

Oracle® Banking Platform

Host Extensibility Guide

Release 2.7.1.1.0

F33669-01

September 2020

Oracle Banking Platform Host Extensibility Guide, Release 2.7.1.1.0

F33669-01

Copyright © 2011, 2020 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	27
Audience	27
Documentation Accessibility	27
Related Documents	27
Conventions	27
1 Objective and Scope	30
1.1 Overview	30
1.2 Objective and Scope	30
1.2.1 Extensibility Objective	30
1.3 Complementary Artefacts	30
1.4 Out of Scope	31
2 Overview of Use Cases	32
2.1 Extensibility Use Cases	32
2.1.1 Extending Service Execution	32
2.1.2 OBP Application Adapters	33
2.1.3 Extending Business Policy	33
2.1.4 User Defined Fields	34
2.1.5 Batch Framework Extension	34
2.1.6 Uploaded File Processing	35
2.1.7 Alert Extension	36
2.1.8 Create New Reports Using Oracle Analytics Publisher (previously called Business Intelligence Publisher)	37
2.1.9 Security Customization	38
2.1.10 Loan Schedule Computation Algorithm	40
2.1.11 Facts and Business Rules	40

2.1.12 Composite Application Service	41
2.1.13 ID Generation	42
2.1.14 OCH Integration	42
2.1.15 Documaker Integration	43
3 Extending Service Executions	44
3.1 Service Extension – Extending the "app" Layer	44
3.1.1 Application Service Extension Interface	45
3.1.2 Default Application Service Extension	46
3.1.3 Application Service Extension Executor	47
3.1.4 Extension Configuration	49
3.1.5 Application Service Extension Using Groovy	50
3.2 Extended Application Service Extension – Extending the "appx" Layer	51
3.2.1 Extended Application Service Extension Interface	53
3.2.2 Default Implementation of Appx Extension	54
3.2.3 Configuration	55
3.2.4 Extended Application Service Extension Executor	56
3.2.5 Application Service "appx" Extension using Groovy	58
3.3 End-to-End Example of an Extension	60
3.4 Support for Middleware Specific Tasks and Application service	64
3.4.1 Pre and Post Middleware Specific Transaction Tasks Overview	64
3.4.2 Sample Configuration	65
3.4.3 Custom Application Service	68
4 OBP Proxy Extension	70
5 OBP Application Adapters	74
5.1 Adapter Implementation Architecture	74
5.1.1 Package Diagram	74

5.1.2 Adapter Mechanism Class Diagram	76
5.1.3 Adapter Mechanism Sequence Diagram	76
5.2 Examples of Adapter Implementation	77
5.2.1 Example 1 – EventProcessingAdapter	77
5.2.2 Example 2 – DispatchAdapter	79
5.2.3 Example 3 - Adapter Implementation Using Groovy	80
5.3 Customizing Existing Adapters	82
5.3.1 Custom Adapter Example 1 – DispatchAdapter	82
5.3.2 Custom Adapter Example 2 – PartyKYCCheckAdapter	83
6 Business Policy Extension	88
6.1 Base Implementation of Business Policy	88
6.2 Extending Business Policy	89
6.3 Configuration	90
6.4 Extensions Using Groovy	90
7 Batch Framework Extensions	92
7.1 Typical Business Day in OBP	92
7.2 Overview of Categories	93
7.2.1 Beginning of Day (BOD)	93
7.2.2 Cut-off	93
7.2.3 End of Day (EOD)	94
7.2.4 Internal EOD	94
7.2.5 Statement Generation	94
7.2.6 Customer Communication	94
7.3 Batch Framework Architecture	94
7.3.1 Static View	95
7.3.2 Dynamic View	96

7.4 Batch Framework Components	98
7.4.1 Category Components	98
7.4.2 Shell Components	99
7.4.3 Stream Components	100
7.4.4 Database Components	102
7.5 Batch Configuration	102
7.5.1 Creation of New Category	102
7.5.2 Creation of Bean Based Shell	105
7.5.3 Creation of Procedure Based Shell	110
7.5.4 Population of Other Parameters	112
7.6 Batch Execution	114
8 Uploaded File Data Processing	116
8.1 Configuration	117
8.1.1 Database Tables and Setup	118
8.1.2 File Handlers	121
8.1.3 Record Handlers for Both Header and Details	122
8.1.4 DTO and Keys Classes for Both Header and Details	123
8.1.5 XFF File Definition XML	125
8.2 Processing	128
8.2.1 API Calls in the Handlers	128
8.2.2 Processing Adapter	129
8.3 Outcome	130
8.4 Failure/Exception Handling	131
9 Alerts Extension	132
9.1 Transaction as an Activity	132
9.1.1 Activity Record	132

9.1.2 Attaching Events to Activity	133
9.1.3 Event Record	133
9.1.4 Activity Event Mapping Record	134
9.1.5 Activity Log DTO	135
9.1.6 Alert Metadata Generation	135
9.1.7 Alert Message Template Maintenance	138
9.1.8 Alert Maintenance	139
9.2 Alert Subscription	140
9.2.1 Transaction API Changes	141
9.3 Alert Processing Steps	143
9.4 Alert Dispatch Mechanism	146
9.5 Adding New Alerts	149
9.5.1 New Alert Example	150
9.5.2 Testing New Alert	151
9.6 Support For Derived Facts	152
10 Creating New Reports Using Oracle Analytics Publisher	158
10.1 Data Objects for the Report	158
10.2 Catalog Folder	161
10.3 Data Source	162
10.4 Data Model	162
10.5 XML View of Report	166
10.6 Layout of the Report	167
10.7 View Report in Oracle Analytics Publisher	168
10.8 OBP Batch Report Configuration - Define the Batch Reports	169
10.9 OBP Batch Report Configuration - Define the Batch Report Shell	169
10.10 OBP Batch Report Configuration - Define the Batch Report Shell Depend- encies	170

10.11 OBP Batch Report Configuration	170
10.11.1 Batch Report Generation for a Branch Group Code	170
10.11.2 Batch Report Generation Status	171
10.11.3 Batch Report Generation Path	171
10.12 OBP Adhoc Report Configuration	172
10.12.1 Define the Adhoc Reports	172
10.12.2 Define the Adhoc Report Parameters	173
10.12.3 Define the Adhoc Reports to be listed in Screen	173
10.12.4 Adding Screen Tab for Report Module	173
10.13 Adhoc Report Generation – Screen 7775	174
10.14 Adhoc Report Viewing – Screen 7779	175
11 Security Customizations	178
11.1 OPSS Access Policies / Matrix Auth – Adding Attributes	180
11.1.1 Steps	180
11.1.1.1 Example of Matrix_auth conditional rule	183
11.2 OAAM Fraud Assertions – Adding Attributes	186
11.2.1 Steps	186
11.3 Security Validators	188
11.3.1 Customer Validators	189
11.3.2 Account Validators	189
11.3.3 Business Unit Validators	189
11.4 Customizing User Search	190
11.4.1 Steps	190
11.5 Customizing One-Time-Password (OTP) Processing Logic	190
11.5.1 Steps	190
11.6 Customizing Role Evaluation	190

11.6.1 Steps	191
11.7 Customizing Limits Exclusions	191
11.7.1 Steps	191
11.8 Customizing Business Rules	191
11.8.1 Steps to Update the Business Rules by Browser	191
11.8.2 Steps to Update the Business Rules in JDeveloper	201
12 Loan Schedule Computation Algorithm	206
12.1 Adding a New Algorithm	206
12.2 Consuming Third Party Schedules	209
13 Facts and Rules Configuration	210
13.1 Facts	210
13.1.1 Type of Facts	210
13.1.2 Facts Vocabulary	211
13.1.3 Generation of Facts using Eclipse Plug-in	212
13.1.4 Object Facts	232
13.2 Business Rules	235
13.2.1 Rules Engine	235
13.2.2 Rules Creation by Guided Rule Editor	235
13.2.3 Rules Creation By Decision Table	236
13.2.4 Rules Storage	237
13.2.5 Rules Deployment	238
13.2.6 Rules Versioning	238
13.3 Rules Configuration in Modules	238
13.3.1 Generic Rules Configuration	239
13.4 Rules Migration	242
13.4.1 Rules Configured for Modules	242

14 Composite Application Service	246
14.1 Composite Application Service Architecture	246
14.2 Multiple APIs in Single Module	247
15 ID Generation	254
15.1 Database Setup	255
15.1.1 Database Configuration	256
15.2 Automated ID Generation	256
15.3 Custom ID Generation	259
16 Extensibility of Domain Objects using Flex Fields	262
16.1 Flex Field - Provisioning	262
16.1.1 How to know Maximum Flex Fields Provisioned for Entity?	262
16.1.2 Increase Maximum Flex Fields Provisioned for Entity (Optional Step)	263
16.2 Flex Field - Utilization	263
16.2.1 Maintain Flex Field Metadata using Seed Data Configuration (Fast Path: OPA006) Page	263
16.3 Runtime Storage and Retrieval of Flex Field Attribute values	267
16.4 Flex Field - Fact support	268
16.5 Flex Field – Validation Support	270
16.6 Flex Field – Usage Instructions	279
17 Extensibility of Domain Objects - Dictionary Pattern	282
17.1 Customized Domain Object Attribute Placeholders	283
17.2 Customized Domain Object DTO Interceptor in UI Layer	284
17.2.1 Interceptor Hook to Persist Customized Domain Object Attributes	284
17.2.2 Interceptor Hook to Fetch Customized Domain Object Attributes	285
17.3 Dictionary Data Transfer from UI to Host	286
17.3.1 Customized Domain Object DTO Transfer from UI to Host	286
17.3.2 Customized Domain Object DTO transfer from Host to UI	290

17.4 Translating Dictionary Data into Custom Domain Object	294
17.4.1 Instantiation and Persistence of Custom Domain Objects	294
17.4.2 Fetching of Customized Domain Objects	295
17.4.3 Defining of Customized Domain Objects	296
17.5 Customized Domain Object ORM Configuration	297
17.5.1 Case 1 - Non-Inheritance based mapping	297
17.5.2 Case 2 - Mapped as ORM Subclass	300
17.5.3 Case 3 - Mapped as ORM Union-Subclass or Joined-Subclass	301
17.5.4 Case 4 - Mapped as ORM Component	304
17.6 Extensibility using Dictionary in Origination Application	304
17.6.1 ICustomDataHandler's as DictionaryArray Interceptor	304
17.6.2 Create Customized Abstract Domain Object Class	305
17.6.3 Create Customized Abstract Domain Object ORM Mapping File	306
17.6.4 Create Customized Abstract Domain Object Attribute Columns	306
17.7 Extensibility using Attributes of Various Supported Datatypes	307
17.8 Customized Domain Object having Collection of Objects as Attributes	312
17.9 Limitation to Extensibility using Dictionary Pattern	315
18 OCH Integration	318
18.1 Integration Adapter Interface	318
18.2 Abstract Integration Adapter Class	319
18.3 Sample Integration Adapter	320
18.4 Integration Abstract Assembler	321
18.5 Sample Assembler	322
19 Documaker Integration	324
19.1 XSD	324
19.2 JAXB Classes	325

19.3 Extractors	328
19.4 Seed Entries	329
19.4.1 JAXB Package Entry	329
19.4.2 Extractor Entry	330
20 Algorithm Extensions	332
20.1 Overview	332
20.2 Algorithm Spots	332
20.3 Algorithm Components	334
20.4 List of Algorithm Spots	343

List of Figures

Figure 2–1 Extending Service Execution	32
Figure 2–2 OBP Application Adapters	33
Figure 2–3 Extending Business Policy	34
Figure 2–4 Batch Framework Extension	35
Figure 2–5 Upload File Processing	36
Figure 2–6 Alerts Extension	37
Figure 2–7 Creating New Reports	38
Figure 2–8 Security Customization	39
Figure 2–9 Loan Schedule Computation Algorithm	40
Figure 2–10 Facts and Business Rules	41
Figure 2–11 Composite Application Service	41
Figure 2–12 ID Generation	42
Figure 2–13 OCH Integration	43
Figure 3–1 Standard Set of Framework Method Calls	45
Figure 3–2 Extension Hook for Document Type Application Service	46
Figure 3–3 Default Application Service Extension	47
Figure 3–4 Application Service Extension Executor	48
Figure 3–5 Extension Factory Hook for Document Type Application Service	48
Figure 3–6 Factory Implementation of Extension Hook for Document Type Application Service	49
Figure 3–7 Application Service Extension Using Groovy	50
Figure 3–8 PROP_ID and CATEGORY_ID	51
Figure 3–9 SUMMARY_TEXT	51
Figure 3–10 Add Groovy Library to Classpath	51
Figure 3–11 Extended Application Service Extension	52

Figure 3–12 Extended Application Service Extension - Post and Pre Hook	53
Figure 3–13 Extension Hook for Document Type Application Service Spi Ext	54
Figure 3–14 Default Implementation of Appx Extension	55
Figure 3–15 Extended Application Service Extension Executor	56
Figure 3–16 Extension Factory Hook for Document Type Application Service Spi Ext	57
Figure 3–17 Factory Implementation of Extension Hook for Document Type Applic- ation Service Spi Ext	58
Figure 3–18 Application Service Appx Extension using Groovy	59
Figure 3–19 PROP_ID and CATEGORY_ID	59
Figure 3–20 SUMMARY_TEXT	59
Figure 3–21 Add Groovy Library to Classpath	59
Figure 3–22 Maintenance of Document Types	60
Figure 3–23 Document Type Application Service Spi Ext - Appx Layer	61
Figure 3–24 Doc Type Application Service Spi Ext - Appx Layer	62
Figure 3–25 Document Type Application Service Spi Ext - App Layer	63
Figure 3–26 Doc Type Application Service Spi Ext - App Layer	64
Figure 3–27 Pre and Post Middleware Specific Transaction Tasks Overview	65
Figure 3–28 FLX_FW_MW_TASKS	66
Figure 3–29 FLX_FW_MW_TASKS_DTO_DEFN	66
Figure 3–30 FLX_FW_MW_TASKS_DTO_MAP	67
Figure 3–31 FLX_MD_SERVICE_ATTR	67
Figure 3–32 FLX_MD_GEN_ATTR_LEGACY_B	68
Figure 3–33 Custom Application Service	68
Figure 5–1 Package Diagram	75
Figure 5–2 Adapter Mechanism Class Diagram	76
Figure 5–3 Adapter Mechanism Sequence Diagram	77

Figure 5–4 Adapter Implementation Using Groovy	80
Figure 5–5 Credit Card Adapter Implementation Using Groovy	81
Figure 5–6 Modify AdapterFactories.properties for GroovyCreditCardAdapterFactory	81
Figure 5–7 Modify Preferences.xml for GroovyCreditCardAdapterFactory	81
Figure 5–8 Add Groovy Library to Classpath	82
Figure 5–9 Party KYC Status Check Adapter Interface	84
Figure 5–10 Default Implementation of I Party KYC Check Adapter Interface	84
Figure 5–11 KYC Adapter Factory with Mocking Support	85
Figure 6–1 Business Policy Extension	88
Figure 6–2 validate() method in AbstractBusinessPolicy.java	89
Figure 6–3 validatePolicy() in creditCardBusinessPolicy.java	89
Figure 6–4 Add a preference for custom business policy in preferences.xml	90
Figure 6–5 Extensions using Groovy	91
Figure 7–1 Business Day in OBP	93
Figure 7–2 Batch Framework Architecture - Static View	96
Figure 7–3 Dynamic View Sequence Diagram	97
Figure 7–4 State Diagram of a Shell	98
Figure 7–5 Creation of New Category	105
Figure 7–6 Population of Other Parameters	112
Figure 7–7 Population of Other Parameters - General Tab	112
Figure 7–8 Population of Other Parameters - Connection Pool	113
Figure 7–9 Population of Other Parameters - Set IS_DB_RAC	113
Figure 7–10 Population of Other Parameters - Specify Data	114
Figure 7–11 Batch Execution	114
Figure 8–1 Uploaded Data File Processing Framework	117
Figure 8–2 File Handlers	122

Figure 8–3 Record Handlers for Both Header and Details	123
Figure 8–4 DTO and Keys Classes for Both Header and Details - HeaderRecDTOKey	124
Figure 8–5 DTO and Keys Classes for Both Header and Details - AbstractDTORec	125
Figure 8–6 XXF File Definition XML	127
Figure 8–7 API Calls in Adapters	129
Figure 8–8 Processing Adapter	130
Figure 9–1 Sample script for Activity Record	133
Figure 9–2 Sample script for Event Record	134
Figure 9–3 Activity Event Mapping Record	134
Figure 9–4 Activity Log DTO	135
Figure 9–5 Metadata Generation	136
Figure 9–6 Service Data Attribute Generation	137
Figure 9–7 Alert Message Template Maintenance	139
Figure 9–8 Alert Maintenance	140
Figure 9–9 Alert Subscription	141
Figure 9–10 Transaction API Changes - Service Call	141
Figure 9–11 Transaction API Changes - Conditional Evaluation	142
Figure 9–12 Transaction API Changes - persistActivityLog(..)	142
Figure 9–13 Transaction API Changes - Activity Log	142
Figure 9–14 Transaction API Changes - Register Activity	143
Figure 9–15 Alert Processing Steps	144
Figure 9–16 Event Processing Status Type	145
Figure 9–17 Batch Alerts	146
Figure 9–18 Alert Dispatch Mechanism	147
Figure 9–19 Alert Dispatch Mechanism - Dispatcher Factory	148

Figure 9–20 Alert Dispatch Mechanism - Destination	149
Figure 9–21 Alert.Party.FirstName	153
Figure 9–22 Facts in Alerts Framework	153
Figure 9–23 Alert.Party.PartyId	153
Figure 9–24 Alert.Party.Prefix and Alert.Party.LastName	154
Figure 9–25 Message Template (Fast Path: AL03)	154
Figure 9–26 Placeholder for Derived Facts	155
Figure 9–27 Alert Maintenance (Fast Path: AL04)	155
Figure 9–28 Alert Maintenance - Map the New Message Template Placeholders ..	156
Figure 9–29 Alert Maintenance - Facts List	156
Figure 9–30 Alert Maintenance - Mapping Completed	157
Figure 9–31 Alert Mail on Mobile Number Update in Contact Point screen	157
Figure 10–1 Creating New Reports	158
Figure 10–2 Global Temporary Table	159
Figure 10–3 Report Record Type	159
Figure 10–4 Report Table Type	160
Figure 10–5 Report DML Function	160
Figure 10–6 Report DDL Function	161
Figure 10–7 Catalog Folder	162
Figure 10–8 Data Source	162
Figure 10–9 Data Model	163
Figure 10–10 Data Set	164
Figure 10–11 Group Fields	164
Figure 10–12 XML Structure and Labels	165
Figure 10–13 XML Code	165
Figure 10–14 Add Input Parameters	166

Figure 10–15 XML View of Report	166
Figure 10–16 Layout of the Report - Create Layout	167
Figure 10–17 Layout of the Report - Batch Job Results	168
Figure 10–18 View Report in Oracle Analytics Publisher	168
Figure 10–19 Batch Report Generation for a Branch Group Code	171
Figure 10–20 Batch Report Generation Path	172
Figure 10–21 Adhoc Report Generation - Report Request	174
Figure 10–22 Adhoc Report Generation - Report Generated	175
Figure 10–23 Advice Report	176
Figure 10–24 View Generated Adhoc Report	177
Figure 11–1 Security Customizations Interface	179
Figure 11–2 Security Use Case with Access Checks and Assertions	180
Figure 11–3 Add Attributes Name for Service ID	181
Figure 11–4 Add Service Attributes in SM500	182
Figure 11–5 Constraint Attribute Config with Associated Adapter	182
Figure 11–6 Select Resource Type and Add Service ID	184
Figure 11–7 Add PolicyTable Details	185
Figure 11–8 Add Condition	186
Figure 11–9 Add or Modify Fraud Rules in OAAM - Data Tab	187
Figure 11–10 Add or Modify Fraud Rules in OAAM - Conditions Tab	188
Figure 11–11 Log in to SOA Composer Application screen	192
Figure 11–12 Search Rules file	193
Figure 11–13 Search Rules file -View	194
Figure 11–14 Composite to Configure Routing Rules	195
Figure 11–15 Stages of Approval	196
Figure 11–16 Add pattern-Submission Payload Type	197

Figure 11–17 Add New Test	198
Figure 11–18 Select Fact with Appropriate Value	199
Figure 11–19 Create Participant From Group	200
Figure 11–20 Validate Rules File	201
Figure 11–21 Expand Business Rules	202
Figure 11–22 Create New Rule	203
Figure 11–23 Existing Rule	204
Figure 11–24 Change the approval group	205
Figure 12–1 Add New Algorithm	206
Figure 12–2 Create New Installment	207
Figure 13–1 Select Window Preferences	213
Figure 13–2 Window Preferences - OBP Plugin Development	214
Figure 13–3 Enter the Preferences Fact values	215
Figure 13–4 Fact Properties - aggregateCodeFilePath	216
Figure 13–5 Fact Properties - sourceFilePath	217
Figure 13–6 Start Host Server	218
Figure 13–7 Select Open Perspective value	219
Figure 13–8 Fact Explorer	220
Figure 13–9 Fact Vocabulary	221
Figure 13–10 Domain Category	222
Figure 13–11 Fact Groups	223
Figure 13–12 Facts	224
Figure 13–13 Business Definition Tab	225
Figure 13–14 Value Definition Tab	225
Figure 13–15 Enum Definition Tab	226
Figure 13–16 Aggregate Definition Tab	227

Figure 13–17 Aggregate File Tab	228
Figure 13–18 Creating New Fact - Add	229
Figure 13–19 Creating New Fact - Fact Business Definition	230
Figure 13–20 Creating New Fact - Domain Group	231
Figure 13–21 Saving New Fact	231
Figure 13–22 Saving New Fact - Fact Added	232
Figure 13–23 Designate Class as Object Fact	233
Figure 13–24 Object Fact in UI	234
Figure 13–25 Generic Rule Configuration	240
Figure 13–26 Rule Author - Decision Table	241
Figure 13–27 Rule Author - Expression Builder	242
Figure 14–1 Composite Application Service Architecture	247
Figure 15–1 Configuration of ID Generation Process	254
Figure 15–2 Automated ID Generation - Single Record View	257
Figure 15–3 Automated ID Generation - Generate Submission ID	258
Figure 15–4 Automated ID Generation - Submission ID Generation Service	258
Figure 15–5 Custom ID Generation - Custom ID Generator	259
Figure 15–6 Custom ID Generation - Custom ID Generation Constants	260
Figure 15–7 Custom ID Generation - Custom Pattern Based Generator	261
Figure 16–1 OPA006 - Error	264
Figure 16–2 OPA006 - Duplicate Attribute	264
Figure 16–3 OPA006 - Enumeration Type	265
Figure 16–4 OPA006 - Consecutive Attributes Names	265
Figure 16–5 OPA006- Invalid Input for Attribute Data Type.	266
Figure 16–6 Attribute Data Type as Date	266
Figure 16–7 Runtime Storage Saved Data	267

Figure 16–8 Runtime Storage Retrieved Data	268
Figure 16–9 Flex Field - Fact support Saved Data	268
Figure 16–10 Flex Field - Fact support Data in Table	269
Figure 16–11 Flex Field - Fact support Seed	269
Figure 16–12 Flex Field - Fact support After Seed	269
Figure 16–13 Flex Field - Validation Support - Metadata	275
Figure 16–14 Flex Field - Validation Support - Mandatory Validation - Input	275
Figure 16–15 Flex Field - Validation Support - Mandatory Validation -Output	276
Figure 16–16 Flex Field - Validation Support - Min/Max Lenght Validation Input ...	276
Figure 16–17 Flex Field - Validation Support - Min/Max Lenght Validation Output .	277
Figure 16–18 Flex Field - Validation Support - Pattern (Regex) Validation Input ...	277
Figure 16–19 Flex Field - Validation Support - Pattern (Regex) Validation Output .	278
Figure 16–20 Flex Field - Validation Support - Enum Validation Input	278
Figure 16–21 Flex Field - Validation Support - Enum Validation Output	278
Figure 16–22 Flex Field - Validation Support - Date Validation Input	279
Figure 16–23 Flex Field - Validation Support - Date Validation Output	279
Figure 17–1 Extensibility of Domain Objects - Framework	283
Figure 17–2 Code Extract	284
Figure 17–3 Interceptor Hook to Persist Customized Domain Object	285
Figure 17–4 Interceptor Hook to Fetch Customized Domain Object	286
Figure 17–5 JSONClient constructs the JSON Object	287
Figure 17–6 SerializeDictionaryArray to include GenericName and Value attrib- utes	288
Figure 17–7 Host Server JSONFacade extracts the attribute of JSON Object	289
Figure 17–8 AbstractJSONFacade's getDictionaryArray method	290
Figure 17–9 Host Server JSONFacade constructs the JSON Object	291

Figure 17–10 AbstractJSONFacade's serializeDictionaryArray to include Generic Name and Value attributes	292
Figure 17–11 UI Server JSONClient extracts the DictionaryArray attribute	293
Figure 17–12 AbstractJSONBindingStub's getDictionaryArray method	294
Figure 17–13 Instantiation of DataTransferObjects	296
Figure 17–14 Adding Discriminator Column Mapping in Existing ORM file	298
Figure 17–15 ORM File Mapping to Customized Domain Object	298
Figure 17–16 Adding New Java File to the Customized Domain Object	299
Figure 17–17 Adding Extra Columns along with the Discriminator Column	299
Figure 17–18 Adding a New ORM File Mapping to Customized Domain Object	300
Figure 17–19 Adding New Java File to Customized Domain Object	301
Figure 17–20 New ORM File Mapping	302
Figure 17–21 Adding New Java File	302
Figure 17–22 Create a New Table CZ_NAB_LM_PROPOSED_FACILITY	303
Figure 17–23 CustomDataHandler's as DictionaryArray Interceptor	305
Figure 17–24 Create Customized Abstract Domain Object Class	306
Figure 17–25 Create Customized Abstract Domain Object ORM Mapping File	306
Figure 17–26 Create Customized Abstract Domain Object Attribute Columns	306
Figure 17–27 Customized Message Template Class	308
Figure 17–28 Domain Object Table	309
Figure 17–29 ORM File	309
Figure 17–30 JUnit Test Case	310
Figure 17–31 JUnit Adds Table Record	310
Figure 17–32 Dictionary Array Values	311
Figure 17–33 Customized Domain Object having collection of Objects as Attributes	312
Figure 17–34 Member Attributes of Customized Domain Object	313

Figure 17–35 Dictionary Array Elements	313
Figure 17–36 Customized Domain Object constructed by AbstractAssembler	314
Figure 17–37 Dictionary Array returned by AbstractAssembler	315
Figure 18–1 Integration Adapter Interface	319
Figure 18–2 Abstract Integration Adapter Class	320
Figure 18–3 Sample Integration Adapter	321
Figure 18–4 Integration Abstract Assembler	322
Figure 18–5 Sample Assembler	323
Figure 18–6 AddressChangeLetter.xsd	324
Figure 18–7 CustomizedAddressChangeLetter.xsd	325
Figure 18–8 JAXB Classes	326
Figure 18–9 Generate JAXB Classes from Customized XSD	326
Figure 18–10 JAXB Classes Customized XSD	327
Figure 18–11 JAXB Classes in Project	327
Figure 18–12 AddressChangeLetterDataExtractor	328
Figure 18–13 Customized AddressChangeLetterDataExtractor	329
Figure 19–1 Algorithm Spot Interface	332
Figure 19–2 Example: Algorithm Spot Interface	333
Figure 19–3 Example: Adding New Algorithm Spot	334
Figure 19–4 Example: Data Validation Algorithm Component	335
Figure 19–5 Example: CharAdhocDateValidation	336
Figure 19–6 New Algorithm Implementation	337
Figure 19–7 Adhoc Characteristic Date Validation	337
Figure 19–8 Algorithm Spot Interface Methods	338
Figure 19–9 Algorithm Spot Interface Methods (continued)	339
Figure 19–10 Generated Artifacts	340

Figure 19–11 Generated Artifacts	341
Figure 19–12 Create Algorithm Type	342
Figure 19–13 Attach Algorithm	343

List of Tables

Table 5–1 Components of Adapter Implementation	74
Table 7–1 Database Server Components	102
Table 7–2 FLX_BATCH_JOB_CATEGORY_MASTER	103
Table 7–3 FLX_BATCH_JOB_GRP_CATEGORY	103
Table 7–4 FLX_BATCH_JOB_CATEGORY_DEPEND	104
Table 7–5 FLX_BATCH_JOB_SHELL_MASTER	106
Table 7–6 FLX_BATCH_JOB_SHELL_DTLS	107
Table 7–7 FLX_BATCH_JOB_SHELL_DEPEND	108
Table 7–8 Driver Table	108
Table 7–9 Actions Table	109
Table 8–1 FLX_EXT_FILE_UPLOAD_MAST	118
Table 8–2 Mandatory Fields in Record Tables	119
Table 8–3 FLX_EXT_FILE_PARAMS	119
Table 8–4 FLX_BATCH_JOB_SHELL_DTLS	120
Table 8–5 XXF File Definition XML	126
Table 8–6 Process Status	131
Table 9–1 FLX_EP_ACT_B	132
Table 9–2 FLX_EP_EVT_B	133
Table 9–3 FLX_EP_ACT_EVT_B	134
Table 9–4 Key Fields in FLX_MD_SERVICE_ATTR	137
Table 13–1 Example of a Decision Table	236
Table 13–2 Actions	237
Table 13–3 Conditions	237
Table 13–4 Rules Versioning	238

Table 13–5 Details of Configured Rules in Modules	242
Table 14–1 Java Classes	247
Table 15–1 FLX_CS_ID_CONFIG_B	255
Table 15–2 FLX_CS_ID_RANGE	255
Table 15–3 FLX_CS_ID_USF	256
Table 16–1 metadata configuration details for validations	270

Preface

This guide explains customization and extension of Oracle Banking Platform.

This preface contains the following topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This guide is intended for the users of Oracle Banking Platform.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/us/corporate/accessibility/index.html>.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/us/corporate/accessibility/support/index.html#info> or visit <http://www.oracle.com/us/corporate/accessibility/support/index.html#trs> if you are hearing impaired.

Related Documents

For more information, see the following documentation:

- For installation and configuration information, see the Oracle Banking Platform Localization Installation Guide - Silent Installation guide.
- For a comprehensive overview of security, see the Oracle Banking Platform Security Guide.
- For the complete list of licensed products and the third-party licenses included with the license, see the Oracle Banking Platform Licensing Guide.
- For information related to setting up a bank or a branch, and other operational and administrative functions, see the Oracle Banking Platform Administrator Guide.
- For information on the functionality and features, see the respective Oracle Banking Platform Functional Overview documents.
- For recommendations of secure usage of extensible components, see the Oracle Banking Platform Secure Development Guide.

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.
<i>italic</i>	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.

1 Objective and Scope

This chapter defines the objective and scope of this document.

1.1 Overview

Oracle Banking Platform (OBP) 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 for its core banking operations, across its retail and corporate offerings.

OBP 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 and Scope

Most product development can be accomplished through highly flexible system parameters and business rules. Further competitive differentiation can be achieved through IT configuration and extension support. In OBP, additional business logic required for certain services is not always a part of the core product functionality but could be a client requirement. For these purposes, extension points and customization support have been provided in the application code which can be implemented by the bank and / or by partners, wherein the existing business logic can be added with or overridden by customized business logic. This way the time consuming activity of custom coding to enable region specific, site specific or bank specific customizations can be minimized.

1.2.1 Extensibility Objective

The broad guiding principles with respect to providing extensibility in OBP 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. Hence, separate budgets specifically for this.
- Standards based - OBP leverages standard tools and technology
- Leverage large development pool for standards based technology.
- Developer tool sets provided as part of JDeveloper and Eclipse for productivity.

1.3 Complementary Artefacts

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

- OBP Zen training course
- OBP installation and configuration
- OBP parameterization as part of implementation
- Functional solution and product user guide

References to plugin indicate the eclipse based OBP development plugin for relevant version of OBP being extended. The plugin is not a product GA artefact and is a means to assist development. Hence, the same is not covered under product support.

1.4 Out of Scope

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

2 Overview of Use Cases

The use cases that are covered in this document shall enable the developer in applying the discipline of extensibility to OBP. While the overall support for customizations is complete in most respects, the same is not a replacement for implementing a disciplined, thoughtful and well-designed approach towards implementing extensions and customizations to the product.

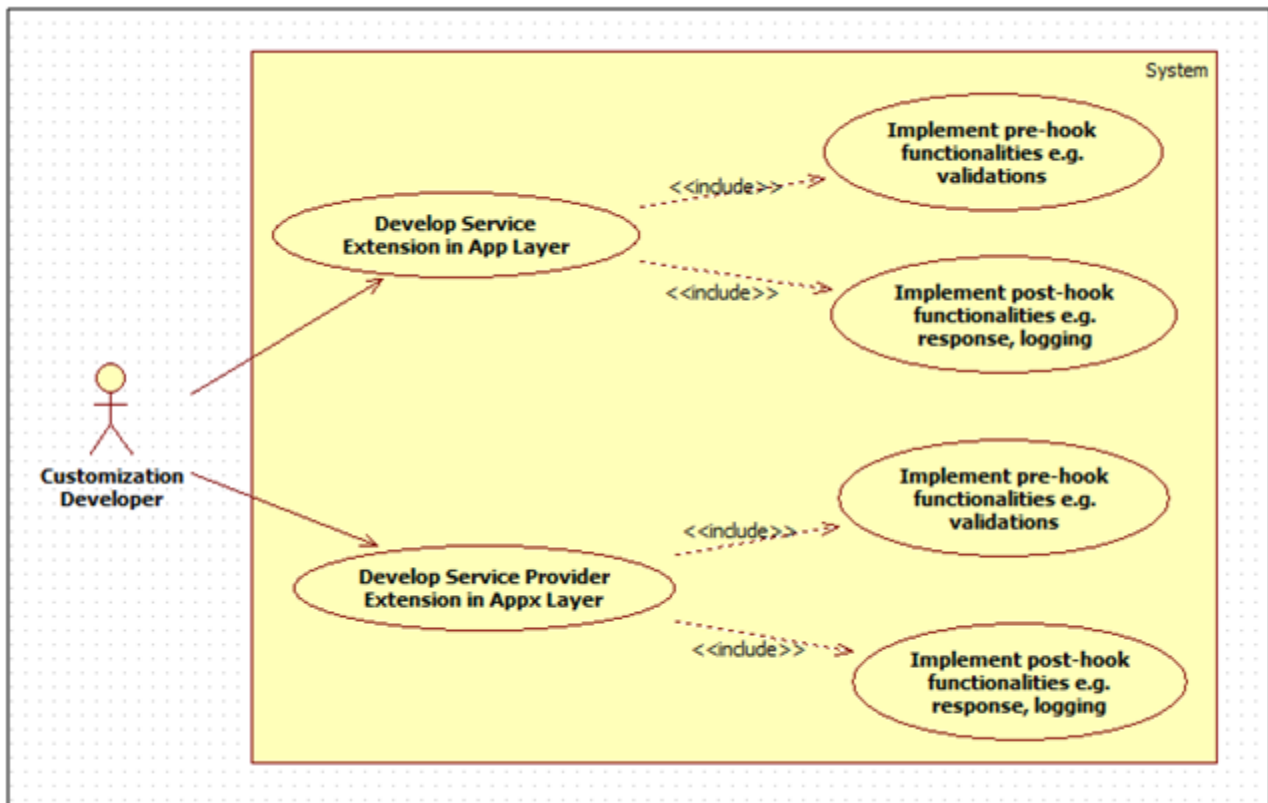
2.1 Extensibility Use Cases

This section gives an overview of the extensibility topics and customization use cases to be covered in this document. Each of these topics is detailed in the further sections.

2.1.1 Extending Service Execution

In OBP, additional business logic might be required for certain services. This additional logic is not part of the core product functionality but could be a client requirement. For these purposes, hooks have been provided in the application code wherein additional business logic can be added or overridden with custom business logic.

Figure 2–1 Extending Service Execution



Following are the two hooks provided:

Service Extensions

This hook resides in the app layer of the application service. This hook is present for, before as well after the actual service execution. The additional business logic has to implement the interface *I<service_name>ApplicationServiceExt* and extend and override the default implementation *Void<service_name>ApplicationServiceExt* provided for the service. 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.

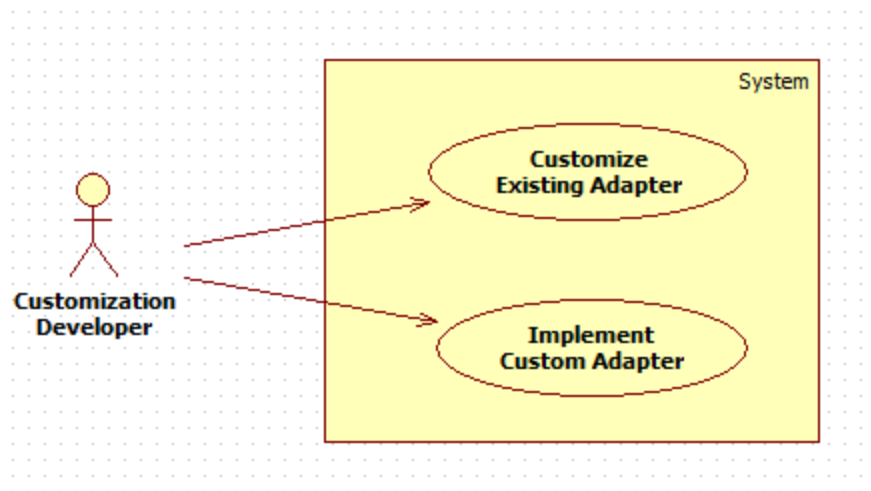
Service Provider Extension

This hook resides in the appx layer of the application service. This hook, too, is present for before as well after the actual service execution. The additional business logic has to implement the interface *I<service_name>ApplicationServiceSpiExt* and extend and override the default implementation *Void<service_name>ApplicationServiceExt* provided for the service. 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.

2.1.2 OBP Application Adapters

In OBP, adapters are used for helping two different modules or systems to communicate with each other. It helps the consuming side adapt to any incompatibility of the invoked interface to work together. This is done to achieve cleaner build time separation of different functional product processor modules. Hence, when Loan Module needs to invoke services of Party Module or Demand Deposit module then an adapter class owned by the Loans module will be used to ensure that functions such as defaulting of values, mocking of an interface, and so on, are implemented in the adapter layer thereby relieving the core module functionality from getting corrupted.

Figure 2–2 OBP Application Adapters



2.1.3 Extending Business Policy

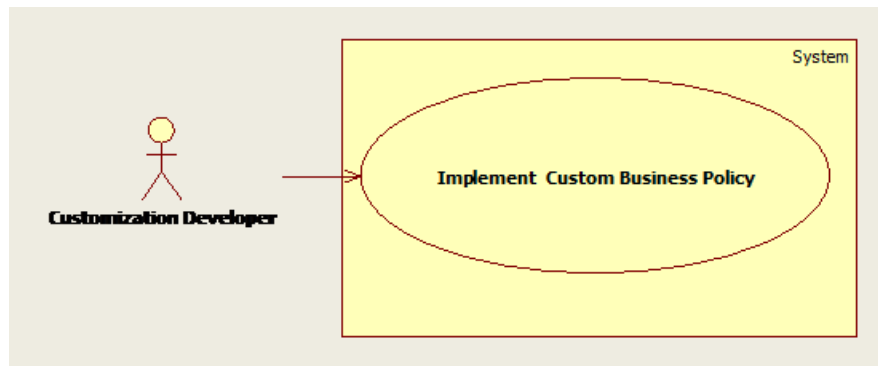
In OBP, business policies are used for common business validations. For instance, credit card number validation to check whether or not the credit card number entered by user complies with the specified format or exists in the record. Business policy implementation strategy is based on factory design pattern and implements a common business policy factory class for each module. All the business policy factory classes

extend to `AbstractBusinessPolicyFactory` Class. `AbstractBusinessPolicyFactory` Class returns the `BusinessPolicy` class instance which extends to `AbstractBusinessPolicy` class. Application service invokes the `validate()` method in `AbstractBusinessPolicy` class which in turn invokes `validatePolicy()` method in the `BusinessPolicy` class.

Custom `BusinessPolicies` are implemented in OBP by configuring preferences in the `preferences.xml` file. In this file a preference for `customBusinessPolicy` is defined which represents a query to the database. For customization, create an entry in the `Flx_or_config_all_b` table with preference name and `businessPolicy` code.

When application service invokes the `createPolicyInstance()` method of the `BusinessPolicyFactory` class, this class invokes a `getPolicyInstance()` method of the `AbstractBusinessPolicy` class which looks for any custom `businessPolicy` class in the database and returns the custom class if it gets one. Otherwise it returns null, and a new instance of base `BusinessPolicy` class is created and returned to the invoking application service.

Figure 2–3 Extending Business Policy



2.1.4 User Defined Fields

Custom Entities: Additional fields can be added to objects / entities from the very base level (ORM / POJO layer) to the front end (View layer) level. This way is more costly since it requires changes at all layers of the application. However, it has an advantage of the ability to use the additional data in the business logic of the application.

- **Client:** The UI of the screen in which the additional data needs to be captured has to be modified for the additional fields. The view-service linkage also needs to be modified for transferring the additional data.
- **Host:** On the host side, the ORM and POJO for the entity have to be modified to save the additional field's data. The service layer has to be modified for any business logic that is affected by the additional fields.

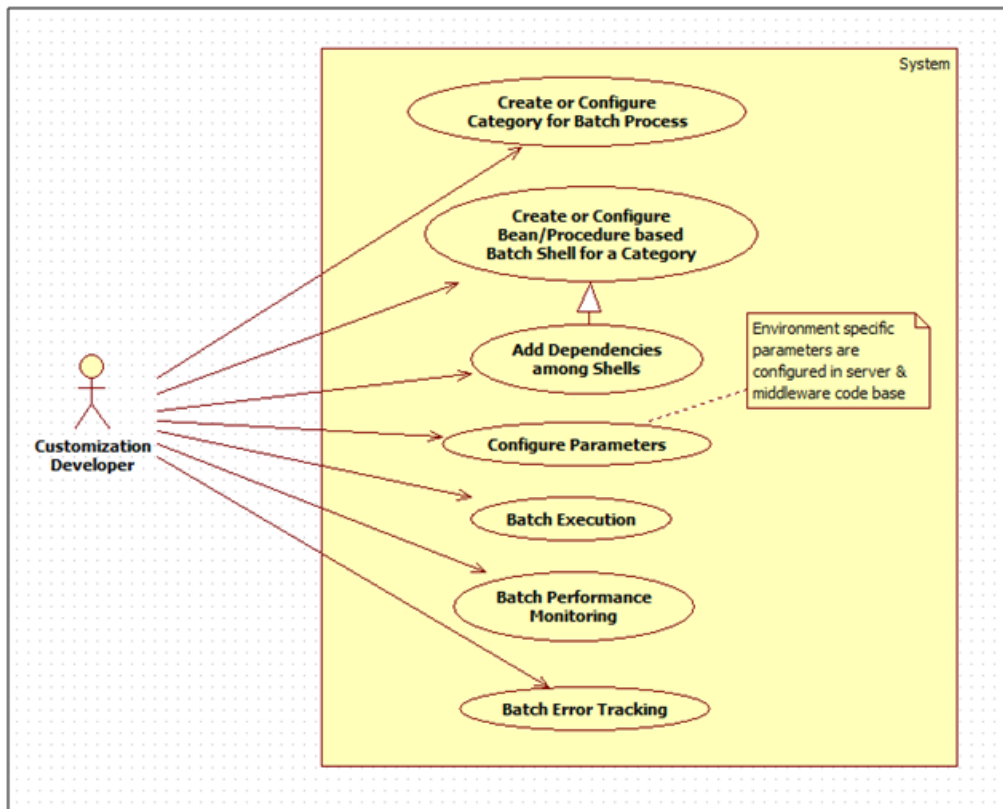
2.1.5 Batch Framework Extension

This extensibility feature is provided because most of the enterprise applications require the bulk processing of records to perform business operations in real-time environments. These business operations include complex processing of large volumes of information that is most efficiently processed with minimal or no user interaction. Such operations includes time-based events (For example, month-end calculations, notices or

correspondence), periodic application of complex business rules processed repetitively across very large data sets (For example, rate adjustments).

All such scenarios form a part of batch processing for multiple records. Thus, Batch processing is used to process billions of records for enterprise applications. There are many categories in OBP Batch Processes like Beginning of Day (BOD), End of Day (EOD), and Statement Generation, and so on, for which the batch execution is performed.

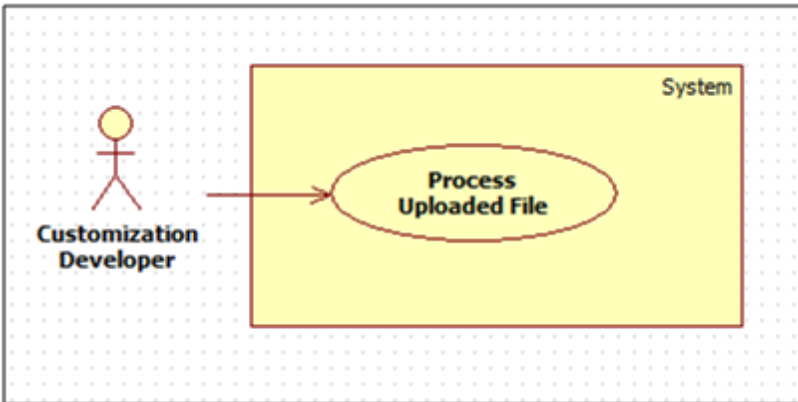
Figure 2–4 Batch Framework Extension



2.1.6 Uploaded File Processing

File processing is an independent process and is done separately after file upload. Every upload provides a unique file ID for the uploaded file. The processing is then done for each upload as per the required functionality. The final status is provided at the end of the processing in the form of ProcessStatus.

An example can be salary credit processing. Once the employer account details (in header records) and the multiple employee account details (in detail records) are uploaded through the file upload, the salary credit processing can be done in which the employer account will be debited and the multiple accounts of the employees will be credited.

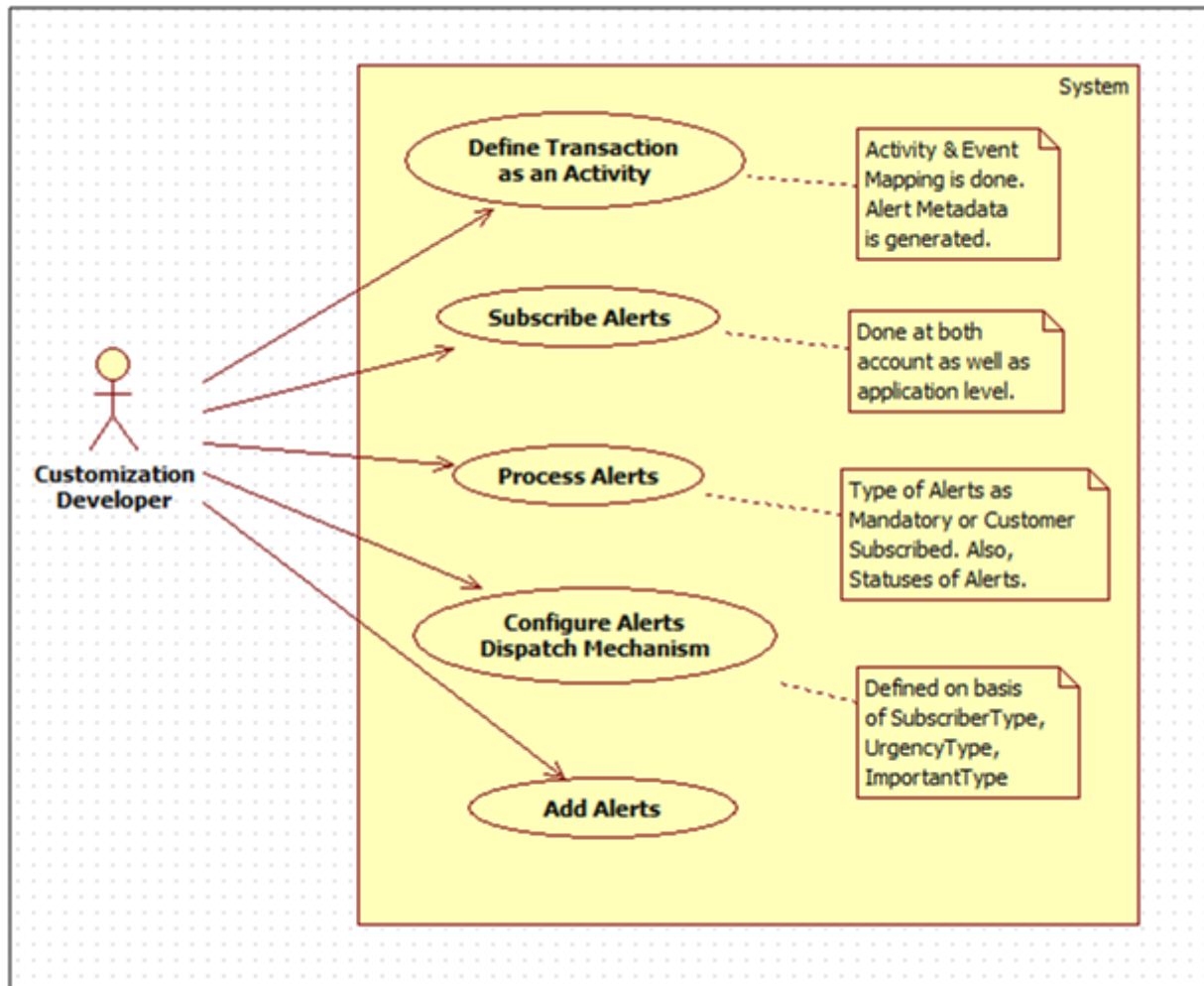
Figure 2–5 Upload File Processing

2.1.7 Alert Extension

OBP has to interface with various systems to transfer data which is generated during business activities that take place during teller operations or processing. The system requires a framework which can support on-line data transfer to interfacing systems.

This extension of event processing module of OBP provides a framework for identifying executing host services as activities and generating / raising events that are configured against the same. Generation of these events results in certain actions that can vary from dispatching data to subscribers (customers or external systems) to execution of additional logic. The action whereby data is dispatched to subscribers is termed as Alert. In OBP application, these Alerts can be customized and configured.

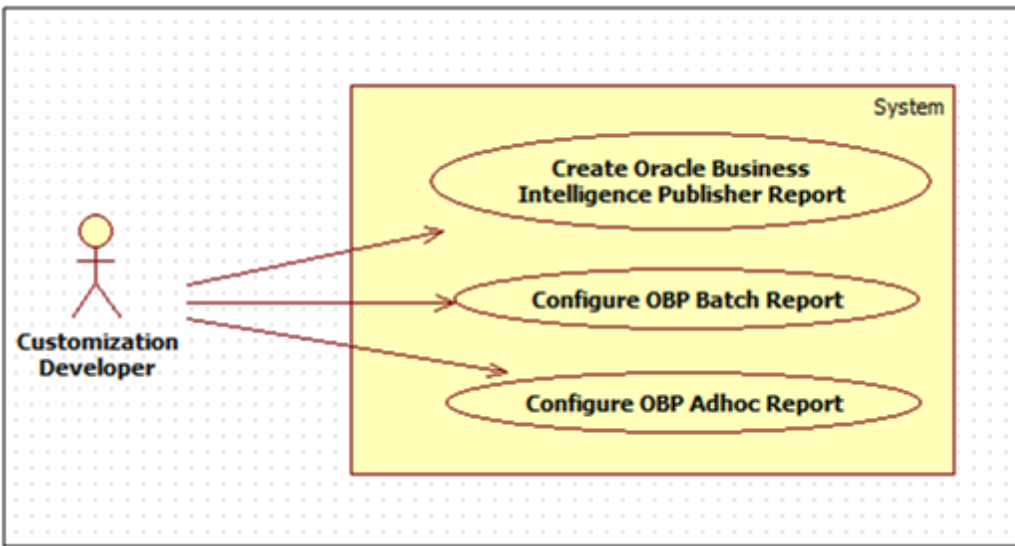
Figure 2–6 Alerts Extension



2.1.8 Create New Reports Using Oracle Analytics Publisher (previously called Business Intelligence Publisher)

OBP application provides functionality for configuring multiple reports through integrated Oracle Analytics Publisher. It is a standalone reporting and document output management solution that allows companies to lower the cost of ownership for separate reporting solutions. The developer can add and configure an Adhoc report to OBP using the Oracle Analytics Publisher.

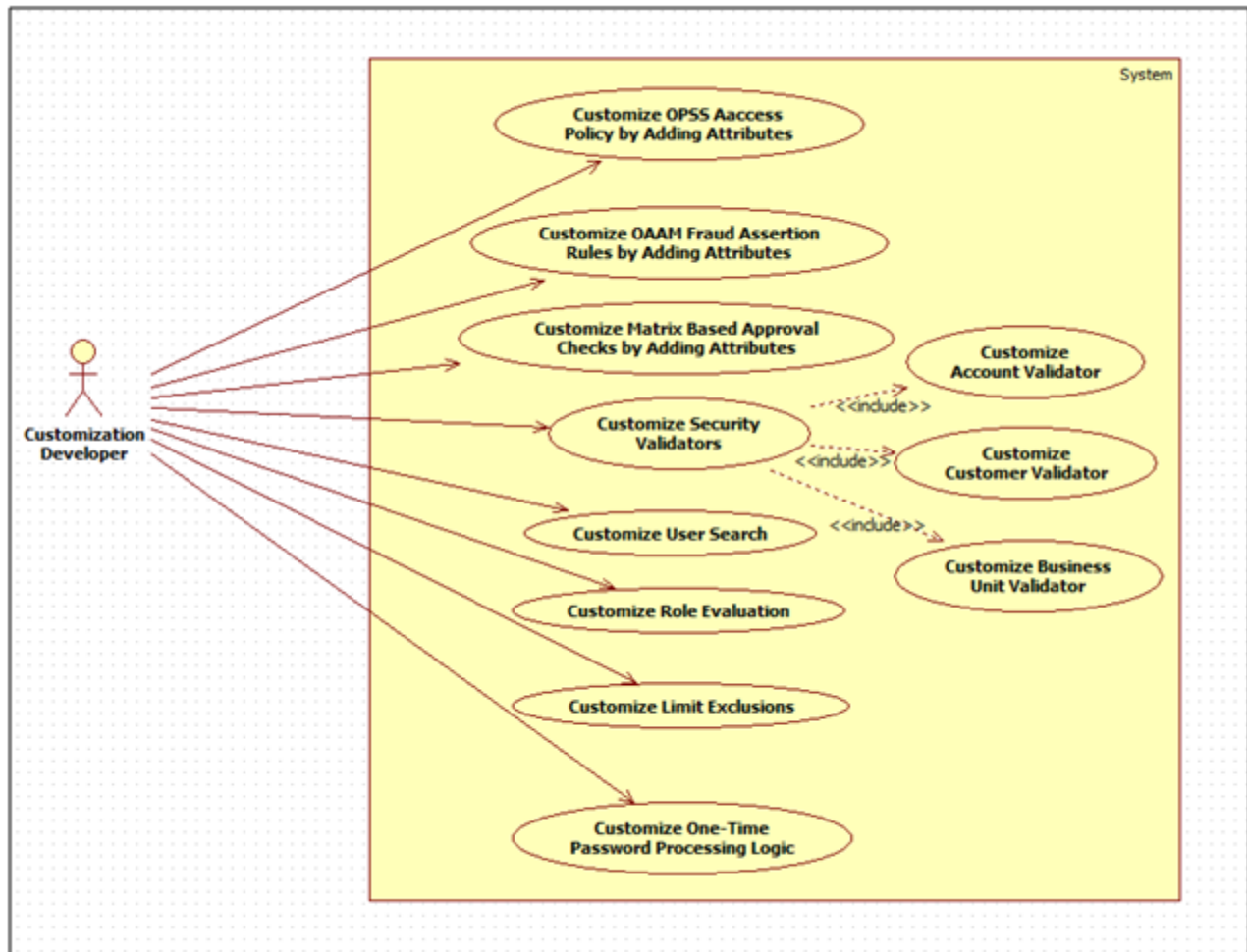
The OBP application also has a batch framework using which a developer can easily add batch processes, also known as batch shells, to the application. The batch framework executes all the batch shells defined in the system as per their configuration. The results of these batch shell executions are stored in the database. In OBP, the user can create and customize the batch reports based on the requirements which can vary from client to client.

Figure 2–7 Creating New Reports

2.1.9 Security Customization

OBP application comprises of several modules which have to interface with various systems in an enterprise to transfer/share data. This data is generated during business activity that takes place during teller operations or processing. While managing the transactions that are within OBP's domain, it also needs to consider security and identity management and the uniform way in which these services need to be consumed by all applications in the enterprise. This is possible if these capabilities can be externalized from the application itself and are implemented within products that are specialized to handle such services.

Figure 2–8 Security Customization



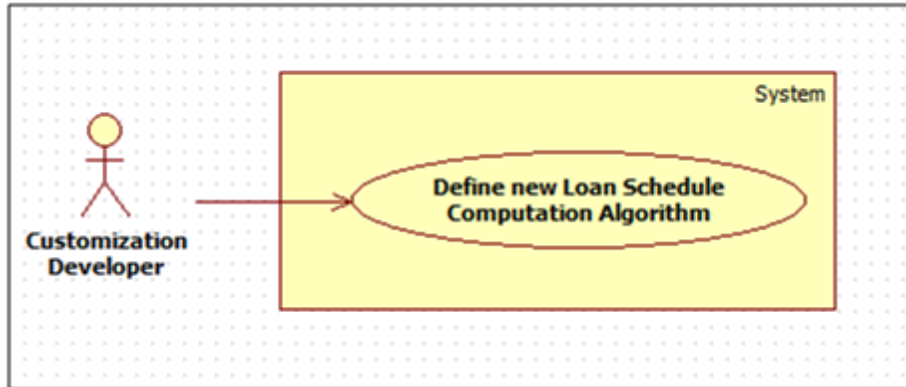
OBP application therefore provides functionality where there is a provision for customizing the security attributes or functions. For example:

- Attributes participating in access policy rules
- Attributes participating in fraud assertion rules
- Attributes participating in matrix based approval checks
- Account validator
- Customer validator
- Business unit validator
- Adding validators
- Customizing user search
- Customizing 2FA 'Send OTP | Validate OTP' logic
- Customizing Role Evaluation
- Customizing Limit Exclusions

2.1.10 Loan Schedule Computation Algorithm

OBP application provides the extensibility by which the additional loan schedule computation algorithm can be customized based on client's requirement.

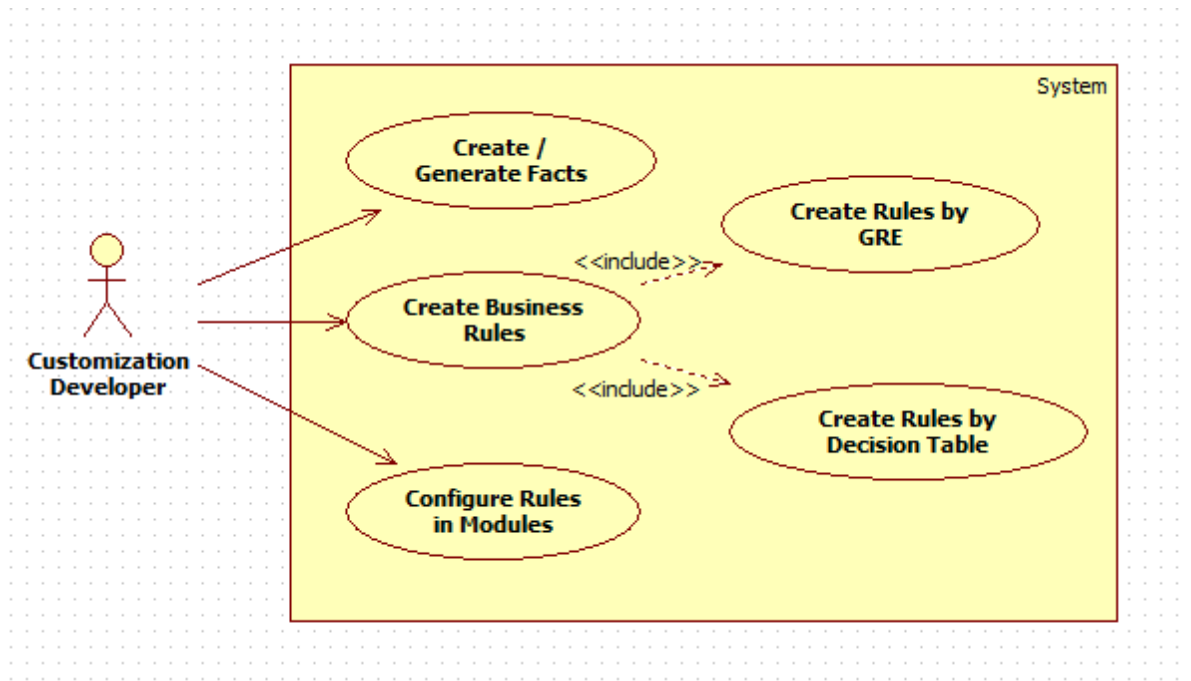
Figure 2–9 Loan Schedule Computation Algorithm



2.1.11 Facts and Business Rules

Fact (in an abstract way) is something which is a reality or which holds true at a given point of time. Business rules are made up of facts. Business Rules are defined for improving agility and for implementing business policy changes. This agility, meaning fast time to market, is realized by reducing the latency from approved business policy changes to production deployment to near zero time. In addition to agility improvements, Business Rules development also requires far fewer resources for implementing business policy changes. This means that Business Rules not only provide agility, it also provides the bonus of cost reduced development.

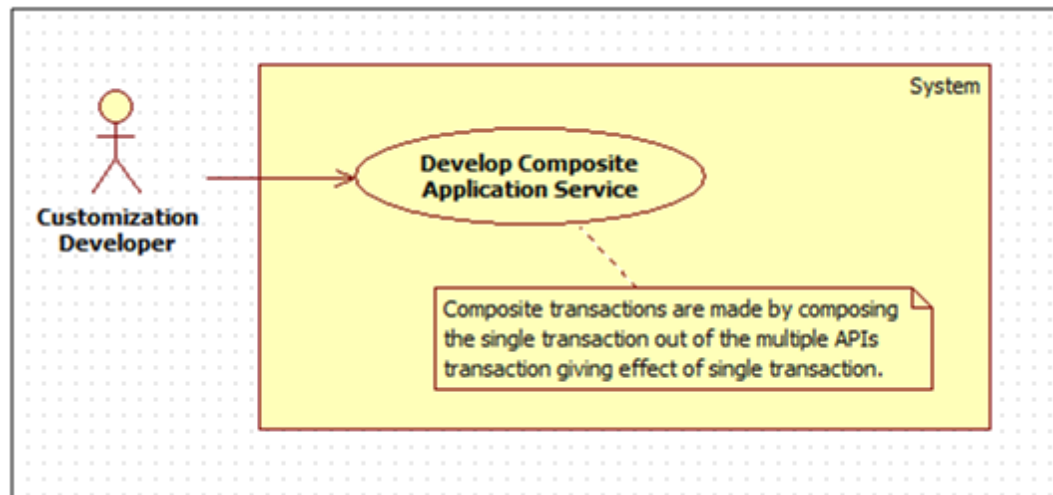
Figure 2–10 Facts and Business Rules



2.1.12 Composite Application Service

OBP provides the extensibility feature by which user can write the composite service in which multiple service calls can be made as part of single call. Transactions in composite application service are made by composing the single transaction out of the multiple APIs transaction that gives the effect of single transaction.

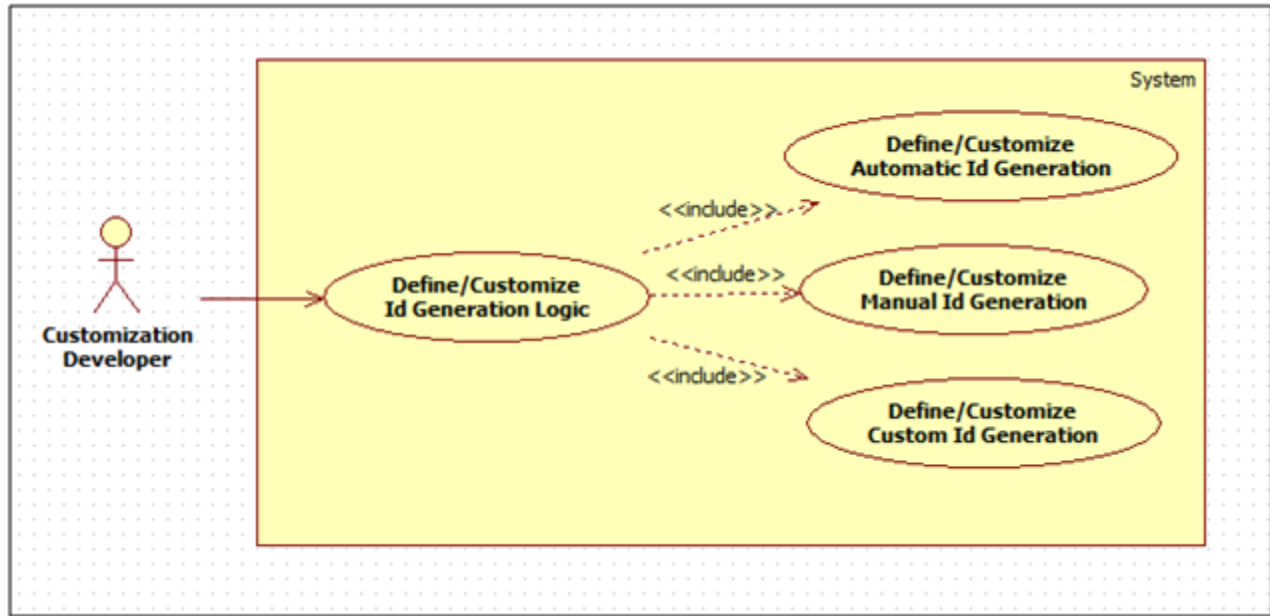
Figure 2–11 Composite Application Service



2.1.13 ID Generation

OBP is shipped with the functionality of ID generation in three ways that is, Automatic, Manual and Custom. These three configurations can be defined by the user as per their requirements and IDs can be generated accordingly.

Figure 2–12 ID Generation

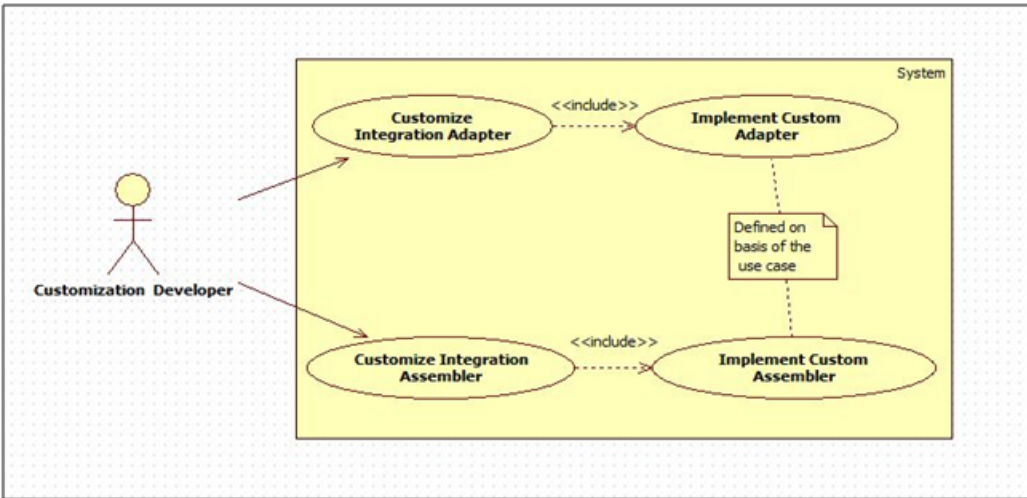


2.1.14 OCH Integration

OBP provides various integration adapters and assemblers which are used to publish customer information to OCH. These adapters and assemblers can be customized for publishing details to OCH.

Customization developer can extend the existing integration adapters to fetch or gather more information about the customer and customize integration assembler to add new mappings.

Figure 2–13 OCH Integration



2.1.15 Documaker Integration

Document generated using Documaker consists of Static and Dynamic data. OBP sends dynamic data (which needs to be populated inside a document) to Documaker Server in XML format. For this purpose, OBP provides various extractors which are used to extract data from OBP and send it to the Documaker server. These extractors can be customized according to the requirement of the data.

Customization developer can extend existing extractors and new extractor can gather additional information that needs to be populated inside a document.

3 Extending Service Executions

This chapter describes how additional business logic can be added prior to execution (pre hook) and/or post the execution (post hook) of particular application service business logic on the host side. Extension prior to a service execution can be required for the purposes of additional input validation, input manipulation, custom logging and so on. A few examples in which the application service extensions in the form of pre and post hook could be required are mentioned below.

Note: Consulting service extensions must be done only at the product SPI level / APPX Level only. It MUST not be on product APP services. Highlight missing SPIs to Product Team.

An application service extension in the form of a pre 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.

An application service extension in the form of a post hook can be important in the following scenarios:

- Output response manipulation
- Third party system calls in the form of web service invocation, JMS interface and so on.
- Custom data logging for subsequent processing or reporting.

The OBP application provides two layers where the pre and post extension hooks for extending service execution can be implemented. These places are:

- Application Service layer – referred to as the “app” layer extension.
- Extended Application Service – referred to as the “appx” layer extension.

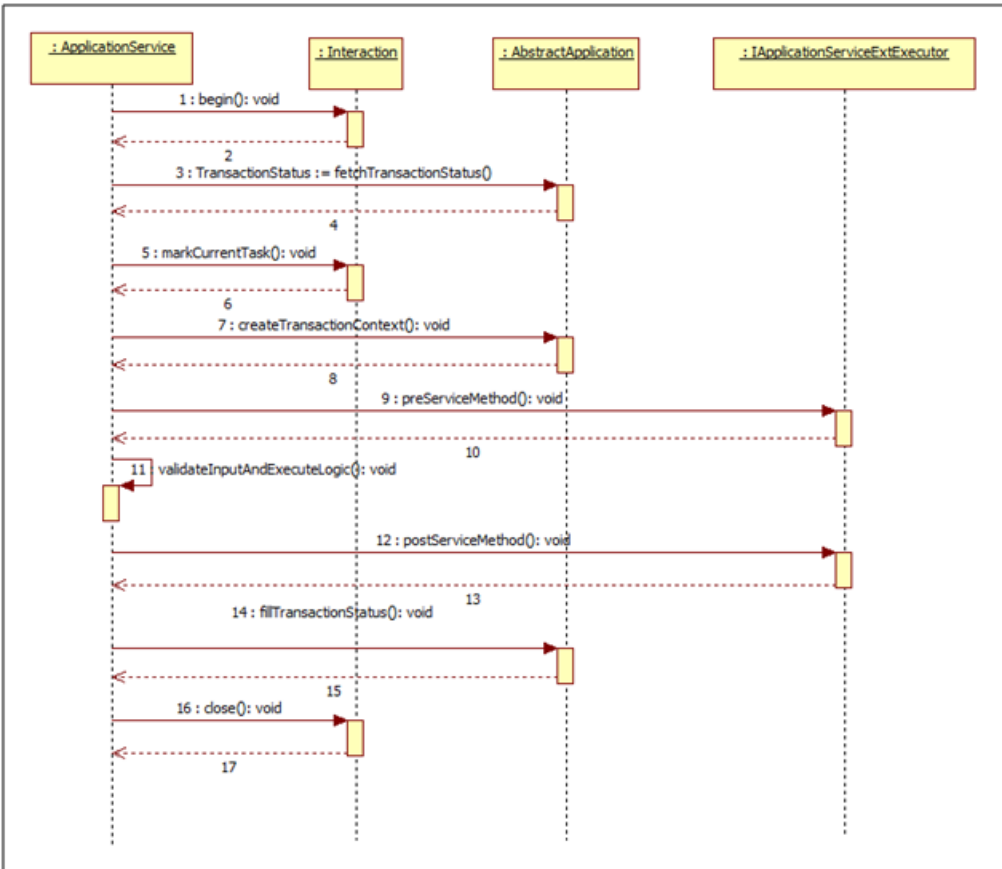
There are few differences in the extensions of the app and appx layer:

- In the appx layer extension, the validations can be added against the user defined fields which is not possible in case of the app layer.
- In the appx layer extension, the service response can be passed when the return type is not transaction status. This response can be validated or updated which is not available in case of app layer.
- In the appx layer, the approvals can be incorporated and can be validated in the appx layer extension which is not possible in app layer.

3.1 Service Extension – Extending the "app" Layer

The "app" layer is referred to as the application service layer. It denotes the business logic that executes as part of a service method exposed by OBP middleware host. Extension points provided as pre and post hooks are present in this layer at the same points in the service. Every application service method has a standard set of framework method calls as shown in the sequence diagram below:

Figure 3–1 Standard Set of Framework Method Calls



The pre hook is provided after the invocation of createTransactionContext call inside the application service. At this step, the transaction context is set and the host application core framework is aware of the executing service with respect to the authenticated subject or the user who has posted the transaction, transaction inputs, financial dates, different determinant types applicable for the executing service, an initialized status and has the ability to track the same against a unique reference number. At this step, the database session is also initialized and accessible enabling the user to use the same in the pre hook for any database access which needs to be made.

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 following sections explain important concepts, which should be understood for extending in this service layer.

3.1.1 Application Service Extension Interface

The OBP plug-in for eclipse generates an interface for the extension of a particular service. The interface name is in the form I<Service_Name>ApplicationServiceExt. This interface has a pair of pre and post method definitions for each application service method of the present. The signatures of these methods are:

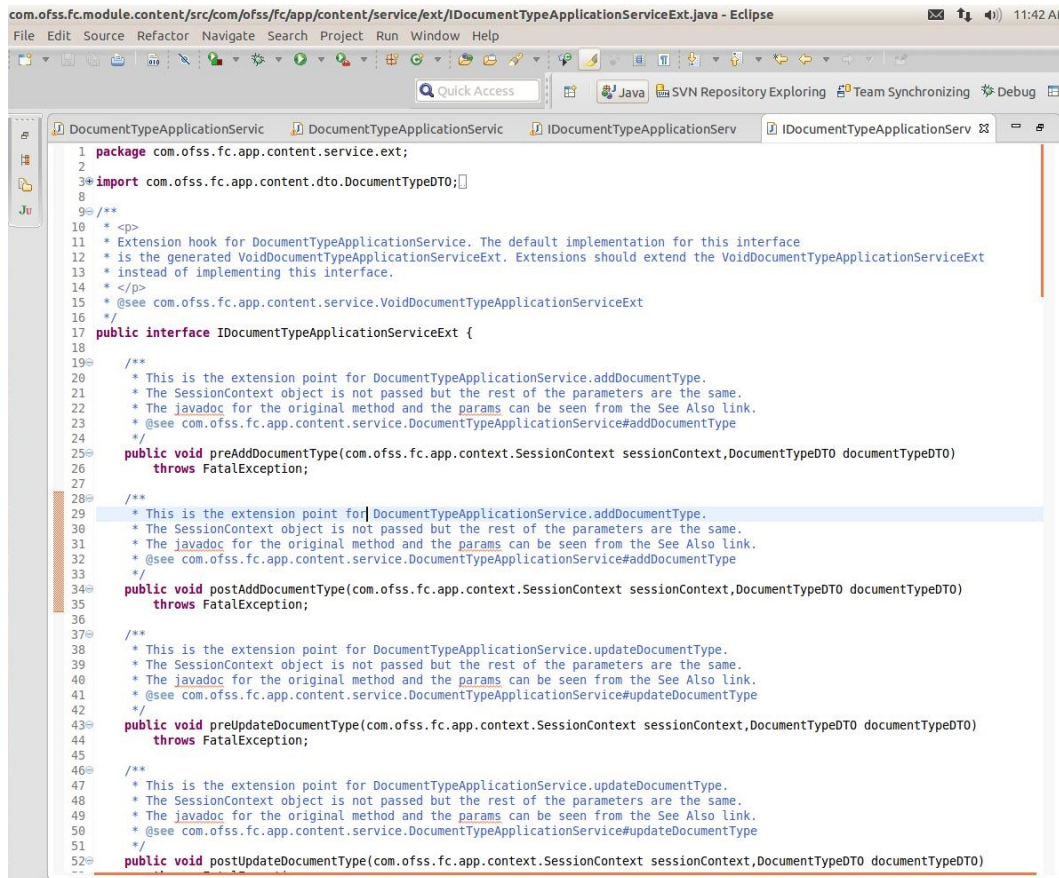
```

public void pre<Method_Name>(<Method_Parameters>) throws
    FatalException;
public void post<Method_Name>(<Method_Parameters>) throws
    FatalException;

```

A service extension class has to implement this interface. The pre method of the extension is executed before the actual service method and the post method of the extension is executed after the service method.

Figure 3–2 Extension Hook for Document Type Application Service



3.1.2 Default Application Service Extension

The OBP plug-in for eclipse generates a default extension for a particular service in the form of the class Void<Service_Name>ApplicationServiceExt. This class implements the aforementioned service extension interface without any business logic if the implemented methods are empty.

The default extension is a useful and 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. This is because the classes are auto-generated and reserved for product use and get overwritten as part of a bulk generation process.

Figure 3–3 Default Application Service Extension

```

1 package com.ofss.fc.app.content.service.ext;
2
3 import com.ofss.fc.app.content.dto.DocumentTypeDTO;
4
5
6
7
8
9 /**
10  * <p>
11  * Extension hook for DocumentTypeApplicationService. The default for the extension points.
12  * Each application service method for DocumentTypeApplicationService has corresponding pre
13  * and post methods. This default implementation returns and does nothing.
14  * Extenders are encouraged to extend this class instead of implementing
15  * the interface as they would have to then implement all methods. This class
16  * is provided for easing the writing of the extensions.
17  * </p>
18  */
19
20 @see com.ofss.fc.app.content.service.DocumentTypeApplicationService
21
22 public class VoidDocumentTypeApplicationServiceExt implements IDocumentTypeApplicationServiceExt {
23
24     /**
25      * This is the extension point for DocumentTypeApplicationService.addDocumentType.
26      * The SessionContext object is not passed but the rest of the parameters are the same.
27      * The javadoc for the original method and the params can be seen from the See Also link.
28      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#addDocumentType
29      */
30     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
31         throws FatalException {
32         return;
33     }
34
35     /**
36      * This is the extension point for DocumentTypeApplicationService.addDocumentType.
37      * The SessionContext object is not passed but the rest of the parameters are the same.
38      * The javadoc for the original method and the params can be seen from the See Also link.
39      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#addDocumentType
40      */
41     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
42         throws FatalException {
43         return;
44     }
45
46     /**
47      * This is the extension point for DocumentTypeApplicationService.updateDocumentType.
48      * The SessionContext object is not passed but the rest of the parameters are the same.
49      * The javadoc for the original method and the params can be seen from the See Also link.
50      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#updateDocumentType
51      */
52     public void preUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
53         throws FatalException {
54         return;
55     }
56
57 }

```

3.1.3 Application Service Extension Executor

The OBP plug-in for eclipse generates a service extension executor interface and an implementation for the executor interface. The naming convention for the generated executor classes which enable 'extension chaining' is as shown below:

```

Interface: <Application Service
Qualifier>ApplicationServiceExtExecutor
Implementation: <Application Service
Qualifier>ApplicationServiceExtExecutor

```

The service extension executor class, on load, creates an instance each of all the extensions defined in the service extensions configuration file. If no extensions are defined for a particular service, the executor creates an instance of the default extension for the service. The executor also has a pair of pre and post methods for each method of the actual service. These methods in turn call the corresponding methods of all the extension classes defined for the service.

Figure 3–4 Application Service Extension Executor

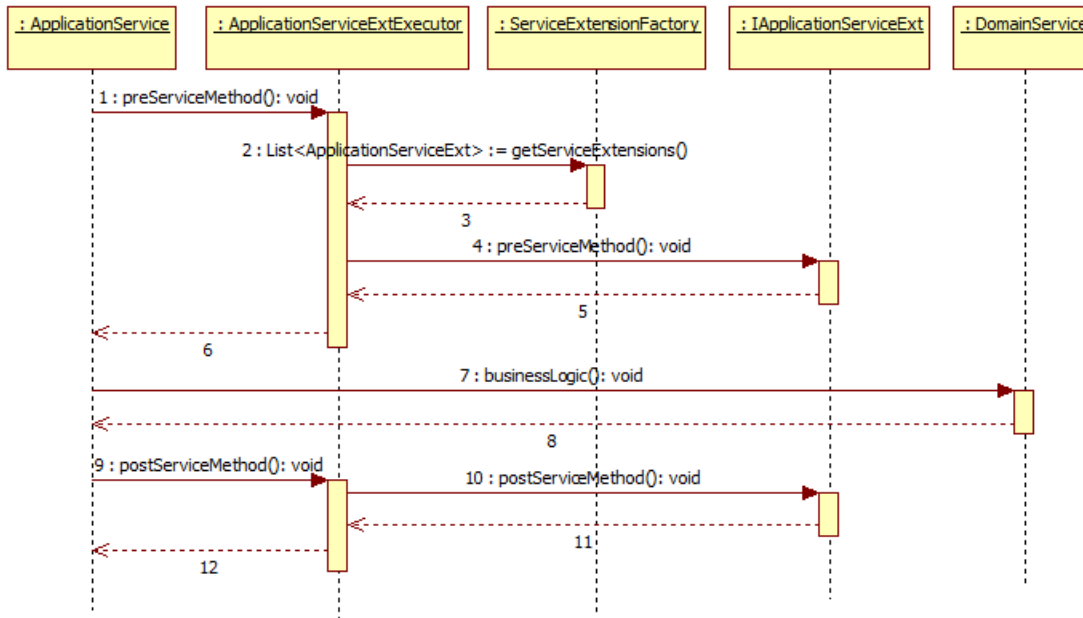
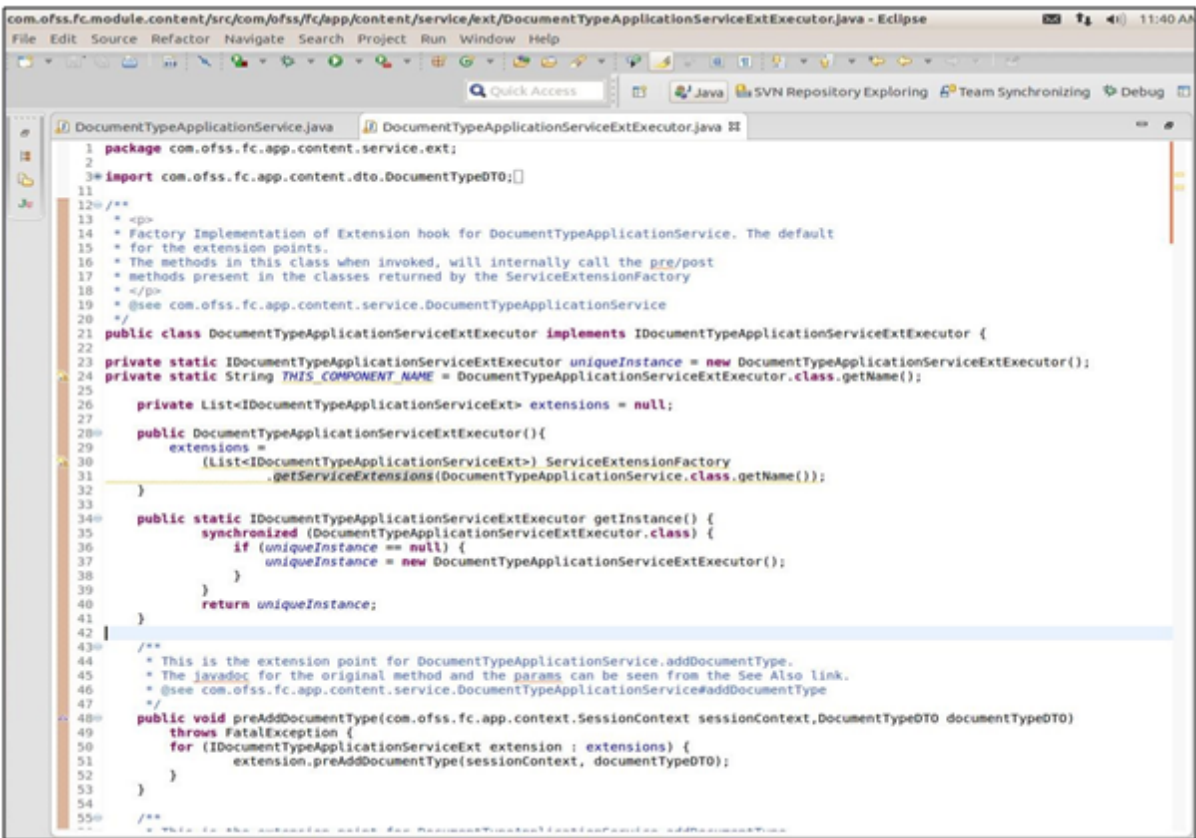


Figure 3–5 Extension Factory Hook for Document Type Application Service

```

com.ofss.fc.module.content/src/com/ofss.fc/app/content/service/ext/IDocumentTypeApplicationServiceExtExecutor.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java SVN Repository Exploring Team Synchronizing Debug
DocumentTypeApplicationService.java DocumentTypeApplicationServiceExtExecutor.jav IDocumentTypeApplicationServiceExtExecutor.js
1 package com.ofss.fc.app.content.service.ext;
2
3 import com.ofss.fc.app.content.dto.DocumentTypeDTO;
4
5 /**
6  * ExtensionFactory hook for DocumentTypeApplicationService. Extension Factories should extend
7  * the IDocumentTypeApplicationServiceExtExecutor
8  */
9
10 public interface IDocumentTypeApplicationServiceExtExecutor {
11
12     /**
13      * This is the extension point for DocumentTypeApplicationService.addDocumentType.
14      * The SessionContext object is not passed but the rest of the parameters are the same.
15      * The javadoc for the original method and the params can be seen from the See Also link.
16      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#addDocumentType
17      */
18     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
19         throws FatalException;
20
21     /**
22      * This is the extension point for DocumentTypeApplicationService.addDocumentType.
23      * The SessionContext object is not passed but the rest of the parameters are the same.
24      * The javadoc for the original method and the params can be seen from the See Also link.
25      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#addDocumentType
26      */
27     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
28         throws FatalException;
29
30     /**
31      * This is the extension point for DocumentTypeApplicationService.updateDocumentType.
32      * The SessionContext object is not passed but the rest of the parameters are the same.
33      * The javadoc for the original method and the params can be seen from the See Also link.
34      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#updateDocumentType
35      */
36     public void preUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
37         throws FatalException;
38
39     /**
40      * This is the extension point for DocumentTypeApplicationService.updateDocumentType.
41      * The SessionContext object is not passed but the rest of the parameters are the same.
42      * The javadoc for the original method and the params can be seen from the See Also link.
43      * @see com.ofss.fc.app.content.service.DocumentTypeApplicationService#updateDocumentType
44      */
45     public void postUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
46         throws FatalException;
47
48 }
  
```


Figure 3–6 Factory Implementation of Extension Hook for Document Type Application Service



3.1.4 Extension Configuration

The extension classes that implement the extension interface are mapped to the application service class with the help of a property file mapping inside `serviceextensions.properties`. The mapping convention to be specified is a service's fully qualified class name to comma separated extensions' fully qualified class name in the following format:

```
<service_class_name>=<extension_class_name>,<extension_class_name>...
```

Example Mapping : `config/properties/serviceextensions.properties`

Single extension configuration

```
com.ofss.fc.app.content.service.DocumentTypeApplicationService=
com.ofss.fc.app.content.service.ext.DocumentTypeApplicationService
Ext
```

Multiple extension configuration

```
com.ofss.fc.app.content.service.DocumentTypeApplicationService=
```

```

com.ofss.fc.app.content.service.ext.in.DocumentTypeApplicationServiceExtension,
com.ofss.fc.app.content.service.ext.in.mum.DocumentTypeApplicationServiceExtension,
com.ofss.fc.app.content.service.ext.in.mum.ExtendedDocumentTypeApplicationService,
com.ofss.fc.app.content.service.ext.in.blr.DocumentTypeApplicationServiceExtension,
com.ofss.fc.app.content.service.ext.in.blr.ExtendedDocumentTypeApplicationService

```

It is possible to configure multiple implementations of pre / post extensions for an executing service in this layer. This is achieved with the help of the extension executor which has the capability to loop through a set of extension implementations which conform to the extension interface which is supported by the service.

3.1.5 Application Service Extension Using Groovy

Application service extension can be implemented using Groovy. The sample code and steps for service extension implementation using groovy is as follows:

Service extension groovy implementation class 'VoidSubmissionDocumentApplicationServiceExt' implementing product service extension interface 'com.ofss.fc.app.Origination.service.core.submissiondocument.ext.ISubmissionDocumentApplicationServiceExt.

Figure 3–7 Application Service Extension Using Groovy

```

4
5 package com.ofss.fc.module.originationGroovy
6
7 import com.ofss.fc.app.context.SessionContext
8 import com.ofss.fc.app.Origination.dto.core.document.DocumentReferenceInputDTO
9 import com.ofss.fc.app.Origination.service.core.submissiondocument.ext.ISubmissionDocumentApplicationServiceExt
10 import com.ofss.fc.common.OriginationConstants
11 import com.ofss.fc.datatype.Date
12 import com.ofss.fc.enumeration.Origination.OfferDocReferenceType
13 import com.ofss.fc.infra.exception.FatalException
14 /**
15  * <p>
16  * Groovy Extension hook for SubmissionDocumentApplicationService. The customization for the extension points.
17  * Each application service method for SubmissionDocumentApplicationService has corresponding pre
18  * and post methods. Whenever the Service Extensions are overridden with Groovy Extensions, the call will
19  * go to corresponding pre/post groovy extensions method and will execute the implemented logic.
20  * </p>
21  * @see com.ofss.fc.app.Origination.service.core.submissiondocument.SubmissionDocumentApplicationService
22  */
23 public class VoidSubmissionDocumentApplicationServiceExt implements ISubmissionDocumentApplicationServiceExt {
24
25     /**
26      * This is the extension point for SubmissionDocumentApplicationService.createDocumentChecklist.
27      * The SessionContext object is not passed but the rest of the parameters are the same.
28      * The javadoc for the original method and the params can be seen from the See Also link.
29      * @see com.ofss.fc.app.Origination.service.core.submissiondocument.SubmissionDocumentApplicationService#createDocumentChecklist
30      */
31     public void preCreateDocumentChecklist(SessionContext sessionContext, DocumentReferenceInputDTO documentReferenceInputDTO) throws FatalException {
32     }
33     /**
34      * This is the extension point for SubmissionDocumentApplicationService.createDocumentChecklist.
35      * The SessionContext object is not passed but the rest of the parameters are the same.
36      * The javadoc for the original method and the params can be seen from the See Also link.
37      * @see com.ofss.fc.app.Origination.service.core.submissiondocument.SubmissionDocumentApplicationService#createDocumentChecklist
38      */
39
40     public void postCreateDocumentChecklist(SessionContext sessionContext, DocumentReferenceInputDTO documentReferenceInputDTO) throws FatalException {
41     }
42

```

3.2 Extended Application Service Extension – Extending the "appx" Layer

Provide the fully qualified name of the above groovy implementation in `flx_fw_config_all_b` against the corresponding service extension `prop_id` and `category_id`.

Figure 3–8 PROP_ID and CATEGORY_ID

PROP_ID	CATEGORY_ID	PROP_VALUE	FACTORY_SHIPPED_FLAG	PROP_COMMENTS
com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService	ServiceExtensions	com.ofss.fc.groovy.test.VoidSubmissionDocumentApplicationServiceExt	Y	(null)

Figure 3–9 SUMMARY_TEXT

SUMMARY_TEXT	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATED_DATE	
com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService	com.ofss.fc.groovy.test.VoidSubmissionDocumentApplicationServiceExt	ServiceExtensions ofssuser	24-NOV-15 03.07.11.00000000	PH ofssuser	24-NOV-15 03.07.11.0

Package the above implementation and add in custom library which the application is referring to and add the groovy li in the classpath of the server which will be taken care by deployment team.

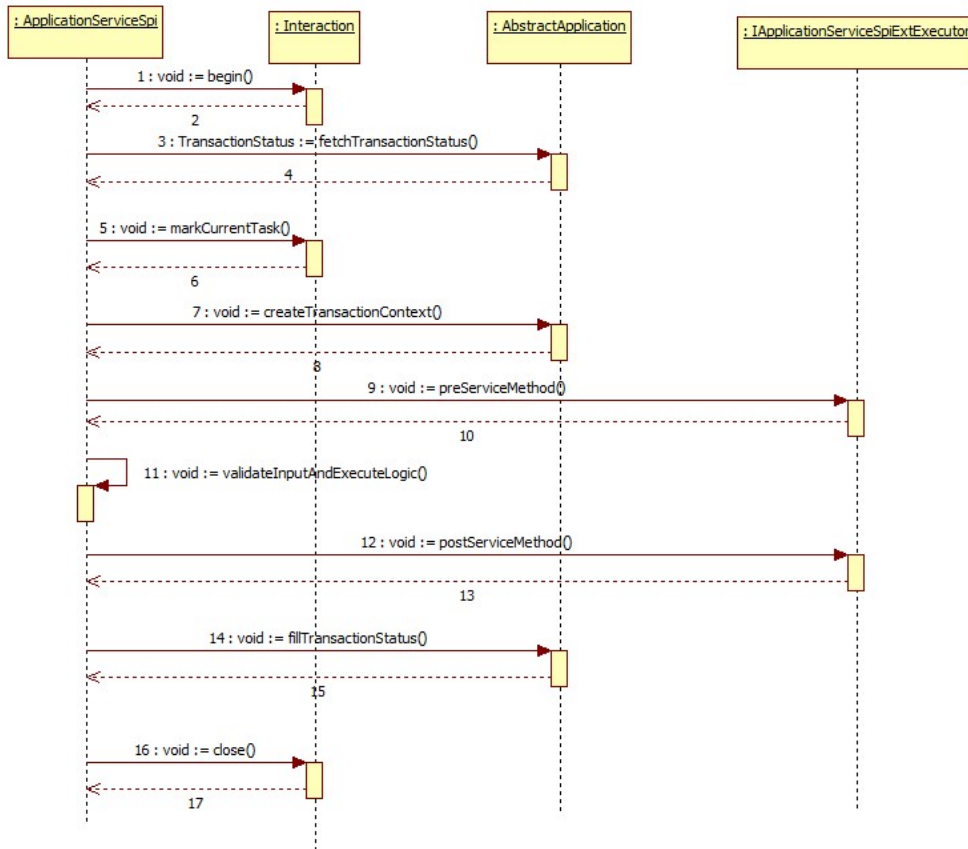
Figure 3–10 Add Groovy Library to Classpath

```
setDomainEnv.sh
465
466 if [ "${PRE_CLASSPATH}" != "" ]; then
467     PRE_CLASSPATH="/scratch/app/product/fmw/obpinatal/obp/obp-thirdparty-app.domain/APP-INF/lib/groovy-all-2.3.10.jar${CLASSPATHSEP}${PRE_CLASSPATH}"
468     export PRE_CLASSPATH
469 else
470     PRE_CLASSPATH="/scratch/app/product/fmw/obpinatal/obp/obp-thirdparty-app.domain/APP-INF/lib/groovy-all-2.3.10.jar"
471     export PRE_CLASSPATH
472 fi
```

3.2 Extended Application Service Extension – Extending the "appx" Layer

The "appx" layer is referred to as the extended application service layer. It represents the business logic that executes as part of a service method exposed by OBP middleware host with additional internal service calls to support extended features such as custom fields (that is, Dictionary pattern). The appx layer also provides extension support, on top of and on the lines of the app layer. The implementation of extension support in this layer is similar to the implementation of extension support in the app layer.

Figure 3–11 Extended Application Service Extension



The pre hook is provided before the invocation of actual application service call inside the extended application service layer. At this step, the extended host application core framework is aware of the executing service with respect to the authenticated subject or the user who has posted the transaction and an initialized status. At this step, the database session is also initialized and accessible enabling the user to use the same in the pre hook for any database access which might be required.

The post hook is provided after the primary application service which is extended in the appx layer along with the remaining internal service calls. This is required to support extended features like approval related processing and to complete execution before marking the service execution status as successful 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 including the approval view input data as their inputs. Additionally, if the response type of the object returned by the primary app layer application service is other *TransactionStatus*, the same is also accepted as input by the post hook.

The following sections explain the important concepts which should be understood for extending in this appx layer.

Figure 3–12 Extended Application Service Extension - Post and Pre Hook

```

3288 public DocumentTypeInquiryResponseSet fetchDocumentTypes(com.ofss.fc.app.context.SessionContext sessionContext,
3289 DocumentClassificationType documentClassificationType, FeeDetailsDTO feeDetails, LinkedUDFDTO linkedUDFDTO)
3290 throws FatalException {
3291     super.checkAccess("com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi.fetchDocumentTypes",
3292 sessionContext, documentClassificationType, feeDetails, linkedUDFDTO);
3293     com.ofss.fc.app.content.service.DocumentTypeApplicationService manager =
3294     new com.ofss.fc.app.content.service.DocumentTypeApplicationService();
3295     Interaction.begin(sessionContext);
3296     com.ofss.fc.appx.content.service.ext.IDocumentTypeApplicationServiceSpiExt helper =
3297     (com.ofss.fc.appx.content.service.ext.IDocumentTypeApplicationServiceSpiExt) com.ofss.fc.appx.ext.ServiceProviderExtension
3298     .getInstance().getServiceProviderExtension(
3299     "com.ofss.fc.appx.content.service.ext.IDocumentTypeApplicationServiceSpiExt",
3300     "com.ofss.fc.appx.content.service.ext.DocumentTypeApplicationServiceSpiExt");
3301     DocumentTypeInquiryResponseSet response = null;
3302     TransactionStatus transactionStatus = fetchTransactionStatus();
3303     String taskCode = null;
3304     try {
3305         extension.preFetchDocumentTypes(sessionContext, documentClassificationType, feeDetails, linkedUDFDTO);
3306         response = manager.fetchDocumentTypes(sessionContext, documentClassificationType);
3307         taskCode = Interaction.fetchCurrentTask();
3308         transactionStatus = applyServiceCharge(sessionContext, feeDetails, taskCode);
3309         fillTransactionStatus(transactionStatus);
3310         Map<String, Object> parentDTOMap = new HashMap<String, Object>();
3311         transactionStatus = addUDF(sessionContext, linkedUDFDTO, parentDTOMap);
3312         fillTransactionStatus(transactionStatus);
3313         extension.postFetchDocumentTypes(sessionContext, documentClassificationType, feeDetails, linkedUDFDTO,
3314 response);
3315         fillTransactionStatus(transactionStatus);
3316         response.setStatus(transactionStatus);
3317     } catch (FatalException e) {
3318         logger.log(Level.SEVERE, Interaction.fetchCurrentTask(), e);
3319         fillTransactionStatus(transactionStatus, e);
3320     } catch (Throwable e) {
3321         logger.log(Level.SEVERE, Interaction.fetchCurrentTask(), e);
3322         fillTransactionStatus(transactionStatus, e);
3323     } finally {
3324         Interaction.close();
3325     }
3326     super.checkResponse(sessionContext, response);
3327     return response;
3328 }
3329
3330 public DocumentTypeApplicationServiceSpi() {
3331     extension =
3332     ((IDocumentTypeApplicationServiceSpiExt) ServiceProviderExtensionFactory
3333     .getServiceProviderExtension(DocumentTypeApplicationServiceSpi.class.getName()));
3334 }
3335 }

```

The following concepts are important for extending in this service provider layer:

3.2.1 Extended Application Service Extension Interface

The OBP plug-in for eclipse generates an interface for the extension of a particular service. The interface name is in the form I<Service_Name>ApplicationServiceSpiExt. This interface has a pair of method definitions for each method of the present in the actual service. The signatures of these methods are:

```

public void pre<Method_Name>(<Method_Parameters>) throws
FatalException;
public void post<Method_Name>(<Method_Parameters>) throws
FatalException;

```

An extended application service extension class has to implement this interface. The pre method of the extension is executed before the actual service method and the post method of the extension is executed after the service method.

Figure 3–13 Extension Hook for Document Type Application Service Spi Ext

```

1 package com.ofss.fc.appx.content.service.ext;
2
3 import com.ofss.fc.app.account.dto.transaction.FeeDetailsDTO;
4
5
6
7
8
9
10 /**
11  * <p>
12  * Extension hook for DocumentTypeApplicationServiceSpi. The default implementation for this interface
13  * is the generated VoidDocumentTypeApplicationServiceSpiExt. Extensions should extend the VoidDocumentTypeApplicationServiceSpiExt
14  * instead of implementing this interface.
15  * </p>
16  * @see com.ofss.fc.appx.content.service.VoidDocumentTypeApplicationServiceSpiExt
17  */
18 public interface IDocumentTypeApplicationServiceSpiExt {
19
20     /**
21      * This is the extension point for DocumentTypeApplicationServiceSpi.preAddDocumentType.
22      * The SessionContext object is not passed but the rest of the parameters are the same.
23      * The javadoc for the original method and the params can be seen from the See Also link.
24      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#preAddDocumentType
25      */
26     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
27         throws FatalException;
28
29     /**
30      * This is the extension point for DocumentTypeApplicationServiceSpi.postAddDocumentType.
31      * The SessionContext object is not passed but the rest of the parameters are the same.
32      * The javadoc for the original method and the params can be seen from the See Also link.
33      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#postAddDocumentType
34      */
35     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
36         throws FatalException;
37
38     /**
39      * This is the extension point for DocumentTypeApplicationServiceSpi.preUpdateDocumentType.
40      * The SessionContext object is not passed but the rest of the parameters are the same.
41      * The javadoc for the original method and the params can be seen from the See Also link.
42      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#preUpdateDocumentType
43      */
44     public void preUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeD
45         throws FatalException;
46
47     /**
48      * This is the extension point for DocumentTypeApplicationServiceSpi.postUpdateDocumentType.
49      * The SessionContext object is not passed but the rest of the parameters are the same.
50      * The javadoc for the original method and the params can be seen from the See Also link.
51      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#postUpdateDocumentType
52      */
53     public void postUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentType

```

3.2.2 Default Implementation of Appx Extension

The OBP plug-in for eclipse generates a default service extension for a particular service in the form of the class `Void<Service_Name>ApplicationServiceSpiExt`. This class implements the aforementioned service provider extension interface without any business logic viz. the implemented methods are empty.

The default extension is a useful and 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. This is because the classes are auto-generated and reserved for product use and may get overwritten as part of a bulk generation process.

Figure 3–14 Default Implementation of Appx Extension

```

24 Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.appx.content.service.ext;
5
6 import com.ofss.fc.app.account.dto.transaction.FeeDetailsDTO;
7 import com.ofss.fc.app.content.dto.DocumentTypeInquiryResponse;
8 import com.ofss.fc.app.content.dto.DocumentTypeInquiryResponseSet;
9 import com.ofss.fc.app.udf.udfservice.dto.LinkUDFDTO;
10 import com.ofss.fc.enumeration.content.DocumentClassificationType;
11 import com.ofss.fc.infra.exception.FatalException;
12
14 * VoidDocumentTypeApplicationServiceSpiExt
18 public class VoidDocumentTypeApplicationServiceSpiExt implements IDocumentTypeApplicationServiceSpiExt {
19
20     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
21     /*
22     Void extension hook implemented with empty pre and post methods for the ApplicationServiceSpi. Each application
23     service method for the ApplicationServiceSpi has corresponding pre and post methods. This default
24     implementation does nothing. Usage guideline mandates extending this class instead of implementing the
25     interface as they would have to then implement all methods. This class is provided for easing the
26     writing of the extensions. An entry of extended class should be put in serviceextensions.properties.
27     */
28     /*
29     DO NOT ADD ANY CODE IN THESE METHODS AS IT WILL GET OVERRITTEN WHEN BULK GENERATION OF CODE HAPPENS
30     AND THE PERSON GENERATING DOES A BULK CHECK-IN !!!
31     */
32
33 }
34
35     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDT
36     /*
37     Void extension hook implemented with empty pre and post methods for the ApplicationServiceSpi. Each application
38     service method for the ApplicationServiceSpi has corresponding pre and post methods. This default
39     implementation does nothing. Usage guideline mandates extending this class instead of implementing the
40     interface as they would have to then implement all methods. This class is provided for easing the
41     writing of the extensions. An entry of extended class should be put in serviceextensions.properties.
42     */
43     /*
44     DO NOT ADD ANY CODE IN THESE METHODS AS IT WILL GET OVERRITTEN WHEN BULK GENERATION OF CODE HAPPENS
45     AND THE PERSON GENERATING DOES A BULK CHECK-IN !!!
46     */
47
48     public void preUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentType
49
50     public void postUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentType
51
52     public void preFetchDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeD
53
54     public void postFetchDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentType

```

3.2.3 Configuration

The service provider extension class to the service class mapping is defined in a property file `ServiceProviderExtensions.properties` under `config/properties`. Multiple extensions can be defined for a particular service provider with the help of the extension executor. The pre and post extensions are defined in the service layer.

The mapping is specified for a service provider extension interface's fully qualified class name to service provider extension class's fully qualified class name in the following format:

```
<service_provider_interface_name>=<service_provider_extension_
class_name>,<service_provider_extesion_class_name>
```

Example Mapping :

```
config/properties/ServiceProviderExtensions.properties
```

Single extension configuration

```
com.ofss.fc.appx.content.service.ext.DocumentTypeApplicationServiceSpi=
```

```
com.ofss.fc.appx.content.service.ext.DocumentTypeApplicationServiceSpiExt
```

Multiple extension configuration

```
com.ofss.fc.appx.content.service.ext.DocumentTypeApplicationServiceSpi=
```

```

com.ofss.fc.appx.content.service.ext.in.DocumentTypeApplicationServiceExt,
com.ofss.fc.appx.content.service.ext.in.mum.DocumentTypeApplicationServiceExt,
com.ofss.fc.appx.content.service.ext.in.mum.ExtendedDocumentTypeApplicationService,
com.ofss.fc.appx.content.service.ext.in.blr.DocumentTypeApplicationServiceExt,
com.ofss.fc.appx.content.service.ext.in.blr.ExtendedDocumentTypeApplicationService

```

3.2.4 Extended Application Service Extension Executor

The OBP plug-in for eclipse generates a service provider extensions executor interface and an implementation class in the form of the following naming convention.

```

I<ApplicationServiceQualifier>ApplicationServiceSpiExtExecutor
<ApplicationServiceQualifier>ApplicationServiceSpiExtExecutor

```

The extended application service extension executor class, on load, creates an instance each of all the extensions defined in the service provider extensions configuration file. If no extensions are defined for a particular service provider, the executor creates an instance of the default extension for the appx service. The executor also has a pair of pre and post methods for each method of the actual appx service. These methods in turn delegate the call to the corresponding methods of all the extension classes configured inside the properties file for the service provider.

