility

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

6.2.1 Adding New And Overriding Existing Components

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

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

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

Sample JSON for **extension.json**

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

In the same manner you can override the partial templates.

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

6.2.2 Add / Modify Validations

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

For OBAPI 18.1 at `<CHANNEL_ROOT_PATH>/extensions/validations/obapi-locale.js`

From OBAPI 18.2 onwards `<CHANNEL_ROOT_PATH>extensions\override\obapi-locale.js`

In this file Implementer can add or override validations.

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

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

6.3 Calling custom REST service

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

For example see the snippet below:

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

7
Libraries

OBAPI 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.

- OBAPI Libraries
  This section provides information about various OBAPI libraries that are provided out of the box.

7.1 OBAPI Libraries

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

8

Digx Scheduler Application

This section describes how to create custom schedulers in OBAPI.

8.1 Create New Scheduler Class

Follow the steps given below while creating new scheduler:

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

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

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

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

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

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

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

```
try {
        ReportRequest service = new
        ReportRequest();service.executeScheduled(sessionContext);
            }
        catch (Exception e)
        {
            logger.log(Level.SEVERE, "Error occurred while executing
ReportSchedulerImpl
```

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

8.2 Configure Scheduler Class

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

Example:

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

9
Consistent UI Download

9.1 Add configurations in the Metadata Tables

The report generation system relies on the following metadata tables

1. `DIGX_CM_TABLE_METADATA`
   Stores information about each table.

Table 9-1 Field Description

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

Table 9-1 (Cont.) Field Description

Property	Description
SERIAL_NUMBER_REQUIRED	Flag to enable serial numbers on the user interface. Supported values are 'Y' to enable and 'N' to disable.
MAX_COLUMNS	Property to limit the number of columns a PDF can show. Default is 6 which can be changed by updating this property.

Example

```
Insert into DIGX_CM_TABLE_METADATA

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

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

2. DIGX_CM_COLUMN_METADATA

Stores information about columns available for a given table.

Table 9-2 Field Description

Property	Description
TABLE_METADATA_ID	Unique identifier for each table. Many to one relationship to DIGX_CM_TABLE_METADATA table and TABLE_CODE column.
NAME	The name of the column with NLS support. Maintain the file with the name "<TABLE_CODE>.properties" at the location "config/resources/nls/tablemetadata" along with the corresponding keys and values. Avoid creating duplicate files, as this file already contains the TABLE_HEADER for the DIGX_CM_TABLE_METADATA table.
COMPONENT_ID	Custom component created for user interface. Used to add custom formatting for specific columns. Default value is 'null'.
DATATYPE	The supported datatypes are String, Number, Date, Currency and Complex. Similar to COMPONENT_ID, which is purely use for UI rendering; Datatypes is for report generation.
PATH	For value fetching, use the data path. The root path of a record is represented by the dot operator ('.'). Use the root path if the entire data object is required. Alternatively, use specific JSON paths when only specific values are required, example "Person.name", here we read name from the Person object.

Table 9-2 (Cont.) Field Description

Property	Description
FIXED	To view column on some condition, Supported values are 'Y' to enable and 'N' to disable.
SORTABLE	Flag to enable serial numbers on the user interface. Supported values are 'Y' to enable and 'N' to disable.
DOWNLOADABLE	The column support for download. Supported values are 'Y' to enable and 'N' to disable.
MIN_WIDTH	The minimum width of the column.
MAX_WIDTH	The maximum width of the column.
SEQUENCE_NO	The position of the column in the table.
LENGTH	The width of the column. The sum of all column lengths for a table code should be 100 to avoid overflow and underflow of table content. If not mentioned framework will auto size the widths.

Example

```
Insert into DIGX_CM_TABLE_METADATA

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

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

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

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

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

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

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

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

9.2 Custom Datatypes for Report Download

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

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

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

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

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

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

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

```
name="CustomDateType"
      select="$data"/>                </xsl:when>                <!--
default handling -->                <xsl:otherwise>
<fo:block>                        <xsl:value-of select="$data"
      />                    </fo:block>                </
xsl:otherwise>            </xsl:choose>
      </xsl:template>
      </xsl:stylesheet>
```