Figure 3–15 Extended Application Service Extension Executor

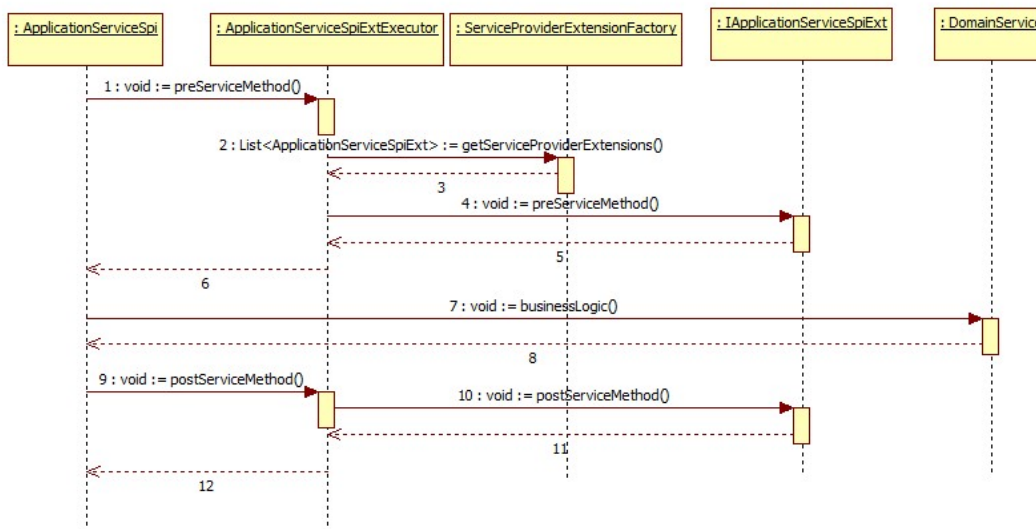


Figure 3–16 Extension Factory Hook for Document Type Application Service Spi Ext

```

1 package com.ofss.fc.appx.content.service.ext;
2
3 import com.ofss.fc.app.account.dto.transaction.FeeDetailsDTO;
4
5
6
7
8
9
10 /**
11  * <p>
12  * ExtensionFactory hook for DocumentTypeApplicationServiceSpi. Extension Factories should implement
13  * the IDocumentTypeApplicationServiceSpiExtExecutor
14  * </p>
15  */
16 public interface IDocumentTypeApplicationServiceSpiExtExecutor {
17
18     /**
19      * This is the extension point for DocumentTypeApplicationServiceSpi.preAddDocumentType.
20      * The javadoc for the original method and the params can be seen from the See Also link.
21      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#preAddDocumentType
22      */
23     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
24         throws FatalException;
25
26     /**
27      * This is the extension point for DocumentTypeApplicationServiceSpi.postAddDocumentType.
28      * The javadoc for the original method and the params can be seen from the See Also link.
29      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#postAddDocumentType
30      */
31     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
32         throws FatalException;
33
34     /**
35      * This is the extension point for DocumentTypeApplicationServiceSpi.preUpdateDocumentType.
36      * The javadoc for the original method and the params can be seen from the See Also link.
37      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#preUpdateDocumentType
38      */
39     public void preUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeD
40         throws FatalException;
41
42     /**
43      * This is the extension point for DocumentTypeApplicationServiceSpi.postUpdateDocumentType.
44      * The javadoc for the original method and the params can be seen from the See Also link.
45      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#postUpdateDocumentType
46      */
47     public void postUpdateDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentType
48         throws FatalException;
49
50     /**
51      * This is the extension point for DocumentTypeApplicationServiceSpi.preFetchDocumentType.
52      * The javadoc for the original method and the params can be seen from the See Also link.
53      * @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi#preFetchDocumentType
54

```

Figure 3–17 Factory Implementation of Extension Hook for Document Type Application Service Spi Ext

```

1 package com.ofss.fc.appx.content.service.ext;
2
3 import com.ofss.fc.app.account.dto.transaction.FeeDetailsDTO;
4
5 /**
6  * Factory Implementation of Extension hook for DocumentTypeApplicationServiceSpi. The default
7  * for the extension points.
8  * The methods in this class when invoked, will internally call the pre/post
9  * methods present in the classes returned by the ServiceProviderExtensionFactory
10 * which implement the extension interface.
11 */
12 @see com.ofss.fc.appx.content.service.DocumentTypeApplicationServiceSpi
13
14 public class DocumentTypeApplicationServiceSpiExtExecutor implements IDocumentTypeApplicationServiceSpiExtExecutor {
15
16     private static IDocumentTypeApplicationServiceSpiExtExecutor uniqueInstance = new DocumentTypeApplicationServiceSpiExtExecutor();
17     private static String THIS_COMPONENT_NAME = DocumentTypeApplicationServiceSpiExtExecutor.class.getName();
18
19     private List<IDocumentTypeApplicationServiceSpiExt> extensions = null;
20
21     public DocumentTypeApplicationServiceSpiExtExecutor(){
22         extensions =
23             (List<IDocumentTypeApplicationServiceSpiExt>) ServiceProviderExtensionFactory
24                 .getServiceProviderExtensions(DocumentTypeApplicationServiceSpi.class.getName());
25     }
26
27     public static IDocumentTypeApplicationServiceSpiExtExecutor getInstance() {
28         if (uniqueInstance == null) {
29             synchronized (DocumentTypeApplicationServiceSpiExtExecutor.class) {
30                 if (uniqueInstance == null) {
31                     uniqueInstance = new DocumentTypeApplicationServiceSpiExtExecutor();
32                 }
33             }
34         }
35         return uniqueInstance;
36     }
37
38     * This is the extension point for DocumentTypeApplicationServiceSpi.preAddDocumentType.
39     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO
40     throws FatalException {
41         for (IDocumentTypeApplicationServiceSpiExt extension : extensions) {
42             extension.preAddDocumentType(sessionContext, documentTypeDTO, feeDetails, linkedUDFDTO);
43         }
44     }
45
46     * This is the extension point for DocumentTypeApplicationServiceSpi.postAddDocumentType.
47     public void postAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, com.ofss.fc.app.content.dto.DocumentTypeDTO

```

3.2.5 Application Service "appx" Extension using Groovy

Application service extension can be implemented using Groovy. The sample code and steps for service extension implementation using groovy is as follows:

Service extension groovy implementation class 'VoidSubmissionDocumentApplicationServiceExt' implementing product service extension interface

'com.ofss.fc.app.Origination.service.core.submissiondocument.ext.ISubmissionDocumentApplicationServiceExt.

Figure 3–18 Application Service Appx Extension using Groovy

```

4
5 package com.ofss.fc.module.originationGroovy
6
7 import com.ofss.fc.app.context.SessionContext
8 import com.ofss.fc.app.origination.dto.core.document.DocumentReferenceInputDTO
9 import com.ofss.fc.app.origination.service.core.submissiondocument.ext.ISubmissionDocumentApplicationServiceExt
10 import com.ofss.fc.common.OriginationConstants
11 import com.ofss.fc.datatype.Date
12 import com.ofss.fc.enumeration.origination.OfferDocReferenceType
13 import com.ofss.fc.infra.exception.FatalException
14 /**
15  * <p>
16  * Groovy Extension hook for SubmissionDocumentApplicationService. The customization for the extension points.
17  * Each application service method for SubmissionDocumentApplicationService has corresponding pre
18  * and post methods. Whenever the Service Extensions are overridden with Groovy Extensions, the call will
19  * go to corresponding pre/post groovy extensions method and will execute the implemented logic.
20  * </p>
21  * @see com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService
22  */
23 public class VoidSubmissionDocumentApplicationServiceExt implements ISubmissionDocumentApplicationServiceExt {
24
25     /**
26      * This is the extension point for SubmissionDocumentApplicationService.createDocumentChecklist.
27      * The SessionContext object is not passed but the rest of the parameters are the same.
28      * The javadoc for the original method and the params can be seen from the See Also link.
29      * @see com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService#createDocumentChecklist
30      */
31     public void preCreateDocumentChecklist(SessionContext sessionContext, DocumentReferenceInputDTO documentReferenceInputDTO) throws FatalException {
32     }
33     /**
34      * This is the extension point for SubmissionDocumentApplicationService.createDocumentChecklist.
35      * The SessionContext object is not passed but the rest of the parameters are the same.
36      * The javadoc for the original method and the params can be seen from the See Also link.
37      * @see com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService#createDocumentChecklist
38      */
39
40     public void postCreateDocumentChecklist(SessionContext sessionContext, DocumentReferenceInputDTO documentReferenceInputDTO) throws FatalException {
41     }
42

```

Provide the fully qualified name of the above groovy implementation in flx_fw_config_all_b against the corresponding service extension prop_id and category_id.

Figure 3–19 PROP_ID and CATEGORY_ID

PROP_ID	CATEGORY_ID	PROP_VALUE	FACTORY_SHIPPED_FLAG	PROP_COMMENTS
com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService ServiceExtensions	com.ofss.fc.groovy.test.VoidSubmissionDocumentApplicationServiceExt	Y		(null)

Figure 3–20 SUMMARY_TEXT

SUMMARY_TEXT	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATED_DATE
com.ofss.fc.app.origination.service.core.submissiondocument.SubmissionDocumentApplicationService com.ofss.fc.groovy.test.VoidSubmissionDocumentApplicationServiceExt ServiceExtensions ofssuser		24-NOV-15 03:07:11.000000000	PN ofssuser	24-NOV-15 03:07:11.000000000

Package the above implementation and add in custom library which the application is referring to and add the groovy li in the classpath of the server which will be taken care by deployment team.

Figure 3–21 Add Groovy Library to Classpath

```

setDomainEnv.sh
465
466 if [ "${PRE_CLASSPATH}" != "" ] ; then
467     PRE_CLASSPATH="/scratch/app/product/fmw/ohp/install/ohp/ohp.thirdparty.app.domain/APP-INF/lib/groovy-all-2.3.10.jar${CLASSPATHSEP}${PRE_CLASSPATH}"
468     export PRE_CLASSPATH
469 else
470     PRE_CLASSPATH="/scratch/app/product/fmw/ohp/install/ohp/ohp.thirdparty.app.domain/APP-INF/lib/groovy-all-2.3.10.jar"
471     export PRE_CLASSPATH
472 fi

```

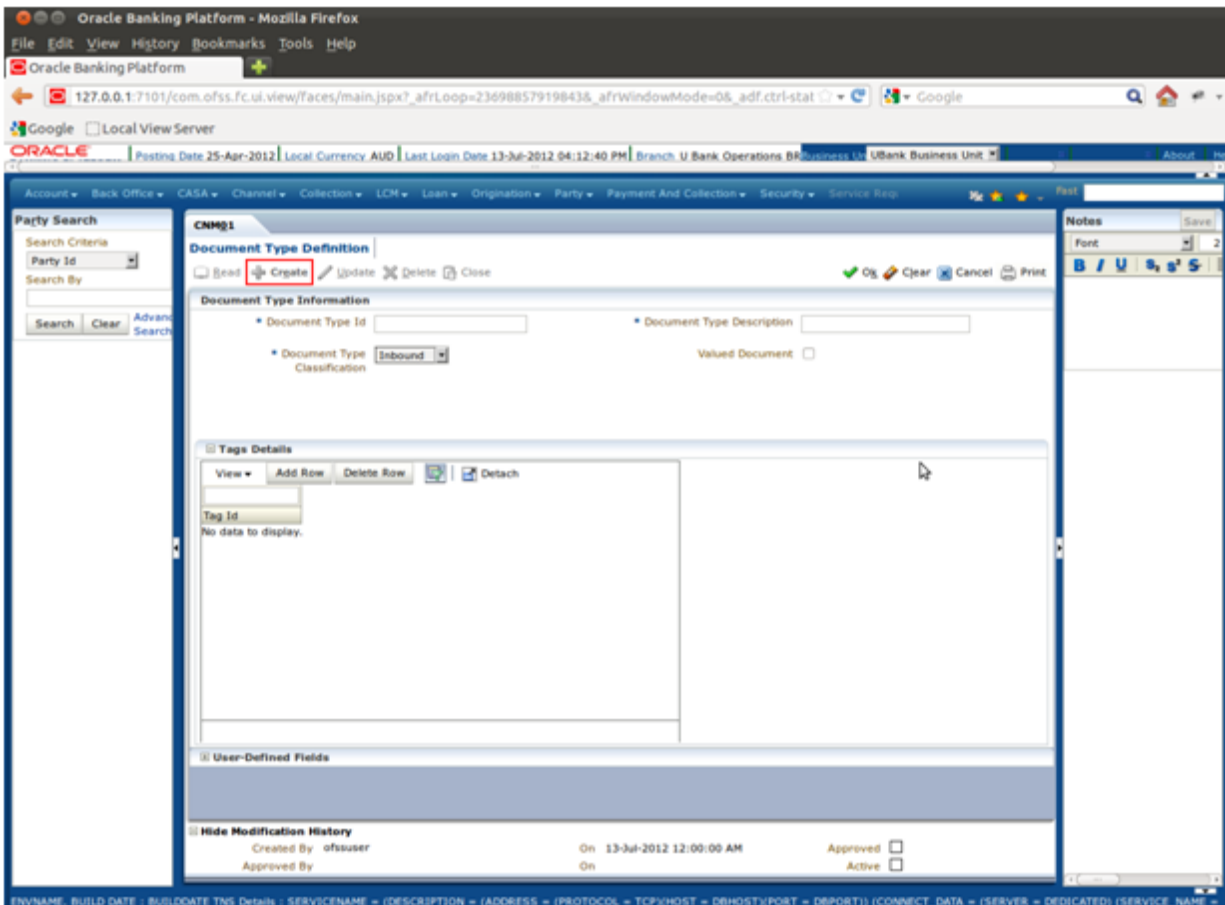
3.3 End-to-End Example of an Extension

This section gives an end-to-end example of extensions written in the appx layer using the extended application service extensions as well as app layer application service extensions. The example shall implement this by extending the default implementation of the appx extension class `Void<ApplicationServiceQualifier>ApplicationServiceSpiExt` class and app extension class `Void<ApplicationServiceQualifier>ApplicationServiceExt`.

For example, Back Office -> Content -> Document Type Definition screen of the application.

This screen is used for the maintenance of Document Types defined in the application.

Figure 3–22 Maintenance of Document Types

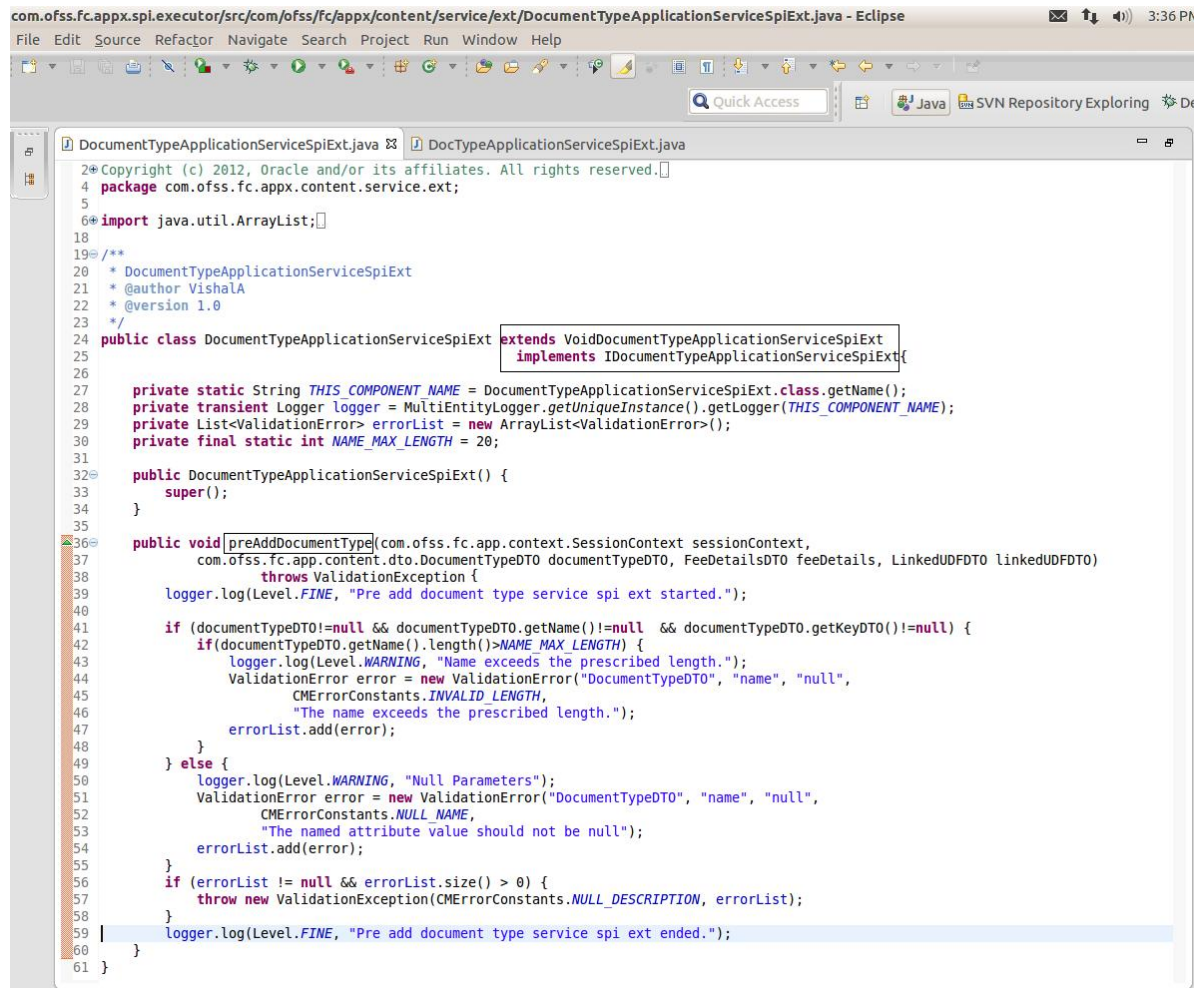


The Create tab of the screen allows a user to create document types in the application. On click of Ok, and after successful validation of the input entered by the user, the screen extracts the values. It calls the `DocumentTypeApplicationServiceSpi` (in appx layer) and `DocumentTypeApplicationService` (in app layer) on the host application to save the document type in the system.

In this example, we have added multiple extensions to this service of the appx layer through the extension executor, where the update of the description is done in one of the extension and check the length of name in another in the pre extension method.

3.3 End-to-End Example of an Extension

Figure 3–23 Document Type Application Service Spi Ext - Appx Layer



```
com.ofss.fc.appx.spi.executor/src/com/ofss/fc/appx/content/service/ext/DocumentTypeApplicationServiceSpiExt.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access Java SVN Repository Exploring
DocumentTypeApplicationServiceSpiExt.java DocTypeApplicationServiceSpiExt.java
2* Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.appx.content.service.ext;
5
6* import java.util.ArrayList;
18
19 /**
20  * DocumentTypeApplicationServiceSpiExt
21  * @author Vishala
22  * @version 1.0
23  */
24 public class DocumentTypeApplicationServiceSpiExt extends VoidDocumentTypeApplicationServiceSpiExt
25 implements IDocumentTypeApplicationServiceSpiExt {
26
27     private static String THIS_COMPONENT_NAME = DocumentTypeApplicationServiceSpiExt.class.getName();
28     private transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
29     private List<ValidationError> errorList = new ArrayList<ValidationError>();
30     private final static int NAME_MAX_LENGTH = 20;
31
32     public DocumentTypeApplicationServiceSpiExt() {
33         super();
34     }
35
36     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext,
37         com.ofss.fc.app.content.dto.DocumentTypeDTO documentTypeDTO, FeeDetailsDTO feeDetails, LinkedUDFDTO linkedUDFDTO)
38         throws ValidationException {
39         logger.log(Level.FINE, "Pre add document type service spi ext started.");
40
41         if (documentTypeDTO != null && documentTypeDTO.getName() != null && documentTypeDTO.getKeyDTO() != null) {
42             if (documentTypeDTO.getName().length() > NAME_MAX_LENGTH) {
43                 logger.log(Level.WARNING, "Name exceeds the prescribed length.");
44                 ValidationError error = new ValidationError("DocumentTypeDTO", "name", "null",
45                     CMErrorsConstants.INVALID_LENGTH,
46                     "The name exceeds the prescribed length.");
47                 errorList.add(error);
48             }
49         } else {
50             logger.log(Level.WARNING, "Null Parameters");
51             ValidationError error = new ValidationError("DocumentTypeDTO", "name", "null",
52                 CMErrorsConstants.NULL_NAME,
53                 "The named attribute value should not be null");
54             errorList.add(error);
55         }
56         if (errorList != null && errorList.size() > 0) {
57             throw new ValidationException(CMErrorsConstants.NULL_DESCRIPTION, errorList);
58         }
59         logger.log(Level.FINE, "Pre add document type service spi ext ended.");
60     }
61 }
```

Figure 3–24 Doc Type Application Service Spi Ext - Appx Layer

```

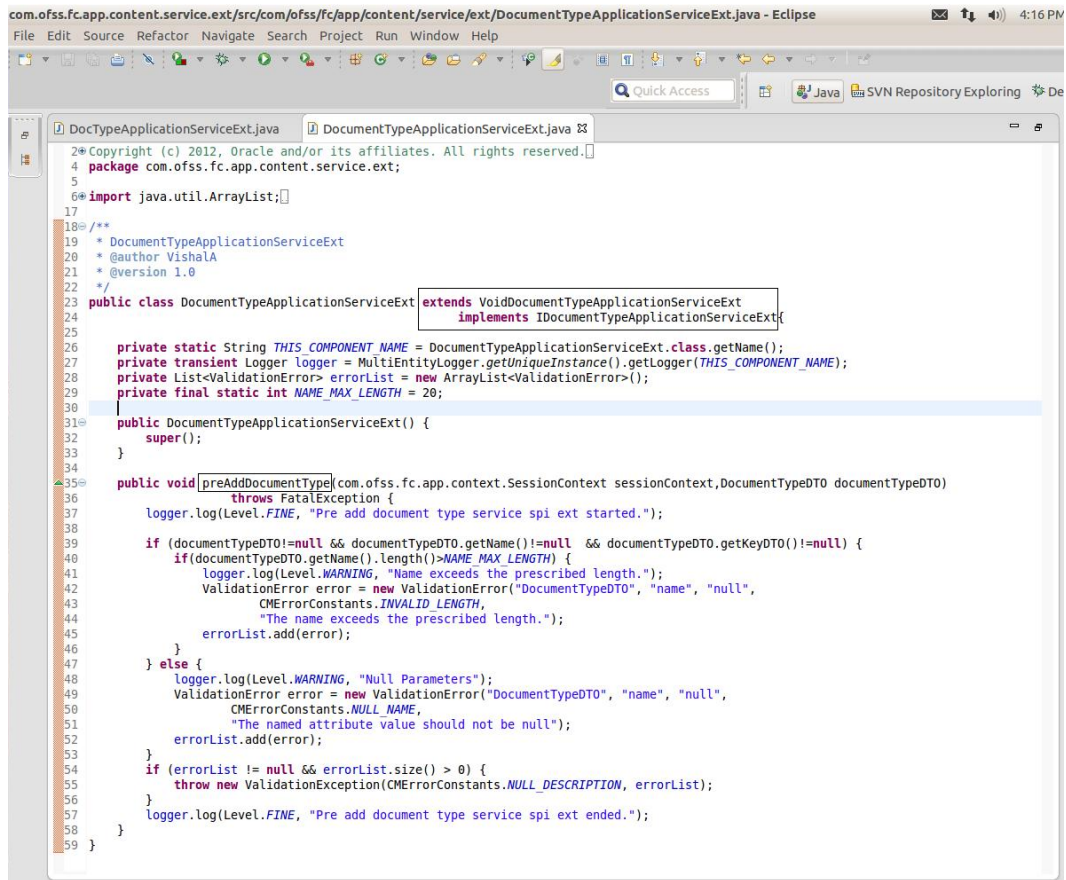
2* Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.appx.content.service.ext;
5
6* import java.util.ArrayList;
18
19**
20 * DocTypeApplicationServiceSpiExt
21 * @author VishalA
22 * @version 1.0
23 */
24 public class DocTypeApplicationServiceSpiExt extends VoidDocTypeApplicationServiceSpiExt
    implements IDocumentTypeApplicationServiceSpiExt {
25
26
27     private static String THIS_COMPONENT_NAME = DocTypeApplicationServiceSpiExt.class.getName();
28     private transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
29     private List<ValidationError> errorList = new ArrayList<ValidationError>();
30
31     public DocTypeApplicationServiceSpiExt() {
32         super();
33     }
34
35     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext,
36         com.ofss.fc.app.content.dto.DocumentTypeDTO documentTypeDTO, FeedDetailsDTO feedDetails, LinkedUDFDTO linkedUDFDTO)
37         throws ValidationException {
38         logger.log(Level.FINE, "Pre add document type service spi ext started.");
39
40         if (documentTypeDTO!=null && documentTypeDTO.getDescription()!=null) {
41             String newDescription = documentTypeDTO.getDescription();
42             concat("This sample description is appended to the earlier description.");
43             documentTypeDTO.setDescription(newDescription);
44         } else {
45             logger.log(Level.WARNING, "Null Parameters");
46             ValidationError error = new ValidationError("DocumentTypeDTO", "description", "null",
47                 CMErrConstants.NULL_DESCRIPTION,
48                 "The description attribute value should not be null");
49             errorList.add(error);
50             if (errorList != null && errorList.size() > 0) {
51                 throw new ValidationException(CMErrConstants.NULL_DESCRIPTION, errorList);
52             }
53         }
54         logger.log(Level.FINE, "Pre add document type service spi ext ended.");
55     }
56 }
57

```

In this example, we have added multiple extensions to the service of the app layer through the extension executor. We have implemented a not null and size check on the document tags in pre hook of the app layer to validate that document tags are sent as input in the application service.

3.3 End-to-End Example of an Extension

Figure 3–25 Document Type Application Service Spi Ext - App Layer



```
com.ofss.fc.app.content.service.ext/src/com/ofss/fc/app/content/service/ext/DocumentTypeApplicationServiceExt.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access Java SVN Repository Exploring De

DocTypeApplicationServiceExt.java DocumentTypeApplicationServiceExt.java
2 * Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.app.content.service.ext;
5
6 import java.util.ArrayList;
7
18 /**
19  * DocumentTypeApplicationServiceExt
20  * @author VishalA
21  * @version 1.0
22  */
23 public class DocumentTypeApplicationServiceExt extends VoidDocumentTypeApplicationServiceExt
24     implements IDocumentTypeApplicationServiceExt{
25
26     private static String THIS_COMPONENT_NAME = DocumentTypeApplicationServiceExt.class.getName();
27     private transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
28     private List<ValidationError> errorList = new ArrayList<ValidationError>();
29     private final static int NAME_MAX_LENGTH = 20;
30
31     public DocumentTypeApplicationServiceExt() {
32         super();
33     }
34
35     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
36         throws FatalException {
37         logger.log(Level.FINE, "Pre add document type service spi ext started.");
38
39         if (documentTypeDTO != null && documentTypeDTO.getName() != null && documentTypeDTO.getKeyDTO() != null) {
40             if (documentTypeDTO.getName().length() > NAME_MAX_LENGTH) {
41                 logger.log(Level.WARNING, "Name exceeds the prescribed length.");
42                 ValidationError error = new ValidationError("DocumentTypeDTO", "name", "null",
43                     CMErrorsConstants.INVALID_LENGTH,
44                     "The name exceeds the prescribed length.");
45                 errorList.add(error);
46             }
47         } else {
48             logger.log(Level.WARNING, "Null Parameters");
49             ValidationError error = new ValidationError("DocumentTypeDTO", "name", "null",
50                 CMErrorsConstants.NULL_NAME,
51                 "The named attribute value should not be null");
52             errorList.add(error);
53         }
54         if (errorList != null && errorList.size() > 0) {
55             throw new ValidationException(CMErrorsConstants.NULL_DESCRIPTION, errorList);
56         }
57         logger.log(Level.FINE, "Pre add document type service spi ext ended.");
58     }
59 }
```

Figure 3–26 Doc Type Application Service Spi Ext - App Layer

```

20 Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.app.content.service.ext;
5
6 import java.util.ArrayList;
7
18 /**
19  * DocTypeApplicationServiceExt
20  * @author VishalA
21  * @version 1.0
22  */
23 public class DocTypeApplicationServiceExt extends VoidDocumentTypeApplicationServiceExt
24     implements IDocumentTypeApplicationServiceExt {
25
26     private static String THIS_COMPONENT_NAME = DocTypeApplicationServiceExt.class.getName();
27     private transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
28     private List<ValidationError> errorList = new ArrayList<ValidationError>();
29
30     public DocTypeApplicationServiceExt() {
31         super();
32     }
33
34     public void preAddDocumentType(com.ofss.fc.app.context.SessionContext sessionContext, DocumentTypeDTO documentTypeDTO)
35         throws FatalException {
36         logger.log(Level.FINE, "Pre add document type service spi ext started.");
37
38         if (documentTypeDTO != null && documentTypeDTO.getDescription() != null) {
39             String newDescription = documentTypeDTO.getDescription();
40             concat("This sample description is appended to the earlier description.");
41             documentTypeDTO.setDescription(newDescription);
42         } else {
43             logger.log(Level.WARNING, "Null Parameters");
44             ValidationError error = new ValidationError("DocumentTypeDTO", "description", "null",
45                 CMErrorsConstants.NULL_DESCRIPTION,
46                 "The description attribute value should not be null");
47             errorList.add(error);
48             if (errorList != null && errorList.size() > 0) {
49                 throw new ValidationException(CMErrorsConstants.NULL_DESCRIPTION, errorList);
50             }
51         }
52         logger.log(Level.FINE, "Pre add document type service spi ext ended.");
53     }
54 }
55

```

3.4 Support for Middleware Specific Tasks and Application service

In case of OBP middleware implementation, SPI layer has ability to perform tasks before and after execution of application service. Also, you can have customized implementation of application service.

Following are the advantages of this feature:

1. OBP signatures and Spi content will be same across all sites irrespective of OBP-middleware or Product processor implementation.
2. No appreciable change is required when the bank migrates from OBP Middleware to a full-fledged OBP product processor implementation.
3. OBP Middleware signatures are self-sufficient to address integrations to non-OBP core servicing systems and there is no need for wrapper consulting Spi class to be created.

3.4.1 Pre and Post Middleware Specific Transaction Tasks Overview

- Methods 'performMiddlewareSpecificPreTransactionTasks' and 'performMiddlewareSpecificPostTransactionTasks' is available in every spi to execute tasks.

- Pre tasks generally includes request enrichment, pre transaction auditing, business policy validations, post tasks generally includes alert processing, notification to external system.
- For example, in HDFC bank, in fund transfer transactions referenceNumber field is defaulted in pre processing if request comes from net banking.
- Tasks will be performed only in case of middleware implementation.
- Response enrichment: Response fields can be populated via metadata mapping.
- Example: “com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails” method look like this.

Figure 3–27 Pre and Post Middleware Specific Transaction Tasks Overview

```

public PartyInquiryResponse fetchPartyDetails(com.ofss.fc.app.context.SessionContext sessionContext,
                                             com.ofss.fc.app.party.dto.core.PartyDTO partyDTO,
                                             WorkItemViewObjectDTO[] workItemViewObjectDTO) throws FatalException {
    super.checkAccess("com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails", sessionContext, partyDTO, workItemViewObjectDTO);
    ThreadAttribute.set(ThreadAttribute.VO_OBJECT_LOCAL, workItemViewObjectDTO);
    Interaction.begin(sessionContext);
    extension = (IPartyApplicationServiceSpiExtExecutor) com.ofss.fc.appx.ext.ServiceProviderExtensionFactory.getServiceProviderExtensionExecutor(PartyApplicationServiceSpi.c
    PartyInquiryResponse response = new PartyInquiryResponse();
    TransactionStatus transactionStatus = fetchTransactionStatus();
    try {
        extension.preFetchPartyDetails(sessionContext, partyDTO, workItemViewObjectDTO);
        super.performMiddlewareSpecificPreTransactionTasks(sessionContext, partyDTO, workItemViewObjectDTO);
        IPartyApplicationService iPartyApplicationService = createBusinessServiceInstance("com.ofss.fc.app.party.service.core.PartyApplicationService");
        response = iPartyApplicationService.fetchPartyDetails(sessionContext, partyDTO);
        super.performMiddlewareSpecificPostTransactionTasks(response, transactionStatus, sessionContext, partyDTO, workItemViewObjectDTO);
        extension.postFetchPartyDetails(sessionContext, partyDTO, workItemViewObjectDTO, response);
        fillTransactionStatus(transactionStatus);
        response.setStatus(transactionStatus);
    } catch (InvocationTargetException e) {
        logger.log(Level.SEVERE, Interaction.fetchCurrentTask(), e.getTargetException());
        fillTransactionStatus(transactionStatus, e.getTargetException());
    } catch (FatalException e) {
        logger.log(Level.SEVERE, Interaction.fetchCurrentTask(), e);
        fillTransactionStatus(transactionStatus, e);
    } catch (Throwable e) {
        logger.log(Level.SEVERE, Interaction.fetchCurrentTask(), e);
        fillTransactionStatus(transactionStatus, e);
    } finally {
        fillServiceResponse(response, transactionStatus);
        Interaction.close();
    }
    super.checkResponse(sessionContext, response);
    return response;
}

```

3.4.2 Sample Configuration

Middleware task configuration is based on channel and service Id. The DB tables associated with the execution steps are:

- **FLX_FW_MW_TASKS:** This table is used to make entries for middleware specific task based on channel and service id.

Sample entry for 'com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails' will look like, where PartyDeatilsAdapter is having several methods to perform tasks like pre business policy, post business policy, pre and post processing.

Figure 3–28 FLX_FW_MW_TASKS

CHANNEL_ID	APP_SERVICE_NAME	METHOD_NAME	CATEGORY_ID	EXECUTION_ORDER	CALL_ATTR_ID
1	com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi	fetchPartyDetails	PreProcessing	1	PARTY_DETAILS_PreProcessing
2	com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi	fetchPartyDetails	PreBusinessPolicy	1	PARTY_DETAILS_PreBusinessPolicy
3	com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi	fetchPartyDetails	PostProcessing	1	PARTY_DETAILS_PostProcessing
4	com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi	fetchPartyDetails	PostBusinessPolicy	1	PARTY_DETAILS_PostBusinessPolicy

ADAPTER_FACTORY_CONSTANT	ADAPTER_NAME	ADAPTER_METHOD	DTO_CLASS	IS_ENABLED	IGNORE_EXCEPTION
PARTY_ADAPTER_FACTORY	PartyDetailsAdapter	fetchPartyDetailsPreProcessing	com.ofss.fc.app.party.dto.core.PartyDTO	Y	N
PARTY_ADAPTER_FACTORY	PartyDetailsAdapter	fetchPartyDetailsPreBusinessPolicy	com.ofss.fc.app.party.dto.core.PartyDTO	Y	N
PARTY_ADAPTER_FACTORY	PartyDetailsAdapter	fetchPartyDetailsPostProcessing	com.ofss.fc.app.party.dto.core.PartyDTO	Y	N
PARTY_ADAPTER_FACTORY	PartyDetailsAdapter	fetchPartyDetailsPostBusinessPolicy	com.ofss.fc.app.party.dto.core.PartyDTO	Y	N

- **FLX_FW_MW_TASKS.DTO_DEFN:** This table is used to make entires for DTO class and DTO fields for response enrichment purpose.

Sample entry for service name

'com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails' will look like, where PartyInquiryResponse is response type and AdapterResponsibilityChainResponse is type of holder dto for response enrichment.

Figure 3–29 FLX_FW_MW_TASKS.DTO_DEFN

DTO_CLASS	FIELD_NAME
1 com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse	postExecutionResponse1
2 com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse	preExecutionResponse1
3 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	partyType
4 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	individualDTO
5 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	organizationDTO
6 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	bankingEntityDTO
7 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	trustDTO
8 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	comments
9 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	dateofOnboarding
10 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	roleSpecificDetail
11 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	partyWarningIndicatorDTO
12 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	employmentHistoryDTO
13 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	applicantMarketingConsentDTO
14 com.ofss.fc.app.party.dto.core.PartyInquiryResponse	blacklisted

- **FLX_FW_MW_TASKS.DTO_MAP:** This table is used to establish mapping between flw_fw_mw_tasks_dto_defn columns(dto class and dto field) and task entry defined in flx_fw_mw_tasks column (call_attr_id).

Sample entry for service id

'com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails (service name + service method name) task entry and response enrich dto field mappings, where field name postExecutionResponse is having mapping with PartyDetailsAdapter method fetchPartyDetailsPostProcesing through cod_attr_id PARTY_DATAILS_PreProcessing.

Figure 3–30 FLX_FW_MW_TASKS_DTO_MAP

SERVICE_ID	CATEGORY_ID	DTO_CLASS	FIELD_NAME	COD_ATTR_ID
com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails	PostProcessing	com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse	postExecutionResponse1	PARTY_DETAILS_PostProcessing
com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails	PreProcessing	com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse	preExecutionResponse1	PARTY_DETAILS_PreProcessing
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	partyType	PartyType
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	individualDTO	IndividualDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	organisationDTO	OrganisationDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	bankingEntryDTO	BankingEntryDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	trustDTO	TrustDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	comments	Comments
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	dateofOnboarding	DateofOnboarding
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	roleSpecificDetail	RoleSpecificDetail
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	partyWarningIndicatorDTO	PartyWarningIndicatorDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	employmentHistoryDTO	EmploymentHistoryDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	applicantMarketingConsentDTO	ApplicantMarketingConsentDTO
DEFAULT	ResponseDTO	com.ofss.fc.app.party.dto.core.PartyInquiryResponse	blacklisted	BlackListed

- FLX_MD_SERVICE_ATTR:** This table is used to keep entry for source and destination dto for response enrichment purpose through column entry ref_field_defn_id.

Sample entry for service id

'com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails', where enriched dto fields through adapter method having cod_attr_id like RESP_ENRICH.X.

Figure 3–31 FLX_MD_SERVICE_ATTR

COD_SERVICE_ATTR_ID	TYP_DATA_AVAIL	TYP_DATA_SRC	COD_ATTR_ID
1 com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails.RESP_ENRICH.PartyWarningIndicatorDTO.DTO	INDIRECT	OUTPUT	RESP_ENRICH.PartyWarningIndicatorDTO
2 com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails.RESP_ENRICH.DateofOnboarding.DTO	INDIRECT	OUTPUT	RESP_ENRICH.DateofOnboarding

COD_SERVICE_ID	PARAMETER_NAME	REF_ENT_DEFN_ID	KEY_SERVICE_ATTR_ID	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATE_DATE	OBJECT_VERSION_NUMBER
com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails	(null)	(null)	(null)	SETUP	30-MAY-17	SETUP	30-MAY-17	1
com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails	(null)	(null)	(null)	SETUP	30-MAY-17	SETUP	30-MAY-17	1

OBJECT_STATUS	REF_FIELD_DEFN_ID
A	com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse.PreExecutionResponse1,com.ofss.fc.app.party.dto.core.PartyInquiryResponse.PartyWarningIndicatorDTO
A	com.ofss.fc.framework.domain.adapter.AdapterResponsibilityChainResponse.PostExecutionResponse1,com.ofss.fc.app.party.dto.core.PartyInquiryResponse.DateofOnboarding

- FLX_MD_GEN_ATTR_LEGACY_B:** This table contains all the attributes for metadata, while making entry for attribute which has to enriched you can append responseenrich in cod_constraint_attr_id so you can differentiate between actual service attributes entry and response enriched entry.

Sample entry for PartyInquireResponse fields with their respective data type.

Figure 3–32 FLX_MD_GEN_ATTR_LEGACY_B

COD_CONSTRAINT_ATTR_ID	TXT_CONSTRAINT_ATTR_NAME	DATA_TYPE	CREATED_BY	CREATION_DATE	LAST_UPDATED_BY	LAST_UPDATE_DATE	OBJECT_VERSION_N...	OBJECT_STATUS
294	PartyType	com.ofss.fc.enumeration.party.PartyType	SETUP	30-MAY-17 12.46.38.000000000	EM (null)	(null)		1A
295	Blacklisted	java.lang.Boolean	SETUP	30-MAY-17 12.46.38.000000000	EM (null)	(null)		1A
296	IndividualDTO	com.ofss.fc.app.party.dto.individual.IndividualDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
297	OrganizationDTO	com.ofss.fc.app.party.dto.organization.OrganizationDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
298	BankingEntityDTO	com.ofss.fc.app.party.dto.organization.BankingEntityDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
299	TrustDTO	com.ofss.fc.app.party.dto.trust.TrustDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
300	Comments	com.ofss.fc.app.party.dto.core.CommentDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
301	RESP_ENRICH.DateofOnboar...	com.ofss.fc.datatype.Date	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
302	RoleSpecificDetail	com.ofss.fc.app.party.dto.core.RoleSpecificDetailDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
303	RESP_ENRICH.PartyWarning...	com.ofss.fc.app.party.dto.core.PartyWarningIndicatorDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
304	EmploymentHistoryDTO	com.ofss.fc.app.party.dto.individual.EmploymentHistoryDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A
305	ApplicantMarketingConsen...	com.ofss.fc.app.party.dto.core.MarketingConsentDTO	SETUP	30-MAY-17 02.02.29.000000000	EM (null)	(null)		1A

3.4.3 Custom Application Service

- In SPI method createBusinessServiceInstance is used to get customized instance of application service.
- Custom Application Service name is maintained 'CustomEntities' preferences.
- For example com.ofss.fc.appx.party.service.core.PartyApplicationServiceSpi.fetchPartyDetails can call com.ofss.cz.hdfc.app.party.service.core.PartyApplicationService.fetchPartyDetails.

Figure 3–33 Custom Application Service

```
private IPartyApplicationService createBusinessServiceInstance(String businessServiceName) throws InvocationTargetException {
    IPartyApplicationService partyApplicationService = null;
    Object customApplicationServiceInstance = getCustomBusinessServiceInstance(businessServiceName);
    if (customApplicationServiceInstance != null) {
        partyApplicationService = (IPartyApplicationService) customApplicationServiceInstance;
    } else {
        partyApplicationService = new PartyApplicationService();
    }
    return partyApplicationService;
}
```


4 OBP Proxy Extension

OBP Proxy Extension functionality is driven by a preference named "ProxyFacadeExtension" whose key-value properties are provided by a java class - **com.ofss.fc.common.ProxyFacadeExtensionConfig**. This java class will have fully qualified name (replacing '.' With '_') of a proxy as a variable name and fully qualified name of a target proxy as a variable value.

For example,

```
public final String com_ofss_fc_xyz_ProductProxyFacade =
    "com.ofss.fc.osb.xyz.ProductProxyFacade"; // notice usage of '_' in
    place of '.' in variable name.
```

Sample Existing Code:

```
public TransactionStatus addReferenceObject(SessionContext
    sessionContext, ReferenceObjectDTO referenceObjectDTO) throws
    FatalException, ServiceException {
    if (logger.isLoggable(Level.FINE)) {
        logger.log(Level.FINE, THIS_COMPONENT_NAME + " addReferenceObject()
            Entry");
        logger.log(Level.FINE, logAppServiceMessage(sessionContext));
        logger.log(Level.FINE, logAppServiceMessage(referenceObjectDTO));
    }
    TransactionStatus returnObj = null;
    try {
        this.overrideProtocol
            ("ReferenceObjectApplicationServiceProxy.addReferenceObject");
        this.populateDictionaryData(referenceObjectDTO);
        if ("JSON".equals(protocol) && "APP".equals(hostApplicationLayer))
        {

            com.ofss.fc.app.me.service.referencedata.service.json.client.Refer
                enceObjectApplicationServiceJSONClient jsonStub = new
                com.ofss.fc.app.me.service.referencedata.service.json.client.Refer
                enceObjectApplicationServiceJSONClient(jsonServiceUrl);
            returnObj = jsonStub.addReferenceObject(sessionContext,
                referenceObjectDTO);
        } else if ("LOCAL".equals(protocol) && "APP".equals
            (hostApplicationLayer)) {
            try {
                Object[] args = new Object[] { sessionContext, referenceObjectDTO
                };
                String stringToCompleteClassName =
                    "com.ofss.fc.app.me.service.referencedata.ReferenceObjectApplicati
                    onService";
                Object obj = ReflectionHelper.getInstance().getClass
                    (stringToCompleteClassName).newInstance();
```

```

returnObj = (TransactionStatus) ReflectionHelper.getInstance
().invokeMethod(obj, "addReferenceObject", args);
} catch (Exception e) {
throw new ServiceException(SERVICE_NOT_AVAILABLE, e);
}
} else {
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, "No valid protocol
and hostApplicationLayer combination found");
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, SERVICE_NOT_
AVAILABLE);
}
this.populateTransactionStatus(returnObj);
} catch (IOException e) {
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, e);
throw new ServiceException(SERVICE_NOT_AVAILABLE, e);
}
if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, THIS_COMPONENT_NAME + " addReferenceObject()
Exit");
logger.log(Level.FINE, logAppServiceMessage(returnObj));
}
return returnObj;
}

```

Sample Existing Code will be changed to:

```

public TransactionStatus addReferenceObject(SessionContext
sessionContext, ReferenceObjectDTO referenceObjectDTO) throws
FatalException, ServiceException {

if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, THIS_COMPONENT_NAME + " addReferenceObject()
Entry");
logger.log(Level.FINE, logAppServiceMessage(sessionContext));
logger.log(Level.FINE, logAppServiceMessage(referenceObjectDTO));
}
TransactionStatus returnObj = null;
try {
if (isProxyExtended(this)) {
Serializable overriddenResponse = invokeExtendedProxy(this,
sessionContext, "addReferenceObject", referenceObjectDTO);
if (overriddenResponse != null) {
if (overriddenResponse instanceof TransactionStatus) {
return (TransactionStatus) overriddenResponse;
} else {
logger.log(Level.SEVERE,
THIS_COMPONENT_NAME,

```

```

"Invalid response returned from the overridden proxy. Response
expected is an instance of TransactionStatus.");
throw new ServiceException(BranchErrorConstants.FC_OVR_RESP_INV);
}
} else {
logger.log(Level.SEVERE,
THIS_COMPONENT_NAME,
"Null response returned from the overridden proxy. Response
expected is an instance of TransactionStatus.");
throw new ServiceException(BranchErrorConstants.FC_OVR_RESP_NULL);
}
} else {
this.populateDictionaryData(referenceObjectDTO);
if ("JSON".equals(protocol) && "APP".equals(hostApplicationLayer))
{

com.ofss.fc.app.me.service.referencedata.service.json.client.Refer
enceObjectApplicationServiceJSONClient jsonStub = new
com.ofss.fc.app.me.service.referencedata.service.json.client.Refer
enceObjectApplicationServiceJSONClient(jsonServiceUrl);
returnObj = jsonStub.addReferenceObject(sessionContext,
referenceObjectDTO);
} else if ("LOCAL".equals(protocol) && "APP".equals
(hostApplicationLayer)) {
try {
Object[] args = new Object[] { sessionContext, referenceObjectDTO
};
String stringToCompleteClassName =
"com.ofss.fc.app.me.service.referencedata.ReferenceObjectApplicati
onService";
Object obj = ReflectionHelper.getInstance().getClass
(stringToCompleteClassName).newInstance();
returnObj = (TransactionStatus) ReflectionHelper.getInstance
().invokeMethod(obj, "addReferenceObject", args);
} catch (Exception e) {
throw new ServiceException(SERVICE_NOT_AVAILABLE, e);
}
} else {
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, "No valid protocol
and hostApplicationLayer combination found");
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, SERVICE_NOT_
AVAILABLE);
}
this.populateTransactionStatus(returnObj);
}
} catch (Throwable e) {
logger.log(Level.SEVERE, THIS_COMPONENT_NAME, e);
throw new ServiceException(SERVICE_NOT_AVAILABLE, e);
}
}

```

```
if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, THIS_COMPONENT_NAME + " addReferenceObject()
Exit");
logger.log(Level.FINE, logAppServiceMessage(returnObj));
}
return returnObj;
}
```

5 OBP Application Adapters

An adapter, by definition, helps the interfacing or integrating components to 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 to adapt to any incompatibility of the invoked interface to 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.

In OBP, adapters are used for the above purposes as well as to achieve cleaner build time separation of different functional product processor modules. Hence, when Loan Module needs to invoke services of Party Module or Demand Deposit module then an adapter class owned by the Loans module will be used to ensure that functions such as defaulting of values, mocking of an interface, and so on, are implemented in the adapter layer thereby relieving the core module functionality from getting corrupted.

The design of the adapter layer is based on the Separated Interface design pattern and the access mechanism of the adapters by modules is implemented using an Abstract Factory design pattern. The adapter implementation is explained in the sections below as it exists in OBP.

5.1 Adapter Implementation Architecture

This section provides a detailed explanation of the adapter implementation architecture.

5.1.1 Package Diagram

The components of adapter implementation in OBP are structurally placed in separate projects to enable OBP to achieve build time independence between functional modules of the product. The way this is achieved is detailed in the table below and depicted with package diagram, class diagrams and an example usage mechanism.

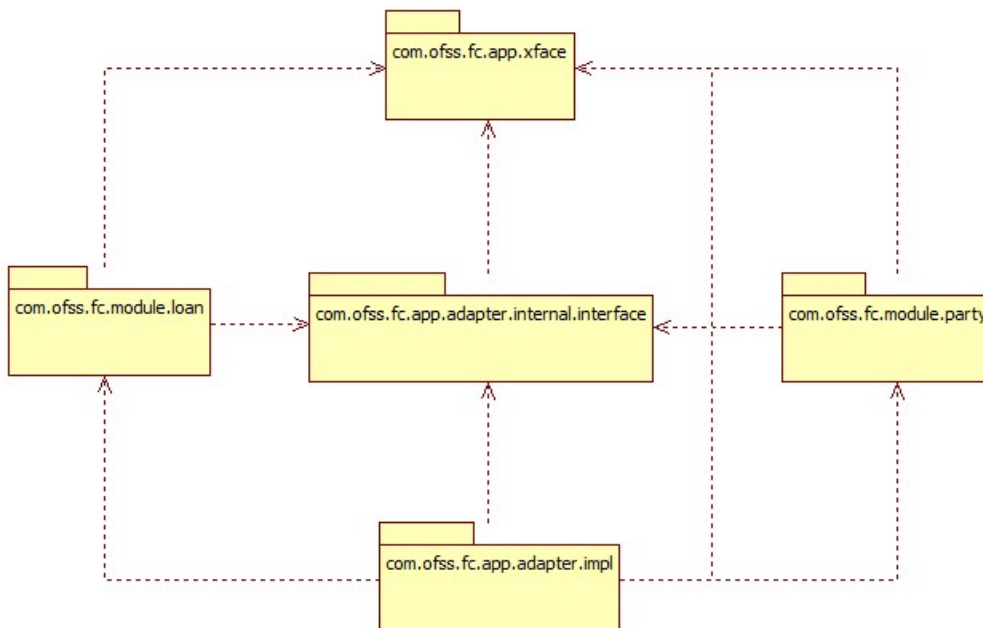
Table 5–1 Components of Adapter Implementation

Sr.	Project Name	Description	Example
1	com.ofss.fc.app.xface	DTO project. Holds all DTOs that are used in the module application services request and response DTOs.	
2	com.ofss.fc.app.adapter.internal.interface	Package contains adapter interfaces for	com.ofss.fc.app.adapter.ep.IEventProcessingAdapter Abstract Factory com.ofss.fc.app.adapter.AdapterFactory

Sr.	Project Name	Description	Example
		all modules and the abstract factory implementation (i.e. factory of adapter factories).	
3	com.ofss.fc.app.adapter.impl	This project has the implementation of adapter interfaces and corresponding adapter factories.	com.ofss.fc.app.adapter.ep. impl.EventProcessingAdapter com.ofss.fc.app.adapter.ep. impl.EventProcessingAdapterFactory

Hence, if Loans module (that is, com.ofss.fc.module.loan) and Party module (that is, com.ofss.fc.module.party) are any two modules that need to communicate, the package dependency diagram is depicted below:

Figure 5–1 Package Diagram



The dependencies among the packages as shown in the diagram are:

- Package `com.ofss.fc.app.adapter.internal.interface` only depends on DTO's.
- Any module package depends on the Adapter interfaces and DTO's to communicate with another module.
- Package `com.ofss.fc.app.adapter.impl` depends on all the packages.

In this manner, the loans module is developed into a functional module which is structurally modular and independent in terms of development and build from the party module and vice versa. Same is true for all modules developed in OBP.

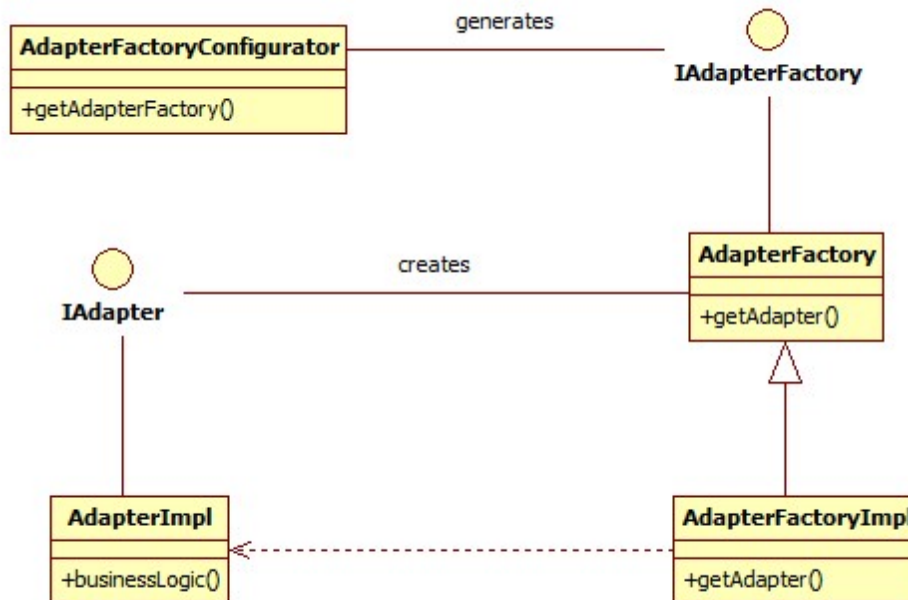
5.1.2 Adapter Mechanism Class Diagram

An Application Service in calling module calls the `getAdapterFactory()` method of class `AdapterFactoryConfigurator` which returns an instance of an implementation of the abstract class `AdapterFactory`. The class of instance is decided by the string parameter passed to the method.

The `getAdapter()` method in the `AdapterFactory` returns an adapter instance. The class of instance is decided by the string parameter passed to the method.

The Application Service then uses this adapter instance to access any data from an application service within another module.

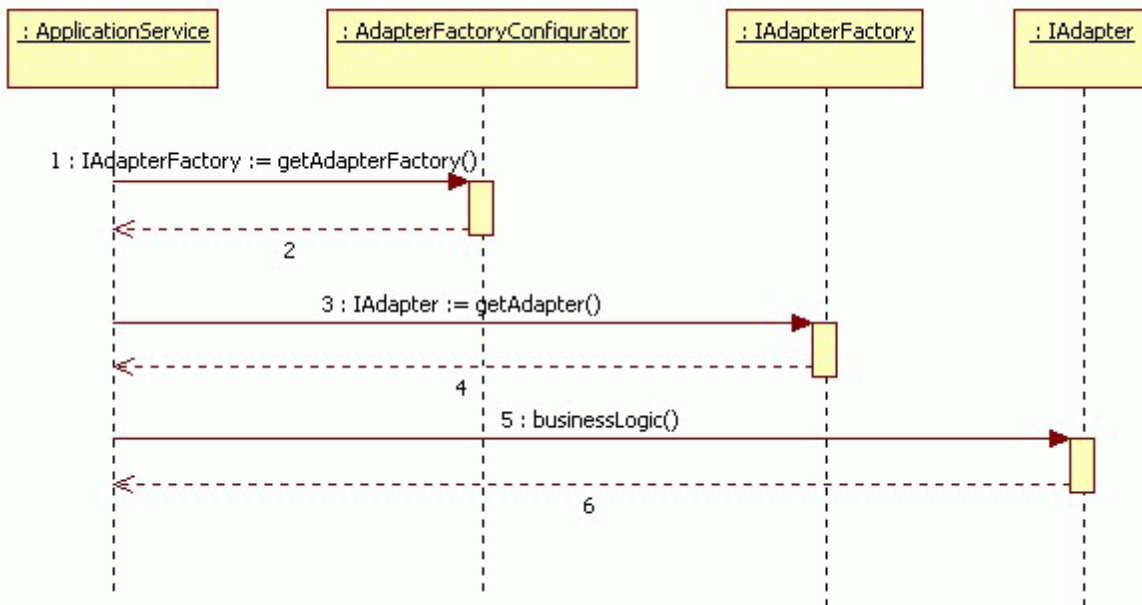
Figure 5–2 Adapter Mechanism Class Diagram



5.1.3 Adapter Mechanism Sequence Diagram

A method in an application service gets an instance of a desired adapter factory by calling `getAdapterFactory()` method of `AdapterFactoryConfigurator` class. The instance of the adapter factory obtained is used to call `getAdapter()` method to get an instance of the adapter. This adapter instance has all the methods to communicate to the service in another module.

Figure 5–3 Adapter Mechanism Sequence Diagram



5.2 Examples of Adapter Implementation

This section provides multiple adapter usage scenarios with code snippets. The section also has examples describing the steps for implementing custom adapters and their factory implementation. The same mechanism applies to all adapters which are provided by different modules in OBP. The adapter factory additionally supports mocking of the adapter. OBP depends on the DI feature function supported by Jmock to enable mocking of adapters.

The custom adapter, adapter factory and corresponding constants are depicted in code samples below:

5.2.1 Example 1 – EventProcessingAdapter

Code snippet to invoke a method `processActivityEvents()` in alerts module from a different module:

```

... Constants definition ...
public static final String EVENT_PROCESSING = "EVENT_PROCESSING";
public static final String MODULE_TO_ACTIVITY =
"ModuleToActivityAdapter";
... Adapter usage ...
com.ofss.fc.app.adapter.IAdapterFactory adapterFactory =
AdapterFactoryConfigurator.getInstance().getAdapterFactory
(ModuleConstant.EVENT_PROCESSING);
IEventProcessingAdapter adapter = (IEventProcessingAdapter)
adapterFactory.getAdapter (EventProcessingAdapterConstant.MODULE_
TO_ACTIVITY);
adapter.processActivityEvents();
  
```

The parameters passed in the **getAdapterFactory()** and **getAdapter()** methods are String constants denoting instance of which class has to be returned. These string values are maintained as constants. In the example given below, the string constant is created in a constants file (in this example, it the constants file is `ModuleConstant`).

```
public static final String EVENT_PROCESSING = "EVENT_PROCESSING";
```

An entry is made in `AdapterFactories.properties` corresponding to the string constant. This entry specifies the adapter factory class corresponding to the above constant which should be instantiated and returned. The adapter factory has the intelligence of all adapters along with adapter methods which are mocked as and where required.

```
EVENT_
PROCESSING=com.ofss.fc.app.adapter.impl.ep.EventProcessingAdapterF
actory
```

While implementing the adapter factory, developers can choose to have a separate factory specific constants on the basis of which to manage multiple adapters from the same factory. Alternatively, developers can choose to create an adapter factory each for an adapter and its interface. The constants form the basis for instantiating and returning of respective adapters by the factory.

The respective adapter constant and corresponding usage in the **getAdapter** method of the adapter factory class is shown in a sample below.

```
... Adapter Factory Method ...
public IEventProcessingAdapter getAdapter(String adapter,
NameValuePair[] nameValues) {
EventProcessingAdapter eventProcessingAdapter = null;
If (adapter.equalsIgnoreCase(EventProcessingAdapterConstant.MODULE_
TO_ACTIVITY)) {
eventProcessingAdapter = new EventProcessingAdapter();
}
return eventProcessingAdapter;
}
```

The adapter implementation (that is, *EventProcessingAdapter*) can have implementation of the methods defined in the adapter interface it implements. This implementation is typically delegated calls to services of the module which is invoked by the consuming module. For example, the *EventProcessingAdapter* can implement the method *processActivityEvents()*.

```
public void processActivityEvents(ApplicationContext
applicationContext, HashMap<String, String> activityMap) throws
FatalException {
EventProcessorApplicationService eventApplicationService =
new EventProcessorApplicationService();
eventApplicationService.processActivityEvents
(AdapterContextHelper.fetchSessionContext(), key, activityDataId);
}
```

5.2.2 Example 2 – DispatchAdapter

Similar to the implementation of *EventProcessingAdapter*, an adapter implementation is provided by product for dispatch of an SMS alert. This adapter will always get customized during implementation depending on the SMS gateway used by the customer at their end.

The code snippet to invoke a method *dispatchSMS()* in alerts module by using the adapter interface is depicted below.

```

... Constants definition ...
public static final String EVENT_PROCESSING_DISPATCH = "EVENT_
PROCESSING_DISPATCH";
public static final String EP_TO_DISPATCH = "EpToDispatchAdapter";

... Adapter usage ...
com.ofss.fc.app.adapter.IAdapterFactory adapterFactory =
AdapterFactoryConfigurator.getInstance().getAdapterFactory
(ModuleConstant.EVENT_PROCESSING_DISPATCH);

adapter = (IDispatchAdapter) adapterFactory.getAdapter
(EventProcessingAdapterConstant.EP_TO_DISPATCH);
adapter.dispatchSMS();

```

An entry in *AdapterFactories.properties* corresponding to the *DispatchAdapterFactory* would look as below. This entry specifies the adapter factory class corresponding to the above constant which should be instantiated and returned.

```

EVENT_PROCESSING_
DISPATCH=com.ofss.fc.app.adapter.impl.ep.DispatchAdapterFactory

```

The adapter *DispatchAdapter* is used in the alerts module to dispatch a message to an SMS destination endpoint. It has a method called *dispatchSMS(...)* and the default implementation is currently to write the SMS text generated as part of alert processing into a file called SMS.txt.

```

public boolean dispatchSMS(String recipientId, String
dispatchMessage) throws FatalException {
return writeToFile(recipientId, dispatchMessage);
}

```

The customization developer can override this method by supplying a customized implementation of the adapter. Such custom implementation of the *dispatchSMS(...)* method invokes the APIs provided by the gateway client. A sample implementation which overrides the default implementation of *dispatchSMS* could look like the one below:

```

public boolean dispatchSMS(String recipientId, String
dispatchMessage) throws FatalException {
NewGatewayAPI newGateway = new NewGatewayAPI();
newGateway.sendMessage(recipientId, dispatchMessage);
}

```

5.2.3 Example 3 - Adapter Implementation Using Groovy

Groovy adapter implementation acts as a wrapper on the product. Adapter implementation in OBP is used to make service call from one module to another module.

Existing product adapter will be overridden by the new custom made adapter for Groovy. This new Groovy adapter would contain groovy implementation methods which might call groovy files internally to perform desired functionality.

For example, for CreditCardAdapter, the following steps would have to be followed for implementation of a custom Groovy Adapter.

Develop a *CustomGroovyAdapter* and *CustomGroovyAdapterFactory*. As a guideline, the custom adapter should extend the existing adapter and override the methods which need to be replaced with the new functionality. Given below are examples of customizing the adapters which are detailed above.

The respective adapter constant and corresponding usage in the getAdapter method of the adapter factory class is shown in a sample below.

Figure 5–4 Adapter Implementation Using Groovy

```

import com.ofss.fc.app.adapter.AdapterFactory;

/**
 * This class represents GroovyCreditCardAdapterFactory.This factory class creates a GroovyCreditCardAdapter Object.
 *
 * @author sambedita.nayak@oracle.com
 */
class GroovyCreditCardAdapterFactory extends AdapterFactory{

    private static final String THIS_COMPONENT_NAME = GroovyCreditCardAdapterFactory.class.getName();
    private static GroovyCreditCardAdapterFactory instance;
    private static boolean isMockEnabled=false;

    /**
     * Return the empty new instance of the class in case it is not present<br>
     * else return the instance that is already present<br>
     * This method internally synchronizes on the class monitor to ensure that<br>
     * only one caller can create the instance at any given point in time thereby<br>
     * ensuring that this class is present as an singleton instance inside the JVM.<br>
     */
    public static GroovyCreditCardAdapterFactory getInstance() {

        synchronized (GroovyCreditCardAdapterFactory.class) {
            if (instance == null) {
                instance = new GroovyCreditCardAdapterFactory();
            }
            return instance;
        }
    }

    /**
     * This method will create the instance of GroovyCreditCardAdapter and return the same.If mocking is enabled, the method would return a mocked <br>
     * instance of the adapter.
     *
     * @param adapterClass
     * @return ICreditCardAdapter
     */
    public ICreditCardAdapter getAdapter(String adapterClass) {

        ICreditCardAdapter adapter = null;
        adapterClass = "GroovyCreditCardAdapter";
        if (adapterClass.equals('GroovyCreditCardAdapter')) {
            if (!isMockEnabled) {
                adapter = new GroovyCreditCardAdapter();
            }
            return adapter;
        } else {
            return adapter;
        }
    }
}

```

OBP gives an adapter implementation for CreditCard. The adapter implements to the interface shown below. The interface method *inquireCreditCardDetailsForCardNumber* would be overridden by the customization developer while providing the actual implementation of the desired functionality.

Figure 5–5 Credit Card Adapter Implementation Using Groovy

```

import com.ofss.fc.app.adapter.card.ICreditCardAdapter;

/**
 *This class represents GroovyCreditCardAdapter.This factory class creates a GroovyCreditCardAdapter Object.
 *This class contains the interaction services to be used from credit card (cc) module to Groovy credit card.
 *
 * @author sambenay
 */
class GroovyCreditCardAdapter implements ICreditCardAdapter{

    /**
     * This method is used to call the fetchCreditCardDetailsForCardNumber method of class GroovyCreditCardAppService in order to fetch<br>
     * the details of credit card.
     * @param SessionContext
     *      ,sessionContext, , Session Context which must contain the session context information such as user id,
     *      branch, branch code, channel etc.
     * @param String ,cardNumber
     * @return {@link com.ofss.fc.app.dto.credit.account.CreditCardDetailsDTO} This class represents DTO class which has details about the credit card
     * @throws FatalException
     */

    @Override
    public CreditCardDetailsDTO inquireCreditCardDetailsForCardNumber(SessionContext sessionContext, String cardNumber) throws FatalException {
        GroovyCreditCardAppService groovyCreditCardAppService = new GroovyCreditCardAppService();

        return groovyCreditCardAppService.fetchCreditCardDetailsForCardNumber();
    }

    public CreditCardBasicDetailsDTO inquireBasicCreditCardDetailsForCardNumber(SessionContext sessionContext, String cardNumber) throws FatalException {}
    public CreditCardDetailsDTO inquireCreditCardDetailsForCardReferenceNumber(SessionContext sessionContext, String cardReferenceNumber) throws FatalException {}
    public CreditCardBasicDetailsDTO inquireBasicCreditCardDetailsForCardReferenceNumber(SessionContext sessionContext, String cardReferenceNumber){}
    public CreditCardDetailsDTO openCreditCardAccount(SessionContext sessionContext, CreditCardAccountInputDTO creditCardAccountInputDTO) throws FatalException {}
    public CreditCardDetailsDTO performPostOpenCreditCardAccountOperation(SessionContext sessionContext, CreditCardAccountInputDTO creditCardAccountInputDTO){}
    public CreditCardDetailsDTO amendCardLimit(SessionContext sessionContext, CreditCardAccountInputDTO creditCardAccountInputDTO) throws FatalException {}
    public CreditCardDetailsDTO performPostAmendCardLimitOperation(SessionContext sessionContext, CreditCardAccountInputDTO creditCardAccountInputDTO){}
    public AddOnCardHolderDTO linkAddOnCardHolder(SessionContext sessionContext, AddOnCardHolderDTO addOnCardHolderDTO) throws FatalException {}
    public TransactionStatus updateBundleBenefits(SessionContext sessionContext, CreditCardAccountInputDTO creditCardAccountInputDTO) throws FatalException {}
    public ExternalCardDetailsDTO inquireExternalCardDetails(SessionContext sessionContext, ExternalCardInquiryRequestDTO externalCardInquiryRequestDTO){}

```

Assume the same are named as *GroovyCreditCardAdapter* which conforms to the interface of the product Credit Card adapter and *GroovyCreditCardAdapterFactory* which would return an instance of the custom adapter. As a guideline, the custom adapter should extend the existing adapter and override the methods which need to be replaced with new functionality.

The entry in *AdapterFactories.properties* corresponding to the *CreditCardAdapterFactory* would have to be modified to instantiate and return the *GroovyCreditCardAdapterFactory*. In preferences.xml, the custom *GroovyCreditCardAdapterFactory* has overridden the *AdapterFactories*.

Figure 5–6 Modify AdapterFactories.properties for GroovyCreditCardAdapterFactory

```

<Preference name="AdapterFactories" overriddenBy="GroovyCreditCardAdapterFactory"
  PreferencesProvider="com.ofss.fc.infra.config.impl.JavaConstantsConfigProvider"
  parent="" propertyFileName="com.ofss.fc.common.AdapterFactoriesConfig"
  syncTimeInterval="60000" />

```

In preferences.xml, the following has been defined for the Custom *GroovyCreditCardAdapterFactory*.

Figure 5–7 Modify Preferences.xml for GroovyCreditCardAdapterFactory

```

<Preference name="GroovyCreditCardAdapterFactory"
  PreferencesProvider="com.ofss.fc.infra.config.impl.DBBasedPropertyProvider"
  parent="jdbcpreference"
  propertyFileName="select prop_id, prop_value from flx_fw_config_all_b where category_id='GroovyCreditCardAdapterFactory'"
  syncTimeInterval="60000" />

```

Insert a record in table `flx_fw_config_all_b` to identify a Customized Domain Object in the following manner, where the fully qualified name of the groovy adapter factory can be specified.

```
Insert into FLX_FW_CONFIG_ALL_B (CATEGORY_ID, PROP_ID, PROP_
VALUE, PROP_COMMENTS, OBJECT_VERSION_NUMBER, CREATED_BY, CREATION_
DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, FACTORY_
SHIPPED_FLAG) values
('GroovyAdapterFactory', 'Groovy', 'com.ofss.fc.groovy.origination.G
roovyCreditCardAdapterFactory', 'Class for deriving
groovy', 1, 'ofssuser', SYSDATE, 'ofssuser', SYSDATE, 'A', 'Y');
```

The implementation should be packaged and added as part of custom library which the application is referring to and the groovy library will be added in the classpath of the server as below, which will be taken care by deployment team.

Figure 5–8 Add Groovy Library to Classpath

```
if [ "${PRE_CLASSPATH}" != "" ] ; then
    PRE_CLASSPATH="/scratch/app/product/fmw/obpinstall/obp/obp.thirdparty.app.domain/APP-INF/lib/groovy-all-2.3.10.jar${CLASSPATHSEP}${PRE_CLASSPATH}"
    export PRE_CLASSPATH
else
    PRE_CLASSPATH="/scratch/app/product/fmw/obpinstall/obp/obp.thirdparty.app.domain/APP-INF/lib/groovy-all-2.3.10.jar"
    export PRE_CLASSPATH
fi
```

5.3 Customizing Existing Adapters

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.

5.3.1 Custom Adapter Example 1 – DispatchAdapter

The example of DispatchAdapter is further explained here on how to customize the same. This is followed up by an example of customizing a party KYC status check adapter for further clarity and reference.

Depending on the client the SMS gateway they use and thus the corresponding interface to communicate with the gateway will differ. Also, OBP by default does not support interfacing with any SMS gateway. Hence, customization of Dispatch Adapter is essential. The following steps can be followed for implementation of a custom *DispatchAdapter*.

Develop a *CustomDispatchAdapterFactory* and *CustomDispatchAdapterFactory*. As a guideline, the custom adapter should extend the existing adapter and override the methods which need to be replaced with the new functionality. Given below are examples of customizing the adapters which are detailed above.

The custom adapter, adapter factory and corresponding constant are depicted as a sample below:

```
... CustomDispatchAdapterFactory Method ...
public IDispatchAdapter getAdapter(String adapter, NameValuePair[]
nameValues) {
    IDispatchAdapter adapter = null;
```

```
If (adapter.equalsIgnoreCase(EventProcessingAdapterConstant.EP_TO_
DISPATCH)) {
    adapter = new CustomDispatchAdapter();
}
return adapter;
}
```

The custom adapter implementation (that is, *CustomDispatchAdapter*) has the implementation of the methods defined in the adapter interface it implements. For example, the *CustomDispatchAdapter* would implement the method *dispatchSMS()* to reflect the desired functionality.

The entry in *AdapterFactories.properties* corresponding to the *DispatchAdapterFactory* can be modified to instantiate and return the *CustomDispatchAdapterFactory*. The same is shown below.

```
Original entry
EVENT_PROCESSING_
DISPATCH=com.ofss.fc.app.adapter.impl.ep.DispatchAdapterFactory
Changed entry
EVENT_PROCESSING_
DISPATCH=com.ofss.fc.app.adapter.impl.ep.CustomDispatchAdapterFact
ory
```

This changed entry specifies the custom adapter factory class corresponding to the constant which is referred to in the product. The new entry shall ensure that the *AbstractFactory* instantiates and returns an instance of *CustomDispatchAdapterFactory* instead of the original *DispatchAdapterFactory* supplied with product.

5.3.2 Custom Adapter Example 2 – PartyKYCCheckAdapter

OBP ships an adapter implementation for KYC check of a party. The adapter implements to the interface shown below. The interface method *performOnlineKYCCheck* can be overridden by the customization developer while supplying the actual implementation of the desired functionality.

```
public interface IPartyKYCCheckAdapter {
    @External(name = "KYC", info = "Perform Online KYC Check")
    public abstract KYCHistoryDTO performOnlineKYCCheck(KYCHistoryDTO
kycCheckDTO) throws FatalException;
}
```

This adapter is integrated in product and the default implementation of the KYC check returns a successful KYC check as shown below. This is depicted in the code snippets below.

Figure 5–9 Party KYC Status Check Adapter Interface

```

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

import com.ofss.fc.app.party.dto.core.KYCHistoryDTO;

/**
 * This interface represents the Party KYC status check adapter interface. Default implementation of <br>
 * this interface would return the KYCHistoryDTO with a KYC status indicating successful completion of<br>
 * the KYC for party.
 *
 * @author OBPDev
 * @version 1.0
 */
public interface IPartyKYCCheckAdapter {

    @External(name = "KYC", info = "Perform Online KYC Check")
    public abstract KYCHistoryDTO performOnlineKYCCheck(KYCHistoryDTO kycCheckDTO) throws FatalException;
}

```

Figure 5–10 Default Implementation of I Party KYC Check Adapter Interface

```

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

import java.util.logging.Level;

/**
 * Default implementation of IPartyKYCCheckAdapter interface. This would complement the adapter mocking<br>
 * done in the corresponding adapter factory.
 * @author shravank
 */
public class PartyKYCCheckAdapter implements IPartyKYCCheckAdapter {

    private static final String THIS_COMPONENT_NAME = PartyKYCCheckAdapter.class.getName();
    private Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
    private MultiEntityLogger formatter = MultiEntityLogger.getUniqueInstance();

    /**
     * This method would return the KYCHistoryDTO with a KYC status indicating successful completion of<br>
     * the KYC for party.
     */
    @Override
    public KYCHistoryDTO performOnlineKYCCheck(KYCHistoryDTO kycCheckDTO) throws FatalException {
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, formatter.formatMessage("Entered performOnlineKYCCheck."));
        }
        kycCheckDTO.getAutomaticKYCDetails().setKycStatus(KYCStatus.CONFIRMED);
        kycCheckDTO.getAutomaticKYCDetails().setKycProcessStage(KYCProcessStage.Complete);
        kycCheckDTO.getAutomaticKYCDetails().setKycComments("KYC Status maintained by Party");
        String bankCode = (String) FCRTThreadAttribute.get(FCRTThreadAttribute.USER_BANK);
        Date postingDate = new CoreService().fetchBankDates(bankCode).getCurrentDate();
        kycCheckDTO.getAutomaticKYCDetails().setKycDate(postingDate);
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, formatter.formatMessage("Exit performOnlineKYCCheck with KYCStatus:UNCONFIRMED and KYCProcessStage:Pending "));
        }
        return kycCheckDTO;
    }
}

```

```

... PartyKYCCheckAdapter performOnlineKYCCheck Method ...
public KYCHistoryDTO performOnlineKYCCheck(KYCHistoryDTO
kycCheckDTO) throws FatalException {
kycCheckDTO.getAutomaticKYCDetails().setKycStatus
(KYCStatus.CONFIRMED);
kycCheckDTO.getAutomaticKYCDetails().setKycProcessStage
(KYCProcessStage.Complete);
kycCheckDTO.getAutomaticKYCDetails().setKycComments("KYC Status
maintained by Party");
...

```

5.3 Customizing Existing Adapters

```
kycCheckDTO.getAutomaticKYCDetails().setKycDate(postingDate);
return kycCheckDTO;
}
```

In actual product implemented in production at the customer site, this is replaced with an online KYC status check against a third-party system or the appropriate KYC agency external system interface. Hence, this would always be a customization point during an implementation.

Depending on the client the KYC system uses, the corresponding interface to communicate will differ. Hence, customization of the party KYC status check adapter implementation is essential. The following steps would have to be followed for implementation of a custom *PartyKYCStatusCheckAdapter*.

The implementation of *getAdapter* method of KYC adapter factory with mocking support is given in the sample below for reference.

Figure 5–11 KYC Adapter Factory with Mocking Support

```
/**
 * This method returns instance of the KYC Adapter. If mocking is enabled, the method would return a mocked<br>
 * instance of the adapter. Mocking helps in cases where the interface undergoes a change and the same has<br>
 * to be handled with minor code changes at the adapter level.
 * @return Object Instance of the adapter
 */
public Object getAdapter(String adapter) {
    if (CommonAdapterConstants.PARTY_KYC_ADAPTER.equals(adapter)) {
        if (!isMockEnabled) {
            return new PartyKYCCheckAdapter();
        } else {
            Mockery context = new Mockery();
            final IPartyKYCCheckAdapter mockPartyKYCCheckAdapter = context.mock(IPartyKYCCheckAdapter.class);
            try {
                context.checking(new Expectations() {
                    {
                        allowing(mockPartyKYCCheckAdapter).performOnlineKYCCheck(with(any(KYCHistoryDTO.class)));
                        final KYCHistoryDTO kycCheckDTO = new KYCHistoryDTO();
                        KYCDetailsDTO automaticKYCDetails = new KYCDetailsDTO();
                        automaticKYCDetails.setKycStatus(KYCStatus.CONFIRMED);
                        automaticKYCDetails.setKycProcessStage(KYCProcessStage.Complete);
                        automaticKYCDetails.setKycComments("KYC Status maintained by Party");
                        String bankCode = (String) FCRTThreadAttribute.get(FCRTThreadAttribute.USER_BANK);
                        Date postingDate = new CoreService().fetchBankDates(bankCode).getCurrentDate();
                        automaticKYCDetails.setKycDate(postingDate);
                        kycCheckDTO.setAutomaticKYCDetails(automaticKYCDetails);
                        will(returnValue(kycCheckDTO));
                    }
                });
            } catch (Exception e) {
                throw new MockAdapterException(InfraErrorConstants.MOCK_METHOD_NOT_CONFIGD, e, PartyKYCCheckAdapterFactory.class.getName());
            }
            return mockPartyKYCCheckAdapter;
        }
    } else {
        throw new ConfigurationInitializationException(InfraErrorConstants.ADAPTER_NOT_FOUND, PartyKYCCheckAdapterFactory.class.getName());
    }
}
```

... Constants definition ...

```
public static final String PARTY_KYC_ADAPTER_FACTORY = "PARTY_KYC_
ADAPTER_FACTORY";
```

```
public static final String PARTY_KYC_ADAPTER =
"PartyKYCCheckAdapter";
```

... PartyKYCStatusCheckAdapterFactory getAdapter Method ...

```
if (AdapterConstants.PARTY_KYC_ADAPTER.equals(adapter)) {
    if (!isMockEnabled) {
        return new PartyKYCCheckAdapter();
    } else {
        // 1. Creation of Mockery Object
        Mockery context = new Mockery();
        final IPartyKYCCheckAdapter mockPartyKYCCheckAdapter = context.mock
        (IPartyKYCCheckAdapter.class);
        try {
```

```

context.checking(new Expectations() {
{
    allowing(mockPartyKYCCheckAdapter).performOnlineKYCCheck(with(any
    (KYCHistoryDTO.class)));
    final KYCHistoryDTO kycCheckDTO = new KYCHistoryDTO();
    KYCDetailsDTO automaticKYCDetails = new KYCDetailsDTO();
    automaticKYCDetails.setKycStatus(KYCStatus.CONFIRMED);
    automaticKYCDetails.setKycProcessStage(KYCProcessStage.Complete);
    automaticKYCDetails.setKycComments("KYC Status maintained by
    Party");
    String bankCode = (String) FCRThreadAttribute.get
    (FCRThreadAttribute.USER_BANK);
    Date postingDate = new CoreService().fetchBankDates
    (bankCode).getCurrentDate();
    automaticKYCDetails.setKycDate(postingDate);
    kycCheckDTO.setAutomaticKYCDetails(automaticKYCDetails);
    will(returnValue(kycCheckDTO));
}
});
} catch (Exception e) {
    throw new
    MockAdapterException(InfraErrorConstants.MOCK_METHOD_NOT_CONFIGD,
    e, PartyKYCCheckAdapterFactory.class.getName());
}
return mockPartyKYCCheckAdapter;
}
}

```

To override the default implementation of the KYC check, the customization developer has to implement a custom adapter and its corresponding adapter factory. Assume the same are named as *CustomPartyKYCStatusCheckAdapter* which conforms to the interface of the product KYC check adapter and *CustomPartyKYCStatusCheckAdapterFactory* which would return an instance of the custom adapter. As a guideline, the custom adapter should extend the existing adapter and override the methods which need to be replaced with new functionality.

Therefore, *CustomPartyKYCStatusCheckAdapter* can override and provide an actual implementation of the methods defined in the default product adapter interface. For example, the adapter implements the method *performOnlineKYCCheck()* to reflect the desired functionality.

The entry in *AdapterFactories.properties* corresponding to the *PartyKYCCheckAdapterFactory* can be modified to instantiate and return the *CustomPartyKYCCheckAdapterFactory*. The same is shown below.

```

Original entry
PARTY_KYC_ADAPTER_
FACTORY=com.ofss.fc.app.adapter.impl.party.PartyKYCCheckAdapterFac
tory
Changed entry
PARTY_KYC_ADAPTER_FACTORY=
com.ofss.fc.app.adapter.impl.party.CustomPartyKYCCheckAdapterFacto
ry

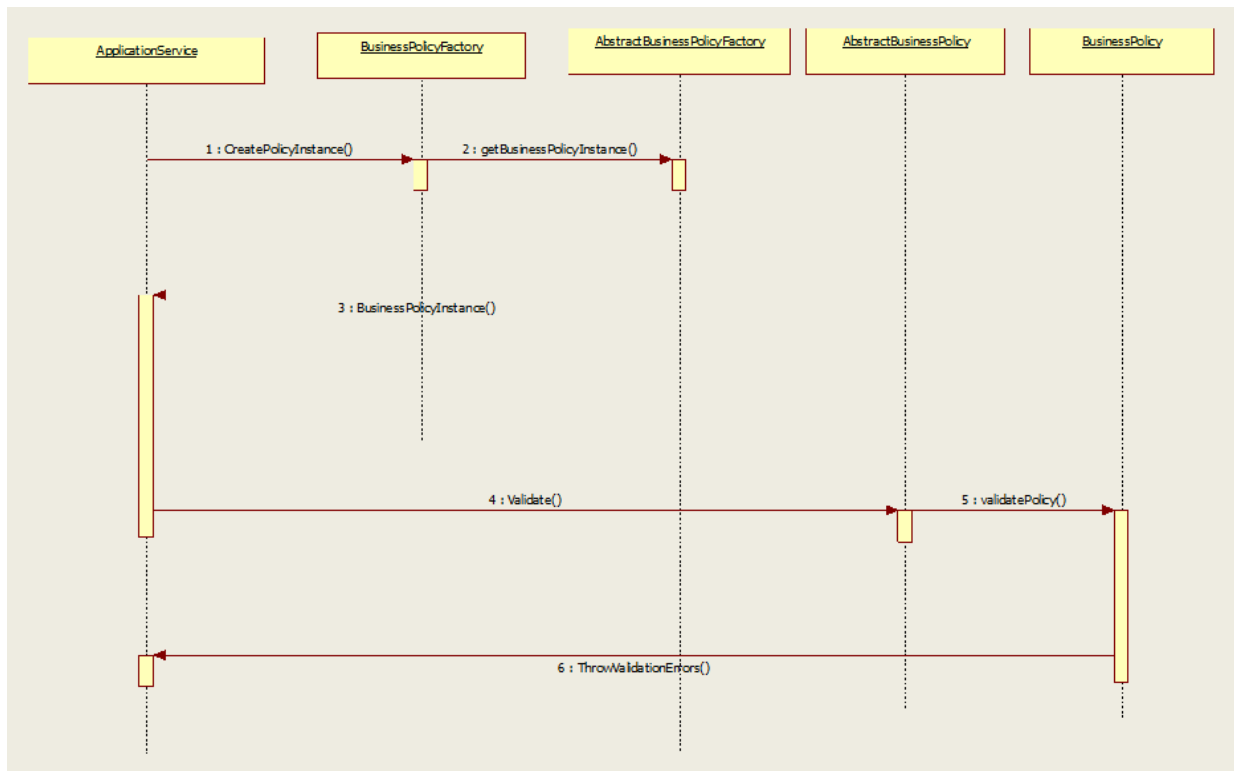
```

This changed entry specifies the custom adapter factory class corresponding to the constant which is referred to in the product. The new entry shall ensure that the *AbstractFactory* instantiates and returns an instance of *CustomPartyKYCCheckAdapterFactory* instead of the original *PartyKYCCheckAdapterFactory* supplied by the product.

6 Business Policy Extension

This chapter describes how custom business policies are implemented in OBP for overriding business validations. Business policy extensions are useful in overriding or extending the existing validations.

Figure 6–1 Business Policy Extension



The sequence diagram above shows a generic view of base implementation of business policy. Whenever business validations are required, application service invokes `createPolicyInstance()` methods in the business policy factory of the corresponding module. This business policy factory extends to `AbstractBusinessPolicyFactory` class which is maintained at framework level. `CreatePolicyInstance()` method in the business policy factory class invokes `getBusinessPolicyInstance()` method to look for any custom business policy class present in the database. If there is no custom class present, it creates an instance of base business policy class and return it to the invoking application service. Then application service invokes the `validate()` method in `AbstractBusinessPolicy` class which in turn invokes `validatePolicy()` method implemented in base business policy class. All the validation logic is written in this method and it throws validation error if any of the validation conditions fails.

6.1 Base Implementation of Business Policy

The sequence diagram, [Figure 6–1](#), shows a generic view of base implementation of business policy.

For more clarification let's take an example of `creditCardDetailsBusinessPolicy` implementation. Following are the code snippets of different key methods:

- validate() method in AbstractBusinessPolicy.java

Figure 6–2 validate() method in AbstractBusinessPolicy.java

```
public abstract class AbstractBusinessPolicy {
    /**
     * This attribute indicates validationErrors. it will hold all the validation errors in the list.
     */
    protected List<ValidationError> validationErrors;
    /**
     * This attribute represents the error code for the policy.
     */
    private String policyErrorCode;

    /**
     * This represents the validate method for the policy. This methods needs to be over-riden in every policy.
     */
    public abstract void validatePolicy();

    /**
     * This method needs to be invoked by every caller of the business policy. This method would invoke the validate
     * policy method of the policy and then throw exception in case the policy is not validated for the data provided to
     * the policy.
     */
    public final void validate() {
        validatePolicy();
        checkForError();
    }

    /**
     * This method needs to be invoked by every caller of the business policy. This method would invoke the validate
     * policy method of the policy and then throw exception in case the policy is not validated for the data provided to
     * the policy.
     */
    public final void validate(String policyErrorCode) {
        setPolicyErrorCode(policyErrorCode);
        validatePolicy();
        checkForError();
    }
}
```

- validatePolicy() in creditCardBusinessPolicy.java

Figure 6–3 validatePolicy() in creditCardBusinessPolicy.java

```
public CreditCardDetailsBusinessPolicy(CreditCardDetailsBusinessPolicyDTO creditCardBusinessPolicyDTO) {
    this.creditCardBusinessPolicyDTO = creditCardBusinessPolicyDTO;
}

/**
 * Validate Policy. Includes, <br>
 * <ul>
 * <li>Validate Card Details</li>
 * </ul>
 * @param sessionContext
 * @param creditCardDetailsDTO
 * @exception
 * <ul><li>{@link CardErrorConstants#CARD_NUMBER_NOT_PRIMARY}</li>
 * <li>{@link CardErrorConstants#CARD_NOT_ACTIVE}</li>
 * <li>{@link CardErrorConstants#BLOCKED_FOR_LIMIT_CHANGE_ADD_ON_CARD}</li>
 * <li>{@link CardErrorConstants#CARD_STATUS_CLOSED}</li>
 * </ul>
 */
@Override
public void validatePolicy() {
    if (this.creditCardBusinessPolicyDTO.getCreditCardDetailsDTO() != null) {
        validateCardDetails(this.creditCardBusinessPolicyDTO.getSessionContext(), this.creditCardBusinessPolicyDTO.getCreditCardDetailsDTO());
    }
    if (this.creditCardBusinessPolicyDTO.getCreditCardBasicDetailsDTO() != null) {
        validateCardDetails(this.creditCardBusinessPolicyDTO.getSessionContext(), this.creditCardBusinessPolicyDTO.getCreditCardBasicDetailsDTO());
    }
}
}
```

6.2 Extending Business Policy

Custom implementation of business policy is achieved by defining a preference for customBusinessPolicy in preferences.xml which represents a query to the FLX_FW_CONFIG_ALL_B table in the database. To

override a base business policy, class name of the custom business policy with the policy code is inserted into the table. As a guideline, the custom business class should extend the product base business policy, to inherit the product base implementation. Base code already handles the fetching of custom class, if any, from the table. If customization of a policy is not required then query returns null and base business policy is implemented.

6.3 Configuration

For custom business policy implementation following configuration steps are required to be followed:

1. Add a preference for custom business policy in preferences.xml.

Figure 6–4 Add a preference for custom business policy in preferences.xml

```
<Preference name="CustomBusinessPolicyPreferences"
  PreferencesProvider="com.ofss.fc.infra.config.impl.DB8basedPropertyProvider"
  parent="jdbcpreference"
  propertyFileName="select prop_id, prop_value from flx_fw_config_all_b where category_id = 'CustomBusinessPolicy'"
  syncTimeInterval="600000" />
```

2. Add an entry in FLX_FW_CONFIG_ALL_B table in database with custom class name and policy code.

```
INSERT INTO FLX_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 ('FC_CC_BP_
001','CustomBusinessPolicy','com.ofss.fc.module.originationGr
oovy.CreditCardDetailsBusinessPolicyGroovy','Y','This is
accessed from
AbstractBusinessPolicyFactory.getCustomBusinessPolicyNameTDS'
','','ofssuser',to_date('09/05/2016 11:25:30', 'dd/mm/rrrr
hh:mi:ss'),'ofssuser',to_date('09/05/2016 11:25:30',
'dd/mm/rrrr hh:mi:ss'),'A',1);
```

6.4 Extensions Using Groovy

Groovy is a lightweight, dynamically typed object-oriented programming language. It has got similarities with java and can run on jvm platform. Groovy class provides the functionalities for interacting with a java program so can be efficiently used as extensions for customization purpose.

In addition to the configurations mentioned above, add the groovy-all-2.3.10.jar in the classpath of weblogic server in setDomain.sh file, which will be done by deployment team. No other specific configuration is needed.

Following is the snippet of a groovy custom business policy class implemented for creditCardDetails validations:

Figure 6–5 Extensions using Groovy

```
package com.ofss.fc.module.originationGroovy
import com.ofss.fc.app.card.service.credit.policy.CreditCardDetailsBusinessPolicy

public class CreditCardDetailsBusinessPolicyGroovy extends AbstractBusinessPolicy{
    private static final String THIS_COMPONENT_NAME = CreditCardDetailsBusinessPolicyGroovy.class.getName()

    private CreditCardDetailsBusinessPolicyDTO creditCardBusinessPolicyDTO= null

    public CreditCardDetailsBusinessPolicyGroovy(){
        super();
    }

    public CreditCardDetailsBusinessPolicyGroovy(CreditCardDetailsBusinessPolicyDTO creditCardBusinessPolicyDTO) {
        super(creditCardBusinessPolicyDTO);
        this.creditCardBusinessPolicyDTO = creditCardBusinessPolicyDTO
    }

    @Override
    public void validatePolicy(){
        // validation logic goes here..
    }
    public CreditCardDetailsBusinessPolicyDTO getCreditCardBusinessPolicyDTO() {
        return creditCardBusinessPolicyDTO;
    }

    public void setCreditCardBusinessPolicyDTO(CreditCardDetailsBusinessPolicyDTO creditCardBusinessPolicyDTO) {
        this.creditCardBusinessPolicyDTO = creditCardBusinessPolicyDTO;
    }
}
```

7 Batch Framework Extensions

Most of the enterprise applications require bulk processing of records to perform business operations in real-time environments. These business operations include complex processing of large volumes of information that are most efficiently processed with minimal or no user interaction. Such operations would typically include time-based events (for example, month-end calculations, notices or correspondence), periodic application of complex business rules processed repetitively across very large data sets (for example, rate adjustments). All such scenarios form a part of batch processing. Thus, batch processing is used to process billions of records for enterprise applications.

There are few primary categories in OBP Batch Processes:

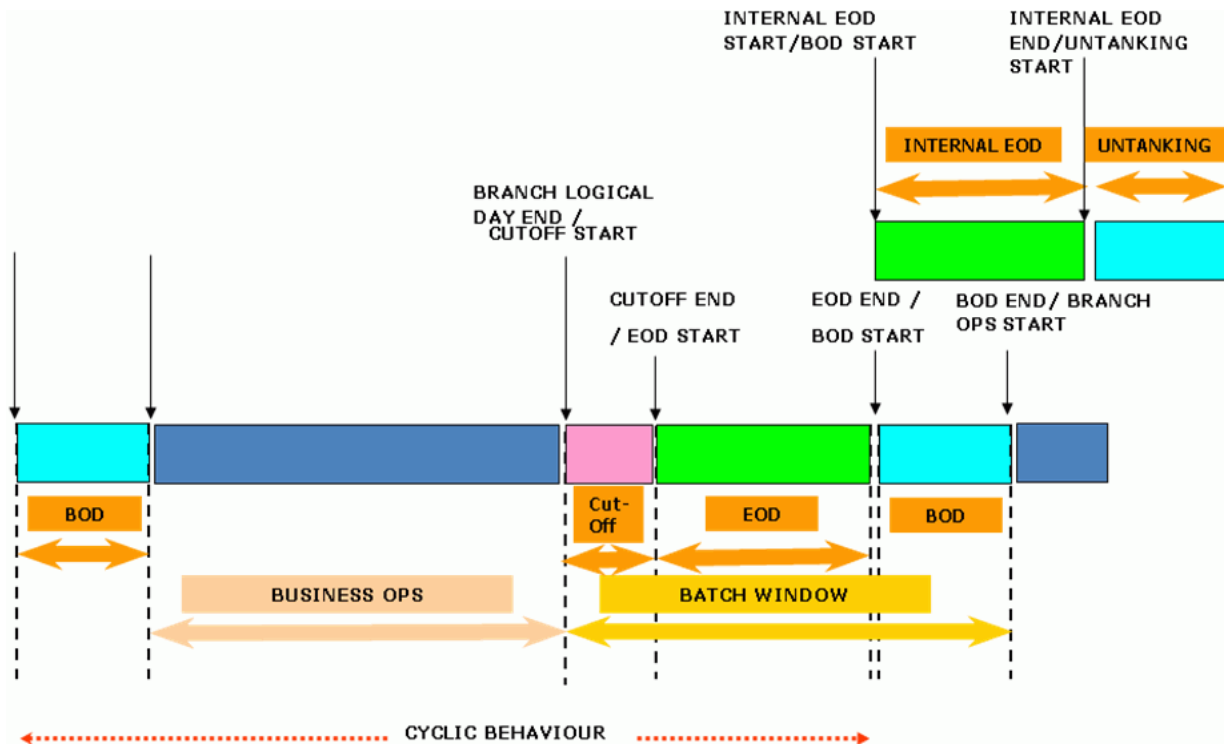
- Beginning of Day (BOD)
- Cut-off
- End of Day (EOD)
- Internal EOD
- Statement Generation
- Customer Communication

Additional categories can also be configured as per the requirement.

7.1 Typical Business Day in OBP

The following graphic describes a typical business day in OBP:

Figure 7–1 Business Day in OBP



7.2 Overview of Categories

This topic describes the categories in OBP Batch Processes.

7.2.1 Beginning of Day (BOD)

The activities for a new day of the bank / branch begin with the BOD (beginning of day). This is a batch process which executes a group of shells (programs) which are required to be performed before the normal day-to-day operations at the branch can be started. The BOD typically includes

- TD Maturity and Interest Processing
- Standing instructions execution (Based on setup)
- Loan Charging, Drawdown and Auto-Disbursement
- Value date processing of cheques (Based on the setup)
- Reports Generation

7.2.2 Cut-off

Cut-off is a process that sets the trigger for modules to start logging transactions with a new date.

It also marks cut-off for the channel transactions.

7.2.3 End of Day (EOD)

Once all the operations for the current working date of the branch are completed and all the transactions are posted the Branch EOD batch is started. This batch executes a group of shells (programs) which are required to be performed before the Business Date of the branch is changed to the next working date. It marks the end of a business day. The EOD typically includes:

- DDA Sweep-Out Instruction
- Loan Rate Change
- Term Deposit Lien Expiry and Interest Capitalization
- DDA Balance Change, Rate Change, Interest Capitalization and Settlement
- Account and Party Asset Classification
- Loan Interest Computation
- Accounting Verification

7.2.4 Internal EOD

This category performs all the activities which do not affect the customer account but are related to bank internal processing. Internal EOD typically includes:

- Interest Accrual and Compounding
- Deferred Ledger Balance Update
- Balance Period Creation
- Financial Closure

7.2.5 Statement Generation

This category performs different statement generation activities on the monthly or yearly basis. It typically includes:

- Periodic PL balance history Generation
- CASA Statement Generation
- Loan Statement Generation
- TD Statement Generation

7.2.6 Customer Communication

This category performs different communications which needs to be done with the customer on the regular basis. It typically includes:

- Regular Account Balance Notification On Specified Date

7.3 Batch Framework Architecture

This section describes the architecture of the Batch Framework.

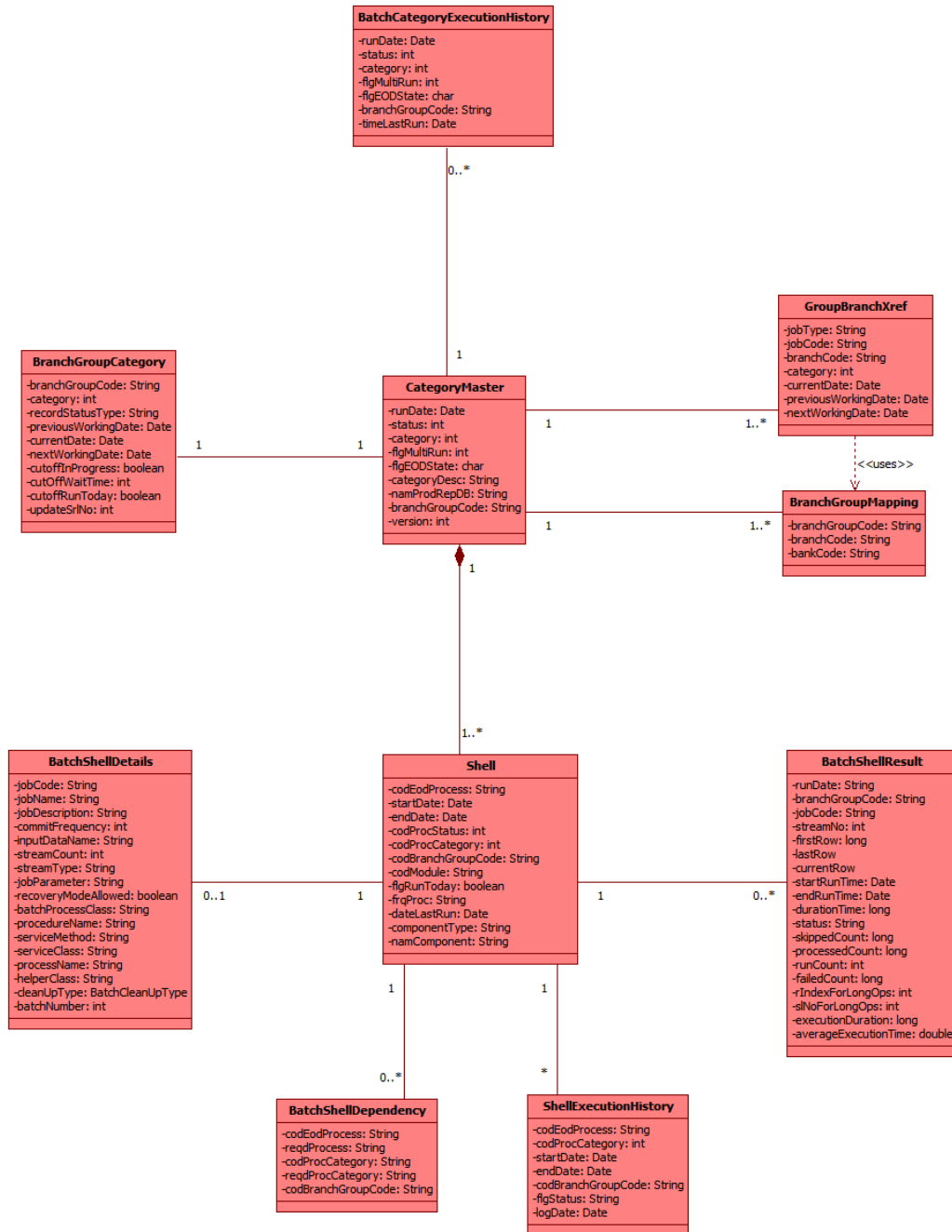
7.3.1 Static View

The static view of batch framework shows the architecturally significant classes included in the batch framework being developed. It is in line with the overall design and development guidelines and principles. This section shows the class diagrams representing the static model of the batch framework emphasizing the static structure of the system using objects, attributes and relationships.

Class Diagram

The following diagram depict details about the different classes of the code which are involved in the batch execution.

Figure 7–2 Batch Framework Architecture - Static View



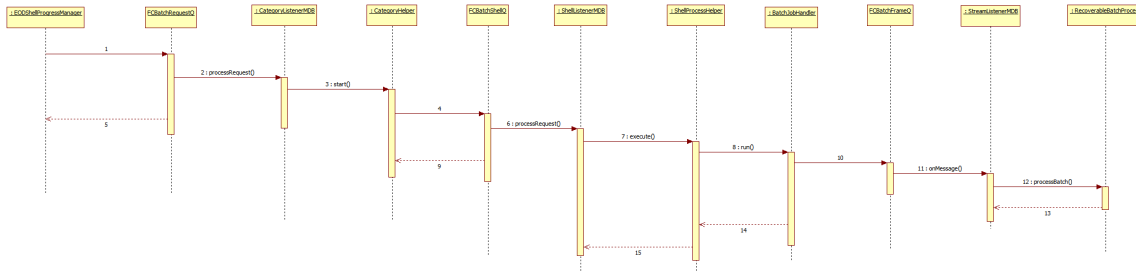
7.3.2 Dynamic View

This section emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.

Sequence Diagram

The following diagram depicts the sequence diagram for Batch framework. It provides details about the flow of control during the batch execution.

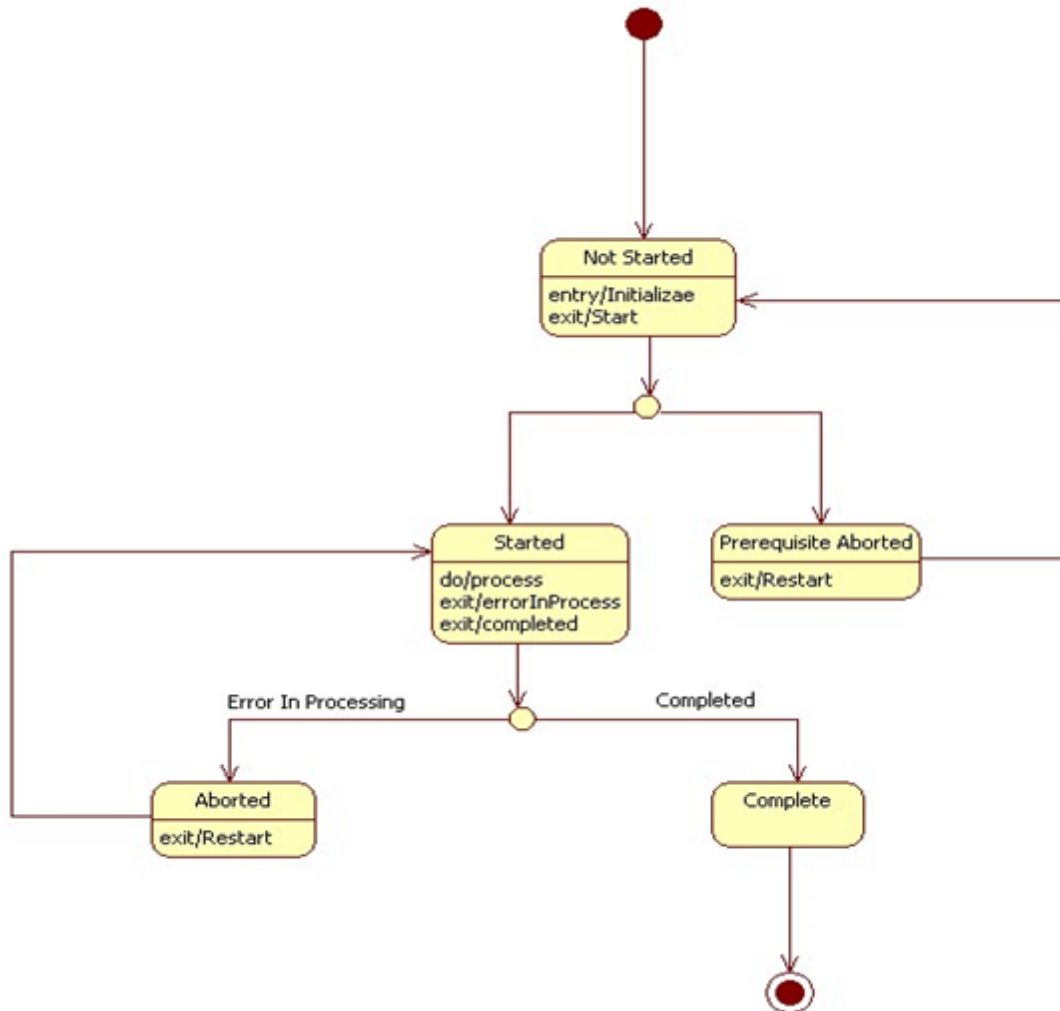
Figure 7–3 Dynamic View Sequence Diagram



State Diagram of a Shell

When the end of day batch starts, every shell is reset to Not Started. During the course of the batch, the shell status will change till the shell is completed. The transitions of shell execution are explained in the state diagram below:

Figure 7–4 State Diagram of a Shell



7.4 Batch Framework Components

This section describes the batch framework components.

7.4.1 Category Components

This section describes the category components.

CategoryListenerMDB

This MDB listens to the FCBBatchRequestQ and delegate to CategoryHelper for further processing.

CategoryHelper

This class starts or restarts a category depending upon the request received.

It will validate the input xml Request, validate the prerequisites for starting/restarting a category, get the list of shells that can be initiated on a category start/shell completion, prepare the Batch XML Message for each of the shell and send a message to FCBBatchShellQ for each Shell to be started.

It also services requests initiation of the next shell after a shell has been successfully completed.

7.4.2 Shell Components

This section describes the shell components.

ShellListenerMDB

This MDB listens on ShellRequestQ and delegate to ShellProcessHelper for processing.

ShellProcessHelper

This class validates the input request and calls appropriate batch handler to start the shell. It will call:

- BatchFrameworkShellHelper for non-report Java Bean Based Shell
- ProcedureShellHelper for Procedure based shell
- BatchReportShellBean for report shells
- BatchReportRestartShellBean for report epilogue shells

After successful completion of shell, it sends an 'InitiateNext' request to the CategoryHelper to initiate subsequent shells. If the shell is aborted, this class will mark the shell as aborted.

ShellRootHelper

This is the base class which is required for each shell processing. It Implements the IBatchHandler Interface. All the batch handlers extend this class.

This class contains the common methods which need to invoked for processing each shell for example, method to parse the request, methods used to acquire and release lock for shell, method to initiate the shell and mark the shell as complete upon successful completion.

BatchFrameworkShellHelper

This SSB extends ShellRootHelper. It is responsible for executing non report Java Bean based shells. This class will validate the process date of the request, prepare a BatchContext entity encapsulating the batch run details and call BatchJobHandler to run the shell.

BatchJobHandler

This class is responsible for putting the stream requests in queue. It will get the Batch Processes (1 Batch Process per stream) by calling BatchProcessManager and post them to the Stream Queue.

After posting the stream requests, it will start polling on the status of the streams till either all streams are completed or any one of the streams is aborted. If the streams are completed, it will return 'Success' as the status else it will return the status as 'Failure'.

BatchProcessManager

This component acts as a manager for the complete batch process. The functionalities include finding the pending batch processes and creating batch processes and returning the list of batch processes to be initiated.

If the shell is being restarted, this class will fetch the aborted batch processes, reset them and return list of reset Batch Processes to be re-initiated.

If the shell is being started, it will call `BatchJobHelper` to populate the driver table and create the batch processes and return the list of batch processes to be initiated.

BatchJobHelper

This class is responsible for populating the driver table and creating the Batch Processes.

ProcedureShellHelper

This class is used to process DB procedure based shells. This class will fetch the procedure to be executed from the 'flx_batch_job_shell_master' table and execute it.

BatchReportShellBean

This class is responsible initiating the generation of reports. It will call `ReportJobRequestor` to fetch the reports to be generated, prepare the generation request and post the requests to the Report Queue.

After the successful posting of requests, the report shell will be marked as complete. The report generation will be done in parallel to the execution of subsequent shells.

BatchReportRestartShellBean

This class is used for the epilogue shell in each category which has reports generation.

This class will check whether all the reports have been generated or not. This class will call `ReportJobRequestor` which will poll on the status of the reports till all the reports are completed or aborted.

If the aborted reports are to be regenerated, it will also post the messages to regenerate aborted reports.

7.4.3 Stream Components

This section describes the stream components.

StreamListenerMDB

This MDB is responsible for listening to the stream queue. It delegates the processing to `StreamProcessHelper`.

StreamProcessHelper

This class is responsible for starting the batch process. It calls `RecoverableBatchProcess` to start the process.

BatchProcess

This component is the base class for processing the batch process. The `StreamProcessHelper` calls this class for starting the batch process. This class will initialize the `BatchShellResult`, clear the `StaticCacheRegistry` (if the `BatchProcess` is the first `BatchProcess` of a category), process the `BatchProcess`, retry the processing of the `BatchProcess` (if the earlier failure was due to `StaleState` or `PKDuplication`) and finalize the `BatchShellResult` status depending on success/failure.

The call to process a batch request is routed through this class to the subclass.

RecoverableBatchProcess

This component processes the batch data and inherits the `BatchProcess` class. This class will process all the records in the sequence number range specified in the `BatchShellResult`.

This class will fetch the records from the driver table and process them sequentially.

To execute each record, it will call service method of the service class stored in the `BatchShellDetails` table using reflection. If there is any exception, it will call the exception handler method of the service class if the service class implements the `IBatchExceptionHandler` interface.

It will commit the transaction at the end of commit size. If all the records are executed successfully, the stream is marked as complete. If any record fails, the stream is marked as aborted.

Recoverable Batch Process can handle the failure of a record in the following ways depending upon the set up.

- Recoverable Batch Process with Recovery Mode ON: When a record fails, the previous records in the commit size will be committed and marked as success, the failed record will be marked as failed and the execution of batch process resumes from the record after the failed record. Hence in this mode all the successful records are committed and the failed records are marked as failed.
- Recoverable Batch Process with Recovery Mode OFF: In this mode, when a record fails the earlier records in the commit size are marked as skipped for the current run, the failed record is marked as failed and execution of batch process resumes from the record after the failed record.

Simple Batch Process

While executing the shell as a Simple Batch Process, the stream will be executed till the first failed record. When a record fails, the previous records in the commit size will be committed and the shell will be aborted. The records after the failed record will be skipped in the current run.

SimpleBatchProcess class is no longer used

The functionality of SimpleBatchProcess is executed through RecoverableBatchProcess by specifying the FLG_PROCESS_TYPE as "SBP" in the flx_batch_job_shell_dtls table. In the flx_batch_job_shell_dtls table:

- FLG_PROCESS_TYPE column indicates whether it is RecoverableBatchProcess (RBP) or SimpleBatchProcess (SBP).
- FLG_RECOVERY_MODE column indicates whether the Recovery mode is ON or OFF
- Simple Batch Process should have Recovery Mode as ON.

For Example:

```
Total Number of records =20;  
Commit Frequency = 10  
Failed Records = 5, 18
```

The shell will be executed as follows:

- Recoverable Batch Process with Recovery Mode ON:
 - Records 5 and 18 will be skipped and rest all the records will be committed successfully
- Recoverable Batch Process with Recovery Mode OFF:
 - Records 1 - 5 will be skipped.
 - Records 6 - 15 will be committed successfully.
 - Records 16-18 will be skipped
 - Records 19 - 20 will be committed successfully
- Simple Batch Process:
 - Records 1- 4 will be committed successfully. Rest of the records will be skipped.

7.4.4 Database Components

The Database Server houses the following components:

Table 7-1 Database Server Components

Batch Framework Tables	Description
flx_batch_job_category_master	This table contains details of each of the category per branch group. This table contains the description, last run date and the multi run flag for the category. The status, state flag and the last Run Date for each category is maintained and validated from this table during batch run.
flx_batch_job_grp_category	This table contains the previous, current and the next run date for each category per branch group.
flx_batch_job_category_depend	This table contains the category dependencies.
flx_batch_job_shell_master	This table contains details of each shell per branch group. Shell wise status, Last Run Date, process category and frequency of shell execution are the critical attributes of this table. This table will also specify whether the shell is Java Bean based shell or Procedure Based shell.
flx_batch_job_shell_depend	This table contains the dependencies of and for each shell in flx_batch_job_shell_master.
flx_batch_job_shell_dtls	This table will contain the details for executing Java Bean Based shell.
flx_<module>_drv_<action>	This driver table contains the batch execution details for the particular action.
flx_<module>_actions_b	This table defines the action type, action name and action executor which gets mapped to the driver table. The action type value is populated as action sequence in the driver table.
flx_batch_job_shell_results	This table contains execution details of each stream of each shell for each batch run per branch group.
flx_batch_job_brm_grp_mapping	This table will contain the mapping between the branch group and the branches.
flx_batch_job_grp_brm_xref	This table will contain the list of branches for which a category is being run. This table will be used only when a category is running.

7.5 Batch Configuration

The following section defines the configuration which needs to be done in order to create a new category or add a new batch shell for batch execution using the batch framework.

7.5.1 Creation of New Category

The following steps explain the creation of new category:

1. Create an entry in **flx_batch_job_category_master**:

This contains the new category name and category code along with branch group code to be defined here.

Table 7–2 FLX_BATCH_JOB_CATEGORY_MASTER

Columns	Description
DAT_EOD_RUN	This column specifies the date on which the category was last run.
COD_EOD_STATUS	This column specifies the Status of the last category run. 0 - Successful Completion ; 1 - The process was aborted after start.
COD_PROC_CATEGORY	This column specifies the category code. 1 - EOD, 2 - BOD etc. Any number of process categories can be defined.
FLG_MULTI_RUN	This column specifies whether this category can be run multiple times. 0 - Multi-Run is disabled; 1 - Multi-Run is enabled.
FLG_EOD_STATE	This column specifies the flag indicating the state of the category. R - Running ; C - Completed (i.e. not running).
TXT_CATEGORY	This column specifies the category description.
COD_BRANCH_GROUP_CODE	This column specifies the code of the Branch Group of the category.
OBJECT_VERSION_NUMBER	This column specifies the version number of the category.
NAM_PROD_REP_DB	This column mentions about the database repository.

2. Create an entry in **flx_batch_job_grp_category**:

This contains branch group code, new category code, bank code and dates relating to run the category.

Table 7–3 FLX_BATCH_JOB_GRP_CATEGORY

Columns	Description
BRANCH_GROUP_CODE	This column specifies the Branch Group Code.
COD_PROC_CATEGORY	This column specifies the procedure category.
DAT_LAST_PROCESS	This column specifies the date on which the category was last run.
DAT_PROCESS	This column specifies the current date of the category.
DAT_NEXT_PROCESS	This column specifies the next date of the category.

3. Create an entry in **flx_batch_job_category_depend** (if required):

This table will contain the category dependency. If the category does not depend on any other category, no entry in this table is required.

Table 7-4 FLX_BATCH_JOB_CATEGORY_DEPEND

Columns	Description
COD_PROC_CATEGORY	This column specifies the procedure category.
COD_BRANCH_GROUP_CODE	This column specifies the branch group code.
COD_PROC_REQD_CATEGORY	This column specifies the dependency of the required procedure category which needs to be run before this category.
COD_PROC_VALIDATION_DATE	This column defines the validation time. It can be Current/Previous.

4. Create bean or procedure based shells:

New shells (bean/procedure based, as shown in the section below) are created and linked to the category by populating the `cod_proc_category` column in those tables with the new category code created in the `flx_batch_job_category_master`.

5. Add enumeration:

In the middleware code, add an enum value in the `ProcessCategoryType.java` for the category.

6. Add category code in the property file:

In the middleware code, add the entry for the category in the `ProcessCategoryType_en.properties` file.

7. Middleware changes:

If any validations required or any dependency on other categories we can make changes in `EODShellProgressManager.java` file accordingly.

Figure 7–5 Creation of New Category

```

568         // "1" Eod Category
569         if (validateDependency(dependCategory, dateLastProcess)) {
570             EntityManager.throwFatalException(PCRJConstants.MID_EEC_EOD_NOT_DONE, null);
571         }
572     }
573     /* For Internal System EOD to start, Eod for that day should have been run */
574     if (processCategory.equals(ProcessCategoryType.INTERNAL_SYSTEM_EOD.getValue())) {
575         // "16" Internal system EOD Category
576         dependCategory = ProcessCategoryType.END_OF_DAY.getValue();
577         // "1" Eod Category
578         if (validateDependency(dependCategory, dateProcess)) {
579             EntityManager.throwFatalException(PCRJConstants.MID_EEC_CURR_EOD_NOT_DONE, null);
580         }
581     }
582     /* For UnTanking to start, Internal System EOD for that day should have been run */
583     if (processCategory.equals(ProcessCategoryType.UNTANKING.getValue())) {
584         // "20" UnTanking Category
585         dependCategory = ProcessCategoryType.INTERNAL_SYSTEM_EOD.getValue();
586         // "16" Internal system EOD Category
587         if (validateDependency(dependCategory, dateProcess)) {
588             EntityManager.throwFatalException(PCRJConstants.MID_EEC_INTERNAL_EOD_NOT_DONE, null);
589         }
590     }
591     /* For Automatic EFS to start, cutoff for that day should not have been run */
592     if (processCategory.equals(ProcessCategoryType.AUTOMATIC_EFS.getValue())) {
593         // "13" Automatic EFS
594         dependCategory = ProcessCategoryType.CUTOFF.getValue();
595         // "1" Eod Category
596         if (validateDependency(dependCategory, dateLastProcess)) {
597             EntityManager.throwFatalException(PCRJConstants.MID_EEC_CUTOFF_NOT_DONE, null);
598         }
599     }
600     /* For Automatic Mark write off to start, Automatic write off should run once before */
601     if (processCategory.equals(ProcessCategoryType.MARK_FOR_WRITE_OFF.getValue())) {
602         // "14" Mark write off
603         dependCategory = ProcessCategoryType.AUTOMATIC_WRITE_OFF.getValue();
604         // "15" Automatic write off
605         if (validateDependency(dependCategory, dateLastProcess)) {
606             EntityManager.throwFatalException(PCRJConstants.MID_ACC_MARK_WRIT_OFF, null);
607         }
608     }
609     // date to be considered
610     eodStatusConsolidatedResponse = getCategoryStatus(processCategory, jobCode, null);
611     eodStatusConsolidatedResponse.setProcessDate(dateProcess);
612     eodStatusConsolidatedResponse.setNextProcessDate(dateNextProcess);
613     eodStatusConsolidatedResponse.setStatus(status);
614     extension.postValidateProcessFlow(sessionContext, processCategory, jobCode);
615     fillTransactionStatus(status);
616 } catch (FatalException e) {

```

7.5.2 Creation of Bean Based Shell

In this batch execution (Type "B"), the business logic is provided in the service method of the java class.

1. Create an entry for Shell Parameters in the table **FLX_BATCH_JOB_SHELL_MASTER**.

Table 7-5 FLX_BATCH_JOB_SHELL_MASTER

Columns	Description
COD_EOD_PROCESS	Process code. This is the name of the program module that will be started as a process by the EOD monitor.
TXT_PROCESS	Process name to be displayed in the new UI screen.
FRQ_PROC	Frequency at which this process is to be run. 1 - Daily 2 - Weekly 3 - Fortnightly 4 - Monthly 5 - Bi-monthly 6 - Quarterly 7 - Half-yearly 8 - Yearly.
COD_PROC_STATUS	Process Status Code 0 - Complete 1 - Started 2 - Not Started 3 - Aborted 4 - Prerequisite Aborted 5 - Prerequisite Absent.
NUM_PROC_ERROR	Last error returned by this process.
FLG_RUN_TODAY	Flag indicating whether process to be run today Y/N.
COD_PROC_CATEGORY	Category code to which this shell belongs to e.g.: 1 - EOD, 2 - BOD and so on.
SERVICE_KEY	Service method to be executed.
NAM_COMPONENT	Name of the Procedure (if procedure based batch execution) or fully qualified class name of the Batch Handler (if bean based). com.ofss.fc.bh.batch.BatchFrameworkShellHelper - java bean based shell com.ofss.fc.bh.batch.BatchReportShellBean - procedure based shell for reports com.ofss.fc.bh.batch.BatchReportRestartShellBean - procedure based for report epilogue shell
TYPE_COMPONENT	This indicates whether the specified nam_component is Java class or Function. P stands for Function and B Stand for the Java Class.
NAM_DBINSTANCE	The DB instance for PROD or REP (reports).
COD_BRANCH_GROUP_CODE	Specifies the branch group code that a branch is part of.
OBJECT_VERSION_NUMBER	This column specifies the version number of the category.

2. Create an entry for Shell Details in the table **FLX_BATCH_JOB_SHELL_DTLS**.

This table contains the following parameters;

Table 7–6 FLX_BATCH_JOB_SHELL_DTLS

Columns	Description
COD_SHELL	A unique code for batch shell.
SHELL_NAME	Provide a name to batch shell.
SHELL_DESCRIPTION	Description about the batch shell.
COMMIT_FREQUENCY	Provide the commit frequency thus, after every this no of records have been processed the framework would commit those set of records
FLG_RECOVERY_MODE	Flag indicating whether recovery mode is ON or OFF. Possible values are 'Y' and 'N' only. This would be only used by Batch Processes which support recovery mode functionality but there might be batch processes which would ignore this flag (e.g.: SBP).
FLG_STREAM_TYP	Define the type of stream for the batch shell. This would have three possible values ('S' - fixed no of streams, 'R' - fixed no of rows, 'N' - no streams).
STREAM_COUNT	Define the no of streams to be created for the batch shell. This is only applicable if the StreamType is marked as 'S' or 'R'.
INPUT_DRV_NAME	Define the fully classified class name mapped to the driver table.
INPUT_SHELL_PARAM	Define the name for the shell parameter.
SERVICE_CLASS_NAME	Define the fully classified class name for the service class. This class is the starting point of the business logic execution. In case of service class name as ActionSetProcessor, the action sequence column is populated in the driver table. The execution is done corresponding to those actions.
SERVICE_METHOD_NAME	Define only method name of the service. The service method should have input parameter as driver table entity.
DRV_POP_PROC_NAME	Defines the name procedure used for driver table population. The procedure should have three input parameters branch group code, process date and next process date. Use only procedures instead of packages for data population. Because db2 will not support Package.
FLG_PROCESS_TYPE	It defines the type of process RBP or SBP. In RBP (Recoverable Batch Process) if any records fails in batch it will continue and execute rest of the records in the stream. But in case of SBP (Simple Batch process) it will abort the stream.
HELPER_CLASS_NAME	It defines the helper class for caching big queries.
BATCH_NO	Indicates the batch number for the shell.

3. Create an entry for Shell Execution Order in the table **FLX_BATCH_JOB_SHELL_DEPEND**.

Table 7-7 FLX_BATCH_JOB_SHELL_DEPEND

Columns	Description
COD_EOD_PROCESS	Process code. This is the name of the program module that will be started as a process by the EOD monitor.
COD_REQD_PROCESS	Required process code after which the framework will run process code.
COD_PROC_CATEGORY	Category of the Process Code. 1 - EOD, 2 - BOD and so on.
COD_REQD_PROC_CAT	Category of the required process code. 1 - EOD, 2 - BOD and so on.
COD_BRANCH_GROUP_CODE	This column specifies the branch group code.

If the shell is not dependent on any other shell or category then no need to keep an entry in this table.

4. Create a new driver table (the name of the table prefix by **FLX_<ModuleCode>_drv_<>**) for the Batch Shell. This is the table from which the data will be picked up for processing by the defined batch process. This table should be populated by the procedure written for population of the driver table. This table would contain the following parameters:

Table 7-8 Driver Table

Column	Description
DATE_RUN	Defines the date on which the batch job was run (process date). Value in this column needs to be populated by the driver table population procedure.
SEQ	Sequence no for the data present in the table i.e. simple sequence from 1 to maximum number of records present in table. Value in this column needs to be populated by the driver table population procedure.
PROCESS_RESULT	Define the column which would contain the result of processing of each record of this table. This column would be updated the framework with values 0,1, 2,3 or 4 indicating not processed, processing of record successful, failed with business exception , failed with framework exception or failed with SQL exception respectively.
ERROR_CODE	Define the column for error code. This would be updated the framework with the error code returned by the processing logic (currently updating the exception if any occurred).
BRANCH_CODE	Attribute specifies the branch code in which the shell is executed.
BRANCH_GROUP_CODE	Attribute specifies the branch group code that a branch is part of.
ERROR_DESC	Attribute specifies error description. This will populated by the batch framework in case the record aborts.
ACTION_SEQUENCE	In case of service action as ActionSetProcessor, the batch execution is done through the executor framework defined in the action table of the

Column	Description
(Optional)	module. The details of this action table in mentioned below. If user want to execute multiple actions, then the comma separated action_type can be defined in this column. They will be executed based on the defined priorities.
<Custom_Columns>	Define the other columns required which would contain the data required by the processing logic. Typical examples would be a column containing accountNo (if the main logic is per account) or customerId or txnRefNo etc. We can have multiple such columns which are used for per record processing for e.g. we can have two columns branchCode, accountNo.

Note

DATE_RUN, SEQ, BRANCH_GROUP_CODE columns are part of the Unique key. For example, flx_in_drv_eod_actions

5. Add the entry of the action in the actions table (FLX_<ModuleCode>_actions_b) for the shell where the service method is defined as ActionSetProcessor in the details table. This table would contain the following parameters, for example, flx_td_actions_b.

Table 7–9 Actions Table

Column	Description
ACTION_TYPE	Stores the type of action to be performed. The defined action type is populated in the action sequence column of the driver table.
ACTION_LEVEL	Stores the action level of the action as 0,1,2 based on the execution status.
PRIORITY	Stores the priority of the action.
ENTITY_STATUS	Stores the status of the entity.
ACTION_NAME	User friendly name of the action.
ACTION_DESC	Stores the description of the action.
ACTION_EXECUTOR	Stores the name of the action executor which needs to be executed when the service action is populated as ActionSetProcessor.
HOLIDAY_TREATMENT	Stores the holiday treatment of the action.
HOLIDAY_EPOCH_TYPE	Stores the holiday epoch type of the action.

6. Create a procedure (the name of the proc prefixed with **ap_<Module Code>_pop_drv**) which would populate the data in the driver table, created above. This procedure would be called at the first time when the Batch shell is run. The procedure will have only three arguments branch group code, process date and next process date. For example, ap_in_pop_drv_eod_actions.

7. Create an entity which extends **AbstractBatchData** and map this entity to the driver table. This entity name would be the one which will carry the data to be processed for batch processing. This should be provided in the InputDataName column of flx_batch_job_shell_dtls table. e.g.: InterestEODActionSetBatchData
8. Map the entity to the driver table in the hbm. The entity attributes should represent only Extra columns added in the driver table. They shouldn't be mapped to the seq, date_run, error_code, process_result columns. For example, InterestEODActionSet.hbm.xml.
9. Make additions in **batch-mappings.cfg** file for the new hbm entities created for BatchData. For example, account-mappings.cfg.xml
10. Create **Helper Class** for caching big queries in Application layer. The fully qualified class name of the helper class needs to be defined in the **HELPER_CLASS_NAME** column of the FLX_BATCH_JOB_SHELL_DTLS table. For example, InterestEODActionSetBatchDataHelper.java
11. Create a **service processor class** with the **service method** which processes the batch application. For example, ActionSetProcessor

The fully qualified class name of this service processor class need to be defined in the **SERVICE_CLASS_NAME** column of the FLX_BATCH_JOB_SHELL_DTLS table.

This processing method defined in this class should be specified in the **SERVICE_METHOD_NAME** column of the FLX_BATCH_JOB_SHELL_DTLS table.

The service method should have two input arguments - ApplicationContext and AbstractBatchData.

If the shell needs to handle the batch exceptions, the service processor class should implement IBatchHandler interface.

Note

The above steps would suffice for creating a batch shell to be run using the new Batch Framework. The Results of the shell will be present in the FLX_BATCH_JOB_SHELL_RESULTS table.

7.5.3 Creation of Procedure Based Shell

In this batch execution (Type "P"), the business logic is provided in the Stored Procedures.

1. Create an entry for **Shell Parameters** in the table **FLX_BATCH_JOB_SHELL_MASTER**. Same as described in the above section.
2. Create an entry for **Shell Execution Order** in the table **FLX_BATCH_JOB_SHELL_DEPEND**. Same as briefed in the above section if there is any dependency with any other shell.
3. Create a **function** in Database which contains the Business logic. This function will be used for batch procedure based execution and the signature of the function must have the arguments as shown in the example:

```
CREATE OR REPLACE FUNCTION ap_as_batch_verify
(var_pi_cod_brn_grp_code VARCHAR2,
var_pi_cod_user_no NUMBER,
var_pi_cod_user_id VARCHAR2,
var_pi_dat_process DATE,
var_pi_nam_bank VARCHAR2,
```

```

var_pi_cod_stream_id NUMBER,
var_pi_cod_eod_process VARCHAR2,
var_pi_cod_proc_category NUMBER) RETURN NUMBER AS
VAR_L_RETCODE NUMBER;
BEGIN
VAR_L_RETCODE := 0;
-----1. Init Restart-----
----
BEGIN
plog.error('var_pi_dat_process : ' || var_pi_dat_process);
var_l_ret_code := ap_ba_init_restart(var_pi_cod_eod_process,
var_pi_cod_brn_grp_code,
var_pi_cod_proc_category);
IF (var_l_ret_code != 0) THEN
BEGIN
IF (var_l_ret_code = -2) THEN
RETURN var_l_ret_code;
ELSE
ora_raiseerror(SQLCODE, 'Error in executing Init Restart ', 53);
RETURN 95;
END IF;
END;
END IF;
END;
-----2.Bisuness Logic-----
----
...we can write a piece of code ...or a new proc which contain all
the business logic...
-----3.Finish Restart-----
----
BEGIN
var_l_ret_code := ap_ba_finish_restart(var_pi_cod_eod_process,
var_pi_cod_brn_grp_code,
var_pi_cod_proc_category,
var_pi_dat_process);
IF (var_l_ret_code != 0) THEN
ora_raiseerror(SQLCODE, 'Error in executing Finish Restart ', 76);
RETURN 95;
END IF;
END;
-----
-----
return 0;
EXCEPTION
WHEN OTHERS THEN
ora_raiseerror(SQLCODE,
'Execution of ap_as_batch_verify Failed',
37);

```

```

RETURN 95;
END; /

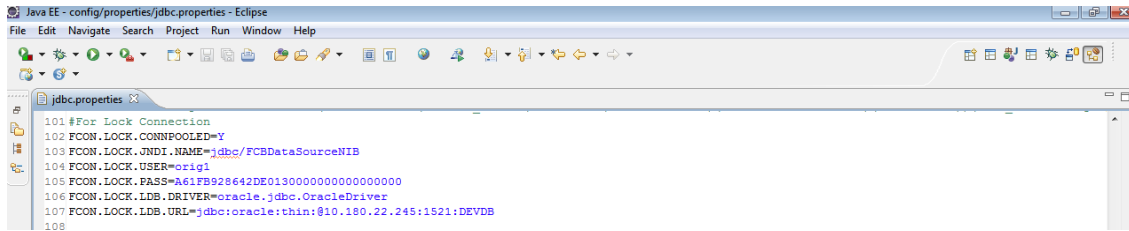
```

7.5.4 Population of Other Parameters

The following procedures describe the population of other parameters:

1. Create database credential details for Lock Connection in the jdbc.properties file

Figure 7–6 Population of Other Parameters



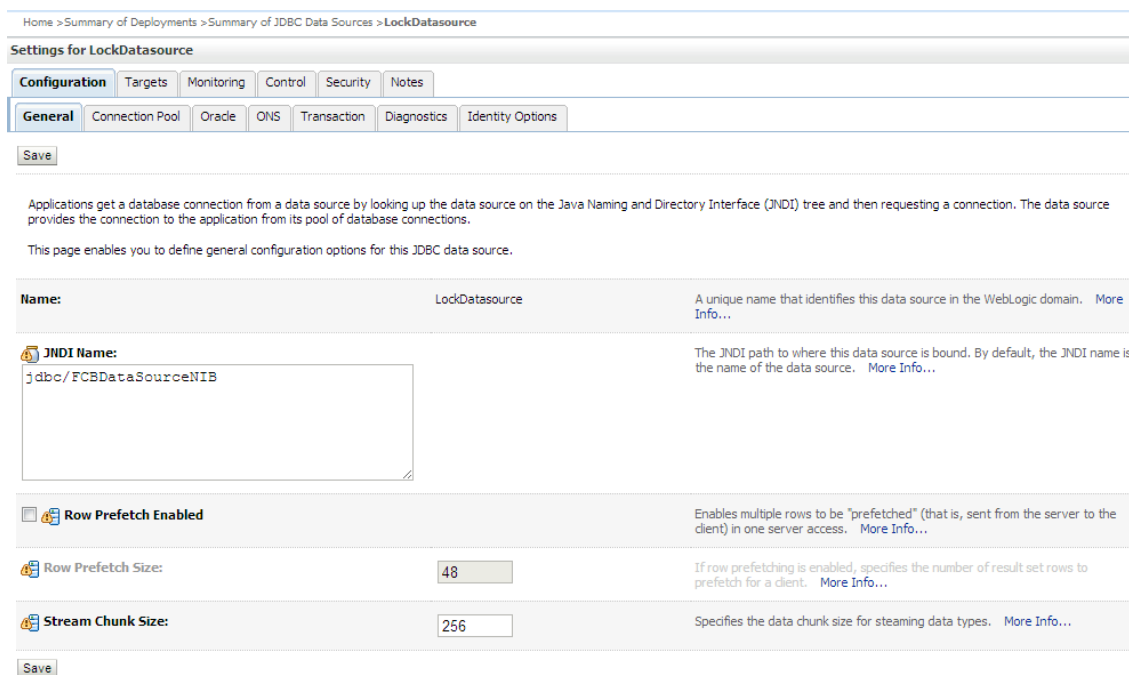
```

101 #For Lock Connection
102 FCON.LOCK.CONNPOOLED=Y
103 FCON.LOCK.JNDI.NAME=jdbc/FCBDataSourceNIB
104 FCON.LOCK.USER=orag1
105 FCON.LOCK.PASS=A61FB928642DE01300000000000000000
106 FCON.LOCK.LDB.DRIVER=oracle.jdbc.OracleDriver
107 FCON.LOCK.LDB.URL=jdbc:oracle:thin:@10.180.22.245:1521:DEVDB
108

```

2. Create datasource on the host server where the batch needs to be executed

Figure 7–7 Population of Other Parameters - General Tab



Home > Summary of Deployments > Summary of JDBC Data Sources > LockDataSource

Settings for LockDataSource

Configuration Targets Monitoring Control Security Notes

General Connection Pool Oracle ONS Transaction Diagnostics Identity Options

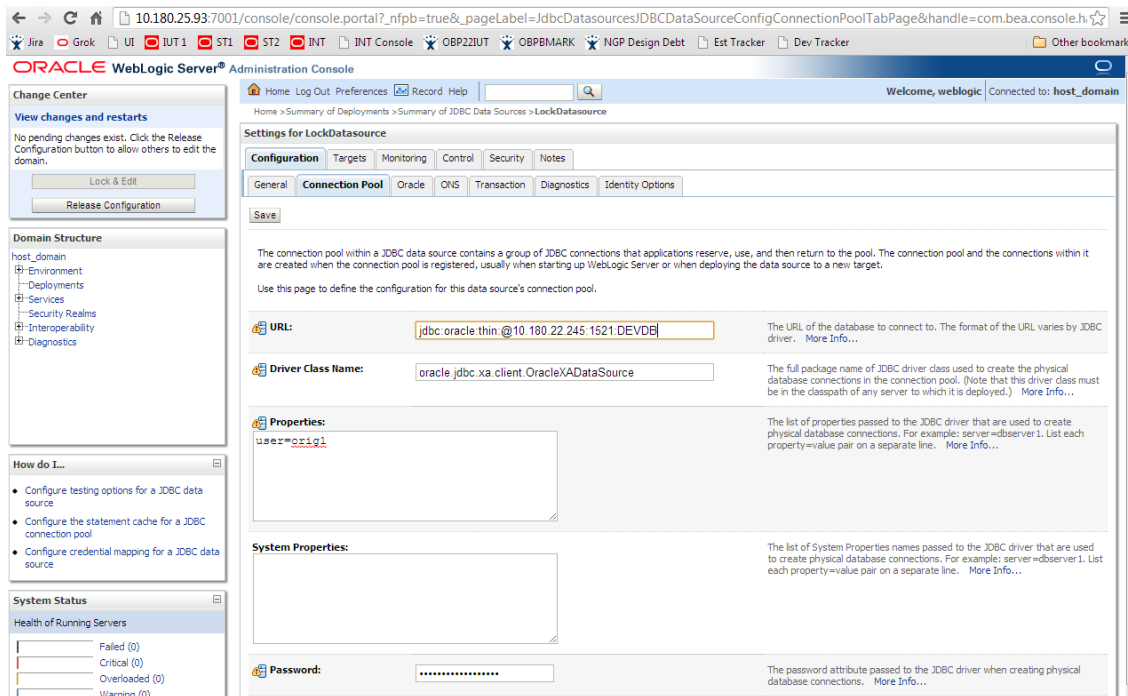
Save

Applications get a database connection from a data source by looking up the data source on the Java Naming and Directory Interface (JNDI) tree and then requesting a connection. The data source provides the connection to the application from its pool of database connections.

This page enables you to define general configuration options for this JDBC data source.

Name:	LockDataSource	A unique name that identifies this data source in the WebLogic domain. More Info...
JNDI Name:	jdbc/FCBDataSourceNIB	The JNDI path to where this data source is bound. By default, the JNDI name is the name of the data source. More Info...
<input type="checkbox"/> Row Prefetch Enabled		Enables multiple rows to be "prefetched" (that is, sent from the server to the client) in one server access. More Info...
Row Prefetch Size:	48	If row prefetching is enabled, specifies the number of result set rows to prefetch for a client. More Info...
Stream Chunk Size:	256	Specifies the data chunk size for streaming data types. More Info...

Save

Figure 7–8 Population of Other Parameters - Connection Pool

3. Enable Node Affinity for Batch Processing (Optional)

This feature can be used for Clustered Database environment. In this feature connections taken by threads are pinned to a particular database node explicitly in order to reduce Cluster Wait events.

- To enable this feature, set `IS_DB_RAC = true` in `jdbc.properties` file and specify the number of DB nodes.

Figure 7–9 Population of Other Parameters - Set IS_DB_RAC

```
41 #Denotes if the data base is running in cluster mode.
42 IS_DB_RAC=true
43 #Denotes the number of nodes in the db cluster.
44 NO_OF_DB_NODES=2
45
```

- Create a separate data for each node in the cluster. Each of these connections will have the IP of an individual node instead of the SCAN-IP. Specify the data source configuration per node in the cluster in `jdbc.properties`.

Figure 7–10 Population of Other Parameters - Specify Data

```

109 #Used in Clustered env for pinning connection to stream
110 FCON.BATCH1.CONNPOOLED=Y
111 FCON.BATCH1.JNDI.NAME=jdbc/FCBDataSource1
112 FCON.LOCK.USER=orig1
113 FCON.LOCK.PASS=A61FB928642DE01300000000000000000
114 FCON.LOCK.LDB.DRIVER=oracle.jdbc.OracleDriver
115 FCON.LOCK.LDB.URL=jdbc:oracle:thin:@10.180.22.245:1521:DEVDB
116 #Used in Clustered env for pinning connection to stream
117 FCON.BATCH2.CONNPOOLED=Y
118 FCON.BATCH2.JNDI.NAME=jdbc/FCBDataSource2
119 FCON.LOCK.USER=orig1
120 FCON.LOCK.PASS=A61FB928642DE01300000000000000000
121 FCON.LOCK.LDB.DRIVER=oracle.jdbc.OracleDriver
122 FCON.LOCK.LDB.URL=jdbc:oracle:thin:@10.180.22.245:1521:DEVDB
123

```

7.6 Batch Execution

The user can execute the batch process from the task code EOD10 screen. User needs to select the process category, job type and job code. The corresponding shells get populated in the table below which can be started by clicking on the start/restart button.

User can also monitor the performance by clicking on the Refresh button available in the Category Details section. The execution of the batch takes care of shell dependencies and the dependent shells are run once their dependencies are executed.

Figure 7–11 Batch Execution

The screenshot displays the Oracle Banking Platform interface for the EOD10 End of Day process. The 'Category Details' section shows the following information:

- Process Category: Forward Health Check
- Job Type: GROUP
- Job Code: BRN_GRP_1
- Category Start Time: 07-Jan-2013 14:57:54
- Category Status: Completed
- Process Date: 05-Jan-2014
- Next Process Date: 06-Jan-2014
- Category End Time: 07-Jan-2013 14:58:14
- Last Refreshed Time: 07-Jan-2013 15:02:01

The 'Process' section contains a table with the following data:

Name Of Shell	State	Health	Module Code	Streams	Number Of Streams	Start Time	End Time	Execution Time	Duration	Wait Time	# of Aborts
Health Checkup Shell	complete	✓	LN	false	1	2013-01-07 14:57:57	2013-01-07 14:58:00		00:00:20		

8 Uploaded File Data Processing

In Banks, there are multiple times when the bulk load of data is available in the form of files which needs to be uploaded and processed in the banking application. An example for the same can be salary credit processing. The salary credit data is provided by the organizations in the form of files where employer account needs to be debited and the multiple accounts of the employees needs to be credited for the provided data in the files.

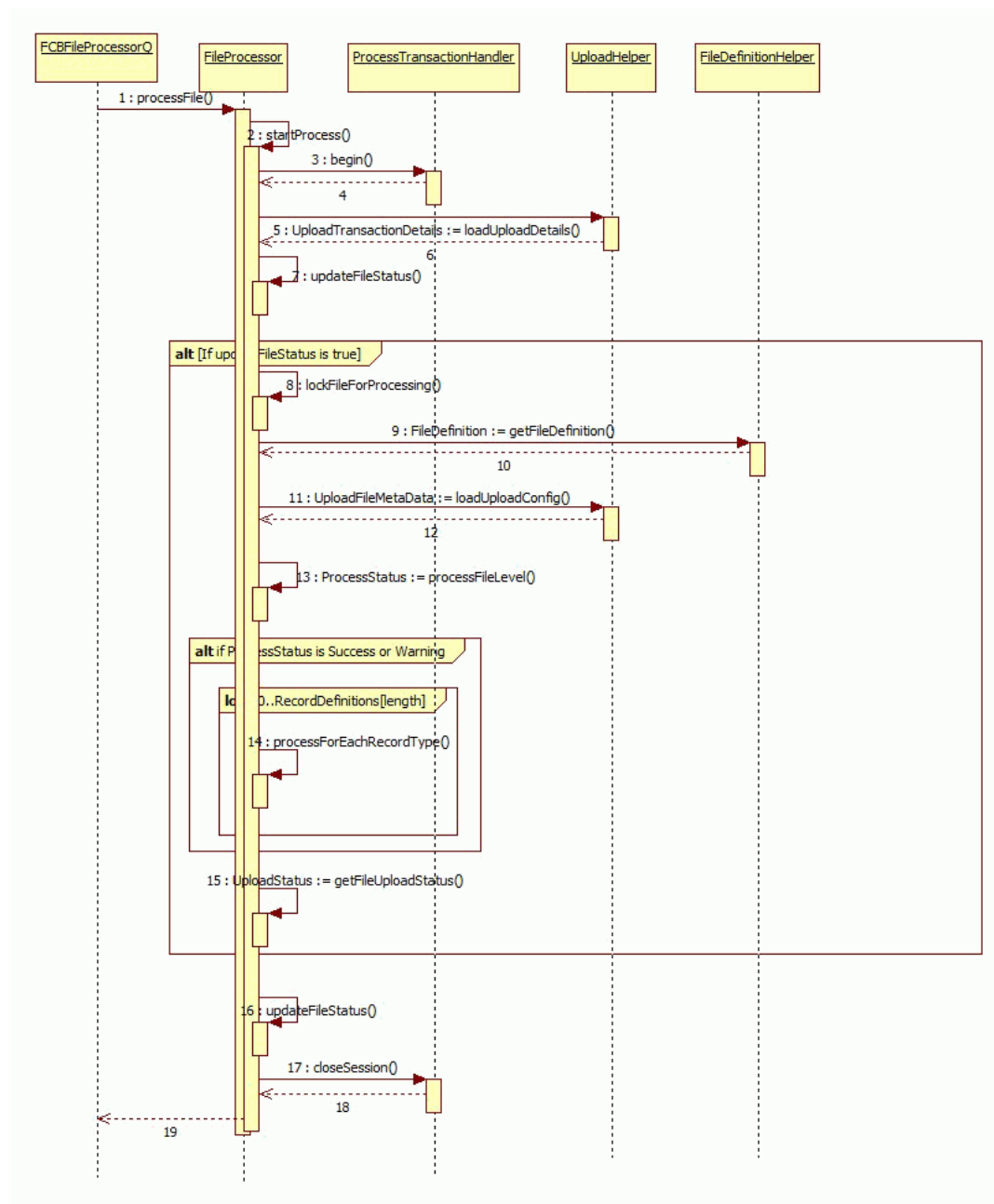
In OBP, file upload and file processing are two independent processes where the upload of file is done as one process and the processing on the uploaded data is done as another process. Every upload provides a unique file ID for the uploaded file. The processing is then done for each uploaded file and the final status is then provided at the end of the processing in the form of ProcessStatus.

The below section, from the extensibility perspective, provides the detailed understanding of the steps involved in the business logic processing of the files once the files are uploaded from the upload services. After the upload of the data, the data gets populated in the temporary tables in the database with the unique file id, which is then used for the processing of the uploaded file for the required business logic.

In the above mentioned salary credit example, the employer account details (in the form of header records) and the multiple employee account details (in the form of detail records) can be uploaded in OBP through the file upload, functionality which can then be processed for debiting the employer account and crediting the multiple salary accounts of the employees.

The framework of the uploaded file processing is shown in the sequence diagram below:

Figure 8–1 Uploaded Data File Processing Framework



From the implementation perspective, the following sections describe the configuration and processing of uploaded file.

8.1 Configuration

The configuration part of the uploaded file processing requires definition of the following components that needs to be defined for the processing on the uploaded file.

8.1.1 Database Tables and Setup

In case of file processing, there is one master table and individual record process tables for the recordType.

- FLX_EXT_FILE_UPLOAD_MAST

Table 8-1 FLX_EXT_FILE_UPLOAD_MAST

Column Name	Description
COD_FILE_ID	This defines the primary key identifier as file id for each specific file.
COD_XF_SYSTEM	This identifies the system to which the file type is associated. This indicates that the file is received from or sent to the particular system indicated by the system code.
FILE_TYPE	This identifies the type of file that is being uploaded. For every file type the format is defined. The file type can be like TXN .
NAM_HOFF_FILE	Name of the uploaded file.
TXT_NRRTV	File Narration for the uploaded file.
COD_ORG_BRN	This stores the originating branch code from where the file is uploaded.
CTR_BATCH_NO	This identifies the batch number of the file upload. This is generated internally.
DAT_FILE_PROCESS	The process date as specified while uploading a file.
COD_FILE_STATUS	Indicates the current status of the file.
DAT_FILE_UPLOAD	Indicates when the file was uploaded.
DAT_TIM_PROC_START	The start time indicates the time the processing starts.
DAT_TIM_PROC_END	The end time indicates the time the processing ends.
DAT_FILE_REVERSE	Indicates when the file was reversed.
CTR_TOTAL_REC	This value indicates the total records in the file.
CTR_PROCESS_REC	This Value indicates the number of records processed for a file.
CTR_REJECT_REC	This Value indicates the number of records rejected for a file.
FILE_SIZE	This value indicates the size of the file in bytes.
COMMENTS	The file Comments for the uploaded file if the processing fails.
FILE_CHECK_SUM	This column is used to store check sum of the file.
FROM_ODI	This flag is used to indicate whether upload is happening from ODI.
CURR_RECORD_TYPE	This column denotes the current record type being processed, updated after every recordType is successfully processed.

- FLX_EXT_<<Process>>_HEADERRECDTO e.g. FLX_EXT_SALCREDIT_HEADERRECDTO
- FLX_EXT_<<Process>>_DETAILRECDTO e.g. FLX_EXT_SALCREDIT_DETAILRECDTO

The file Id and record Id together as the key forms the record identifier in the record tables. The mandatory fields in the record tables are mentioned below. The additional required fields should be defined as the additional columns in the record tables.

Table 8–2 Mandatory Fields in Record Tables

Column Name	Description
RECORDID	This defines the primary key identifier as record id in the table. This is generated for every record.
FILEID	This is the primary key identifier as file id for the specific file.
RECORDTYPE	The type of record; possible values 'H', 'D' and 'F'.
RECORDNAME	Name of the record type; possible values 'Header', 'Detail' and 'Footer'.
DATA	Stores the complete data of each row of the file. This is populated for inquiry purposes that the user can view the contents of the record as it was read from the file.
LENGTH	Total length of DATA. This value is populated after the record is parsed.
COMMENTS	Comment update at the time of GEFU Upload and Processing of record.
RECORDSTATUS	List of Record Status : 1-UPLOADED, 2-FAILED_UPLOAD, 3-CANCELLED, 4-INPROGRESS, 5-PROCESSED, 6-FAILED_PROCESS, 7-REVERSED, 8-FAILED_REVERSED, 9-ABORTED, 10-MARKED_FOR_PROCESS.
DATE_RUN	This column holds the value of batch job's run date.
SEQ	This column holds the value of batch job's sequence number.
PROCESS_RESULT	This column holds the value of batch job process result.
ERROR_CODE	This column holds the value of batch job's error code.
ERROR_DESC	This column indicates the Error Description.
BRANCH_CODE	This column holds the branch code of the branch.
BRANCH_GROUP_CODE	This column holds the value of branch Group code.

- FLX_EXT_FILE_PARAMS

This table contains the information about the file definition template which is used to define the handlers, DTO and other details required for the processing of the uploaded file.

Table 8–3 FLX_EXT_FILE_PARAMS

Column Name	Description
COD_XF_SYSTEM	This identifies the system to which the file type is associated. This indicates that the file is received from or sent to the particular system indicated by the system code.
FILE_TYPE	This identifies the type of file that is being uploaded. For every file type the format is defined. The file type can be like TXN.
NAM_XF_SYSTEM	Name of the system to which the file type is associated. This indicates that the file is received from or sent to the particular system indicated by the system code.
NAM_FILE_TYPE	This is name of the type of file that is being uploaded. For every file type the

Column Name	Description
	format is defined. The file type would be like PYMT (Payment File) or SAL (Salary Upload).
NAM_UPLOAD_TMPL	XFF file definition template name.
FLG_OUTPUT_REQD	Once the processing of all the records is complete, a check is made if its value is 'Y' and then the response file is generated accordingly.
FLG_FILE_TRANSACTIONAL	Used to decide, whether File level validation is required or not.
CTR_COMMIT_SIZE	Used to commit records in batch while processing, so it's the batch size.
RELATIVE_PATH	If provided, this searches for xff file in the path: base_folder/folder_name_mentioned_here.
COD_ADHOC_REQUEST_CLASS	Adhoc request class name
CTR_UPLOAD_COMMIT_SIZE	Used to commit records in batch while validation, so it's the batch size.
FLAG_DUPLICATE_FILE_CHECK	This flag is used to indicate whether duplicate file check is required or not.
FLAG_FROM_ODI	This flag is used to indicate whether upload is happening from ODI.

- FLX_BATCH_JOB_SHELL_DTLS

This table contains the information about the batch processing with bean based shell mechanism as described in the 'Batch Framework Extension' section. The sample values are provided below:

Table 8-4 FLX_BATCH_JOB_SHELL_DTLS

Columns	Description	Sample Values
COD_SHELL	A unique code for batch shell. For example, 'upld_batch_shell_<ProcessType>'	upld_batch_shell_SalCredit
SHELL_NAME	Name for batch shell	GEFU Processing Shell For Salary Credit
SHELL_DESCRIPTION	Description about the batch shell	GEFU Processing Shell For Salary Credit
COMMIT_FREQUENCY	Commit frequency	100
FLG_RECOVERY_MODE	Recovery mode - ON / OFF	Y
FLG_STREAM_TYP	Type of stream : 'S' - fixed no of streams, 'R' - fixed no of rows, 'N' - no streams	S

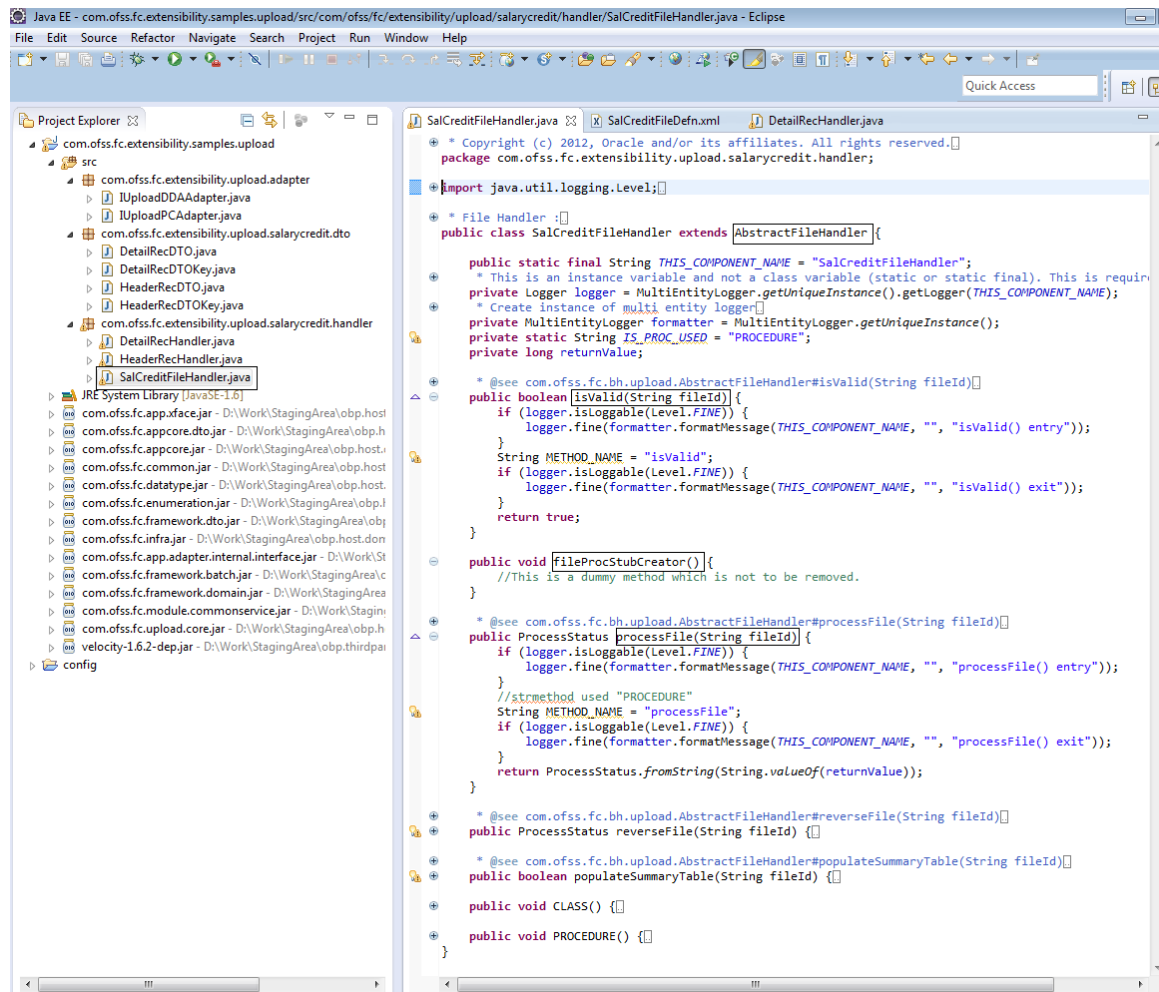
Columns	Description	Sample Values
STREAM_COUNT	No of streams for the batch shell. Applicable only for StreamType as 'S' or 'R'	2
INPUT_DRV_NAME	Fully classified class name mapped to the driver table	com.ofss.fc.entity.upload.AbstractRecordDTO
INPUT_SHELL_PARAM	Name for the shell parameter	AbstractRecordDTO
SERVICE_CLASS_NAME	Fully classified class name - starting point of the business logic execution	com.ofss.fc.upload.processor.batch.BatchRecordProcessor
SERVICE_METHOD_NAME	Method name of the service	processRecord
DRV_POP_PROC_NAME	Defines the name procedure used for driver table population	ap_gefu_pop_drv_gefu_rec
FLG_PROCESS_TYPE	RBP (Recoverable Batch Process) if any records fails in batch, it will continue and execute rest of the records in the stream or SBP (Simple Batch process) it will abort the stream	RBP
HELPER_CLASS_NAME	Helper class for caching big queries	com.ofss.fc.upload.processor.batch.GEFUBatchJobHelper
BATCH_NO	Batch number for the shell	1

8.1.2 File Handlers

File Handler class is written for processing of the uploaded file and should extend the AbstractFileHandler. The class name of the File Handler is mentioned in the File Definition XML. In this class, the following abstract methods should be implemented:

- `isValid()` : To check if the particular uploaded file is valid. Validations such as, is the file uploaded duplicate or not, or are the header details valid or not are done as part of file level validations.
- `processFile()` : To write the actual processing business logic where the functionality is implemented, if required, or else a default blank implementation is executed.

Figure 8–2 File Handlers

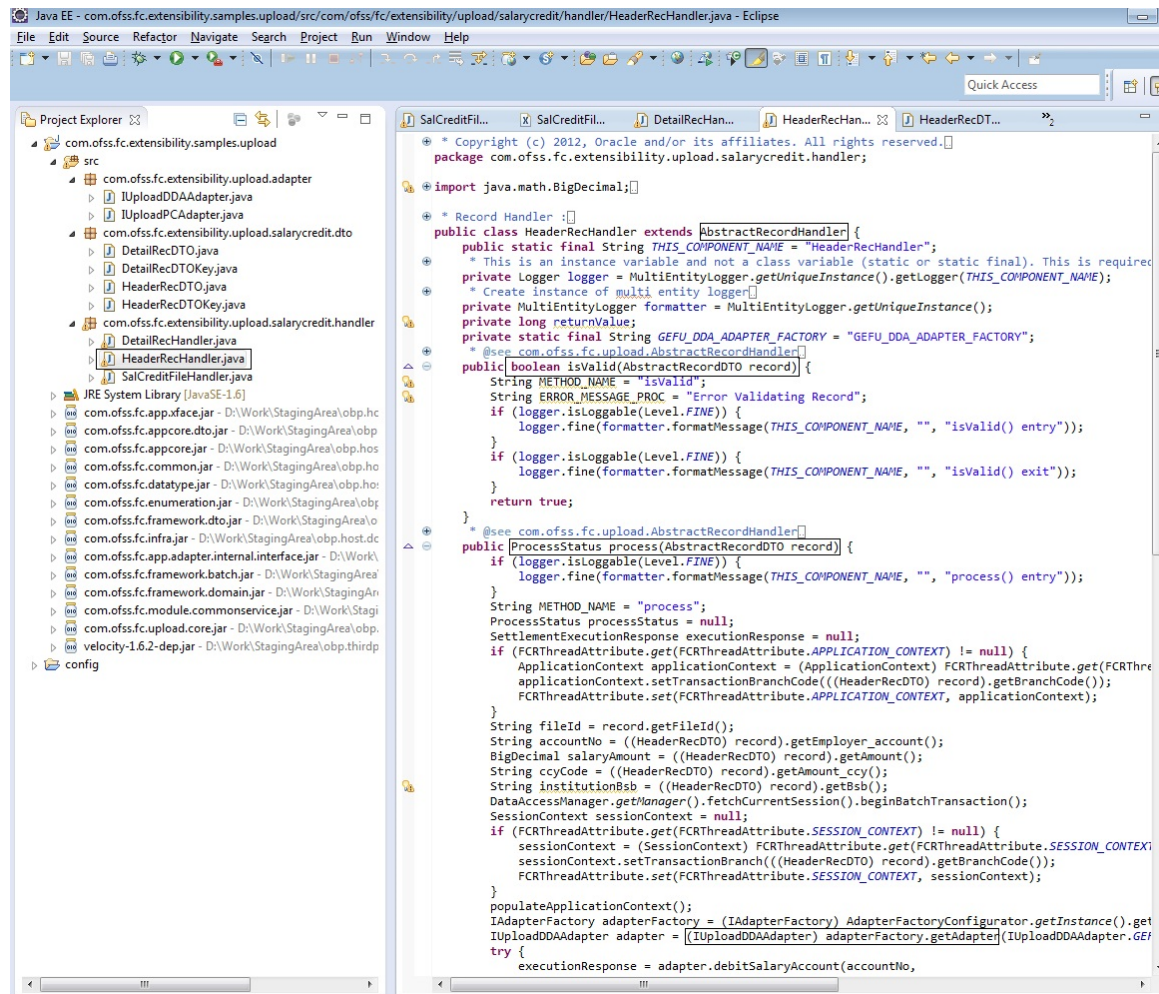


8.1.3 Record Handlers for Both Header and Details

This class provides the methods for record level validations and processing. It should extend the AbstractRecordHandler. The class name of the Record Handlers are also mentioned in the File Definition XML. The following abstract method needs to be implemented in this class:

- `isValid()` : To check if the particular uploaded record is valid for the processing purpose.
- `process()` : To write the actual processing business logic where the functionality is implemented. It is called once the file is successfully validated.

Figure 8–3 Record Handlers for Both Header and Details



8.1.4 DTO and Keys Classes for Both Header and Details

This is a persistent class for the particular process. This class provides the fields which represents the characteristics of the record data. This class is defined for each record type of a file.

Figure 8–4 DTO and Keys Classes for Both Header and Details - HeaderRecDTOKey

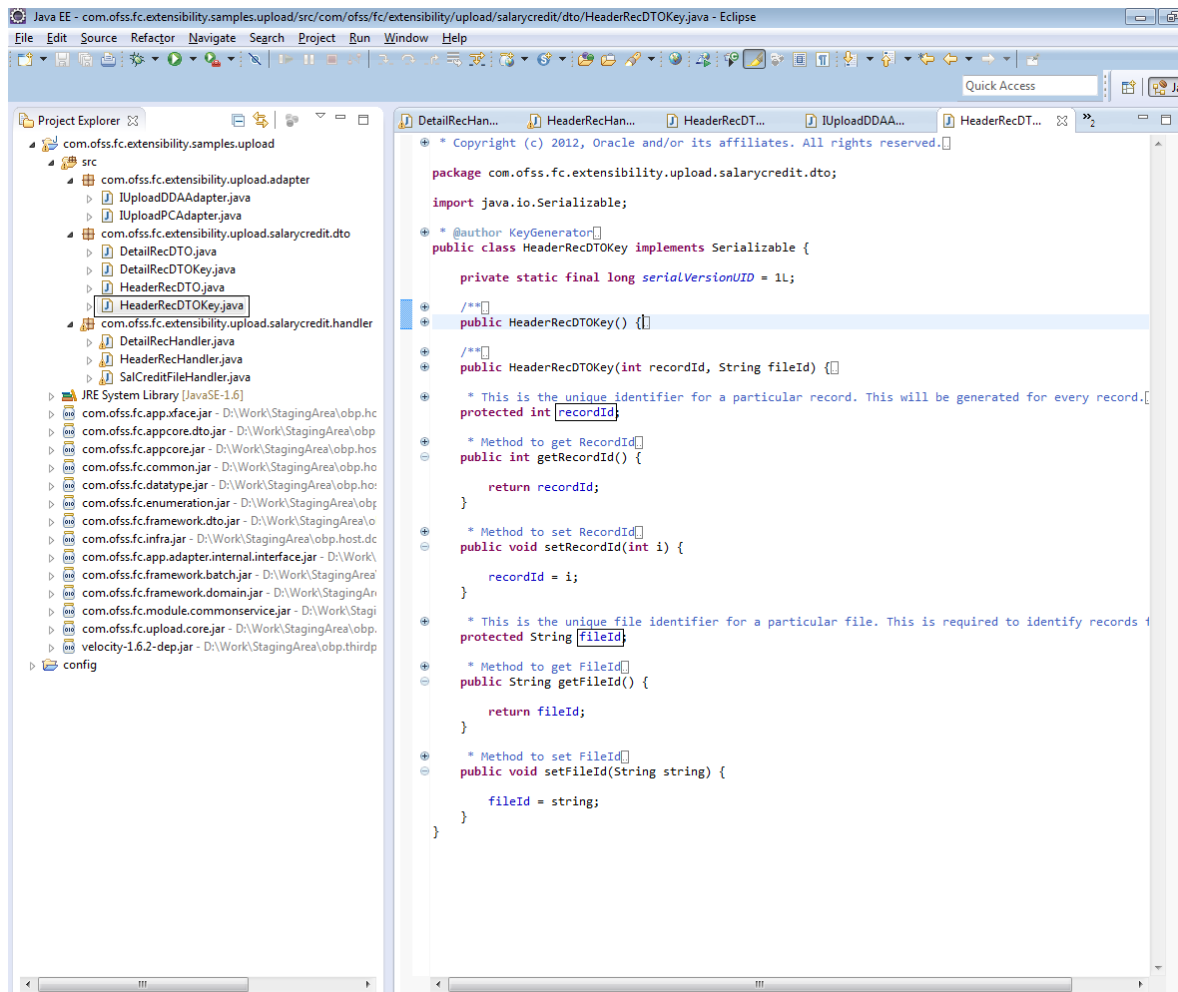
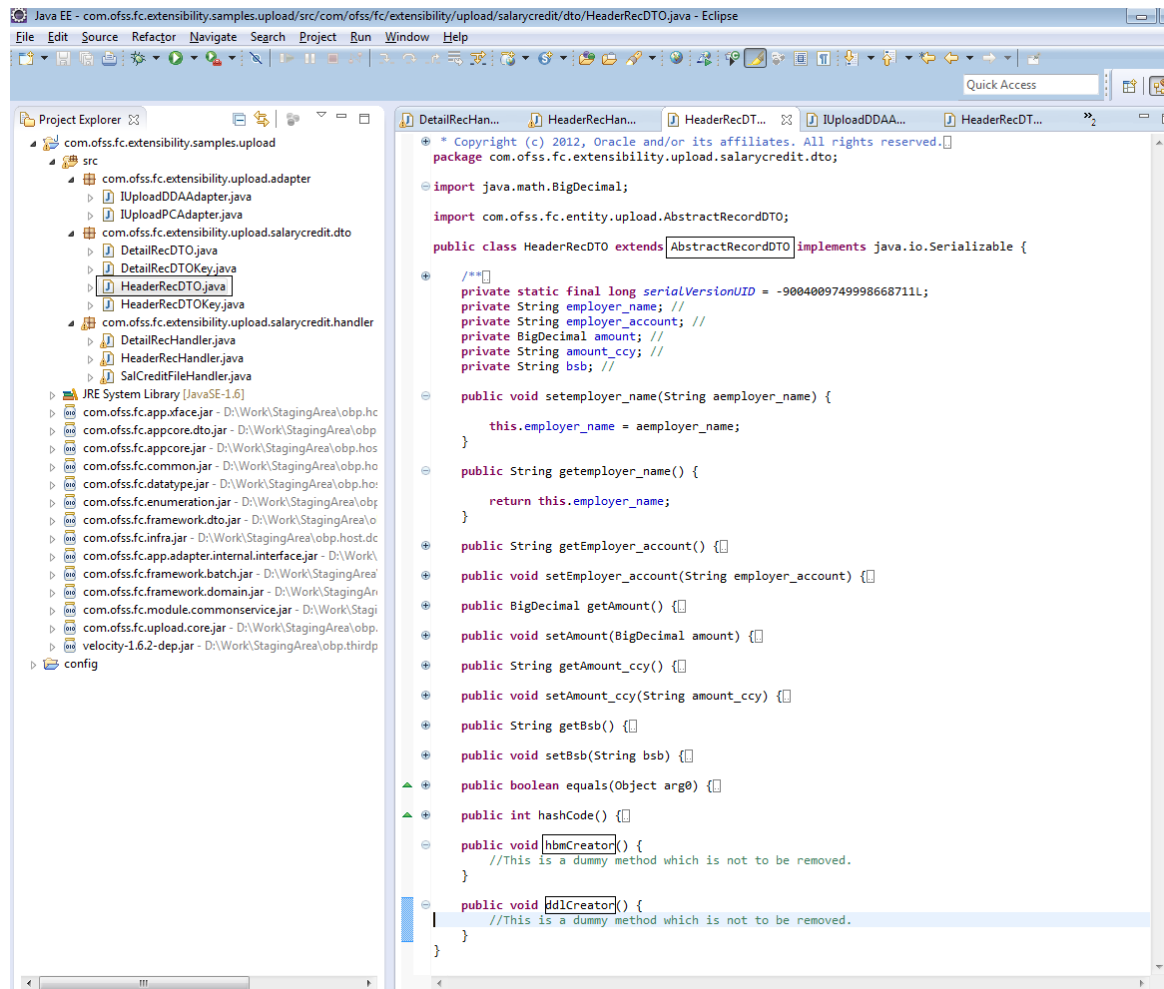


Figure 8–5 DTO and Keys Classes for Both Header and Details - AbstractDTORec



8.1.5 XFF File Definition XML

The xff file contains all the information about the different record type DTOs, the fields in those DTOs and the handlers pertaining to the uploaded file. The name of the xff file is mentioned in the FLX_EXT_FILE_PARAMS table. The file details are read from each tag in xff file and interpreted as described below in the table. The record element can occur N number of times based on number of record types present, for example if a particular upload has three record types Header, Detail and Trailer then there will be three elements for Record, each describing the three record types.

There are two one-to-many relationship in the file definition xml file:

- One 'File' element can have many 'Record' elements, depending upon the number of recordType present for this upload.
- One 'Record' element can have many 'Field' elements, depending upon the number of fields present for this recordType of upload.

Table 8–5 XXF File Definition XML

Elements	Attributes	Description
File		Contains all details about the FileHandler, there is only once occurrence of this element.
	fileName	This denotes logical name of the file.
	validationClassName	Fully qualified name of the FileHandler class.
	encryptionClass	This denotes the name of the class that is used for encryption (optional).
	charSet	This denotes the Charset of the file.
	delimiter	This denotes delimiter coming in the file (optional).
	comments	This is used to store comment on the file (optional).
	lengthInBytes	This Boolean variable is used to denote whether the file's length has to be calculated in bytes.
	xffSystem	Name of xff file system, name should be same as mentioned in COD_XF_SYSTEM in table FLX_EXT_FILE_PARAMS.
	fileType	Name of file type, name should be same as mentioned in FILE_TYPE in table FLX_EXT_FILE_PARAMS.
Record		Child element of "File" can have any number of occurrences depending upon number of RecordType for a particular Upload.
	recordHandlerClassName	Fully qualified name of the Handler class for this RecordType.
	recordType	This denotes record type which can be "Header", "Detail" or "Trailer"
	streamingAllowed	Indicates if the streaming is allowed for the record; Possible values are true or false.
	dtoClassName	Name of DTO for this particular recordType.
	recordName	Name of this record.
	multiplicity	This denotes whether this record type will appear only once in the file or multiple times. Value of this field will be either 1 (for only once) or -1 (for multiple times).
	maxFields	This denotes the maximum number of fields coming in the record type.
	comments	This stores comments (optional).
	maxLogicalRecords	This denotes maximum number of records that may come of this record type.
	parent	
	lastFieldOfVariableLength	This denotes whether the last field of the record is variable or not. This value can be either "true" or "false".
Field		Child element of "Record" can have as many occurrences as the number of fields in a particular recordType.

Elements	Attributes	Description
	name	Name of the field.
	type	This denotes field type. E.g.:- 'CHAR', 'NUMBER' and so on.
	length	Length of field.
	format	This denotes format of the field.
	recordIdentifier	This denotes whether this field is used to identify the record. Value of this field can be either true or false.
	nullable	This denotes whether this field can be null or not.
	defaultValue	Default value of this field if any.
	comments	This stores the comment on the field (optional).
	crossReferenceID	If another field wants to refer to this field then this id will be used.

Figure 8–6 XXF File Definition XML

The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the file structure, with 'SalCreditFileDefn.xml' selected under the 'xxf' directory. The Editor on the right shows the XML content of the file, which defines two records: 'Header' and 'Detail'. Each record is defined with a handler class, DTO class, and a list of fields with their respective attributes.

```

<File fileName="SalCredit"
  validationClassName="com.ofss.fc.upload.pc.salarycredit.handler.SalCreditFileHandler"
  encryptionClass="" charSet="UTF-8" delimiter="," comments=""
  lengthInBytes="true" xffSystem="SalCredit" fileType="TXW">
  <Record
    recordHandlerClassName="com.ofss.fc.upload.pc.salarycredit.handler.HeaderRecHandler"
    recordType="H" streamingAllowed="false"
    dtoClassName="com.ofss.fc.upload.pc.salarycredit.dto.HeaderRecDTO"
    recordName="Header" multiplicity="1" maxFields="2" comments=""
    maxLogicalRecords="1" parent="" lastFieldOfVariableLength="false">
    <Field name="recordType" type="CHAR" length="1" format=""
      recordIdentifier="true" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="employer_name" type="CHAR" length="40" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="employer_account" type="CHAR" length="16" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="amount" type="BigDecimal" length="16" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="amount_ccy" type="CHAR" length="3" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="bsb" type="CHAR" length="10" format=""
      recordIdentifier="false" nullable="true" defaultValue="" comments=""
      crossReferenceID="" />
  </Record>
  <Record
    recordHandlerClassName="com.ofss.fc.upload.pc.salarycredit.handler.DetailRecHandler"
    recordType="D" streamingAllowed="true"
    dtoClassName="com.ofss.fc.upload.pc.salarycredit.dto.DetailRecDTO"
    recordName="Detail" multiplicity="1" maxFields="2" comments=""
    maxLogicalRecords="1" parent="" lastFieldOfVariableLength="false">
    <Field name="recordType" type="CHAR" length="1" format=""
      recordIdentifier="true" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="account_no" type="CHAR" length="16" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="salary_amount" type="BigDecimal" length="16" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="salary_amount_ccy" type="Char" length="3" format=""
      recordIdentifier="false" nullable="false" defaultValue="" comments=""
      crossReferenceID="" />
    <Field name="bsb" type="Char" length="10" format=""
      recordIdentifier="false" nullable="true" defaultValue="" comments=""
      crossReferenceID="" />
  </Record>
</File>

```

8.2 Processing

Processing of an uploaded file is done on two levels, one on file level and the other on Record level. The processing is initially triggered when a message is sent on to a JMS Queue. The message is then picked up by an MDB which parses the message into a key value pair, and then passes it on to the FileProcessor by passing the processor type as an input. Based on the processor type, that is, header or detail record, the file processor initiates respective processing by invoking specific business logic written as file or record level handlers.

The processing of the business logic to different Service APIs of different modules are carried in the handler classes of the records. The processForRecordType() method of the FileProcessor invokes the respective handler classes that is, if the Header section is being processed, it invokes the HeaderHandler class.

As per the process, the headers are processed first and then the details records. Each and every record is processed individually. As soon as a file is picked for processing, its status is changed to InProgress so that the same file is not picked by any other process for processing. Individual records are processed based on its record type.

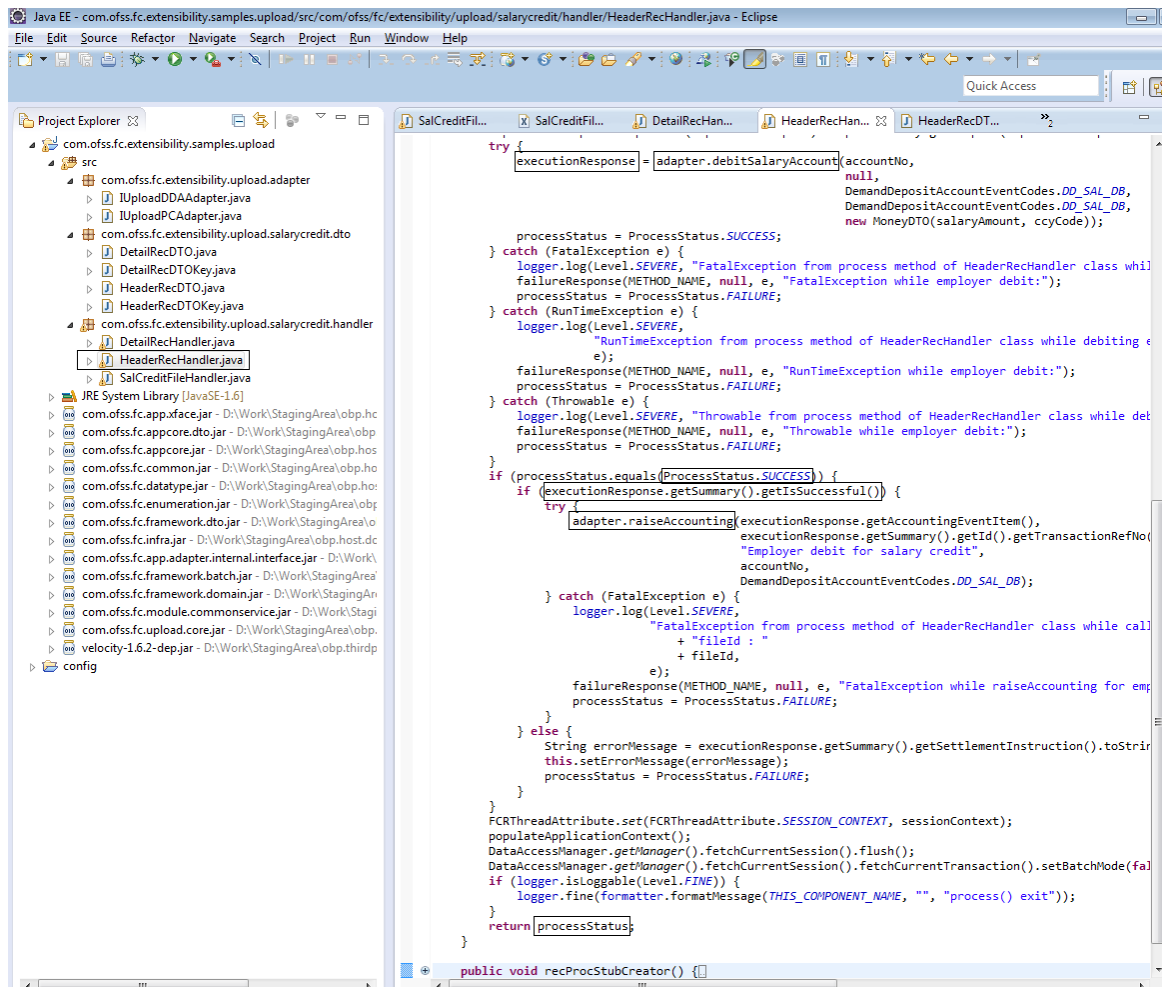
8.2.1 API Calls in the Handlers

The API calls of different exposed application services are called from the handlers. The respective method call from the adapter will return the response object which can be further used for another adapter call as the input value or for the validation purpose. In the following example, it is shown that the salary account is debited for the user and then the returned response summary is used for validation purpose before raising the accounting for that account.

```
<Response1>=Adapter1.<method call>(<method parameters>)  
If(<Validation on Response1>) {  
  <Response2>=Adapter2.<method call>(<method parameters containing  
  Response1>) }  
Example:
```

```
executionResponse = adapter.debitSalaryAccount()  
if(executionResponse.getSummary().getIsSuccessful()) {  
  adapter.raiseAccounting(); }  
}
```

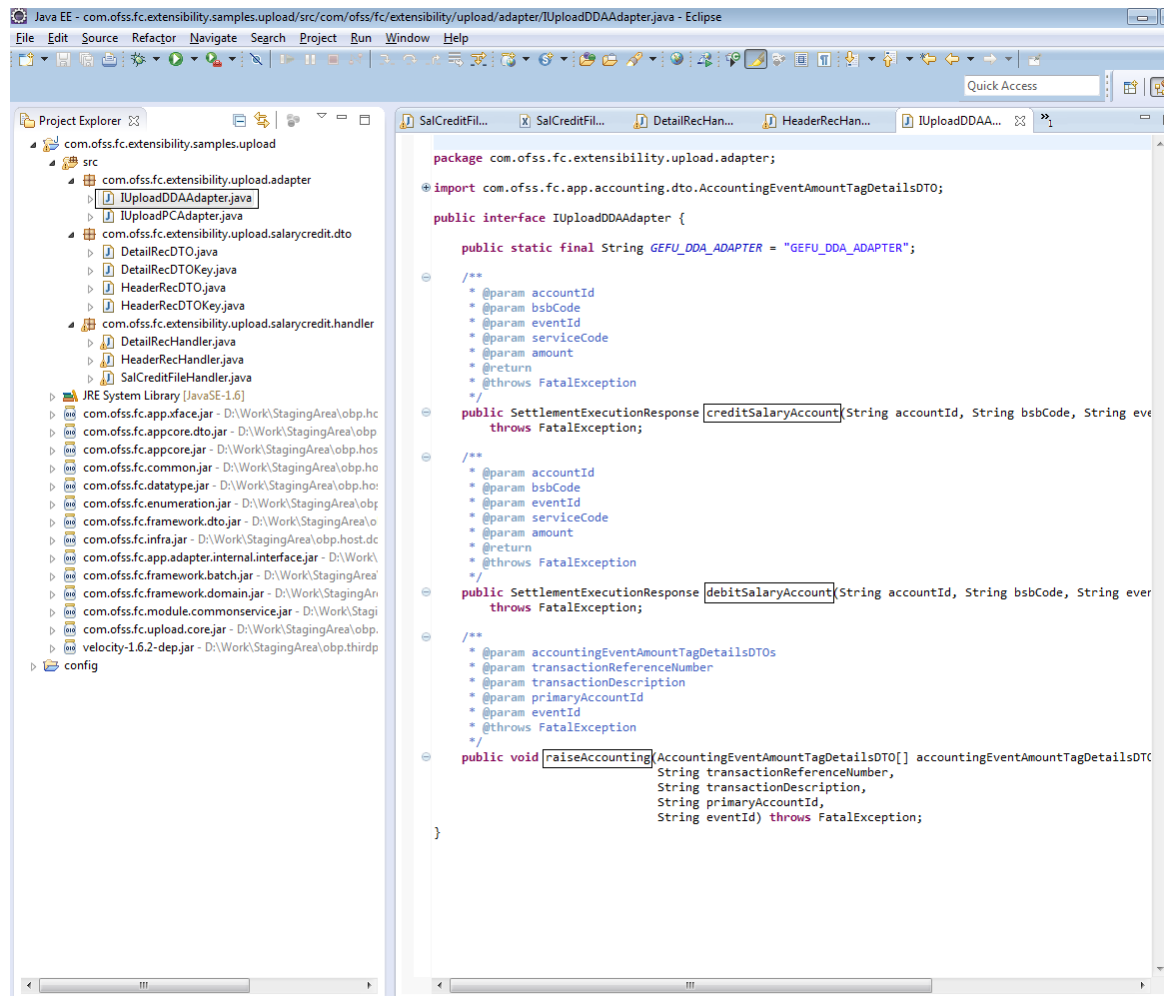

Figure 8–7 API Calls in Adapters



8.2.2 Processing Adapter

The processing adapters need to be implemented for invoking the required application service API. In the example, the new methods as `creditSalaryAccount()`, `debitSalaryAccount()` and `raiseAccounting()` are implemented by the user based on their requirements.

Figure 8–8 Processing Adapter



8.3 Outcome

In case of header or footer, there is only one Record for these record types, hence based on Record Level Status returned, the processing status is set, if RecordLevelStatusType is SUCCESS or WARNING, the PROCESSING_STATUS will be marked as SUCCESS else FAILURE.

In case of detail records, processing status is decided based on the criteria that is, if NumberOfRecords with record processing status as FAILED is equal to totalNoOfRecords then overall ProcessStatus is FAILED or if less than totalNoOfRecords then overall ProcessStatus is WARNING and if zero then overall ProcessStatus is SUCCESS. Also, in case there is error in insertion of any record to the working table then overall ProcessStatus is FAILED.

Each record on processing can have any one of the three process status. If process status is success it moves to the next record. If process status is warning then it moves to the next record but marks the record as failed. If process status is failure then an Exception is raised and the file is marked as Failed.

Table 8–6 Process Status

Status Name	Value	Description
SUCCESS	0	Processing of this record is a success. Further record processing should continue.
FAILURE	1	Processing of this record has failed. Further record processing should not continue.
WARNING	2	Processing of this record has failed. Further record processing should continue.

On successful processing, the record will get persisted into the respective table and return a status of '5' to the invoked method.

But, in case of failure, the status is returned as '6' for that particular record and it continues with the next record for processing. Also the exceptions raised during a failure can be appended into the "comments" column of the respective table.

8.4 Failure/Exception Handling

There can be processing failure in case of any validations failure caused by the service. In case of any exceptions raised, it will be handled in the handler class.

While invoking an API when the SessionContext variables are not passed properly it would result in null. 'Invalid user id' will be added in the comments column and the processing will not happen.

The exceptions raised during processing can be logged into the comments column of the respective table by calling the `setErrorMessage()` method. In case of process failure in file handling, this method can also be invoked from inside the catch block of the `processFile()` method:

```
this.setErrorMessage(errorMessage);  
processStatus = ProcessStatus.FAILURE;
```

9 Alerts Extension

OBP has to interface with various systems to transfer data which is generated during business activities that take place during teller operations or processing. OBP Application is, therefore, provided with the framework which can support on-line data transfer to interfacing systems.

The event processing module of OBP provides a mechanism for identifying executing host services as activities and generating or raising events that are configured against the same. Generation of these events results in certain actions that can vary from dispatching data to subscribers (customers or external systems) to execution of additional logic. The action whereby data is dispatched to subscribers is termed as *Alert*.

The following sections provides an overview of what the developer needs to do in order to add a new *Activity* and an *Event* which will be raised on execution of the said that activity. We will be using a sample activity and event to illustrate the steps.

Use Case: In the *Party -> Contact Information -> Contact Info* screen, user can create or update the contact details for a party. This screen has many attributes like *telephone number, email, do not disturb info* and so on. We will be registering this *update* transaction as an *Activity* and creating *Events* which will be raised on this activity.

9.1 Transaction as an Activity

This section describes how existing or new online transactions can be supported and recognized as activity for the events that are setup in the system with action, subscriber and dispatch configuration already in place. A transaction can be either financial or maintenance executing in the application server middleware host environment. This kind of setup is particularly useful when we have external systems like CEP, BAM to which data needs to be dispatched online.

The procedure for creating activities and events for a *financial* transaction is a subset of the same for a *maintenance* transaction. The aforementioned use case describes a maintenance transaction.

9.1.1 Activity Record

You will need to create a record for the activity in the table FLX_EP_ACT_B which stores all the recognized activities. This table has the following columns:

Table 9–1 FLX_EP_ACT_B

Column Name	Use	Example
COD_ACT_ID	The unique activity id for the activity. This id will be used in the activity - event mapping as well	'com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo'
TXT_ACT_NAME	Activity name	'ContactPointApplicationService.updateContactPoint.dndInfo'
TXT_ACT_DESC	Meaningful description of the activity	'DND Info Change'

Column Name	Use	Example
MODULE_ TYPE	Module code for the module of which the transaction is a part off	'PI'
CREATED_ BY	User id of the user creating this record	'SYSTELLER'
CREATION_ DATE	Creation date of this record	to_date('20110310', 'YYYYMMDD')
LAST_ UPDATED_ BY	User id of the user last updating this record	'SYSTELLER'
LAST_ UPDATE_ DATE	Last update date of this record	to_date('20110310', 'YYYYMMDD')
OBJECT_ VERSION_ NUMBER	Version number of this record	1
OBJECT_ STATUS	Status of this record	'A'

Sample script for Activity Record:

Figure 9–1 Sample script for Activity Record

```
--for insertion of activity record
DELETE FROM FLX_EP_ACT_B WHERE COD_ACT_ID =
'com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo';
INSERT INTO FLX_EP_ACT_B (COD_ACT_ID, TXT_ACT_NAME, TXT_ACT_DESC, MODULE_TYPE, FLG_IP_REQD, FLG_OP_REQD, FLG_LOG_REQD, TXT_LOG_CLASS,
CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_DATE, OBJECT_VERSION_NUMBER, OBJECT_STATUS)
VALUES ('com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo',
'ContactPointApplicationService.updateContactPoint.dndInfo', 'DND Info Change', 'PI', null, null, null, null, 'SYSTELLER', to_date
('20110310', 'YYYYMMDD'), 'SYSTELLER', to_date('20110310', 'YYYYMMDD'), 1, 'A');
```

9.1.2 Attaching Events to Activity

Recognized events can be attached to recognized activities. The mapping in this case can be many-to-many viz., an activity can raise multiple events and an event can be raised by multiple activities.

9.1.3 Event Record

You will need to create an event record in the table FLX_EP_EVT_B which stores all the recognized events. This table has the following columns:

Table 9–2 FLX_EP_EVT_B

Column Name	Use	Example
COD_EVENT_ ID	The unique event id for this event. This id will be used in the activity - event mapping as well.	'PI_UPD_DND_INFO'

Column Name	Use	Example
TXT_EVENT_TYP	The type of event	'ONLINE'
TXT_EVENT_DESC	Meaningful description for the event	'DND Info Updated'
EVENT_CATEGORY_ID	The category code for this event	2

Sample script for Event Record:

Figure 9–2 Sample script for Event Record

```
--for insertion of event record
DELETE FROM FLX_EP_EVT_B WHERE COD_EVENT_ID = 'PI_UPD_DND_INFO';
INSERT INTO FLX_EP_EVT_B (COD_EVENT_ID, TXT_EVENT_TYP, TXT_EVENT_DESC, EVENT_CATEGORY_ID)
VALUES ('PI_UPD_DND_INFO', 'ONLINE', 'DND Info Updated', 2);
```

9.1.4 Activity Event Mapping Record

You will need to create an activity event mapping record in the table FLX_EP_ACT_EVT_B which stores the mapping between all activities and events. This table has the following columns:

Table 9–3 FLX_EP_ACT_EVT_B

Column Name	Use	Example
COD_ACT_ID	The unique activity id as specified in the activity table	'com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo'
COD_EVENT_ID	The unique event id as specified in the event table	'PI_UPD_DND_INFO'
TXT_ACT_EVT_DESC	Meaningful description for the activity event mapping	'DND Info Updated'
TXT_EVT_TYP	The type of event	'OTHER'
TXT_ACT_EVT_TYP	The type of activity event mapping	'ONLINE'

Sample script for Activity Event Mapping Record:

Figure 9–3 Activity Event Mapping Record

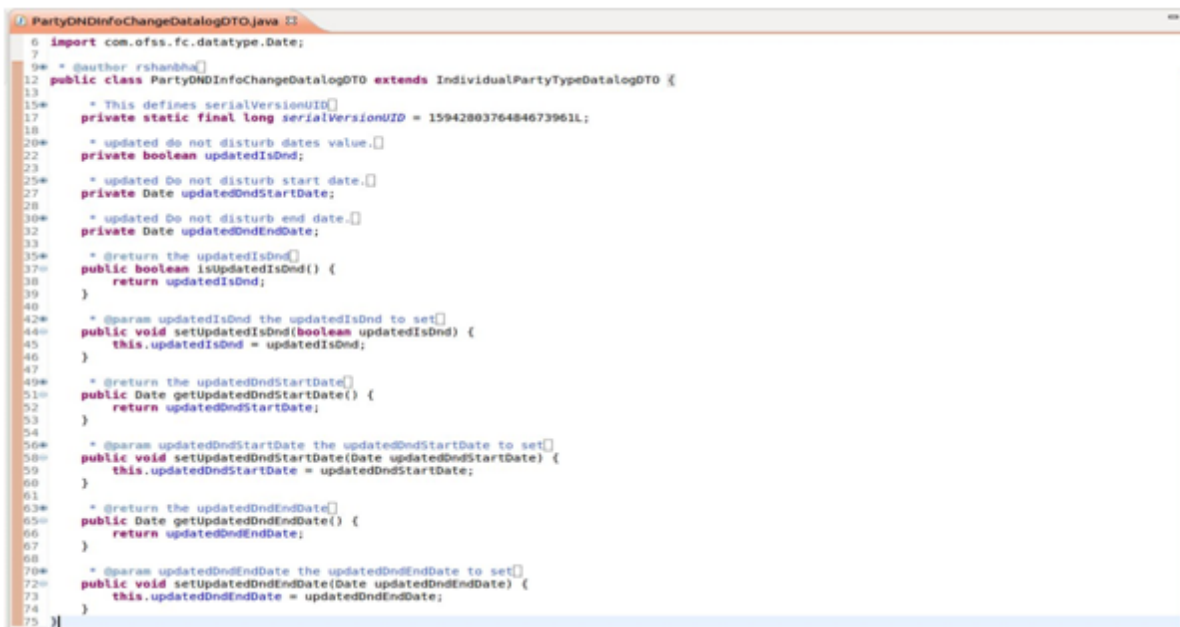
```
--for insertion of activity - event mapping
DELETE FROM FLX_EP_ACT_EVT_B WHERE COD_ACT_ID =
'com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo' AND COD_EVENT_ID = 'PI_UPD_DND_INFO';
INSERT INTO FLX_EP_ACT_EVT_B (COD_ACT_ID, COD_EVENT_ID, TXT_ACT_EVT_DESC, TXT_EVT_TYP, TXT_ACT_EVT_TYP)
VALUES ('com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint.dndInfo', 'PI_UPD_DND_INFO', 'DND Info
Updated', 'OTHER', 'ONLINE');
```

9.1.5 Activity Log DTO

In order to transfer activity data to the actions defined for the event, you need to develop data objects to contain the activity data. The DTO should implement the interface *com.ofss.fc.xface.ep.dto.IActivityLog*. Module specific activity log DTO's which already implement the *IActivityLog* interface are present. These DTO's contain the application specific and module specific activity data. You can extend the module's DTO class and add the transaction specific activity data.

For party module, the class *com.ofss.fc.app.party.dto.alert.IndividualPartyTypeDatalogDTO* is one of the classes that implement the *IActivityLog* interface. For the aforementioned activity, the activity log DTO can be as follows:

Figure 9–4 Activity Log DTO



```

1  PartyDNDInfoChangeDatalogDTO.java
2  import com.ofss.fc.datatype.Date;
3
4  9*  * @author rshantha
5
6  12 public class PartyDNDInfoChangeDatalogDTO extends IndividualPartyTypeDatalogDTO {
7
8  15*  * This defines serialVersionUID
9  17 private static final long serialVersionUID = 1594280376484673961L;
10
11  20*  * updated do not disturb dates value.
12  22 private boolean updatedIsDnd;
13
14  25*  * updated Do not disturb start date.
15  27 private Date updatedDndStartDate;
16
17  30*  * updated Do not disturb end date.
18  32 private Date updatedDndEndDate;
19
20  33
21  35*  * @return the updatedIsDnd
22  37 public boolean isUpdatedIsDnd() {
23  38     return updatedIsDnd;
24  39 }
25
26  42*  * @param updatedIsDnd the updatedIsDnd to set
27  44 public void setUpdatedIsDnd(boolean updatedIsDnd) {
28  45     this.updatedIsDnd = updatedIsDnd;
29  46 }
30
31  47*  * @return the updatedDndStartDate
32  49 public Date getUpdatedDndStartDate() {
33  50     return updatedDndStartDate;
34  51 }
35
36  54*  * @param updatedDndStartDate the updatedDndStartDate to set
37  56 public void setUpdatedDndStartDate(Date updatedDndStartDate) {
38  57     this.updatedDndStartDate = updatedDndStartDate;
39  58 }
40
41  61*  * @return the updatedDndEndDate
42  63 public Date getUpdatedDndEndDate() {
43  64     return updatedDndEndDate;
44  65 }
45
46  68*  * @param updatedDndEndDate the updatedDndEndDate to set
47  70 public void setUpdatedDndEndDate(Date updatedDndEndDate) {
48  71     this.updatedDndEndDate = updatedDndEndDate;
49  72 }
50
51  73 }
52  75 }

```

9.1.6 Alert Metadata Generation

This section describes the different types of alert metadata generation.

Metadata Generation

To generate metadata for alerts you need to have plugin.

1. Once you have plugin you need to set properties in preferences in windows tab for Service Publisher, Service Deployer and Workspace Path.
2. Go to your DTO class and right-click that class and click the following : *Oracle Banking Platform -> Generate DTO Metadata*.

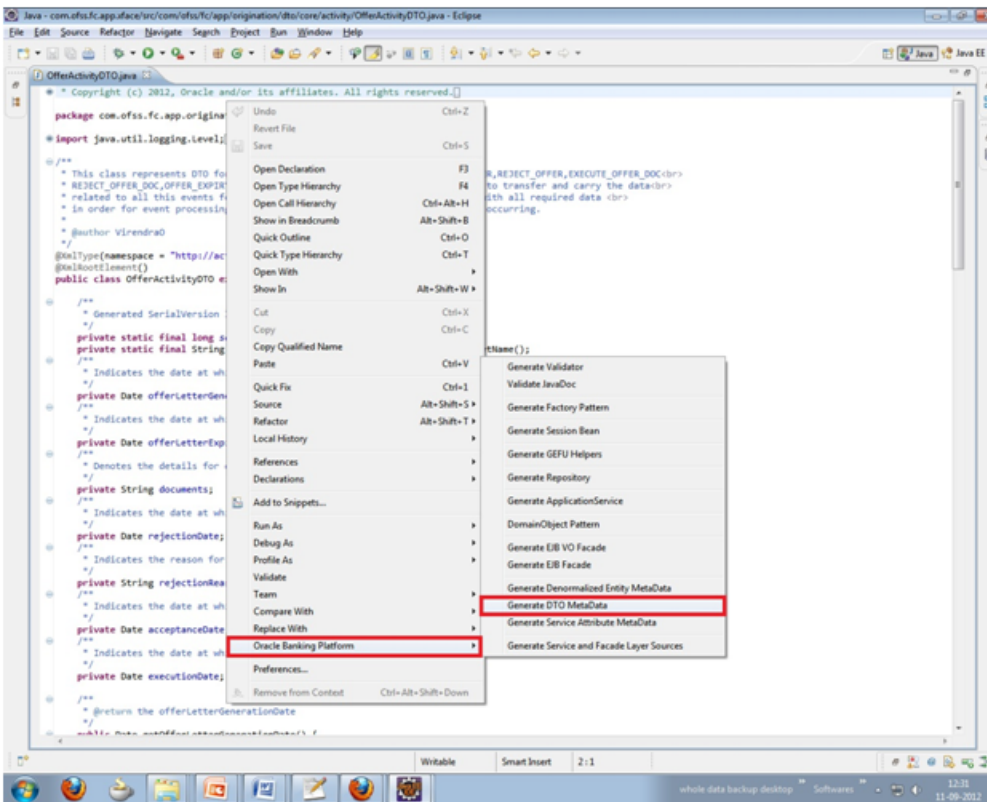
This will generate the insert scripts for following two tables:

- FLX_MD_DATA_DEFN
- FLX_MD_FIELDS_DEFN

These scripts will be generated in your config folder by default. The path of this script is:

WorkspaceDirectory -> config -> meta-data-scripts -> incr-meta-data.log

Figure 9–5 Metadata Generation



Service Data Attribute Generation

After generating metadata, we need to generate service attribute which will be mapped with facts which will be used in data bindings in Alert Maintenance screen AL04.

To generate we need to activity ID class for specific event, DTO is used for this activity ID.

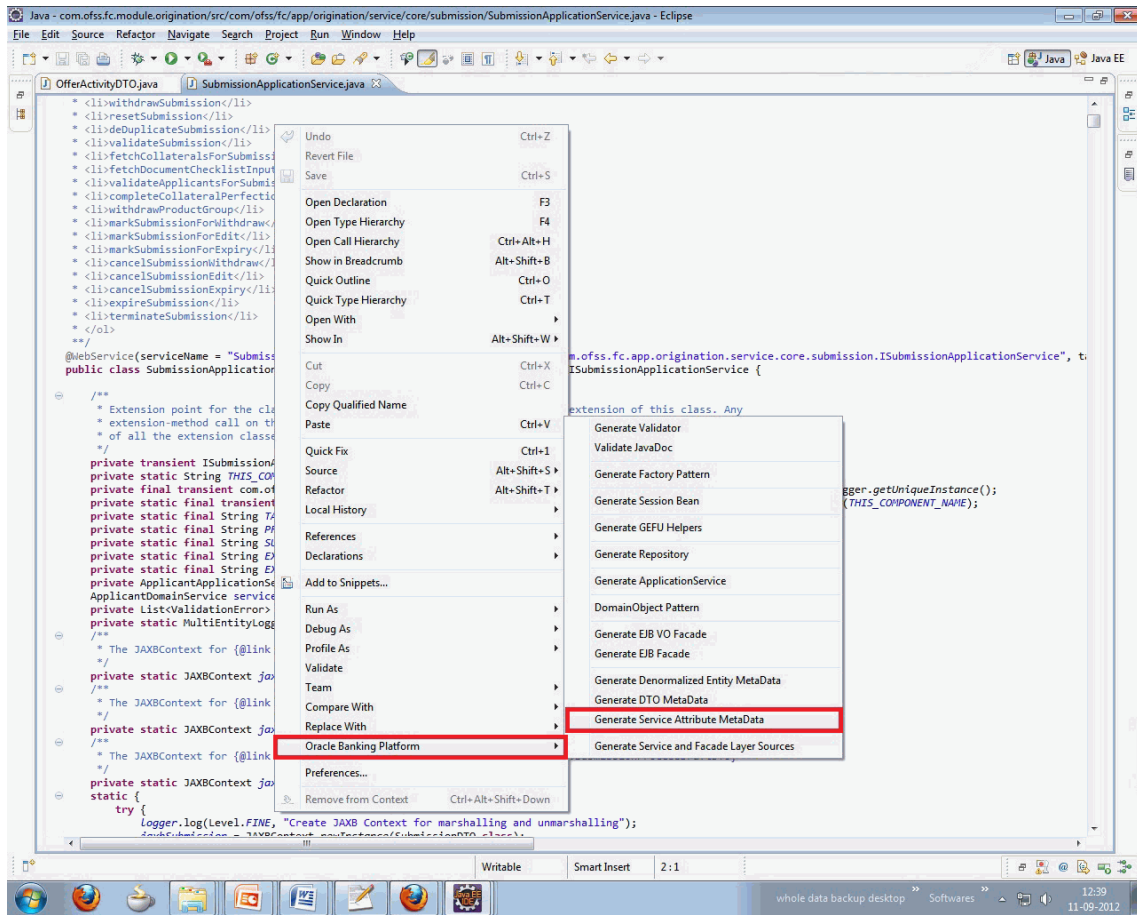
1. Right-click that debug service and select *Oracle Banking Platform -> Generate Service Attribute Metadata*.
2. In this case also insert scripts will be generate in same location as metadata attributes.

This will generate the insert scripts for following tables:

- FLX_MD_SERVICE_INPUTS
- FLX_MD_SERVICE_OUTPUT
- FLX_MD_SERVICE_ATTR

There are some steps in generating of service attribute which are as follows:

Figure 9–6 Service Data Attribute Generation



FLX_MD_SERVICE_ATTR is used to map the alert activity attribute with the fact code and to map the alert activity attribute with the DTO field to extract the data from.

As an example, the key fields in FLX_MD_SERVICE_ATTR for an alert activity attribute have been listed below:

Table 9–4 Key Fields in FLX_MD_SERVICE_ATTR

Column	Description
COD_SERVICE_ATTR_ID	The Unique ID for the Attribute of any Activity configured for an alert. For example, com.ofss.fc.app.account.service.accountaddresslinkage.AccountAddressLinkageApplicationService.createAccountAddressLinkage.Alert.Party.Address.City.DTO
TYP_DATA_SRC	Indicates the Data Source(entity/input/DTO) for the Attribute of the Resource
COD_ATTR_ID	This field indicates the Fact Code. For example, Alert.Party.Address.City

Column	Description
COD_SERVICE_ID	This field indicates the Activity ID. For example, com.ofss.fc.app.account.service.accountaddresslinkage.AccountAddressLinkageApplicationService.createAccountAddressLinkage
REF_FIELD_DEFN_ID	<p>This field indicates the DTO leaf field from which the data is extracted. For e.g.: com.ofss.fc.app.dda.dto.alert.AccountAddressLinkageAlertDTO.Address, com.ofss.fc.datatype.PostalAddress.City</p> <p>Data for this column is interpreted /extracted as follows.</p> <pre>com.ofss.fc.datatype.PostalAddress address = com.ofss.fc.app.dda.dto.alert.AccountAddressLinkageAlertDTO.getAddress(); String city = address.getCity()</pre>

9.1.7 Alert Message Template Maintenance

User will maintain template format and template ID to be used for the alerts definition.

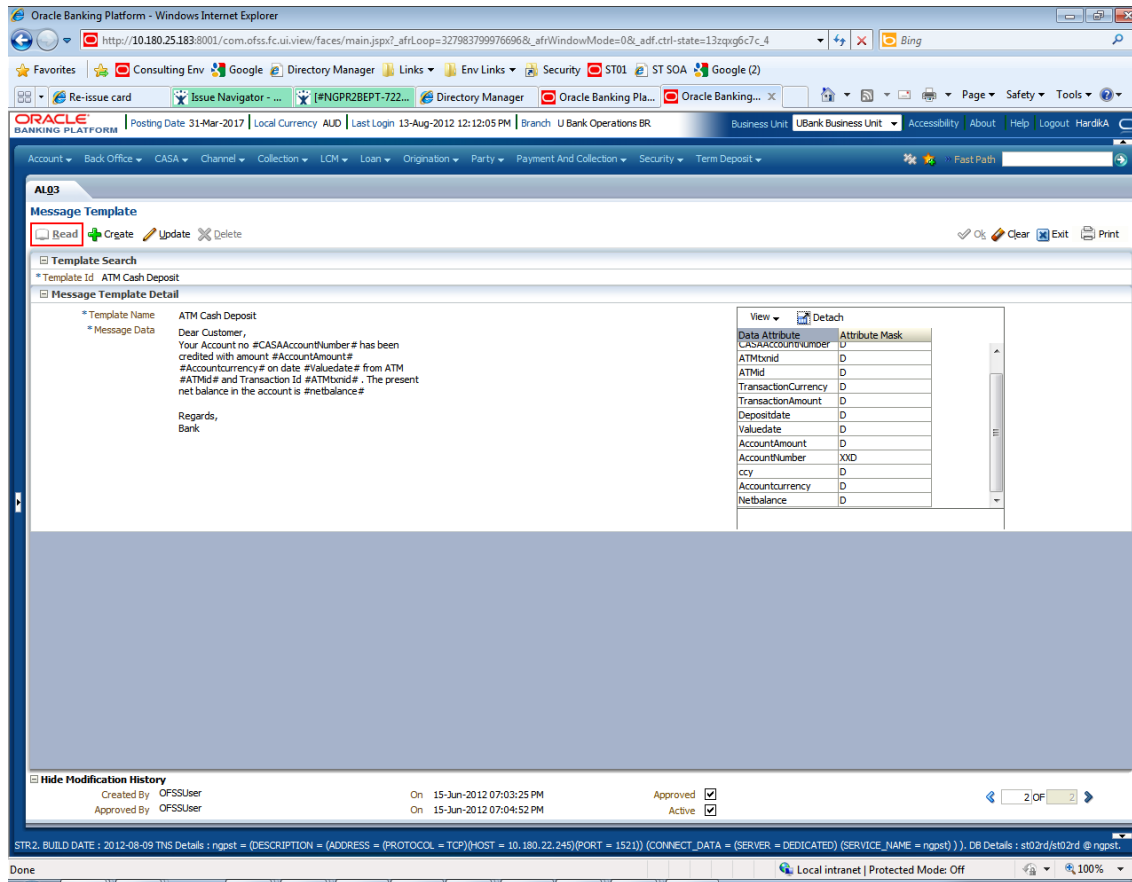
These messages need to be defined only if the same template is going to be used for multiple events. Else there is a provision to define the message template during the definition of the alert itself.

All data elements defined within the '#' symbol will be defaulted in the panel below as data attribute.

For example, your account Number #Acct No# has been credited with #currency# #transaction amount# being cash deposited.

The user can Mask certain digits in data elements that are preceded with '#' under the 'Attribute Mask' column.