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

9.3 Implement IPaginable and add XmlRootElement annotation on Response Object

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

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

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

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

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

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

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

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

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

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

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



}

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

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

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

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

9.4 Adding content before and after table in PDF Reports

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

   Use the below starter template,

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

   <xsl:stylesheet

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

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

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

   <xsl:template match="/">

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

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

   </xsl:call-template>

   </xsl:template>

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

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

   </xsl:stylesheet>

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

   <xsl:stylesheet

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

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

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

   <xsl:template match="/">

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

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

   </xsl:call-template>

   </xsl:template>

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

   <xsl:param name="data" />

   <fo:block>

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

   </fo:block>

   </xsl:template>

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

```
<xsl:param name="data" />
<fo:block>
<xsl:value-of select="$data/status/apiType" />
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

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

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

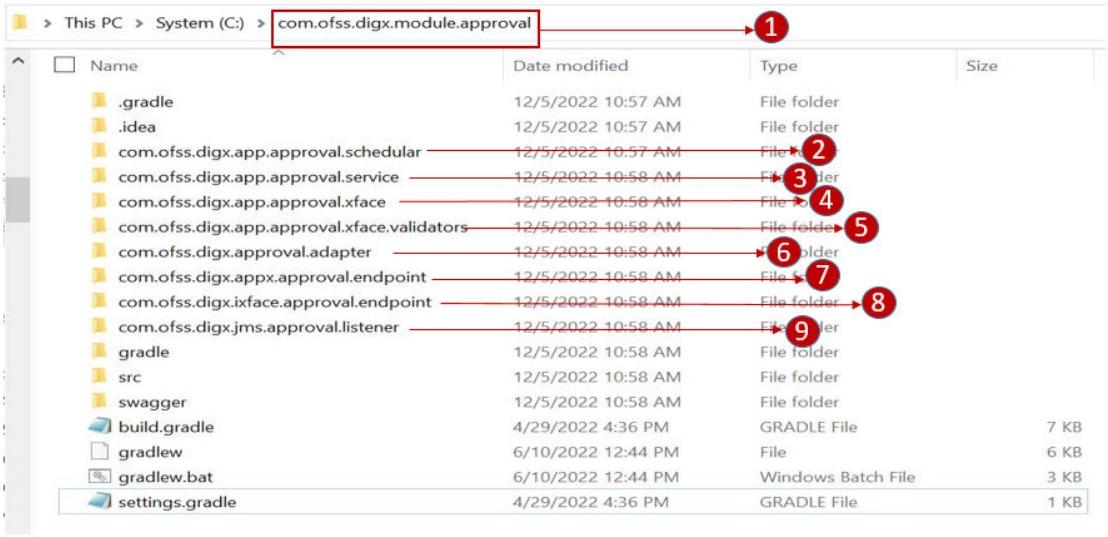
10

Package and Deploy Customisations

10.1 Base product packaging

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

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



1. Main-module-project.
2. Sub-project containing all the schedulers required by the module.
3. Sub-project containing all the services comprising the module. (Majority extensions fall under this subproject).
4. Sub-project containing all the Data Transfer Objects used in other sub-projects
5. Sub-project containing validators used to validate Data Transfer Objects facilitating input to OBAPI.
6. Sub-project containing classes used to make cross module calls.
7. Sub-project exposing endpoints to UI for end user to interact with OBAPI.
8. Subproject exposing endpoints used by other module's services to consume the services of this module via REST.
9. Sub-project containing JMS listeners required for the functioning of this module.

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

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

10.2 Customisation packaging

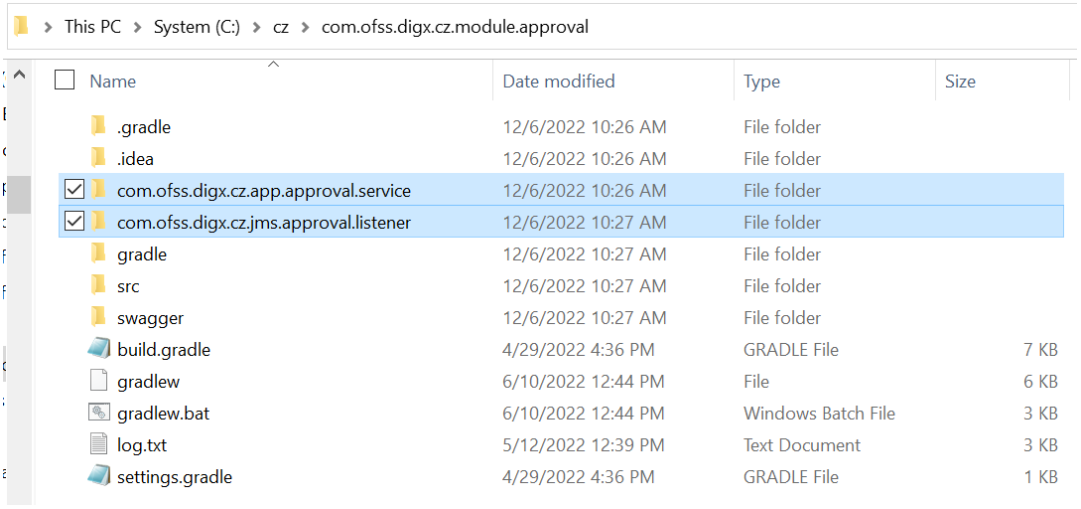
Customizations or extensions can be broadly classified into 2 as mentioned below

- Customizations in existing service layer without the need to expose a new customized REST endpoint
- Customizations to add new war

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

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

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



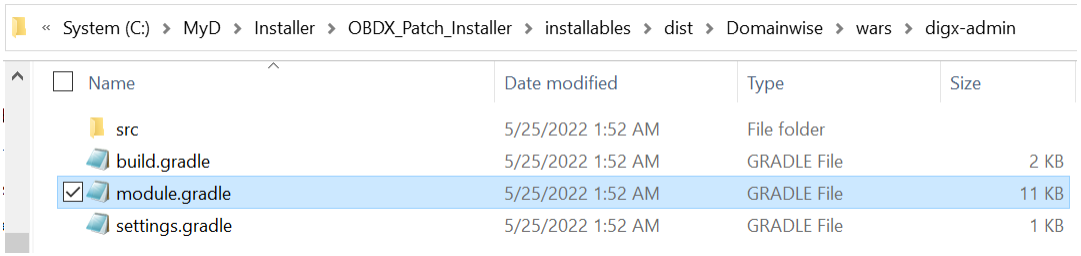
2. **Adding customised jars as dependencies in build scripts:-**
These custom jars can then be added to the war of the domain using the gradle scripts provided in the installer as demonstrated below:

The patch set installer has the following folder structure

`OBDX_Patch_Installer\installables\dist\Domainwise\wars`

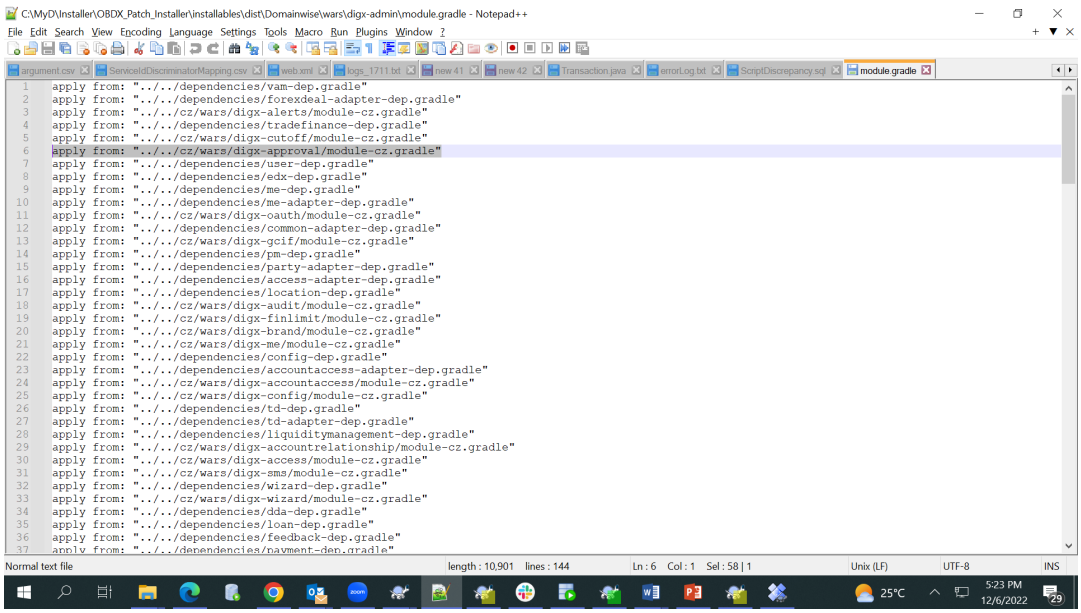
Taking ahead the current customization example we will refer module approval packaged within domain digx-admin. Please refer the below mentioned file for module approval. (As module approval is packaged in the domain named digx-admin)

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



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

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



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

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