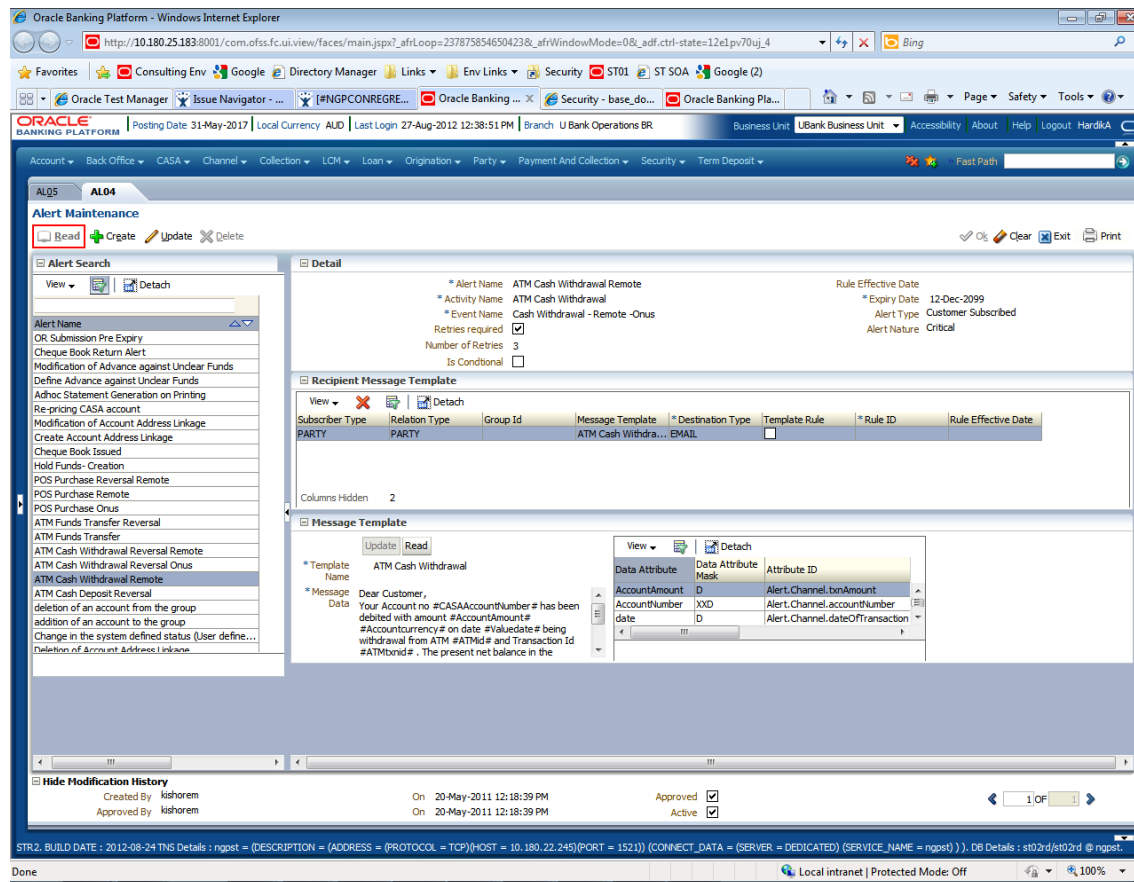
Figure 9–7 Alert Message Template Maintenance



9.1.8 Alert Maintenance

Given below is the Alert Maintenance screen.

Figure 9–8 Alert Maintenance



We can define the alert name, expiry date, alert type (Customer Subscribed/ Mandatory) and link this with predefined activity and event. These entries are fed to table "flx_ep_act_evt_acn_b".

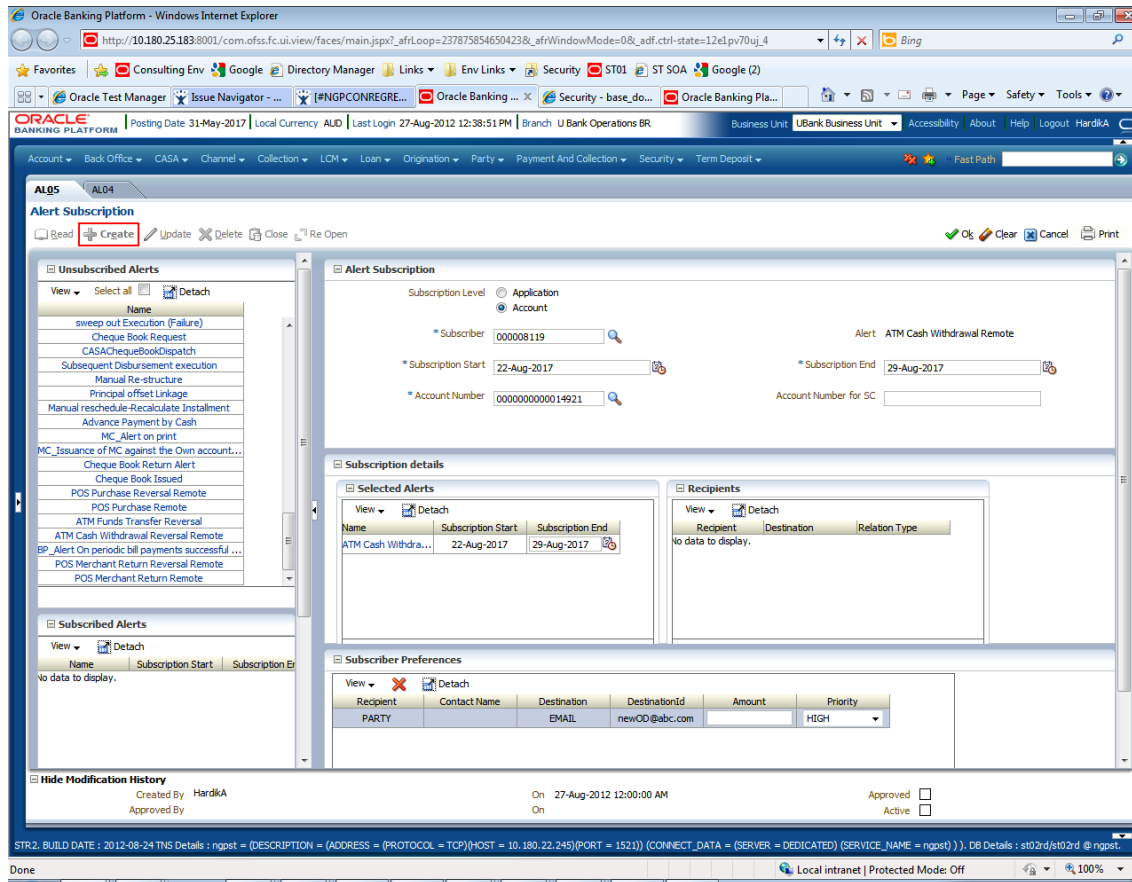
Now, we need to link a Recipient Message Template/s with this alert. For this we drag recipients from the Recipient Panel on to the Recipient Message Template Panel. In this setup, we define the kind of recipient and link this to predefined Message Template and Destination Types. The entry for this goes to table "flx_ep_evt_rec_b".

Finally, we need to complete the Message Template Mapping Configuration for each Recipient Message Template. For this, we map each data attribute of each Recipient Message Template with a corresponding attribute (Fact Code) from the drop down. This drop down populates fact codes configured for this activity id in the metadata table FLX_MD_SERVICE_ATTRIBUTE. The entry for this goes to table "flx_ep_msg_src_b"

9.2 Alert Subscription

Subscription can be done for alerts at **account level** or at **application level** (called as subscription level).

Figure 9–9 Alert Subscription



9.2.1 Transaction API Changes

You will need to modify the transaction API to support the newly registered activities. This section gives an overview of how the developer needs to modify the transaction API.

The entry point for activity business logic would be the service call for the transaction. In the aforementioned use case, the service call would be

com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint(...).

Figure 9–10 Transaction API Changes - Service Call

```
public TransactionStatus updateContactPoint(SessionContext sessionContext, ContactPointDTO dto) throws FatalException {
    super.checkAccess("com.ofss.fc.app.party.service.contact.ContactPointApplicationService.updateContactPoint", sessionContext, dto);
    if (logger.isLoggable(Level.FINE)) {
        logger.log(Level.FINE,
            MultiEntityLogger.getUniqueInstance()
                .formatMessage("Entered method updateContactPoint with partyId as '%s', contact point type as '%s' and",
                    dto.getPartyId(),
                    dto.getContactPoint() == null ? "null" : dto.getContactPoint().toString(),
                    dto.getPreferenceType() == null ? "null" : dto.getPreferenceType().toString()));
    }
    Interaction.begin(sessionContext);
    TransactionStatus transactionStatus = fetchTransactionStatus();
    try {
        Interaction.markCurrentTask(PartyTaskConstants.CONTACT_PREFERENCE);
        createTransactionContext(sessionContext, MaintenanceType.MODIFICATION);
    }
}
```

If the activity needs to be conditional, then the logic for evaluating the conditions should be present inside the service call. This should be followed by the invocation of the routine to register the activity. In the

forementioned use case, the activity should be registered only if the *update* transaction updates the attributes associated with *DND Information*. Following code snippet shows the conditional evaluation and invocation of the call to register activity.

Figure 9–11 Transaction API Changes - Conditional Evaluation

```

} else if(dto.isDnd() != telephoneNumberExisting.isDnd()
    || (dto.getDndStartDate() != null && !dto.getDndStartDate().equals(telephoneNumberExisting.getDndStartDate()))
    || (dto.getDndEndDate() != null && !dto.getDndEndDate().equals(telephoneNumberExisting.getDndEndDate()))) {
    if (logger.isLoggable(Level.FINE)) {
        logger.log(Level.FINE, MultiEntityLogger.getUniqueInstance()
            .formatMessage("Registering activity for alerts on change of dnd details for party '%s'",
                dto.getPartyId()));
    }
    partyAlertHelper.persistActivityLog(CONTACTPOINT_UPDATECONTACTPOINT_DNDINFO,
        dto,
        sessionContext,
        partyName.getFullName(),
        EVENTCODE_DNDINFO_CHANGE);
}
}

```

The *persistActivityLog(..)* routine primarily takes the *Activity Id*, *Event Id* and *Activity Log DTO*. This routine first calls a helper routine to populate the activity log DTO with the activity data and then passes on the DTO to the appropriate *Event Processing Adapter* which will register the activity and generate associated events.

Figure 9–12 Transaction API Changes - persistActivityLog(..)

```

/**
 * This method logs/registers the activity log DTO
 *
 * @param MI,String,activityId, The ActivityId for which we need to log the data for the further processing of
 * alerts.
 * @param MI,Object,object, holds the new data, sent as DTO's.
 * @param MI,SessionContext,sessionContext, holds the session data, used for creating the ApplicationContext.
 * @param MI,String,partyName, holds the Party Name.
 * @param MI,String,eventId, holds the event Id.
 * @throws FatalException
 */
public void persistActivityLog(String activityId, Object object, SessionContext sessionContext, String partyName, String eventId)
    throws FatalException {
    IActivityLog activityLog = populateActivityLog(activityId, object, partyName);
    if (activityLog != null) {
        com.ofss.fc.app.adapter.IAdapterFactory adapterFactory = AdapterFactoryConfigurator.getInstance()
            .getAdapterFactory(ModuleConstant.EVENT_PROCESSING);
        IEventProcessingAdapter adapter = (IEventProcessingAdapter) adapterFactory.getAdapter(EventProcessingAdapterConstant.MODULE_TO_ACTIVITY);
        adapter.registerActivityAndGenerateEvent(createApplicationContext(sessionContext), activityId, eventId, new Date(), activityLog);
    } else {
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, "PartyAlertHelper.persistActivityLog : ActivityLog is null");
        }
    }
}
}

```

You will need to add the business logic to populate the activity log DTO with the data specific to the transaction and the activity. This logic can be present inside the activity helper class for the module. Module specific activity attributes can also be populated in this logic. Following code snippet shows the activity log DTO population with activity data for the aforementioned activity.

Figure 9–13 Transaction API Changes - Activity Log

```

private IActivityLog populateActivityLogForDNDInfoChange(Object object, String partyName) {
    ContactPointDTO contactPointDTO = (ContactPointDTO) object;
    PartyDNDInfoChangeDataLogDTO activityLog = new PartyDNDInfoChangeDataLogDTO();
    activityLog.setCustomerId(contactPointDTO.getPartyId());
    activityLog.setPartyId(contactPointDTO.getPartyId());
    activityLog.setFullName(partyName);
    activityLog.setUpdatedIsDnd(contactPointDTO.isDnd());
    activityLog.setUpdatedDndStartDate(contactPointDTO.getDndStartDate());
    activityLog.setUpdatedDndEndDate(contactPointDTO.getDndEndDate());
    activityLog.setCriticalNotification(true);
    return activityLog;
}
}

```

Figure 9–14 Transaction API Changes - Register Activity

```

/**
 * Used to register an Activity with an associated Event
 *
 * @param activityID
 * @param eventID
 * @param eventProcessingDate
 * @param activityLog
 * @return
 * @throws FatalException
 */
public String registerActivityAndGenerateEvent(ApplicationContext applicationContext,
                                             String activityID,
                                             String eventID,
                                             Date eventProcessingDate,
                                             Object logObject) throws FatalException {

    ActivityLog activityLog = (ActivityLog) logObject;
    ActivityRegistrationApplicationService activityManager = new ActivityRegistrationApplicationService();
    SessionContext sessionContext = AdapterContextHelper.fetchSessionContext();
    if (sessionContext == null) {
        sessionContext = AdapterContextHelper.fetchBasicSessionContext(applicationContext);
    }
    ActivityEventKeyDTO activityEventKeyDTO = new ActivityEventKeyDTO();
    activityEventKeyDTO.setActivityId(activityID);
    activityEventKeyDTO.setEventId(eventID);
    ActivityRegistrationResponse response = activityManager.registerActivityAndGenerateEvent(sessionContext,
                                                                                          activityEventKeyDTO,
                                                                                          eventProcessingDate,
                                                                                          activityLog);

    return response.getActivityDataId();
}

```

The *Event Processing Adapter* contains the logic to register the activity and generate events. You can use the existing adapter class *com.ofss.fc.app.adapter.impl.ep.EventProcessingAdapter* or write your own custom adapter which must implement the interface *com.ofss.fc.app.adapter.impl.ep.IEventProcessingAdapter*.

All the above steps would suffice to support a transaction as an activity and raise events on the activity.

On successful completion of the transaction and the activity registration and event generation, you can view the activity log in the table `FLX_EP_ACT_LOG_B` and the generated events log in the table `FLX_EP_EVT_LOG_B`.

Actions associated with the activity events would pick up the activity and event data from these tables for processing.

9.3 Alert Processing Steps

For any modules the starting point is *EventProcessingAdapter* method named 'registerActivityAndGenerateEvent'.

Through this we call 'registerActivityAndGenerateEvent' method of *ActivityRegistrationApplicationService* which marks actually registration of your activities and events.

During this activity the entries are made in table `FLX_EP_ACT_LOG_B` and `FLX_EP_EVT_LOG_B` with appropriate comments depending on type of Alerts whether it is Mandatory (M) or Customer Subscribed (S).

There is one flag maintained in `FLX_EP_EVT_LOG_B` viz. `FLG_PROCESS_STAT`, which specifies status of event.

In this step various validations are also performed such as checking if email Id of recipient is mentioned and so on.

However, the final processing of alerts is managed in 'Interaction.java' when it is about to close that is, call is made in 'manageLastInteraction'.

Figure 9–15 Alert Processing Steps

```

759  * @param manager
760  * @throws FatalException
761  */
762  private static void manageLastInteraction(InteractionWrapper interactionWrapper, TransactionStatus status) throws FatalEx
763
764      Interaction interaction = interactionWrapper.getInstance();
765      // This try block is specifically for clearInteraction
766      try {
767          try {
768              // Set the transaction reference number
769              if (interaction.transactionKeyMap.size() > 0) {
770                  String txnRefno = (String) interaction.transactionKeyMap.keySet().iterator().next();
771                  status.setInternalReferenceNumber(txnRefno);
772                  String bankCode = (String) FCRTThreadAttribute.get(FCRTThreadAttribute.USER_BANK);
773                  String branchCode = (String) FCRTThreadAttribute.get(FCRTThreadAttribute.TRANSACTION_BRANCH);
774                  // Constants.MAX_VALID_OLTP_STATUS) {
775                  if (status.getReplyCode() < ErrorManager.MIN_COD_RESP_VOID) {
776                      TransactionKey key = (TransactionKey) interaction.transactionKeyMap.get(txnRefno);
777                      IAdapterFactory csAdapterFactory = AdapterFactoryConfigurator.getInstance()
778                          .getAdapterFactory(CommonAdapterFactoryC
779                      ICoreApplicationServiceAdapter coreServiceAdapter = (ICoreApplicationServiceAdapter) csAdapterFactory
780                      // status.getIsOverridden()) {
781                      BranchDatesDefinitionInquiryResponse response = coreServiceAdapter.fetchBranchDates((SessionContext)
782                          bankCode,
783                          branchCode);
784                      storeTransactionReference(txnRefno, key, new Date(key.getTransactionDate()), response.getBranchDatesI
785                  }
786              }
787              //
788              //For 2PC JMS transactions have to be posted prior to the running transaction commit , for thread mode let th
789              //and then spawn off EP processing threads
790              if (ConfigurationFactory.getInstance().getConfigurations(ALERT_POLLER_POOL).get(POLLER_TYPE, JDK_POOL).equals(A
791                  processActivityEvents();
792                  rollbackOrCommit(interaction, status);
793              } else {
794                  rollbackOrCommit(interaction, status);
795                  processActivityEvents();
796              }
797          } catch (FCRInfraException e) {
798              ErrorManager.logException(fetchCurrentTask(), e);
799              ErrorManager.throwFatalException(e);
800          } finally {
801              DataAccessManager.getManager().closeSession();
802              checkUnclosedSessions(interactionWrapper);
803          }
804      } finally {

```

EventProcessStatusType

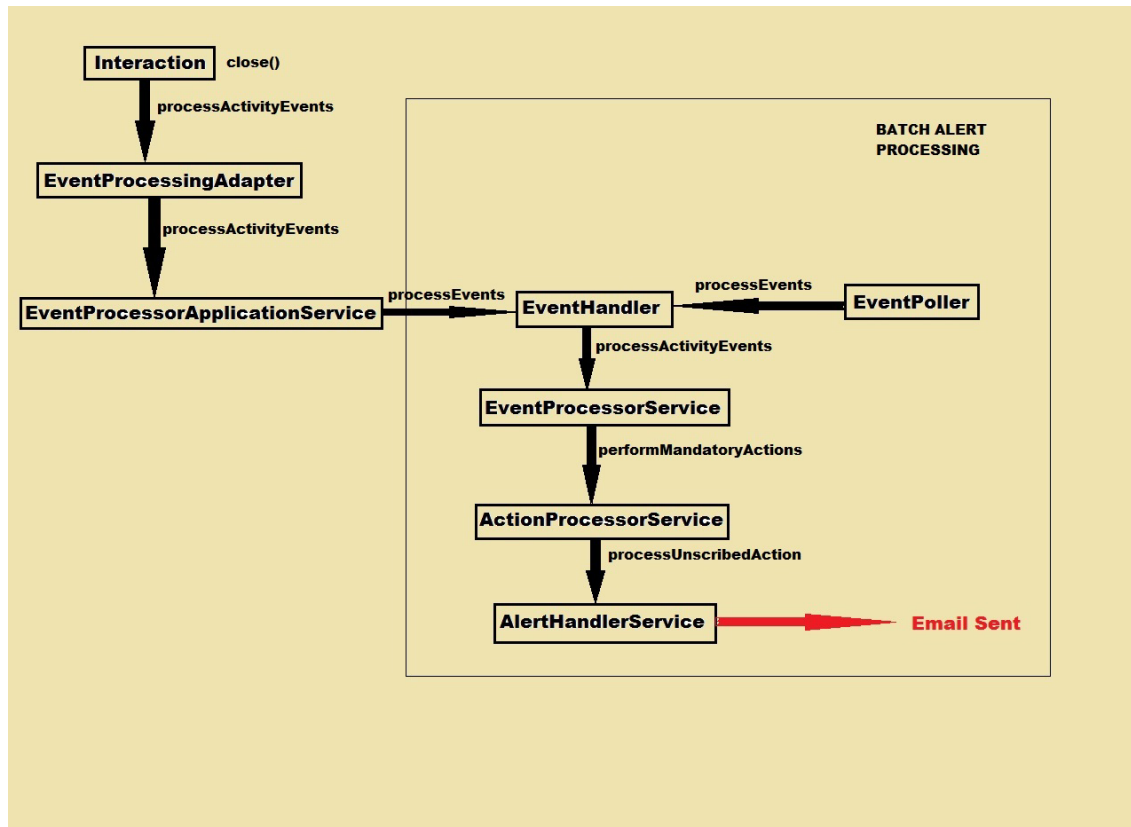
This shows status of event throughout cycle of event processing from Registration of event to Dispatch of Alert. (It is maintained in FLX_EP_EVT_LOG_B table as "flg_process_stat").

The various statuses of events are as follows:

- GENERATED("G")
- COMPLETED("C")
- NO_SUBSCRIPTION("N")
- ABORTED("A")
- INITIATED("I")
- REINITIATED("R")

For any event online or batch, when it is logged for first time it is marked as Generated "G" in flx_ep_evt_log_b table.

Figure 9–16 Event Processing Status Type

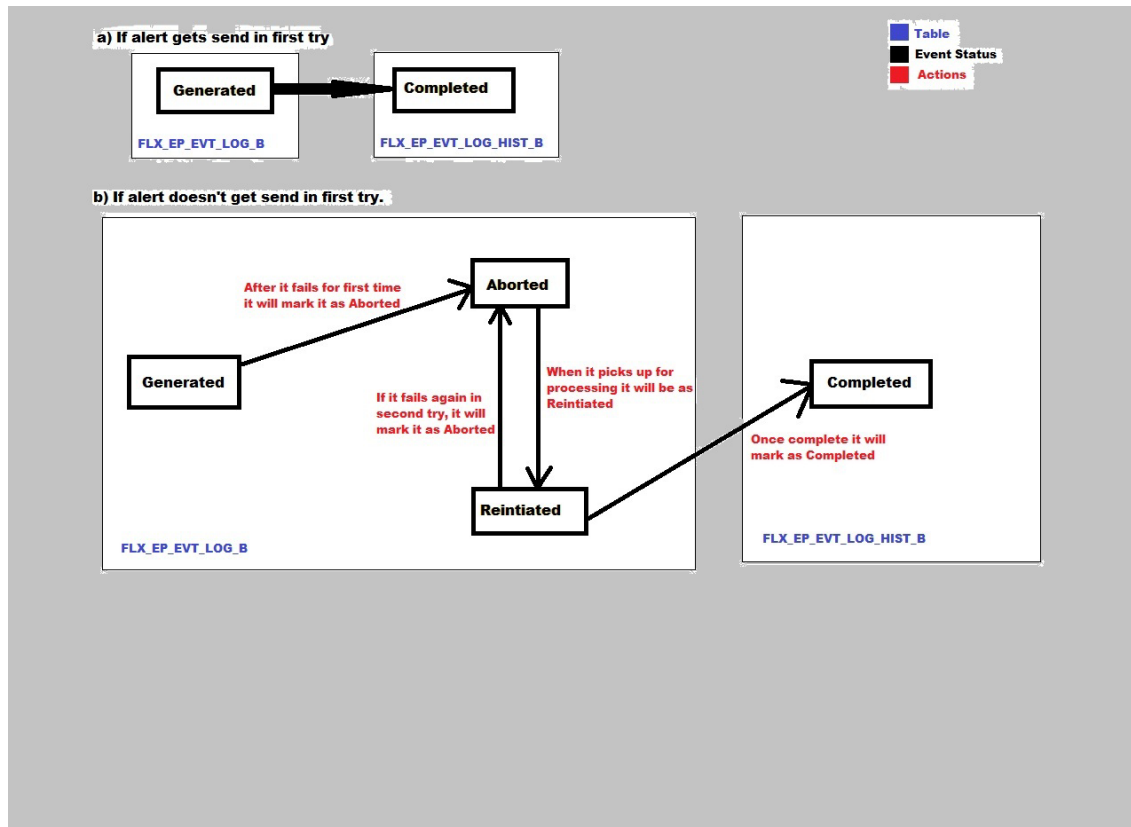


JMS (Java Messaging Service) is used for dispatch of alerts.

For Online Alerts:

- **Direct Approach:** If alert gets send in first try, flg_process_stat is as "G" in FLX_EP_EVT_LOG_B and alert is dispatched through JMS, and then entry for that event record is moved to FLX_EP_EVT_LOG_HIST_B and flg_process_stat is marked as "C".
- **EventPoller:** If alert gets failed in first retry it will mark status as "R". In this case EventPoller will pick the failed event and complete its processing and mark status as "A" and then entry for that event record is moved to FLX_EP_EVT_LOG_HIST_B and flg_process_stat is marked as "C".
- **For Batch Alerts:** In case of batch alerts as no Interaction.close() is called, the direct approach is not used in Batch Alerts. In this case only EventPoller approach is used.

Figure 9–17 Batch Alerts



9.4 Alert Dispatch Mechanism

The dispatch mechanism is triggered by the *AlertHandlerService* for dispatching subscribed actions of type *Alert*. The processing is implemented as part of the respective handlers. The handler services delegate the call to the *Dispatcher* based on the type of *DestinationType* configured in the *Recipient* at the time of *ActivityEventAction* maintenance which involves *RecipientMessageTemplate* setup.

The module provides definition of multiple dispatch detail configurations on the basis of *SubscriberType* and various configuration parameters like *UrgencyType*, *ImportantType* in the *AlertTemplate*.

The dispatcher uses the *DispatchDataConverter* to convert the data captured as part of activity registered in the system into data which can be dispatched to the target subscriber.

Figure 9–18 Alert Dispatch Mechanism

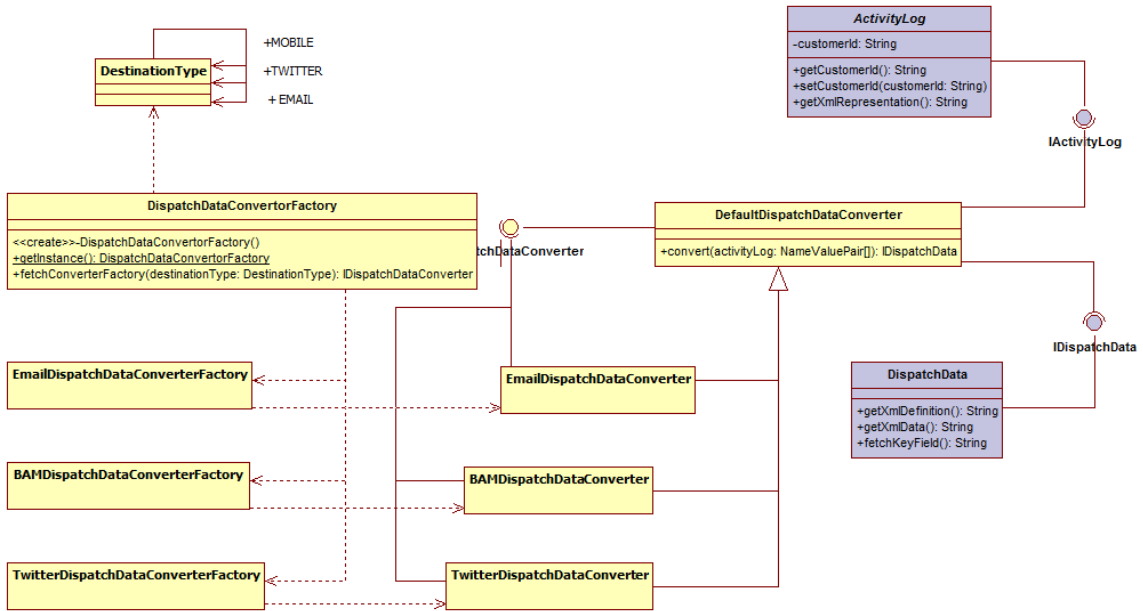


Figure 9–19 Alert Dispatch Mechanism - Dispatcher Factory

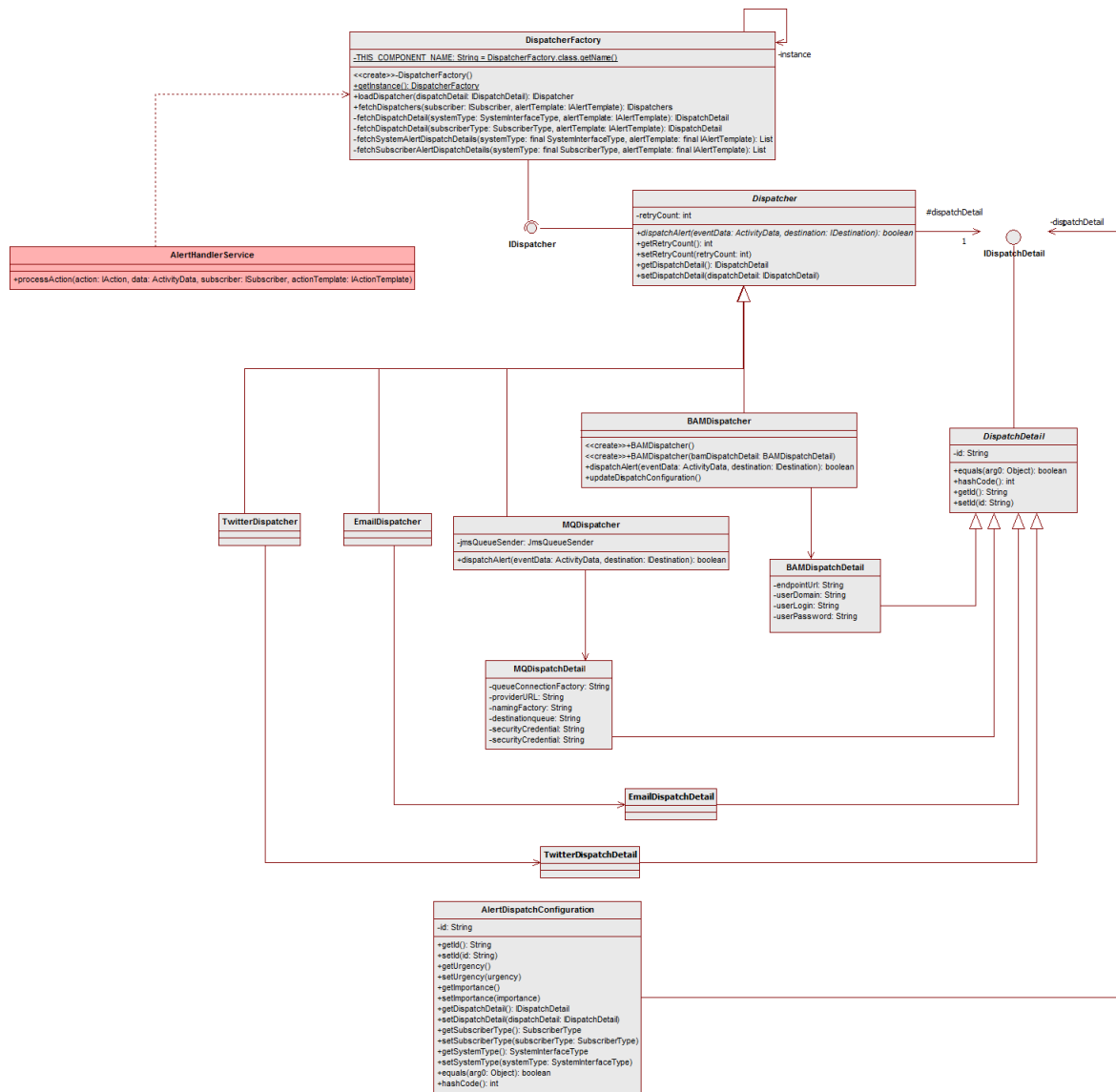
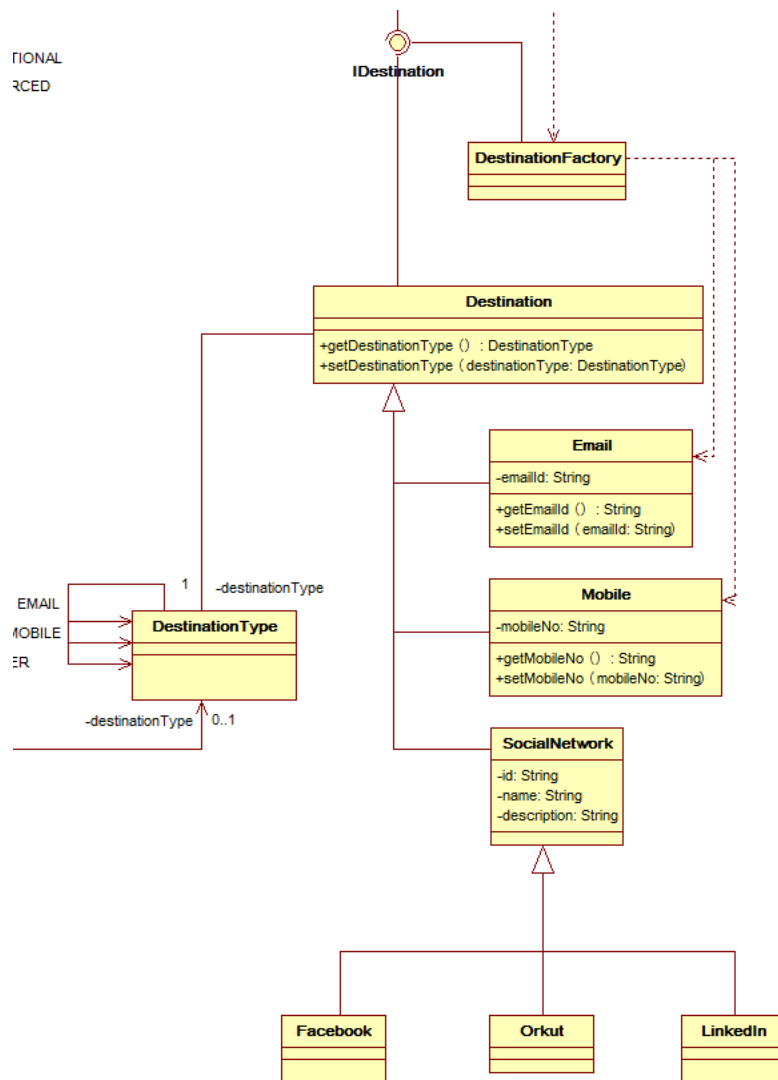


Figure 9–20 Alert Dispatch Mechanism - Destination



The various Destination Types are coded as per the above diagram. This existing framework makes it further extensible as per the requirements that is, you can add more destination types.

9.5 Adding New Alerts

To add a new alert:

1. Implement the Service Extension Interface for the application service of the method for which alert is to be raised.
2. Use either the `preServiceMethod()` or `postServiceMethod()` hook for the method in the implemented service extension class depending on the requirement.
3. The method should call the `registerActivityAndGenerateEvent()` of the `EventProcessingAdapter` class. In case a custom adapter is required the custom adapter method should call

registerActivityAndGenerateEvent() of ActivityRegistrationApplicationService.

4. New Activity ID, Event ID and implementation of IActivityLogDTO have to be created.

9.5.1 New Alert Example

This example will explain the above points in detail.

Use Case: A new alert has to be added after updating a party name.

The class PartyNameApplicationService has a method updateIndividualName() that does this activity.

Create the extension class, say PartyNameApplicationServiceExt, for this application service by implementing its extension interface IPartyNameApplicationServiceExt. Since the alert should be raised after updation of party name we will use the postUpdateIndividualName() method.

Within the method a call to registerActivityAndGenerateEvent() in EventProcessingAdapter should be made.

Code snippet for the call:

```
com.ofss.fc.app.adapter.IAdapterFactory adapterFactory =
AdapterFactoryConfigurator.getInstance().getAdapterFactory
(ModuleConstant.EVENT_PROCESSING);
IEventProcessingAdapter adapter = (IEventProcessingAdapter)
adapterFactory.getAdapter(EventProcessingAdapterConstant.MODULE_TO_
ACTIVITY);
adapter.registerActivityAndGenerateEvent(applicationContext,
activityId, eventId, new Date(), activityLog);
```

In case a new customer adapter has to be used, a call to registerActivityAndGenerateEvent() in ActivityRegistrationApplicationService should be made from within the adapter. A class called ActivityEventKeyDTO is used which captures the event ID and activity ID.

Code snippet for the call:

```
ActivityRegistrationApplicationService activityManager = new
ActivityRegistrationApplicationService();
ActivityEventKeyDTO activityEventKeyDTO = new ActivityEventKeyDTO
();
activityEventKeyDTO.setActivityId(activityID);
activityEventKeyDTO.setEventId(eventID);
ActivityRegistrationResponse response =
activityManager.registerActivityAndGenerateEvent
(sessionContext, activityEventKeyDTO, eventProcessingDate,
activityLog);
```

The signature for the method is:

```
public String registerActivityAndGenerateEvent(ApplicationContext
applicationContext,
String activityID,
String eventId,
Date eventProcessingDate,
Object logObject) throws FatalException;
```

Create new activityID, eventID and logObject to be passed to this method.

ActivityID and EventID as explained in detail in the above section have to be added in the following database tables. If data is not added in the tables, a runtime exception will occur while displaying the alert.

FLX_EP_ACT_B stores all the recognized activities.

FLX_EP_EVT_B stores all the recognized events.

FLX_EP_ACT_EVT_B which stores the mapping between all activities and events.

The Activity ID denotes the actual action that should raise the event within the application service and hence for ease of understanding it should ideally be the fully qualified name of the method.

Eg.com.ofss.fc.app.party.service.contact.PartyNameApplicationService.updateIndividualName

The Event ID can be anything that denotes the event

For example, UPDATED_PARTY_NAME

The logObject is an implementation of IActivityLogDTO. For the new alert a new implementation has to be created. The DTO should have fields mapped to the placeholders in the new alert to be added

For example, for the alert "Your name has been updated from #previous_Name# to #new_Name# successfully."

the following DTO has to be made. The variables have to map to the placeholders in the alert template.

```
public class PartyNameChangeLogDTO implements IActivityLogDTO {
    private static final long serialVersionUID = -3492413059506052931L;
    private String updatedName;
    private String registeredOldName;
    //getters and setters for the variables
}
```

The DTO has to be populated with relevant data

```
E.g... private IActivityLog
populateActivityLogForIndividualPartyNameChange() {
    PartyNameChangeLogDTO activityLog = new PartyNameChangeLogDTO();
    activityLog.setUpdatedName("Andrew Matthew");
    activityLog.setRegisteredOldName("Andy Matthew");
    return activityLog;
}
```

9.5.2 Testing New Alert

JUnit test cases can be used to test the alert created by supplying sample input data. The example below shows how the above new alert can be tested.

```
public void testPartyUpdateName() throws IOException {
    String testCase = "PartyUpdateName";
    ActivityRegistrationApplicationService
    activityRegistrationApplicationService
    = new ActivityRegistrationApplicationService();
    ActivityEventKeyDTO activityEventKeyDTO = new ActivityEventKeyDTO
    ("com.ofss.fc.app.party.service.contact.
    PartyNameApplicationService.updateIndividualName ", " UPDATED_PARTY_
    NAME");
}
```

```

Date date = new Date();
SessionContext sessionContext = getSessionContext();
com.ofss.fc.app.party.dto.alert.PartyNameChangeLogDTO activityLog
= new com.ofss.fc.app.party.dto.alert.PartyNameChangeLogDTO ();
activityLog.setUpdatedName("Andrew Matthew");
activityLog.setRegisteredOldName("Andy Matthew");
try{
ActivityRegistrationResponse response
=
activityRegistrationApplicationService.registerActivityAndGenerate
Event(
sessionContext, activityEventKeyDTO, date, activityLog);
TransactionStatus result= response.getStatus();
dumpTransactionStatus("ActivityRegistrationApplicationService", "
testPartyUpdateName ", result);
logger.log(Level.FINER, "The ErrorCode is: "+ result.getErrorCode
());
} catch (FatalException e) {
logger.log(Level.SEVERE,"FatalException from"+THIS_COMPONENT_
NAME+". testPartyUpdateName ",e);
fail("Unexpected failure from " + THIS_COMPONENT_NAME + ".
testPartyUpdateName ");
}
}

```

For testing with the JUnit test cases we need to update the PoolType property in the AlertPollerPool.properties as follows:

```
PoolType=JDK
```

The value should be JDK for testing with JUnit (standalone application) and JMS if the application is run on WebLogic server.

9.6 Support For Derived Facts

Alerts are generated by assigning values to Facts that are mapped to the Alert Message Template placeholders.

These values are derived from the ActivityLog attributes based on the seed data that maintains the mapping information between the ActivityLog attributes and the Facts.

In Facts Module there is a provision to co-relate different Facts and derive the value of one Fact based on the value of the related Fact. This is done by maintaining the relationship in certain Fact tables.

The same support for Derived Facts has been included in Alerts framework.

For example, to add Party First Name information to an Alert this Fact has to be defined.

The following inserts are used to create this Fact with the name Alert.Party.FirstName.

Figure 9–21 Alert.Party.FirstName

```

Insert into flx_fa_facts_b
(FACT_CODE,FACT_NAME,FACT_DESC,DOMAIN_CODE,DOMAIN_CATEGORY_CODE,VALUE_TYPE,BUSINESS_TYPE_CODE,FACT_VALUE_REGEX,FACT_VAL_ERR_MSG,RETRIEVAL_KEY,CREATED_
BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATE_DATE,OBJECT_VERSION_NUMBER,LAST_UPDATE_LOGIN,FACT_CLASS,SHORT_NAME,DOMAIN_OBJECT_EXTN)
values ('Alert.Party.FirstName','Alert Party FirstName', 'FirstName', 'Banking', 'Alert', 'Open', 'Alphanumeric', null, null,
'Alert.Party.FirstName', 'manojpalk',to_timestamp('02-MAR-11 12.20.18.199000000 PM','DD-MON-RR HH.MI.SS.FF AM'),'manojpalk',to_timestamp('02-MAR-11
12.20.18.199000000 PM','DD-MON-RR HH.MI.SS.FF AM'),1,null,'NonFinancial','Alert.Party.FirstName','CZ');

Insert into flx_fa_group_xref (FACT_CODE, GROUP_CODE, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_DATE, OBJECT_VERSION_NUMBER,
LAST_UPDATE_LOGIN) values ('Alert.Party.FirstName','Alert',null,null,null,null,null,null);

```

In Alerts framework, the facts that are available by default are:

Figure 9–22 Facts in Alerts Framework

```

1 Alert.MultiEntity.LegalEntity.Code denoting Legal Entity Code
2 Alert.MultiEntity.LegalEntity.Name denoting LegalEntity Name
3 Alert.MultiEntity.MarketEntity.Code denoting Market Entity Code
4 Alert.MultiEntity.MarketEntity.Name denoting Market Entity Name
5 Alert.MultiEntity.BusinessUnit.Code denoting Business Unit Code
6 Alert.MultiEntity.BusinessUnit.Name denoting Business Unit Name
7 Alert.Submission.SourcingEntityType denoting Sourcing Entity Type
8 Alert.Submission.SourcingEntityId denoting Sourcing Entity Id
9 Alert.Party.PartyId denoting the Party Id
10

```

In addition to these Facts all the Facts that have been mapped with the Service Attributes of the Activity log for the Activity Id of the Alert are available to the Alerts Framework for usage.

Facts that can be derived from any of the above Facts can be added to this list.

To relate and derive value of Alert.Party.FirstName with the help of available Fact Alert.Party.PartyId, the relationship information and value derivation logic must be maintained in the Facts tables.

Figure 9–23 Alert.Party.PartyId

```

Insert into flx_fa_value_bindings (FACT_CODE,PARAM_NAME,DATA_TYPE_CODE,PARAM_DESC,VARIABLE_BASED_FACT,LITERAL_FACT_VALUE,BINDING_TYPE)
values ('Alert.Party.FirstName','partyId',null,null,'Alert.Party.PartyId',null,'Variable');

Insert into flx_fa_value_datasources
(FACT_CODE,DATA_SOURCE_CODE,JDBC_DERIVATION_QUERY,HQL_DERIVATION_QUERY,JAVA_DERIVATION_CLASS,DSN_CODE,DB_FUNCTION_NAME)
values ('Alert.Party.FirstName','HQL',null,'SELECT a.firstName FROM com.ofss.fc.domain.party.entity.individual.IndividualName a WHERE
a.id.partyNameType=com.ofss.fc.enumeration.PartyNameType.Legal and a.id.partyId = :partyId',null,null,null);

```

FLX_FA_VALUE_BINDINGS defines the relationship and FLX_FA_VALUE_DATASOURCES defines the data derivation logic.

Similarly, additional derived Facts: Alert.Party.Prefix and Alert.Party.LastName can be maintained.

Figure 9–24 Alert.Party.Prefix and Alert.Party.LastName

```

For 'Alert.Party.Prefix':-
Insert into flx_fa_value_bindings (FACT_CODE,PARAM_NAME,DATA_TYPE_CODE,PARAM_DESC,VARIABLE_BASED_FACT,LITERAL_FACT_VALUE,BINDING_TYPE)
values ('Alert.Party.Prefix','partyId',null,null,'Alert.Party.PartyId',null,'Variable');

Insert into flx_fa_value_datasources (FACT_CODE,DATA_SOURCE_CODE,JDBC_DERIVATION_QUERY,HQL_DERIVATION_QUERY,JAVA_DERIVATION_CLASS,DSN_CODE,DB_FUNCTION_NAME)
values ('Alert.Party.Prefix','HQL',null,'SELECT b.name from com.ofss.fc.domain.party.entity.global.Prefix b where b.prefixKey.id in (SELECT
a.prefixPrimary FROM com.ofss.fc.domain.party.entity.identity.PartyName a WHERE a.id.partyNameType=com.ofss.fc.enumeration.PartyNameType.Legal and
a.id.partyId = :partyId)',null,null,null);

For 'Alert.Party.LastName':-
Insert into flx_fa_value_bindings (FACT_CODE,PARAM_NAME,DATA_TYPE_CODE,PARAM_DESC,VARIABLE_BASED_FACT,LITERAL_FACT_VALUE,BINDING_TYPE)
values ('Alert.Party.LastName','partyId',null,null,'Alert.Party.PartyId',null,'Variable');

Insert into flx_fa_value_datasources (FACT_CODE,DATA_SOURCE_CODE,JDBC_DERIVATION_QUERY,HQL_DERIVATION_QUERY,JAVA_DERIVATION_CLASS,DSN_CODE,
DB_FUNCTION_NAME) values ('Alert.Party.LastName','HQL',null,'SELECT a.lastName FROM com.ofss.fc.domain.party.entity.individual.IndividualName a WHERE
a.id.partyNameType=com.ofss.fc.enumeration.PartyNameType.Legal and a.id.partyId = :partyId',null,null,null);

```

Use and test the maintenance and generation of Alerts using Derived Facts.

Figure 9–25 Message Template (Fast Path: AL03)

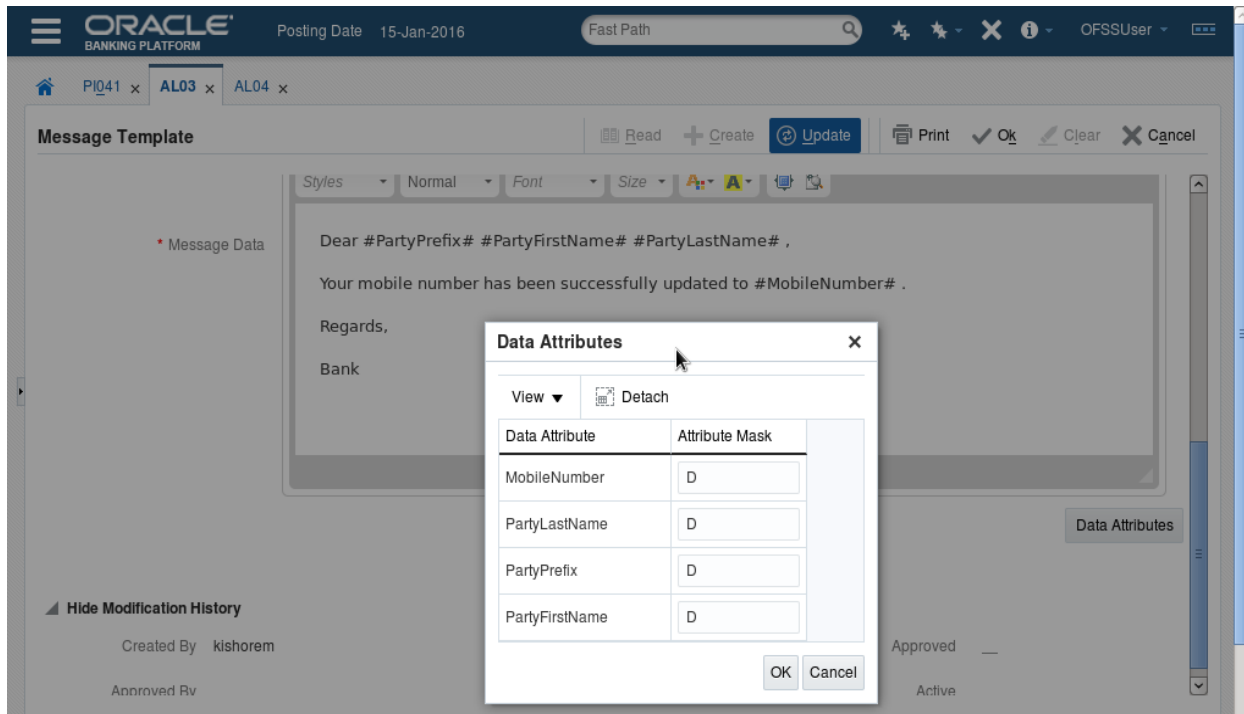
The screenshot shows the Oracle Banking Platform interface for maintaining a message template. The top navigation bar includes the Oracle logo, the text 'ORACLE BANKING PLATFORM', the posting date '15-Jan-2016', a search bar with 'Fast Path', and user information 'OFSSUser'. Below the navigation bar, there are tabs for 'PI041', 'AL03', and 'AL04'. The main content area is titled 'Message Template' and includes a toolbar with 'Read', 'Create', 'Update', 'Print', 'Ok', 'Clear', and 'Exit' buttons. The template details are as follows:

- Template ID: PartyUpdateMobileNO
- Template Name: PartyUpdateMobileNO
- Destination Type: EMAIL
- Subject Line: BANK_ALERT
- Message Data: Dear Customer,

The message content is displayed in a rich text editor with various formatting options like bold, italic, underline, font size, and color.

First, alter the existing Alert Message Template using the placeholder for the derived facts.

Figure 9–26 Placeholder for Derived Facts



Next, map the new Message Template placeholders in Alert Maintenance screen with the Derived Facts, which will also appear in the drop down of the Facts that are available to the Alerts Framework.

Figure 9–27 Alert Maintenance (Fast Path: AL04)

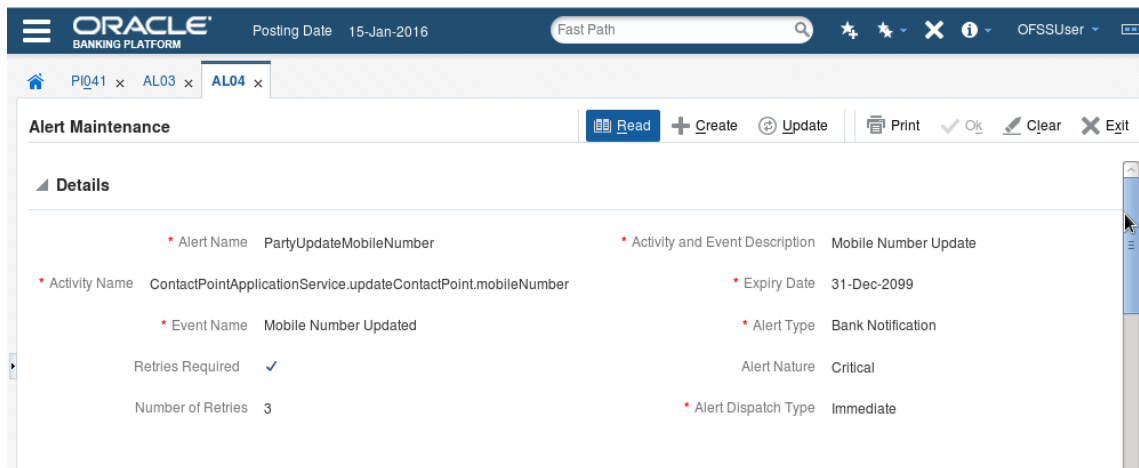


Figure 9–28 Alert Maintenance - Map the New Message Template Placeholders

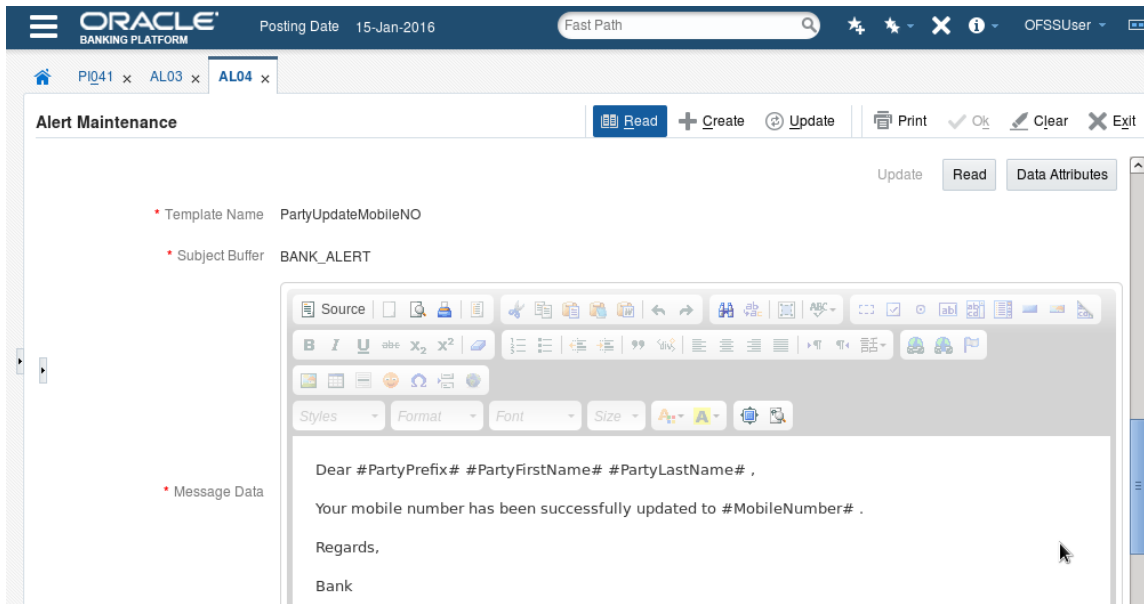


Figure 9–29 Alert Maintenance - Facts List

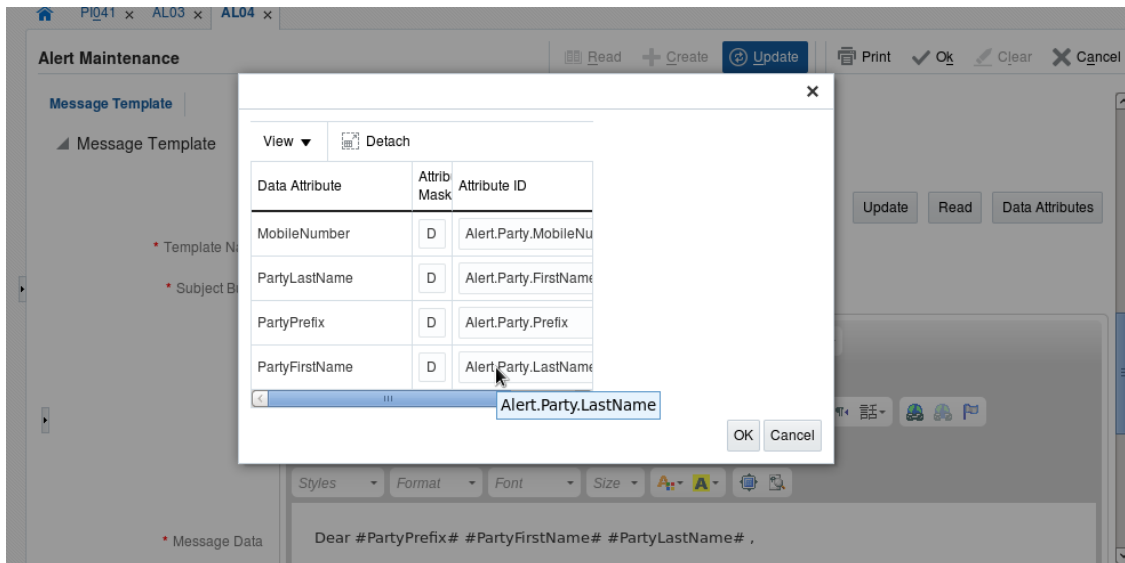
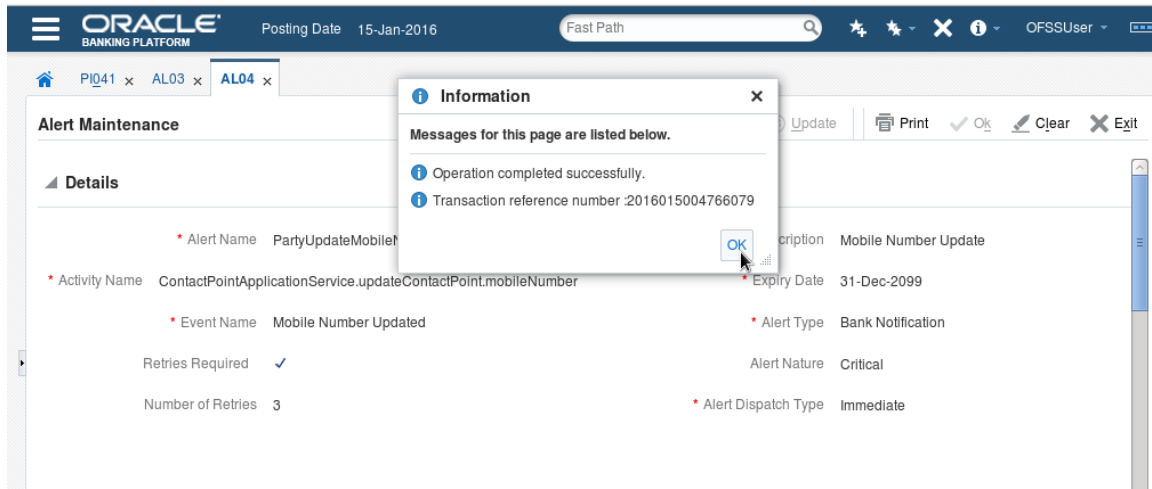
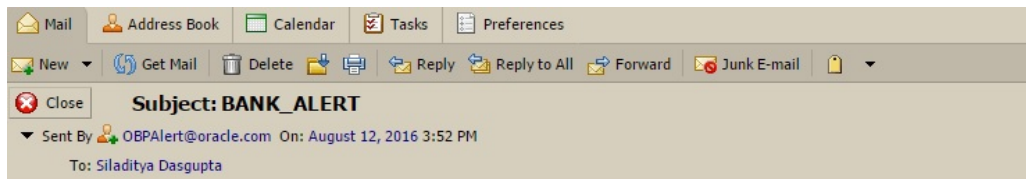


Figure 9–30 Alert Maintenance - Mapping Completed



Next, perform a Mobile Number updation from the Contact Point screen. This triggers the Alert that was altered earlier and the following mail is received.

Figure 9–31 Alert Mail on Mobile Number Update in Contact Point screen



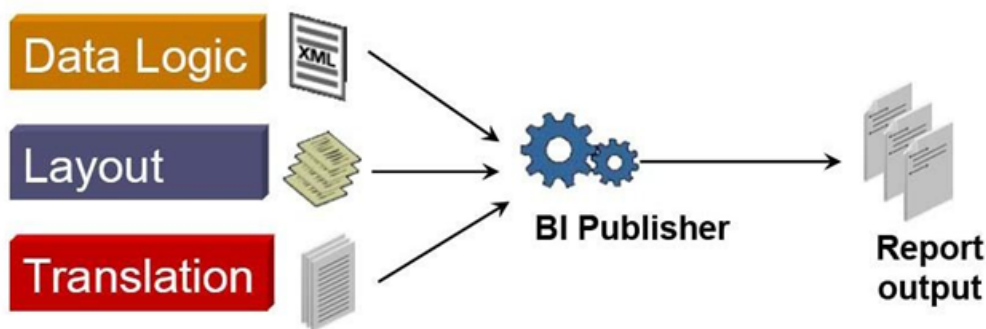
Dear Ms GLBL Piku .
 Your mobile number has been successfully updated to 1234567311 .
 Regards,
 Bank

The Alerts Framework has been able to substitute the place holders of the Message Template with the Fact values derived from Derived Fact derivation logic in Facts Framework.

10 Creating New Reports Using Oracle Analytics Publisher

Oracle Analytics Publisher is a standalone reporting and document output management solution that allows companies to lower the cost of ownership for reporting solutions. Its strength is that it separates the data model from the actual report formatting/layout. Oracle Analytics Publisher relies on 2 fundamental components to create reports, XML data and a template that represents the look and feel of the report. The XML data can be generated from any number of sources and Oracle Analytics Publisher makes accessing data in the proper format easy. Templates can be created in Microsoft Word and Adobe Acrobat allowing almost anyone familiar with these desktop applications the ability to create reports.

Figure 10–1 Creating New Reports



The following sections will give an overview of Oracle Analytics Publisher. The developer will be able to add and configure an *Adhoc* report to OBP using the Oracle Analytics Publisher.

Use Case: The OBP application has a batch framework using which a developer can easily add batch processes, also known as *batch shells*, to the application. The batch framework executes all the batch shells defined in the system as per their configuration. The results of these batch shell executions are stored in the database. We will be adding a report using Oracle Analytics Publisher for the execution results summary for batch shells.

10.1 Data Objects for the Report

The *Data Model* of the report invokes the database to fetch the data for the report through certain data objects that we will need to create. The primary data objects needed for the reports are as follows:

Global Temporary Table

You will need to create a *Global Temporary Table* based on the fields required for the report data. This table should mandatory have the field *SESSION_ID* of *NUMBER* type. The naming convention followed in OBP for the global temporary table's name is *RPT_<Module_Code>_R<Report_Number>*.

For the aforementioned use case, the script for creating the global temporary table would be as shown below.

Figure 10–2 Global Temporary Table

```

-- Global temporary table for the report
DROP TABLE RPT_PI_R007;
CREATE GLOBAL TEMPORARY TABLE RPT_PI_R007
(
  COD_SHELL                VARCHAR2(30),
  TXT_PROCESS_NAME         VARCHAR2(120),
  COD_PROC_CATEGORY        NUMBER(3),
  TXT_CATEGORY             VARCHAR2(20),
  DATE_RUN                 CHAR(8),
  STREAM_START_TIME        DATE,
  STREAM_END_TIME          DATE,
  PROCESSED_COUNT          NUMBER(38),
  COD_BRANCH_GROUP_CODE    VARCHAR2(10),
  EXECUTION_DURATION       NUMBER,
  SESSION_ID               NUMBER
)
on commit preserve rows;

```

Report Record Type

You will need to create a *Type* object with the fields present in the global temporary table. This type will represent a single row of data for the report. The naming convention followed in OBP for the report record type's name is *REP_REC_<Report_Id>*.

For the aforementioned use case, the script for creating the report record type would be as shown below.

Figure 10–3 Report Record Type

```

-- Record type for the report
CREATE OR REPLACE TYPE REP_REC_PI007 AS OBJECT
(
  COD_SHELL                VARCHAR2(30),
  TXT_PROCESS_NAME         VARCHAR2(120),
  COD_PROC_CATEGORY        NUMBER(3),
  TXT_CATEGORY             VARCHAR2(20),
  DATE_RUN                 CHAR(8),
  STREAM_START_TIME        DATE,
  STREAM_END_TIME          DATE,
  PROCESSED_COUNT          NUMBER(38),
  COD_BRANCH_GROUP_CODE    VARCHAR2(10),
  EXECUTION_DURATION       NUMBER,
  SESSION_ID               NUMBER
);

```

Report Table Type

You will need to create a *Type* object which will be a table of the previously created report record type. This type will represent the set of rows of data for the report. The naming convention followed in OBP for the report table type's name is *REC_TAB_<Report_Id>*.

For the aforementioned use case, the script for creating the report table type would be as shown below.

Figure 10–4 Report Table Type

```
-- Table type for the report
CREATE OR REPLACE TYPE REP_TAB_PI007 AS TABLE OF REP_REC_PI007;
```

Report DML Function

You will need to create a DML function which will be invoked to populate the previously created global temporary table with the data required to be displayed in the report. This function can have parameters as per the developer's requirements with filtering the data or inserting additional data. The naming convention followed in OBP for the report DML function's name is *AP_DML_<Report_Id>*.

For the aforementioned use case, the script for the report DML function would be as shown below.

Figure 10–5 Report DML Function

```
-- DML function for the report
CREATE OR REPLACE FUNCTION AP_DML_PI007(var_l_session_id IN NUMBER,
                                         var_bank_code IN VARCHAR2,
                                         var_cod_shell IN VARCHAR2)
RETURN NUMBER AS
var_l_cod_shell VARCHAR2(30);
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
-- input parameter
IF (var_cod_shell IS NULL or length(trim(var_cod_shell)) = 0) THEN
var_l_cod_shell := '%';
ELSE
var_l_cod_shell := var_cod_shell;
END IF;

--delete existing data for the session
DELETE FROM RPT_PI_R007 WHERE SESSION_ID = var_l_session_id;

--insert data into the table
INSERT INTO RPT_PI_R007
(COD_SHELL, TXT_PROCESS_NAME, COD_PROC_CATEGORY, TXT_CATEGORY, DATE_RUN, STREAM_START_TIME,
STREAM_END_TIME, PROCESSED_COUNT, COD_BRANCH_GROUP_CODE, EXECUTION_DURATION, SESSION_ID)
SELECT DISTINCT
BJSR.COD_SHELL, BJSR.TXT_PROCESS_NAME, BJSR.COD_PROC_CATEGORY, BJCM.TXT_CATEGORY, BJSR.DATE_RUN, BJSR.STREAM_START_TIME,
BJSR.STREAM_END_TIME, BJSR.PROCESSED_COUNT, BJSR.COD_BRANCH_GROUP_CODE, BJSR.EXECUTION_DURATION, var_l_session_id
FROM
FLX_BATCH_JOB_SHELL_RESULTS BJSR, FLX_BATCH_JOB_SHELL_MASTER BJSM,
FLX_BATCH_JOB_CATEGORY_MASTER BJCM, FLX_BATCH_JOB_BRN_GRP_MAPPING BJBGH
WHERE
BJSR.COD_SHELL = BJSM.COD_EOD_PROCESS AND BJSR.COD_SHELL LIKE var_l_cod_shell AND
BJSM.COD_PROC_CATEGORY = BJCM.COD_PROC_CATEGORY AND BJSM.COD_BRANCH_GROUP_CODE = BJBGH.BRANCH_GROUP_CODE AND
BJBGH.BANK_CODE = var_bank_code
ORDER BY DATE_RUN;

--commit
COMMIT;
RETURN 0;

EXCEPTION
WHEN OTHERS THEN ORA_RAISERROR(SQLCODE, 'Execution of AP_DML_PI007 failed', 500);
END;
```

Report DDL Function

You will need to create a DDL function which will be invoked to fetch data required to be displayed in the report from the global temporary table and wrap it in the previously created report table type. The naming convention followed in OBP for the report DDL function's name is *AP_DDL_<Report_Id>*.

For the aforementioned use case, the script for creating report DDL function would be as shown below.

Figure 10–6 Report DDL Function

```

-- DDL function for creating the report
CREATE OR REPLACE FUNCTION AP_DDL_PI007(var_bank_code IN VARCHAR2,
                                         var_cod_shell IN VARCHAR2)
RETURN REP_TAB_PI007 AS

v_ret          REP_TAB_PI007;
var_l_session_id NUMBER;
dml_function_result NUMBER;

BEGIN
  var_l_session_id := USERENV('SESSIONID');
  dml_function_result := AP_DML_PI007(var_l_session_id, var_bank_code, var_cod_shell);

  SELECT
    CAST
    (
      MULTISET
      (
        SELECT
          COD_SHELL, TXT_PROCESS_NAME, COD_PROC_CATEGORY, TXT_CATEGORY, DATE_RUN, STREAM_START_TIME,
          STREAM_END_TIME, PROCESSED_COUNT, COD_BRANCH_GROUP_CODE, EXECUTION_DURATION, SESSION_ID
        FROM RPT_PI_R007
        WHERE SESSION_ID = var_l_session_id
        ORDER BY DATE_RUN
      )
    )
  AS REP_TAB_PI007
  INTO v_ret
  FROM DUAL;

RETURN v_ret;

EXCEPTION
  WHEN OTHERS THEN ORA_RAISERROR(SQLCODE, 'Execution of AP_DDL_PI007 failed', 500);

END;

```

Data Model for the Report

Once you have created the data objects for the report in the database, you can start adding and configuring the report using Oracle Analytics Publisher. Log in to the Oracle Analytics Publisher application and follow these steps.

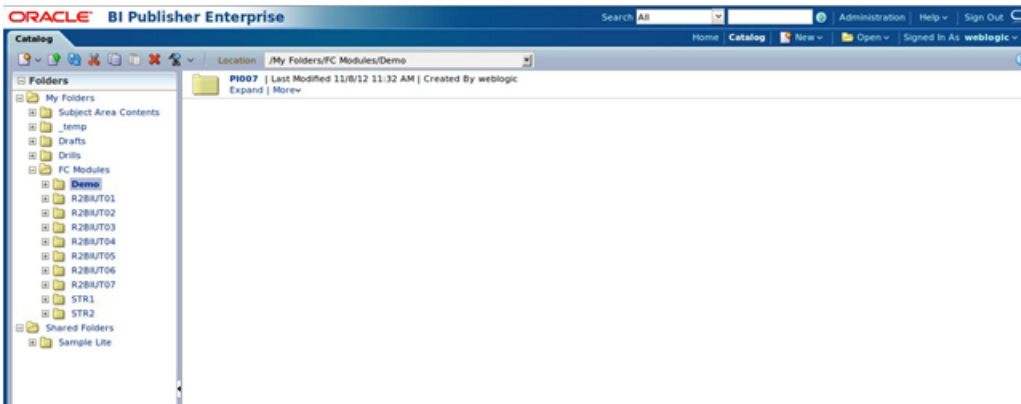
You can log in to the Oracle Analytics Publisher application deployed on `http://<IP ADDRESS><PORT>/xmlpserver/` with the credentials `weblogic/weblogic1`.

10.2 Catalog Folder

Before creating the data model or the layout for the report, you should create a folder to save the model and layout. You can find the link for the Catalog tab on the home screen. Click it and create a folder for your report at an appropriate location.

For the aforementioned use case, you can create a folder `PI007` at the location `/My Folders/FC Module/Demo` as shown below.

Figure 10–7 Catalog Folder

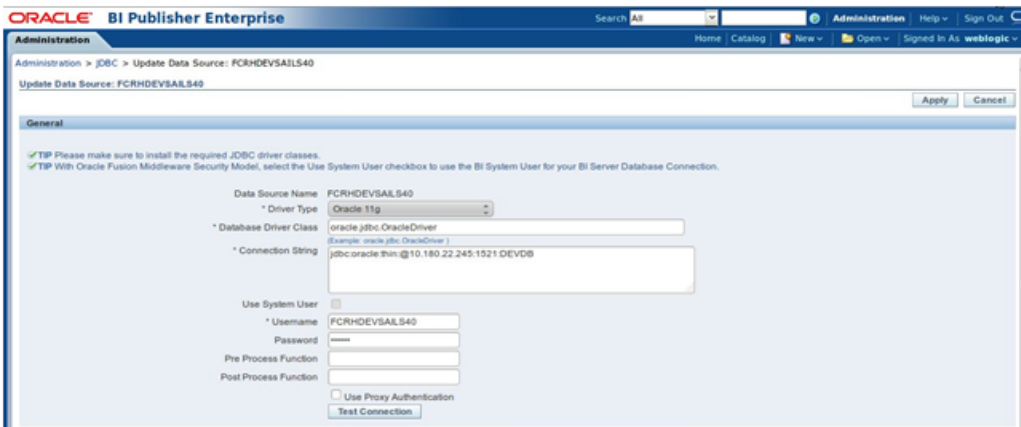


10.3 Data Source

You will need to add the data source from which the data will be fetched to be displayed in the report. The data source can be a *JDBC Connection*, *JNDI Connection*, *File*, *LDAP Connection* and so on. You can find the link for the *Administration* tab on the home screen. Click it and choose the appropriate data source connection type. Enter the required parameter values and validate the connection. Save the data source with an appropriate name.

For the aforementioned use case, you can add the JDBC Connection data source as show below.

Figure 10–8 Data Source



10.4 Data Model

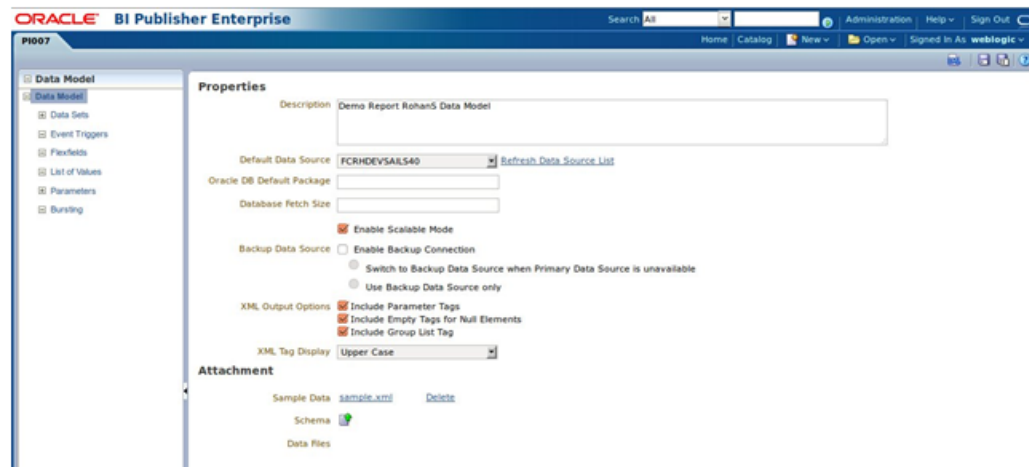
You will need to create a data model to back the report. This data model represents the report data fetched using the data objects and formatted into XML data. You can find the link to *Create Data Model* on the home screen of Oracle Analytics Publisher. Click it and follow these steps:

1. Enter an appropriate *description* for the data model.
2. Choose the previously created *data source* from the list displayed.

3. Check the Enable Scalable Model option.
4. Check the Include Parameter Tags option.
5. Check the Include Empty Tags for Null Elements option.
6. Check the Include Group List Tags option.
7. You can leave the rest of the options to default.

For the aforementioned use case, you can create data model as shown below.

Figure 10–9 Data Model



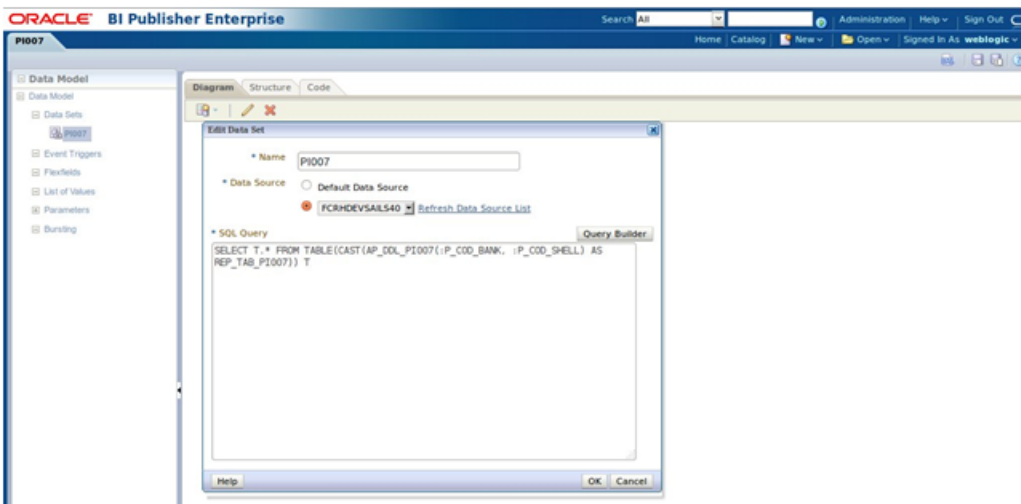
Data Set

After creating the data model, you will need to create a data set of the fields required to be displayed in the report. You can find the link for *Data Sets* on the left side pane of the screen. To create the data set, follow these steps:

1. In the Create Data Set icon, choose the option Create Data Set from SQL Query.
2. Enter an appropriate *name* for the data set.
3. Choose the previously created *data source* from the list displayed.
4. Enter the SQL query which will be used to fetch the data for the report. The results returned should be of the *Report Table Type* previously created.

For the aforementioned use case, you can create the data set as shown below.

Figure 10–10 Data Set



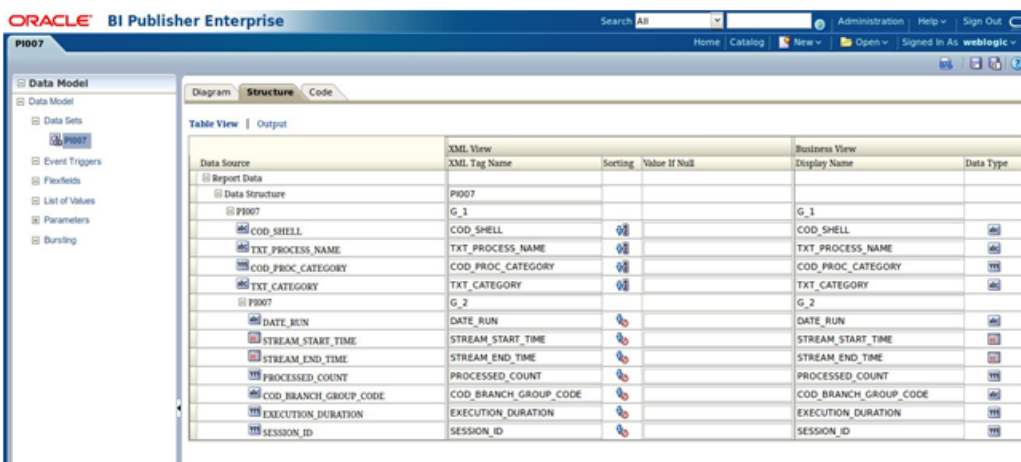
On click of OK, a data set will be created with all the fields as defined in the previously created *Report Record Type*.

You can group the fields as per the requirements of the report:

1. Select the field on which you want to group and choose *Group By*.
2. After creating a group, you can move fields between the groups.
3. You can also set field which will be used to sort the data displayed in a group.

For the aforementioned use case, you can group the fields as shown below.

Figure 10–11 Group Fields



You can view and edit the XML structure and labels of the report data in the *Structure* tab in a tabular format.

For the aforementioned use case, the structure would be as shown below:

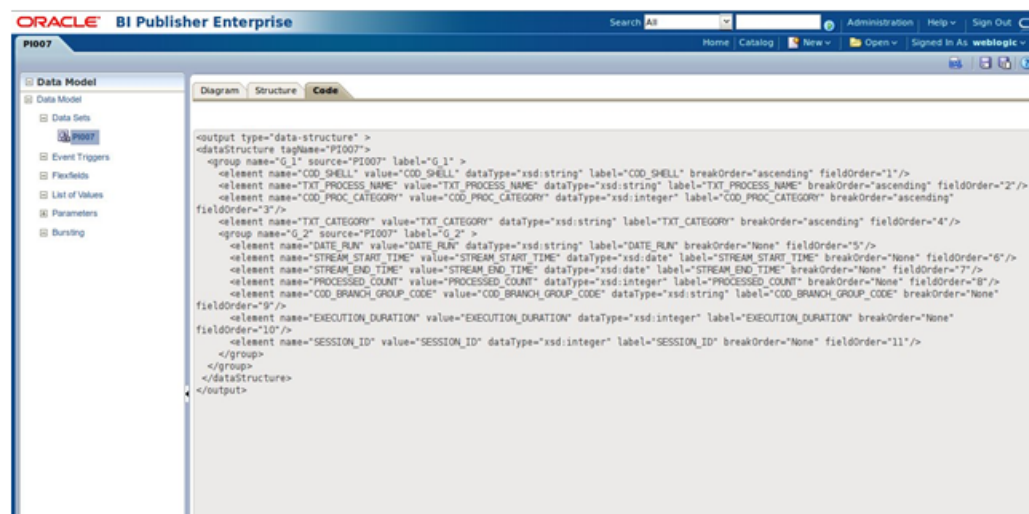
Figure 10–12 XML Structure and Labels



You can view the actual XML code in the *Code* tab.

For the aforementioned use case, the XML code would be as shown below.

Figure 10–13 XML Code



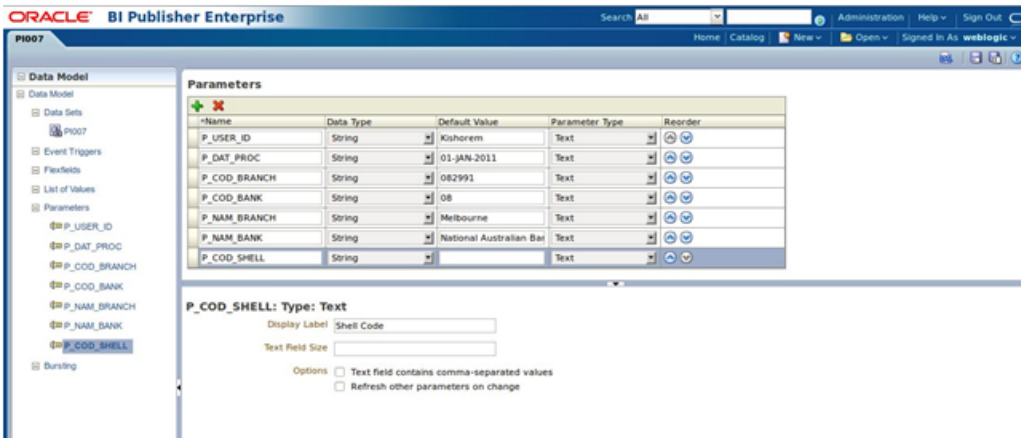
Input Parameters

You can define the *Input Parameters* required by the report in the *Parameters* tab present on the left hand side pane of the screen. To define input parameters, follow these steps:

1. In the **Parameters** tab, click the icon for *Add Parameter*.
2. Enter the name, type, display label and default value for the parameter.
3. Repeat the above steps to define as many parameters as required.

For the aforementioned use case, you can add parameters as shown below:

Figure 10–14 Add Input Parameters



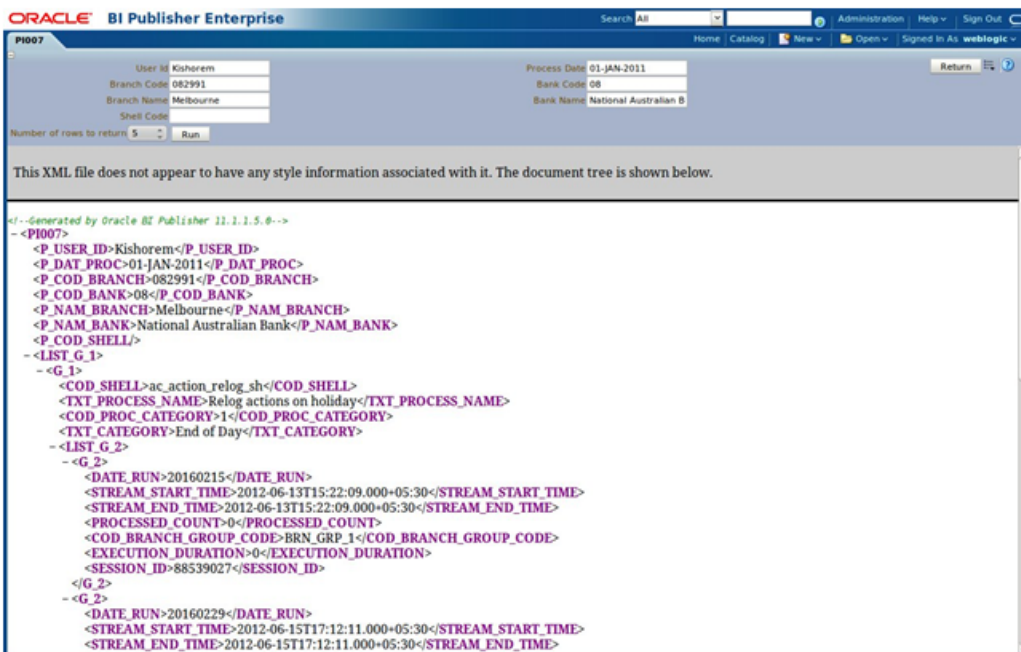
10.5 XML View of Report

After following the above steps, save the data model in the previously created catalog folder with an appropriate name. You can view the report without the layout in the XML form by clicking on the icon for *XML View*.

In the XML view, you will see input fields for the previously defined *input parameters*. Enter appropriate values in those fields and click *Run*. You will be able to see the XML representation of the report data.

For the aforementioned use case, the XML representation of the report data would be as shown below.

Figure 10–15 XML View of Report



10.6 Layout of the Report

A report needs to be presented in an appropriate format. The format can vary from report to report and client to client. Oracle Analytics Publisher separates the data model from the layout making it convenient for the developer.

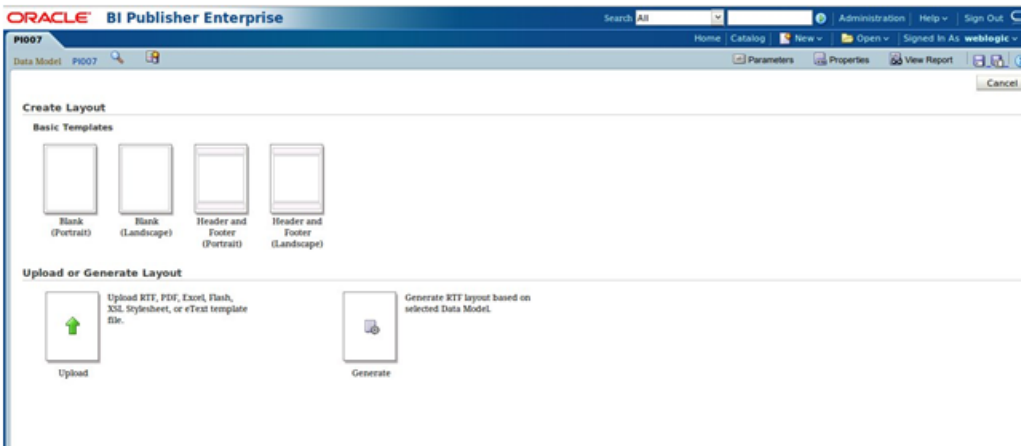
Anybody familiar with using Microsoft Word or Adobe Acrobat can use the corresponding plug-ins for these tools to create a layout for a report. You can create a rich layout using these standalone applications with Oracle Analytics Publisher plug-ins and then upload them to the Oracle Analytics Publisher application for use in your report.

The Oracle Analytics Publisher application can generate a very basic layout for your report from the data set. You can download the generated layout, modify it as per your layout requirements and upload it to the Oracle Analytics Publisher application for use in your report.

The Oracle Analytics Publisher application also allows the user to create a layout on the web. It has a rich set of tools to with drag and drop features and a ready link to the data set fields. You can create a layout in this fashion and use it in your report.

You can find the link to *Add New Layout* on the right side of the screen. Click it to get the options to *create*, *generate* or *upload* a layout.

Figure 10–16 Layout of the Report - Create Layout

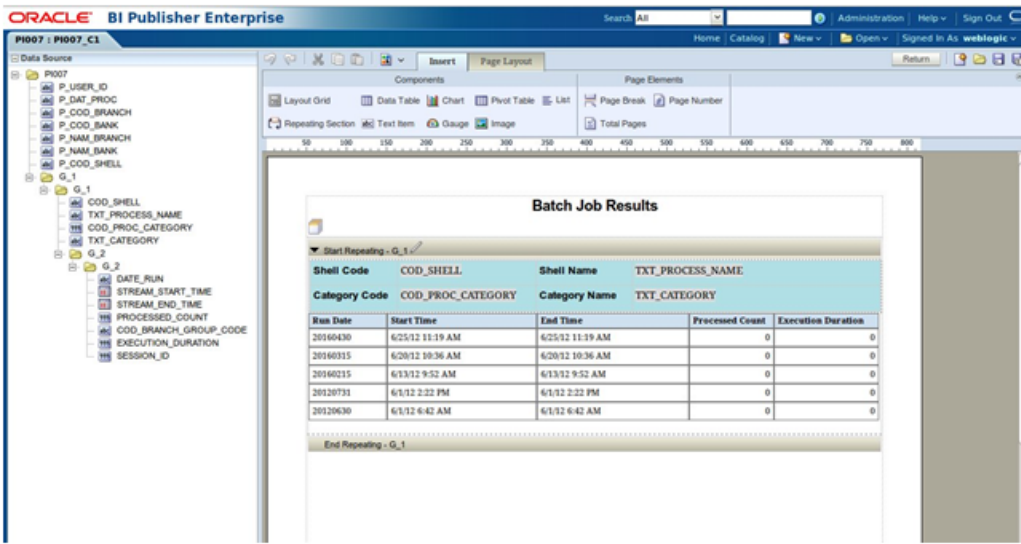


Choose from the *Basic Templates* to create a layout from a template. The layout editor screen will open. The previously created data set fields are present on the left pane of the screen. The toolbar present on top of the layout has tools to insert *Layout Grid*, *Data Table*, *Repeating Section*, *Text Item*, *List*, *Image*, *Page Break*, *Page Number*, elements.

You can drag and drop the layout and data set elements on to the layout as per your requirements. After making the required modifications, save the layout and return to the previous screen.

For the aforementioned use case, the layout for the report would be as shown below.

Figure 10–17 Layout of the Report - Batch Job Results



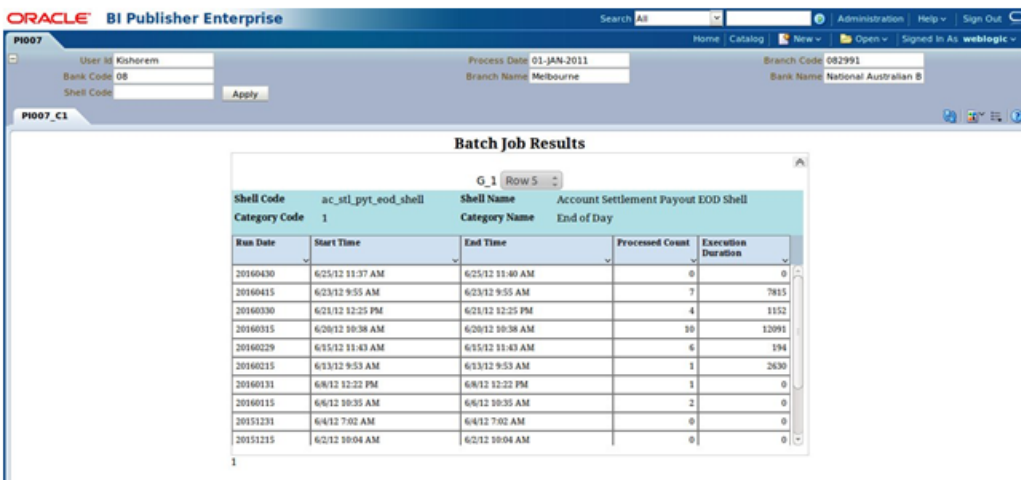
10.7 View Report in Oracle Analytics Publisher

After saving the *Data Model* and *Layout*, you can view the report in Oracle Analytics Publisher. Click the **View Report** link on the top right corner of the screen to open the report screen.

You will be able to see the input fields for the input parameters defined for the report. Enter appropriate values in these fields and click **Apply**. The report will be generated and displayed on the screen with the applicable data returned by the previously created *Data Model* and formatted as per the previously created *Layout*.

For the aforementioned use case, the final report would be as shown below.

Figure 10–18 View Report in Oracle Analytics Publisher



You can export the report in *HTML*, *PDF*, *Excel*, *RTF* or *PowerPoint* formats by clicking on the icon for *Export* on the right top corner of the screen and choosing the corresponding export option.

10.8 OBP Batch Report Configuration - Define the Batch Reports

Entries are required in three tables as given below to generate reports during EOD.

```
insert into FLX_BATCH_JOB_SHELL_MASTER (COD_EOD_PROCESS, TXT_
PROCESS, TXT_PROCESS_NAME, FRQ_PROC, DAT_LAST_RUN, DAT_SCHEDULED_
RUN, TXT_PROC_PARAM, COD_PROC_STATUS, NUM_PROC_ERROR, FLG_RUN_
TODAY, COD_PROC_CATEGORY, FLG_MONTH_END, FLG_MNT_STATUS, COD_MNT_
ACTION, COD_LAST_MNT_MAKERID, COD_LAST_MNT_CHKRID, DAT_LAST_MNT,
CTR_UPDAT_SRLNO, COD_MODULE, DAT_PROC_START, DAT_PROC_END, TXN_KEY,
SERVICE_KEY, NAM_COMPONENT, TYPE_COMPONENT, NAM_DBINSTANCE, RETRY_
COUNTER, NON_RETRY_COUNTER, COD_UNSTREAMED_PROCESS, COD_BRANCH_
GROUP_CODE)
values ('ch_eod_report_shell', 'CASA EOD Reports', 'CASA EOD
Reports', '1', to_date('15-02-2012', 'dd-mm-yyyy'), to_date('15-12-
2007', 'dd-mm-yyyy'), '99', 0, 0, 'Y', 1, 0, 'A', ' ', 'SETUP1',
'SETUP2', to_date('09-02-2002', 'dd-mm-yyyy'), 2, 'CH', to_date
('21-08-2008 09:54:57', 'dd-mm-yyyy hh24:mi:ss'), to_date('28-02-
2011 05:02:41', 'dd-mm-yyyy hh24:mi:ss'), 'DUMMY', 'execute',
'com.ofss.fc.bh.batch.BatchReportShellBean', 'B', 'PROD', 0, 0,
'ch_eod_report_shell', 'BRN_GRP_1');
```

Cod_proc_category = 1, for EOD; 2, for BOD and 16 for Internal System EOD

Nam_component is the same for all report shells.

Also we are using Branch_Group_Category = 'BRN_GRP_1' for all these report shells.

10.9 OBP Batch Report Configuration - Define the Batch Report Shell

```
Insert into FLX_BATCH_JOB_SHELL_DEPEND (COD_EOD_PROCESS, COD_REQD_
PROCESS, COD_PROC_CATEGORY, COD_REQD_PROC_CAT, FLG_MNT_STATUS, COD_
MNT_ACTION, COD_LAST_MNT_MAKERID, COD_LAST_MNT_CHKRID, DAT_LAST_
MNT, CTR_UPDAT_SRLNO, COD_BRANCH_GROUP_CODE)
Values ('ch_eod_report_shell', 'dd_eod_action', 1, 1, 'A', ' ',
'SETUP', 'SETUP', to_date('30-06-1995', 'dd-mm-yyyy'), 2, 'BRN_GRP_
1');
```

Here, in the first column is the report shell name and second is the name of the shell after which this shell should run. So 'ch_bod_report_shell' runs after 'dd_bod_action'. The remaining columns are self explanatory.

```
COD_PROC_CATEGORY=1 , for EOD; 2, for BOD and 16 for Internal
System EOD
COD_REQD_PROC_CAT=1, for EOD; 2, for BOD and 16 for Internal System
EOD
```

Also we are using Branch_Group_Category = 'BRN_GRP_1' for all these report shells.

10.10 OBP Batch Report Configuration - Define the Batch Report Shell Dependencies

```

Insert into flx_ba_report_ctrl (COD_REPORT_ID, FLG_REP_ADV, COD_
MODULE, NAM_REPORT, TYP_REPORT, FRQ_REPORT, FLG_PRINT, FLG_DELETE,
CTR_REP_COPIES, COD_PRIORITY, COD_ACCESS_LVL, COD_FILEID, BUF_INV_
VAR1, BUF_INV_VAR2, BUF_INV_VAR3, BUF_INV_VAR4, BUF_INV_VAR5, FLG_
MNT_STATUS, COD_MNT_ACTION, COD_LAST_MNT_MAKERID, COD_LAST_MNT_
CHKRID, DAT_LAST_MNT, CTR_UPDAT_SRLNO, FLG_SOURCE, FLG_SPLIT, FLG_
PROD_REP, COD_REPORT_DB_PREFIX, FLG_APPLY_SC, REF_UDF_NO, XPATH,
FLG_REPORT_SERVER)
values ('CH318', 'R', 'CH', 'CASA BALANCE LISTING', 'E', '1', '1',
'0', 1, 0, 0, 10047, ' ', ' ', ' ', ' ', ' ', ' ', 'A', ' ', 'PHASE_2',
'PHASE_2', to_date('01-11-1999', 'dd-mm-yyyy'), 2, 'P', 'Y', 'P',
'PROD', ' ', ' ', ' ', 'B');

```

Entry for each report should be here with typ_report = 'I' for Internal System EOD; 'E' for EOD and 'B' for BOD.

Currently, for EOD and BOD eod_report_shell and bod_report_shell will take care of all non CASA and TD EOD and BOD reports respectively.

No separate module specific shell is required during EOD and BOD. That is to mention Entry 3 alone is sufficient during EOD and BOD categories for any module. However, entries are needed for all three entries for batch report generation during any other category.

10.11 OBP Batch Report Configuration

This section describes the OBP batch report configuration.

10.11.1 Batch Report Generation for a Branch Group Code

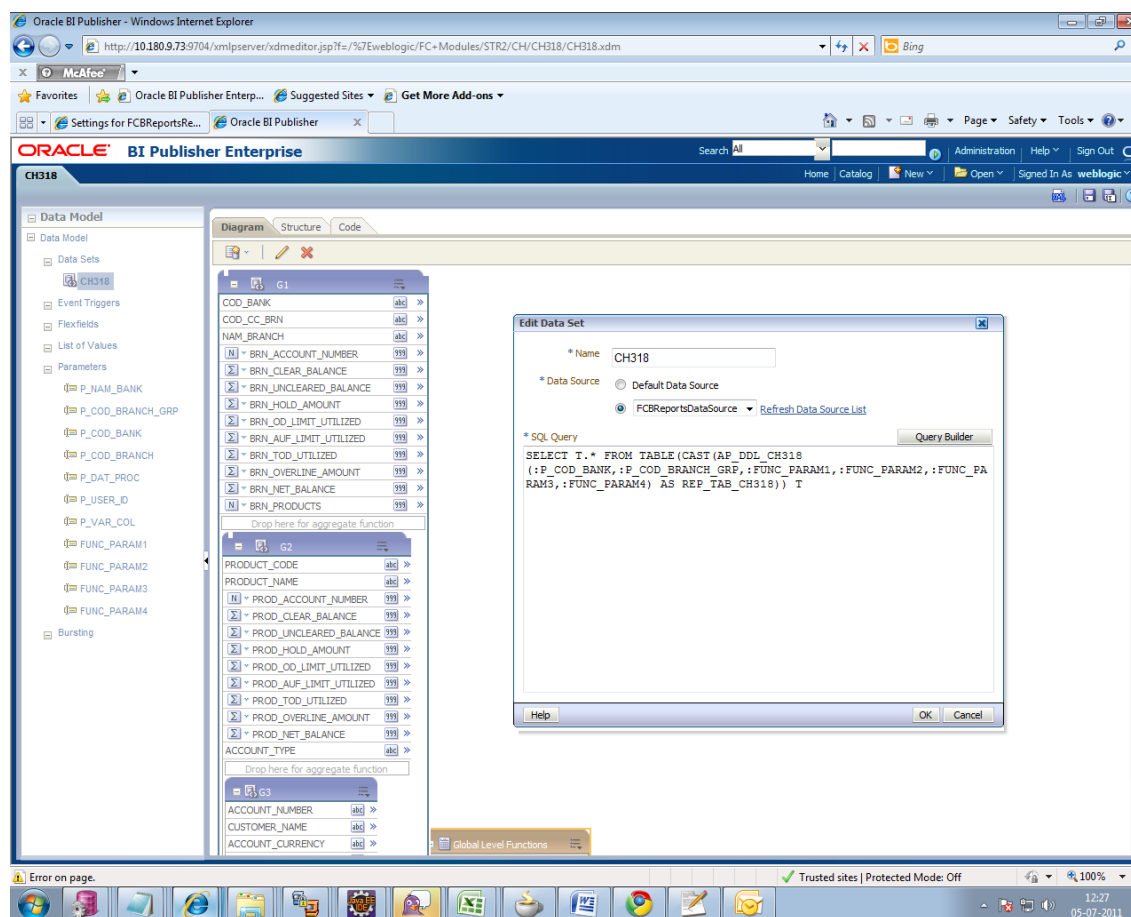
During Batch Process, a report should be generated for all branches linked to the respective Branch Group Code.

For any Batch Report to make use of the Branch Group Code getting passed by the application, a parameter 'P_COD_BRANCH_GRP' has to be defined in the Data Model.

The Data Model should pass this parameter to the Report Related DDL Function.

The Report Related DML Function filters all branch codes from FLX_BATCH_JOB_RESULTS_FILTERED that belong to the same Branch Group Code.

Figure 10–19 Batch Report Generation for a Branch Group Code



10.11.2 Batch Report Generation Status

At the end of all batch processes BA_REPORT_RESTART gets logged with the generated report status as D -> Done or F->Failed.

10.11.3 Batch Report Generation Path

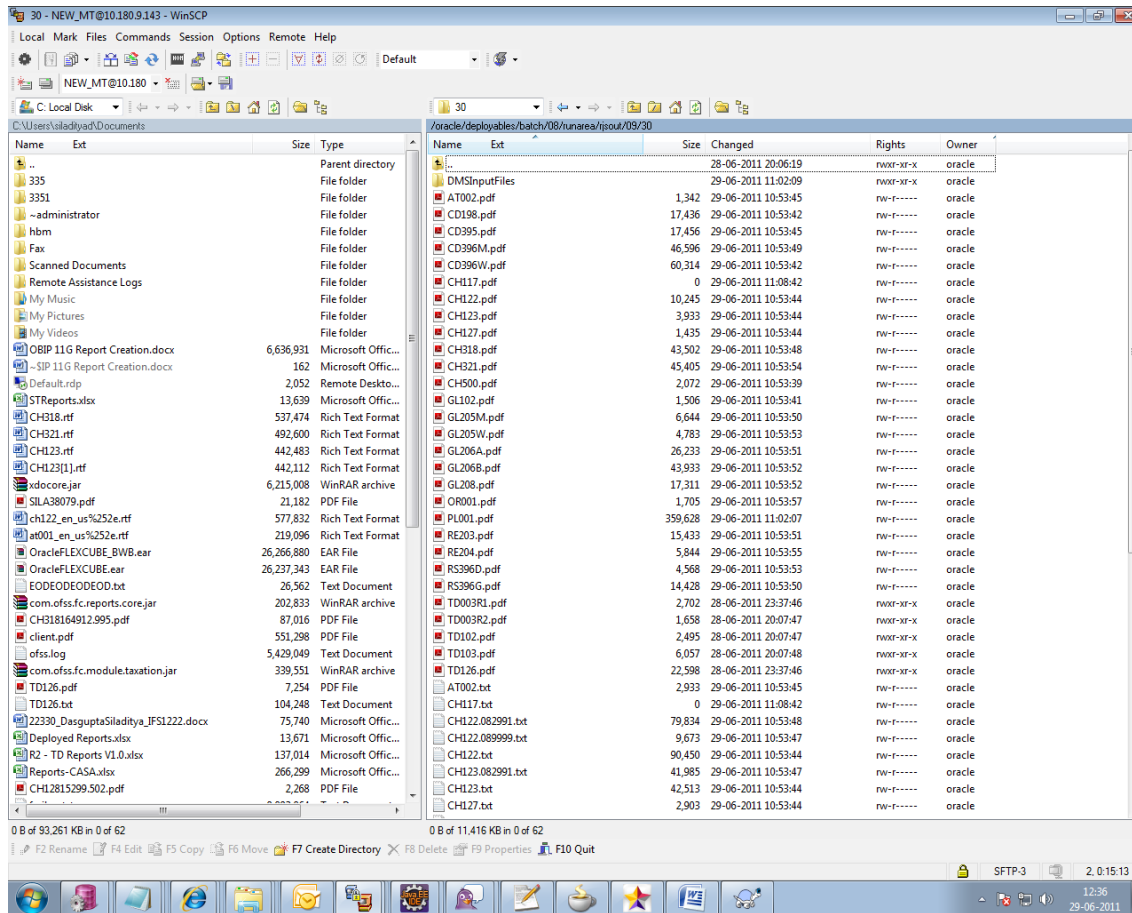
The reports (for example, 30th September 2008) will be generated as shown in the host side screen-shot.

Locate these reports at this location in the host server.

/oracle/deployables/batch/08/runarea/rjsout/09/30 which actually is of the format

/config/./.<BankCode>/runarea/rjsout/<MM>/<DD>

Figure 10–20 Batch Report Generation Path



10.12 OBP Adhoc Report Configuration

This section describes the OBP adhoc report configuration.

10.12.1 Define the Adhoc Reports

Define the adhoc reports as follows:

```

Insert into flx_ba_report_ctrl (COD_REPORT_ID, FLG_REP_ADV, COD_
MODULE, NAM_REPORT, TYP_REPORT, FRQ_REPORT, FLG_PRINT, FLG_DELETE,
CTR_REP_COPIES, COD_PRIORITY, COD_ACCESS_LVL, COD_FILEID, BUF_INV_
VAR1, BUF_INV_VAR2, BUF_INV_VAR3, BUF_INV_VAR4, BUF_INV_VAR5, FLG_
MNT_STATUS, COD_MNT_ACTION, COD_LAST_MNT_MAKERID, COD_LAST_MNT_
CHKRID, DAT_LAST_MNT, CTR_UPDAT_SRLNO, FLG_SOURCE, FLG_SPLIT, FLG_
PROD_REP, COD_REPORT_DB_PREFIX, FLG_APPLY_SC, REF_UDF_NO, XPATH,
FILE_DESC, FLG_REPORT_SERVER)
values ('CH318', 'R', 'CH', 'CASA BALANCE LISTING', 'A', '1', '1',
'0', 1, 0, 0, 10047, ' ', ' ', ' ', ' ', ' ', ' ', 'A', ' ', 'PHASE_2',
'PHASE_2', to_date('01-11-1999', 'dd-mm-yyyy'), 2, 'P', 'Y', 'P',
'PROD', ' ', ' ', ' ', 'Savings Listing Reports', 'B');

```

10.12.2 Define the Adhoc Report Parameters

Define the adhoc report parameters as follows:

```
INSERT INTO flx_ba_report_params (COD_REPORT_ID,FLG_REP_ADV,COD_
SERIAL,NAM_PROMPT, COD_FLD_TYP,LEN_FLD,FLG_DELETE,DAT_LAST_MNT,NAM_
VAL_ROUTINE,REQD_DESC) VALUES ('CH318','R',1,'Branch
Code',0,0,'N','01-NOV-99','','Y')
/
INSERT INTO flx_ba_report_params (COD_REPORT_ID,FLG_REP_ADV,COD_
SERIAL,NAM_PROMPT, COD_FLD_TYP,LEN_FLD,FLG_DELETE,DAT_LAST_MNT,NAM_
VAL_ROUTINE,REQD_DESC) VALUES ('CH318','R',2,'Product
Code',0,0,'N','01-NOV-99','','Y')
/
INSERT INTO flx_ba_report_params (COD_REPORT_ID,FLG_REP_ADV,COD_
SERIAL,NAM_PROMPT, COD_FLD_TYP,LEN_FLD,FLG_DELETE,DAT_LAST_MNT,NAM_
VAL_ROUTINE,REQD_DESC) VALUES ('CH318','R',3,'From Date (DD-MMM-
YYYY)',8,0,'N','01-NOV-99','','Y')
/
```

Also COD_FLD_TYP = 8 will ensures the host side date format validations.

COD_FLD_TYP = 0 is for string type parameters.

Corresponding to each of the above sequence of parameters appearing in screen, a mandatory parameter 'FUNC_PARAM<Parameter Sequence Number>' should be defined in Oracle Analytics Publisher Data Model. So the input parameter 'FUNC_PARAM2' defined in data model should correspond to Product Code as defined above.

10.12.3 Define the Adhoc Reports to be listed in Screen

Define the group name as follows:

For Adhoc Report, column FILE_DESC of report master table FLX_BA_REPORT_CTRL contains the name of the group under which the report will be listed in 7775 screen.

10.12.4 Adding Screen Tab for Report Module

For adding a Screen Tab do the following:

```
com.ofss.fc.ui.view.brop.jar@
public_
html/com/ofss/fc/ui/view/brop/reportRequest/form/ReportRequest.jsff
<af:commandNavigationItem partialSubmit="true" text="#{rb7775.LBL_
Reconciliation}"
binding="#{ReportRequest.cn11}" id="cn11" immediate="true"
actionListener="#{ReportRequest.processMode}" selected="false">
<f:attribute name="mode" value="Reconciliation"/>
</af:commandNavigationItem>

com.ofss.fc.ui.view.brop.jar@
/com/ofss/fc/ui/view/brop/reportRequest/backing/ReportRequest.java
```

```

private RichCommandNavigationItem cnill;
Add following accessors:-
public void setCnill(RichCommandNavigationItem cnill) {
this.cnill = cnill;
}
public RichCommandNavigationItem getCnill() {
return cnill;
}

```

Also modify the selection tab highlighting portion of the code.

com.ofss.fc.ui.view.brop.jar@

/com/ofss/fc/ui/view/brop/reportRequest/rb/ReportRequest_en.properties

LBL_Reconciliation = Reconciliation

10.13 Adhoc Report Generation – Screen 7775

The adhoc report can be generated using the following screen:

Figure 10–21 Adhoc Report Generation - Report Request

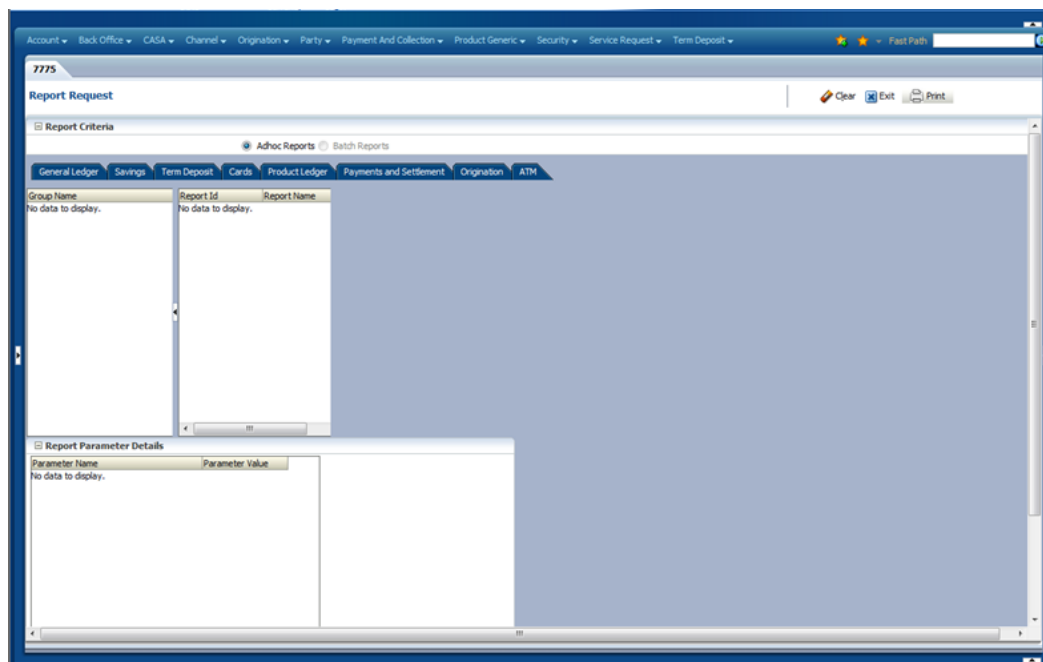
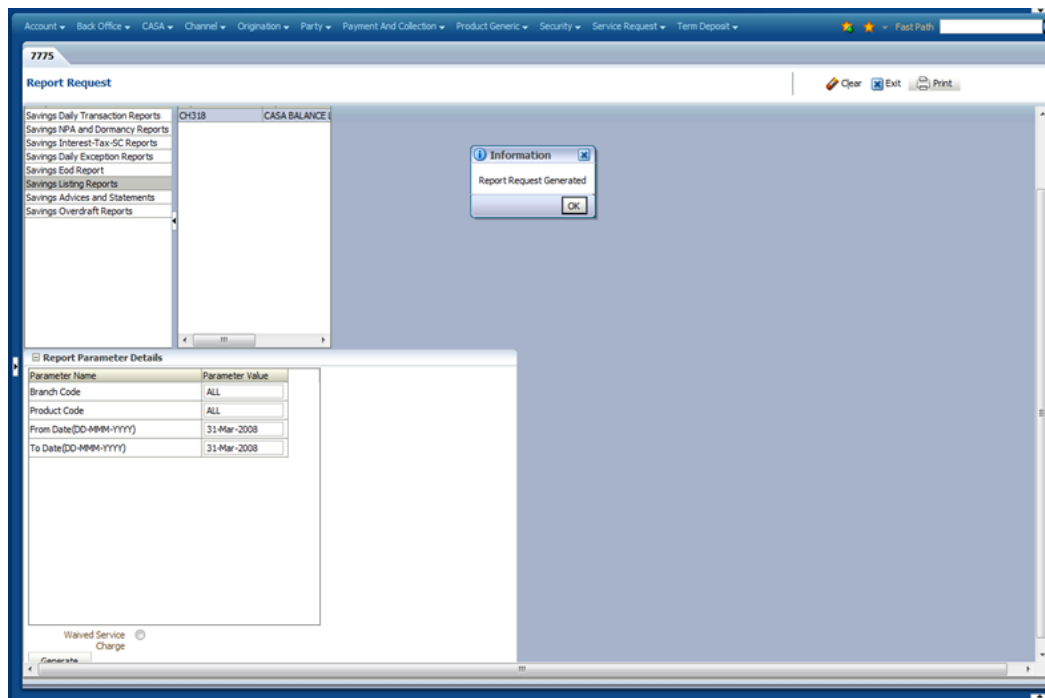


Figure 10–22 Adhoc Report Generation - Report Generated

On filling the parameters and clicking on 'Generate' the report request gets successfully posted.

At the end of Adhoc report generation, RJS_REQUESTS gets logged with the generated report status as D -> Done, F-> Failed.

10.14 Adhoc Report Viewing – Screen 7779

The adhoc report can be viewed using the following screen:

Figure 10–23 Advice Report

The screenshot displays the Oracle FLEXCUBE interface in a Windows Internet Explorer browser. The page title is 'Advice Report'. Below the title, there are navigation options: 'Print or Cancel Advice/Report', 'Clear', 'Exit', and 'Print'. A 'User Id' dropdown menu is set to 'X000071'. The main section is titled 'Advice/Report List' and contains a table with the following data:

ID	Report/Advice Name	Status	Star
CH318	CASA BALANCE LISTING	Done	201
CH318	CASA BALANCE LISTING	Done	201
CH318	CASA BALANCE LISTING	Pending	201
CH318	CASA BALANCE LISTING	Failed	201
CH321	Dormant Accounts Statistics	Failed	201
CH321	Dormant Accounts Statistics	Failed	201
CH318	CASA BALANCE LISTING	Failed	201
CH318	CASA BALANCE LISTING	Failed	201
CH318	CASA BALANCE LISTING	Done	201
CH321	Dormant Accounts Statistics	Failed	201
OR001	Origination Application Failed Report	Failed	201
OR001	Origination Application Failed Report	Failed	201

At the bottom of the table, it indicates 'Columns Hidden: 2'. The browser's address bar shows the URL: http://10.180.9.106:7001/com.ofss.fc.ui.view/faces/main.jspx?_afrcLoop=62392076004776&_afrcWindowMode=0&_afrcCtrlState=jjh9p7gg8_4. The Windows taskbar at the bottom shows the system tray with the date 29-06-2011 and time 12:45.

On selecting the correct user id that generated the report we get the reports generated by that user.

Now sort the Transaction Number (right most column) in the descending order.

Select the top record and click 'View Report'.

Figure 10–24 View Generated Adhoc Report

Account Number	Customer Name	Account Currency	Account Status	Clear Balance	Uncleared Balance	Hold
000047845	Keith Watson	AUD	Inactive	54.21	0.00	
000047861	franklin joseph	AUD	Regular	420.00	20.00	
000047896	franklin joseph	AUD	Regular	712.74	0.00	
000024125	Brad Pitt	AUD	Regular	0.00	0.00	
000024176	Randy Orton	AUD	Regular	0.00	0.00	
000024192	John GGG Cena	AUD	Regular	22,189.61	0.00	
000024205	Atul KKK Sinha	AUD	Regular	0.00	0.00	
000024256	Kanh Do	AUD	Regular	993,838.02	0.00	
000024301	Andy Flower	AUD	Regular	26,810.07	0.00	
000024408	Shane Watson	AUD	Regular	3,016.62	0.00	
000024491	Aaron Lo	AUD	Regular	10,079.18	0.00	
000024504	JJJJJJJJJJ RRRRRRR	AUD	Regular	0.00	0.00	
000024627	jay more	AUD	Regular	110,263.88	0.00	
000024686	Harry Jonto	AUD	Regular	0.00	0.00	
000024889	Shane Watson	AUD	Regular	5,021.16	0.00	
000024897	Shane Watson	AUD	Regular	14,063.40	0.00	
000024918	Shane Watson	AUD	Regular	41,169.39	50,000.00	
000025013	John GGG Cena	AUD	Regular	25,227.97	0.00	
000025144	franklin pearl	AUD	Regular	2,322.47	2,795.00	
000025179	ansdnn asnasn	AUD	Regular	0.00	0.00	
000025320	brad hopes	AUD	Regular	1,108.71	0.00	
000025347	HHHHHHH MMMMMM	AUD	Regular	0.00	0.00	
000025363	adam gilchrist	AUD	Regular	0.00	0.00	
000025435	Charlotte Collins	AUD	Regular	0.00	0.00	
000025443	Charlotte Collins	AUD	Regular	100,491.50	0.00	
000048098	Darryl Molley	AUD	Regular	102,275,320.27	0.00	1,
000048100	ice ice	AUD	Regular	0.00	0.00	
000048119	ice 1	AUD	Regular	50,000.00	0.00	
000048127	iceice ice	AUD	Regular	0.00	0.00	
000048135	Aishwarya ram	AUD	Regular	100,009.81	0.00	
000048151	ice ice	AUD	Regular	0.00	0.00	
000048207	Martin Berchmans	AUD	Regular	95,680.26	0.00	

The report is rendered in the front end.

11 Security Customizations

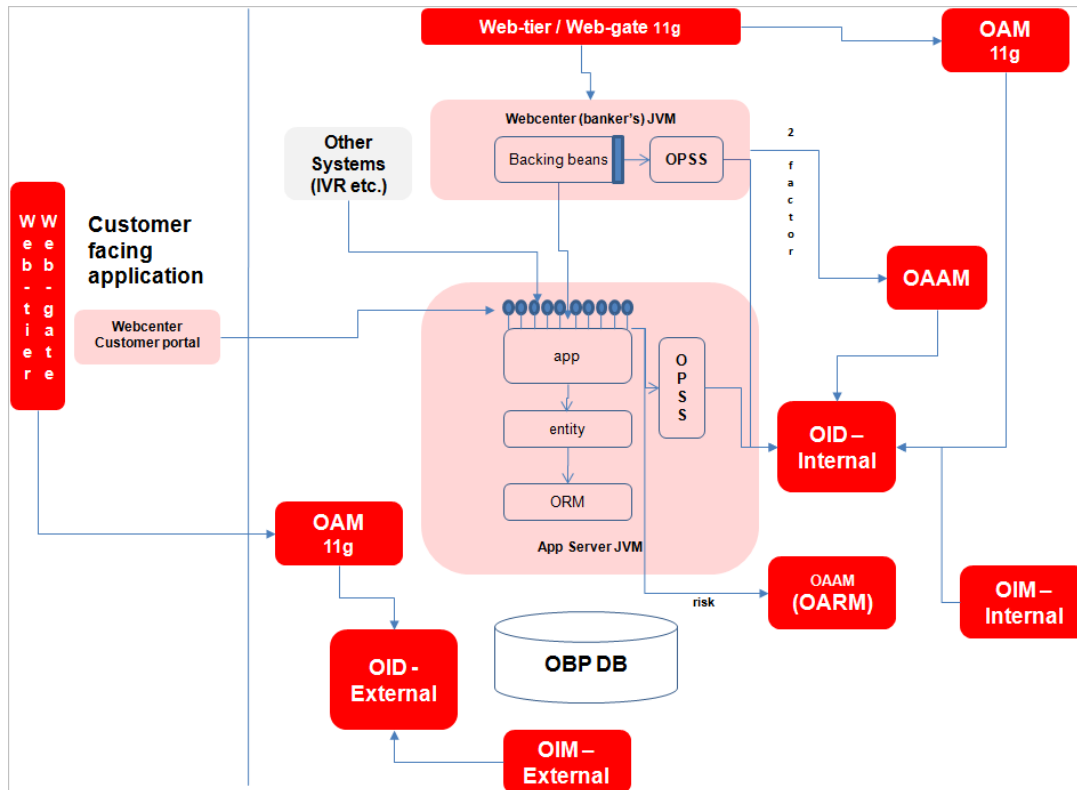
OBP comprising of several modules has to interface with various systems in an enterprise to transfer or share data which is generated during business activity that takes place during teller operations or processing. While managing the transactions that are within OBP's domain, it is needed to consider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

This is possible if these capabilities can be externalized from the application itself and are implemented within products that are specialized to handle such services. Examples of these services include authentication against an enterprise identity-store, creating permissions and role-based authorization model that controls access to not only the components of the application, but also the data that is visible to the user based on fine-grained entitlements.

The following security functions are provided with the extensibility features:

- Attributes participating in access policy rules
- Attributes participating in fraud assertion rules
- Attributes participating in matrix-based approval checks
- Account validator
- Customer validator
- Business unit validator
- Adding validators
- Customizing user search
- Customizing of a 'Send OTP | Validate OTP' logic
- Customizing Role Evaluation
- Customizing Limit Exclusions
- Adding approval checks

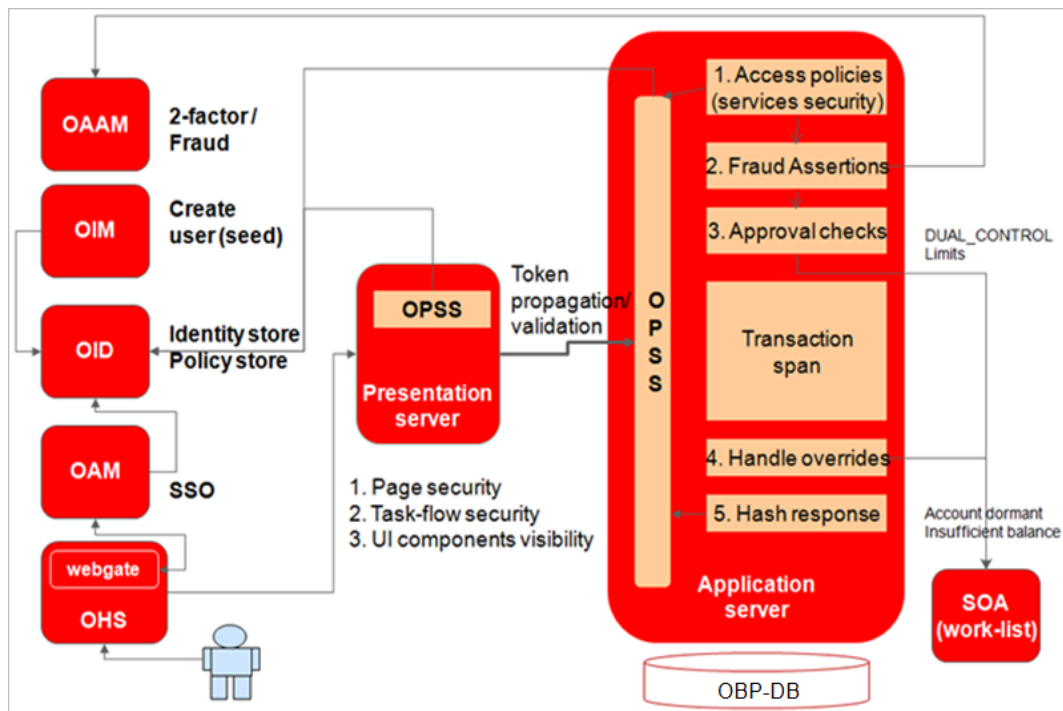
Figure 11–1 Security Customizations Interface



- Oracle Identity Manager (OIM) is used for managing user provisioning.
- Oracle Access Manager (OAM) is used for managing declarative authentication and SSO.
- Oracle Platform Security Services (OPSS) is used for runtime evaluation of authn / authz.
- Oracle Adaptive Access Manager (OAAM)/Oracle Adaptive Risk Manager (OARM) is used for step-up authentication and fraud management.
- Authorization Policy Manager (APM) is used to manage access policy definitions.
- SM502- Policy Management screen is used to manage access policy definitions
- Oracle Internet Directory (OID) is used as the identity/policy store.

A high-level security use case has the following access checks and assertions.

Figure 11–2 Security Use Case with Access Checks and Assertions



11.1 OPSS Access Policies / Matrix Auth – Adding Attributes

OBP uses OPSS to assert role-based access policies. Access policies are rules-based to give more flexibility.

Example of an access policy rule:

```
Grant
Role = RetailBranchOperationsExecutive
Service=com.ofss.fc.app.dda.service.transaction.DemandDepositCashT
ransactionService.depositCash
Action = perform
IF DepositCash_IsEmployeeAccount=false AND DepositCash_
IsRestrictedAccount=false
```

In the above example, the following facts (attributes) make up the access policy rule:

```
DepositCash_IsEmployeeAccount
DepositCash_IsRestrictedAccount
```

The security framework allows for addition to the facts that can be used in rules. The steps to do this are mentioned in the next section.

11.1.1 Steps

Following steps are needed to add an extra attribute to an access policy rule.

You can use SM500 screen or use 'PolicyStoreSetup' -> script 'seedFreshPolicyStore.sh'.

Below are the steps to use consulting attributes in 'com.ofss.fc.approval.lendingspi_confirmstructuresolution' routing rule:

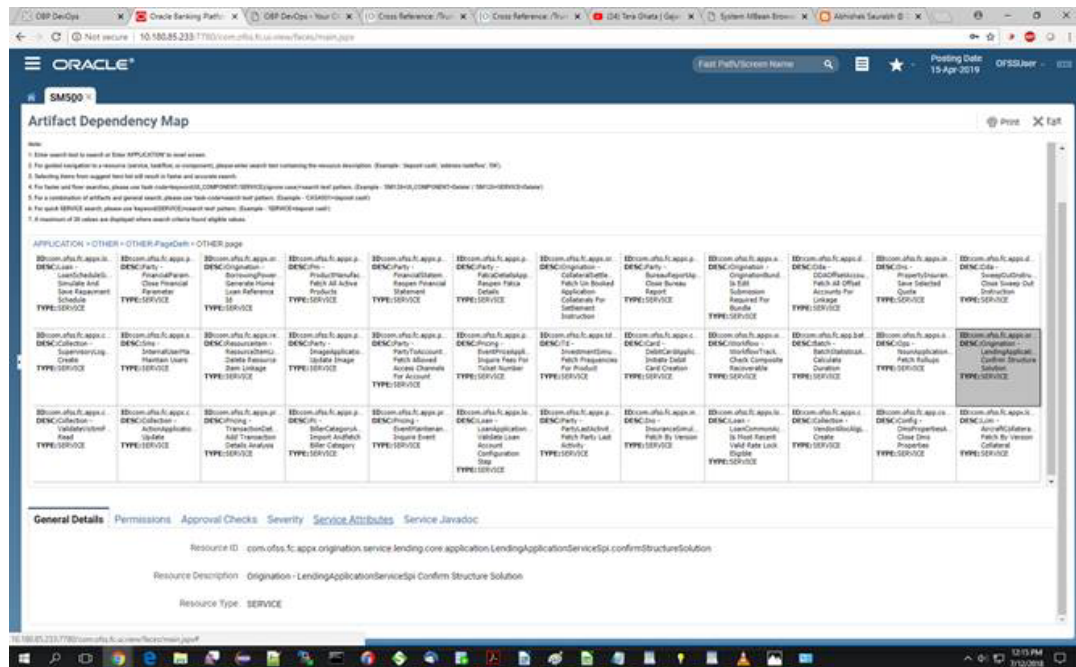
1. Add the attribute name for service_id in SM500:

com.ofss.fc.appx.Origination.service.lending.core.application.LendingApplicationServiceSpi.confirmStructureSolution,AllowedPolicyAttributes,LendingStructureSolution_IsTaskIncludesSecuredProduct,N,2/1

- a. Search for serviceld

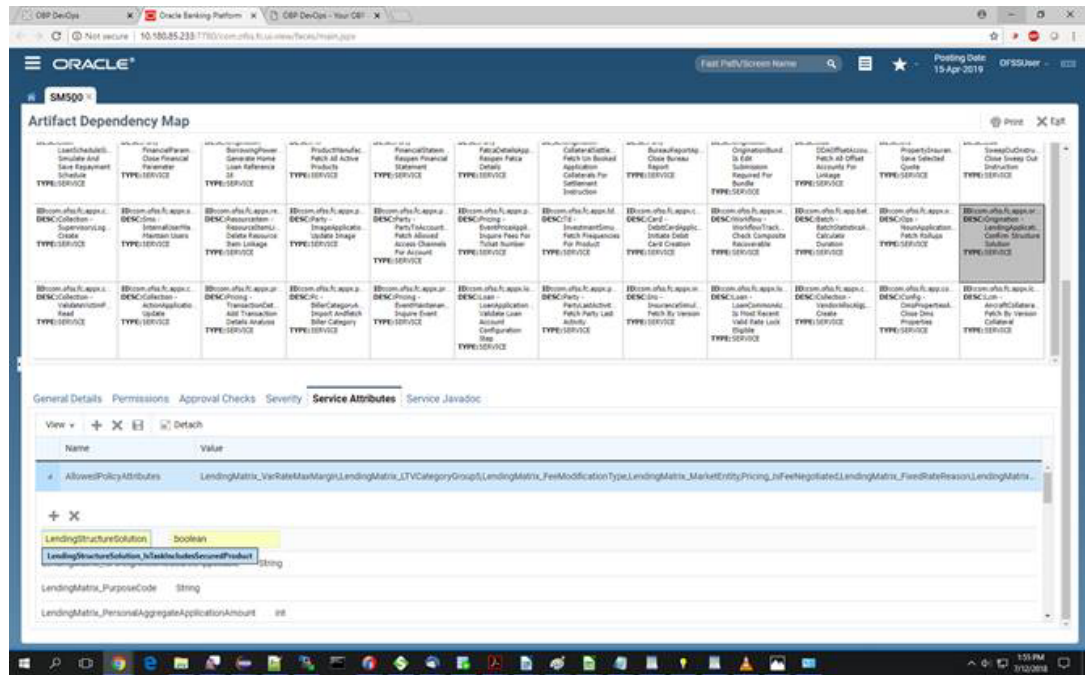
'com.ofss.fc.appx.Origination.service.lending.core.application.LendingApplicationServiceSpi.confirmStructureSolution' in SM500

Figure 11–3 Add Attributes Name for Service ID



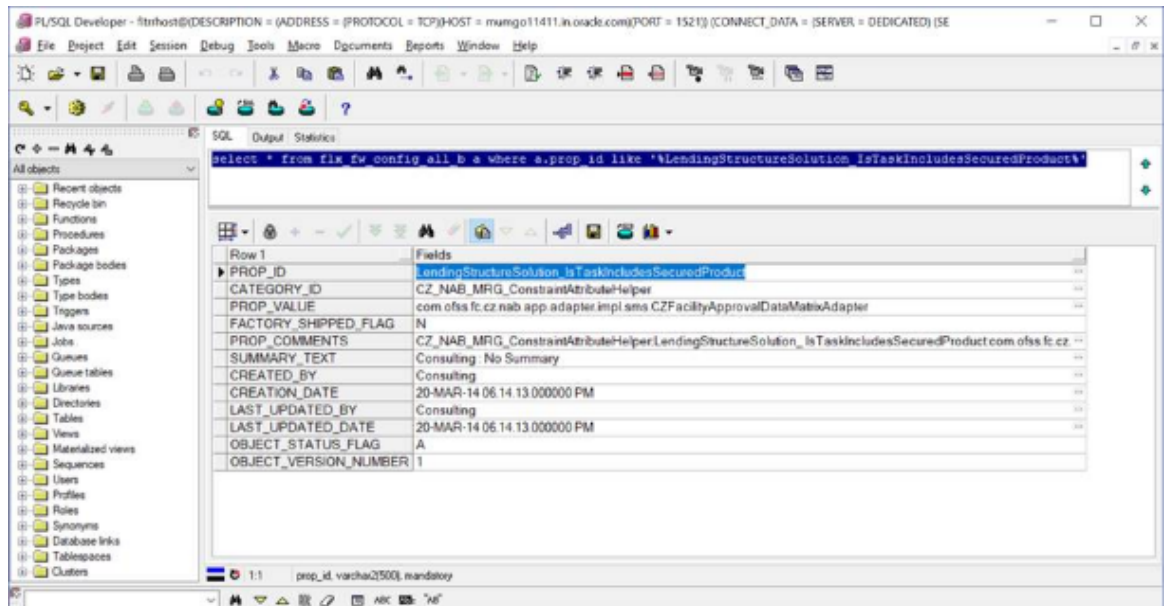
- b. Add attribute name 'LendingStructureSolution_IsTaskIncludesSecuredProduct' with appropriate datatype in 'allowed policy attributes' under 'Service Attributes' tab in SM500 as shown below:

Figure 11–4 Add Service Attributes in SM500



2. Define the attribute in Constraint Attribute Config with associated adapter to fetch attribute value.

Figure 11–5 Constraint Attribute Config with Associated Adapter



3. Add derivation logic to set the attribute value in Adapter that is mapped in config.
Develop custom adapter to retrieve attribute value. Attribute should be structured along similar lines as the other adapters used for the same purpose.

Example:

```
Attribute - LendingStructureSolution_
IsTaskIncludesSecuredProduct
Adapter -
public
com.ofss.fc.app.adapter.impl.sms.FacilityApprovalDataMatrixAd
apter {
public Boolean getIsTaskIncludesSecuredProduct() {
if (facilityApprovalDataDTO != null &&
facilityApprovalDataDTO.getIsTaskIncludesSecuredProduct() !=
null)
return facilityApprovalDataDTO.getIsTaskIncludesSecuredProduct
();
else
return false;
} }
```

Note

The naming convention of the attribute should be as follows:

The first part of the attribute till the '-' delimiter identifies the transaction. The remaining part with CamelCase is prefixed with a 'get' to form the method in the adapter.

4. Add entry in ConstraintAttributeHelper.properties to link the attribute to the adapter.

```
Public final String LendingStructureSolution_
IsTaskIncludesSecuredProduct =
"com.ofss.fc.app.adapter.impl.sms.FacilityApprovalDataMatrixA
dapter";
```

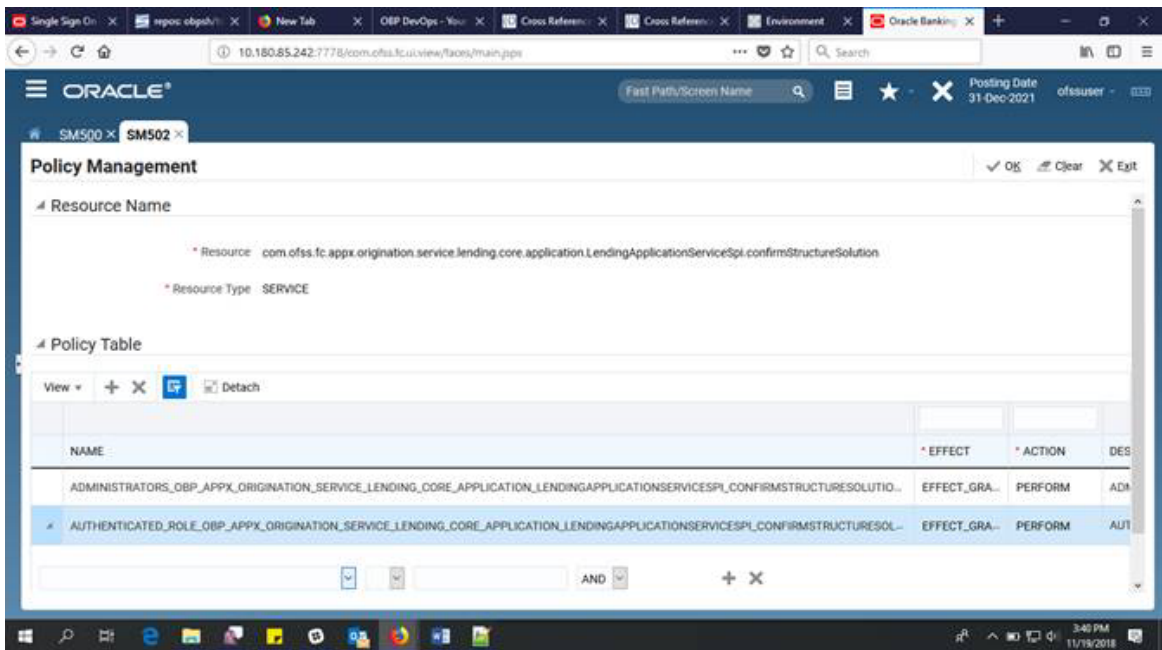
5. Add/Modify access policy/rule using SM502 screen or using 'refreshMatrixAuthPolicies.sh' script available in 'PolicyStoreSetup' utility.

11.1.1.1 Example of Matrix_auth conditional rule

After severity configuration in SM500, open SM502 screen to create Matrix_auth conditional rule for ServiceID.

1. In SM502, select Resource Type as 'SERVICE' and in Resource add 'ServiceID'.and then press Enter.

Figure 11–6 Select Resource Type and Add Service ID

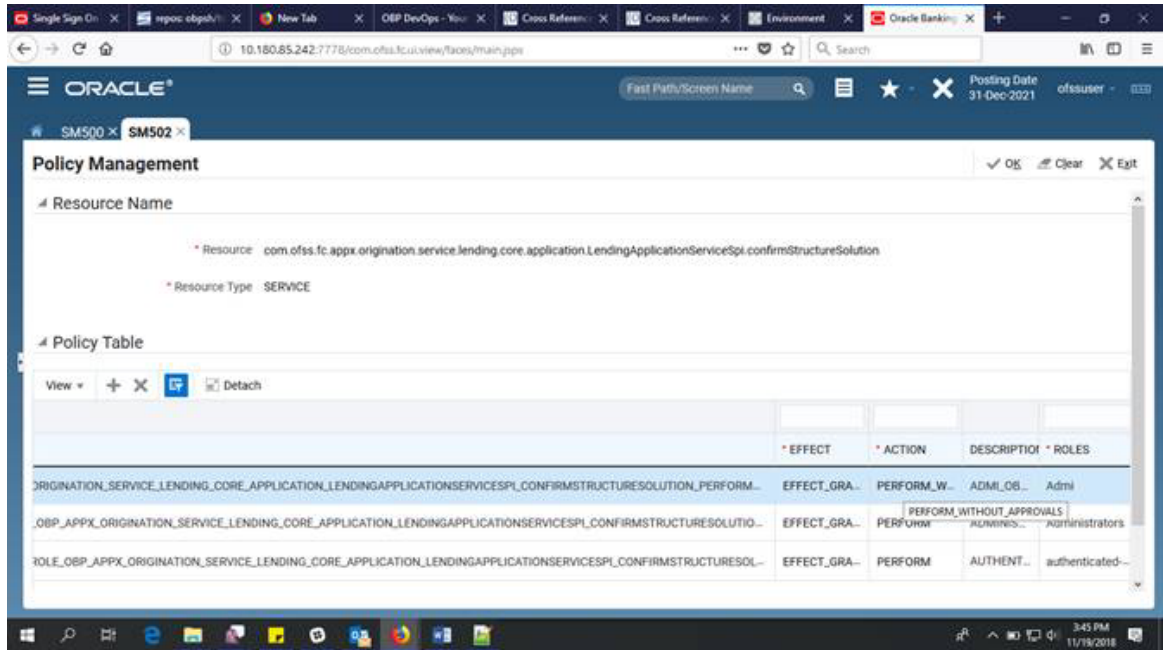


2. Now create policy for 'PERFORM_WITHOUT_APPROVALS' for the resource.

Under PolicyTable, click on Add, and select following details:

- EFFECT: GRANT
- ACTION: PERFORM_WITHOUT_APPROVALS
- ROLES: Administrators

Figure 11–7 Add PolicyTable Details



- Then, add condition using attribute 'LendingStructureSolution_Margin'.

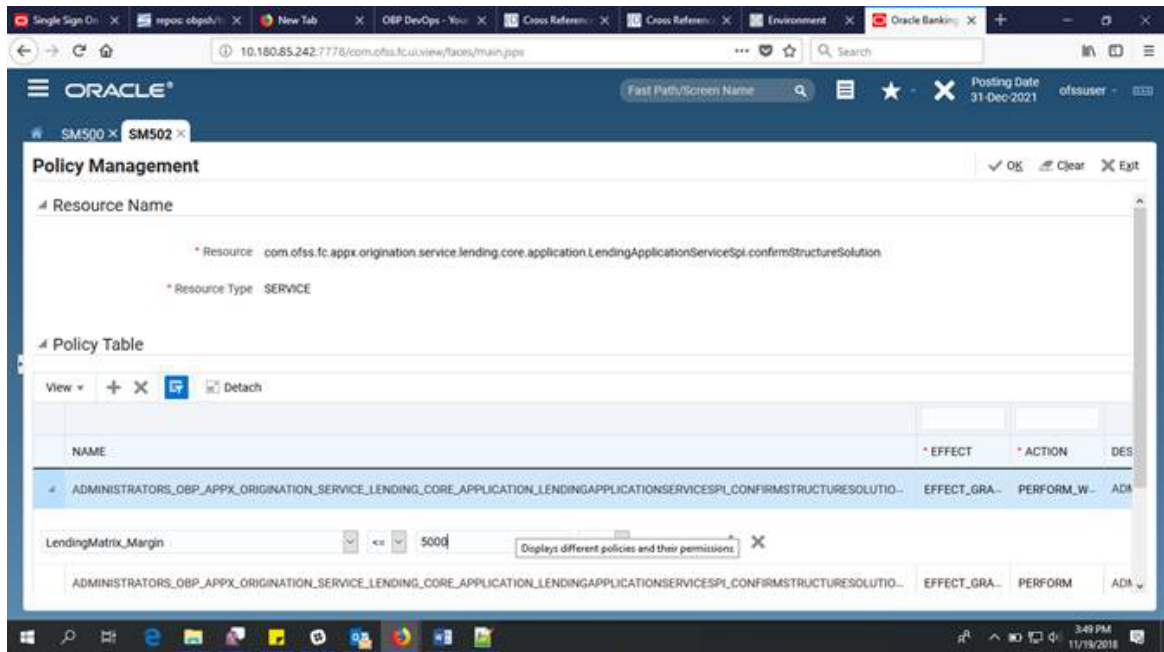
Condition:

If LendingStructureSolution_Margin<=5000 , then PerformWithOutApprovals

else perform,

So, if LendingStructureSolution_Margin>5000 then Transaction will go for Approval.

Figure 11–8 Add Condition

**Note**

The only difference between the policy semantics in `matrix_auth` and access policy is the 'Action'. ['perform' versus 'performWithoutApprovals']

11.2 OAAM Fraud Assertions – Adding Attributes

OBP uses OAAM to assert fraud policies consisting of rules to identify potentially fraudulent transactions.

Attributes used in fraud identification rules:

```
payee_id, account_number
```

The security framework allows for addition to this list of facts. The steps to do this are mentioned in the next section.

11.2.1 Steps

Following steps are needed to add an attribute to an existing OAAM transaction:

1. Add the attribute under 'AllowedPolicyAttributes' against the particular resource.
2. Add attribute in OID under the 'Attributes' entry.
3. Develop custom adapter to retrieve attribute value.
4. Add entry in `ConstraintAttributeHelper.properties` to link the attribute to the adapter.

The above steps are exactly the same as mentioned in the previous section.

1. Add seed data in the following tables to persist the mapping between OID attributes and OAAM attributes.

flx_sm_fraud_txn_attributes (stores OAAM transaction key to OAAM attribute mapping) and

flx_sm_fraud_assert_attributes (stores OBP attributeName - oaamAttributeName mapping.

Example -

```
insert into Flx_Sm_Fraud_Txn_Attributes (TRANSACTION_KEY,
ATTRIBUTE_NAME)
values ('payment', 'is_2fa_completed')
/
insert into flx_sm_fraud_assert_attributes (ATTRIBUTE_KEY,
FRAUD_ATTRIBUTE_NAME)
values ('OutgoiOBPaymentService_Is2FACompleted', 'is_2fa_
completed')
/
```

2. Add/Modify fraud rules in OAAM to use the extra attribute

Figure 11–9 Add or Modify Fraud Rules in OAAM - Data Tab

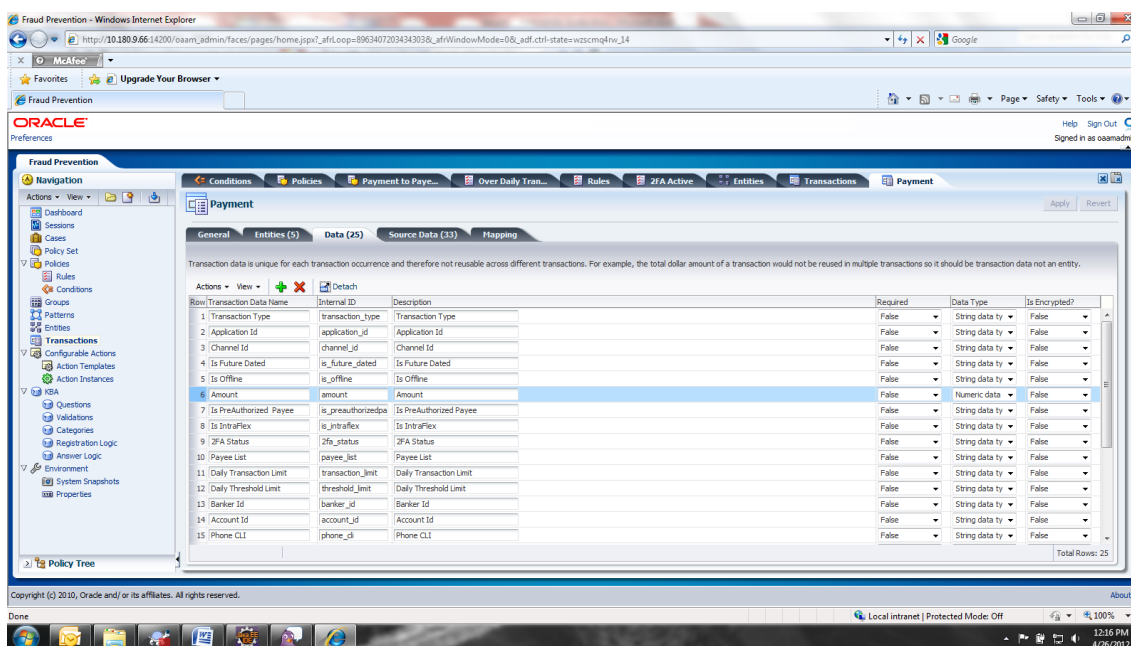
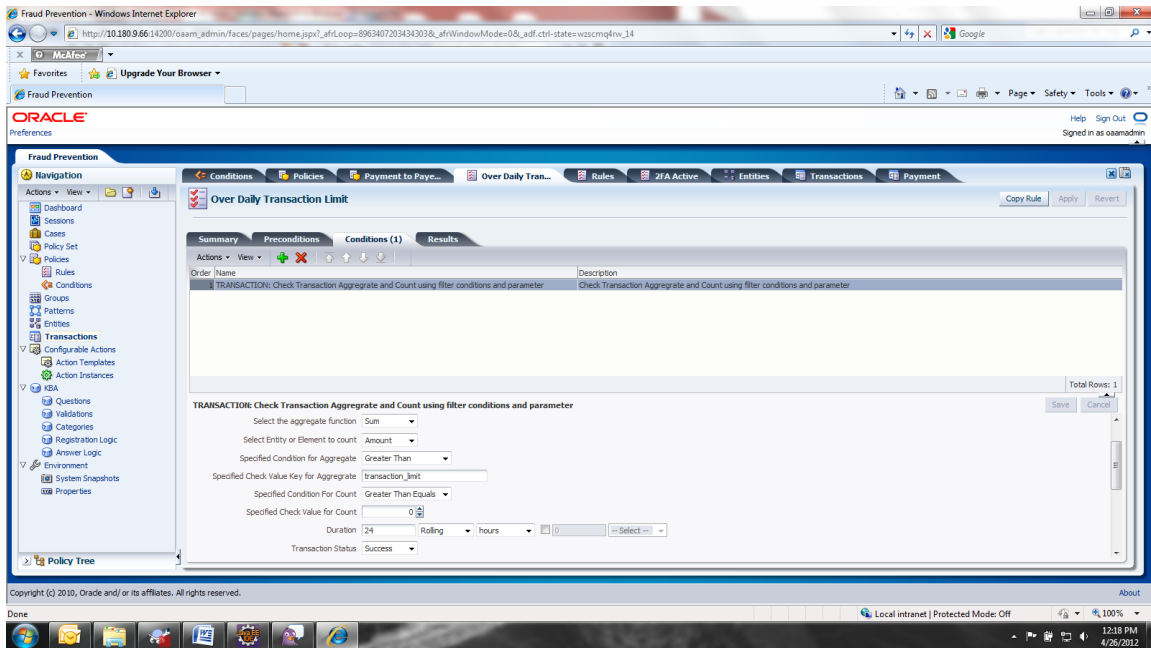


Figure 11–10 Add or Modify Fraud Rules in OAAM - Conditions Tab



11.3 Security Validators

In addition to OPSS access policies, there are additional validators that perform security checks. The sole purpose of these validators was to give hooks to enable site specific security logic that would be complicated enough and hence cannot be provisioned as rules.

Note

These additional validators come into effect only when the following property is set.

```
APPLICATION_SECURITY_VALIDATOR=true
```

The role, channel, service and the attributes available in the execution context are passed to the validators.

The validators implement the interface
`com.ofss.fc.app.adapter.impl.sms.validator.IExtendableApplicationValidator`

There are three types of security-validation categories:

- Customer validators
- Account validators
- Business unit validators

There can be multiple validator classes contributing to each individual category.

The package structure of the validators is required to be:

```
'com.ofss.fc.app.adapter.impl.sms.validator'
```

11.3.1 Customer Validators

This validator returns a Boolean signifying whether the logged-in user can perform a transaction on the customer.

Step 1

Add property in `ApplicationValidators.properties`

```
com.ofss.fc.app.dda.service.account.core.DDAInquiryApplicationService.fetchBasicDetails.CustomerValidators=RestrictedAccountApplicationValidator,EmployeeAccountApplicationValidator
```

Step 2

Develop custom validator along the lines of existing adapters.

11.3.2 Account Validators

This validator returns a Boolean signifying whether the logged-in user can perform a transaction on the account.

Step 1

Add property in `ApplicationValidators.properties`

```
com.ofss.fc.app.dda.service.account.core.DDAInquiryApplicationService.fetchBasicDetails.AccountValidators=RestrictedAccountApplicationValidator,EmployeeAccountApplicationValidator
```

Step 2

Develop custom validator along the lines of existing adapters.

11.3.3 Business Unit Validators

This validator returns a Boolean signifying whether the logged-in user can perform a transaction on the business unit.

Step 1

Add property in `ApplicationValidators.properties`

```
APPLY_BUSINESS_UNIT_VALIDATION_TO_ALL_SERVICES=false
com.ofss.fc.app.dda.service.account.core.DDAInquiryApplicationService.fetchBasicDetails.BusinessUnitValidators=BusinessUnitApplicationValidator
BusinessUnitValidators=GlobalBusinessUnitApplicationValidator
```

Step 2

Develop custom validator along the lines of existing adapters.

Note

BusinessUnit validation can be global, in which case the following property is set.

```
APPLY_BUSINESS_UNIT_VALIDATION_TO_ALL_SERVICES=true
```

11.4 Customizing User Search

OBP application services use `SessionContext` as an input parameter to set the context of the user interacting with the system. The session-context is populated out of the user's details in `OID`. Across implementations, the user metadata (objectclasses, attributes) is expected to be different resulting in the requirements to have a custom user search capability.

The security framework provides an extension point to inject a custom search. The steps are given in the next section.

11.4.1 Steps

`SecurityConstants.properties` contains attributes that enable custom user searches.

Step 1

Add properties in `SecurityConstants.properties`.

```
CUSTOM_SEARCH_
CLASS=com.ofss.fc.domain.ixface.sms.service.utils.CustomUserSearch
Adapter.retrieveUserUsingExtendableAttributes
CUSTOM_SEARCH_PARAM=nagactualaccessid
```

Step 2

Develop custom user search adapter.

11.5 Customizing One-Time-Password (OTP) Processing Logic

OBP uses `OAAM` for step-up authentication and fraud assertions. Customer is asked to enter a one-time password (OTP) if `OAAM` suspects the transaction to be fraudulent. The logic to send or validate an OTP is implemented using a custom hook. Details of the extension are given in the next section.

11.5.1 Steps

`OAAM.properties` contains a property that provides an extension for second factor password generation / dispatch.

Steps:

1. Add property for the class implementing 2FA in `OAAM.properties`

```
TWO_FACTOR_AUTH_
SERVICE=com.ofss.fc.domain.ixface.oaam.TwoFactorAuthService
```

2. Develop custom class.

11.6 Customizing Role Evaluation

`OPSS` can be configured to add a user in multiple groups (enterprise roles), as a result of which a user can have multiple application roles. OBP uses the most significant role amongst this list to query the user's severity configuration.

The default role-evaluator can be overridden to provide custom role evaluation logic. The steps to do this are given in the next section.

11.6.1 Steps

SecurityConstants.properties contains an attribute that provides an extension for a custom role evaluator.

Step 1

Replace property value in SecurityConstants.properties

```
ROLE_
EVALUATOR=com.ofss.fc.domain.sms.entity.user.roleEvaluationCriteria.SimpleRoleEvaluator
```

Step 2

Develop custom role evaluator.

Currently, the default role evaluator returns the role that has the maximum limits for the service.

11.7 Customizing Limits Exclusions

OBP application services evaluate transaction limits for various services. The assertion logic excludes limits checks for certain conditions. Example, if the customer is transferring funds to his own accounts. Banks have site-specific requirements to exclude transactions from limits checks. The security framework provides an extension point to inject a custom limits exclusions adapter. The steps are given in the next section.

11.7.1 Steps

LimitsExclusions.properties contains a property that enables custom limit exclusions logic for a particular service.

Step 1

Add properties in LimitsExclusions.properties

```
EXCLUSION_PACKAGE_NAME=com.ofss.fc.app.adapter.impl.sms.exclusions
com.ofss.fc.app.dda.service.transaction.DemandDepositFundsTransfer
Service.
transferFundsToBeneficiaries=TransferFundsExclusionValidator
```

Step 2

Develop custom limits exclusions adapter.

11.8 Customizing Business Rules

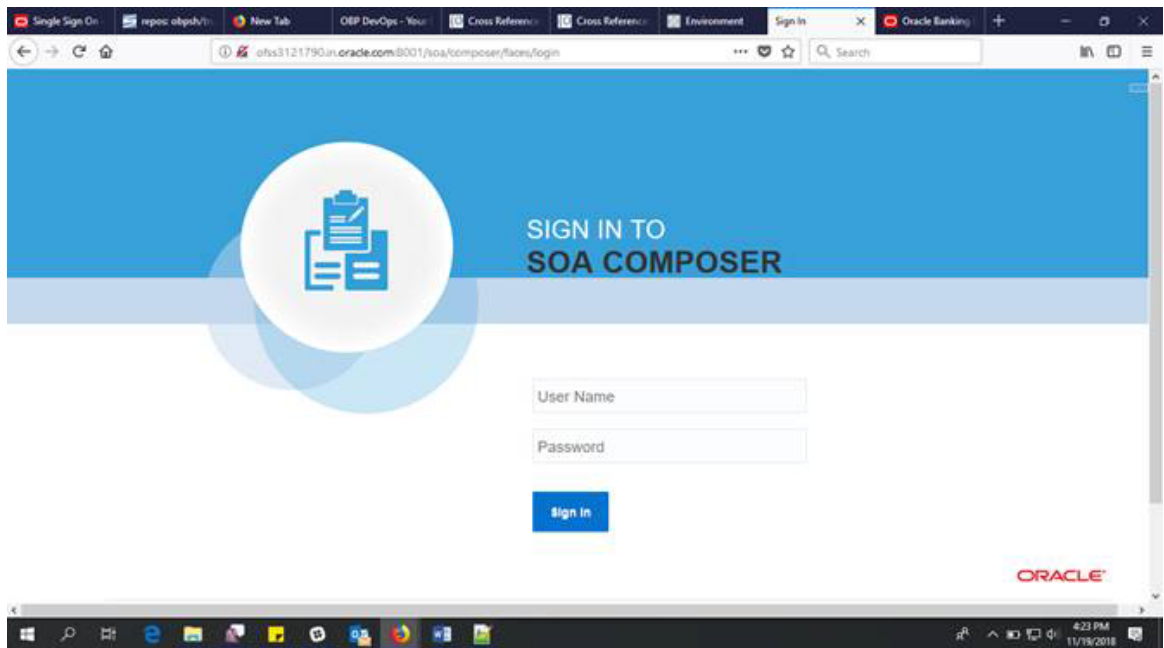
BPEL approval process business rules can be configured and it is based on input authorizations raised during transaction processing at OBP host. The steps for configuring the business rules of the approvals are given in the below section.

11.8.1 Steps to Update the Business Rules by Browser

Following is the Rules setup to be done in SOA Composer:

1. Log in to SOA Composer application of the OBP.

Figure 11–11 Log in to SOA Composer Application screen



2. Search rules file with .rules extension in search box.

Ex: Search 'HT_SubmissionSpi_ConfirmSubmissionRules.rules' in search box.

Figure 11–12 Search Rules file

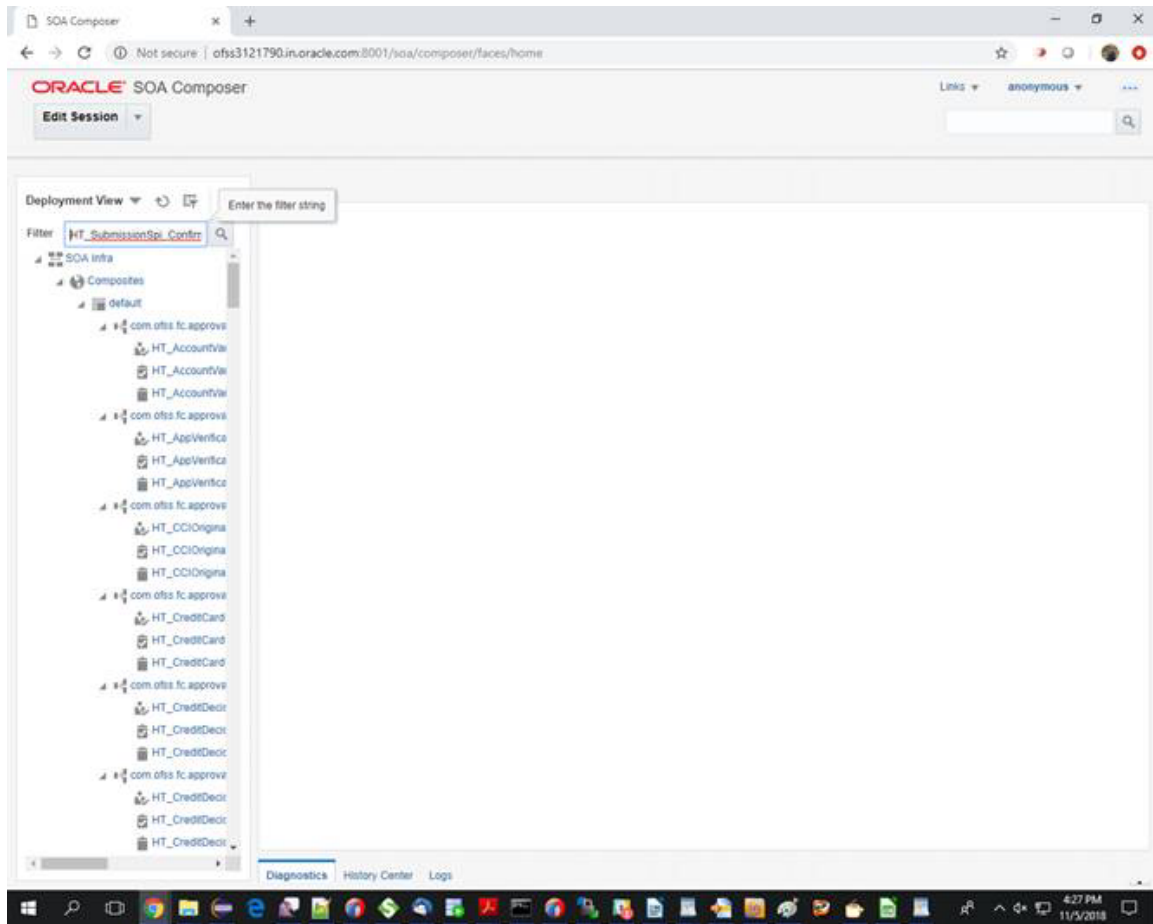
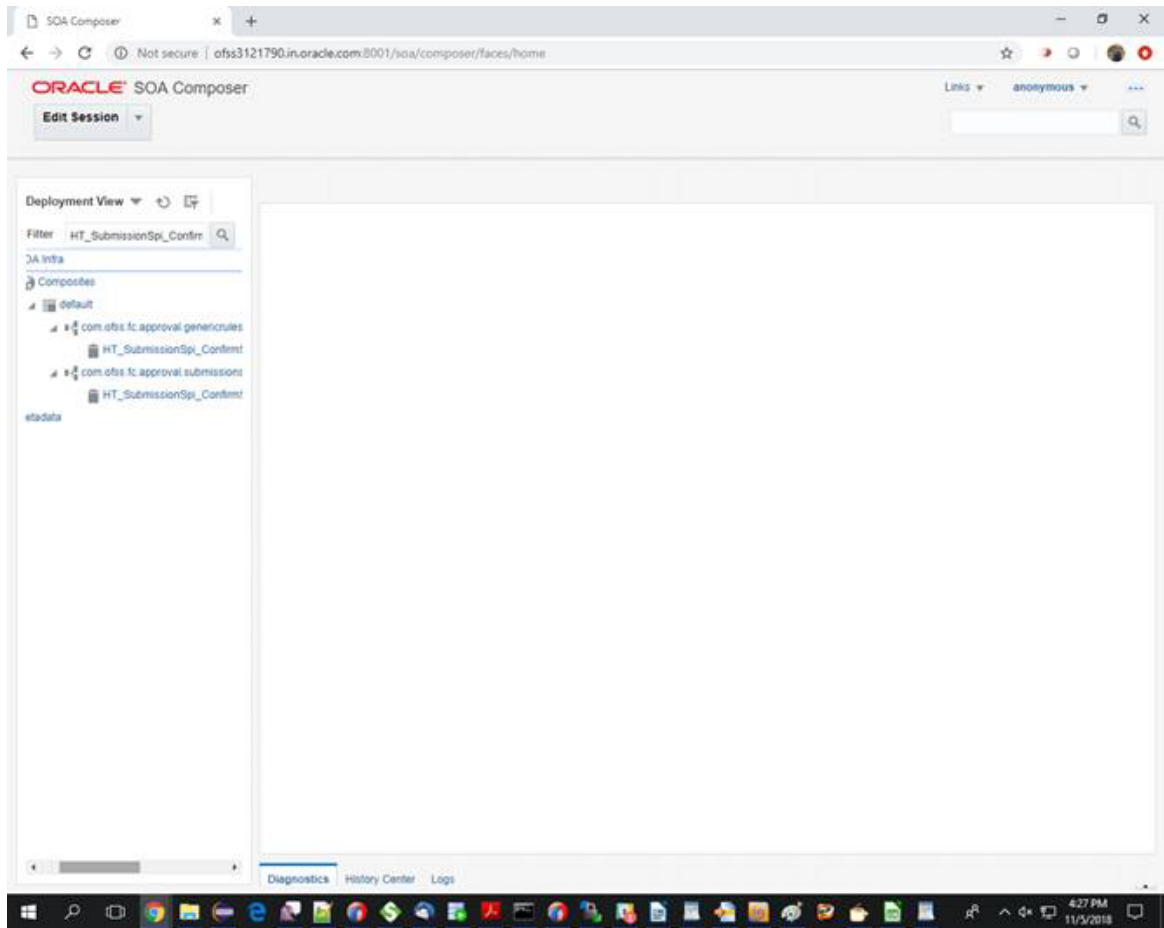
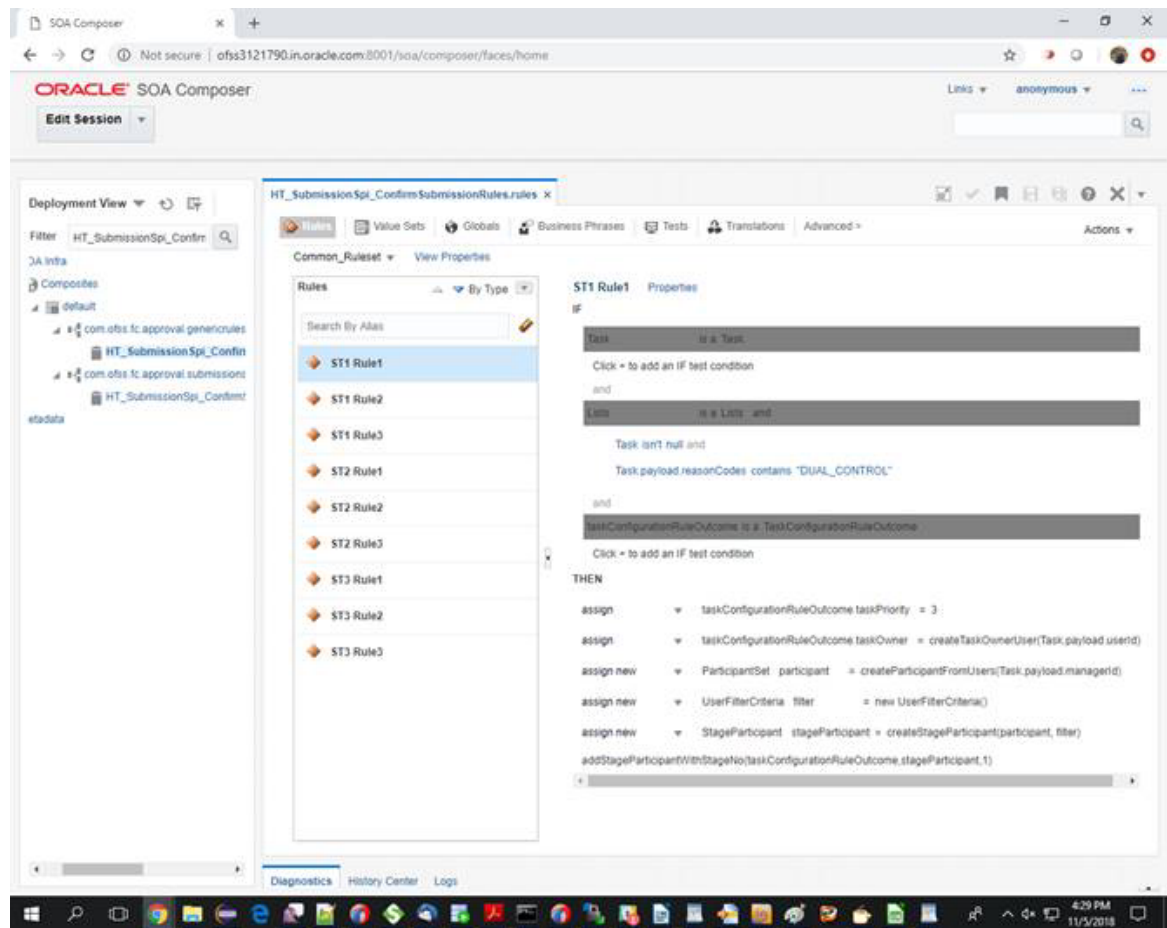


Figure 11–13 Search Rules file -View

3. Select 'com.ofss.fc.approval.genericrulesapprovalspi.executeapprovalrulesorigcore\SOA\oracle\rules\HT_SubmissionSpi_ConfirmSubmissionRules.rules' composite to configure routing rules.

Figure 11–14 Composite to Configure Routing Rules

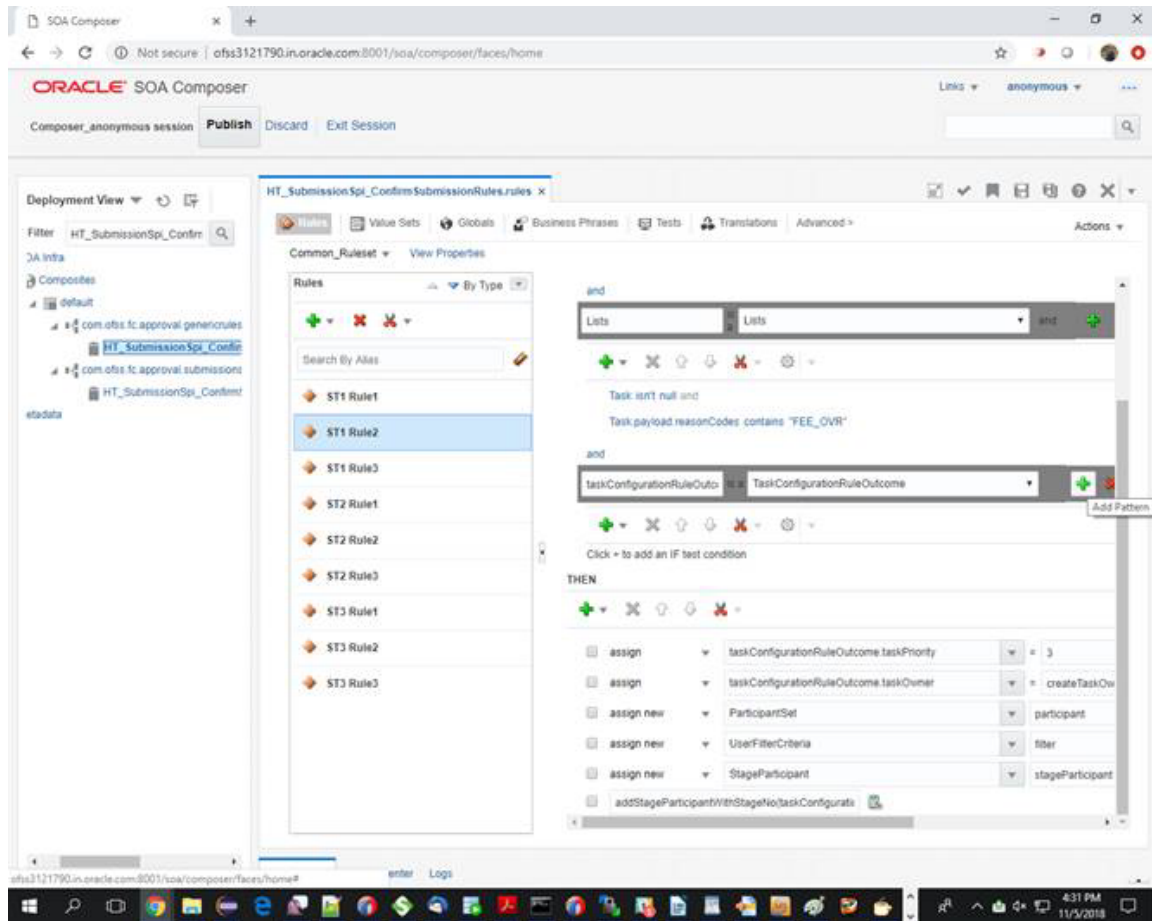


4. Select the stages of approval where the condition in rule is required to be updated.

To edit rules file, click on the 'EDIT Session' button and then select RuleID under 'common_ruleset'.

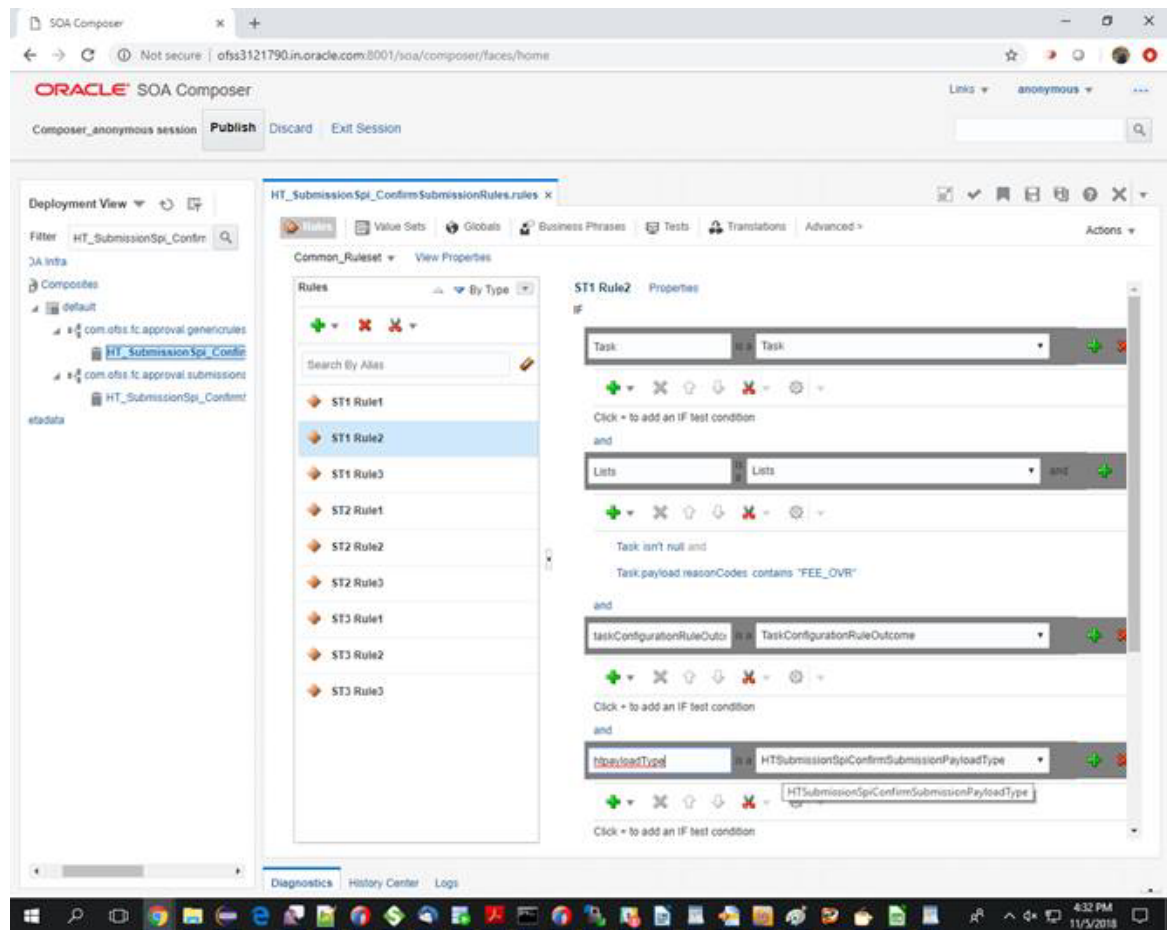
In the below case, select 'ST1Rule2' and then, add new condition using 'Add Pattern' option.

Figure 11–15 Stages of Approval



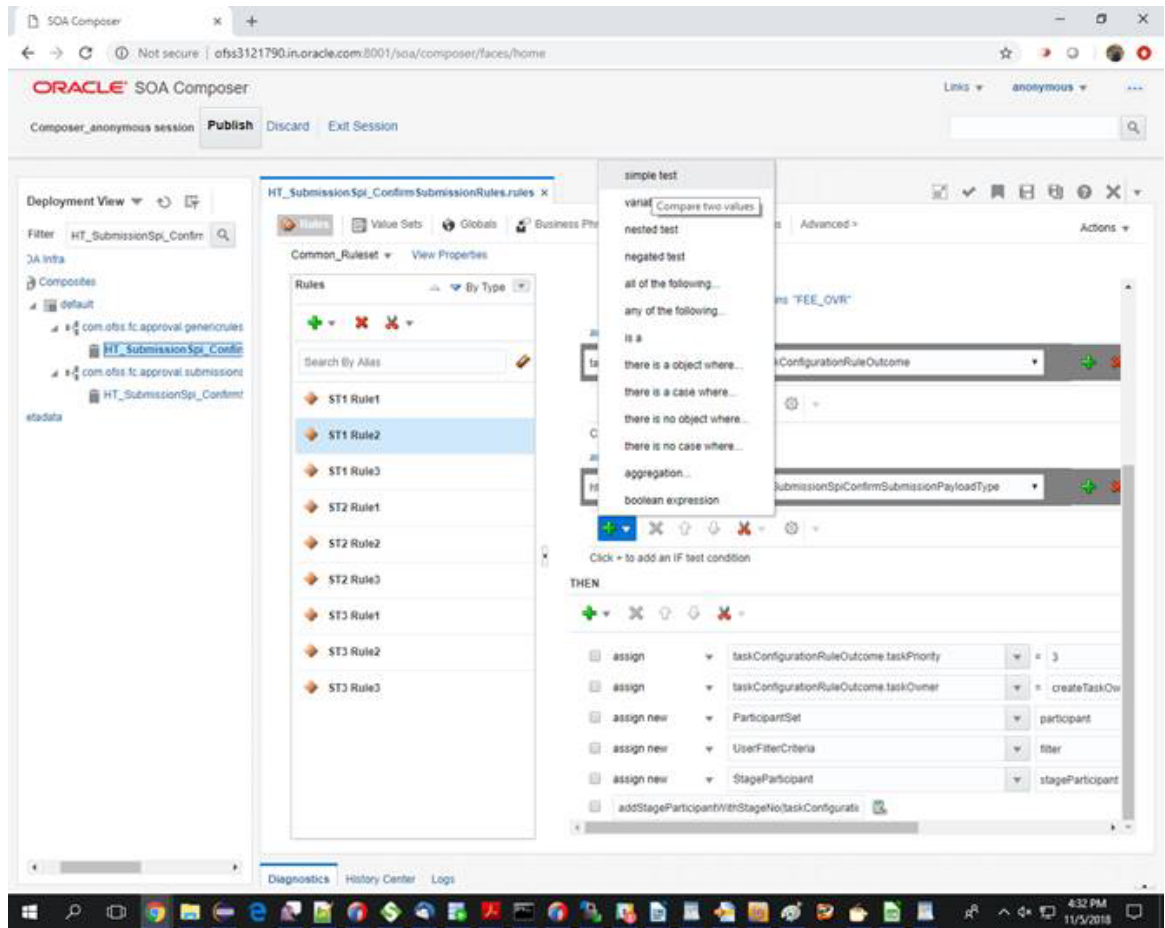
5. On clicking the Add Pattern, select fact type from drop down and then select payloadType.
Ex: select 'HTSubmissionSpiConfirmSubmissionPayloadType' from drop down as shown below:

Figure 11–16 Add pattern-Submission Payload Type



- After stage selection, select the specific rule where the condition needs to be updated. The existing condition can be updated or the new test condition (simple/variable) can be added. You can add new test using below option.

Figure 11–17 Add New Test

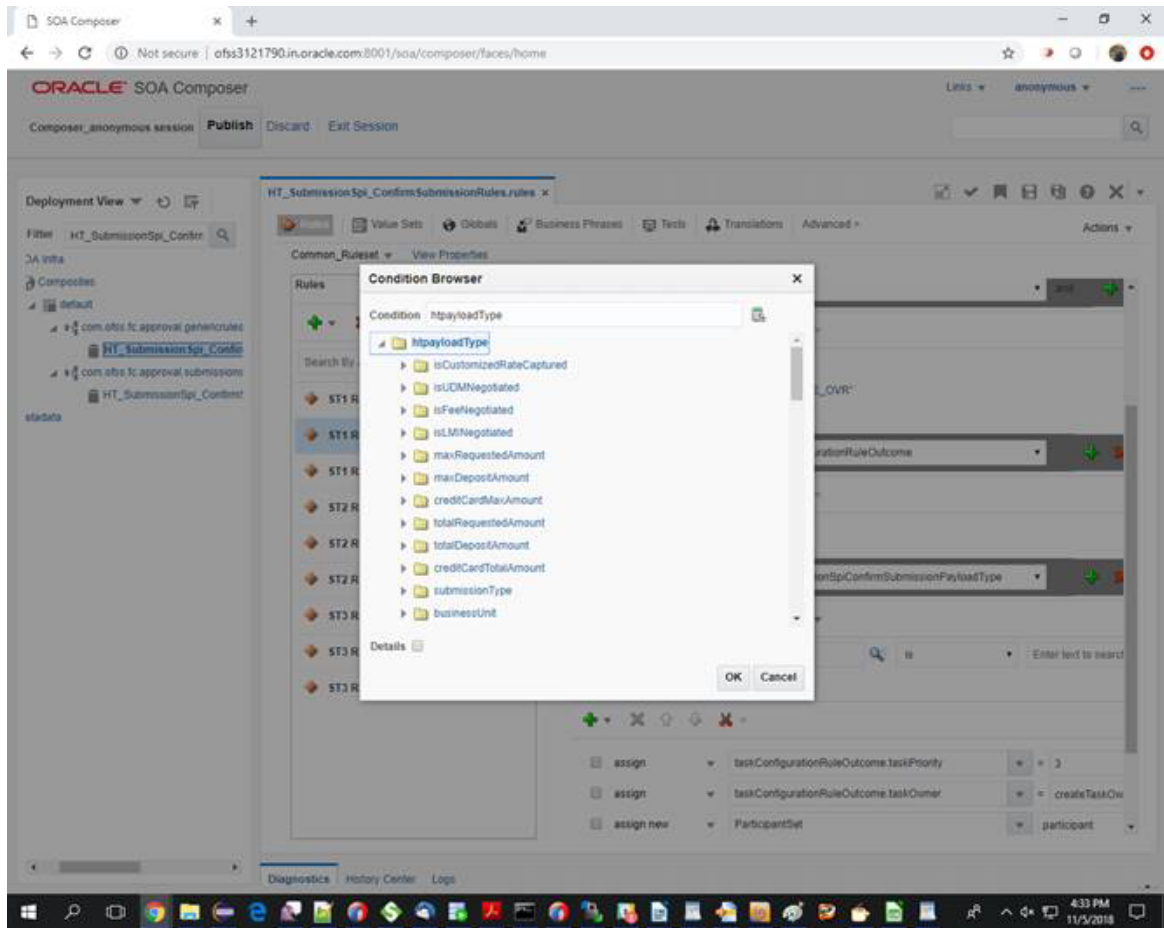


7. After selection of the test condition, the new expression row appears where the variable, the operator and the expression value can be selected.

On selection of the search button next to the variable select box, the condition browser opens where the specific task can be selected.

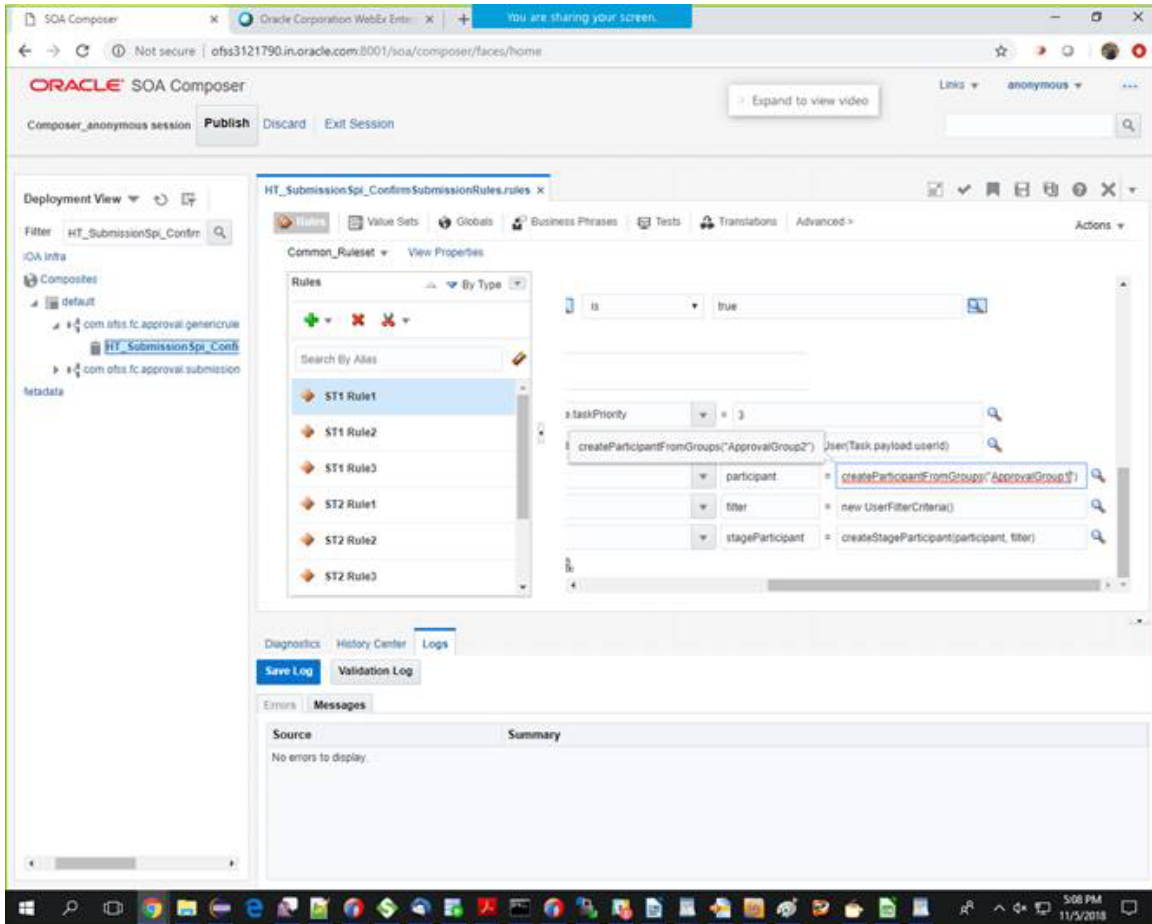
Select 'fact' in left side expression with appropriate value on right side.

Figure 11–18 Select Fact with Appropriate Value



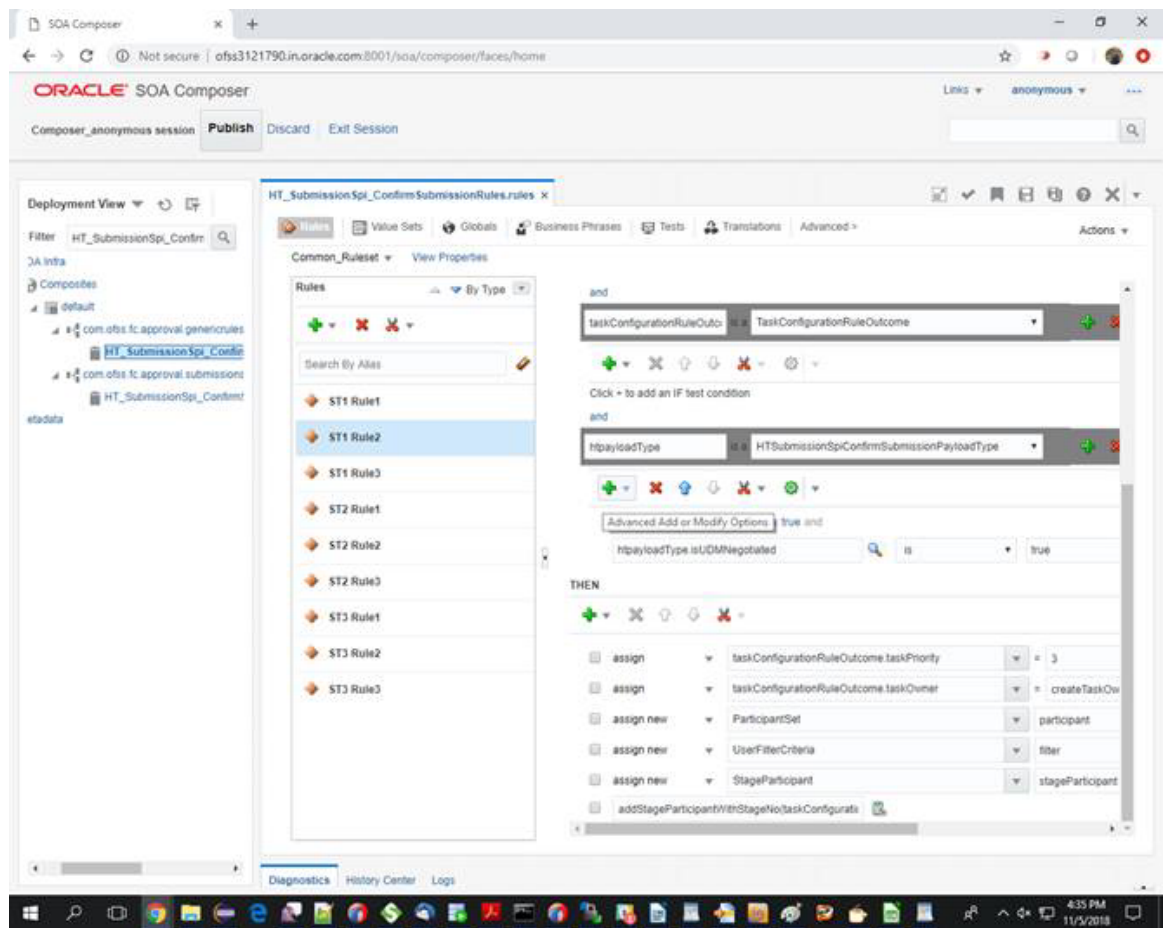
8. If you want to assign to a particular group then use below function:
`createParticipantFromGroups("ApprovalGroup2")`

Figure 11–19 Create Participant From Group



9. Validate rules file and then save all changes and publish.

Figure 11–20 Validate Rules File



11.8.2 Steps to Update the Business Rules in JDeveloper

Following are the steps to update the business rules in JDeveloper.

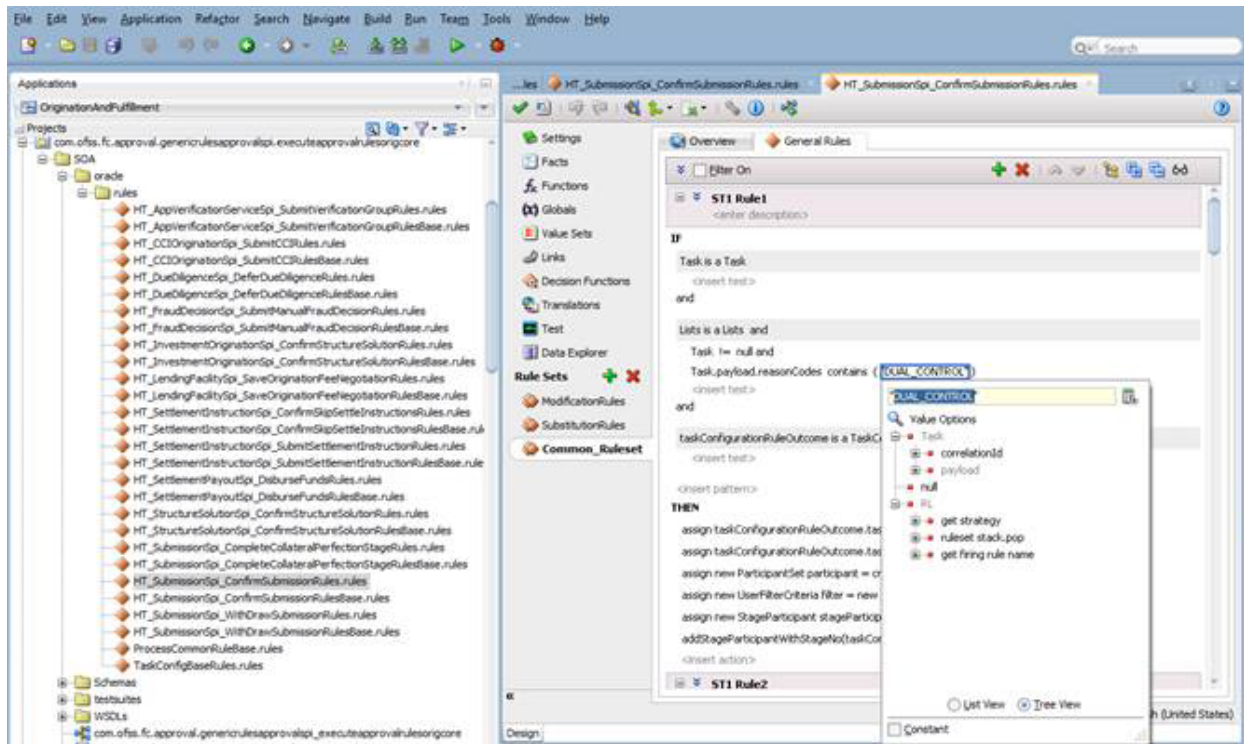
Step 1

Configure the JDeveloper in the customization option and the particular process jar import as the SOA project in the customizable mode. The details of this step are explained in the Oracle Banking Platform SOA Extensibility Guide in the SOA Customization section.

Step 2

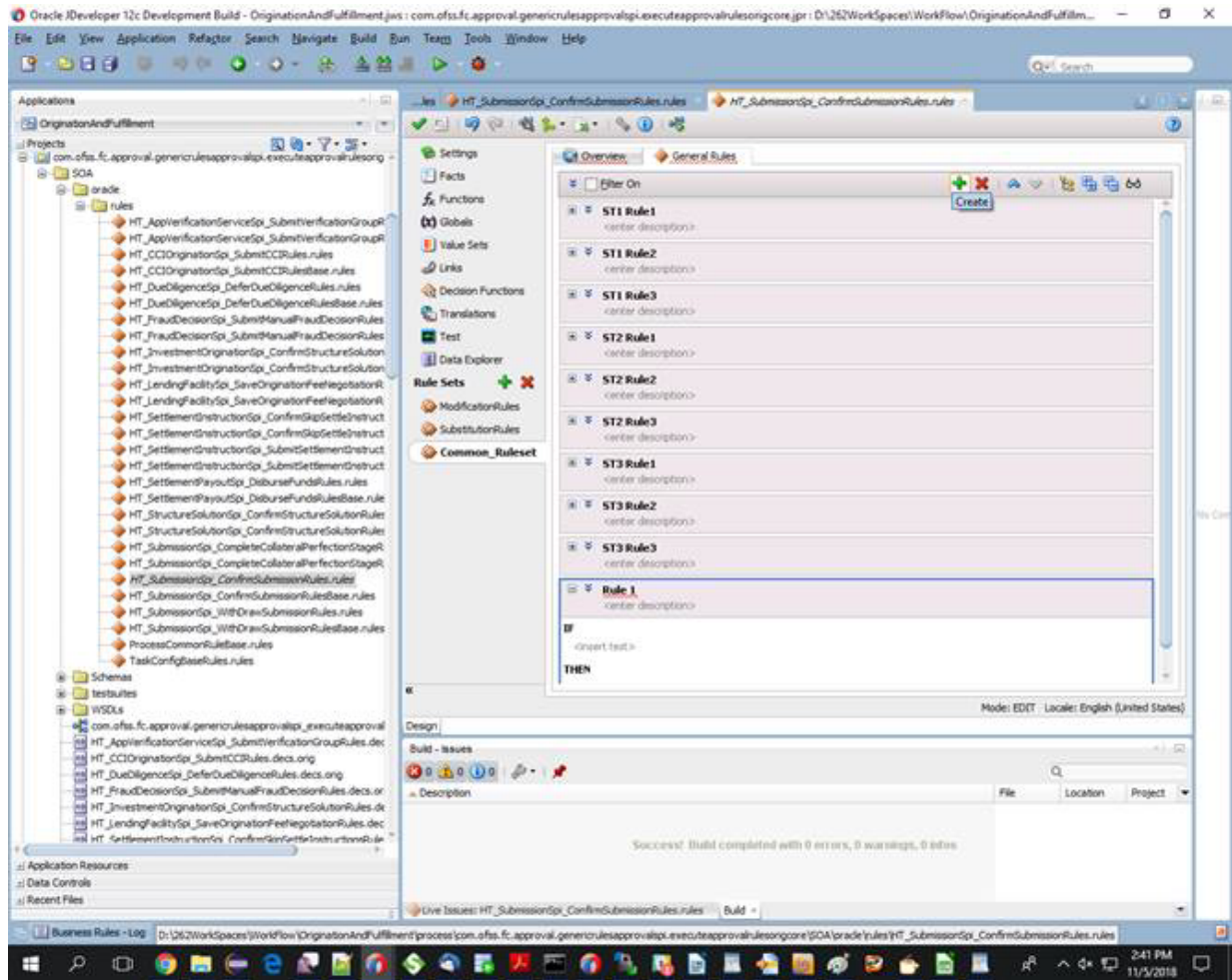
Expand the Business Rules which are inside oracle folder of your SOA project. You will see two .rules files in it. One will be <<HumanTaskName>>Rules.rules file and the other will be <<HumanTaskName>>RulesBase.rules file. Double Click and open the <<HumanTaskName>>Rules.rules file. The existing approval stages and rulesets will be available in the file.

Figure 11–21 Expand Business Rules

**Step 3**

Create a new rule in the format 'ST<Stage Number>_Rule<Rule Number>' by clicking the Create button in the Rulesets. The new rules/decision table can be added to a stage. Populate the rule with the conditions in 'if' and 'then' block.

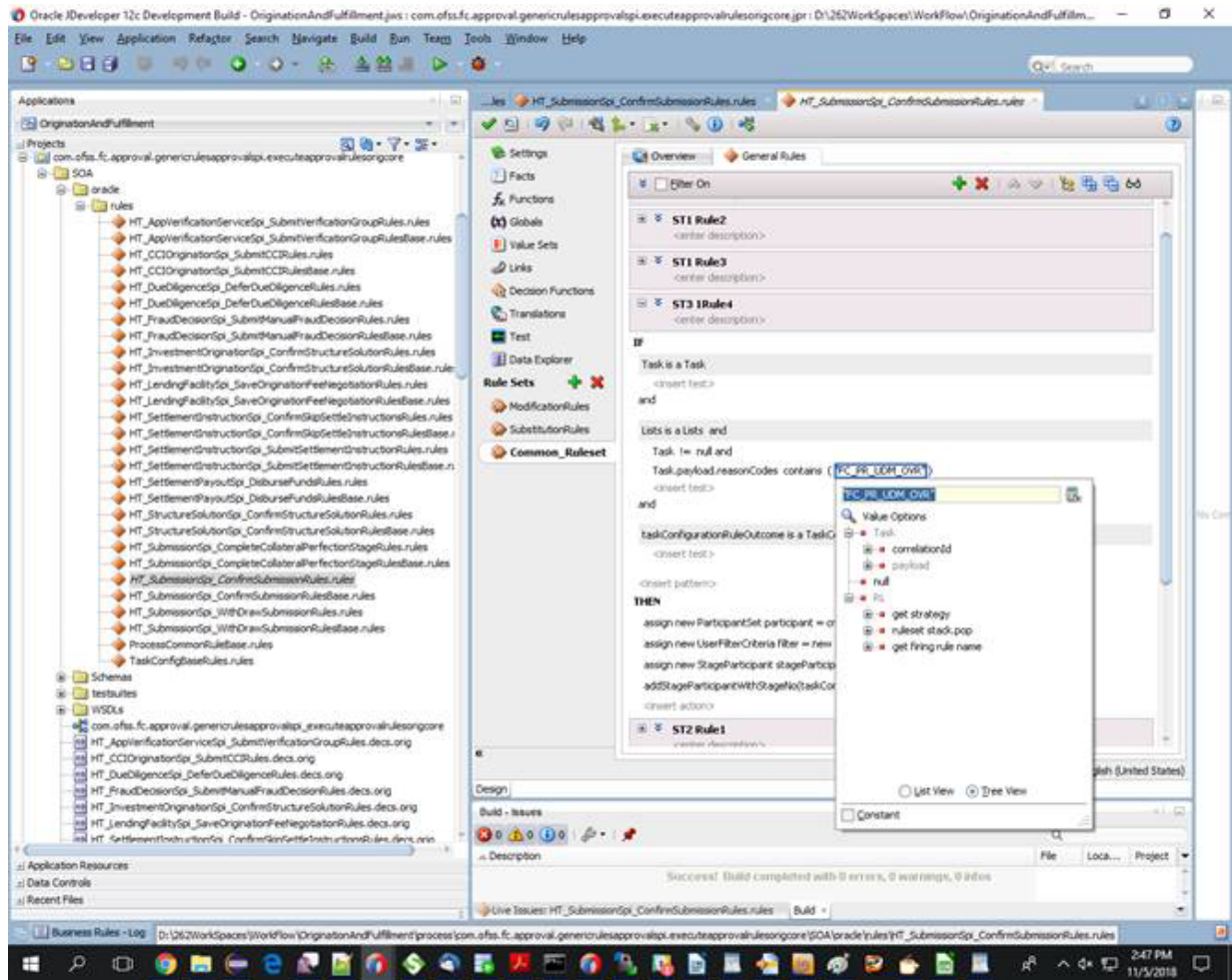
Figure 11–22 Create New Rule



Step 4

In the exiting stage, existing rule can also be added or modified.

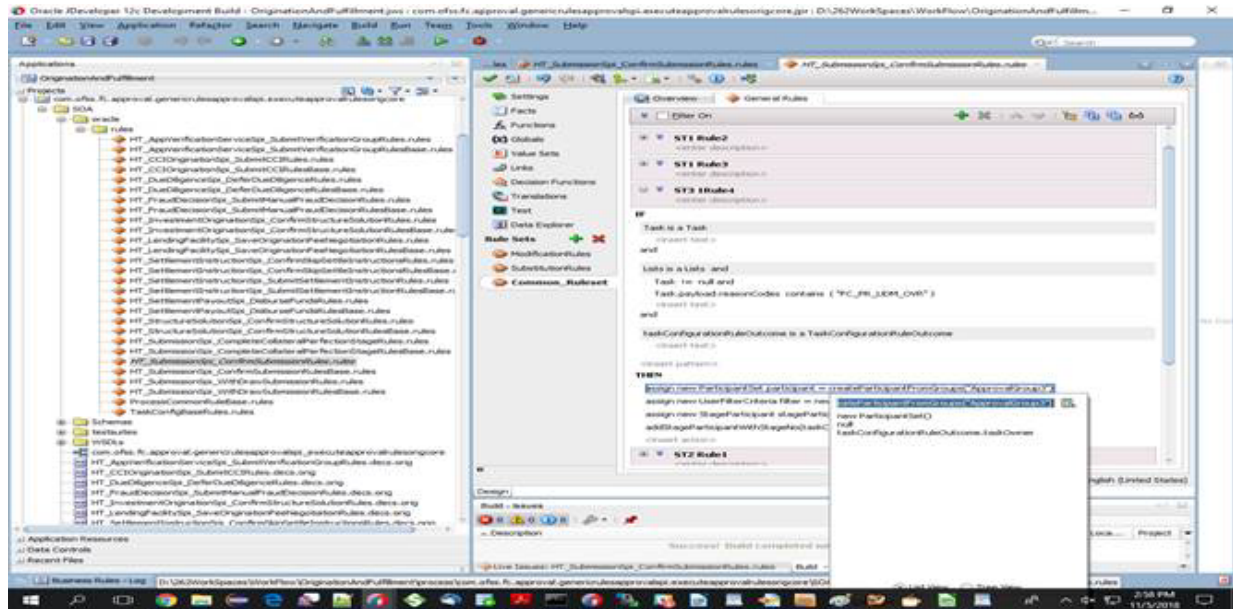
Figure 11–23 Existing Rule



Step 5

Change the participant or approval group by editing below function.

Figure 11–24 Change the approval group



Step 6

Deploy the project jar as explained in the Oracle Banking Platform SOA Extensibility Guide in the SOA Customization section.

Note

Make sure to activate rules which have to be executed by editing properties of each rules.

12 Loan Schedule Computation Algorithm

OBP application provides the extensibility by which the additional loan schedule computation algorithm can be added or customized based on client's requirement.

12.1 Adding a New Algorithm

For adding a new algorithm following additions need to be done.

LoanCalculationMethodType.properties contains list of available loan stage algorithms in the system in the form of key-value pairs. For example, ARM=ARM

This list is used as part of screen LNM43 to populate a drop down called Computation Formula.

An entry has to be made in this file to appear as a choice in the drop-down list.

Figure 12–1 Add New Algorithm

The screenshot displays the 'Installment Rule Details' screen in the Oracle Banking Platform. The interface includes a navigation menu at the top with options like 'Account', 'Back Office', 'CASA', 'Channel', 'Collection', 'LCM', 'Loan', 'Operational Services', 'Origination', 'Party', and 'Payment And Collection'. The main content area is divided into several sections:

- Rule Information:** Contains 'Rule ID' and 'Rule Name', both set to 'ABC'.
- Computation:** Contains 'Computation Formula' set to 'ARM'.
- Dates:** Contains 'Date Basis for Installment or Principal' and 'Date Basis for Interest', both set to 'Calendar'.
- Rounding Rules:** Contains 'Rounding Method' set to 'No Round' and 'Nearest Amount to Round To'.

At the bottom of the screen, there is a 'Show Modification History' link.

This screen is used to create a new Installment Rule. For example: ABC. We can choose the new algorithm for the new rule.

Figure 12–2 Create New Installment

The screenshot shows the Oracle Banking Platform interface for creating a new installment schedule. The main window is titled "Loan Schedule Type" and contains several sections:

- Schedule Details:** Fields for "Schedule Code" (sch) and "Schedule Name" (sch).
- Definition:** A table with columns: Stage Number, Stage Name, Installment Rule, Default Payments or Term, Installment Frequency, Interest Repayment Frequency, and Interest Capitalization Frequency. The table contains one row with Stage Number 1 and Installment Rule 102.
- Stage type:** Fields for "Stage Name", "Installment Rule" (102), "Installment Type" (Moratorium), and "Default Payments or Term".
- Interest Processing:** Fields for "End of Period Treatment" (Capitalize Arrears + UnCharged Interest), "Interest Compounding Frequency" (Monthly), and "Rest Period Frequency" (None).
- Installment Details:** A section for further configuration.

Screen LNM98 is used to create new schedule codes from existing instalment rules. A new schedule code can be made using the new instalment rule created above.

A schedule generator class is created to implement a schedule code. The property file **ScheduleCalculator.properties** stores this relation in the form:

Schedule_Type_Code=Schedule_Calculator_Class

If a new schedule generator class is created for the new schedule code above, an entry in this file has to be made.

```
Example: IOI-EIPI-PMI_IntOnly-Month_Pr-Month_Ann=
com.ofss.fc.domain.schedule.loan.generator.NewPrincipalRepaymentSc
heduleGenerator;
```

The key is the SCHEDULE_CODE column in the table FLX_SH_SCHEDULE_TYPE_B.

The **PrincipalRepaymentScheduleGeneratorFactory** reads this property file and creates an instance of the schedule generator class associated with the schedule type code passed. The following code snippet shows how it is done

```
IPrincipalRepaymentScheduleGenerator calculator = null;
String calculatorClassName = calculators.get(scheduleTypeCode);
calculator = (IPrincipalRepaymentScheduleGenerator)
ReflectionHelper.getInstance().getClassInstance
(calculatorClassName);
```

```
// If schedule calculator is not found then do nothing
```

```

if (calculator == null) {
    calculator = new PrincipalRepaymentScheduleGenerator();
}

```

Currently, in the application this property file is empty and hence an instance of `PrincipalRepaymentScheduleGenerator` is returned by default.

The new schedule generator class has to implement the interface `IPrincipalRepaymentScheduleGenerator` which is the base for all schedule generators.

The important methods in it are:

```

public SortedMap<Integer, PrincipalRepaymentPeriod> defineStages
    (SortedMap<Integer, PrincipalRepaymentPeriod> repaymentStages,
    AccountScheduleAttributesDTO accountParameters, Money
    amountForScheduleGeneration, Date onDate);
public LoanScheduleCalculatorOutputData defineSchedule(Date
    definitionDate, SortedMap<Integer, PrincipalRepaymentPeriod>
    repaymentStages, AccountScheduleAttributesDTO accountParameters,
    SortedMap<LoanInterestType, List<NetRateDTO>> interestRates, Money
    mountForScheduleGeneration);
public LoanScheduleCalculatorOutputData generateRepaymentRecords
    (Date generationDate, SortedMap<Integer, PrincipalRepaymentPeriod>
    repaymentSchedule, AccountScheduleAttributesDTO accountParameters,
    Money totalPrincipalToRepay, SortedMap<LoanInterestType,
    List<NetRateDTO>> interestRates, List<PrincipalScheduleTransaction>
    scheduleTransactionHistory, SortedMap<Date,
    PrincipalsScheduleInterestBase> interestBaseHistory, SortedMap<Date,
    Money> feeDetails);

```

The method `generateAndSavePrincipalSchedule()` of `ScheduleApplicationService` creates and processes the instance of a schedule generator as follows:

```

IPrincipalRepaymentScheduleGenerator scheduleGenerator =
    PrincipalRepaymentScheduleGeneratorFactory.getInstance
    ().createScheduleGeneratorInstance
    (accountParameters.getScheduleTypeCode());

```

The methods in the schedule generator call the business logic for the instalment rules (stage algorithms) part of the schedule code. This logic is written in a Stage generator class. New Stage generator class has to be created for the new algorithm created above.

For example, `EPIARMRepaymentStageGenerator.class` is created for EPI and ARM.

This class has to implement interface **`IPrincipalRepaymentPeriodGenerator`** which is the base for all stage generators. The important methods in it are:

```

public void defineStage(LoanRepaymentStageDTO repaymentStage);
public void define(LoanRepaymentStageDTO
    repaymentStage, AccountScheduleAttributesDTO accountParameters, Date
    definitionDate, List<NetRateDTO> interestRates, SortedMap<Integer,

```



```
LoanRepaymentStageDTO> allRepaymentStages, SortedMap<Date,
PrincipalScheduleInterestBase> interestBaseHistory,
List<PrincipalScheduleTransaction> scheduleTransactionHistory);
public SortedMap<Date, LoanRepaymentRecordDTO> generate
(LoanRepaymentStageDTO repaymentStageToBeGenerated,
AccountScheduleAttributesDTO accountParameters, Date
generationDate, List<NetRateDTO> interestRates, SortedMap<Integer,
LoanRepaymentStageDTO> allRepaymentStages, SortedMap<Date,
RepaymentDate> repaymentDates, SortedMap<Date,
LoanRepaymentRecordDTO> allRepaymentRecords, SortedMap<Date,
PrincipalScheduleInterestBase> interestBaseHistory,
List<PrincipalScheduleTransaction> scheduleTransactionHistory,
SortedMap<Date, Money> feeDetails);
```

The entry for the new Stage generator class has to be made in **StageCalculator.properties**.

```
For example,
ARM=com.ofss.fc.domain.schedule.loan.generator.EPIARMRepaymentStageGenerator
```

The **PrincipalScheduleRepaymentPeriodGeneratorFactory** class reads this property file and based on the installment rule passed as parameter creates an instance of its corresponding stage generator class. The following code snippet shows it

```
IPrincipalRepaymentPeriodGenerator stageGenerator =
PrincipalScheduleRepaymentPeriodGeneratorFactory.getInstance()
.createStageGeneratorInstance(repaymentStage.getInstallmentRule())
```

12.2 Consuming Third Party Schedules

As mentioned above the **PrincipalRepaymentScheduleGeneratorFactory** reads the property file `ScheduleCalculator.properties` which has entry for the schedule generator class to be used for a schedule code. For using third party schedule algorithms, an entry in this file has to be made against the required schedule codes.

```
IOI-EIPI-PMI_IntOnly-Month_Pr-Month_Ann=
com.ofss.external.ScheduleAlgoExt.XYZScheduleGenerator;
```

13 Facts and Rules Configuration

This chapter explains the facts and rules configuration details.

13.1 Facts

Fact (in an abstract way) is something which is a reality or which holds true at a given point of time. Business rules are made up of facts.