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

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

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

   `OBDX_Patch_Installer\installables\gradle-repo`

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

   com.ofss.digx.cz.module.approval

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

   Now coming to**artefact**

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

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

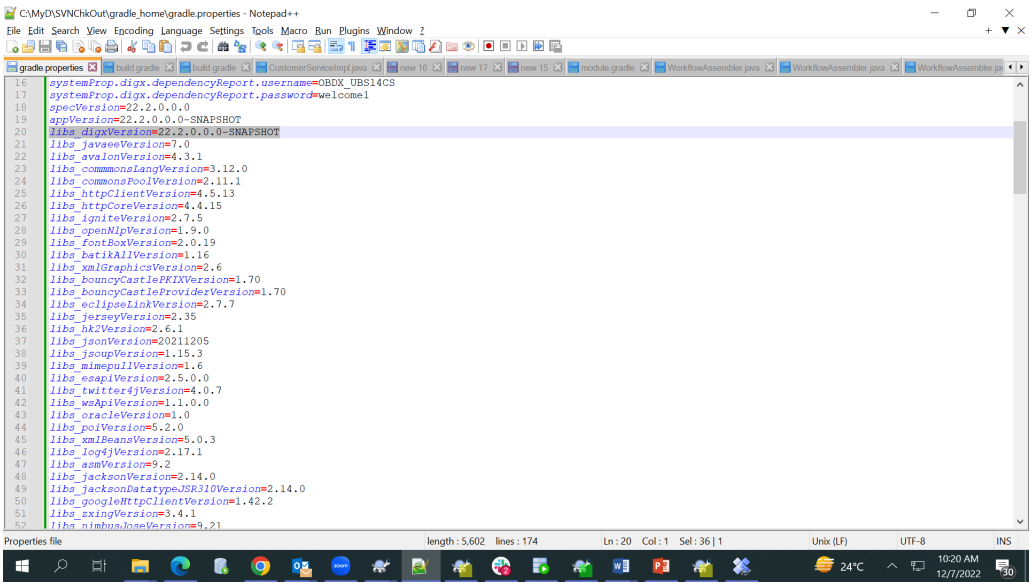
   Finally the **version** is

$libs_digxVersion

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

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

If the value of the variable is as shown below



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

Consequently the final folder structure should be as below

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

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

10.2.2 Customizations to add new war

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

11

List of Topics

This user manual is organized as follows:

Table 11-1 List of Topics

Topics	Description
Preface	This topic provides information on the introduction, intended audience, list of topics, and acronyms covered in this guide.
Objective and Scope	This topic provides information about extensibility objective and scope of it.
Architecture of Service Tier	This topic explains OBDX framework (also known as DIGX framework).
Extensible Points in Service Tier	This topic provides the various extensible points / hooks provided by OBDX framework
Extensible Points in Approval	This topic provides the extensible points in Approval framework.
Architecture of GUI Tier	This topic provides the structure of the UI artifacts and some of the important artifacts
Extensible Points in GUI Tier	This topic provides the guidelines for UI Extensibility.
Libraries	This topic provides how the OBDX has bundled its platform features and capabilities in various librarie s based on logical separation of features.
Digx Scheduler Application	This topic provides how to create custom schedulers in OBDX.
Consistent UI Download	This topic provides how to Implement IPaginable and add XmlRootElement annotation on Response Object, Adding configurations in the Metadata Tables, Custom Datatypes for Report Download.
Package and Deploy Customisations	This topic provides details of base prodccut packaging and customsing packaging.

Index