A fact can be classified in two ways:

- **Literal Fact** - Any number, text or other information that represents a value. It is a fixed value. For example, 100, 2.95, "Mumbai".
- **Variable Fact** - A fact whose value keeps changing over a period of time For example, Account Balance, Product Type.

For example, If a customer maintains an Average Quarterly Balance of Rs.10,000 then waive off his quarterly account maintenance fees. Here, the Average Quarterly Balance is a variable fact while the Rs.10,000 is a literal fact.

13.1.1 Type of Facts

There are two types of facts:

- Direct Facts with input name value pair
- Derived Facts

Services will be exposed for various operations on the facts. These services are broadly classified into two types:

- Fact Inquiry Service
- Fact Derivation Service

For deriving the fact value, different type of datasource can be used:

- Java DataSource - Derivation from Java class
- JPQL DataSource - JPQL Query column
- JDBC DataSource - SQL Query column
- DbFunction DataSource - Derivation from database function

Fact Definition can be further categorized into:

- **Fact Value Definition** - Definition to Derive Fact Value. It is used mostly in Rule Execution.
- **Fact Enum Definition** - Definition to Derive Permissible values for a fact. It is used mostly in Rule Creation.

13.1.2 Facts Vocabulary

Facts Vocabulary is a list or collection of all facts pertaining to a specific field or domain. A standard vocabulary of facts aids users in defining their business rules. For example, the Facts Vocabulary of the Banking domain can contain common and familiar facts such as Account Balance, Customer Type, Product Type, Loan-To-Value Ratio. The Facts Vocabulary of the Cards domain may contain common facts such as Total Credit Limit, Available Credit Limit, Available Cash Limit.

A vocabulary is defined for variable facts. Each fact has a definition and can have source information.

Definition

The fact definition indicates common properties of the fact such as its name, its data type, which domain, domain category and group it belongs to, key for retrieving value and a brief description.

Variable facts would be defined for a domain and a domain category. Domain categories are the sub-systems inside a domain. For example, Lending, Term Deposits, Demand Deposits are the categories of Banking domain. There are some variable facts which would be common across all the categories in a given domain. For example, Customer and Bank data is common for all the categories of Banking domain. Such facts can be classified under a special category called "Global".

The facts are further categorized under various groups. One fact can belong to one or more Groups. For example, In a Banking domain, Customer Type, Birth Date, Gender are Global facts belonging to the group Individual Customer Details. Account Balance, Account Opening Date are facts in Lending category belonging to the group Account Details. Loan-to-value (LTV) ratio, Sanctioned Amount are Facts in Lending category and belong to multiple groups such as Consumer Loan, Home Loan, Agriculture Loan. There are some variable facts which do not really fall into any specific group, such facts are classified under a special group called "Others".

A variable fact value can be received as input from the consumer of eRules in the form of key-value pair, the key here is defined as *RetrievalKey*. The fact will also have a data source for value derivation in case the fact value is not an input.

Some variable facts can have a permissible list of values defined and the rule creator will be restricted to use only those values which are defined in the permissible list of a given fact. All facts will have a *FactValueType* defined as either *Enumerated* (indicates that the fact has a permissible list of values) or *OpenEnded* (indicates that the fact value is a free text). For facts with *FactValueType* as *Enumerated*, data source information will be defined in the vocabulary to derive the list of values.

Variable facts will have a grouping based on *BusinessDataType*. For example, Variable facts like Transaction Amount, Sanctioned Amount, and Disbursed Amount can be grouped under "Amount". Variable facts like Available Balance, Book Balance belong to "Balance" *BusinessType* and so on.

These *BusinessDataType* will in turn have *PrimitiveDataType*. For example, Amount and Balance will have *PrimitiveDataType* as double.

With the help of *BusinessDataType* grouping a list of facts belonging to a particular group can be displayed for user selection while defining rules, rate charts, policies and so on. During actual rule execution the respective *PrimitiveDataType* (that is, int, double, String and so on) of the *BusinessDataType* will be used.

Literal facts will only have a *PrimitiveDatatype*.

Source

Some facts can be derived, if they are not received as input. Also associated with some facts is a list of permissible values for the fact at the time of rule/policy definition. All these information forms the part of source data. The Fact Derivation layer is responsible for deriving the value of a fact and the list of permissible values for the fact based on source information defined in the vocabulary.

Deriving Enumeration (applicable list of values) for a Fact

A Variable fact can hold any value at a given point of time. But some can have a standard set of applicable values defined and the value held by such facts would be always within the range of this list of values.

For example, Account Balance as a variable fact can hold any value at a given point of time, a set of values cannot be defined for such facts. Hence, no list of permissible values will be defined for Account Balance. However, the variable fact Customer Gender can have only one of two possible values namely - Male or Female.

While defining the rules, the permissible list of values will be derived for such facts and user selection will be restricted to this list.

Deriving Value for a Fact

During rule execution, a set of fact information will be sent by the consumer of eRules in the form of key-value pair. But this might not be a complete set of fact information required for executing pricing rules. Hence some facts will have to be derived if they are not received as input.

During rule execution, the required facts would be determined, value will be fetched from *RetrievalKey* of the fact if received as input else the value will be derived.

13.1.3 Generation of Facts using Eclipse Plug-in

The fact objects can be generated either by populating the database tables directly or by using the eclipse plug-in. This plug-in is created for fact generation purpose in OBP application.

A local host server needs to be configured in eclipse before processing for configuration of the fact plug-in. For fact generation purpose, the following steps need to be followed.

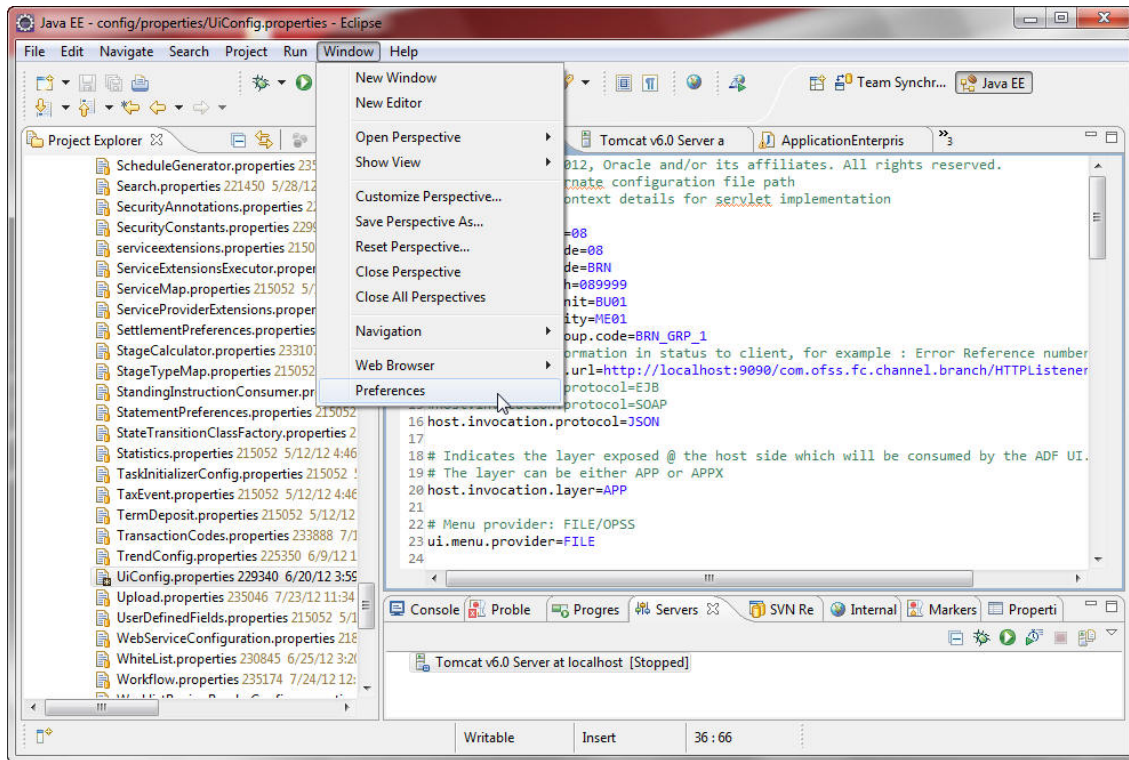
Get the Fact Plugin from the development team.

Put the latest fact generation plugin (**com.ofss.fc.util.plugin.fact_x.x.x.jar**) in the plug-in folder of eclipse.

Restart Eclipse

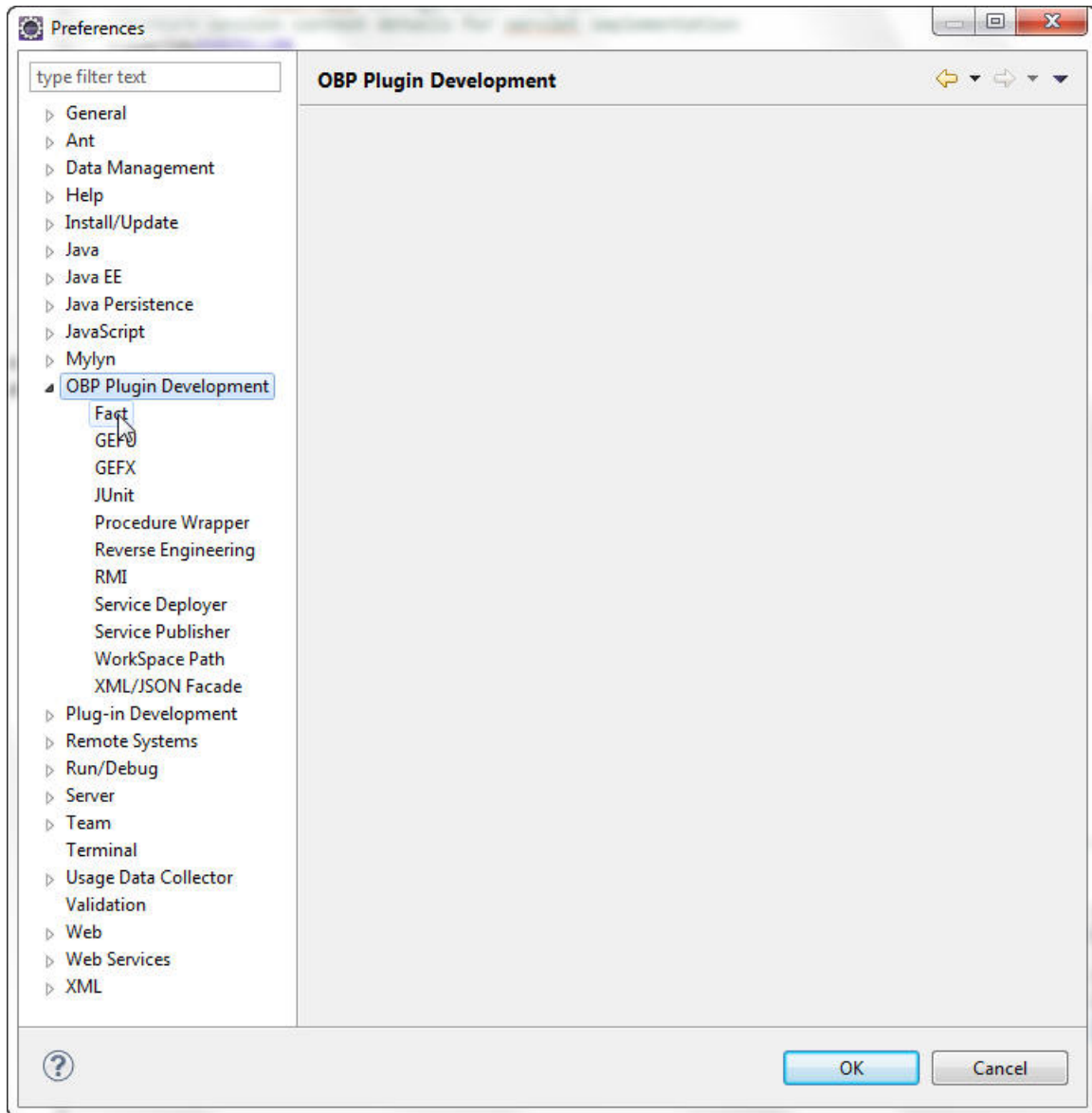
1. In eclipse, go to Window -> Preferences.

Figure 13–1 Select Window Preferences



2. Now in Preferences Window, go to **OBP Plugin Development** -> **Fact**.

Figure 13–2 Window Preferences - OBP Plugin Development



3. Enter the values as mentioned:

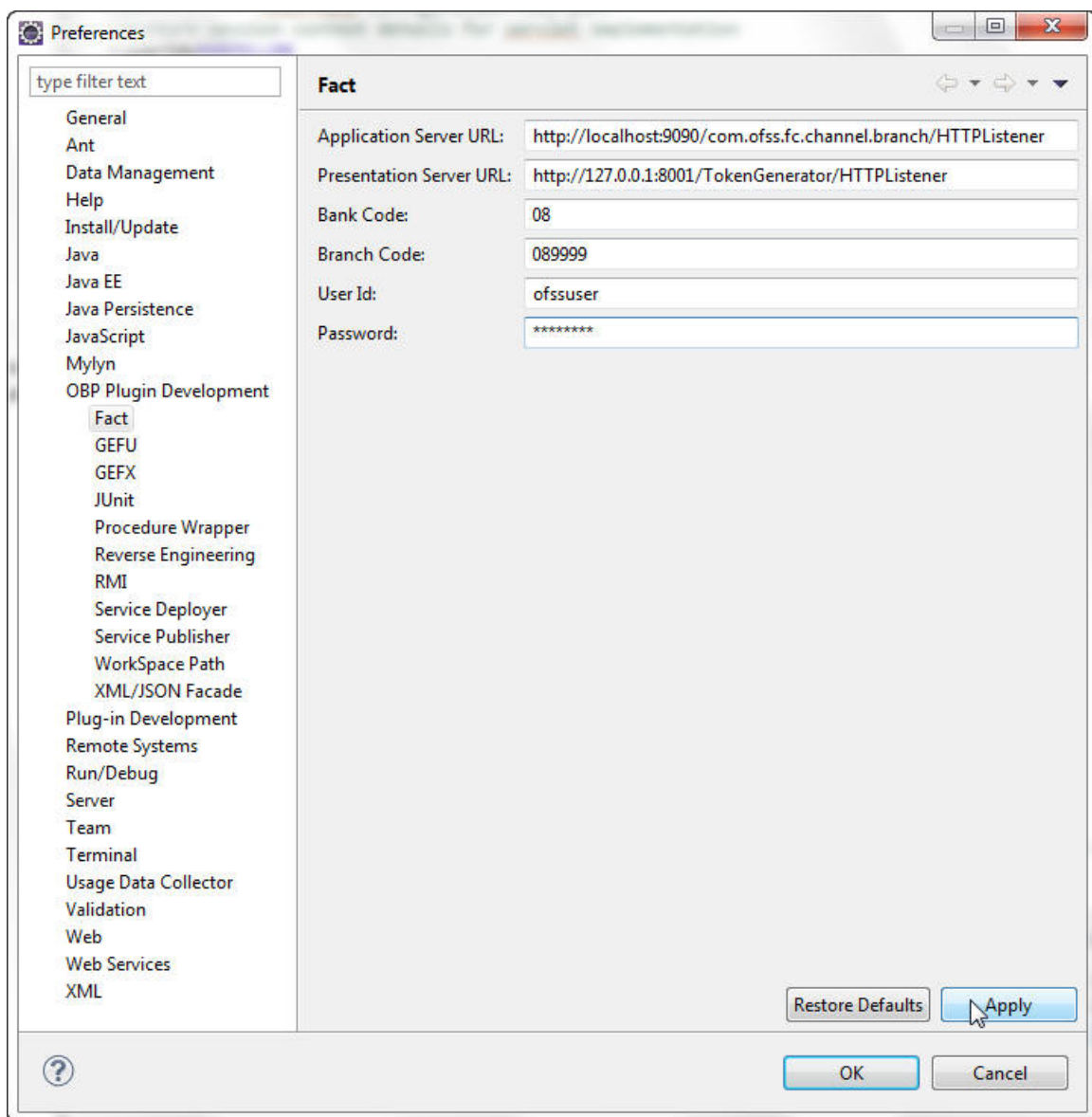
- **Application Server URL:** Local Host Server Listener URL
Example: `http://localhost:9090/com.ofss.fc.channel.branch/HTTPListener`
- **Presentation Server URL:** Token Generator Application URL
Example: `http://127.0.0.1:8001/TokenGenerator/HTTPListener`

If using the plug-in in local eclipse workspace, it will not be used, but a value must be provided, you can use it from example value.

For security configured environment, it will be used, and then it should be provided properly.

- **Bank Code:** Bank code (Example: 08)
- **Branch Code:** Branch Code (Example: 089999)
- **User Id:** username (Example: ofssuser)
- **Password:** Password (Example: welcome1)

Figure 13–3 Enter the Preferences Fact values



4. Now click **Apply**, and then click **Ok**.
5. Open Fact.properties and modify:
 - **aggregateCodeFilePath**: The location of host workspace.
 - **sourceFilePath**: The location of src directory of com.ofss.fc.fact project.

Figure 13–4 Fact Properties - aggregateCodeFilePath

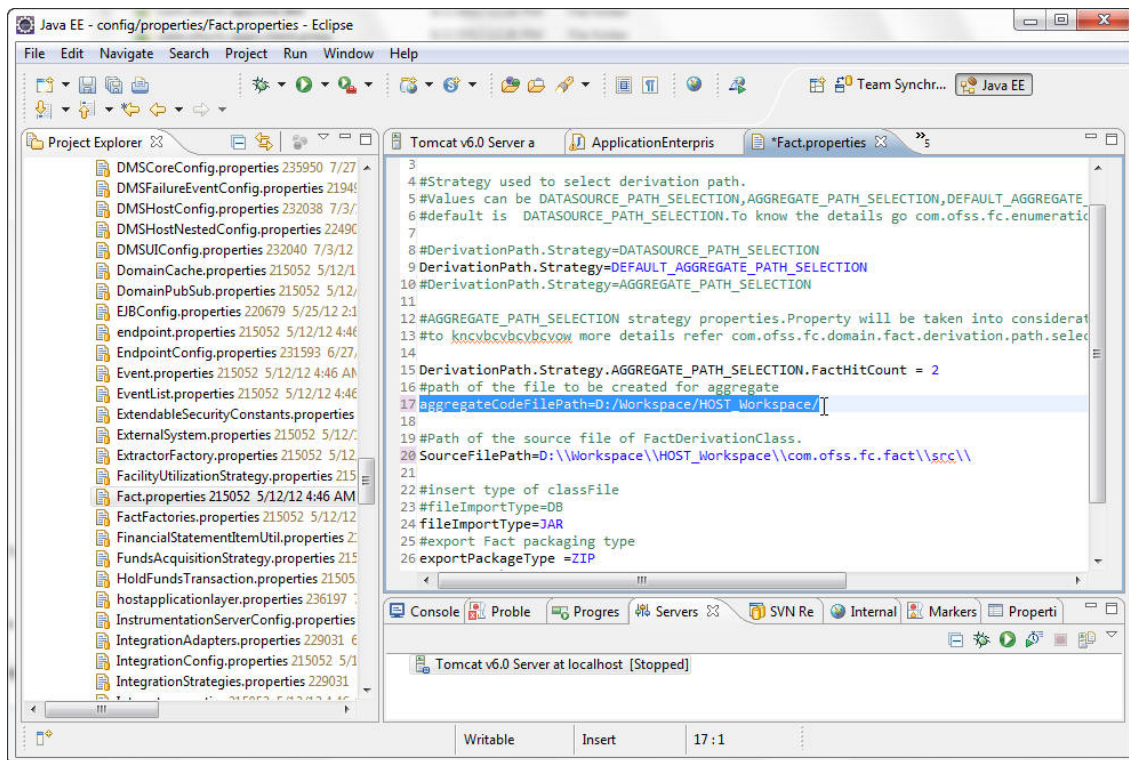
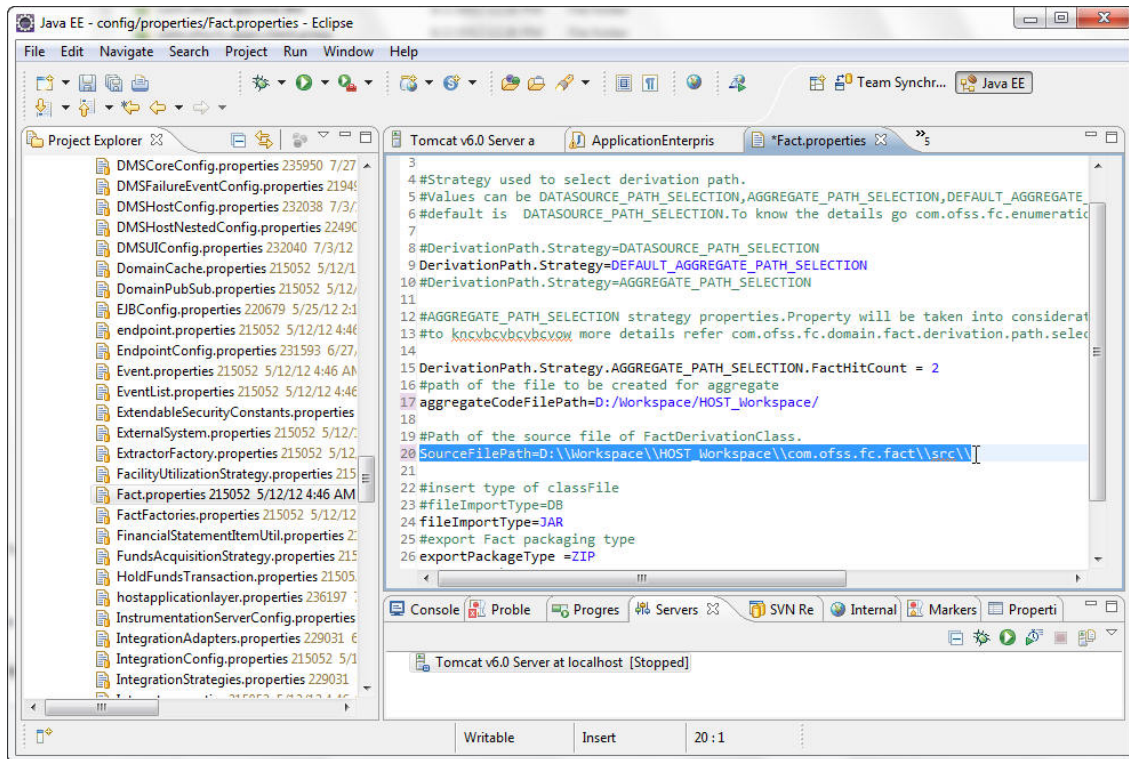
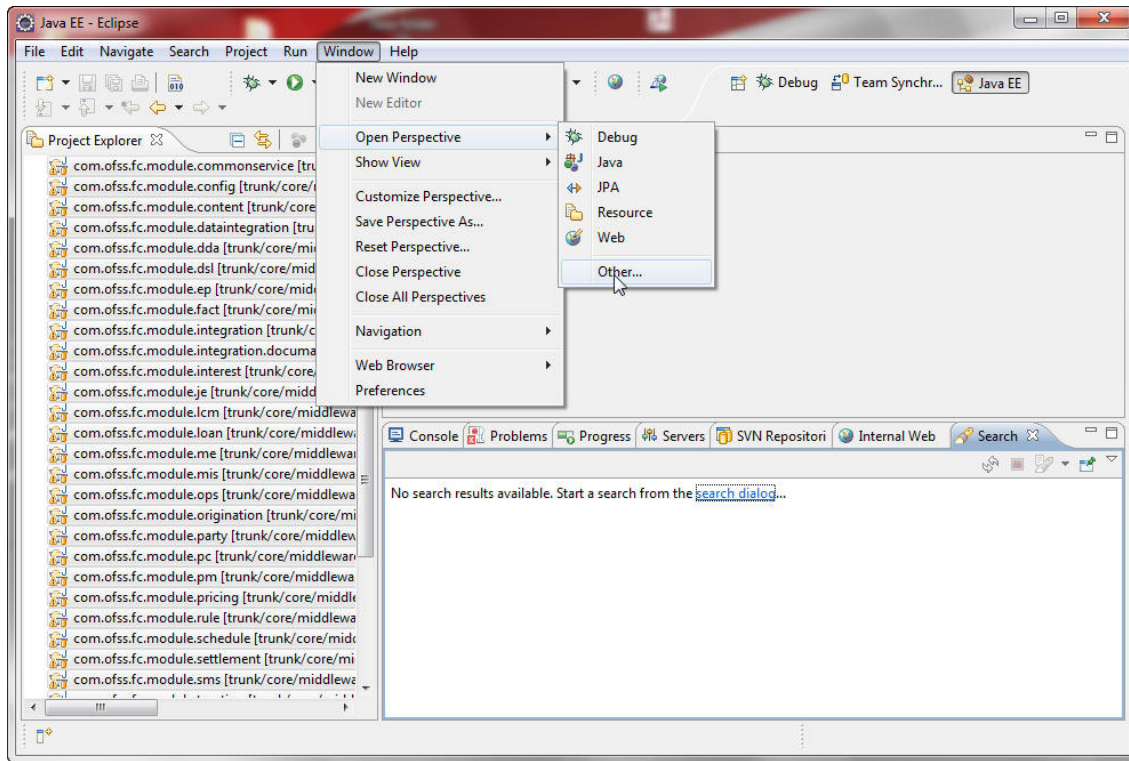


Figure 13–5 Fact Properties - sourceFilePath



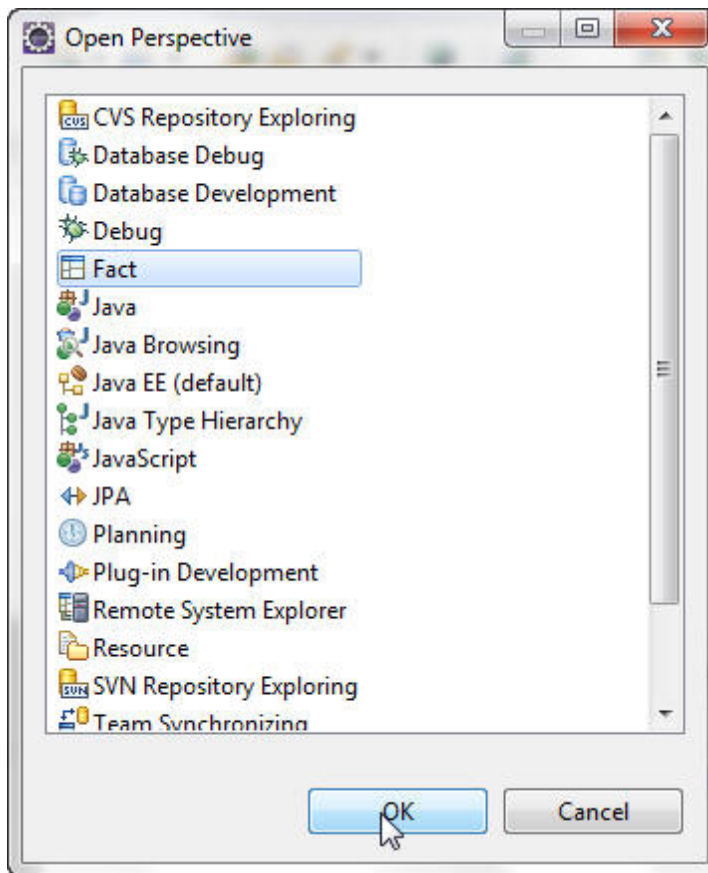
6. Now start the Host server.
7. In eclipse, go to Window -> Open Perspective -> Other.

Figure 13–6 Start Host Server

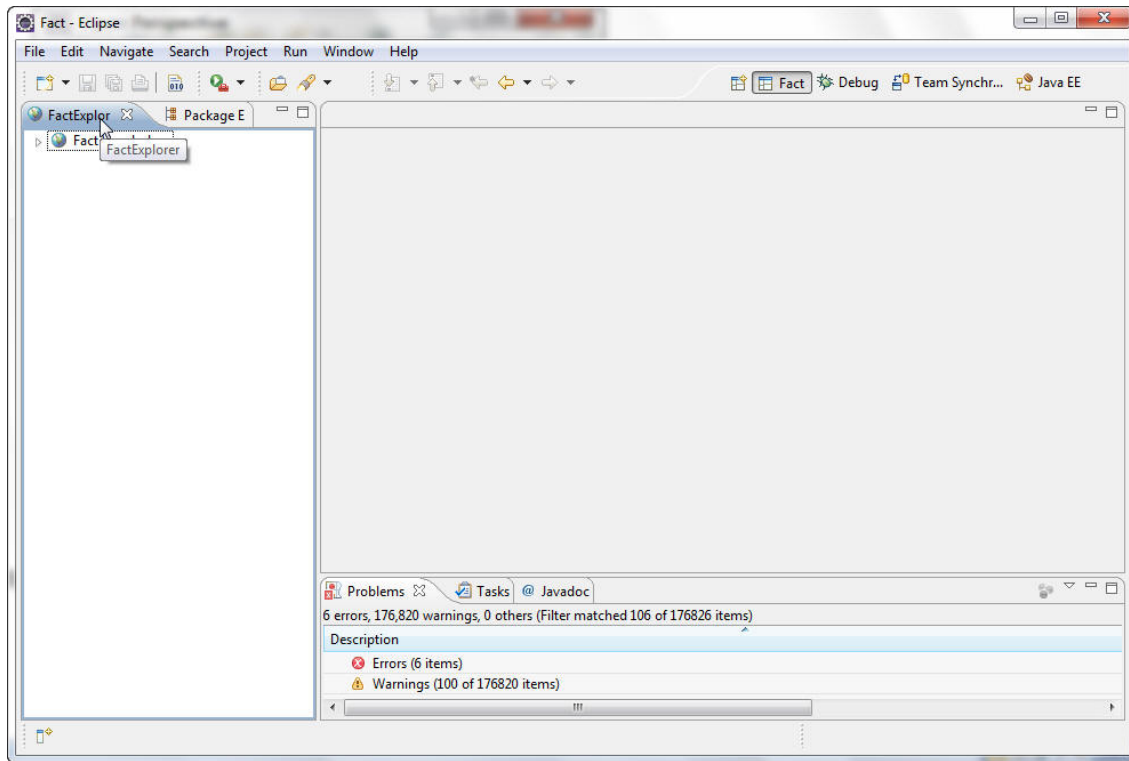


8. Now in Open Perspective window select **Fact**.
9. Click **Ok**.

Figure 13–7 Select Open Perspective value

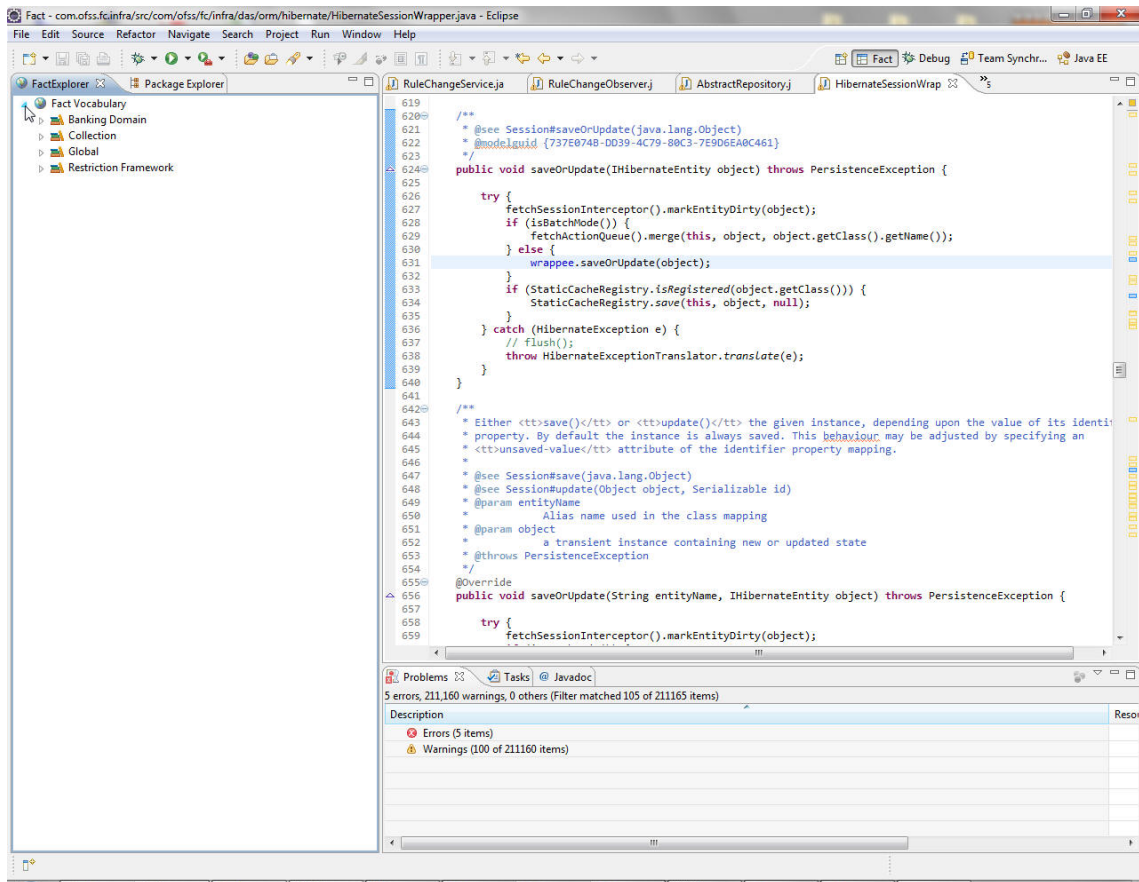


It will open **Fact Explorer** perspective, where **Fact Vocabulary** is available.

Figure 13–8 Fact Explorer

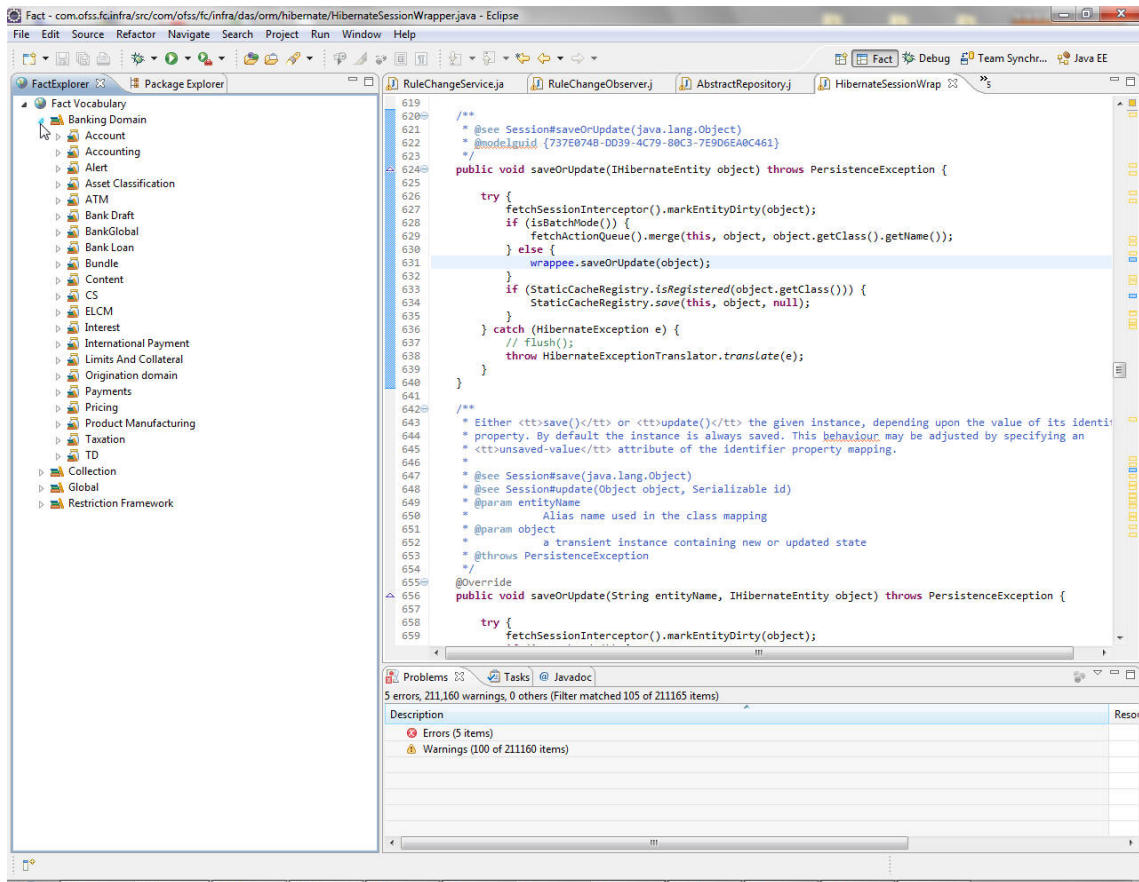
10. Now refresh and expand **Fact Vocabulary**. Expanding Fact Vocabulary will show the **Domain** names.

Figure 13–9 Fact Vocabulary



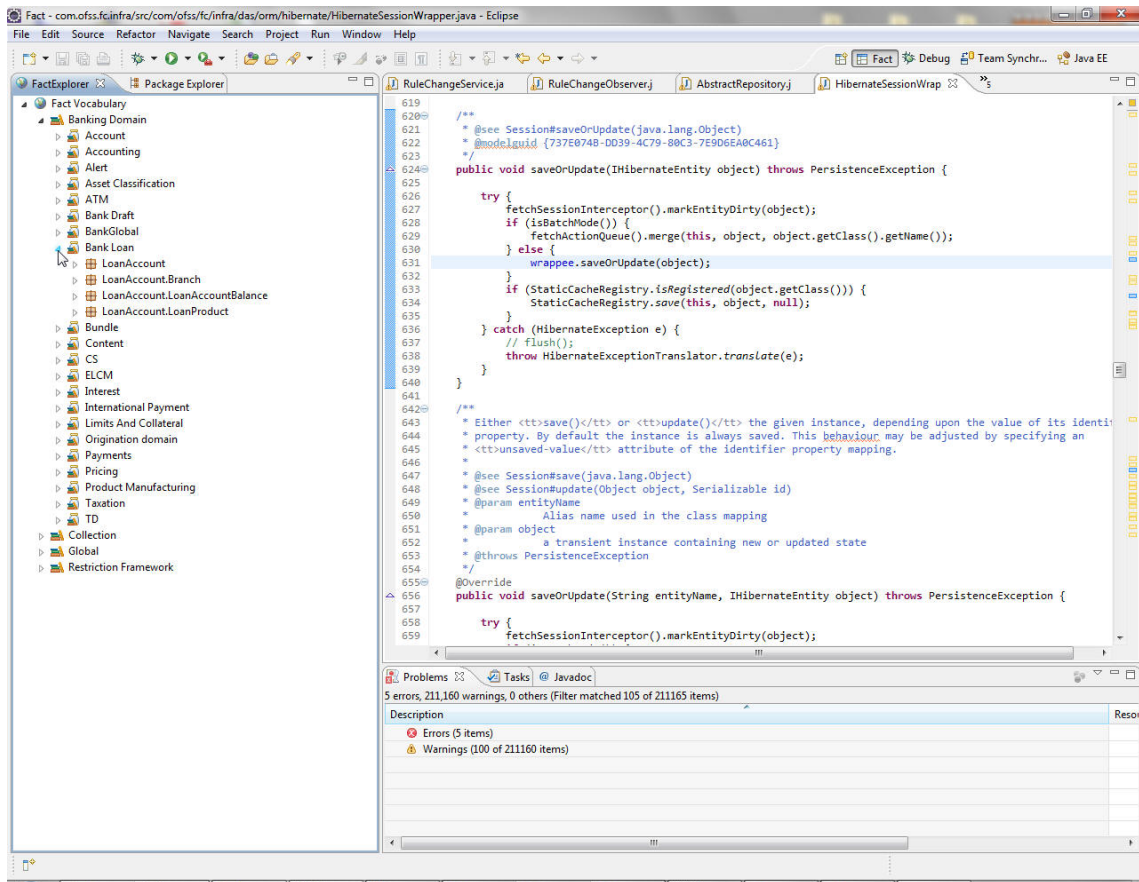
Each Domain contains its **Domain Category** names.

Figure 13–10 Domain Category



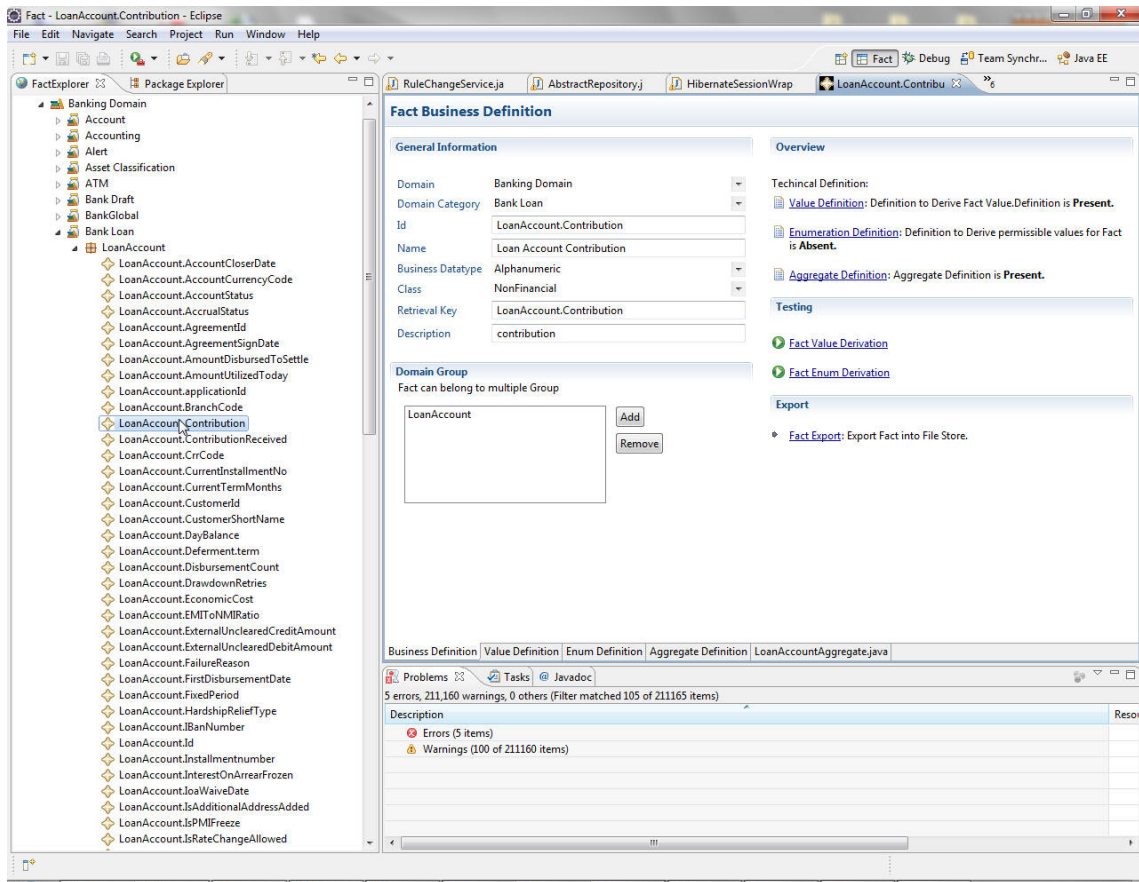
Each Domain category contain its **Fact Groups**

Figure 13–11 Fact Groups



Each Fact Groups contains its **Facts**.

Figure 13–12 Facts



- To see the details of any fact, just double-click it. The details will be shown in a fact window containing some tabs. Move to each tab to show the details.

Figure 13–13 Business Definition Tab

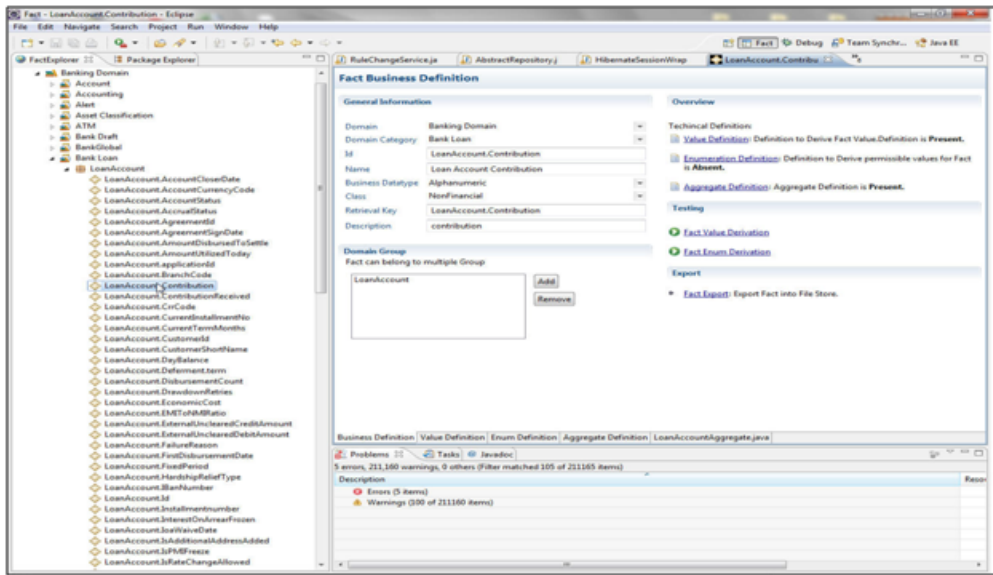


Figure 13–14 Value Definition Tab

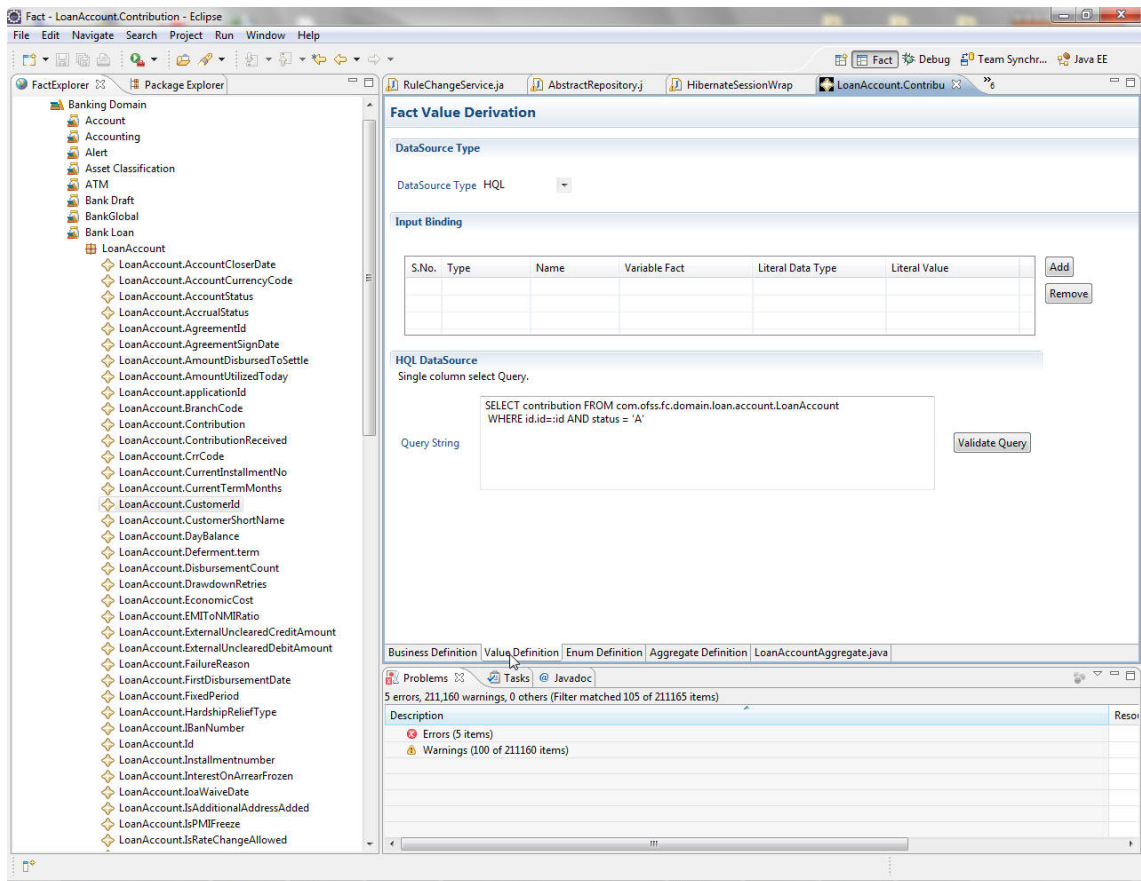


Figure 13–15 Enum Definition Tab

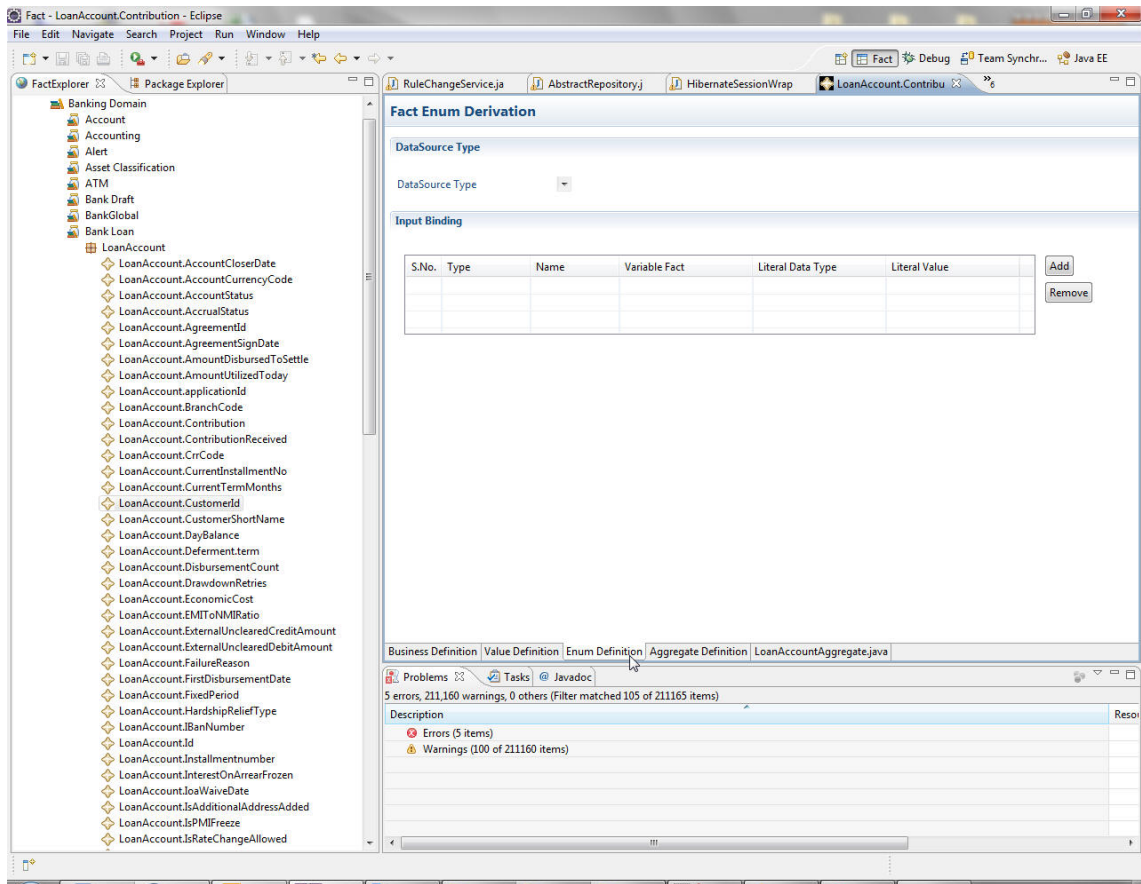


Figure 13–16 Aggregate Definition Tab

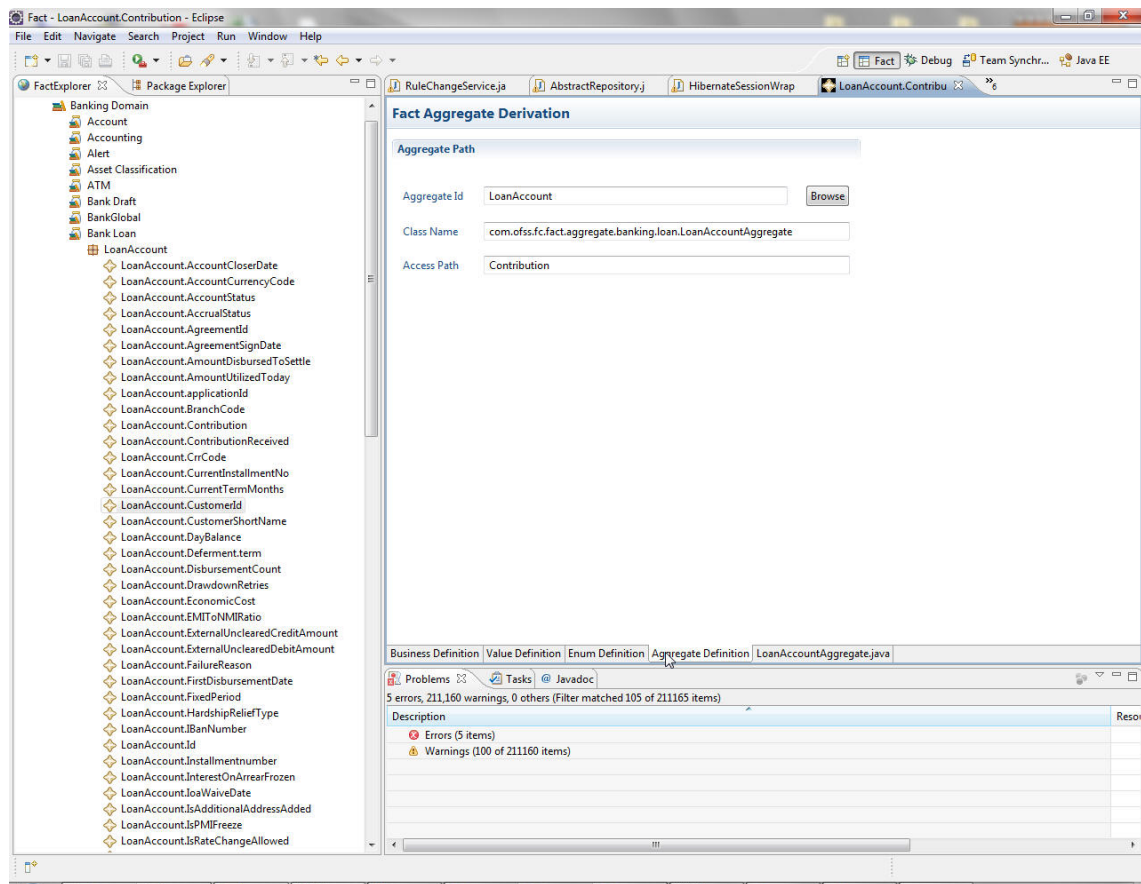
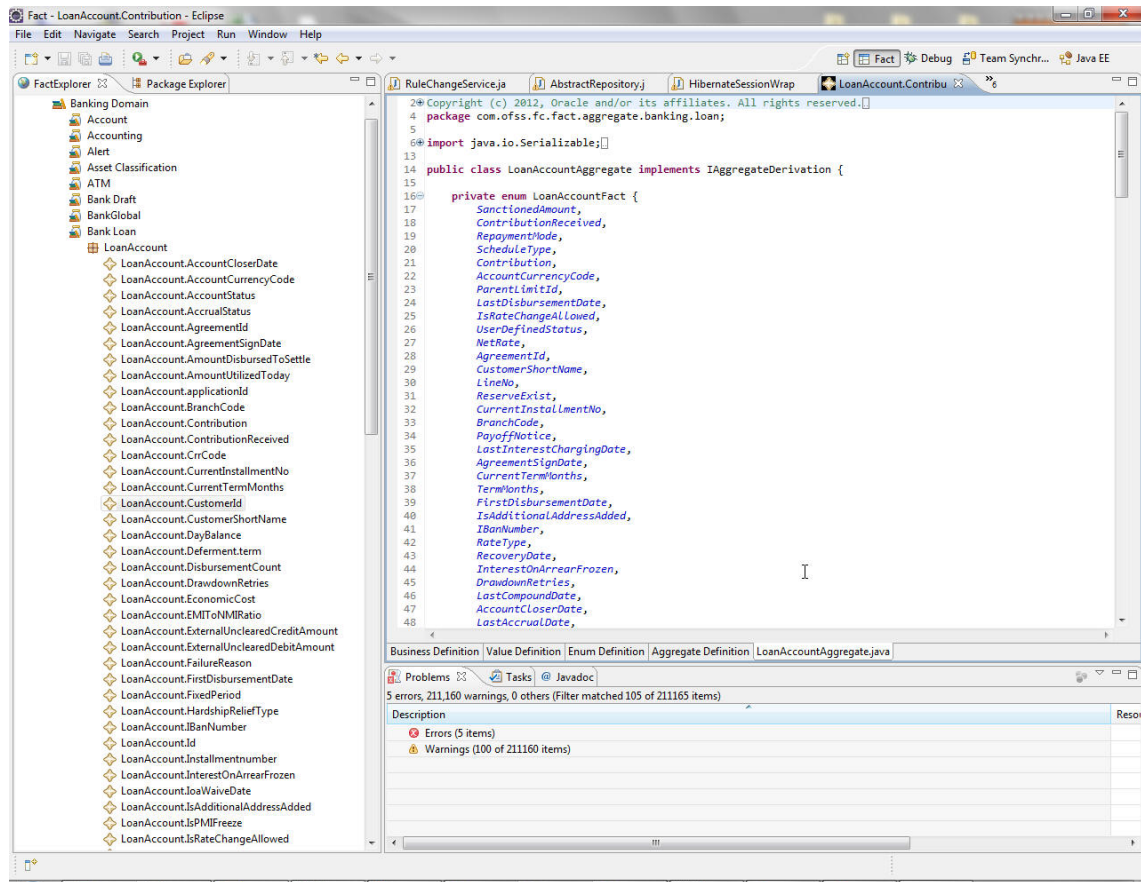
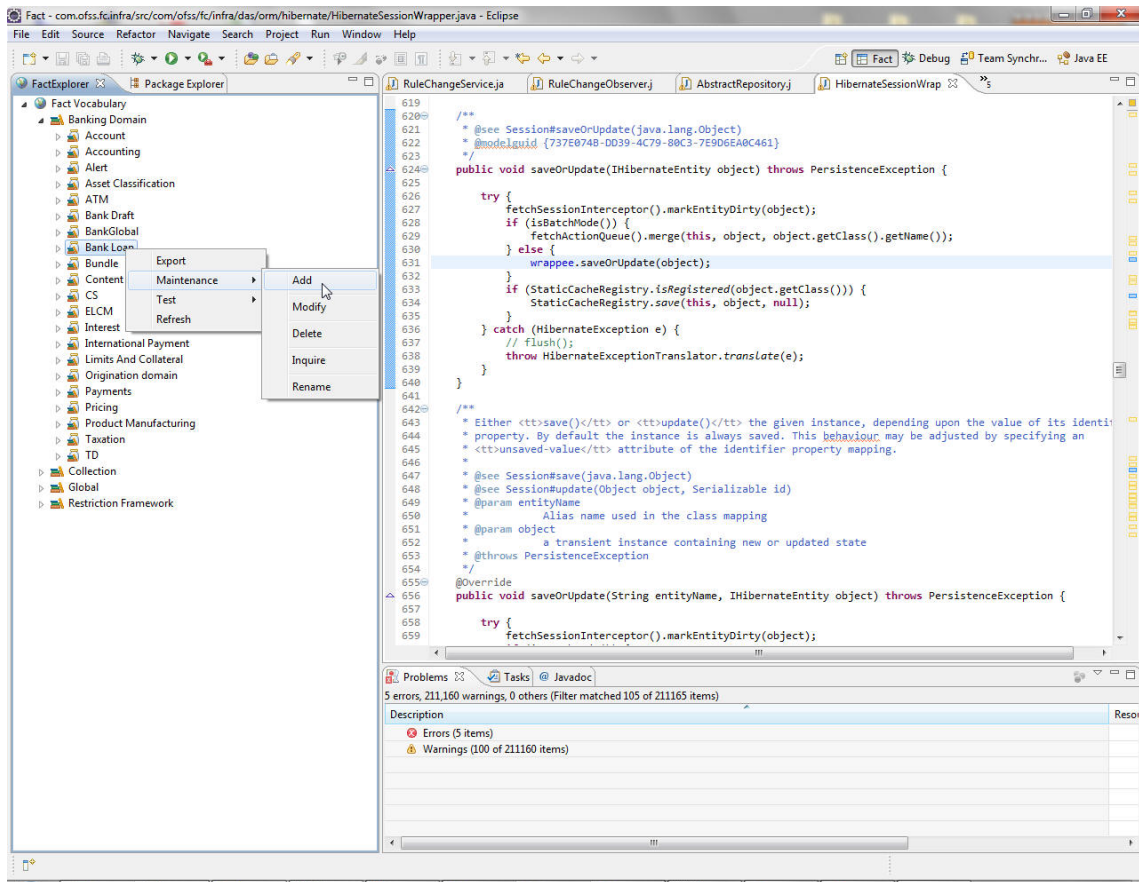


Figure 13–17 Aggregate File Tab



12. Creating **New Fact**: Right-click any domain Category in which Fact is to be created. Go to Maintenance -> Add.

Figure 13–18 Creating New Fact - Add



13. Enter required details for the facts in the new fact window.

All fields of Business definition tab are required for creation of any fact.

Fields of other tabs may be or may not be required. It depends on the fact to be created.

Figure 13–19 Creating New Fact - Fact Business Definition

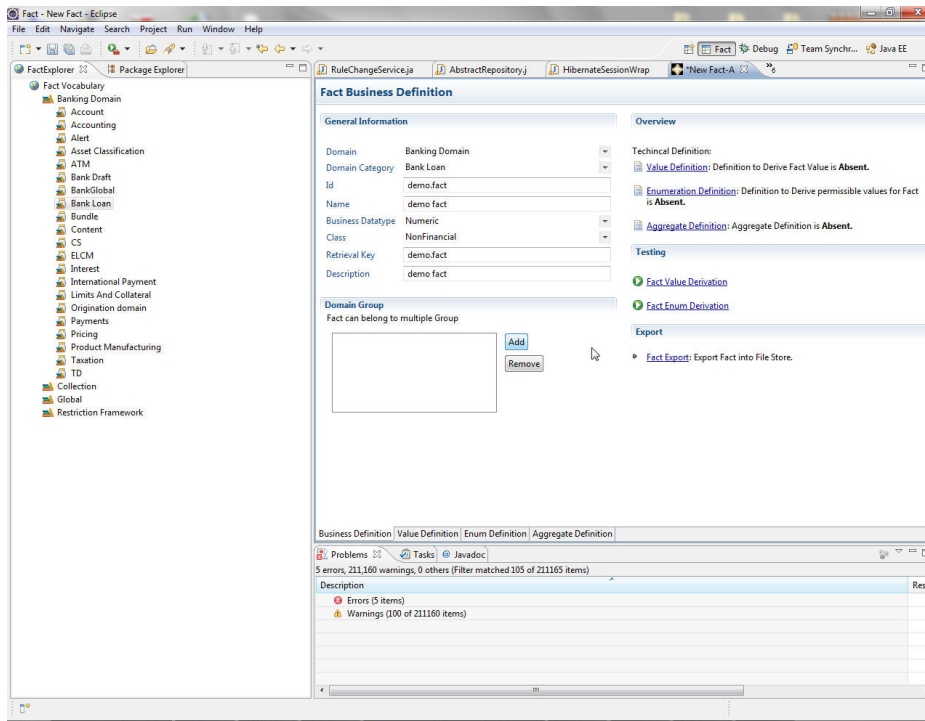
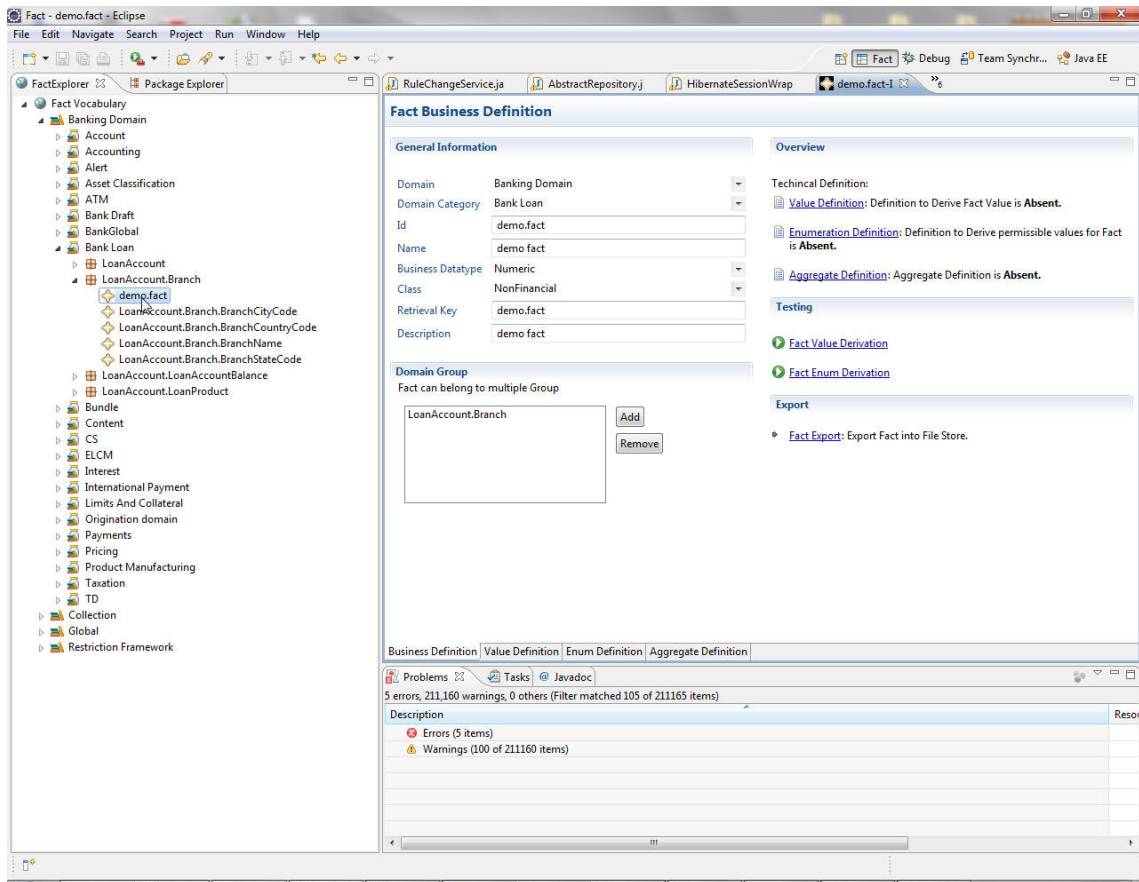


Figure 13–20 Creating New Fact - Domain Group



14. Enter the values in the fields and press CTRL+S, click **Yes** to save and fact will be created.

Figure 13–21 Saving New Fact

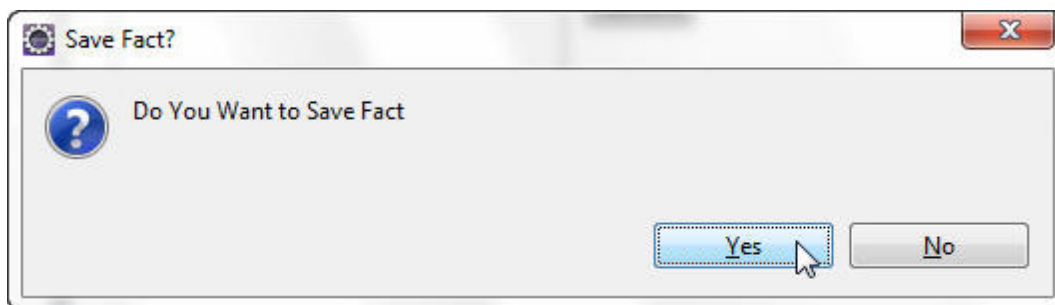
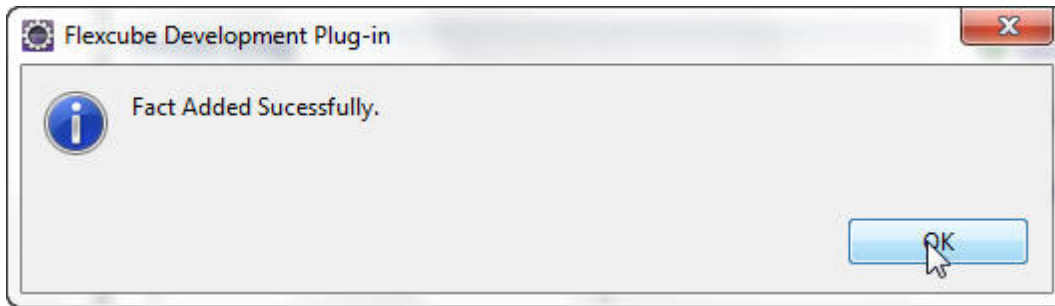


Figure 13–22 Saving New Fact - Fact Added



15. Modification of **Existing Fact**: To modify an existing fact, right-click the fact -> Maintenance -> Modify.

It opens the fact details in editable mode. Change whatever required and then save it using 'CTRL+R+S'.

Fact Perspective also provide following facilities:

- Maintenance Operations on Fact
 - Add
 - Modify
 - Inquire
 - Fact Derivation Test
 - Fact Value Derivation Test
 - Fact Enum Derivation Test
 - Fact Import - Import Fact from File Store to Database store
 - Fact Export - Export Fact from Database store to File store.

13.1.4 Object Facts

Apart from the normal facts that have to be maintained explicitly, there is a way to define an object as a fact. The idea behind having object fact is to ease the fact definition phase when a particular class holds maximum attributes that are likely to be used in a given rule along the execution path. The advantages are as follows:

- No need of having individual fact definitions for each of the attribute in the class.
- The entire class can be made an object fact and the fact derivation takes the responsibility of scanning through this class object for fact value.
- The caller module will have the object already loaded in most of the scenarios.
- Ease of passing the facts through fact context, no need to remember the fact IDs of all the facts to a granular level. Once the parent fact is passed in the fact context with the class name as the fact id, the attributes are automatically scanned for the respective values as required.

Designate a class as Object Fact

To make a class an object fact, an entry for it needs to be made in the table: "flx_fa_object_facts_b".

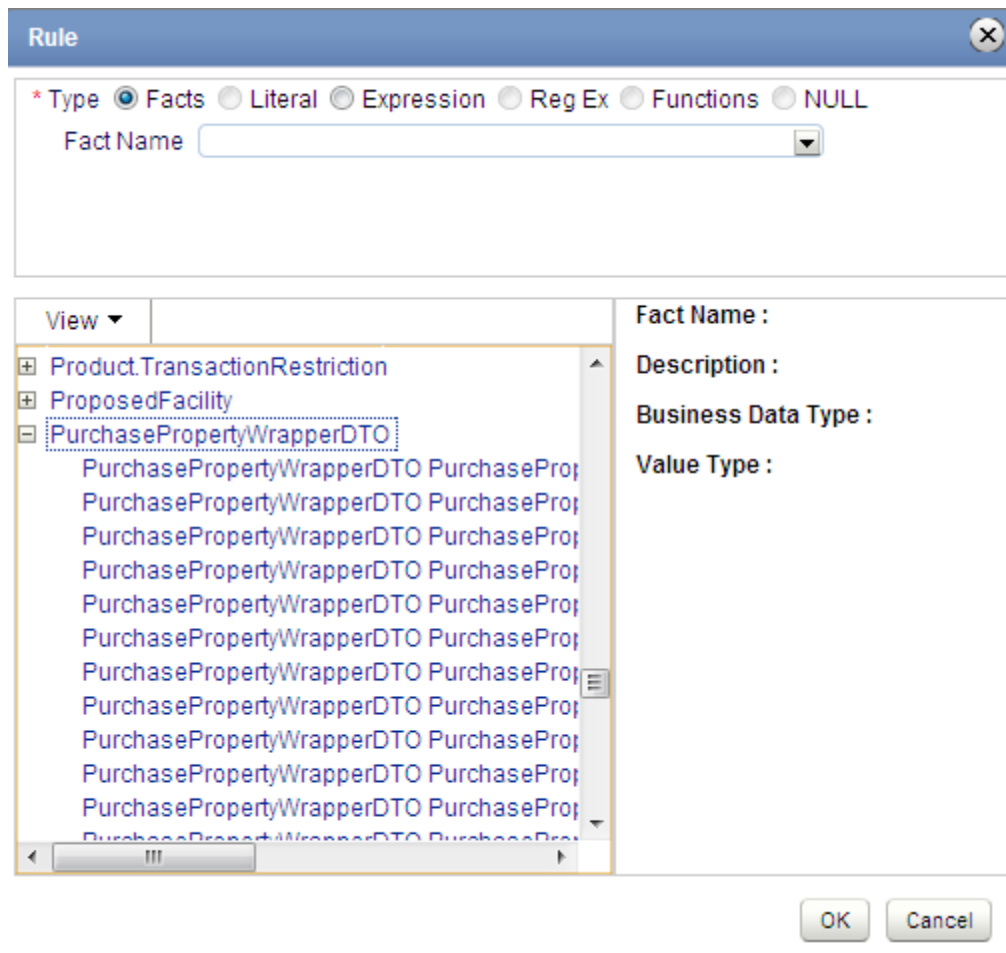
Figure 13–23 Designate Class as Object Fact

Row 1	Fields
FACT_GRP_NAME	PurchasePropertyWrapperDTO
FULL_QUAL_NAME	com.ofss.fc.app.Origination.dto.lending.core.pg.PurchasePropert ...
DOMAIN_CODE	Banking
DOMAIN_CATEGORY_CODE	OR
CREATED_BY	
CREATION_DATE	
LAST_UPDATED_BY	
LAST_UPDATE_DATE	
▶ OBJECT_VERSION_NUMBER	1
LAST_UPDATE_LOGIN	
DOMAIN_OBJECT_EXTN	CZ
FACT_NAME	PurchasePropertyWrapperDTO

Object Fact in UI

The usage of the object fact will be same as any other fact in the UI.

Figure 13–24 Object Fact in UI



Fact definitions for Object Fact

Building the fact definitions for an object fact is done as follows:

1. Once a class is designated as an object fact, it will be looked up at the time of loading the fact vocabulary.
2. The individual attribute access methods (getters or Boolean access methods that is, ones that start with "is") will be scanned to get the name of the attributes.
3. Once the attribute names and their data types are obtained, the FactBusinessDefinition object is created for it.
4. A variable fact object is also created and registered in the fact registry on the host.
5. The step 3 and 4 will be recursive, done for all the nested objects with the object fact till the leaf fact is found (that is, the one that can be used in the rule for instance data type could be any Java data types like String or Integer, or the OBP data types like Money or Duration)

13.2 Business Rules

Business Rules are defined for improving agility and for implementing business policy changes. This agility, meaning fast time to market, is realized by reducing the latency from approved business policy changes to production deployment to near zero time. In addition to agility improvements, Business Rules development also requires far fewer resources for implementing business policy changes. This means that Business Rules not only provides agility, it also provides the bonus of reduced development cost.

13.2.1 Rules Engine

A rule engine is a mechanism for executing 'business rules'. Business rules are simple business-oriented statements that encode business decisions of some kind, often phrased very simply in an if/then conditional form.

For instance, a business rule for a Banking system might be: Given a Customer and his location, if all of the following conditions are met:- The Customer is High Net worth Individual (HNI) - The Location is Metro - The Location is not Delhi{ }. The consequence is a 20% Discount in Application fee for Home loan. These business rules are not new: they are the business logic that is the core of many business software applications. These rules are expressed as a subset of requirements. They are statements like "give a twenty-percent discount to non-Delhi Metro HNI Customers"

The primary difference with a rule engine is the way these rules are expressed; instead of embedding them within the program, these are encoded in business rule form.

Rule engines are not limited to execution; they often come with other tools to manage rules. Enterprise Rule Engine has all the options such as creation, deployment, storage, versioning and other such administration of rules either individually, or in groups.

13.2.2 Rules Creation by Guided Rule Editor

Any kind of rule can be created using this tool. User can freely enter business rules in text area, throughout the rule creation tool.

Standard Rule created in GRE comprises of following elements:

```
[mandatory]
If
[condition] {AND/OR [condition]}*
Then
[Action]+
[optional]*
Else If
[condition] {AND/OR [condition]}*
Then
[Action]+
[optional]?
Else
[Action]+
where
* = 0 or more Occurrence
?= 0 or 1 Occurrence
+= 1 or more Occurrence
```

Features of Guided Rule Editor (GRE)

The features of GRE are:

- The 'if' block is mandatory block at the beginning of the structure.
- If (true) kind of condition is not supported. The condition should be comprised of 'LHS operator RKH'. There is parenthesis support in the UI. But you have to add it manually. Validation of parenthesis is supported.
- Nested 'if' is not supported from UI as of now.
- Conditions and actions are added by clicking the '+' button.
- After adding Condition user can add 'AND/OR Condition' by clicking '+' button at the End of Condition
- Different types of Actions can be added under 'Then'.
- Any number of 'Else if' can be added after 'If'.
- The condition for 'Else if' should differ from its previous 'if' or 'Else if' condition. Warning should be shown to user in this case.
- At most one 'Else' condition can be added to this 'if-else if-else' structure.
- No 'Else if' can be added after 'Else'.
- Real time rule structure preview in the bottom panel.
- Rule template / fragment for re usability.
- Facts will be used to create the rules

13.2.3 Rules Creation By Decision Table

Decision tables are a precise yet compact way to model complicated logic. Decision tables, like if-then-else, associate conditions with actions to perform. But, unlike the control structures found in traditional programming languages, decision tables can associate many independent conditions with several actions in an elegant way.

Example:

Table 13–1 Example of a Decision Table

Conditions & its alternatives			Actions
Customer Type	Location Type	Location	Discount
HNI	Metro	Mumbai	20% of App. fee
HNI	Metro	Delhi	No discount
HNI		Jaipur	No discount

The features of Decision Table are:

- The decision table contains rows and columns. Each row is considered to be a rule. In normal circumstances, the decision table is evaluated from top to bottom sequentially evaluating the various rules. It does not stop even if a rule fires. However, there is an option to stop processing of the decision

table in case a rule is satisfied. There should be a special fixed column in the decision table (towards the right) which allows the decision table author to stop further evaluation of rules in case the current rule fires.

- Decision table should be expandable, that is, Rows and columns can be added dynamically.

Various functions for column and row manipulation should be available:

- Add Column After
- Add Column Before
- Add Row Above
- Add Row Below
- Delete Column
- Delete Row
- Move Column
- Move Row
- Sort Column Data Ascending
- Sort Column Data Descending
- Column Headers indicate condition / action
- Decision table should be editable to input data/conditions/actions

If a condition or action has range the column should be split in to two columns to accept the minimum and maximum values. Option to automatically fill series of values. When clicked on row, a brief description about the condition should appear. Decision table will have brief description for the conditions and actions setup. Import and export data between Decision Table and Excel Spread Sheet.

13.2.4 Rules Storage

Rules created are stored in database tables as conditions and actions first, then these database tables are used to create executable rule in java programming language and compiled.

Table 13–2 Actions

ActionID	Outvariable	Expression	Datatype
ACTION1	Discount Fee	0.2*App Fee	Double
ACTION2	Discount Fee	0	Double
ACTION3	Discount Fee	0	Double

Table 13–3 Conditions

ConditionID	LeftExpression	Relational Operator	RightExpression	LinkedConditionID	LinkedConditionalOperator	ActionID	RuleID	Version
CON1	CustomerType	==	HNI	CON2	&&	ACTION1	RULE1	1

ConditionID	LeftExpression	Relational Operator	RightExpression	LinkedConditionID	LinkedConditionalOperator	ActionID	RuleID	Version
CON2	Location Type	==	METRO	CON3	&&		RULE1	1
CON3	Location	==	MUMBAI				RULE1	1
CON4	CustomerType	==	HNI	CON5	&&	ACTION2	RULE1	1
CON5	Location Type	==	METRO	CON6	&&		RULE1	1
CON6	Location	==	DELHI				RULE1	1
CON7	CustomerType	==	HNI	CON8	&&	ACTION3	RULE1	1
CON8	Location	==	JAIPUR				RULE1	1

13.2.5 Rules Deployment

Rules are put together in compiled java class which are stored in jar file and deployed on the server at runtime. This deployed jar is available for applications which are going to execute the rules.

13.2.6 Rules Versioning

Each time rule is modified new version is created for the rule and stored.

Table 13–4 Rules Versioning

RuleID	Version	Name	Effective Date
RULE1	1	DiscountRule	01/01/2009
RULE1	2	DiscountRule	31/03/2009

13.3 Rules Configuration in Modules

Rules can be configured for multiple modules and multiple screens. The list of screens where the rule definition taskflows are used is mentioned below:

- Facts are used by configuring the fact context. Fact Context contains information about interacting Module. This need to be set to interact with Fact layer. Fact Context has been categorized at Domain Level.

For example, BankingFactContext will be used in Banking domain. This context has setters method for Facts which are generic in that domain. For example, BankingFactContext has *setAccountId* method. Interacting module need to fill maximum information available. These methods are setters for Facts which will always has input like *AccountId*, *PartyId*, *TransactionAmount* and so on.

- It is possible that at the time of interaction, Module already has some derivable Facts which are not going to change in the interaction. For example, *LnAccountProduct* at the time of Interest calculation.

- Module will send such Facts using *addFact* method, using *_retrievalKey* of the Fact referring Fact vocabulary. The benefit of sending such facts is these Facts won't get derived again. At the time of Fact Derivation, if *RetrievalKey* is present in the input FactMap, same value will be returned as a Fact value. If *RetrievalValue* is not present the Fact will be derived.
- Module will send maximum Fact information available at the time of interaction for better performance.

For example, at the time of Loan Account Opening, Pseudo code will look like:

```
// create fact context.
BankingFactContext lnFactContext = new BankingFactContext
("LN");
lnFactContext.setPartyId(001);
// Set max available information
lnFactContext.addFact("LnAppliedAmount",2000);
lnFactContext.addFact("LnProductType","Home");
lnFactContext.addFact("LnRiskCategory",1);
lnFactContext.addFact("CustType","VIP");
```

At the time of CashTransaction Event, code will look like:

```
// create fact context.
BankingFactContext casaFactContext = new BankingFactContext
("CASA");
casaFactContext.setPartyId(003);
casaFactContext.setAccountId("111111111111");
casaFactContext.setTransactionAmount(new BigDecimal(122));
casaFactContext.setTransactionCurrency(104);
casaFactContext.setTransactionAmountInAcy(new BigDecimal
(122));
// Set max available information
casaFactContext.addFact("CustType", "VIP");
casaFactContext.addFact("CASAAccountType", "Saving");
```

13.3.1 Generic Rules Configuration

Generic Rules can be configured through the screen RL001 where the new rule can be defined or the existing rule can be updated for multiple domains and domain category. The authoring mode of rule creation can be chosen as GRE or Decision Table.

Figure 13–25 Generic Rule Configuration

The screenshot displays the 'Rule Author' interface for rule RL001. At the top, a navigation menu includes 'Account', 'Back Office', 'CASA', 'Channel', 'Collection', 'LCM', 'Loan', 'Operational Services', 'Origination', 'Party', 'Payment And Collection', and 'Fast Path'. The main area is titled 'Rule Details' and contains the following information:

- Domain Id:** Banking
- Domain Category Id:** CS
- Rule Id:** OD_RL_1
- Effective Date:** 02-Jan-2013
- Authoring Mode:** GRE
- Domain Name:** Banking Domain
- Domain Category Name:** CS
- Name:** OD Rule
- Description:** OD Rule Desc
- Version:** 1

The rule logic is organized into three sections:

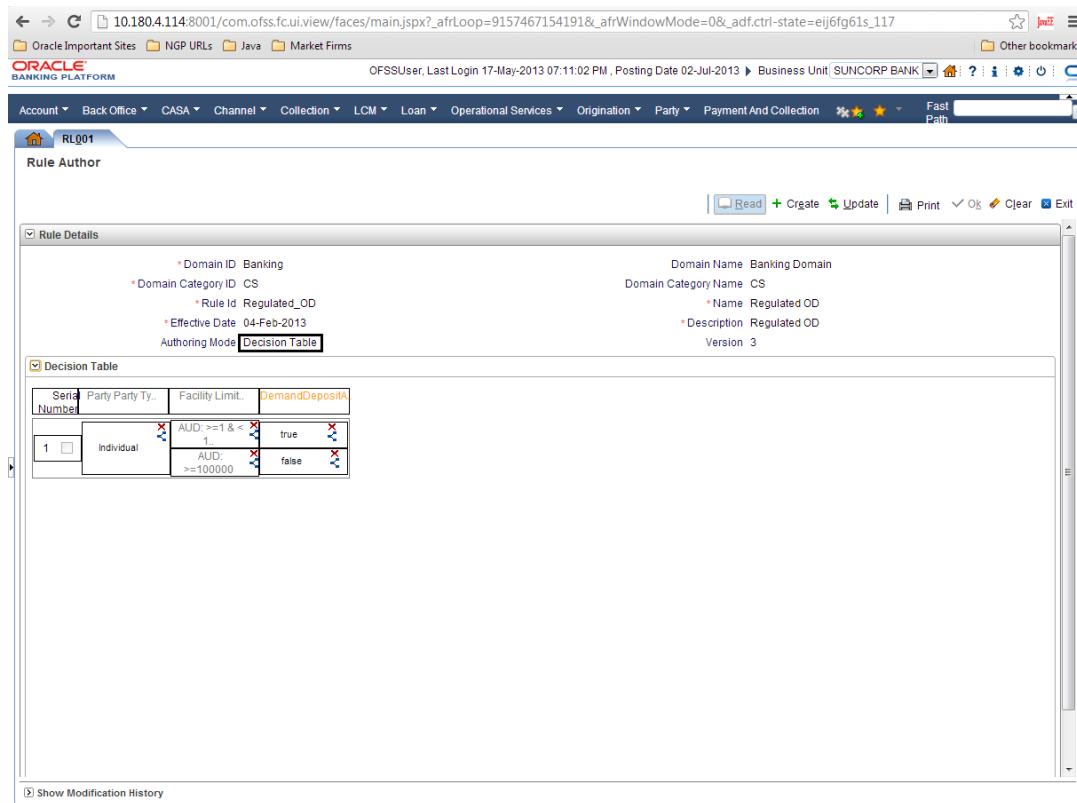
- IF:** A complex logical expression involving 'AssetClassification' fields (Fees, Interest, TOD, Overline, Suspended Fees, Suspended Interest) with conditions like 'greater than 0' and 'less than equal to 8'. The 'Then' clause sets 'Classification Code equal to 101' and 'AC.ClassificationReason equal to D'.
- Else If:** A second logical expression with conditions like 'greater than 16' and 'less than equal to 24'. The 'Then' clause sets 'Classification Code equal to 103' and 'AC.ClassificationReason equal to D'.
- Else If:** A third logical expression with conditions like 'greater than 24'. The 'Then' clause sets 'Classification Code equal to 104' and 'AC.ClassificationReason equal to D'.

At the bottom, the 'Hide Modification History' section shows:

- Created By: OFSSUser, On: 22-Feb-2013 12:17:46 PM, Approved:
- Approved By: OFSSUser, On: 22-Feb-2013 12:17:46 PM, Active:

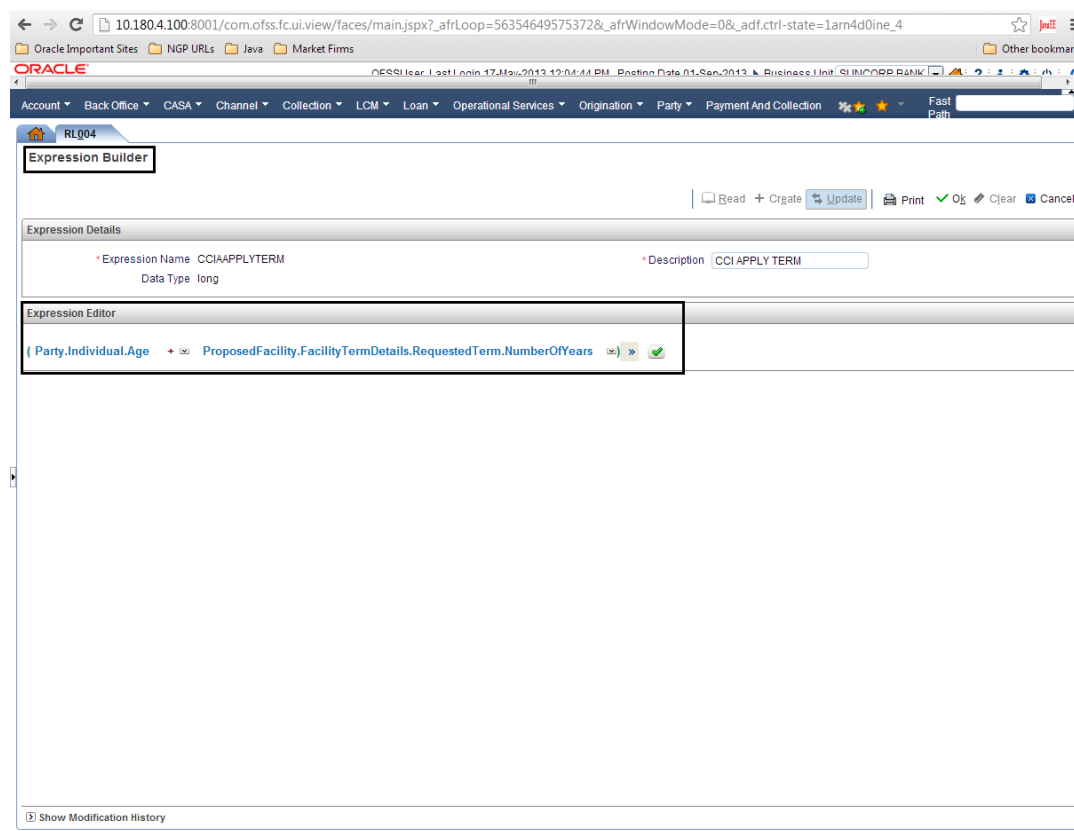
Navigation controls at the bottom right include '<< 1 of 1 >>'.

Figure 13–26 Rule Author - Decision Table



Different expressions can be defined in the expression builder screen. The expression once defined can also be used as one of the expressions in GRE.

Figure 13–27 Rule Author - Expression Builder



13.4 Rules Migration

This section describes the rules migration.

13.4.1 Rules Configured for Modules

Rule taskflows can be added to different modules. User can set up different rules based on the screen requirements.

Table 13–5 Details of Configured Rules in Modules

Module	Screen	Rule Type	Rule Description
Alerts	AL04 - Alert Maintenance	GRE	User can create the new message template rule or use the existing rule. In this rule, the message template of the alert is selected based on the selected rule criteria. For example, if there is a particular party id, then the specific alert needs to be sent.
Content	CNM03 - Document Policy Definition	Decision Table	There are two types of rules (Inbound Rule and Outbound Rule) defined for each event in the document policies. These rules primarily define the checklist of documents based on different input values. The inbound rule are defined for the scenario of the

Module	Screen	Rule Type	Rule Description
			documents being inputted to the system and the outbound rule are defined for the scenario of the documents being retrieved from the system and displayed to the end user. For example, In document policy of new applications, there is a event for identity verification. The inbound rule can be defined for the category of the documents which are required to be uploaded for the verification purpose on the basis of the Party Agency Type and the Party Type.
Pricing	PR006 - Price Definition	Generic Rule Author	Price can be rule based that is, amount of fee to be charged or price code to be charged comes from rule
Pricing	PR005 - Interest/Margin Index Code Definition	Generic Rule Author	Interest Index can be Rule Based i.e. Interest rate to be applied comes as outcome of rule.
Pricing	PR004 - Rate Chart Maintenance	Generic Rule Author	Rate Chart can be Rule Based i.e. Interest index to be used comes as outcome of rule.
Pricing	PR007 - Price Policy Chart Maintenance	Decision Table	Price policy chart internally gets stored as Rule. It basically defines Prices/RateCharts applicable when criteria is satisfied which is mentioned in rule.
Pricing	PR040 - Fee Computation Analysis	Generic Rule Author	This screen provides analysis as how the fee for particular transaction (happened in past) was computed. In case of Rule Based Fees charged in transaction, this screen displays details of that rule along with input fact values used during rule evaluation.
Pricing	PR017 - Interest Rate Derivation Analysis	Generic Rule Author	This screen provides analysis as how the interest rate for particular account was computed. In case of Rule Based Rate Chart and Rule Based Index, this screen displays details of that rule along with input fact values used during rule evaluation.
Tax	TDS01 - Tax Parameter Maintenance	Decision Table	This rule is used to maintain the exemption limit and that exemption limit will be used at the time of tax computation.
Product Manufacturing	PM011 - Define Interest Rule	GRE/ Decision Table	In the Rule and Expression task flow is consumed to create Rule or Expression, which is used to derive the BaseForInterest for Calculation of Interest. During EOD, module send facts which is used derive the BaseForInterest by executing the Rule or Expression whichever is attached to the IRD.
Asset Classification	RL001 - Rule Author	GRE	This rule is used to derive the Asset Classification code of an account during the Account level classification batch shell. The facts will be the days past due date of various outstanding arrears. The rules will be created under 'LN' and 'CS' and linked to a plan in Asset Classification Plans (NP002).

Module	Screen	Rule Type	Rule Description
			Rule for Facility-level classification: This rule is maintained only if the 'Applicability level' in NP001 is 'Facility'. This rule is used to derive the Classification code for a Facility during the Facility-level batch classification. The rule will be created under the Domain Category 'AC' and is linked via Asset Classification Preference (NP001).
Collections	RULE01 - RuleSet	GRE/Decision Table	<p>Collection module's rules are defined as RuleSet. The RuleSet can be incorporated for the batch processing to filter accounts coming to collection.</p> <p>In RuleSet screen, multiple rules can be combined together as a single object called ruleset. The RuleSet functionality in rule engine provides the user with the facility to design the sequence of execution of rules where multiple rules need to be asserted for the same set of inputs. User would be able to select and wire the already existing rules and their sequence as per his/her requirement.</p> <p>There can be output dependent rules defined. For example,</p> <p>Rule 1 is: If(FACILITY_ID equal to TEST_FACILITY_ID) Then Account Type equal to FIXED Else If (FACILITY_ID equal to AAA) Then Account Type equal to 0</p> <p>Rule 2 is: If (ACCOUNT_TYPE equal to FIXED) Then ARS_ASSESSED_AMOUNT equal to 70000</p> <p>In the above case, rule 2 will be executed only if rule 1 satisfies the condition.</p>

14 Composite Application Service

OBP Application provides with the functionality of adding composite application services which call multiple application services in one request. The transactions in these composite application services are called composite transactions and are made by composing the single transaction out of the multiple APIs transaction that gives the effect of single transaction.

Using APIs, single transaction can be composed of multiple transactions using very little effort. However, this cannot be done at run time. Following points have to be taken in to account while making a new composite transaction out of existing API transactions:

- Both the transactions should be passed in the same session context except overridden warnings. Overridden warnings from one transaction are passed as an input to next transaction.
- Decision of whether to commit the transaction or rollback the same must be explicitly handled by the composite transaction. The beginning and closing of interaction should be handled by the composite transactions.

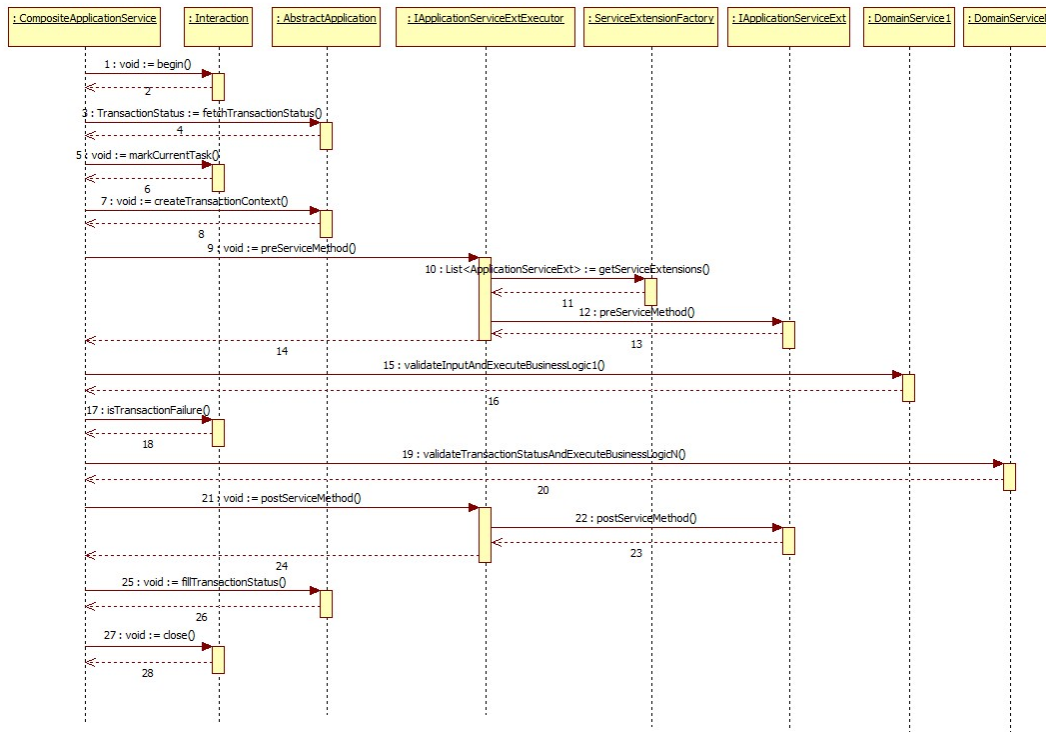
For the transaction control of the transaction manager, there are two defined patterns:

- **With Interaction.begin**
 - The interaction begins to ensure that the transaction reference number is maintained same across all participating APIs
 - Required for supporting reversal of composite financial APIs
 - Context information for entire call is maintained and used.
 - Similar to any other API
- **With TransactionManager**
 - Scope restricted to database transaction
 - All APIs in the composite have the same commit scope
 - Unique transaction reference generated for each API
 - Can be thought of as a workflow with APIs participating in the same DB commit scope
 - The composite transactions can be handled in two scenarios:
 - Calling multiple APIs in the same module
 - Calling multiple APIs in different modules by making the adapter call

14.1 Composite Application Service Architecture

The following depicts the sequence diagram for the composite transactions where two of the domain service calls are shown which can be extended to multiple domain service (1..N) calls. After every domain service call, 'isTransactionFailure()' call needs to be made to check the transaction status before proceeding for the next domain service call.

Figure 14–1 Composite Application Service Architecture



14.2 Multiple APIs in Single Module

For writing the composite service API which calls multiple services API, the following Java classes are needed with respect to new services as mentioned in the below table:

Table 14–1 Java Classes

Class Name	Description
Composite Service Interface	This provides the method definitions for the composite services.
Composite Service Class	This provides the implementation class for the composite services. In this class, we write methods which make the calls to different service APIs. The response of one service API can be used for making calls in another service APIs. The final response of the composite service is then created with the response objects of other service APIs and then transferred back to the adapter calls.
Executor Interface	This provides the extension pre-hook and post-hook method definitions for the service calls.
Executor Classes	This provides the implementation class for the executor interface.
Composite API Response Object	This provides the final response object which is passed to the adapter calls.

One of the sample composite service method 'TDAccountPayinApplicationService.openAccountWithPayin' is shown below. In this service method, there are two methods of two different services:

- tdAccountApplicationService.openAccount
- tdDepositApplicationService.openDeposit

These service methods are called where the new account is created and then the returned account id from first service is used to do the payin by creating a new deposit for that account.

```

package com.ofss.fc.app.extensibility.td.service.composite;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.ofss.fc.app.AbstractApplication;
import com.ofss.fc.app.Interaction;
import com.ofss.fc.app.agent.dto.agent.AgentArrangementLinkageDTO;
import com.ofss.fc.app.context.SessionContext;
import
com.ofss.fc.app.extensibility.td.dto.composite.TDAccountPayinResponse;
import
com.ofss.fc.app.extensibility.td.service.composite.ext.IExtendedTermDepositApplicationServiceExtExecutor;
import com.ofss.fc.app.td.dto.account.TermDepositAccountOpenDTO;
import com.ofss.fc.app.td.dto.account.TermDepositAccountResponse;
import com.ofss.fc.app.td.dto.deposit.PayinResponse;
import
com.ofss.fc.app.td.dto.transaction.payin.PayinTransactionDTO;

import
com.ofss.fc.app.td.service.account.ITermDepositAccountApplicationService;
import
com.ofss.fc.app.td.service.account.TermDepositAccountApplicationService;
import
com.ofss.fc.app.td.service.deposit.DepositApplicationService;
import
com.ofss.fc.app.td.service.deposit.IDepositApplicationService;
import com.ofss.fc.common.td.TermDepositTaskConstants;
import com.ofss.fc.enumeration.MaintenanceType;
import com.ofss.fc.infra.exception.FatalException;
import com.ofss.fc.infra.exception.RuntimeException;
import com.ofss.fc.infra.log.impl.MultiEntityLogger;
import com.ofss.fc.service.response.TransactionStatus;
/**
 * The TDAccountPayinApplicationService class exposes
 functions/services to perform the sample of composite operations.
 This extensibility sample services includes: opening account and
 deposit
 * @author Ofss

```



```
*/
public class ExtendedTermDepositApplicationService extends
AbstractApplication implements
IExtendedTermDepositApplicationService {
/**
* Extension point for the class. This is the factory implementation
for the extension of this class.
* Any extension-method call on this factory instance, internally
triggers a call to corresponding
* extension methods of all the extension classes returned by the
ServiceExtensionFactory
*/
private transient IExtendedTermDepositApplicationServiceExtExecutor
extension;
// This attribute holds the component name
private final String THIS_COMPONENT_NAME =
ExtendedTermDepositApplicationService.class.getName();
/**
* This is an instance variable and not a class variable (static or
static final). This is required to
* support multi-entity wide logging.
*/
private transient Logger logger =
MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_
NAME);
/ Create instance of multi entity logger
private transient MultiEntityLogger formatter =
MultiEntityLogger.getUniqueInstance();
/**
* @param sessionContext
* @param termDepositAccountOpenDTO
* @return TermDepositAccountResponse
* @throws FatalException
*/
public TDAccountPayinResponse openAccountWithPayin(SessionContext
sessionContext,
TermDepositAccountOpenDTO termDepositAccountOpenDTO,
PayinTransactionDTO payinTransactionDTO,
AgentArrangementLinkageDTO agentArrangementLinkageDTO
) throws FatalException {
super.checkAccess
("com.ofss.fc.app.td.service.composite.TDAccountPayinApplicationSe
rvice.openAccountWithPayin", sessionContext,
termDepositAccountOpenDTO, payinTransactionDTO,
agentArrangementLinkageDTO);
if (logger.isLoggable(Level.FINE)) {
```

```

logger.log(Level.FINE, formatter.formatMessage("Entered into
openAccountWithPayin(). Input : termDepositAccountOpenDTO %s
",THIS_COMPONENT_NAME, termDepositAccountOpenDTO.toString()));
}
Interaction.begin(sessionContext);
TransactionStatus transactionStatus = fetchTransactionStatus();
TermDepositAccountResponse tdAccountResponse = null;
String newAccountId = null;
PayinResponse payinResponse = null;
TDAccountPayinResponse tdAccountPayinResponse = new
TDAccountPayinResponse();
ITermDepositAccountApplicationService tdAccountApplicationService
= new TermDepositAccountApplicationService();
IDepositApplicationService tdDepositApplicationService= new
DepositApplicationService();
try {
Interaction.markCurrentTask(TermDepositTaskConstants.TD_ACCOUNT_
ATTRIBUTE);
createTransactionContext(sessionContext, MaintenanceType.ADDITION);
extension.preOpenAccountWithPayin(sessionContext,
termDepositAccountOpenDTO,
payinTransactionDTO, agentArrangementLinkageDTO);
termDepositAccountOpenDTO.setBankCode(sessionContext.getBankCode
());
if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, formatter.formatMessage("Entered into
tdAccountApplicationService.openAccount().
Input : termDepositAccountOpenDTO %s ",THIS_COMPONENT_NAME,
termDepositAccountOpenDTO.toString()));
}
tdAccountResponse = tdAccountApplicationService.openAccount
(sessionContext, termDepositAccountOpenDTO);
if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, formatter.formatMessage("Exiting from
tdAccountApplicationService.openAccount().
Input : termDepositAccountOpenDTO %s ", THIS_COMPONENT_NAME,
termDepositAccountOpenDTO.toString()));
}
if(tdAccountResponse!=null && tdAccountResponse.getAccountId
()!=null &&
!Interaction.isTransactionFailure(transactionStatus)) {
newAccountId = tdAccountResponse.getAccountId();
payinTransactionDTO.getAccountTransactionDTO().setAccountId
(newAccountId);
if (logger.isLoggable(Level.FINE)) {
Logger.log(Level.FINE, formatter.formatMessage("Entered into
tdDepositApplicationService.openDeposit().

```

```
Input : payinTransactionDTO %s ", THIS_COMPONENT_NAME,
termDepositAccountOpenDTO.toString());
}
payinResponse = tdDepositApplicationService.openDeposit
(sessionContext, payinTransactionDTO, agentArrangementLinkageDTO);
if (logger.isLoggable(Level.FINE)) {
logger.log(Level.FINE, formatter.formatMessage("Exiting from
tdDepositApplicationService.openDeposit().
Input : payinTransactionDTO %s ", THIS_COMPONENT_NAME,
termDepositAccountOpenDTO.toString()));
}
if (payinResponse != null) {
tdAccountPayinResponse.setAccountId(payinResponse.getAccountId());
tdAccountPayinResponse.setDepositId(payinResponse.getDepositId());
tdAccountPayinResponse.setDepositStatus
(payinResponse.getDepositStatus());
tdAccountPayinResponse.setNetInterestRate
(payinResponse.getNetInterestRate());
tdAccountPayinResponse.setAccountingEventItem
(payinResponse.getAccountingEventItem());
tdAccountPayinResponse.setMaintenanceType
(payinResponse.getMaintenanceType());
tdAccountPayinResponse.setMaturityAmount
(payinResponse.getMaturityAmount());
tdAccountPayinResponse.setProductCode(payinResponse.getProductCode
());
tdAccountPayinResponse.setInterestStartDate
(payinResponse.getInterestStartDate());
tdAccountPayinResponse.setValueDate(payinResponse.getValueDate());
tdAccountPayinResponse.setStatus(payinResponse.getStatus());
}
}
extension.postOpenAccountWithPayin(sessionContext,
termDepositAccountOpenDTO, payinTransactionDTO,
agentArrangementLinkageDTO);
fillTransactionStatus(transactionStatus);
tdAccountPayinResponse.setStatus(transactionStatus);
} catch (FatalException fatalException) {
logger.log(Level.SEVERE, formatter.formatMessage("FatalException
from openAccountWithPayin()"), fatalException);
fillTransactionStatus(transactionStatus, fatalException);
} catch (RuntimeException fcrException) {
logger.log(Level.SEVERE, "RunTimeException from
openAccountWithPayin()", fcrException);
fillTransactionStatus(transactionStatus, fcrException);
} catch (Throwable throwable) {
logger.log(Level.SEVERE, "Throwable from openAccountWithPayin()",
throwable);
fillTransactionStatus(transactionStatus, throwable);
```

```
    } finally {  
        Interaction.close();  
    }  
    super.checkResponse(sessionContext, payinResponse);  
    if (logger.isLoggable(Level.FINE)) {  
        logger.log(Level.FINE, formatter.formatMessage("Exiting from  
openAccountWithPayin()."));  
    }  
    return tdAccountPayinResponse;  
    }  
}
```


15 ID Generation

OBP is shipped with the functionality of generation of the IDs in three ways that is, Automatic, Manual and Custom. These three configurations can be defined by the user as per their requirements:

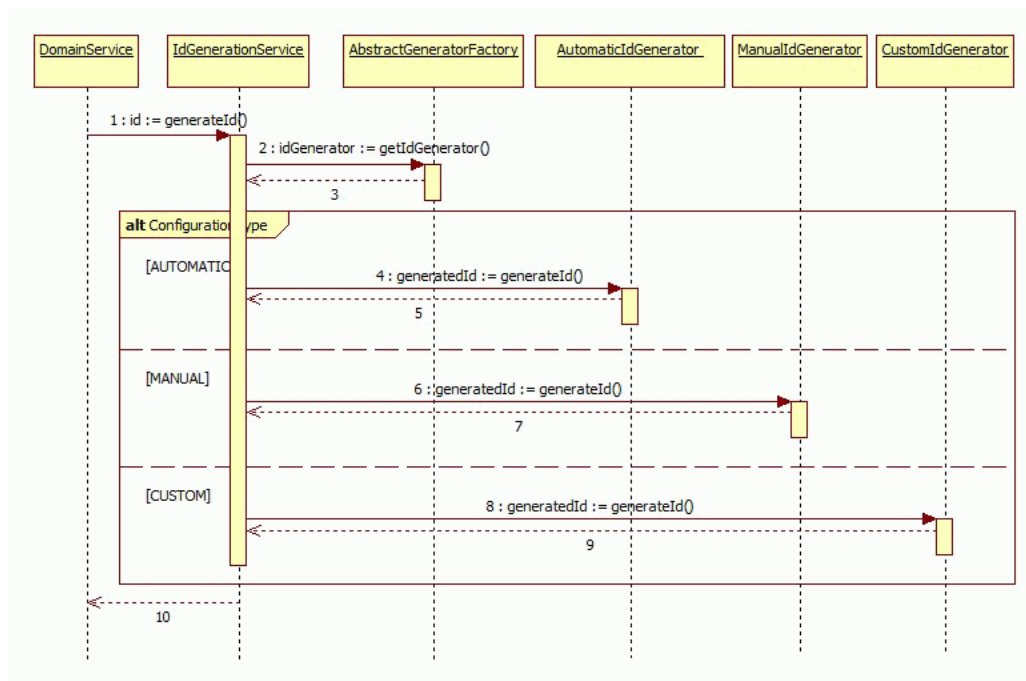
If the configuration type for the ID generation is set to automatic, the ID is generated as per the defined generation logic for the automated ID generation. You can set the pattern, sequence, weights and check digit modulo and modify the automatic generation logic.

If the configuration type is set to manual then the ID will be input and it will be checked in the database if it is unique. For the ID, a certain range of serial numbers can be reserved in the range table by the custom developer and the teller can select it from amongst the ranges while doing the manual entry.

In case the bank's requirement is to have the different ID generation process which can be written or modified, then the extensibility feature is provided in OBP. In this feature, customized ID generation logic can be written and can be plugged in the OBP application by creating the custom ID generation class and doing the required configurations in the database.

The configuration of the ID generation process is shown in the sequence diagram below where the generator is selected based on the set configuration type.

Figure 15–1 Configuration of ID Generation Process



From the implementation perspective, the following sections describe the change in configurations required for customizing the ID generation.

15.1 Database Setup

The configuration part of the ID generation requires the following components which need to be defined in the OBP application. The following tables are involved to store the generation logic details for ID generation:

- **FLX_CS_ID_CONFIG_B**: This is the main config table where the identifier is defined with the combination of the category and sub category columns. The type of generation logic is determined based on the configuration set in the CONFIG_TYPE column of this table.

Table 15–1 FLX_CS_ID_CONFIG_B

Column Name	Description
CATEGORY_ID	Represents the Category Example: Party, Origination, DDA and so on
SUB_CATEGORY_ID	Represents the Sub Category Example: PartyId, AccountNo and so on
PATTERN_TXT	Represents the pattern in which the ID is generated Example: SSSSSSSSC, NNNBBBBYYYYSSSSSS
CONFIG_TYP	Represents Generation type values are AUT for Automatic, MAN for Manual, CUS for Custom
GENERATOR_CLASS_NAME	Fully Qualified classname of ID generator for config type Custom
SEQ_VALUE	Running Serial Number
WEIGHT	Comma separated Weight for each character defined in the pattern text Example: '0,0,7,6,5,4,3,2', '3,8,1'
CHK_DIGIT_MODULO	Check digit modulo
CREATED_BY	Indicates the User who created the row
CREATION_DATE	Indicates the date and time of the creation of the row
LAST_UPDATED_BY	Indicates the User who last updated the row
LAST_UPDATE_DATE	Indicates the date and time of the last update of the row
OBJECT_VERSION_NUMBER	Indicates the version number, Used to implement optimistic locking
OBJECT_STATUS_FLAG	Status Flag Example: A

- **FLX_CS_ID_RANGE**: This table is used to determine the range of the values which the ID can take.

Table 15–2 FLX_CS_ID_RANGE

Column Name	Description
RANGE_ID	Represents the identifier for the range definition
RANGE_NAME	Represents the name defined for the range Example: Party, DDA
RANGE_START	Defines the beginning value for the range
RANGE_CURRENT	Defines the current value for the range
RANGE_END	Defines the ending value for the range
CATEGORY_ID	Represents the Category defined in FLX_CS_ID_CONFIG_B
SUB_CATEGORY_ID	Represents the Sub Category defined in FLX_CS_ID_CONFIG

- **FLX_CS_ID_USF**: This table is used to determine the user selected fields for the ID generation logic.

Table 15–3 FLX_CS_ID_USF

Column Name	Description
USF_ID	Represents the identifier for the user selected fields
USF_NAME	Represents the name for the user selected fields
IS_FIXED_FLAG	Defines if the user selected fields are fixed
CATEGORY_ID	Represents the Category defined in FLX_CS_ID_CONFIG_B
SUB_CATEGORY_ID	Represents the Sub Category defined in FLX_CS_ID_CONFIG_B

15.1.1 Database Configuration

In case of existing ID generation logic in the database, end user can update the seed data scripts by modifying configuration type and other parameters (pattern, sequence, weight and check digit modulo). While in case of new type of ID generation logic, an insert sql can be added in the scripts of tables.

15.2 Automated ID Generation

For the configuration type as automatic, user needs to set the CONFIG_TYPE as "AUT" in the FLX_CS_ID_CONFIG_B table. The ID generation logic is determined based on the set values in the config table for the pattern, sequence, weight and check digit modulo. The three attributes 'sequence', 'weights' and 'check digit modulo' are primarily used for calculation of the check digit.

ID Generation with Sequence and Range

ID is picked using the database sequence. This is needed in the case where serial number is used as part of an ID. Database sequence is used to avoid deadlock while trying to update, a sequential value stored and retrieved as part of the configuration in-case where the application is multiple threaded. This might lead to 'gaps' in the sequence of ids generated, if an exception occurs in the Transaction. However, this suffices as the errors related to deadlocks are mitigated.

For the first call to derive the value, the sequence for the specific configuration pattern is created, with names as CATEGORYTYPE_SUBCATEGORYTYPE_SEQ. The creation of this sequence happens only once in the lifecycle of application deployment. For example, TD (category) and AccountId (sub-category), the sequence generated is TD_ACCOUNTID_SEQ. And, for the successive requests, the already created sequence is used for sequence generation.

ID Generation with Pattern Text

The pattern text is split and an array is created of the characters. In case of mask ID configuration's pattern, ID configuration's text patterns are split. If the value is found to contain the special character (out of range [65-90]), it will be appended as it is to generated ID. Following are the conditions of ID generation with pattern text:

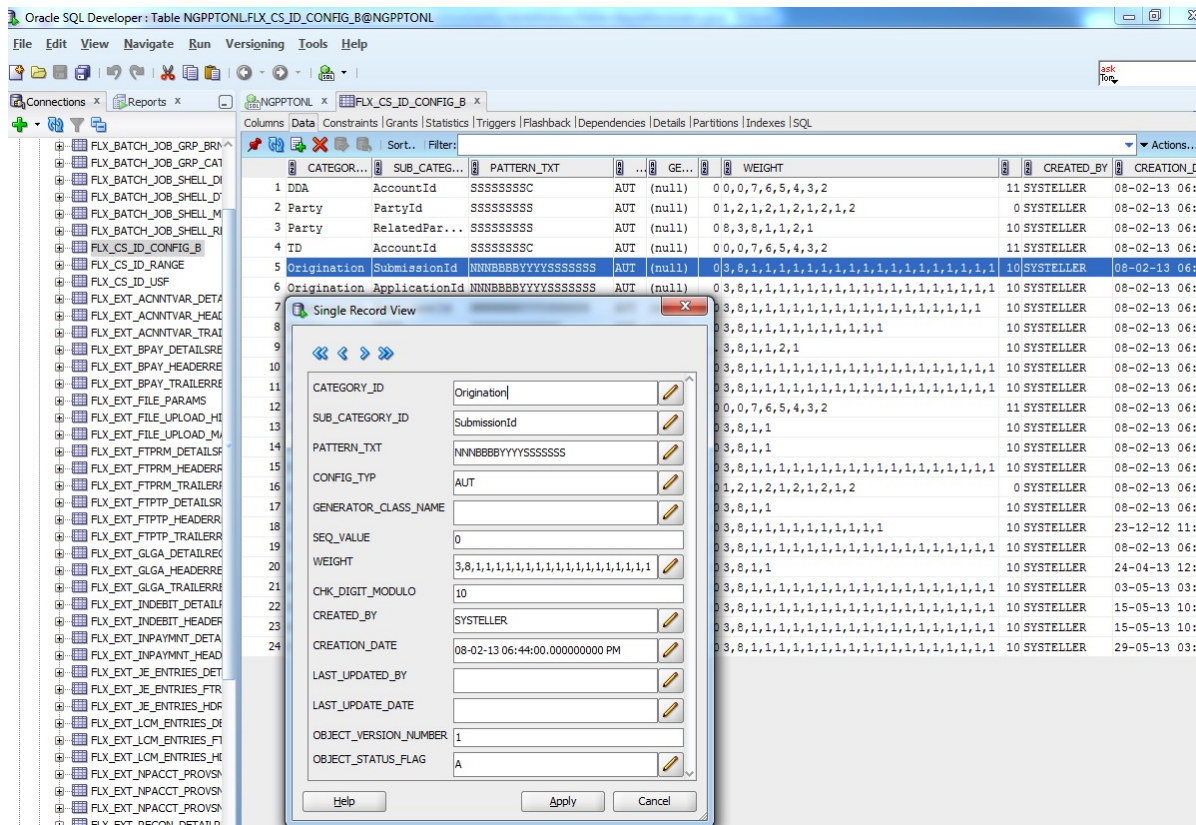
- If the pattern value is not the special character and the ID value is 'S' that is, SerialNumber, then range is looked upon:
 - If the range is defined, the current position of the range is determined based on category and sub-category. If the current position value's length is greater than pattern length, then characters between [0-length of pattern] will be generated ID, else zeros are prefixed before

current position value of range until it's size becomes pattern's length. For example, the pattern is 'SSSSSS' and the generated range gives the value as '2345' then the actual value will become '002345'.

- If range is not defined, then next value from sequence category_subCategory_SEQ is picked, it'll also be corrected to the size of pattern's length as mentioned in case of above example.
- If the pattern value contains 'C', that is, check digit. Check digit computation is done and then appended the computed value to the pre computed ID value. The input value, weight and check digit modulo are used for calculation of check-digit. The input value can be sequence ID or can be the ASCII value in case the inputs are characters. The weights will be comma separated string of the digits to be used for the calculation.
- If the pattern value contains 'R', related party identifier is used for that value.
- If the pattern value doesn't match any of the above character, the value is fetched from the pattern map for the pattern's ID and the length is adjusted to the pattern's attribute length. These pattern map characters need to be passed by the caller service for calculation.

For example, let us take the submissionId with the pattern as NNNYYYYBBBSSSSS in the database.

Figure 15–2 Automated ID Generation - Single Record View



The pattern hashmap 'value' will be populated and passed by the caller with the key value pair as pattern character as key and its corresponding value. As shown below, 'N' will contain name value, 'Y' will contain year value and 'B' will contain branch code.

Figure 15–3 Automated ID Generation - Generate Submission ID

```

301 public SubmissionApplicationService() {}
307
309 * This method is used to generate a submission Identifier based on the header information which includes
334 public SubmissionCreationResponse generateSubmissionId(SessionContext sessionContext, HeaderDTO submissionHeaderDTO) throws FatalException {}
395
397 * Generate submission_id for the Submission. this method populates the value of each format mask based on the input
405 private SubmissionKey generateSubmissionId(Submission submission) throws FatalException {
406
407     if (logger.isLoggable(Level.FINE)) {
408         logger.log(Level.FINE, formatter.formatMessage(THIS_COMPONENT_NAME + ".generateSubmissionId() Entry."));
409     }
410     HashMap<String, String> values = new HashMap<String, String>();
411     String branch = submission.getCommonData().getApplicationBranch() + "0000";
412     String year = submission.getSubmissionHeader().getSubmissionCreationDate().getYear() + "0000";
413     String type = submission.getSubmissionType() + "";
414     String name = "SUB";
415     values.put(SubmissionIdGenerationService.NAME, name);
416     values.put(SubmissionIdGenerationService.YEAR, year);
417     values.put(SubmissionIdGenerationService.BRANCH, branch);
418     values.put(SubmissionIdGenerationService.TYPE, type);
419     SubmissionIdGenerationService service = SubmissionIdGenerationService.getInstance();
420     String submissionId = service.generateSubmissionId(null, -1, values);
421     logger.log(Level.FINE, formatter.formatMessage("generateSubmissionId - submissionId = %s", submissionId));
422     if (logger.isLoggable(Level.FINE)) {
423         logger.log(Level.FINE, formatter.formatMessage(THIS_COMPONENT_NAME + ".generateSubmissionId() Exit."));
424     }
425     return new SubmissionKey(submissionId);
426 }
427

```

Figure 15–4 Automated ID Generation - Submission ID Generation Service

```

2 * Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4
5 package com.ofss.fc.domain.origination.service.core.submission.id.generation;
6
7 import java.util.HashMap;
13
15 * This class generates ID for different entities in Submission such as Submission, Application and Applicant.
19 public final class SubmissionIdGenerationService {
20
22 * Constant for Submission Id
24 private static final String SUBMISSION_ID = "SubmissionId";
26 * Constant for Origination
28 private static final String ORIGINATION = "Origination";
30 * Constant for Application Id
32 private static final String APPLICATION_ID = "ApplicationId";
34 * Constant for Applicant Id
36 private static final String APPLICANT_ID = "ApplicantId";
38 * Constant for name
40 public static final String NAME = "N";
42 * constant for type
44 public static final String TYPE = "T";
46 * Constant for branch
48 private static final String BRANCH = "B";
50 * Constant for year
52 public static final String YEAR = "Y";
54 * Constant for channel
56 private static final String CHANNEL = "H";
57
58 /**
59 * Service to generate the submission id based on configurable format mask. The default format mask is <br/>
60 * <b>"SUB"TBrcdYyyNnnnn</b><br/>
61 * the first three character is constant "SUB", following by one character that indicate the submission type
62 * {@link com.ofss.fc.enumeration.origination.SubmissionType} and combination of Branch Code + Year + Running Serial
63 * Number
64 *
65 * @fch.dar MI,String,submissionId, It is used for manual id generation, not applicable for Automatic Id
66 * generation
67 * @fch.dar MI,String,rangeId, The range id for the generator. It is used to generate the sequence in a
68 * specified range
69 * @fch.dar MI,HashMap,values, The set of values used to generate the submission id. The key is the pattern type
70 * (Name,Type,Branch,Year)
71 * @return The submission id.
72 */
73 public String generateSubmissionId(String submissionId, long rangeId, HashMap<String, String> values) throws FatalException {
74
75     IdGenerator generator = (IdGenerator) AbstractGeneratorFactory.getUniqueInstance().getIdGenerator(ORIGINATION, SUBMISSION_ID);
76     String tempId = null;
77     try {
78         tempId = generator.generateId(submissionId, rangeId, values);
79     } catch (FatalException fatalException) {
80         ErrorManager.throwFatalException(fatalException);
81     }
82     return tempId;
83 }
84

```

The ID will be generated by the automatic generator with first three characters as name, next four digits as year, next three characters of branch and rest with generated sequence as per the mask pattern.

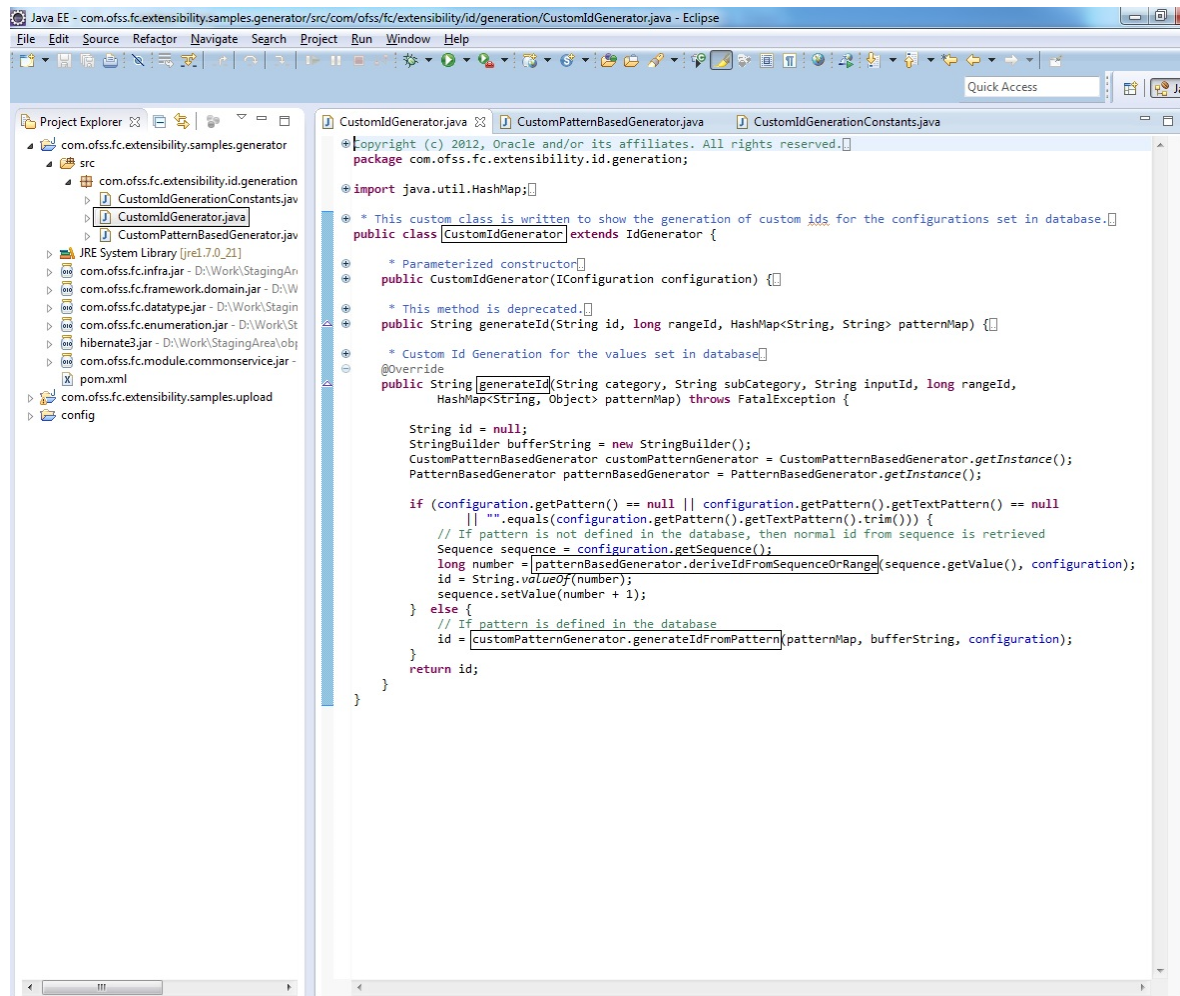
In case of without mask configuration's pattern. If range ID is -1, it means that there is no range defined for the mask configuration, it then picks up the range details with range ID based on the category and sub-category. The generated ID will become the current position of range. If range is not defined in the table, then the sequence needs to be defined and the value is picked based on that. The next value of the sequence will become the generated ID value.

15.3 Custom ID Generation

In case of configuration type as custom, user needs to set the CONFIG_TYPE as 'CUS' in the CONFIG_TYP column in the FLX_CS_ID_CONFIG_B table.

User can customize the ID generator by writing a new custom ID generator class which will need to extend the IdGenerator and write the abstract methods for the ID generation. This class needs to be mentioned in the GENERATOR_CLASS_NAME column of FLX_CS_ID_CONFIG_B table.

Figure 15–5 Custom ID Generation - Custom ID Generator



In case the user want to write the custom generation logic in a specific customized pattern definition, then user can do that by writing the custom constant class and the custom pattern class which can pick the defined pattern from the configuration object set in the PATTERN_TXT column of the FLX_CS_ID_CONFIG_B table of the database. The user will pass the values in the pattern hashmap which will then populate the pattern and generate the ID.

Figure 15–6 Custom ID Generation - Custom ID Generation Constants

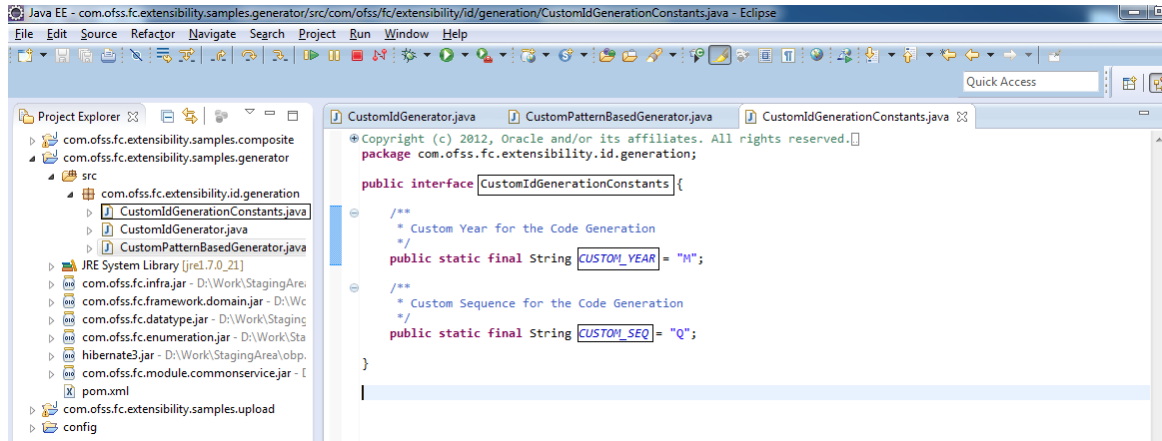


Figure 15–7 Custom ID Generation - Custom Pattern Based Generator

```

Java EE - com.offss.fc.extensibility.samples.generator/src/com/offss.fc.extensibility.id/generation/CustomPatternBasedGenerator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access

Project Explorer
  com.offss.fc.extensibility.samples.generator
    src
      com.offss.fc.extensibility.id.generation
        CustomIdGenerationConstants.java
        CustomIdGenerator.java
        CustomPatternBasedGenerator.java
  JRE System Library [jre.7.0_21]
  com.offss.fc.infra.jar - D:\Work\StagingArea\
  com.offss.fc.framework.domain.jar - D:\Work\StagingArea\
  com.offss.fc.datatype.jar - D:\Work\StagingArea\
  com.offss.fc.enumeration.jar - D:\Work\StagingArea\
  hibernate3.jar - D:\Work\StagingArea\
  com.offss.fc.module.commonservice.jar - D:\Work\StagingArea\
  pom.xml
  com.offss.fc.extensibility.samples.upload
  config

CustomIdGenerator.java CustomPatternBasedGenerator.java CustomIdGenerationConstants.java
  Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
  package com.offss.fc.extensibility.id.generation;
  import java.util.ArrayList;

  * Class to derive the value of the Id from the configuration defined.
  public class CustomPatternBasedGenerator {

  * Represents the Unique Instance of this class per JVM
  private static CustomPatternBasedGenerator uniqueInstance;
  * Private Constructor
  private CustomPatternBasedGenerator() {}
  * @return uniqueInstance for ConfigurationRepository
  * public static CustomPatternBasedGenerator getInstance() {}

  /**
  * @param rangeId, Represents the range Id
  * @param patternMap, HashMap of the pattern attributes defined.
  * @param checkDigitBuffer, StringBuilder holding checkdigit information.
  * @param bufferString
  * @param configuration for which Id is to be generated.
  * @return String, generated Id.
  * For e.g. M-QQQQ will be 2013-2345 if the sequence value is 2345 in the database.
  */
  public String generateIdFromPattern(HashMap<String, Object> patternMap, StringBuilder bufferString,
  IConfiguration configuration) throws FatalException {
    String generatedId;
    PatternBasedGenerator patternBasedGenerator = PatternBasedGenerator.getInstance();
    ArrayList<PatternAttributeDTO> definition = patternBasedGenerator.split(configuration.getPattern())
    .getTextPattern();

    /* Starting the formation of the String based on Definition that is returned */
    Iterator<PatternAttributeDTO> iterator = definition.iterator();
    while (iterator.hasNext()) {
      PatternAttributeDTO patternAttributeDTO = iterator.next();
      /* If special character then place as it is in the buffer */
      if (patternAttributeDTO.getIsSpecialCharacter()) {
        bufferString.append(patternAttributeDTO.getId());
      } else if (CustomIdGenerationConstants.CUSTOM_YEAR.equals(patternAttributeDTO.getId())) {
        /* In case of the custom year the year is added to the code */
        Calendar localCalendar = Calendar.getInstance(TimeZone.getDefault());
        int currentYear = localCalendar.get(Calendar.YEAR);
        bufferString.append(currentYear);
      } else if (CustomIdGenerationConstants.CUSTOM_SEQ.equals(patternAttributeDTO.getId())) {
        /* In case of the custom sequence the sequence value is checked and updated by 1 */
        Sequence sequence = configuration.getSequence();
        long number = patternBasedGenerator.deriveIdFromSequenceOrRange(sequence.getValue(),
        configuration);
        String temporaryGeneratedId = String.valueOf(number);
        sequence.setValue(number + 1);
        bufferString.append(temporaryGeneratedId);
      }
    }
    generatedId = bufferString.toString();
    return generatedId;
  }
}

```

16 Extensibility of Domain Objects using Flex Fields

This chapter describes about the Flex Field provisioning by the product at the Service Layer. Flex Fields are additional attributes provisioned upfront via configuration and with basic validations. By using e Flex Field, banks IT, consultants or partners can configure additional data elements to be part of the domain entities, without the need to add custom code. It provides ability to use these additional data elements in the business logic of the application. These additional elements are available in the service interface in the form of key-value pairs in Dictionary Object. Some important features of Flex Fields are:

- A Flex Field is a flexible data field that can be customized as per business needs without programming.
- Only Seed Metadata Configuration is required to enable Flex Fields.
- Flex field attributes can be provisioned at each entity level [Max 30 attributes per entity].
- Flex Field Attribute are strongly typed at an definition & business logic validation level. However, the Data Type is maintained as String in the entity and VARCHAR (250) in database.
- Service input or data transfer is supported via Dictionary Object (Separate indicator is provided in Dictionary Object to distinguish flex field dictionary object)

Utilizing Flex Fields in OBP requires following the two steps mentioned below:

- **Flex Field Provisioning:** This step denotes ensuring that Table of Entity which is candidate for extension has required number of columns in database. Product out of the box ships sufficient number of Flex Field columns for all Entities eligible for extension. Still if Consultant wants more Flex Field columns to utilize, then this is Optional step is needed.
- **Flex Field Utilization:** This step denotes utilizing required number of Flex Fields by providing its Metadata. Utilized Flex Fields should be always less or equal to Provisioned flex Fields.

16.1 Flex Field - Provisioning

Maximum 30 attributes per entity can be provisioned at each entity level. Each Attribute data type is declared as String in Java Entity (inside superclass AbstractDomainObject) and VARCHAR (255) in table for flexibility.

16.1.1 How to know Maximum Flex Fields Provisioned for Entity?

- FLX_FW_FLX_FLD_ENTITIES_ALL_B seed table contains list of all Product Entity tables for which Flex Fields are provisioned.
- MAX_FLX_FLD_COLUMN_COUNT column in this table denotes Maximum number of Flex Fields currently provisioned for that Entity. This value cannot exceed 30.
- Consultant can utilize only those number of Flex Fields which are provisioned as per MAX_FLX_FLD_COLUMN_COUNT column value.
- If Consultant wants to utilize more Flex Fields than that are already provisioned as per MAX_FLX_FLD_COLUMN_COUNT column value, then it can be done using steps mentioned in section below.

16.1.2 Increase Maximum Flex Fields Provisioned for Entity (Optional Step)

- In order to increase number of Provisioned Flex Fields for Entity, Consultant needs to update value of column MAX_FLX_FLD_COLUMN_COUNT in table FLX_FW_FLX_FLD_ENTITIES_ALL_B. This number should not exceed 30.
- After that Consultant needs to execute stored procedure AP_FW_PROVISION_FLEX_FIELDS.
- This stored procedure will read seed table FLX_FW_FLX_FLD_ENTITIES_ALL_B and alter those entity tables to add missing Flex Fields columns in the table.
- This procedure assumes that few Flex Field Columns might be already present in table and will not impact those columns or tables.

16.2 Flex Field - Utilization

Utilization of Flex Field can be done using two steps:

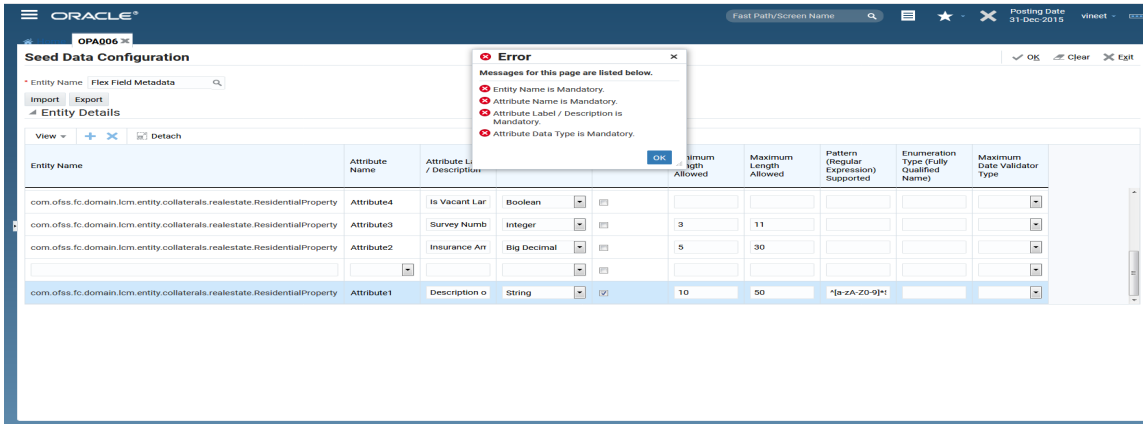
- Maintain Flex Field Metadata by:
 - Using Seed Data Configuration (Fast Path: OPA006) page.
 - Directly making seed entries in table FLX_FW_FLX_FLD_METADATA_B.
- Restart host Server.

16.2.1 Maintain Flex Field Metadata using Seed Data Configuration (Fast Path: OPA006) Page

- Metadata gives information about Domain Object Name and following information about each Flex Field Attribute:
 - Name - Mandatory
 - Label - Mandatory
 - Data Type - Mandatory
 - Is Mandatory - Optional
 - Min Length - Optional
 - Max Length - Optional
 - RegEx Pattern - Optional
 - Enum Type - Mandatory if Data type is Enum
 - Max Date Validator - Optional and Applicable only for Date Data Type
- Go to OPA006 page. Select Entity Name as Flex Field Metadata. Details of existing Flex Field entity Metadata will be displayed in read mode.
- First two columns (Entity Name, Attribute Name) will be in read only mode for existing values (As these are Key values). However, when user clicks Add, these two columns will be editable only for Newly added rows.

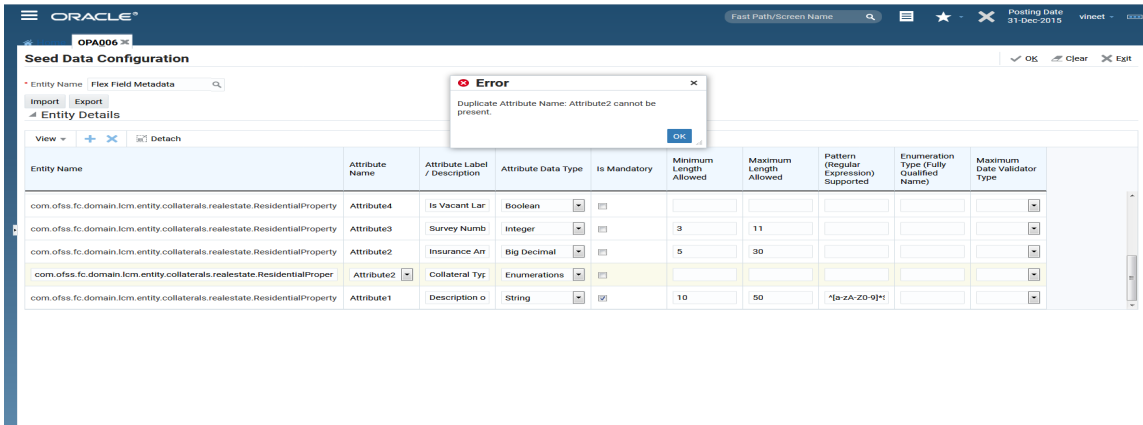
- If user clicks Add to add new row and if a user clicks final OK without entering Entity Name, Attribute Name, Attribute Label or Description, and Attribute Data Type, then the following error is displayed.

Figure 16–1 OPA006 - Error



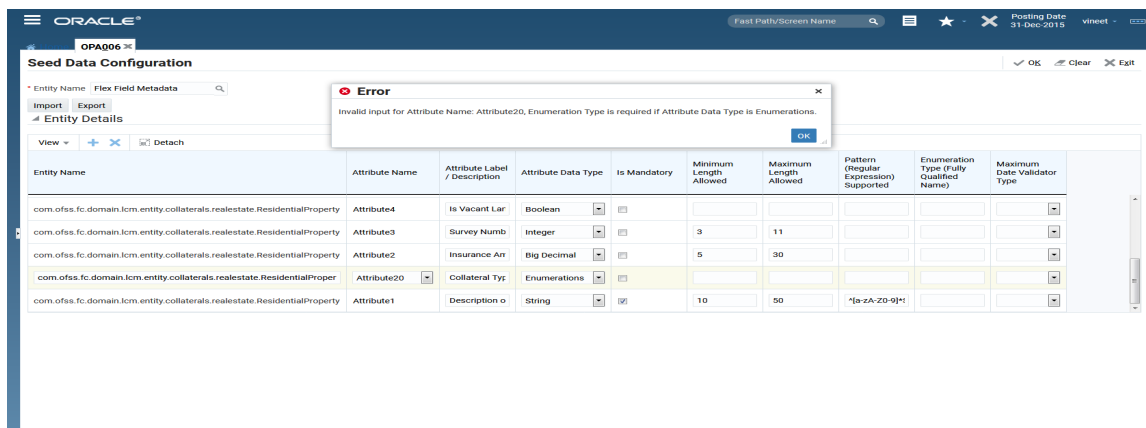
- Duplicate Attribute names cannot be present. If present, then the following error is displayed.

Figure 16–2 OPA006 - Duplicate Attribute



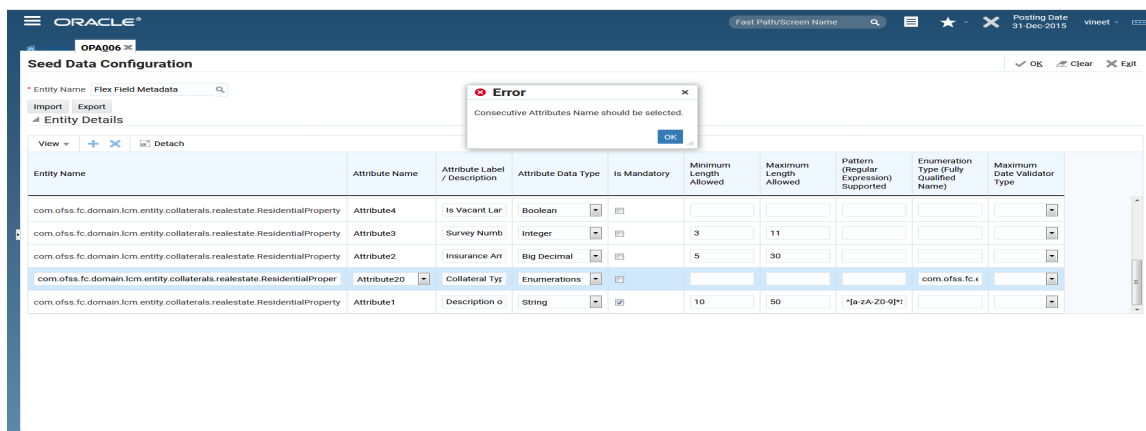
- When Attribute Data type is chosen as Enumerations then Enumeration Type should be present. If not, then the following error is displayed.

Figure 16–3 OPA006 - Enumeration Type



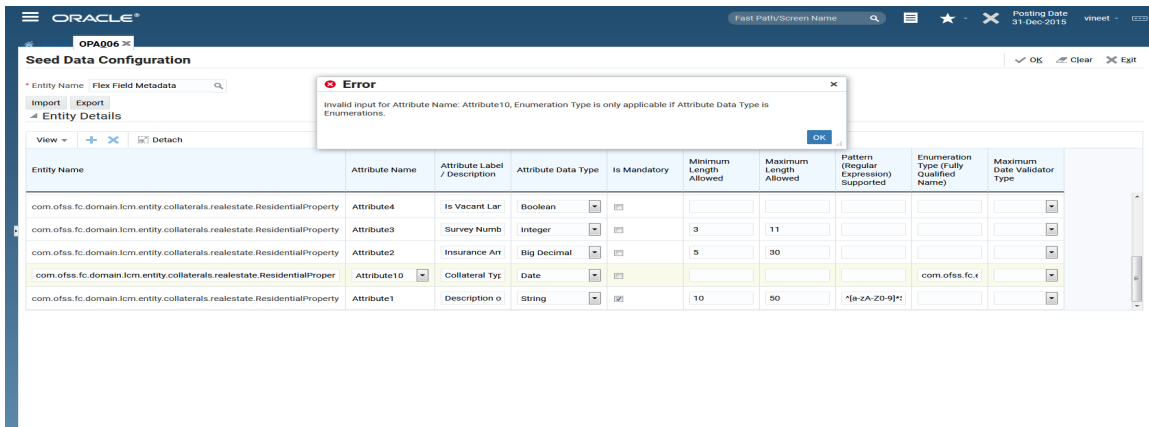
- Consecutive attributes names should be present. User cannot skip any attribute name otherwise the following error is displayed.

Figure 16–4 OPA006 - Consecutive Attributes Names



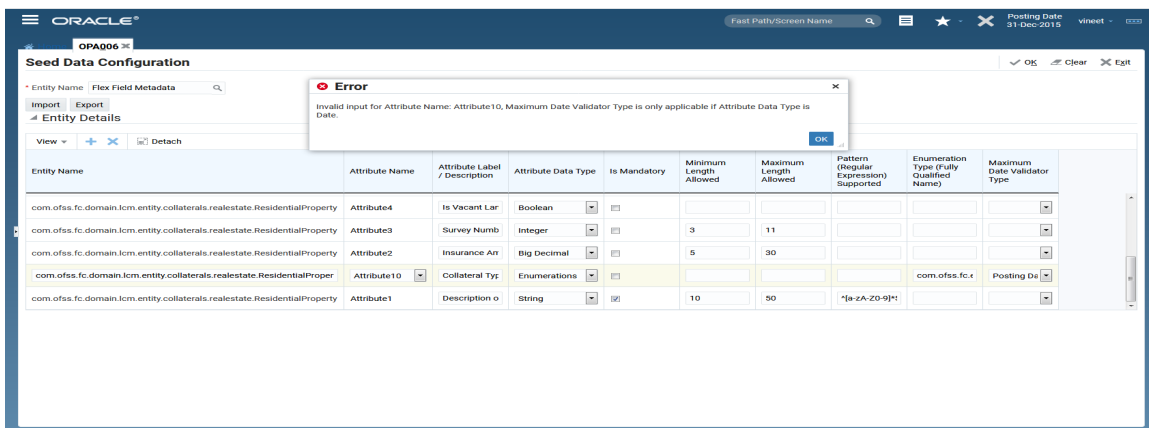
- Enumeration type is only applicable if Attribute Data Type is Enumerations. If other than Enumerations is chosen as an Attribute Data Type, then the following error is displayed.

Figure 16–5 OPA006- Invalid Input for Attribute Data Type.



- Maximum Date Validator type is only applicable if Attribute Data Type is Date. If other than Date is chosen as an Attribute Data Type, then the following error is displayed.

Figure 16–6 Attribute Data Type as Date



- If all information is valid Flex Field Metadata is saved in table FLX_FW_FLX_FLD_METADATA_B.
- Alternatively instead of using OPA006 page, Consultant can directly add seed data in FLX_FW_FLX_FLD_METADATA_B table for required Flex Fields and check-in the same.
- Flex field metadata is mandatory and useful to achieve following:
 - Clearly indicates out of Provisioned Flex Fields which are actually utilized and for which purpose. Flex Field Metadata is mandatory (at least label and data type). This is extremely important when, as a part of previous customization if few Flex Fields are utilized then Consulting should clearly know used fields and available fields.
 - Attribute Description or Label mapping.
 - Attribute Data Type mapping and supporting basic validations.
 - Throwing validation message with respective label for better interpretation.
- After populating Flex Field Metadata, Host Server restart is needed.

- On Host Restart, during ORM loading for each Entity in FLX_FW_FLX_FLD_METADATA_B table Flex Field related ORM mapping is added dynamically as per Metadata using EclipseLinkSessionCustomizer class.
- Out of Provisioned columns only optimal number of table columns are actually used during this dynamic ORM mapping based on fields specified in FLX_FW_FLX_FLD_METADATA_B table.
- Supported sizes are: 5/10/15/20/25/30 fields.
- AbstractDomainObject holds following objects to support these flexible sizes. One of these fields is chosen for ORM mapping per Entity as per Metadata configuration.
 - FiveAttributeFlexField
 - TenAttributeFlexField which extends FiveAttributeFlexField
 - FifteenAttributeFlexField which extends TenAttributeFlexField
 - TwentyAttributeFlexField which extends FifteenAttributeFlexField
 - TwentyFiveAttributeFlexField which extends TwentyAttributeFlexField
 - ThirtyAttributeFlexField which extends TwentyFiveAttributeFlexField

16.3 Runtime Storage and Retrieval of Flex Field Attribute values

- Service input or data transfer is supported via Dictionary Object. Separate indicator named flexField is provided to distinguish flex field dictionary object.
- The attributes which are passed as part of the Dictionary object with the Flex Field indicator, will be persisted as Flex Field in the respective Entity Table. The attribute name follow the name convention as “Attribute<<attribute number>>” [‘A’ caps, like Attribute1, Attribute2, Attribute3, and Attribute30].
- For example, com.ofss.fc.domain.lcm.entity.collaterals.realestate.Industrialproperty this Entity is configured to make use of Flex Field attribute by defining metadata.
- Data is saved using Dictionary input as follows, if API is being tested using SOAP UI Toolkit:

Figure 16–7 Runtime Storage Saved Data

```

<dto:dictionaryArray>
  <dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.rea
  <dto:flexField>true</dto:flexField>
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <!-- Description of Land -->
    <dto:name>Attribute1</dto:name>
    <dto:value>Industrial Property Land Description</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <!-- Vacant Land -->
    <dto:name>Attribute2</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
</dto:dictionaryArray>

```

- Data is retrieved using Dictionary as follows, if API is being tested using SOAP UI Toolkit:

Figure 16–8 Runtime Storage Retrieved Data

```

</response>
<collateralsdto:collateralArrayDTO xsi:type="realstatecollateralsdto:collateralArrayDTO" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dtocommondomainframework:dictionaryArray>
    <dtocommondomainframework:flexField>true</dtocommondomainframework:flexField>
    <dtocommondomainframework:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.realstate.IndustrialProperty</dtocommondomainframework:fullyQualifiedClassName>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute1</dtocommondomainframework:name>
      <dtocommondomainframework:value>Industrial Property Land Description</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute2</dtocommondomainframework:name>
      <dtocommondomainframework:value>N</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute3</dtocommondomainframework:name>
      <dtocommondomainframework:value>N</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute4</dtocommondomainframework:name>
      <dtocommondomainframework:value>N</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute5</dtocommondomainframework:name>
      <dtocommondomainframework:value>N</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
    <dtocommondomainframework:nameValuePairDTOArray>
      <dtocommondomainframework:name>Attribute6</dtocommondomainframework:name>
      <dtocommondomainframework:value>N</dtocommondomainframework:value>
    </dtocommondomainframework:nameValuePairDTOArray>
  </dtocommondomainframework:dictionaryArray>
</collateralsdto:collateralArrayDTO>

```

16.4 Flex Field - Fact support

Flex Fields provisioned can be consumed as facts as below,

- Derived facts can be created, with following Derivation Type.
 - Java Based Derivation
 - JPQL Query Based Derivation
 - SQL Query Based Derivation
- Following is example of JPQL based Fact which uses Flex Field attribute.
- com.ofss.fc.domain.lcm.entity.collaterals.realstate.IndustrialProperty this Entity is configured to make use of FiveAttributeFlexField by defining Flex Field attribute metadata.
- Data will be saved using Dictionary input as follows, if API is being tested using SOAP UI Toolkit:

Figure 16–9 Flex Field - Fact support Saved Data

```

<dto:dictionaryArray>
  <dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.realstate.IndustrialProperty</dto:fullyQualifiedClassName>
  <dto:flexField>true</dto:flexField>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute1</dto:name>
    <dto:value>Industrial Property Land Description</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute2</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute3</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute4</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute5</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
  <dto:nameValuePairDTOArray>
    <dto:name>Attribute6</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
</dto:dictionaryArray>

```

- After this save API call, Data will be stored in table as follows:

Figure 16–10 Flex Field - Fact support Data in Table



COLLATERAL_ID	FF_ATTRIBUTE_1	FF_ATTRIBUTE_2	FF_ATTRIBUTE_3	FF_ATTRIBUTE_4	FF_ATTRIBUTE_5
1	COL201600100000193	Industrial Property Land Description N	(null)	(null)	(null)

- Fact seed can be added to create fact named Collateral.IndustrialProperty.VacantLand, which will return value of Flex Field Attribute1, by defining following Fact Seed. Note that in JPQL query a.fiveAttributeFlexField.attribute1 is used to select value of Flex Field, as for this entity Metadata is given for less than or equal to 5 Fields, hence AbstractDomainObject.fiveAttributeFlexField is chosen as field to be mapped for dynamic Flex Field ORM mapping. Depending upon number of Flex Fields actually utilized, appropriate field out of following AbstractDomainObject class fields can be used in query: fiveAttributeFlexField, tenAttributeFlexField, fifteenAttributeFlexField, twentyAttributeFlexField, twentyFiveAttributeFlexField, thirtyAttributeFlexField.

Figure 16–11 Flex Field - Fact support Seed

```

insert into FLX_FA_FACTS_B (FACT_CODE, FACT_NAME, FACT_DESC, DOMAIN_CODE, DOMAIN_CATEGORY_CODE, VALUE_TYPE, BUSINESS_TYPE_CODE, RETRIEVAL_KEY, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_DATE, OBJECT_VERSION_NUMBER, LAST_UPDATE_LOGIN, FACT_CLASS, SHORT_NAME)
values ('Collateral.IndustrialProperty.VacantLand', 'Collateral.IndustrialProperty.VacantLand', 'VacantLand', 'Banking', 'Open', 'Alphanumeric',
'Collateral.IndustrialProperty.VacantLand', 'SYSTEILLER', to_timestamp('01-02-2012 18:39:25.461000', 'dd-mm-yyyy hh24:mi:ss.ff'), 'SYSTEILLER', to_timestamp(
'01-02-2012 18:39:25.479000', 'dd-mm-yyyy hh24:mi:ss.ff'), 2, null, 'NonFinancial', 'VacantLand');

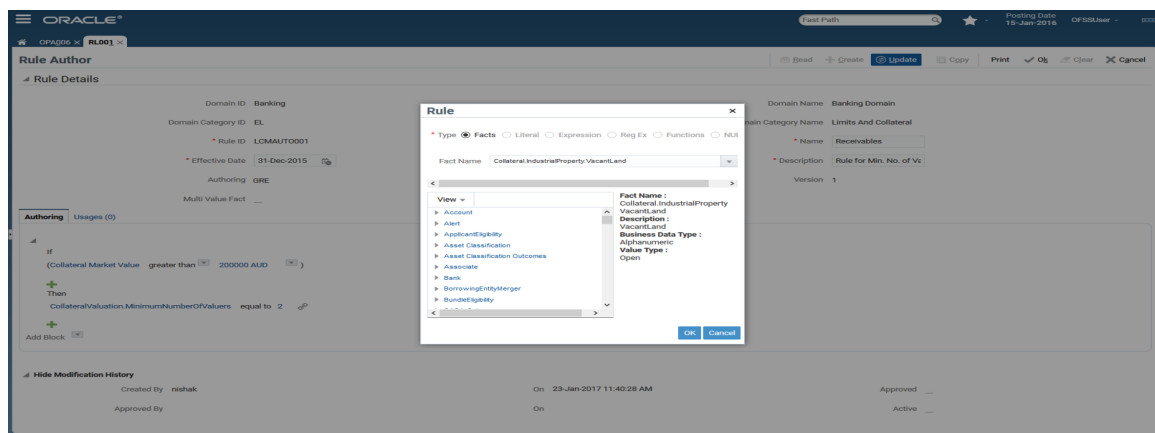
insert into FLX_FA_GROUP_XREF (FACT_CODE, GROUP_CODE, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_DATE, OBJECT_VERSION_NUMBER, LAST_UPDATE_LOGIN)
values ('Collateral.IndustrialProperty.VacantLand', 'Collateral.IndustrialProperty', null, null, null, null, null, null);

insert into FLX_FA_VALUE_BINDINGS (FACT_CODE, PARAM_NAME, DATA_TYPE_CODE, PARAM_DESC, VARIABLE_BASED_FACT, LITERAL_FACT_VALUE, BINDING_TYPE)
values ('Collateral.IndustrialProperty.VacantLand', 'id', null, null, 'Collateral.id', null, 'Variable');

insert into flx_fa_value_datasources (FACT_CODE, JDBC_DERIVATION_QUERY, JAVA_DERIVATION_CLASS, DATA_SOURCE_CODE, HQL_DERIVATION_QUERY, DSN_CODE, DB_FUNCTION_NAME)
values ('Collateral.IndustrialProperty.VacantLand', null, null, 'HQL', 'SELECT a.fiveAttributeFlexField.attribute1 FROM
com.offss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty a WHERE a.collateralKey.id = :id', null, null);
    
```

- After adding this seed, Collateral.IndustrialProperty.VacantLand will be available as Fact while defining Pricing or Rules in OBP as follows:

Figure 16–12 Flex Field - Fact support After Seed



16.5 Flex Field – Validation Support

Basic validations are supported for Flex Fields using Metadata configuration (Refer [Section 16.2.1 Maintain Flex Field Metadata using Seed Data Configuration \(Fast Path: OPA006\) Page](#)). Using Flex field Metadata for each attribute in Entity, supported validation can be configured. Find below the details on the metadata configuration for supported validations. This needs be seeded or can be maintained using OPA006 page. This requires host restart to reflect.

Table 16–1 metadata configuration details for validations

Column	Description	Example
ENTITY_NAME	Name of the entity where flex field is applicable. Full qualified name.	com.ofss.fc.-domain.lcm.entity.collaterals.realestate.ResidentialProperty
ATTRIBUTE_NAME	Name of the attribute of the flex field. Attribute1, Attribute 2, or so on...	Attribute1
LABEL	Label or description of the attribute. When validation error message is thrown, this is used to throw exception.	Description of land

Column	Description	Example
	If not maintained, then the attribute name will be used for the validation message.	
ATTRIBUTE_DATA_TYPE	Attribute data type. Supported Data Types are: STRING / BIG_DECIMAL / INTEGER / BOOLEAN / DATE / ENUM. Used when validating the field based on the data-type.	STRING
IS_MANDATORY	Validator field: Indicates	Y

Column	Description	Example
	whether attribute value is mandatory. Check for not null / empty values.	
MIN_LENGTH	Validator field: Indicates the minimum length required for the attribute. Validates, if maintained some value.	5
MAX_LENGTH	Validator field: Indicates the maximum length required for the attribute. Validates, if maintained some value.	250

Column	Description	Example
PATTERN_REGEX	Validator field: Indicates the regular expression supported by the attribute. Validates, if maintained some value.	^[a-zA-Z0-9]*\$
ENUM_TYPE_NAME	Validator field: Indicates the enumeration type supported by the attribute. Fully qualified name. Checks with the enumeration value sent. Validates, if maintained some value and data type	<<Applicable only for Enum type, e.g., com.ofss.fc.enumeration.lcm.collaterals.CollateralType>>

Column	Description	Example
	is Enum.	
MAX_DATE_VALIDATOR_TYPE	<p>Validator field: Indicates the maximum date validator type. (Posting date / System date / Value date). Validates whether the date is not more than the mentioned date validator type. Date validator type supported are Posting date, System date and Value date. Validates, if maintained some value and data type is date.</p>	<<Applicable only for Date type, e.g., POSTING_DATE>>

- For Example, suppose `com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty` and `com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty` these Entities are configured to make use of Flex Fields by defining attribute metadata as follows:

Figure 16–13 Flex Field - Validation Support - Metadata

The screenshot shows the 'Seed Data Configuration' interface. The 'Entity Name' is 'Flex Field Metadata'. The 'Entity Details' section displays a table with the following columns: Entity Name, Attribute Name, Label, Attribute Data Type, Is Mandatory, Min Length, Max Length, Pattern (Regular Expression), Enumeration Type, and Max Date Validation Type.

Entity Name	Attribute Name	Label	Attribute Data Type	Is Mandatory	Min Length	Max Length	Pattern (Regular Expression)	Enumeration Type	Max Date Validation Type
com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty	Attribute3	Contract Amount	Big Decimal	—					
com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty	Attribute2	Vacant Land	Boolean	—					
com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty	Attribute1	Description of land	String	✓	5	250	*[a-zA-Z0-9]*\$		
com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty	Attribute4	Collateral Type	Enumerations	—				com.ofss.fc.enumeration.lcm.collaterals.Collateral...	
com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty	Attribute3	Contract Date	Date	—					Posting Date
com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty	Attribute2	Vacant Land	Boolean	—					
com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty	Attribute1	Description of land	String	✓	5	250	*[a-zA-Z0-9]*\$		

- Following is examples depict output of Flex Field validation framework, at runtime, when tested using SOAP UI toolkit.
- Mandatory Validation
 - Input:

Figure 16–14 Flex Field - Validation Support - Mandatory Validation - Input

```
<dto:dictionaryArray>
  <dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.re
  <dto:flexField>true</dto:flexField>

  <!--<dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute1</dto:name>
    <dto:value>Industrial Property Land Description</dto:value>
  </dto:nameValuePairDTOArray-->

  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <!-- Vacant Land -->
    <dto:name>Attribute2</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
</dto:dictionaryArray>
```

- Output:

Figure 16–15 Flex Field - Validation Support - Mandatory Validation -Output

```
com.ofss.fc.webservice.servlet.XSIFilter.doFilter(XSIFilter.java:122)
com.ofss.fc.infra.das.orm.SessionCleanupFilter.doFilter(SessionCleanupFilter.java:87)
ava.security.AccessController.doPrivileged(Native Method)
</exceptioninfra:message>
  <exceptioninfra:responseCode>FATAL_INT_ERROR</exceptioninfra:responseCode>
  <exceptioninfra:validationErrors>
    <errorvalidationinfra:associatedSeverity>0</errorvalidationinfra:associatedSeverity>
    <errorvalidationinfra:attributeName>Description of land</errorvalidationinfra:attributeName>
    <errorvalidationinfra:errorCode>2516</errorvalidationinfra:errorCode>
    <errorvalidationinfra:errorMessage>Invalid input.Invalid input. Mandatory field missing</errorvalidationinfra:errorMessage>
    <errorvalidationinfra:errorMessageParams>Invalid input. Mandatory field missing</errorvalidationinfra:errorMessageParams>
    <errorvalidationinfra:objectName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</errorvalidationi
  </exceptioninfra:validationErrors>
</exceptioninfra:FatalException>
</detail>
</S:Fault>
</S:Body>
```

- Min/Max Length Validation

- Input:

Figure 16–16 Flex Field - Validation Support - Min/Max Length Validation Input

```
<dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.co:
<dto:flexField>true</dto:flexField>

<dto:nameValuePairDTOArray>
  <dto:dat/>
  <dto:genericName/>
  <dto:name>Attribute1</dto:name>
  <dto:value>Industrial Property Land Description</dto:value>
</dto:nameValuePairDTOArray>

<dto:nameValuePairDTOArray>
  <dto:dat/>
  <dto:genericName/>
  <!-- Vacant Land -->
  <dto:name>Attribute2</dto:name>
  <dto:value>YN</dto:value>
</dto:nameValuePairDTOArray>
</dto:dictionaryArray>
```

- Output:

Figure 16–17 Flex Field - Validation Support - Min/Max Length Validation Output

```

ofss.fc.infra.das.orm.SessionCleanupFilter.doFilter(SessionCleanupFilter.java:87)
.security.AccessController.doPrivileged(Native Method)
</exceptioninfra:message>
  <exceptioninfra:responseCode>FATAL_INT_ERROR</exceptioninfra:responseCode>
  <exceptioninfra:validationErrors>
    <errorvalidationinfra:associatedSeverity>0</errorvalidationinfra:associatedSeverity>
    <errorvalidationinfra:attributeName>Vacant Land</errorvalidationinfra:attributeName>
    <errorvalidationinfra:attributeValue>YN</errorvalidationinfra:attributeValue>
    <errorvalidationinfra:errorCode>2516</errorvalidationinfra:errorCode>
    <errorvalidationinfra:errorMessage>Invalid input. Invalid input. Field Length mismatch. Max Length Allowed:1</errorvalidationinfra:erro
    <errorvalidationinfra:errorMessageParams>Invalid input. Field Length mismatch. Max Length Allowed:1</errorvalidationinfra:errorMessage
    <errorvalidationinfra:objectName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</errorvalidationinfra:objectN
  </exceptioninfra:validationErrors>
</exceptioninfra:FatalException>
.. ..

```

- Pattern (Regex) Validation

- Input:

Figure 16–18 Flex Field - Validation Support - Pattern (Regex) Validation Input

```

<dto:dictionaryArray>
  <dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</dto:fullyQual
  <dto:flexField>true</dto:flexField>

  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute1</dto:name>
    <dto:value>Industrial Property Land ?#@ Description </dto:value>
  </dto:nameValuePairDTOArray>

  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <!-- Vacant Land -->
    <dto:name>Attribute2</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
</dto:dictionaryArray>

```

- Output:

Figure 16–19 Flex Field - Validation Support - Pattern (Regex) Validation Output

```

security.AccessController.doPrivileged(Native Method)</exceptioninfra:message>
  <exceptioninfra:responseCode>FATAL_INT_ERROR</exceptioninfra:responseCode>
  <exceptioninfra:validationErrors>
    <errorvalidationinfra:associatedSeverity>0</errorvalidationinfra:associatedSeverity>
    <errorvalidationinfra:attributeName>Description of land</errorvalidationinfra:attributeName>
    <errorvalidationinfra:attributeValue>Industrial Property Land :##</errorvalidationinfra:attributeValue>
    <errorvalidationinfra:errorCode>2616</errorvalidationinfra:errorCode>
    <errorvalidationinfra:errorMessage>Invalid input.Invalid input. Field pattern mismatch. Regex allowed:^[a-zA-Z0-9 ]*$/errorvalidationinf
    <errorvalidationinfra:errorMessageParams>Invalid input. Field pattern mismatch. Regex allowed:^[a-zA-Z0-9 ]*$/errorvalidationinfra:error
    <errorvalidationinfra:objectName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</errorvalidationinfra:objectName
  </exceptioninfra:validationErrors>
</exceptioninfra:FatalException>
</detail>
/S:Fault>

```

■ Enum Validation

- Input:

Figure 16–20 Flex Field - Validation Support - Enum Validation Input

```

</dto:nameValuePairDTOArray>
  <!-- Collateral Type (Enum) -->
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute4</dto:name>
    <dto:value>PROPERTYXX</dto:value>
  </dto:nameValuePairDTOArray>

```

- Output

Figure 16–21 Flex Field - Validation Support - Enum Validation Output

```

.ofss.fc.infra.das.orm.SessionCleanupFilter.doFilter(SessionCleanupFilter.java:87)
i.security.AccessController.doPrivileged(Native Method)</exceptioninfra:message>
  <exceptioninfra:responseCode>FATAL_INT_ERROR</exceptioninfra:responseCode>
  <exceptioninfra:validationErrors>
    <errorvalidationinfra:associatedSeverity>0</errorvalidationinfra:associatedSeverity>
    <errorvalidationinfra:attributeName>Collateral Type</errorvalidationinfra:attributeName>
    <errorvalidationinfra:attributeValue>PROPERTYXX</errorvalidationinfra:attributeValue>
    <errorvalidationinfra:errorCode>2616</errorvalidationinfra:errorCode>
    <errorvalidationinfra:errorMessage>Invalid input.Invalid input. Couldnt resolve enum value for the type:
    com.ofss.fc.enumeration.lcm.collaterals.CollateralType</errorvalidationinfra:errorMessage>
    <errorvalidationinfra:errorMessageParams>Invalid input. Couldnt resolve enum value for the type:
    com.ofss.fc.enumeration.lcm.collaterals.CollateralType</errorvalidationinfra:errorMessageParams>
    <errorvalidationinfra:objectName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</errorvalidationinfra:objectName
  </exceptioninfra:validationErrors>
</exceptioninfra:FatalException>
</detail>

```

- Date Validation
 - Input:

Figure 16–22 Flex Field - Validation Support - Date Validation Input

```

<dto:dictionaryArray>
  <dto:fullyQualifiedClassName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProp
  <dto:flexField>true</dto:flexField>
  <!-- Description of Land (String) -->
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute1</dto:name>
    <dto:value>Industrial Property Land Desc</dto:value>
  </dto:nameValuePairDTOArray>
  <!-- Is Vacant Land (Boolean) -->
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute2</dto:name>
    <dto:value>N</dto:value>
  </dto:nameValuePairDTOArray>
  <!-- Contract Date (Date) -->
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute3</dto:name>
    <dto:value>20170115</dto:value>
  </dto:nameValuePairDTOArray>
  <!-- Collateral Type (Enum) -->
  <dto:nameValuePairDTOArray>
    <dto:dat/>
    <dto:genericName/>
    <dto:name>Attribute4</dto:name>
    <dto:value>PROPERTY</dto:value>
  </dto:nameValuePairDTOArray>
</dto:dictionaryArray>

```

- Output:

Figure 16–23 Flex Field - Validation Support - Date Validation Output

```

com.ofss.fc.infra.orm.session.servlet.DefaultValuesFilter.doFilter(DefaultValuesFilter.java:94)
com.ofss.fc.domain.lcm.service.collaterals.CollateralService.updateCollateral(CollateralService.java:366)
com.ofss.fc.app.lcm.service.collaterals.CollateralApplicationService.updateCollateral(CollateralApplicationService.java:562)
com.ofss.fc.app.lcm.service.collaterals.CollateralApplicationServiceSpi.updateCollateral(CollateralApplicationServiceSpi.java:132)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
com.ofss.fc.webservice.servlet.DefaultValuesFilter.doFilter(DefaultValuesFilter.java:94)
ava.security.AccessController.doPrivileged(Native Method)
com.ofss.fc.webservice.servlet.XSIFilter.doFilter(XSIFilter.java:122)
com.ofss.fc.infra.das.orm.SessionCleanupFilter.doFilter(SessionCleanupFilter.java:87)
ava.security.AccessController.doPrivileged(Native Method) </exceptioninfra:message>
<exceptioninfra:responseCode>FATAL_INT_ERROR</exceptioninfra:responseCode>
<exceptioninfra:validationErrors>
  <errorvalidationinfra:associatedSeverity>0</errorvalidationinfra:associatedSeverity>
  <errorvalidationinfra:attributeName>Contract Date</errorvalidationinfra:attributeName>
  <errorvalidationinfra:attributeValue>20170115</errorvalidationinfra:attributeValue>
  <errorvalidationinfra:errorCode>2516</errorvalidationinfra:errorCode>
  <errorvalidationinfra:errorMessage>Invalid input.Invalid input. Date is greater than POSTING_DATE:20160115000000</errorvalidationinfra:err
  <errorvalidationinfra:errorMessageParams>Invalid input. Date is greater than POSTING_DATE:20160115000000</errorvalidationinfra:errorMessage
  <errorvalidationinfra:objectName>com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty</errorvalidationinfra:objectName>
</exceptioninfra:validationErrors>
</exceptioninfra:FatalException>
</detail>
</S:Fault>
</S:Body>

```

16.6 Flex Field – Usage Instructions

Perform the following steps for usage:

- Identify the Entity for which Flex Field support is required.
- Refer seed table FLX_FW_FLX_FLD_ENTITIES_ALL_B to ascertain that Flex Field is already provisioned that Entity Table with sufficient number of MAX_FLX_FLD_COLUMN_COUNT.
- If Flex Field is not provisioned in FLX_FW_FLX_FLD_ENTITIES_ALL_B then, verify with the product team.
- In case Flex Field is provisioned but MAX_FLX_FLD_COLUMN_COUNT is not sufficient then follow steps mentioned in section to get extra columns provisioned [Section 16.1 Flex Field - Provisioning](#)
- Pass or Retrieve the attributes via Dictionary object, as per [Section 16.3 Runtime Storage and Retrieval of Flex Field Attribute values](#).

- If validation required for any of the attributes in the Flex Field, configure or seed the metadata as per [Section 16.5 Flex Field – Validation Support](#) . By dynamic provisioning & configuration page, this will be enabled via configuration page: OPA006.
- If fact required, follow [Section 16.2 Flex Field - Utilization](#) for details.

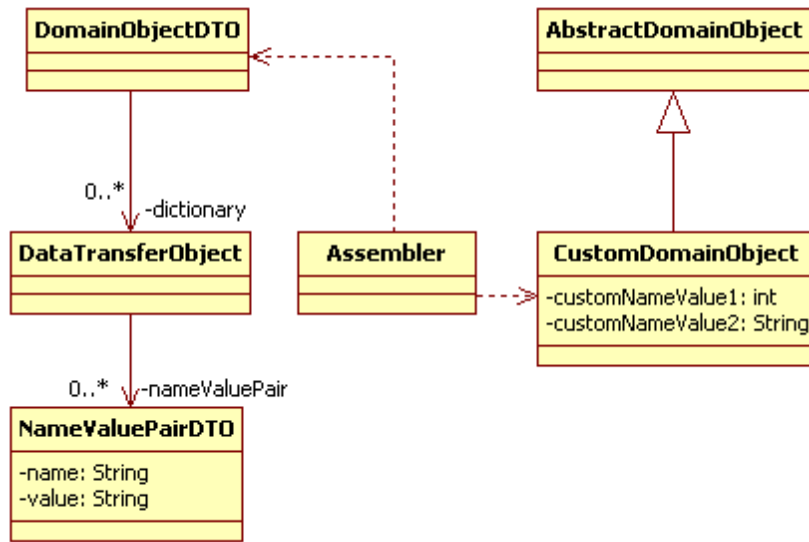
17 Extensibility of Domain Objects - Dictionary Pattern

This chapter describes how consultants or other third parties can extend OBP domain by leveraging the dictionary design pattern to extend any Abstract Domain Object on which a maintenance screen and corresponding services are supported by product and are shipped for a release. This pattern provides true domain model extension capabilities by allowing addition of custom data fields to the underlying domain objects and the database tables mapped to them. In this approach, the data model for the custom fields is extended from that of the domain objects itself and hence can be consumed in business policies or even rules as facts. The dictionary pattern enables using the custom data fields in the extensions, business rules (as facts) and custom business policies as the domain object load from the database retrieves the extended domain object and not just the product domain object.

The framework related changes to make such support available are supported from release 2.3 of the Oracle Banking Platform. These changes have been made across layers including the UI, JSON, Assembler, ORM and DB layer. The changes required to be made by consulting to support the persistence and usage of the extra attributes by extending the product domain object have been discussed in detail in the sections by taking common domain extensibility use cases as examples. The process in which data is transferred from the UI layer, to the host layer is mentioned briefly as points below:

- The proxy layer provides an extension point wherein the additional data fields on the screen can be populated as name value pairs and set in the input request.
- The custom attribute data gets passed through the JSON layer onto the middleware host as part of the application service invocation.
- These name value pairs are translated into the custom domain object which extends the base OBP domain object.
- The custom fields get persisted into the DB along with the domain object fields as part of ORM mapping.
- Exact opposite flow follows for inquiry services in which the data flows back via output response.

Figure 17–1 Extensibility of Domain Objects - Framework



The dictionary data is passed in the request DTO and is therefore available as part of the pre and post application service extensions. The above process is described in detail in the sections below.

17.1 Customized Domain Object Attribute Placeholders

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 accessor (or getter) methods on the DTO to get the individual attribute values from the Transfer Object. All request response classes in OBP application services are modelled as data transfer objects. These 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. The following is mentioned as points below:

- All DTO classes should extend DomainObjectDTO class.
- The DomainObjectDTO class has been made to extend DataTransferObject class.
- This class has a single attribute which is an array of Dictionary class.
- Dictionary class has a single attribute which is an array of NameValuePairDTO

Using an array of name value pairs inside an array of dictionary allows for supporting two dimensional grid structures in the UI layer.

At present whenever any third party requires support for additional attributes in a Domain Object, the information regarding the corresponding Customized Domain Object name and attribute name-value pair is required to be populated as an array of NameValuePairDTO which in turn is set in the Dictionary class as the first and only element of the 'dictionaryArray' attribute of the DataTransferObject. This is shown in the following code extract.

Figure 17–2 Code Extract

```

1 com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO1= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
2 nameValuePairDTO1.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.CustomValue1");
3 nameValuePairDTO1.setValue("Y");
4 com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO2= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
5 nameValuePairDTO2.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.CustomValue2");
6 nameValuePairDTO2.setValue("Y");
7 com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[] nameValuePairDTOArray= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[2];
8 nameValuePairDTOArray[0]=nameValuePairDTO1;
9 nameValuePairDTOArray[1]=nameValuePairDTO2;
10 com.ofss.fc.framework.domain.common.dto.Dictionary dictionary= new com.ofss.fc.framework.domain.common.dto.Dictionary();
11 dictionary.setNameValuePairDTOArray(nameValuePairDTOArray);
12 com.ofss.fc.framework.domain.common.dto.Dictionary[] dictionaryArray = new com.ofss.fc.framework.domain.common.dto.Dictionary[1];
13 dictionaryArray[0]=dictionary;

```

17.2 Customized Domain Object DTO Interceptor in UI Layer

All DTO classes should extend `DomainObjectDTO` in case maintenance fields are required.

For example, 'MessageDataAttributeDTO' Class which extends 'DomainObjectDTO' is used to transfer data between an external system and the application service and persist data for Domain Object 'MessageDataAttribute'.

'CustomizedMessageDataAttribute' is a subclass of this Customizable Maintenance Domain Object called 'MessageDataAttribute' which is extended by the partners or consulting teams to include and subsequently persist extra attributes along with those of 'MessageDataAttribute'.

This information can be mapped as input and output to the application services with the help of dictionaryArray attribute of MessageDataAttributeDTO inherited from DataTransferObject.

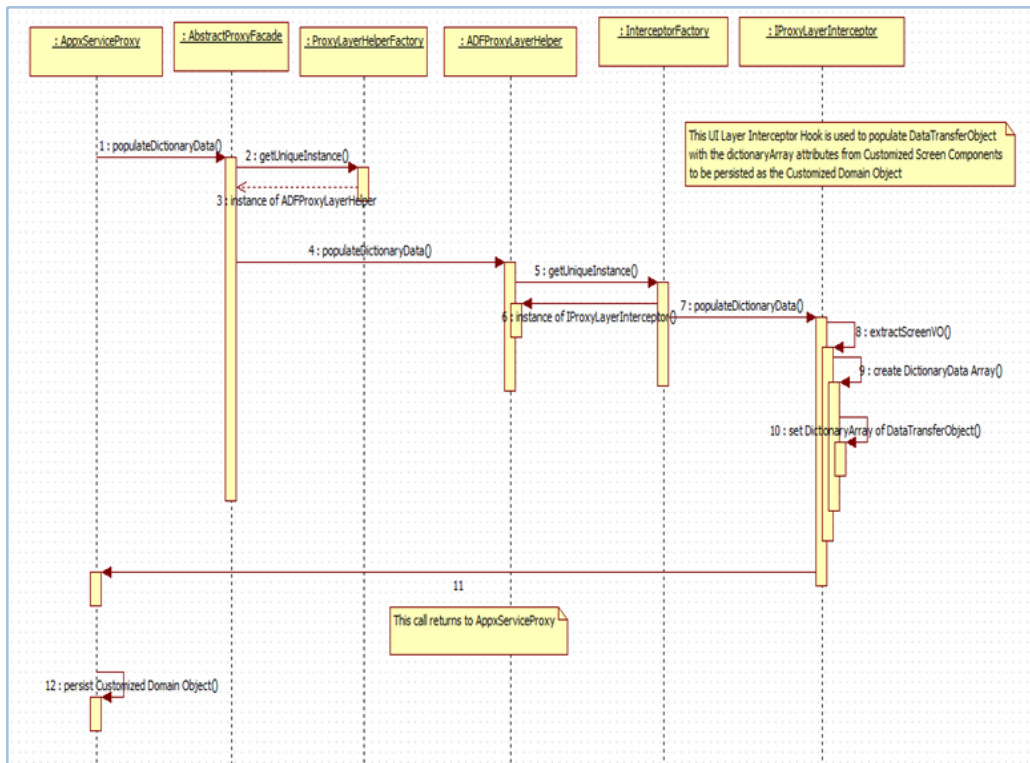
17.2.1 Interceptor Hook to Persist Customized Domain Object Attributes

This UI Layer Interceptor Hook is used during Create or Update mode to populate DataTransferObject with the dictionaryArray attributes from customized Screen Components to be persisted as the Customized Domain Object.

In the UI Layer, the ApplicationServiceProxyFacade is used to send the DataTransferObject on to the Host to be persisted. Before it does so, it uses the InterceptorFactory to instantiate the appropriate IProxyLayerInterceptor defined in the DictionaryInterceptor.properties corresponding to the key for this application service or task code. Thereafter it invokes the 'populateDictionaryArray' method of this IProxyLayerInterceptor to populate DataTransferObject with the dictionaryArray attributes from customized Screen Components. Thereafter, it sends the entire DataTransferObject on to the Host for persistence as the Customized Domain Object.

The following figure provides the details of Interceptor Hook to populate and persist Customized Domain Object.

Figure 17–3 Interceptor Hook to Persist Customized Domain Object



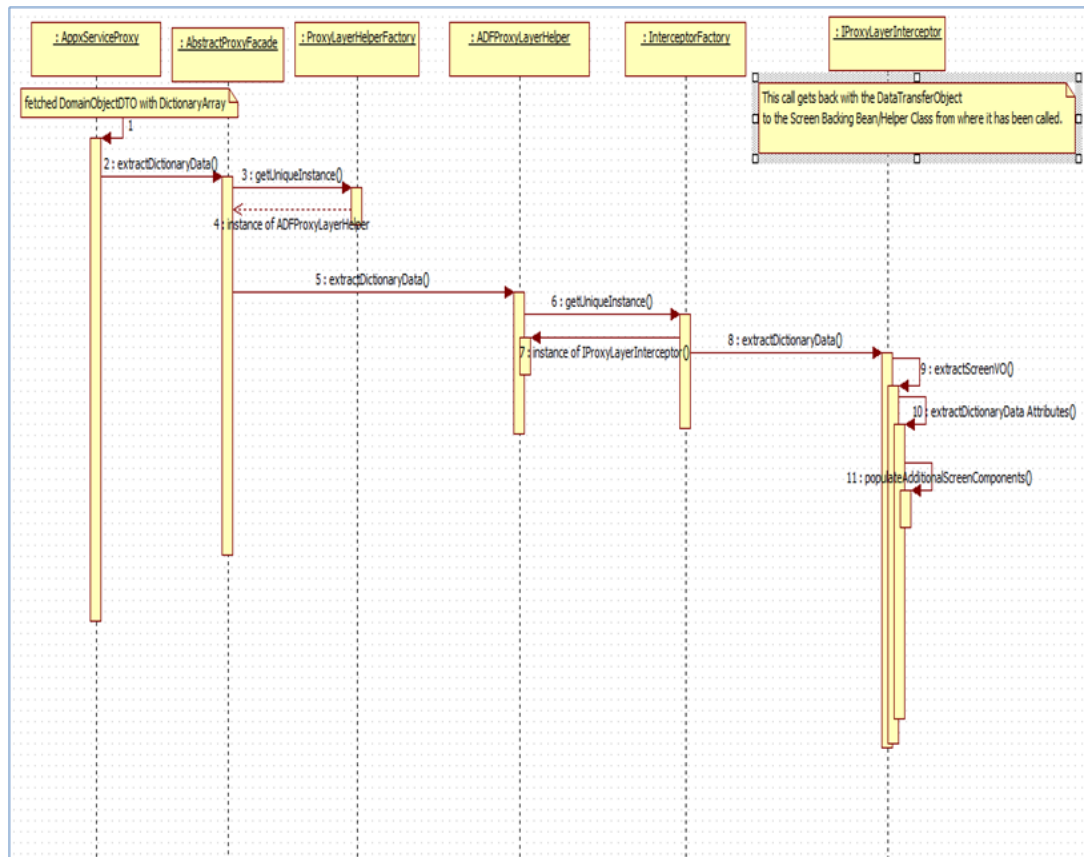
17.2.2 Interceptor Hook to Fetch Customized Domain Object Attributes

This UI Layer Interceptor Hook is used during read mode to extract the dictionaryArray attributes from the DataTransferObject and populate the customized Screen Components with the help of the screen view object.

In the UI Layer, the ApplicationServiceProxyFacade is used to receive the DataTransferObject from the Host. After it does so, it uses the InterceptorFactory to instantiate the appropriate IProxyLayerInterceptor defined in the DictionaryInterceptor.properties corresponding to the key for this application service or task code. Thereafter, it invokes the 'extractDictionaryArray' method of this IProxyLayerInterceptor to extract the dictionaryArray attributes from the DataTransferObject and populate the customized Screen Components with the help of the screen view object. Thereafter, it returns the entire DataTransferObject on to the Screen Backing Bean or Helper Class from where the proxy fetch call was invoked.

The following figure provides the details of Interceptor Hook to fetch Customized Domain Object and populate extra Screen Components.

Figure 17–4 Interceptor Hook to Fetch Customized Domain Object



InterceptorFactory instantiates the appropriate IProxyLayerInterceptor defined in the DictionaryInterceptor.properties corresponding to the key.

Examples of such key value pair is:-

```
com.ofss.fc.appx.ep.service.dispatch.message.service.client.proxy.MessageTemplateApplicationServiceProxyFacade=com.ofss.fc.ui.taskflows.ep.messageTemplateUI.view.interceptor.MessageTemplateUIInterceptor
```

```
com.ofss.fc.appx.party.service.contact.service.client.proxy.ContactPointApplicationServiceProxyFacade=com.ofss.fc.ui.view.party.contactPoint.interceptor.ContactPointUIInterceptor
```

17.3 Dictionary Data Transfer from UI to Host

The section describes the dictionary data transfer from UI to Host.

17.3.1 Customized Domain Object DTO Transfer from UI to Host

In UI server <ApplicationService>JSONClient constructs the JSON Object for <DomainObjectDTO> which includes the dictionaryArray of the DataTransferObject.

For example, in UI server MessageTemplateApplicationServiceJSONClient constructs the JSON Object for MessageTemplateDTO which includes MessageTemplateAttributeDTO and the dictionaryArray of DataTransferObject as shown below.

Figure 17–5 JSONClient constructs the JSON Object

```

1 private JSONArray serializeMessageDataAttributeDTOArray(com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] objInputVals)
2     throws JSONException {
3
4     JSONArray lMessageDataAttributeDTOJSONArray = null;
5     if (objInputVals != null) {
6         lMessageDataAttributeDTOJSONArray = new JSONArray();
7         for (int inumobjInputValArray = 0; inumobjInputValArray < objInputVals.length; inumobjInputValArray++) {
8             JSONObject lObjInputValJSONObject = null;
9             com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO objObjInputVal = objInputVals[inumobjInputValArray];
10            if (objObjInputVal != null) {
11                lObjInputValJSONObject = new JSONObject();
12                lObjInputValJSONObject.put("type", (objObjInputVal.getClass().getName()));
13                com.ofss.fc.framework.domain.common.dto.Dictionary[] arr_objObjInputVal_dictionaryArray = objObjInputVal.getDictionaryArray();
14                JSONArray l1ObjInputValJSONObjectdictionaryArray = serializeDictionaryArray(arr_objObjInputVal_dictionaryArray);
15                try {
16                    lObjInputValJSONObject.put("DictionaryArray", l1ObjInputValJSONObjectdictionaryArray);
17                } catch (Throwable t) {
18                    ErrorManager.overrideError(null, t);
19                }
20            }
21            lMessageDataAttributeDTOJSONArray.put(lObjInputValJSONObject);
22        }
23    }
24    return lMessageDataAttributeDTOJSONArray;
25 }
26

```

<ApplicationService>JSONClient constructs the JSON Object for <DomainObjectDTO> which includes the dictionaryArray of the DataTransferObject

The above process uses AbstractJSONBindingStub class' serializeDictionaryArray to include 'genericName' and 'value' attributes of NameValuePairDTOArray which was inside dictionaryArray attribute of MessageTemplateAttributeDTO.

Figure 17–6 SerializedDictionaryArray to include GenericName and Value attributes

```

89
90 /**
91  * 1D Array of Dictionary representing the set of Union Subclasses of the corresponding Customized
92  * AbstractDomainObject is serialized into a JSONArray
93  *
94  * @param objInputVal
95  * @return
96  * @throws JSONException
97  */
98 protected JSONArray serializeDictionaryArray(Dictionary[] extendedDomainObjectDTOArray) throws JSONException {
99
100     JSONArray lDictionaryJSONArray = new JSONArray();
101     if (extendedDomainObjectDTOArray != null) {
102         for (int i = 0; i < extendedDomainObjectDTOArray.length; i++) {
103             JSONArray lCustomizedNameValuePairDTOJSONArray = new JSONArray();
104             NameValuePairDTO[] nameValuePairDTOArray = extendedDomainObjectDTOArray[i].getNameValuePairDTOArray();
105             for (int j = 0; j < nameValuePairDTOArray.length; j++) {
106                 JSONObject lCustomizedNameValuePairDTOJSONObject = new JSONObject();
107                 lCustomizedNameValuePairDTOJSONObject.putOpt("FactName", nameValuePairDTOArray[j].getGenericName());
108                 lCustomizedNameValuePairDTOJSONObject.putOpt("FactValue", nameValuePairDTOArray[j].getValue());
109                 lCustomizedNameValuePairDTOJSONArray.put(lCustomizedNameValuePairDTOJSONObject);
110             }
111             lDictionaryJSONArray.put(lCustomizedNameValuePairDTOJSONArray);
112         }
113     }
114     return lDictionaryJSONArray;
115 }
116

```

AbstractJSONBindingStub class's serializeDictionaryArray to include "genericName" and "value" attributes of NameValuePairDTOArray

In the Host Server <ApplicationService>JSONFacade extracts the 'DictionaryArray' attribute of JSON Object and sets it as <DomainObjectDTO>'s dictionaryArray attribute.

For example, in the Host Server, MessageTemplateApplicationServiceJSONFacade extracts the 'DictionaryArray' attribute of JSON Object and sets it as MessageDataAttributeDTO's dictionaryArray attribute.

Figure 17–7 Host Server JSONFacade extracts the attribute of JSON Object

```

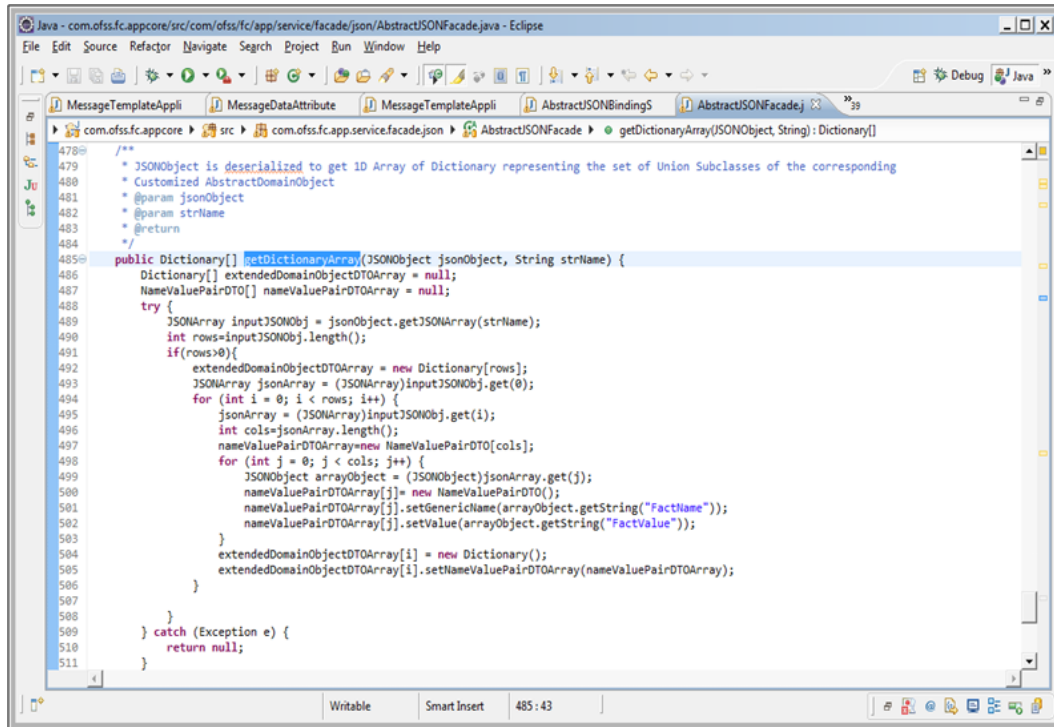
1 private com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] getMessageDataAttributeDTOArray(JSONObject jsonObject, String strName
2 throws JSONException {
3
4     com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] lMessageDataAttributeDTOs = null;
5     JSONArray lMessageDataAttributeDTOJSONArray = jsonObject.optJSONArray(strName);
6     if (lMessageDataAttributeDTOJSONArray != null) {
7         int numobjInputValArray = lMessageDataAttributeDTOJSONArray.length();
8         lMessageDataAttributeDTOs = new com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[numobjInputValArray];
9         JSONObject lobjInputValJSONObject = null;
10        for (int iobjInputVal = 0; iobjInputVal < numobjInputValArray; iobjInputVal++) {
11            com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO lstrName_MessageDataAttributeDTO = null;
12            lobjInputValJSONObject = lMessageDataAttributeDTOJSONArray.optJSONObject(iobjInputVal);
13            if (lobjInputValJSONObject != null) {
14                lstrName_MessageDataAttributeDTO = new com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO();
15                String strMessageDataAttributeDIOType = lobjInputValJSONObject.optString("_type");
16                com.ofss.fc.framework.domain.common.dto.Dictionary[] llstrName_MessageDataAttributeDTO_dictionaryArrays
17                    = getDictionaryArray(lobjInputValJSONObject, "DictionaryArray");
18                lstrName_MessageDataAttributeDTO.setDictionaryArray(llstrName_MessageDataAttributeDTO_dictionaryArrays);
19            }
20            lMessageDataAttributeDTOs[iobjInputVal] = lstrName_MessageDataAttributeDTO;
21        }
22    }
23    return lMessageDataAttributeDTOs;
24 }
25

```

In the Host Server <ApplicationService>JSONFacade extracts the "DictionaryArray" attribute of JSON Object and sets it as <DomainObjectDTO>'s dictionaryArray attribute.

The above process uses AbstractJSONFacade's getDictionaryArray method that unmarshalls the 'genericName' and 'value' from JSON Object to get the dictionaryArray attribute.

Figure 17–8 AbstractJSONFacade's getDictionaryArray method



```

478  /**
479   * JSONObject is deserialized to get ID Array of Dictionary representing the set of Union SubClasses of the corresponding
480   * Customized AbstractDomainObject
481   * @param jsonObject
482   * @param strName
483   * @return
484   */
485  public Dictionary[] getDictionaryArray(JSONObject jsonObject, String strName) {
486      Dictionary[] extendedDomainObjectDTOArray = null;
487      NameValuePairDTO[] nameValuePairDTOArray = null;
488      try {
489          JSONArray inputJSONObj = jsonObject.getJSONArray(strName);
490          int rows=inputJSONObj.length();
491          if(rows>0){
492              extendedDomainObjectDTOArray = new Dictionary[rows];
493              JSONArray jsonArray = (JSONArray)inputJSONObj.get(0);
494              for (int i = 0; i < rows; i++) {
495                  jsonArray = (JSONArray)inputJSONObj.get(i);
496                  int cols=jsonArray.length();
497                  nameValuePairDTOArray=new NameValuePairDTO[cols];
498                  for (int j = 0; j < cols; j++) {
499                      JSONObject arrayObject = (JSONObject)jsonArray.get(j);
500                      nameValuePairDTOArray[j]= new NameValuePairDTO();
501                      nameValuePairDTOArray[j].setGenericName(arrayObject.getString("FactName"));
502                      nameValuePairDTOArray[j].setValue(arrayObject.getString("FactValue"));
503                  }
504                  extendedDomainObjectDTOArray[i] = new Dictionary();
505                  extendedDomainObjectDTOArray[i].setNameValuePairDTOArray(nameValuePairDTOArray);
506              }
507          }
508      } catch (Exception e) {
509          return null;
510      }
511  }

```

AbstractJSONFacade's getDictionaryArray method that unmarshalls the "genericName" and "value" from JSON Object to get the dictionaryArray attribute

17.3.2 Customized Domain Object DTO transfer from Host to UI

In the Host Server <ApplicationService>JSONFacade constructs the JSON Object for <DomainObjectDTO> and the dictionaryArray of DataTransferObject

MessageTemplateApplicationServiceJSONFacade's method serializeMessageDataAttributeDTOArray in Host Server constructs the JSON Object for MessageTemplateDTO which includes MessageTemplateAttributeDTO and the dictionaryArray of DataTransferObject as shown below:

Figure 17–9 Host Server JSONFacade constructs the JSON Object

```

1 private JSONArray serializeMessageDataAttributeDTOJSONArray(com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] objInputVals)
2     throws JSONException {
3
4     JSONArray lMessageDataAttributeDTOJSONArray = null;
5     if (objInputVals != null) {
6         lMessageDataAttributeDTOJSONArray = new JSONArray();
7         for (int inumobjInputValArray = 0; inumobjInputValArray < objInputVals.length; inumobjInputValArray++) {
8             JSONObject lObjInputValJSONObject = null;
9             com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO objObjInputVal = objInputVals[inumobjInputValArray];
10            if (objObjInputVal != null) {
11                lObjInputValJSONObject = new JSONObject();
12                lObjInputValJSONObject.put("_type", (objObjInputVal.getClass().getName()));
13
14                com.ofss.fc.framework.domain.common.dto.Dictionary[] arr_objObjInputVal_dictionaryArray = objObjInputVal.getDictionaryArray();
15                JSONArray l1ObjInputValJSONObjectdictionaryArray = serializeDictionaryArray(arr_objObjInputVal_dictionaryArray);
16                try {
17                    lObjInputValJSONObject.put("DictionaryArray", l1ObjInputValJSONObjectdictionaryArray);
18                } catch (Throwable t) {
19                    ErrorManager.overrideError(null, t);
20                }
21            }
22            lMessageDataAttributeDTOJSONArray.put(lObjInputValJSONObject);
23        }
24    }
25    return lMessageDataAttributeDTOJSONArray;
26 }
27

```

In the Host Server <ApplicationService>JSONFacade constructs the JSON Object for <DomainObjectDTO> and the dictionaryArray of DataTransferObject

The above process uses AbstractJSONFacade's serializeDictionaryArray to include 'genericName' and 'value' attributes of NameValuePairDTOArray which was inside dictionaryArray attribute of MessageTemplateAttributeDTO.

Figure 17–10 AbstractJSONFacade's serializeDictionaryArray to include Generic Name and Value attributes

```

508     }
509     } catch (Exception e) {
510         return null;
511     }
512     return extendedDomainObjectDTOArray;
513 }
514 /**
515  * ID Array of Dictionary representing the set of Union Subclasses of the corresponding Customized AbstractDomainObject is
516  * serialized into a JSONArray
517  * @param objInputVal
518  * @return
519  * @throws JSONException
520  */
521 public JSONArray serializeDictionaryArray(Dictionary[] extendedDomainObjectDTOArray) throws JSONException {
522     JSONArray lDictionaryJSONArray = new JSONArray();
523     if(extendedDomainObjectDTOArray!=null){
524         for (int i = 0; i < extendedDomainObjectDTOArray.length; i++) {
525             JSONArray lCustomizedNameValuePairDTOJSONArray = new JSONArray();
526             NameValuePairDTO[] nameValuePairDTOArray = extendedDomainObjectDTOArray[i].getNameValuePairDTOArray();
527             for (int j = 0; j < nameValuePairDTOArray.length; j++) {
528                 JSONObject lCustomizedNameValuePairDTOJSONObject = new JSONObject();
529                 lCustomizedNameValuePairDTOJSONObject.putOpt("FactName", nameValuePairDTOArray[j].getGenericName());
530                 lCustomizedNameValuePairDTOJSONObject.putOpt("FactValue", nameValuePairDTOArray[j].getValue());
531                 lCustomizedNameValuePairDTOJSONObject.put(lCustomizedNameValuePairDTOJSONObject);
532             }
533             lDictionaryJSONArray.put(lCustomizedNameValuePairDTOJSONArray);
534         }
535     }
536     return lDictionaryJSONArray;
537 }
538 }
539 }
540 }
541 }

```

AbstractJSONFacade's serializeDictionaryArray to include "genericName" and "value" attributes of NameValuePairDTOArray

In the UI Server, <ApplicationService>JSONClient extracts the 'DictionaryArray' attribute of JSON Object and sets it as <DomainObjectDTO>DTO's dictionaryArray attribute.

In the UI Server, MessageTemplateApplicationServiceJSONClient extracts the 'DictionaryArray' attribute of JSON Object and sets it as MessageDataAttributeDTO's dictionaryArray attribute.

Figure 17–11 UI Server JSONClient extracts the DictionaryArray attribute

```

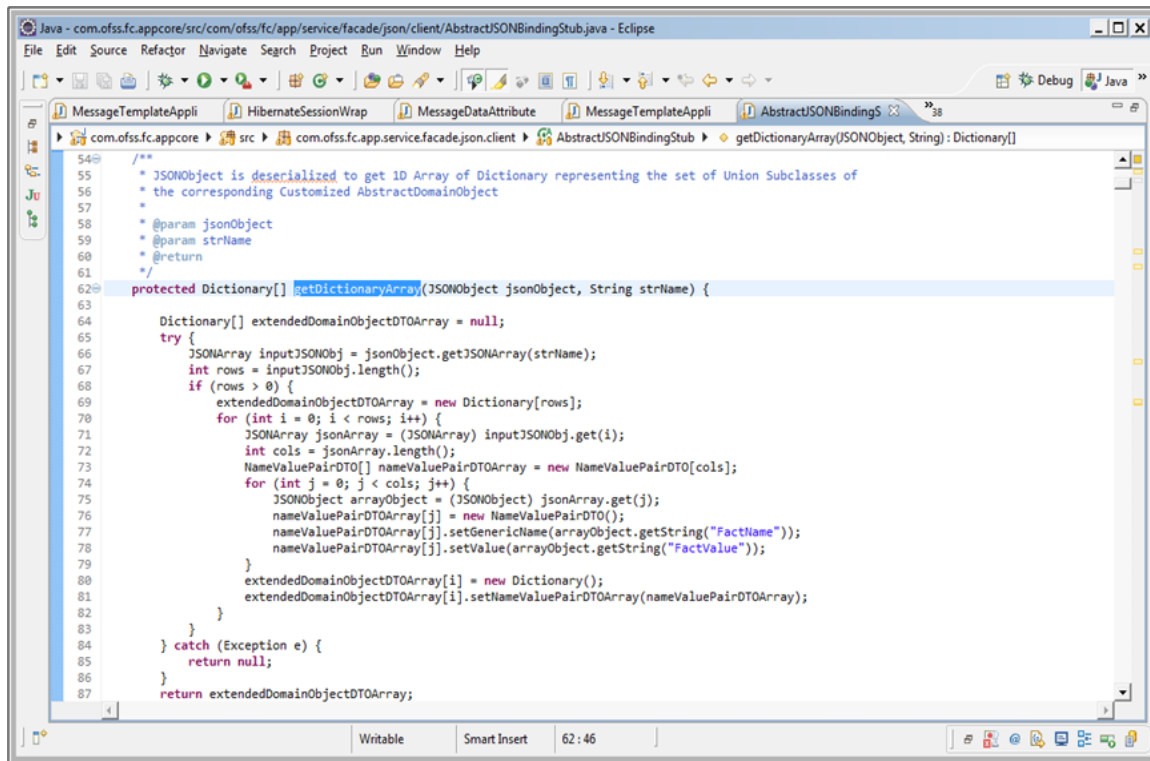
1 private com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] getMessageDataAttributeDTOArray(JSONObject jsonObject, String strName
2     throws JSONException {
3
4     com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[] lMessageDataAttributeDTOs = null;
5     JSONArray lMessageDataAttributeDTOJSONArray = jsonObject.optJSONArray(strName);
6     if (lMessageDataAttributeDTOJSONArray != null) {
7         int numobjInputValArray = lMessageDataAttributeDTOJSONArray.length();
8         lMessageDataAttributeDTOs = new com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO[numobjInputValArray];
9         JSONObject lobjInputValJSONObject = null;
10        for (int iobjInputVal = 0; iobjInputVal < numobjInputValArray; iobjInputVal++) {
11            com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO lstrName_MessageDataAttributeDTO = null;
12            lobjInputValJSONObject = lMessageDataAttributeDTOJSONArray.optJSONObject(iobjInputVal);
13            if (lobjInputValJSONObject != null) {
14                lstrName_MessageDataAttributeDTO = new com.ofss.fc.app.ep.dto.dispatch.message.MessageDataAttributeDTO();
15                String strMessageDataAttributeDTOType = lobjInputValJSONObject.optString("_type");
16                com.ofss.fc.framework.domain.common.dto.Dictionary[] llstrName_MessageDataAttributeDTO_dictionaryArrays
17                    = getDictionaryArray(lobjInputValJSONObject, "DictionaryArray");
18                lstrName_MessageDataAttributeDTO.setDictionaryArray(llstrName_MessageDataAttributeDTO_dictionaryArrays);
19            }
20            lMessageDataAttributeDTOs[iobjInputVal] = lstrName_MessageDataAttributeDTO;
21        }
22    }
23    return lMessageDataAttributeDTOs;
24 }
25

```

In the UI Server, <ApplicationService>JSONClient extracts the "DictionaryArray" attribute of JSON Object and sets it as <DomainObjectDTO>DTO's dictionaryArray attribute

The above process uses AbstractJSONBindingStub's getDictionaryArray method that unmarshalls the 'genericName' and 'value' from JSON Object to get the dictionaryArray attribute.

Figure 17–12 AbstractJSONBindingStub's getDictionaryArray method



AbstractJSONBindingStub's getDictionaryArray method that unmarshalls the "genericName" and "value" from JSON Object

The provision of marshalling and un-marshalling of 'dictionaryArray' attribute of all DataTransferObjects has been included in the JSON layer for all application services.

17.4 Translating Dictionary Data into Custom Domain Object

This section describes the details of translating dictionary data into custom domain object.

17.4.1 Instantiation and Persistence of Custom Domain Objects

If a method has an input parameter that is a DataTransferObject, the first line of the method in the assembler will be of the form:

```
(populateDataTransferObjectDTOMap('Fully Qualified Name of this DataTransferObject',
dataTransferObject);
```

This method is defined in AbstractAssembler.java which newly instantiates referenceDataTransferObjectDTOMap if required and populates the map with the above entry.

This map is used as a set of globally available DataTransferObject's which can be retrieved by invoking another method defined in AbstractAssembler.java which is of the form:

```
retrieveDataTransferObjectDTOMapElement('<Fully Qualified Name of
this DataTransferObject >');
```

Whenever any AbstractDomainObject is instantiated, the Customized AbstractDomainObject should be instantiated instead of the original AbstractDomainObject wherever applicable.

The `AbstractDomainObject` is instantiated with the help of the below code fragment:

```

IAbstractDomainObject domainObject=null;
try {
if (retrieveDataTransferObjectDTOMapElement("<Fully Qualified Name of DataTransferObject from Naming Convention
Rules >").getDictionaryArray() == null) {
domainObject = <Current Process Of Instantiation>;
} else {
domainObject=(IAbstractDomainObject)
getCustomizedDomainObject ( retrieveDataTransferObjectDTOMapElement
(
"<Fully Qualified Name of DataTransferObject from Naming Convention
Rules >"));
}

/***** In AbstractAssembler.java, we have defined the method
public IAbstractDomainObject getCustomizedDomainObject
(DataTransferObject dataTransferObjectDTO)

This method instantiates the CustomizedAbstractDomainObject based
on the value of the attribute "dictionaryArray" of the
DataTransferObject passed as the only parameter. The method also
populates this customized domain object with the extra attribute
values also from the "dictionaryArray" attribute and finally
returns this instance of the Customized Domain Object.
*****/
}
} catch (Exception e) {
domainObject = <Current Process Of Instantiation>;
}

```

17.4.2 Fetching of Customized Domain Objects

If a method has an input parameter that is an `IAbstractDomainObject`, the first line of the method in the assembler will be of the form:

```

populateAbstractDomainObjectMap("<Fully_Qualified_Name_
IAbstractDomainObject>", abstractDomainObject);

```

This method is defined in `AbstractAssembler.java` which newly instantiates `referenceAbstractDomainObjectMap` if required and populates the map with the above entry.

This map is used as a set of globally available `IAbstractDomainObject`'s which can be retrieved by invoking another method defined in `AbstractAssembler.java` which is of the form:

```

retrieveDataTransferObjectDTOMapElement("<Fully_Qualified_Name_
IAbstractDomainObject>");

```

Whenever any `DataTransferObject` is instantiated, we populate its 'dictionaryArray' attribute immediately after it's instantiation.

In `AbstractAssembler.java`, we have defined the method as

```

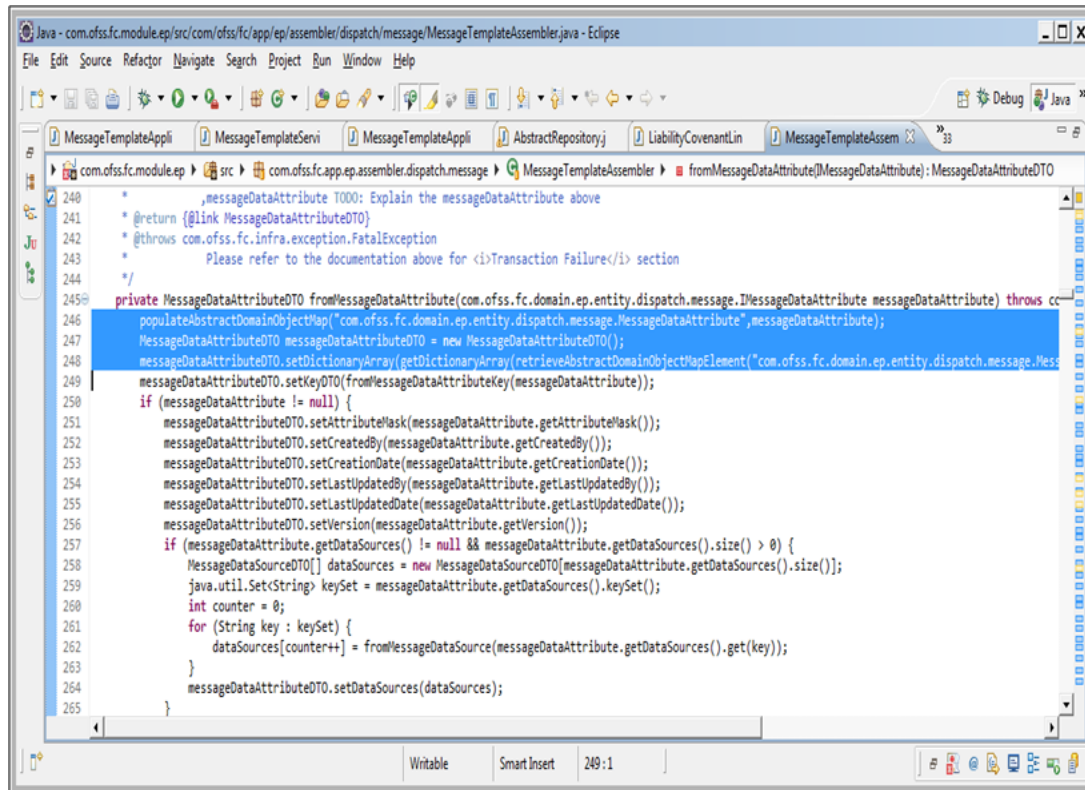
public Dictionary[] getDictionaryArray(IAbstractDomainObject obj)

```

This method creates and returns a dictionary array from the `IAbstractDomainObject` passed to it as input parameter.

Example of final piece of code:

Figure 17–13 Instantiation of DataTransferObjects



17.4.3 Defining of Customized Domain Objects

When we are viewing the customized attributes on the screen, we need to fetch the Customized Abstract Domain Object data into the Domain Object DTO. This is why the customized attributes in the Customized Domain Object have to be populated in the dictionary array of the Domain Object DTO.

This is done in the AbstractAssembler which returns the dictionary array of the Domain Object DTO based on the Abstract Domain Object passed to it, through a method called "getDictionaryArray". To achieve this, the AbstractAssembler firstly needs to understand which is a customized domain object.

In preferences.xml we have defined the following:

```

<Preference name="CustomizedAbstractDomainObjectConfig"
PreferencesProvider="com.ofss.fc.infra.config.impl.DBBasedProperty
Provider"
parent="jdbcpreference"
propertyFileName="select prop_id, prop_value from flx_fw_config_
all_b where category_id = 'CustomizedAbstractDomainObjectConfig'"
syncTimeInterval="600000" />

```

We have to insert a record in table `flx_fw_config_all_b` to identify a Customized Domain Object in the following manner.


```
INSERT INTO FLX_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.fc.domain.ep.entity.action.ActivityEventAction',
'CustomizedAbstractDomainObjectConfig',
'com.ofss.fc.domain.ep.entity.action.CustomizedActivityEventActio
n', 'y', '',
'Customized object of
com.ofss.fc.domain.ep.entity.action.ActivityEventAction',
'ofssuser',
'09-SEP-14 05.53.56.000000 PM', 'ofssuser', '09-SEP-14
05.53.56.000000 PM', 'A', 1);
```

The AbstractAssembler identifies a customized domain object by deciphering the above information.

So every Customized Domain Object has to be defined in flx_fw_config_all_b with category_id = 'CustomizedAbstractDomainObjectConfig'.

Only if such a definition exists, the abstract domain object passed is identified to be a customized domain object and the corresponding Domain Object DTO is provided with its dictionary array.

However, if the abstract domain object passed is not identified to be a customized domain object, the corresponding Domain Object DTO is provided with a dictionary array which has null value.

17.5 Customized Domain Object ORM Configuration

This section describes the details of customized domain object ORM configuration.

17.5.1 Case 1 - Non-Inheritance based mapping

Non-inheritance based mapping refers to those domain objects that are not mapped as a Subclass or Union-Subclass or Joined-Subclass. Let us take the example of the class MessageDataAttribute. The fully qualified class name is 'com.ofss.fc.domain.ep.entity.dispatch.message.MessageDataAttribute'. This class has been mapped in ep.messagesetemplate.orm.xml.

Adding Discriminator column mapping in existing ORM file

Add the discriminator as:- <discriminator column=" DOMAIN_OBJECT_EXTN" type="string"/>

For the purpose of identifying the extended domain object in the corresponding table, add a 'discriminator column' in the corresponding table and update the ORM file. The name of the discriminator column used is DOMAIN_OBJECT_EXTN and the default discriminator value defined is 'CZ'

So any normal Create or Update operation will have a value 'CZ' for DOMAIN_OBJECT_EXTN column.

Figure 17–14 Adding Discriminator Column Mapping in Existing ORM file

```

<entity class="com.ofss.fc.domain.ep.entity.dispatch.message.MessageDataAttribute">
  <discriminator-value>CZ</discriminator-value>
  <table name="FLX_EP_MSG_ATTR_B"/>
  <attributes>
    <embedded-id attribute-type="com.ofss.fc.domain.ep.entity.dispatch.message.MessageDataAttributeKey" name="messageDataAttributeKey">
      <attribute-override name="attributeId">
        <column name="cod_attr_id"/>
      </attribute-override>
      <attribute-override name="messageTemplateId">
        <column name="cod_mess_tmpl_id"/>
      </attribute-override>
    </embedded-id>
    <basic attribute-type="java.lang.String" name="attributeMask">
      <column name="ATTR_MASK"/>
    </basic>
  </attributes>
  <discriminator-column discriminator-type="STRING" name="DOMAIN_OBJECT_EXTN"/>
</entity>

```

A new ORM file mapping to Customized Domain Object is added

The following figure explains adding a new ORM file mapping to Customized Domain Object.

Figure 17–15 ORM File Mapping to Customized Domain Object

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <entity class="com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageDataAttribute"
    parent="com.ofss.fc.domain.ep.entity.dispatch.message.MessageDataAttribute">
    <discriminator-value>FCMA</discriminator-value>
    <attributes>
      <basic attribute-type="java.lang.String" name="customValue1">
        <column name="custom_value1"/>
      </basic>
      <basic attribute-type="java.lang.String" name="customValue2">
        <column name="custom_value2"/>
      </basic>
    </attributes>
  </entity>
</entity-mappings>

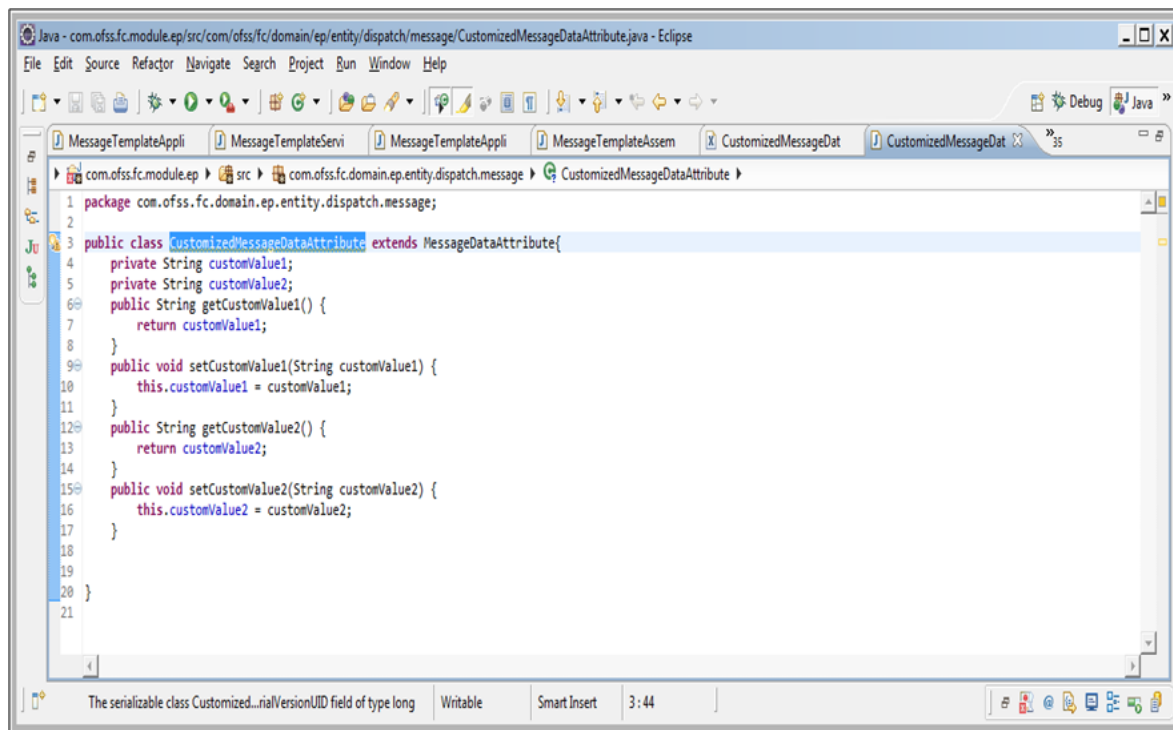
```

For example a new file CustomizedMessageDataAttribute.orm.xml is introduced to include the extra attributes added by consulting or any other third party along with the discriminator value. This file will map to the new customized domain object and will be extending the existing Abstract Domain Object.

Adding new Java File corresponding to the Customized Domain Object

The following figure explains adding new Java file corresponding to the Customized Domain Object.

Figure 17–16 Adding New Java File to the Customized Domain Object

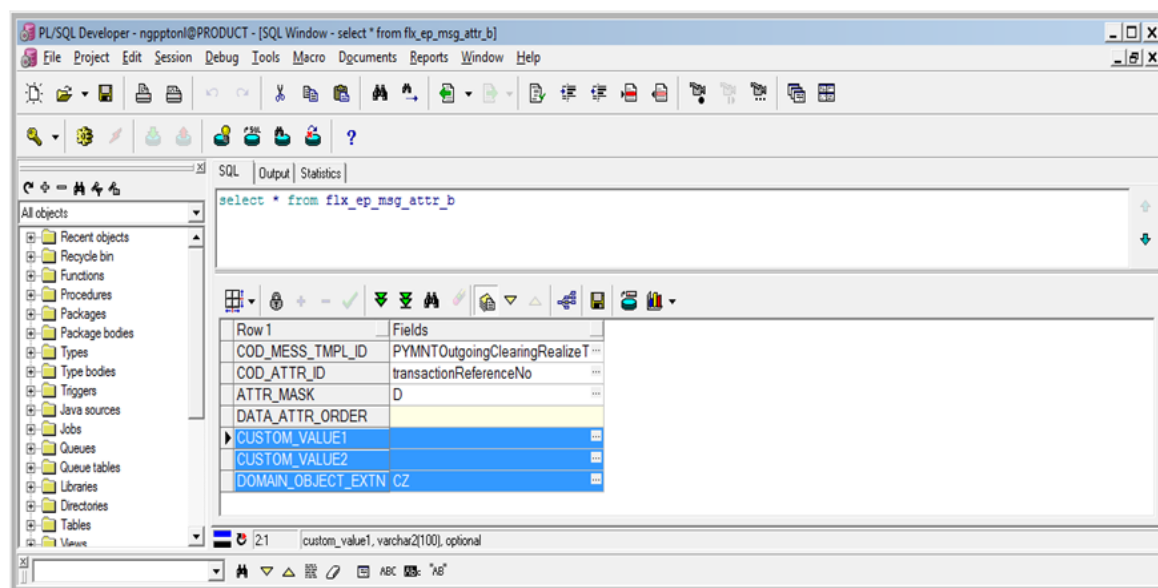


A Java File is added corresponding to the existing Abstract Domain Object. This will be extending the Abstract Domain Object that we are extending.

Adding extra columns along with the discriminator column to the domain object table

The following figure explains adding a new Java file corresponding to the Customized Domain Object.

Figure 17–17 Adding Extra Columns along with the Discriminator Column



The extra columns along with the discriminator column have to be added to the domain object table of this domain object.

In case of Creation or Updation of 'CustomizedMessageDataAttribute' instead of 'MessageDataAttribute' the new discriminator column 'DOMAIN_OBJECT_EXTN' has the value of 'FCMA' instead of 'CZ' and an additional value in columns 'CUSTOM_VALUE1' and 'CUSTOM_VALUE2' in table FLX_EP_MSG_ATTR_B.

In case of Creation or Updation of 'MessageDataAttribute' the new discriminator column 'DOMAIN_OBJECT_EXTN' has the value of 'CZ' and NULL values in columns 'CUSTOM_VALUE1' and 'CUSTOM_VALUE2' in table FLX_EP_MSG_ATTR_B.

17.5.2 Case 2 - Mapped as ORM Subclass

The maintenance domain objects which are mapped as a Subclass already have an existing discriminator. For the purpose of identifying the extended domain object in the same table, we shall be using the existing discriminator.

Let us take the example of 'com.ofss.fc.domain.party.entity.contact.Cellular'. This is mapped as a subclass in ContactPoint.orm.xml.

A new ORM file mapping to Customized Domain Object is added

The following figure explains adding a new ORM file mapping to Customized Domain Object.

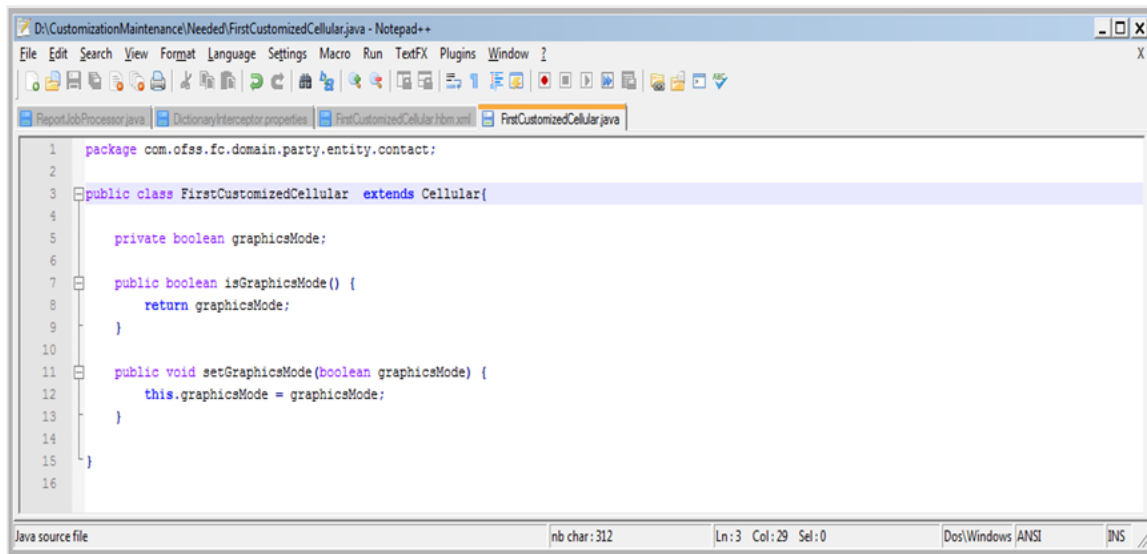
Figure 17–18 Adding a New ORM File Mapping to Customized Domain Object

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm" xmlns:xsi="http://www.w3.org/
  <entity class="com.ofss.fc.domain.party.entity.contact.FirstCustomizedCellular"
    parent="com.ofss.fc.domain.party.entity.contact.Cellular">
    <discriminator-value>FCLR</discriminator-value>
    <attributes>
      <basic attribute-type="java.lang.String" name="graphicsMode">
        <column name="graphics_Mode"/>
      </basic>
    </attributes>
  </entity>
</entity-mappings>
```

A new file FirstCustomizedCellular.orm.xml is introduced to include the extra attributes added by consulting or any other third party along with the discriminator value 'FCLR'. This file will map to the new customized domain object 'com.ofss.fc.domain.party.entity.contact.FirstCustomizedCellular' and will be extending the existing Abstract Domain Object which is 'com.ofss.fc.domain.party.entity.contact.Cellular'.

Adding new Java File corresponding to the Customized Domain Object

The following figure explains adding a new Java File corresponding to the Customized Domain Object.

Figure 17–19 Adding New Java File to Customized Domain Object

A Java File 'com.ofss.fc.domain.party.entity.contact.FirstCustomizedCellular' is added corresponding to the existing Abstract Domain Object. This will be extending the Abstract Domain Object that we are extending.

Adding Extra Columns to the Domain Object Table

The extra columns have to be added to the domain object table of this domain object.

In this case GRAPHICS_MODE column is added to FLX_PI_CONTACT_POINT table.

So in case of Creation or Updation of 'FirstCustomizedCellular' instead of 'Cellular' the existing discriminator column 'CONTACT_POINT_TYPE' has the value of 'FCLR' instead of 'CLR' and an additional value in column 'GRAPHICS_MODE' in table FLX_PI_CONTACT_POINT.

And in case of Creation or Updation of 'Cellular' the existing discriminator column 'CONTACT_POINT_TYPE' has the value of 'CLR' and NULL values in column 'GRAPHICS_MODE' in table FLX_PI_CONTACT_POINT.

17.5.3 Case 3 - Mapped as ORM Union-Subclass or Joined-Subclass

Let us take the example of 'com.ofss.fc.domain.lcm.entity.limits.facility.proposedFacility.ProposedFacility'. This class has been mapped in Facility.orm.xml as a union subclass.

Use the customized entity

'com.ofss.fc.cz.nab.domain.lcm.entity.limits.facility.proposedFacility.CustomizedProposedFacility' for the purpose of extensibility of this domain object.

Adding Discriminator in ORM file where base class has been mapped is not required

The existing Facility.orm.xml file which contains the mapping for "com.ofss.fc.domain.lcm.entity.limits.facility.proposedFacility.ProposedFacility" is not required to be altered.

A new ORM file mapping to Customized Domain Object is added

The following figure explains adding a new ORM file mapped to new Customized Domain Object.

Figure 17–20 New ORM File Mapping

```

CustomizedProposedFacility.orm.xml
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.eclipse.org/eclipselink/xsds/persistence/orm http://www.eclipse.org/eclipselink/xsds/persistence/orm.xsd">
  <entity class="com.ofss.fc.domain.lcm.entity.limits.facility.proposedFacility.CustomizedProposedFacility"
    parent="com.ofss.fc.domain.lcm.entity.limits.facility.proposedFacility.ProposedFacility">
    <table name="CZ_NAB_LM_PROPOSED_FACILITY"/>
    <attributes>
      <basic attribute-type="java.lang.String" name="associatedConsumerLending">
        <column name="Associated_Consumer_Lending"/>
      </basic>
      <basic attribute-type="java.lang.String" name="associatedBusinessLending">
        <column name="Associated_Business_Lending"/>
      </basic>
    </attributes>
  </entity>
</entity-mappings>

```

For example, a new file CustomizedProposedFacility.orm.xml is introduced to include the extra attributes added by consulting or any other third party. This file will map to the new customized domain object and will be extending the existing Abstract Domain Object.

Adding new Java File corresponding to the Customized Domain Object

Figure 17–21 Adding New Java File

```

package com.ofss.fc.cz.nab.domain.lcm.entity.limits.facility.proposedFacility;

import com.ofss.fc.domain.lcm.entity.limits.facility.proposedFacility.ProposedFacility;

public class CustomizedProposedFacility extends ProposedFacility {

    /**
     * Default serial version UID.
     */
    private static final long serialVersionUID = 1L;
    private String associatedConsumerLending;
    private String associatedBusinessLending;
    private String feeNegotiateApprovalCode;

    public String getFeeNegotiateApprovalCode() {

        return feeNegotiateApprovalCode;
    }

    public void setFeeNegotiateApprovalCode(String feeNegotiateApprovalCode) {

        this.feeNegotiateApprovalCode = feeNegotiateApprovalCode;
    }

    public String getAssociatedConsumerLending() {

        return associatedConsumerLending;
    }

    public void setAssociatedConsumerLending(String associatedConsumerLending) {

        this.associatedConsumerLending = associatedConsumerLending;
    }

    public String getAssociatedBusinessLending() {

        return associatedBusinessLending;
    }
}

```

A Java File 'CustomizedProposedFacility.java' is added. This extends the Abstract Domain Object that we are extending.

Create a new table CZ_NAB_LM_PROPOSED_FACILITY similar to the Domain Object Table

We are extending that is, FLX_LM_PROPOSED_FACILITY_B and add the extra columns to the new table.

Figure 17–22 Create a New Table CZ_NAB_LM_PROPOSED_FACILITY

```

1 create table CZ_NAB_LM_PROPOSED_FACILITY as
2 select * from FLX_LM_PROPOSED_FACILITY_B where 1=2;
3
4
5 ALTER TABLE CZ_NAB_LM_PROPOSED_FACILITY ADD ASSOCIATED_CONSUMER_LENDING VARCHAR2 (20)
6 /
7 ALTER TABLE CZ_NAB_LM_PROPOSED_FACILITY ADD ASSOCIATED_BUSINESS_LENDING VARCHAR2 (20)
8 /
9
10 ALTER TABLE CZ_NAB_LM_PROPOSED_FACILITY add FEE_NEGOTIATE_APPROVAL_CODE VARCHAR2 (50)
11 /

```

Adding Customized JPQL Queries whenever the Domain Object is Referred

The following file has the attribute 'CustomizedORMQueriesConfig' to fire the Customized JPQL if required: Preferences.xml.

The attribute is as follows:

```

<Preference name="CustomizedORMQueriesConfig"

PreferencesProvider="com.ofss.fc.infra.config.impl.JavaConstantsCo
nfigProvider"
overriddenBy="CustomizedORMQueriesConfigOverride"
parent="jdbcpreference"
propertyFileName="com.ofss.fc.common.CustomizedORMQueriesConfig"
syncTimeInterval="600000" />

```

The following files have also been changed to fire the Customized JPQL if required.

com.ofss.fc.framework.domain@/com/ofss/fc/framework/repository/AbstractRepository.java

com.ofss.fc.common.jar@/src/com/ofss/fc/common/CustomizedORMQueriesConfig.java

The following file has the attribute 'CustomizedORMQueriesConfigOverride' to fire the Customized JPQL if required.

<lzn>/au/config/Preferences.xml

```

<Preference name="CustomizedORMQueriesConfigOverride"

PreferencesProvider="com.ofss.fc.infra.config.impl.JavaConstantsCo
nfigProvider"
parent=""
propertyFileName="com.ofss.fc.lz.au.common.CustomizedORMQueriesCon
fig"
syncTimeInterval="600000"/>

```

Therefore, `com.ofss.fc.lz.au.common.CustomizedORMQueriesConfig.java` file needs to have the old JPQL query name mapped to the customized query name for this domain object.

Similarly, extensibility of domain objects mapped as joined-subclass can also be done.

17.5.4 Case 4 - Mapped as ORM Component

This relates to only those component classes that implements `IAbstractDomainObject` and should be extensible.

The Java Class corresponding to this component class has to be extended and this new Java Class along with the additional attributes have to be mapped in the ORM file.

The corresponding additional columns have to be added in the domain object table in question.

17.6 Extensibility using Dictionary in Origination Application

In this section, the Application Form page (Fast path: OR097) of the Oracle Banking Platform is taken as an example.

17.6.1 `ICustomDataHandler's` as `DictionaryArray` Interceptor

The backing bean method of OR097 - Application Form

```
'com.ofss.fc.ui.taskflows.OriginationApplicationApplicationFormViewBackingApplicationFormMoveNext()' calls implementation of com.ofss.fc.ui.taskflows.OriginationApplicationCommonHandler.ICustomDataHandler.
```

Implementation of `com.ofss.fc.ui.taskflows.OriginationApplicationCommonHandler.ICustomDataHandler` can be configured in `OriginationConfiguration.properties`. Property name is **`customDataHandler`**

`ApplicationFormHelper.getSubmissionInputDTO()` will give the master DTO for the application form.

Figure 17–23 CustomDataHandler's as DictionaryArray Interceptor

```

public class DefaultDataHandler implements ICustomDataHandler {
    public DefaultDataHandler() {
        super();
    }

    public boolean onMovePrev(ApplicationFormHelper helper,
        ApplicationTrainStepType currentTrainStep,
        ApplicationTrainStepType nextTrainStep) {
        return false;
    }

    public boolean onMoveNext(ApplicationFormHelper helper,
        ApplicationTrainStepType currentTrainStep,
        ApplicationTrainStepType nextTrainStep) {
        if ("PRODUCT_DETAILS".equalsIgnoreCase(currentTrainStep.toString())) {
            ApplicantDataDTO applicantDataDTO = helper.getSubmissionInputDTO().getApplicantDataList()[0];

            com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO1 = new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
            nameValuePairDTO1.setGenericName("com.ofss.fc.domain.origination.entity.core.applicant.CustomizedApplicant.CriminalRecord");
            nameValuePairDTO1.setValue("No");
            com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[] nameValuePairDTOArray = new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[1];
            nameValuePairDTOArray[0] = nameValuePairDTO1;
            com.ofss.fc.framework.domain.common.dto.Dictionary dictionary = new com.ofss.fc.framework.domain.common.dto.Dictionary();
            dictionary.setNameValuePairDTOArray(nameValuePairDTOArray);

            com.ofss.fc.framework.domain.common.dto.Dictionary[] dictionaryArray = new com.ofss.fc.framework.domain.common.dto.Dictionary[1];
            dictionaryArray[0] = dictionary;
            applicantDataDTO.setDictionaryArray(dictionaryArray);
        }
        return false;
    }

    public void onLoad(ApplicationFormHelper helper,
        ApplicationTrainStepType currentTrainStep) {
    }

    public void onSave(ApplicationFormHelper helper,
        ApplicationTrainStepType currentTrainStep) {
    }

    public void onSubmit(ApplicationFormHelper helper) {
    }

    public void onTrainStepClick(ApplicationFormHelper helper,
    }
}

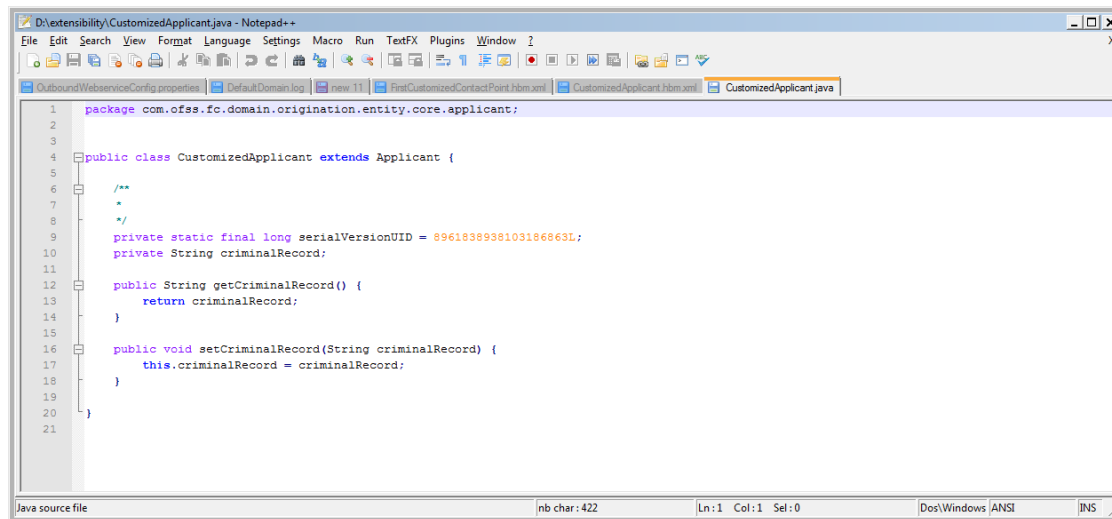
```

This hook should be used to populate the dictionary array of the concerned DTO at the correct stage of application form entry.

17.6.2 Create Customized Abstract Domain Object Class

A new Java File is added corresponding to the existing Abstract Domain Object. This extends the Abstract Domain Object that we are extending.

Figure 17–24 Create Customized Abstract Domain Object Class



```

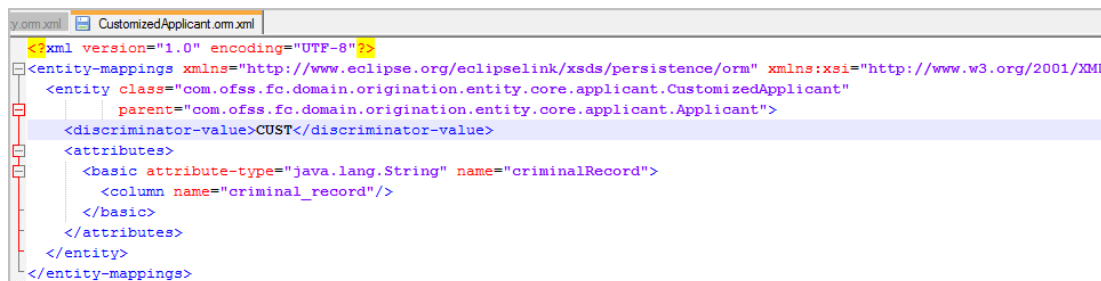
1 package com.ofss.fc.domain.Origination.entity.core.applicant;
2
3
4 public class CustomizedApplicant extends Applicant {
5
6     /**
7     *
8     */
9     private static final long serialVersionUID = 8961838938103186863L;
10    private String criminalRecord;
11
12    public String getCriminalRecord() {
13        return criminalRecord;
14    }
15
16    public void setCriminalRecord(String criminalRecord) {
17        this.criminalRecord = criminalRecord;
18    }
19
20 }
21

```

17.6.3 Create Customized Abstract Domain Object ORM Mapping File

A new file .orm.xml is introduced to include the extra attributes added by consulting or any other third party along with the discriminator value. This file maps to the new customized domain object and extends the existing Abstract Domain Object.

Figure 17–25 Create Customized Abstract Domain Object ORM Mapping File



```

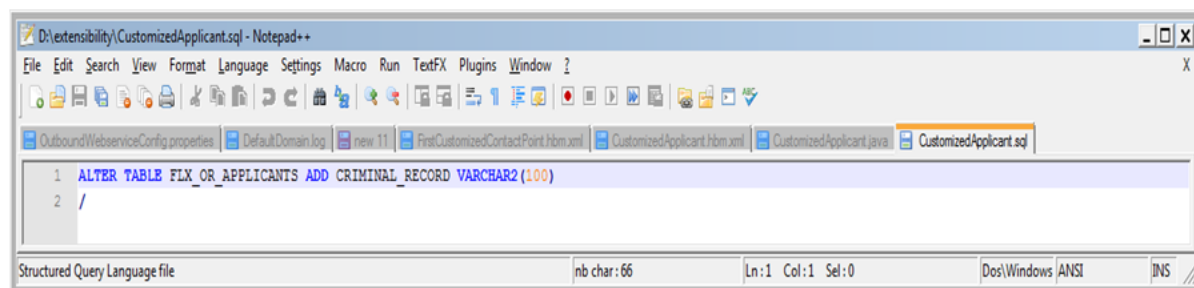
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://www.eclipse.org/eclipselink/xsd/persistence/orm" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  >
  <entity class="com.ofss.fc.domain.Origination.entity.core.applicant.CustomizedApplicant"
    parent="com.ofss.fc.domain.Origination.entity.core.applicant.Applicant">
    <discriminator-value>CUST</discriminator-value>
    <attributes>
      <basic attribute-type="java.lang.String" name="criminalRecord">
        <column name="criminal_record"/>
      </basic>
    </attributes>
  </entity>
</entity-mappings>

```

17.6.4 Create Customized Abstract Domain Object Attribute Columns

The extra columns have to be added to the domain object table of this domain object.

Figure 17–26 Create Customized Abstract Domain Object Attribute Columns



```

1 ALTER TABLE FLX_OR_APPLICANTS ADD CRIMINAL_RECORD VARCHA2(100)
2 /

```

In case of Creation or Updation of 'CustomizedApplicant' instead of 'Applicant' the existing discriminator column 'DOMAIN_OBJECT_EXTN' has the value of 'CUST' instead of 'CZ' and an additional value in column 'CRIMINAL_RECORD' in table FLX_OR_APPLICANTS.

In case of Creation or Updation of 'Applicant' the existing discriminator column 'DOMAIN_OBJECT_EXTN' has the value of 'CZ' and NULL values in column 'CRIMINAL_RECORD' in table FLX_OR_APPLICANTS.

Similarly, other DomainObjectDTO's can have their dictionary arrays populated in the ICustomDataHandler class being used and the corresponding customized domain object will get persisted instead of the usual domain object.

17.7 Extensibility using Attributes of Various Supported Datatypes

Extensibility of maintenance domain objects now supports extended attributes with all data types that have a public constructor with a single argument of data-type "String".

This includes attributes of data-type "com.ofss.fc.datatype.Date" whose "toString()" method should be invoked to set its value in NameValuePairDTO array element of Dictionary array. The value set is of the format given in root.properties file.

Additionally extensibility of maintenance domain objects is now also supporting extended attributes with enumeration data types defined in "com.ofss.fc.enumeration" project.

Here is an example of extensibility of "com.ofss.fc.domain.ep.entity.dispatch.message.MessageTemplate" using attributes of different supported datatypes.

The following customized class is created that contains the additional attributes.

Figure 17–27 Customized Message Template Class

```

package com.ofss.fc.domain.ep.entity.dispatch.message;

import com.ofss.fc.datatype.Date;
import com.ofss.fc.enumeration.ep.DestinationType;

public class CustomizedMessageTemplate extends MessageTemplate{

    private static final long serialVersionUID = 376283690240542791L;

    private Integer attributeInteger;

    private Boolean attributeBoolean;

    private String attributeString;

    private Date attributeDate;

    private DestinationType attributeEnum;

    public Integer getAttributeInteger() {
        return attributeInteger;
    }

    public void setAttributeInteger(Integer attributeInteger) {
        this.attributeInteger = attributeInteger;
    }

    public Boolean getAttributeBoolean() {
        return attributeBoolean;
    }

    public void setAttributeBoolean(Boolean attributeBoolean) {
        this.attributeBoolean = attributeBoolean;
    }

    public String getAttributeString() {
        return attributeString;
    }

    public void setAttributeString(String attributeString) {
        this.attributeString = attributeString;
    }

    public Date getAttributeDate() {
        return attributeDate;
    }

    public DestinationType getAttributeEnum() {
        return attributeEnum;
    }

    public void setAttributeEnum(DestinationType attributeEnum) {
        this.attributeEnum = attributeEnum;
    }
}

```

The following extra columns have been added in the domain object table "flx_ep_msg_tmpl_b".

Figure 17–28 Domain Object Table

Name	Type	Nullable	Default	Storage	Comments
▶ COD_TMPL_ID	VARCHAR2(100)	<input type="checkbox"/>			Indicates unique message template id
DESTINATION_TYPE	VARCHAR2(20)	<input checked="" type="checkbox"/>			Indicates destination type like SMS,Email..
MSG_TMPL_NAME	VARCHAR2(100)	<input checked="" type="checkbox"/>			Indicates message template name
MSG_TMPL_DESC	VARCHAR2(100)	<input checked="" type="checkbox"/>			Indicates message template description
TXT_MSG_TMPL	CLOB	<input checked="" type="checkbox"/>			Indicates message template buffer
CREATED_BY	VARCHAR2(64)	<input checked="" type="checkbox"/>			Indicates the creator
CREATION_DATE	DATE	<input checked="" type="checkbox"/>			Indicates the creation Date
LAST_UPDATED_BY	VARCHAR2(64)	<input checked="" type="checkbox"/>			Indicates the approver of the transaction
LAST_UPDATE_DATE	DATE	<input checked="" type="checkbox"/>			Indicates the last updated date
OBJECT_VERSION_NUMBER	NUMBER(9)	<input checked="" type="checkbox"/>			Indicates the version number. Defaults to 1 for new instances.
OBJECT_STATUS	VARCHAR2(5)	<input checked="" type="checkbox"/>			Indicates current status of the entity.
TXT_SUBJECT_TMPL	CLOB	<input checked="" type="checkbox"/>			Indicates message for subject Buffer
DOMAIN_OBJECT_EXTN	VARCHAR2(100)	<input checked="" type="checkbox"/>			
CUST_INTEGER	NUMBER(9)	<input checked="" type="checkbox"/>			
CUST_BOOLEAN	VARCHAR2(5)	<input checked="" type="checkbox"/>			
CUST_DATE	DATE	<input checked="" type="checkbox"/>			
CUST_STRING	VARCHAR2(100)	<input checked="" type="checkbox"/>			
CUST_ENUM	VARCHAR2(100)	<input checked="" type="checkbox"/>			

The following ORM file maps the customized class attributes with the table columns.

Figure 17–29 ORM File

```

<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  >
  <entity class="com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate"
    parent="com.ofss.fc.domain.ep.entity.dispatch.message.MessageTemplate"
    >
    <discriminator-value>CUST</discriminator-value>
    <attributes>
      <basic attribute-type="int" name="attributeInteger">
        <column name="cust_integer"/>
      </basic>
      <basic attribute-type="java.lang.Boolean" name="attributeBoolean">
        <column name="cust_boolean"/>
        <convert>yesno</convert>
      </basic>
      <basic attribute-type="java.lang.String" name="attributeString">
        <column name="cust_string"/>
      </basic>
      <basic attribute-type="com.ofss.fc.datatype.Date" name="attributeDate">
        <column name="cust_date"/>
        <convert>Date</convert>
      </basic>
      <basic attribute-type="com.ofss.fc.enumeration.ep.DestinationType" name="attributeEnum">
        <column name="cust_enum"/>
        <enumerated>VALUE</enumerated>
      </basic>
    </attributes>
  </entity>
</entity-mappings>

```

The following JUnit test case has been used to test a "create" operation.

Figure 17–30 JUnit Test Case

```

@Test
public void testAddMessageTemplate() {
    String testCase = "testAdd.messageTemplateDTO.";
    MessageTemplateApplicationService applicationService = new MessageTemplateApplicationService();
    SessionContext sessionContext = getSessionContext();
    MessageTemplateDTO messageTemplateDTO = populateMessageTemplateDTO(testCase);
    try {
        deleteMessageTemplate(testCase);

        com.ofss.fc.framework.domain.common.dto.Dictionary[] dictionaryArray= new com.ofss.fc.framework.domain.common.dto.Dictionary[1];
        com.ofss.fc.framework.domain.common.dto.Dictionary dictionaryObject = new com.ofss.fc.framework.domain.common.dto.Dictionary();

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[] nameValuePairDTOArray= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO[5];

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO0= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
        nameValuePairDTO0.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.AttributeInteger");
        nameValuePairDTO0.setValue("100");
        nameValuePairDTOArray[0]=nameValuePairDTO0;

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO1= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
        nameValuePairDTO1.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.AttributeBoolean");
        nameValuePairDTO1.setValue("false");
        nameValuePairDTOArray[1]=nameValuePairDTO1;

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO2= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
        nameValuePairDTO2.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.AttributeString");
        nameValuePairDTO2.setValue("ABCDEFRA");
        nameValuePairDTOArray[2]=nameValuePairDTO2;

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO3= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
        nameValuePairDTO3.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.AttributeDate");
        Date newDate = new Date();
        nameValuePairDTO3.setValue(newDate.toString());
        nameValuePairDTOArray[3]=nameValuePairDTO3;

        com.ofss.fc.framework.domain.common.dto.NameValuePairDTO nameValuePairDTO4= new com.ofss.fc.framework.domain.common.dto.NameValuePairDTO();
        nameValuePairDTO4.setGenericName("com.ofss.fc.domain.ep.entity.dispatch.message.CustomizedMessageTemplate.AttributeEnum");
        nameValuePairDTO4.setValue(com.ofss.fc.enumeration.ep.DestinationType.EMAIL.getEnumValue());
        nameValuePairDTOArray[4]=nameValuePairDTO4;

        dictionaryObject.setNameValuePairDTOArray(nameValuePairDTOArray);
        dictionaryArray[0]=dictionaryObject;
        messageTemplateDTO.setDictionaryArray(dictionaryArray);

        TransactionStatus result = applicationService.addMessageTemplate(sessionContext, messageTemplateDTO);
        assertEquals(result.getErrorCode(), FAPIErrorConstants.MIZD_SUCCESS);
        dumpTransactionStatus("MessageTemplateApplicationService", "testAddMessageTemplate", result);
    } catch (FatalException e) {
        dumpFatalException("MessageTemplateApplicationService", "testAddMessageTemplate", e);
        fail("Unexpected failure from " + THIS_COMPONENT_NAME + ".testAddMessageTemplate");
    }
}

```

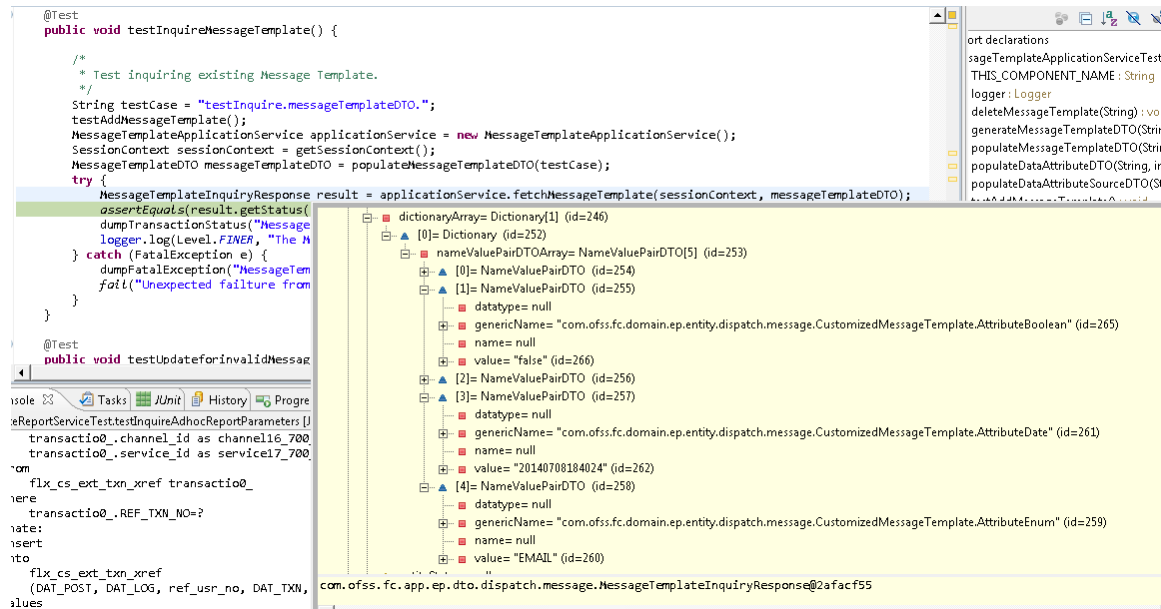
The above JUnit runs to add the following record in the table.

Figure 17–31 JUnit Adds Table Record

Row 1	Fields
▶ COD_TMPL_ID	JUnit_Message
DESTINATION_TYPE	
MSG_TMPL_NAME	JUnit message template
MSG_TMPL_DESC	Message template description via junit test cas
TXT_MSG_TMPL	<CLOB>
CREATED_BY	ofssuser
CREATION_DATE	7/8/2014 6:40:34 PM
LAST_UPDATED_BY	ofssuser
LAST_UPDATE_DATE	7/8/2014 6:40:34 PM
OBJECT_VERSION_NUMBER	1
OBJECT_STATUS	A
TXT_SUBJECT_TMPL	<CLOB>
DOMAIN_OBJECT_EXTN	CUST
CUST_INTEGER	100
CUST_BOOLEAN	0
CUST_DATE	7/8/2014 6:40:24 PM
CUST_STRING	ABCDEFRA
CUST_ENUM	EMAIL

Similarly, a JUnit is run to do "fetch" operation. This fetches the customized record whose dictionary array values have been shown below.

Figure 17–32 Dictionary Array Values



17.8 Customized Domain Object having Collection of Objects as Attributes

Figure 17–33 Customized Domain Object having collection of Objects as Attributes

```
package com.ofss.fc.dictionary;
import java.util.List;

public class CustomizedMessageTemplate extends MessageTemplate{

    private static final long serialVersionUID = 3762836902405427911L;

    private int attributeInt;

    private boolean attributeBool;

    private char attributeChar;

    private Money attributeMoney;

    private Integer attributeInteger;

    private Boolean attributeBoolean;

    private String attributeString;

    private Date attributeDate;

    private DestinationType attributeEnum;

    private List<MessageAttribute> messageAttributeList;
    private List<MessageRecipient> messageRecipientList;

    public Integer getAttributeInteger() {
        return attributeInteger;
    }

    public List<MessageRecipient> getMessageRecipientList() {
        return messageRecipientList;
    }

    public void setMessageRecipientList(List<MessageRecipient> messageRecipientList) {
```


Figure 17–34 Member Attributes of Customized Domain Object

```

package com.ofss.fc.dictionary;

public class MessageAttribute {

    private String messageTag;

    private String masking;

    public String getMessageTag() {
        return messageTag;
    }

    public void setMessageTag(String messageTag) {
        this.messageTag = messageTag;
    }

    public String getMasking() {
        return masking;
    }

    public void setMasking(String masking) {
        this.masking = masking;
    }

}

package com.otss.fc.dictionary;

public class MessageRecipient {

    private String name;

    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

}

```

Figure 17–35 Dictionary Array Elements

```

private void populateDictionaryData(MessageTemplateDTO messageTemplateDTO) {
    Dictionary[] dictionaryArray= new Dictionary[4];
    Dictionary dictionaryObject= new Dictionary();
    dictionaryObject.setFullyQualifiedClassName("com.ofss.fc.dictionary.CustomizedMessageTemplate");
    NameValuePairDTO[] nameValuePairDTOArray= new NameValuePairDTO[6];
    nameValuePairDTOArray[0]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeInteger","100");
    nameValuePairDTOArray[1]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeBoolean","true");
    nameValuePairDTOArray[2]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeString","ABCDEFR");
    nameValuePairDTOArray[3]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeInt","101");
    nameValuePairDTOArray[4]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeBool","true");
    nameValuePairDTOArray[5]=getNameValuePairDTO("com.ofss.fc.dictionary.CustomizedMessageTemplate.AttributeChar","C");
    dictionaryObject.setNameValuePairDTOArray(nameValuePairDTOArray);
    dictionaryArray[0]=dictionaryObject;
    Dictionary dictionaryObject1= new Dictionary();
    dictionaryObject1.setFullyQualifiedClassName("com.ofss.fc.dictionary.CustomizedMessageTemplate.MessageAttributeList");
    NameValuePairDTO[] nameValuePairDTOArray1= new NameValuePairDTO[2];
    nameValuePairDTOArray1[0]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageAttribute.MessageTag","100");
    nameValuePairDTOArray1[1]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageAttribute.Masking","D");
    dictionaryObject1.setNameValuePairDTOArray(nameValuePairDTOArray1);
    dictionaryArray[1]=dictionaryObject1;
    Dictionary dictionaryObject2= new Dictionary();
    dictionaryObject2.setFullyQualifiedClassName("com.ofss.fc.dictionary.CustomizedMessageTemplate.MessageAttributeList");
    NameValuePairDTO[] nameValuePairDTOArray2= new NameValuePairDTO[2];
    nameValuePairDTOArray2[0]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageAttribute.MessageTag","111");
    nameValuePairDTOArray2[1]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageAttribute.Masking","DD");
    dictionaryObject2.setNameValuePairDTOArray(nameValuePairDTOArray2);
    dictionaryArray[2]=dictionaryObject2;
    Dictionary dictionaryObject3= new Dictionary();
    dictionaryObject3.setFullyQualifiedClassName("com.ofss.fc.dictionary.CustomizedMessageTemplate.MessageRecipientList");
    NameValuePairDTO[] nameValuePairDTOArray3= new NameValuePairDTO[2];
    nameValuePairDTOArray3[0]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageRecipient.Name","Raju");
    nameValuePairDTOArray3[1]=getNameValuePairDTO("com.ofss.fc.dictionary.MessageRecipient.Age","35");
    dictionaryObject3.setNameValuePairDTOArray(nameValuePairDTOArray3);
    dictionaryArray[3]=dictionaryObject3;
    messageTemplateDTO.setDictionaryArray(dictionaryArray);
}

```

To construct a CustomizedMessageTemplate having 2 elements in messageAttributeList and 1 element in messageRecipientList, set the dictionaryArray of MessageTemplateDTO as follows:

The dictionaryArray has four elements as highlighted in the above figure.

- The 0th dictionaryArray element will have NameValuePairDTO array of non-collection attributes. This element's fullyQualifiedClassName will be the fully qualified class name of the customized domain object that is being constructed.

- The 1st dictionaryArray element will have NameValuePairDTO array of 1st element of 1st collection attribute. This element's fullyQualifiedClassName will be the fully qualified class name of the customized domain object that is being constructed, appended with "." and 1st collection attribute name.
- The 2nd dictionaryArray element will have NameValuePairDTO array of 2nd element of 1st collection attribute. This element's fullyQualifiedClassName will be the fully qualified class name of the customized domain object that is being constructed, appended with "." and 1st collection attribute name.
- The 3rd dictionaryArray element will have NameValuePairDTO array of 1st element of 2nd collection attribute. This element's fullyQualifiedClassName will be the fully qualified class name of the customized domain object that is being constructed, appended with "." and 2nd collection attribute name.

Figure 17–36 Customized Domain Object constructed by AbstractAssembler

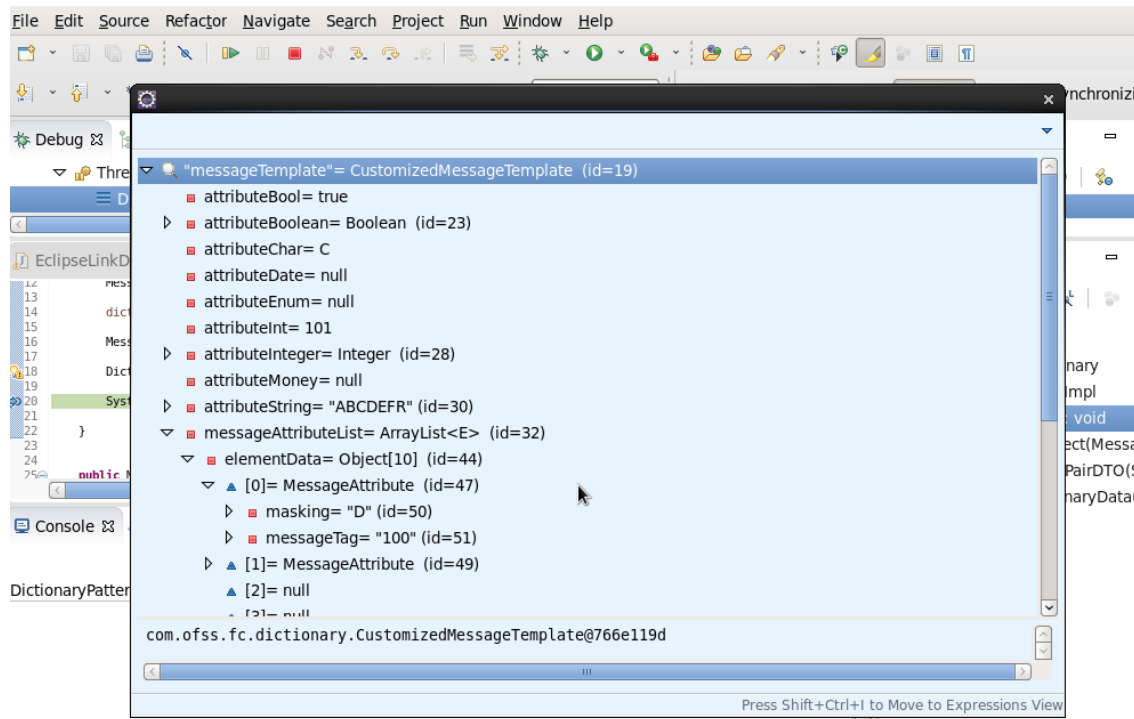
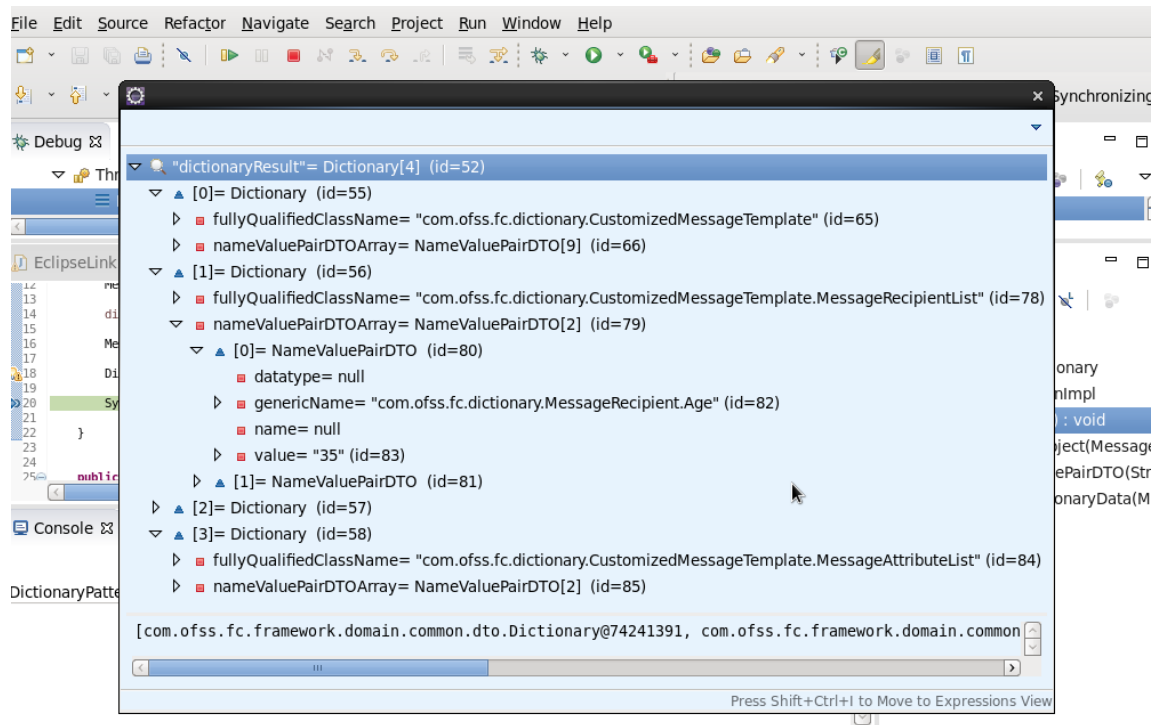


Figure 17–37 Dictionary Array returned by AbstractAssembler



17.9 Limitation to Extensibility using Dictionary Pattern

Extensibility of domain objects using Dictionary pattern is not applicable to those Maintenance Domain Objects that implement `com.ofss.fc.framework.domain.search.ISearchableEntity`.

The following is the list of the `ISearchableEntity`:

- `com.ofss.fc.domain.config.entity.OBPCConfigurationProperty`
- `com.ofss.fc.domain.origination.entity.core.submission.Submission`
- `com.ofss.fc.domain.party.entity.textsearch.PartyAggregateSummary`
- `com.ofss.fc.domain.account.entity.statement.impl.TDFinancialStatementItem`
- `com.ofss.fc.domain.account.entity.statement.impl.LoanFinancialStatementItem`
- `com.ofss.fc.domain.account.entity.statement.impl.DDFinancialStatementItem`
- `com.ofss.fc.domain.account.entity.statement.impl.DDNonFinancialStatementItem`
- `com.ofss.fc.domain.account.entity.statement.impl.LoanNonFinancialStatementItem`
- `com.ofss.fc.domain.account.entity.transactingparty.TransactingParty`
- `com.ofss.fc.domain.lcm.entity.collaterals.businessassets.AllPAPExcept`
- `com.ofss.fc.domain.lcm.entity.collaterals.businessassets.BusinessAssets`
- `com.ofss.fc.domain.lcm.entity.collaterals.businessassets.AllPAP`
- `com.ofss.fc.domain.lcm.entity.collaterals.fixedasset.computerhardware.ComputerHardware`
- `com.ofss.fc.domain.lcm.entity.collaterals.fixedasset.Machinery`

-
- com.ofss.fc.domain.lcm.entity.collaterals.fixedasset.computersoftware.ComputerSoftware
 - com.ofss.fc.domain.lcm.entity.collaterals.fixedasset.FixedAsset
 - com.ofss.fc.domain.lcm.entity.collaterals.fixedasset.Furniture
 - com.ofss.fc.domain.lcm.entity.collaterals.industrybusinessvalue.IndustryBusinessValue
 - com.ofss.fc.domain.lcm.entity.collaterals.agriculture.Agriculture
 - com.ofss.fc.domain.lcm.entity.collaterals.agriculture.crop.Crops
 - com.ofss.fc.domain.lcm.entity.collaterals.agriculture.livestock.LiveStocks
 - com.ofss.fc.domain.lcm.entity.collaterals.agreementandundertaking.NonFinancialAgreementAndUndertaking
 - com.ofss.fc.domain.lcm.entity.collaterals.agreementandundertaking.AgreementAndUndertaking
 - com.ofss.fc.domain.lcm.entity.collaterals.currentassets.inventorystock.InventoryStocks
 - com.ofss.fc.domain.lcm.entity.collaterals.currentassets.CurrentAssets
 - com.ofss.fc.domain.lcm.entity.collaterals.currentassets.bookdebt.BookDebts
 - com.ofss.fc.domain.lcm.entity.collaterals.currentassets.receivable.Receivable
 - com.ofss.fc.domain.lcm.entity.collaterals.automobile.PassengerVehicle
 - com.ofss.fc.domain.lcm.entity.collaterals.automobile.Automobile
 - com.ofss.fc.domain.lcm.entity.collaterals.automobile.GoodsVehicle
 - com.ofss.fc.domain.lcm.entity.collaterals.investmentsecurities.InvestmentSecurities
 - com.ofss.fc.domain.lcm.entity.collaterals.investmentsecurities.SharesStock
 - com.ofss.fc.domain.lcm.entity.collaterals.investmentsecurities.InvestmentSecurity
 - com.ofss.fc.domain.lcm.entity.collaterals.intangibleasset.IntangibleAsset
 - com.ofss.fc.domain.lcm.entity.collaterals.other.OtherCollateral
 - com.ofss.fc.domain.lcm.entity.collaterals.insurance.lifeinsurance.LifeInsurance
 - com.ofss.fc.domain.lcm.entity.collaterals.insurance.Insurance
 - com.ofss.fc.domain.lcm.entity.collaterals.bullion.Bullion
 - com.ofss.fc.domain.lcm.entity.collaterals.cash.TermDeposit
 - com.ofss.fc.domain.lcm.entity.collaterals.cash.CashDeposit
 - com.ofss.fc.domain.lcm.entity.collaterals.Collateral
 - com.ofss.fc.domain.lcm.entity.collaterals.proposedcollateral.ProposedCollateralRequest
 - com.ofss.fc.domain.lcm.entity.collaterals.proposedcollateral.IPARRequest
 - com.ofss.fc.domain.lcm.entity.collaterals.proposedcollateral.SubDivisionRequest
 - com.ofss.fc.domain.lcm.entity.collaterals.proposedcollateral.ConsolidationRequest
 - com.ofss.fc.domain.lcm.entity.collaterals.guarantee.PersonalGuarantee
 - com.ofss.fc.domain.lcm.entity.collaterals.guarantee.Guarantee
 - com.ofss.fc.domain.lcm.entity.collaterals.guarantee.FamilyGuarantee

- com.ofss.fc.domain.lcm.entity.collaterals.guarantee.BankGuarantee
- com.ofss.fc.domain.lcm.entity.collaterals.guarantee.GuaranteeAndIndemnity
- com.ofss.fc.domain.lcm.entity.collaterals.guarantee.GovernmentGuarantee
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.IndustrialProperty
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.WaterProperty
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.CommercialProperty
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.RealEstate
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.ResidentialProperty
- com.ofss.fc.domain.lcm.entity.collaterals.realestate.RuralProperty
- com.ofss.fc.domain.lcm.entity.collaterals.artwork.ArtWork
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.smallaircraft.SmallAirCraft
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.cargoaircraft.CargoAirCraft
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.airframe.AirFrame
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.passengeraircraft.PassengerAirCraft
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.helicopter.HeliCopter
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.aircraftengine.AirCraftEngine
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.otheraircraft.OtherAirCraft
- com.ofss.fc.domain.lcm.entity.collaterals.aircraft.AirCraft
- com.ofss.fc.domain.lcm.entity.collaterals.ship.Ship
- com.ofss.fc.domain.lcm.entity.collaterals.license.WaterLicense
- com.ofss.fc.domain.lcm.entity.collaterals.license.License
- com.ofss.fc.domain.lcm.entity.collaterals.license.liquorlicense.LiquorLicense
- com.ofss.fc.domain.lcm.entity.collaterals.license.fishinglicense.FishingLicense
- com.ofss.fc.domain.lcm.entity.collaterals.license.managementrights.ManagementRights
- com.ofss.fc.domain.lcm.entity.collaterals.license.taxilicense.TaxiLicense
- com.ofss.fc.domain.pc.entity.institution.FinancialInstitution
- com.ofss.fc.framework.audit.AuditItem

18 OCH Integration

This chapter describes how additional information can be added to an Oracle Customer Hub (henceforth mentioned as OCH) publish request. Publishing additional information can be required base on the client requirements, and hence OBP Integration adapters and assemblers need to be extended for such additional informations. Integration adapters are used for gathering data related to a customer, which is further used by assemblers to map OBP DTO to AIA Enterprise Business Objects (henceforth mentioned as EBOs).

OBP OCH integration involves the following steps:

1. Fetching all the data related to customer depending on the use case
2. Conversion of OBP DTO to AIA EBOs
3. Posting the EBO to AIA queue using Asynch JMS protocol

Integration adapters are invoked from the post hook of application service extensions. After the successful execution of the use case, adapters further call Integration assemblers for conversion of DTO to EBO.

After conversion, adapters post the serialized EBO request to AIA queue using Integration strategy, which is fetched on the basis of use case.

A few examples of Integration strategies are as follows:

- **AsyncFireForgetIntegrationStrategyJMS**: It is used in use cases where a response is not expected from OCH. Integration use cases involving creation/updation of customer information use this strategy.
- **SyncIntegrationStrategy**: It is used where a response is required from OCH. Uses cases, like Party Search or Party Deduplication where customer information is fetched from OCH, use this strategy.

A few examples of Integration adapters are:

- **UpdatepartyAdapter**: It is used for populating customer information.
- **ChangeAccountTitleAdapter**: It is used in use cases where customer's account information is to be published to OCH.

A few examples of Integration assemblers are:

- **UpdatePartyAssembler**: It is invoked from UpdatepartyAdapter and maps customer information to EBO attributes.
- **CreateAccountAssembler**: It is invoked from ChangeAccountTitleAdapter and maps customer's account information to respective EBO attribute.

18.1 Integration Adapter Interface

OBP framework contains an interface, `IntegrationAdapter` which provides two basic methods for OCH integration.

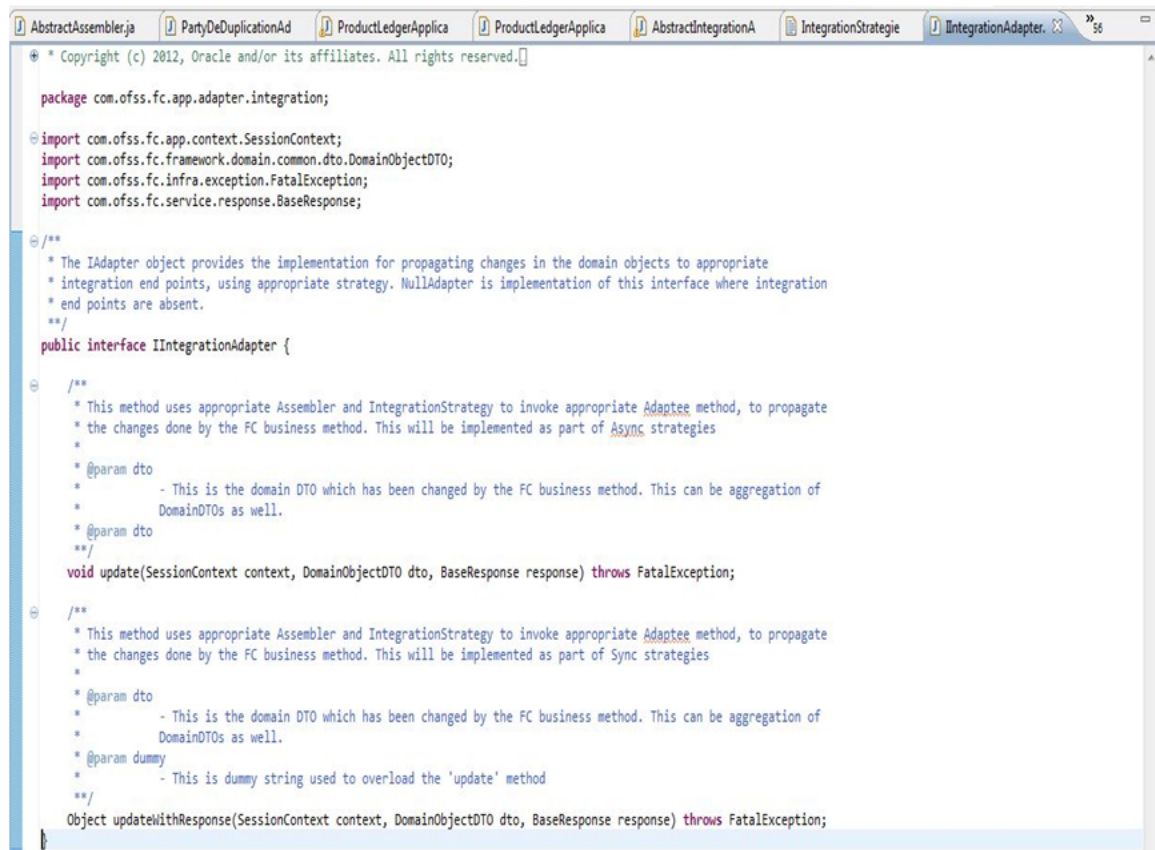
These two methods must be implemented by any adapter implementing the interface and use them for publishing data to OCH. Signature of these two methods are:

```
void update(SessionContext context, DomainObjectDTO dto,
BaseResponse response) throws FatalException;
Object updateWithResponse(SessionContext context, DomainObjectDTO
dto, BaseResponse response) throws FatalException;
```

Update() method is used in the use cases where response is not expected from OCH.

UpdateWithResponse() method is used when the data is required from OCH.

Figure 18–1 Integration Adapter Interface



```
Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.

package com.ofss.fc.app.adapter.integration;

import com.ofss.fc.app.context.SessionContext;
import com.ofss.fc.framework.domain.common.dto.DomainObjectDTO;
import com.ofss.fc.infra.exception.FatalException;
import com.ofss.fc.service.response.BaseResponse;

/**
 * The IAdapter object provides the implementation for propagating changes in the domain objects to appropriate
 * integration end points, using appropriate strategy. NullAdapter is implementation of this interface where integration
 * end points are absent.
 */
public interface IIntegrationAdapter {

    /**
     * This method uses appropriate Assembler and IntegrationStrategy to invoke appropriate Adapter method, to propagate
     * the changes done by the FC business method. This will be implemented as part of Async strategies
     *
     * @param dto
     *     - This is the domain DTO which has been changed by the FC business method. This can be aggregation of
     *     DomainDTOs as well.
     * @param response
     */
    void update(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException;

    /**
     * This method uses appropriate Assembler and IntegrationStrategy to invoke appropriate Adapter method, to propagate
     * the changes done by the FC business method. This will be implemented as part of Sync strategies
     *
     * @param dto
     *     - This is the domain DTO which has been changed by the FC business method. This can be aggregation of
     *     DomainDTOs as well.
     * @param dummy
     *     - This is dummy string used to overload the 'update' method
     */
    Object updateWithResponse(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException;
```

18.2 Abstract Integration Adapter Class

OBP framework has an abstract class AbstractIntegrationAdapter which provides methods for common data, such as audit information or session context etc. This abstract class implements IIntegrationAdapter interface.

All adapters must extend AbstractIntegrationAdapter and implement the two methods defined in the IIntegrationAdapter interface.

Figure 18–2 Abstract Integration Adapter Class

```

public abstract class AbstractIntegrationAdapter implements IIntegrationAdapter {
    protected SessionContext sessionContext;
    protected String serviceId;
    private static final String ALL_SERVICES = "ALL";

    /**
     * Constructor that validates the service to be integrated.
     */
    public AbstractIntegrationAdapter(SessionContext sessionContext, String serviceId) throws ConfigurationInitializationException {
        this.sessionContext = sessionContext;
        this.serviceId = serviceId;
        boolean isAllowed = isIntegrationAllowed(sessionContext.getChannel(), serviceId);
        if ( !isAllowed) {
            throw new ConfigurationInitializationException(InfraErrorConstants.INTEGRATION_NOT_CONFIGURED);
        }
    }

    @Override
    public abstract void update(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException;

    @Override
    public abstract Object updateWithResponse(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException;

    /**
     * @return the sessionContext
     */
    public SessionContext getSessionContext() {
        return sessionContext;
    }

    protected DomainObjectDTO populateCreateAuditInformation(SessionContext sessionContext, DomainObjectDTO dto) {
        dto.setCreatedBy(sessionContext.getUserId());
        dto.setLastUpdatedBy(sessionContext.getUserId());
        return dto;
    }

    protected DomainObjectDTO populateUpdatedAuditInformation(SessionContext sessionContext, DomainObjectDTO dto) {
        dto.setLastUpdatedBy(sessionContext.getUserId());
        return dto;
    }
}

```

18.3 Sample Integration Adapter

The following figure is a sample adapter for customer information:

Figure 18–3 Sample Integration Adapter

```

public class SampleAdapter extends AbstractIntegrationAdapter {

    public SampleAdapter(SessionContext sessionContext, String serviceId) throws ConfigurationInitializationException, FatalException,
        InvocationTargetException {

        super(sessionContext, serviceId);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void update(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException {

        // TODO Auto-generated method stub
        if (dto instanceof IndividualDemographicsDTO) {
            PartyKeyDTO partyKeyDTO = new PartyKeyDTO();
            IntegrationPartyOnBoardingDTO integrationOnBoardingDTO = new IntegrationPartyOnBoardingDTO();
            PartyOnBoardingDTO partyOnBoardingDTO = new PartyOnBoardingDTO();
            PartyInquiryResponse partyInquiryResponse = new PartyInquiryResponse();
            PartyApplicationService partyApplicationService = new PartyApplicationService();
            IndividualDTO individualDTO = new IndividualDTO();
            IndividualDemographicsDTO individualDemographicsDTO = (IndividualDemographicsDTO) dto;
            partyKeyDTO.setPartyId(individualDemographicsDTO.getPartyDemographicsKeyDTO().getPartyId());
            partyInquiryResponse = partyApplicationService.fetchPartyDetailsWithoutDemographics(context, individualDemographicsDTO.getPartyDemographicsKeyDTO()
                .getPartyId());

            individualDTO.setPartyKeyDTO(partyKeyDTO);
            individualDTO.setPartyType(partyInquiryResponse.getPartyType());
            individualDTO.setIndividualDemographicsDTO(individualDemographicsDTO);
            partyOnBoardingDTO.setIndividualDTO(individualDTO);
            integrationOnBoardingDTO.setPartyOnBoardingDTO(partyOnBoardingDTO);
            AbstractAssembler<CanonicalModel, DomainObjectDTO> updatePartyAssembler = IntegrationAssemblerFactory.getInstance()
                .fetchAssemblerInstance("com.ofss.fc.app.integration.dto.common.UpdatePartyAdapter.IntegrationPartyOnBoardingDTO");
            SyncCustomerPartyListEBMType customerEBO = (SyncCustomerPartyListEBMType) updatePartyAssembler.toCanonicalModel(integrationOnBoardingDTO);
            IntegrationStrategy strategy = IntegrationStrategyFactory.getInstance().fetchStrategyInstance("FC-OCH-updateParty");
            strategy.invoke(customerEBO, individualDemographicsDTO.getPartyDemographicsKeyDTO().getPartyId());
        }

        @Override
        public Object updateWithResponse(SessionContext context, DomainObjectDTO dto, BaseResponse response) throws FatalException {

            // TODO Auto-generated method stub
            return null;
        }
    }
}

```

Application Service calls to populate data.

Fetch Assembler

Fetch Strategy and invoke

18.4 Integration Abstract Assembler

OBP framework has an abstract class `AbstractAssembler` which provides design for DTO to EBO conversion. These methods are used while mapping DTO to EBO and vice versa.

Signature of methods are:

```

public abstract T toCanonicalModel(D dto) throws FatalException;
public abstract D fromCanonicalModel(T domainObject) throws
    FatalException;

```

`toCanonicalModel()` is used when DTO is to be converted to EBO and `fromCanonicalModel()` in the other case.

Figure 18–4 Integration Abstract Assembler

```

public abstract class AbstractAssembler<T extends ICanonicalModel, D extends DomainObjectDTO> {
    /**
     * This method needs to be implemented to convert from a DTO array to a canonical object.
     *
     * @param dto
     *     The input DTO which implements Serializable.
     * @return The canonical object instance.
     */
    public abstract T toCanonicalModel(D dto) throws FatalException;

    /**
     * This method needs to be implemented to convert from a canonical object to a DTO array.
     *
     * @param domainObject
     *     Instance of canonical model.
     * @return Instance of DTO
     */
    public abstract D fromCanonicalModel(T domainObject) throws FatalException;
}

```

All the assemblers must implement these two methods for conversion of DTO to EBO and vice versa. Assemblers also populate the header of the request which is posted to the queue.

18.5 Sample Assembler

A sample assembler which extends AbstractAssembler should be like:

Figure 18–5 Sample Assembler

```

public class SampleAssembler extends AbstractAssembler<SyncCustomerPartyListEBMType, IntegrationPartyOnBoardingDTO> {

    @Override
    public SyncCustomerPartyListEBMType toCanonicalModel(IntegrationPartyOnBoardingDTO dto) throws FatalException {

        //Populate OCH EBO using OBP DTO
        SyncCustomerPartyListEBMType syncCustomerPartyListEBMType = new SyncCustomerPartyListEBMType();
        List<SyncCustomerPartyListDataAreaType> syncCustomerPartyListDataAreaTypes = new ArrayList<SyncCustomerPartyListDataAreaType>();
        SyncCustomerPartyListDataAreaType dataArea = new SyncCustomerPartyListDataAreaType();
        //call to populate details using utility
        dataArea.setSyncCustomerPartyList(PartyAssemblerUtility.CustomerPartyData(dto));
        dataArea.setSync(new SyncType());
        syncCustomerPartyListDataAreaTypes.add(dataArea);
        syncCustomerPartyListEBMType.getDataArea().addAll(syncCustomerPartyListDataAreaTypes);
        //call to populate request header
        syncCustomerPartyListEBMType.setEBMHeader(PartyAssemblerUtility.createUpsert());
        syncCustomerPartyListEBMType.setLanguageCode("English");
        return syncCustomerPartyListEBMType;
    }

    @Override
    public IntegrationPartyOnBoardingDTO fromCanonicalModel(SyncCustomerPartyListEBMType domainObject) throws FatalException {

        // Populate OBP Entity using OCH EBO
        IntegrationPartyOnBoardingDTO integrationPartyOnBoardingDTO = new IntegrationPartyOnBoardingDTO();
        PartyOnBoardingDTO partyOnBoardingDTO = new PartyOnBoardingDTO();
        //fetching value of party type
        String partyTypeStr = domainObject.getDataArea().get(0).getSyncCustomerPartyList().getTypeCode().getValue();
        PartyType partyType = (PartyType) EnumerationHelper.getInstance().fromValue(PartyType.class, partyTypeStr);
        //setting party type in OBP DTO
        partyOnBoardingDTO.setPartyType(partyType);
        integrationPartyOnBoardingDTO.setPartyOnBoardingDTO(partyOnBoardingDTO);
        return integrationPartyOnBoardingDTO;
    }
}

```

User can extend assemblers to add more DTO to EBO mapping.

Note

EBOs are generated from AIA wsdl, and can be extended to add extra fields in the custom tag using the standard AIA extension framework. For each newly added field, customization developer must set that field in the assembler.

19 Documaker Integration

This chapter describes how additional information can be added to the data sent to Documaker Server.

OBP sends data to Documaker in XML format. So, one XSD per document is created, and using that XSD, JAXB classes are generated.

19.1 XSD

Example of an existing XSD is **AddressChangeLetter.xsd**.

Figure 18–6 AddressChangeLetter.xsd

```

1 <?xml version="1.0"?>
2 <xs:schema id="AddressChangeLetter"
3   xmlns="http://addresschangeletter.extract.dto.integration.fc.ofss.com"
4   xmlns:context="http://context.common.extract.dto.integration.fc.ofss.com"
5   xmlns:xs="http://www.w3.org/2001/XMLSchema"
6   xmlns:offer="http://offer.extract.dto.integration.fc.ofss.com"
7   targetNamespace="http://addresschangeletter.extract.dto.integration.fc.ofss.com">
8
9   <xs:import namespace="http://context.common.extract.dto.integration.fc.ofss.com"
10    schemaLocation="Context.xsd" />
11   <xs:import namespace="http://offer.extract.dto.integration.fc.ofss.com"
12    schemaLocation="OfferExtract.xsd"/>
13
14   <xs:complexType name="AddressDetails">
15     <xs:sequence>
16       <xs:element name="OldAddress" type="xs:string" />
17       <xs:element name="NewAddress" type="xs:string" />
18     </xs:sequence>
19   </xs:complexType>
20   <xs:element name="AddressChangeLetter">
21     <xs:complexType>
22       <xs:sequence>
23         <xs:element name="Context" type="context:Context" />
24         <xs:element name="AddressDetails" type="AddressDetails" maxOccurs="unbounded"/>
25         <xs:element name="ReceivingParty" type="offer:ReceivingParty" />
26       </xs:sequence>
27     </xs:complexType>
28   </xs:element>
29 </xs:schema>
30

```

Additional elements can be added by creating new XSD and importing current XSD.

For Example,

CustomizedAddressChangeLetter can be created by importing existing **AddressChangeLetter.xsd**

Figure 18–7 CustomizedAddressChangeLetter.xsd

```

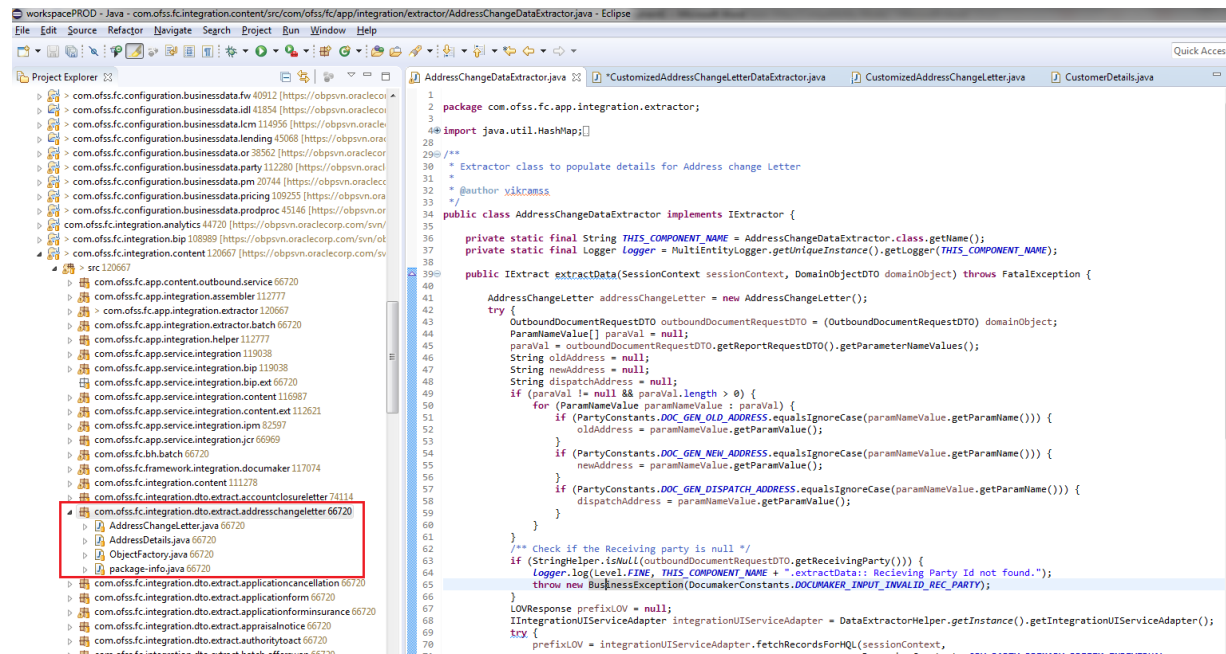
1 <?xml version="1.0"?>
2 <xs:schema id="CustomizedAddressChangeLetter"
3   xmlns="http://customizedaddresschangeletter.extract.dto.integration.fc.ofss.com"
4   xmlns:context="http://context.common.extract.dto.integration.fc.ofss.com"
5   xmlns:xs="http://www.w3.org/2001/XMLSchema"
6   xmlns:offer="http://offer.extract.dto.integration.fc.ofss.com"
7   xmlns:addresschangeletter="http://addresschangeletter.extract.dto.integration.fc.ofss.com"
8   targetNamespace="http://customizedaddresschangeletter.extract.dto.integration.fc.ofss.com">
9
10  <xs:import namespace="http://context.common.extract.dto.integration.fc.ofss.com"
11    schemaLocation="Context.xsd" />
12  <xs:import namespace="http://offer.extract.dto.integration.fc.ofss.com"
13    schemaLocation="OfferExtract.xsd"/>
14  <xs:import namespace="http://addresschangeletter.extract.dto.integration.fc.ofss.com"
15    schemaLocation="AddressChangeLetter.xsd" />
16
17  <xs:complexType name="CustomerDetails">
18    <xs:sequence>
19      <xs:element name="CustomerFullName" type="xs:string" />
20      <xs:element name="DateOfBirth" type="xs:string" />
21      <xs:element name="PhoneNumber" type="xs:string" />
22      <xs:element name="EmailAddress" type="xs:string" />
23    </xs:sequence>
24  </xs:complexType>
25
26  <xs:element name="CustomizedAddressChangeLetter">
27    <xs:complexType>
28      <xs:sequence>
29        <xs:element name="CustomerDetails" type="CustomerDetails"/>
30        <xs:element name="Context" type="context:Context" />
31        <xs:element name="AddressDetails" type="addresschangeletter:AddressDetails" maxOccurs="unbounded"/>
32        <xs:element name="ReceivingParty" type="offer:ReceivingParty" />
33      </xs:sequence>
34    </xs:complexType>
35  </xs:element>
36 </xs:schema>

```

19.2 JAXB Classes

JAXB classes for **AddressChangeLetter.xsd** are present inside **com.ofss.fc.integration.content** project.

Figure 18–8 JAXB Classes



For customization, JAXB classes need to be generated from customized XSD.

Figure 18–9 Generate JAXB Classes from Customized XSD

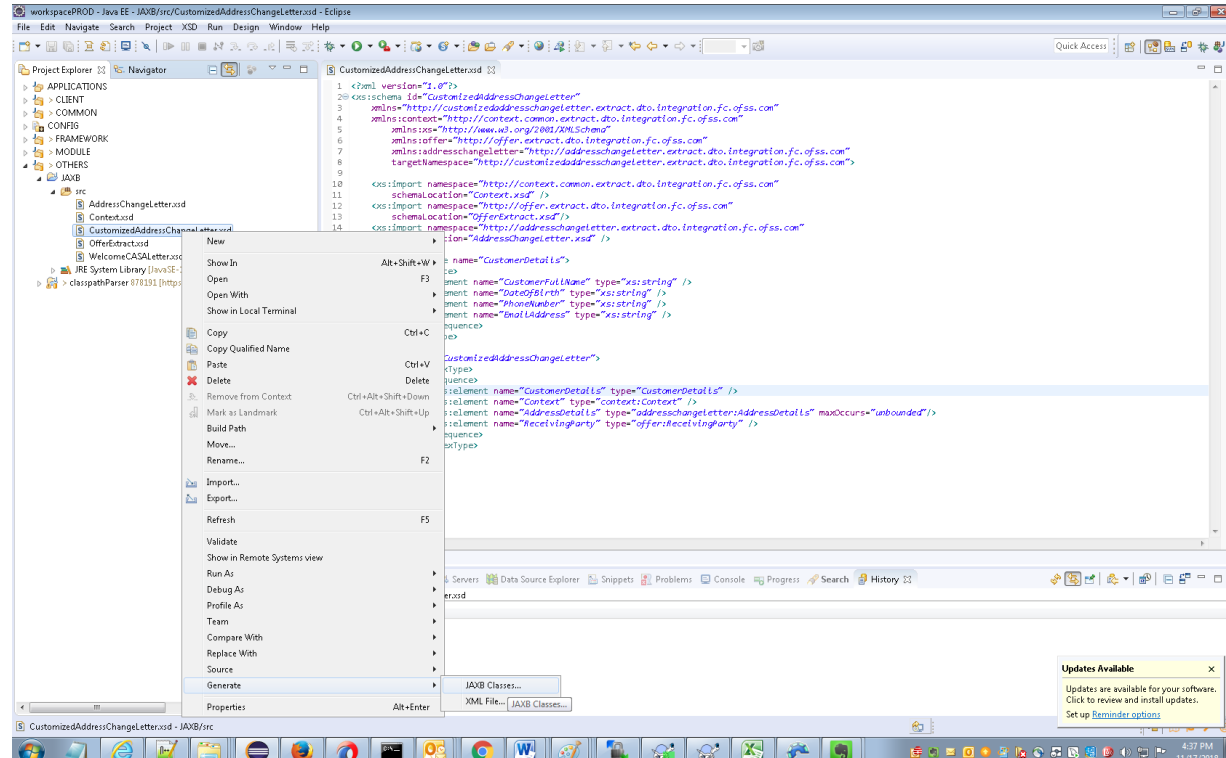
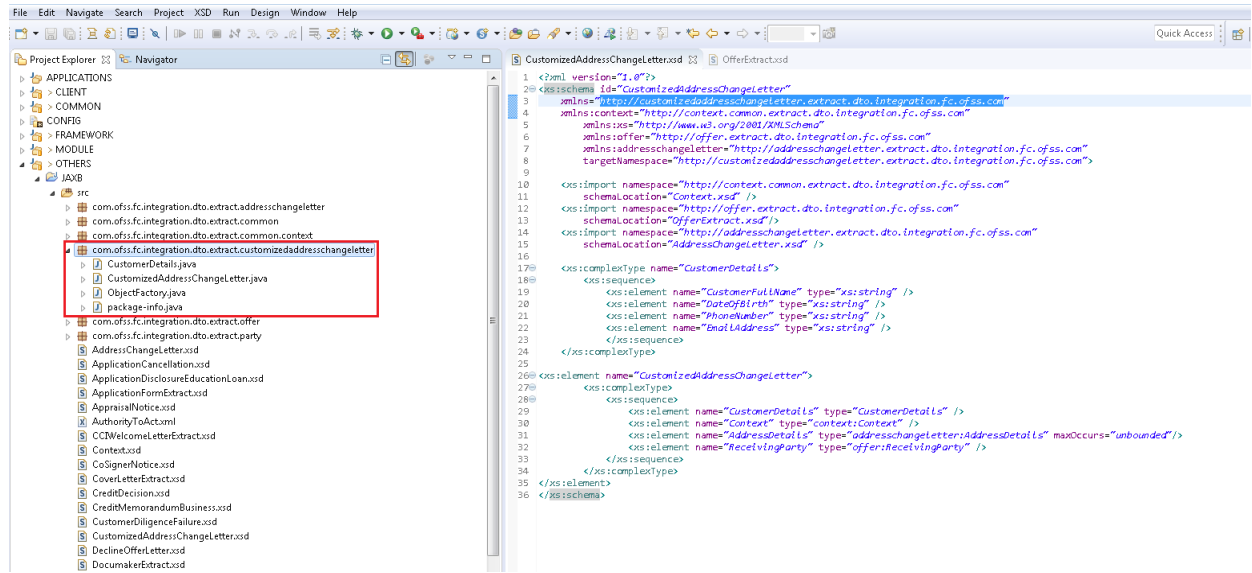
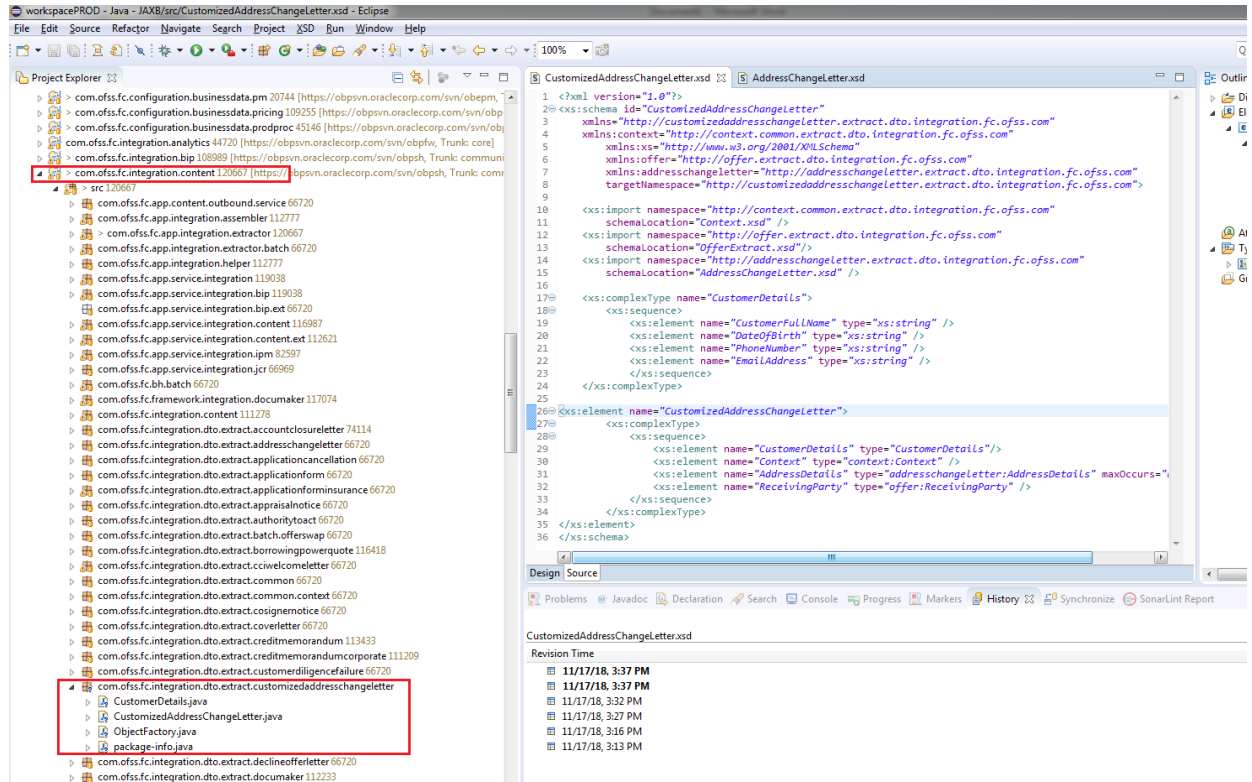


Figure 18–10 JAXB Classes Customized XSD



Once classes are generated, it should be added under same project.

Figure 18–11 JAXB Classes in Project



19.3 Extractors

Extractors are used to extract information from OBP modules. This information is stored in Generated JAXB Classes.

For example, **AddressChangeLetterDataExtractor**. In this extractor, fetch all the details from OBP modules which need to be populated in Address Change Letter and fill all the JAXB classes with those details.

Figure 18–12 AddressChangeLetterDataExtractor

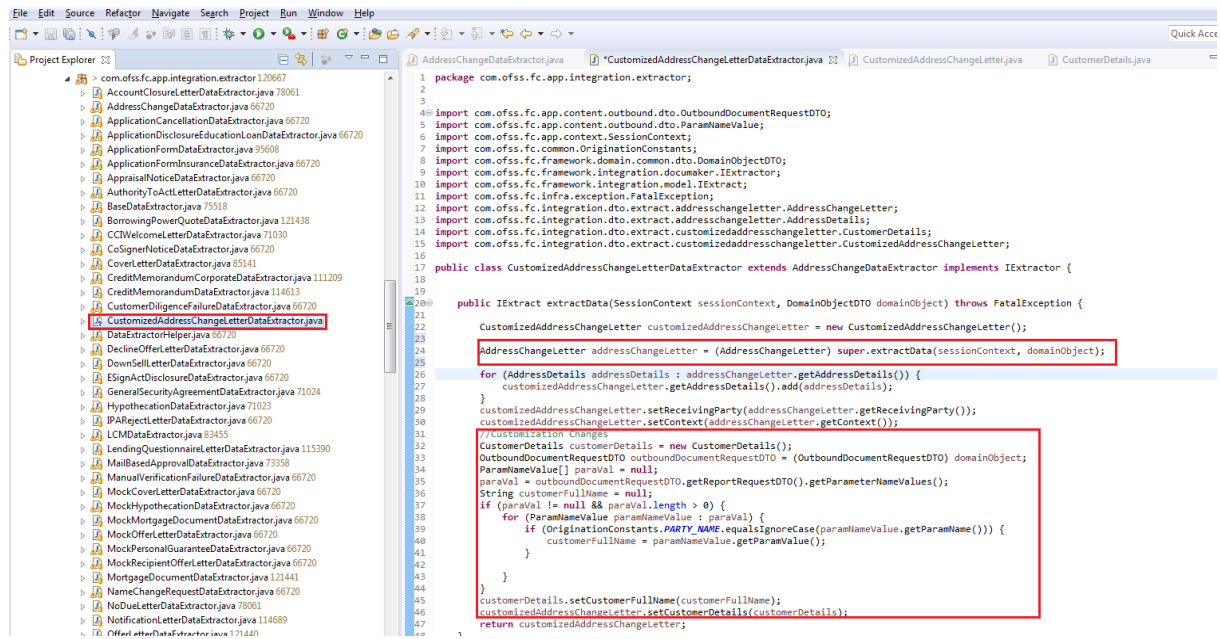
```

1 package com.ofss.fc.app.integration.extractor;
2
3 import java.util.HashMap;
4
5 /**
6  * Extractor class to populate details for Address change Letter
7  *
8  * @author vikramss
9  */
10 public class AddressChangeLetterDataExtractor implements IExtractor {
11
12     private static final String THIS_COMPONENT_NAME = AddressChangeLetterDataExtractor.class.getName();
13     private static final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
14
15     public IExtractor extractData(SessionContext sessionContext, DomainObjectDTO domainObject) throws FatalException {
16         AddressChangeLetter addressChangeLetter = new AddressChangeLetter();
17         try {
18             OutboundDocumentRequestDTO outboundDocumentRequestDTO = (OutboundDocumentRequestDTO) domainObject;
19             ParamNameValue[] paraVal = null;
20             paraVal = outboundDocumentRequestDTO.getReportRequestDTO().getParameterNameValues();
21             String oldAddress = null;
22             String newAddress = null;
23             String dispatchAddress = null;
24             if (paraVal != null && paraVal.length > 0) {
25                 for (ParamNameValue paramNameValue : paraVal) {
26                     if (PartyConstants.DOC_GEN_OLD_ADDRESS.equalsIgnoreCase(paramNameValue.getParamName())) {
27                         oldAddress = paramNameValue.getParamValue();
28                     }
29                     if (PartyConstants.DOC_GEN_NEW_ADDRESS.equalsIgnoreCase(paramNameValue.getParamName())) {
30                         newAddress = paramNameValue.getParamValue();
31                     }
32                     if (PartyConstants.DOC_GEN_DISPATCH_ADDRESS.equalsIgnoreCase(paramNameValue.getParamName())) {
33                         dispatchAddress = paramNameValue.getParamValue();
34                     }
35                 }
36             }
37             /** Check if the Receiving party is null */
38             if (StringHelper.isNull(outboundDocumentRequestDTO.getReceivingParty())) {
39                 logger.log(Level.FINE, THIS_COMPONENT_NAME + "extractData: Receiving Party Id not found.");
40                 throw new BusinessException(DocumakerConstants.DOCUMAKER_INPUT_INVALID_REC_PARTY);
41             }
42             LOWResponse prefixLOV = null;
43             IntegrationUIServiceAdapter integrationUIServiceAdapter = DataExtractorHelper.getInstance().getIntegrationUIServiceAdapter();
44             try {
45                 prefixLOV = integrationUIServiceAdapter.fetchRecordsForHQL(sessionContext,

```

If customization developer needs to add more details to Address Change Letter, then create customized extractor which extends the existing extractor (that is, **AddressChangeLetterDataExtractor**) and additional functionality can be added to new extractor as shown below.

Figure 18–13 Customized AddressChangeLetterDataExtractor



19.4 Seed Entries

This section explains the seed entries.

19.4.1 JAXB Package Entry

Append the package name of created JAXB classes in below seed entry.

Insert into FLX_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

('JAXB_OBJECT_CONTEXT','ExtractorFactory','com.ofss.fc.integration.dto.extract.common:com.ofss.fc.integration.dto.extract.common.context:com.ofss.fc.integration.dto.extract.documaker:com.ofss.fc.integration.dto.extract.offe r:com.ofss.fc.integration.dto.extract.party:com.ofss.fc.integration.dto.extract.addresschangeletter','Y','Cont ext',null,'ofssuser',to_timestamp('08-SEP-14 12.07.32.000000000 PM','DD-MON-RR HH.MI.SS.FF AM'),'ofssuser',to_timestamp('28-AUG-17 12.07.32.000000000 PM','DD-MON-RR HH.MI.SS.FF AM'),'A',1);

For example, in our case, customization developer will update this query by appending package name “com.ofss.fc.integration.dto.extract.customizedaddresschangeletter” to existing one.

Update FLX_FW_CONFIG_ALL_B

Set

PROP_

VALUE='com.ofss.fc.integration.dto.extract.common:com.ofss.fc.integration.dto.extract.common.context:
com.ofss.fc.integration.dto.extract.documaker:com.ofss.fc.integration.dto.extract.offer:com.ofss.fc.integrat
ion.dto.extract.party:com.ofss.fc.integration.dto.extract.addresschangeletter:
com.ofss.fc.integration.dto.extract.customizedaddresschangeletter'

where PROP_ID =' JAXB_OBJECT_CONTEXT';

19.4.2 Extractor Entry

When a new extractor is created, entry for the same needs to be added.

```
INSERT INTO FLX_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 ( 'ADDRESS_
CHANGE','ExtractorFactory','com.ofss.fc.app.integration.extractor.AddressChangeDataExtractor','Y','','','o
fssuser',to_date('03/12/2015 12:32:42', 'dd/mm/yyyy hh:mi:ss'),'ofssuser',to_date('03/12/2015 12:32:42',
'dd/mm/yyyy hh:mi:ss'),'A',1);
```

Customization developer needs to update this query with new extractor name.

```
UPDATE FLX_FW_CONFIG_ALL_B SET PROP_VALUE = 'com.ofss.fc.app.integration.extractor.
CustomizedAddressChangeDataExtractor' WHERE PROP_ID = 'LETTER_OF_OFFER';
```


20 Algorithm Extensions

This chapter explains the Algorithm Extensions for Oracle Banking Platform (OBP) Collections

20.1 Overview

Where the system requires a customization, OBP Collections provide for customizable algorithms. Algorithms provide a powerful and flexible way of extending applications. Base algorithms exist, but can be cloned and modified. Unlike Change Handlers, they are more related to the business functions and events. Also, unlike Change Handlers, they use configurable ("soft") parameters. At upgrades, custom algorithms will not be overwritten.

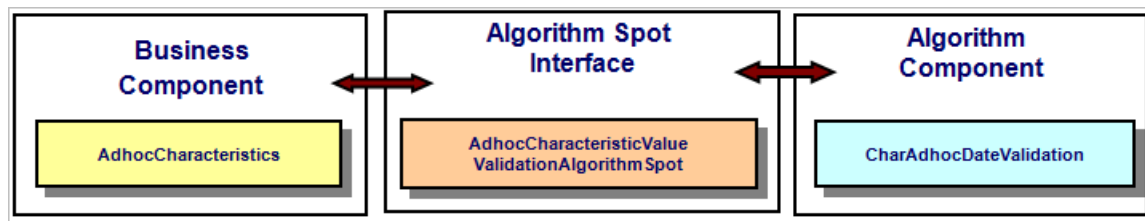
Algorithms are defined in 2 places:

- Database tables: The online Admin menu is used to define the following database components:
 - Algorithm Types
 - Algorithms
 - The event or activity to which the algorithm applies (For example, Characteristics, Date validations, and so on.)
- Framework: The framework requires the implementation class, that is the program that contains the logic, and various generated artifacts.

20.2 Algorithm Spots

The call out places in the system (For example, Date validation for ad hoc characteristics) are known as algorithm spots. Each algorithm spot has an interface class. Communication with an algorithm takes place through the interface. An interface provides abstraction between the base and the customization.

Figure 19–1 Algorithm Spot Interface

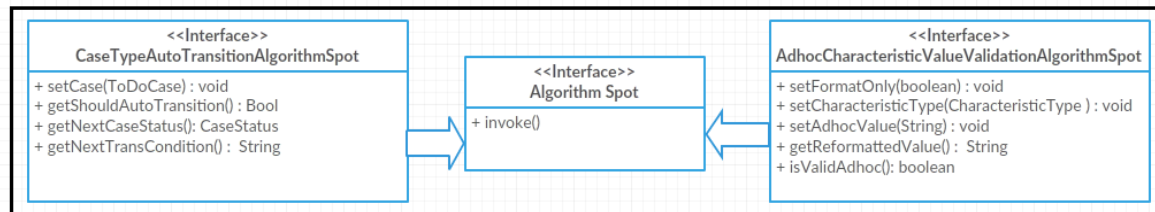


Attributes of an algorithm spot interface class:

- The API to the algorithm component (from the base application).
- It is specific to the algorithm entity type (or system event).
- It defines the hard input parameters for an algorithm. These are the parameters associated with a specific event.
- It defines the output parameters that can be retrieved after the algorithm has been invoked.

- It also specifies the schema defined for a plug-in script.
- It is invoked from the base code at appropriate times (events).
- The methods on the interface are related to the algorithm type. For example, `setAdhocValue` (String value) is only relevant to `AdhocCharacteristicValueValidationAlgorithmSpot`.

Figure 19–2 Example: Algorithm Spot Interface

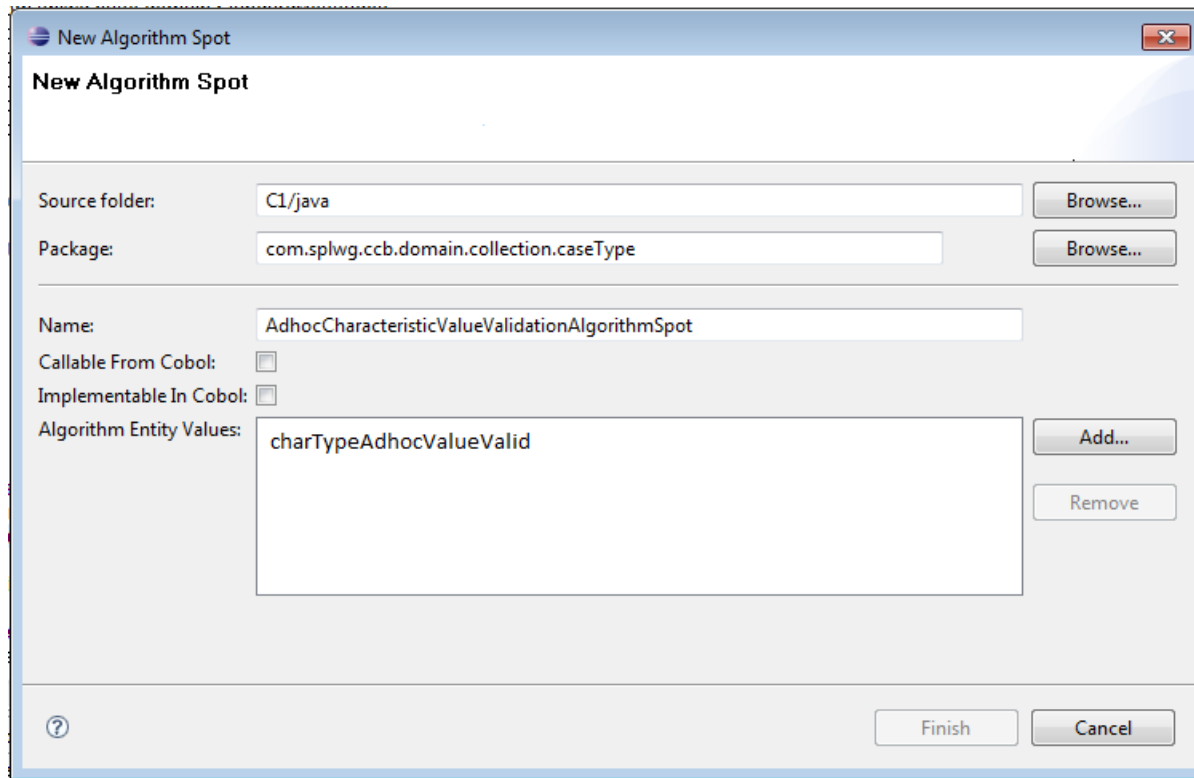


Adding Algorithm Spots:

- Algorithm Spots reference an `AlgorithmEntityLookup` (ALG_ENTITY_FLG) value, so a new lookup value must be added to correspond to the new spot.
- Add an interface that defines the spot using the `@AlgorithmSpot` annotation.
- Properties include:
 - `algorithmEntity`: One or more `AlgorithmEntity` values corresponding to the lookup value described above.
 - `calledFromCobol`: A boolean attribute that lets the framework know if inbound call support is to be supported from COBOL.
 - `implementableInCobol`: A boolean attribute that lets the framework know if it must be able to call an algorithm implemented in COBOL.
- Extend the `AlgorithmSpot` interface.
- Wire up the call to the spot by accessing the `AlgorithmComponent` via `Algorithm.getAlgorithmComponent(...)`

Example: See `AdhocCharacteristicValueValidationAlgorithmSpot`

Figure 19–3 Example: Adding New Algorithm Spot



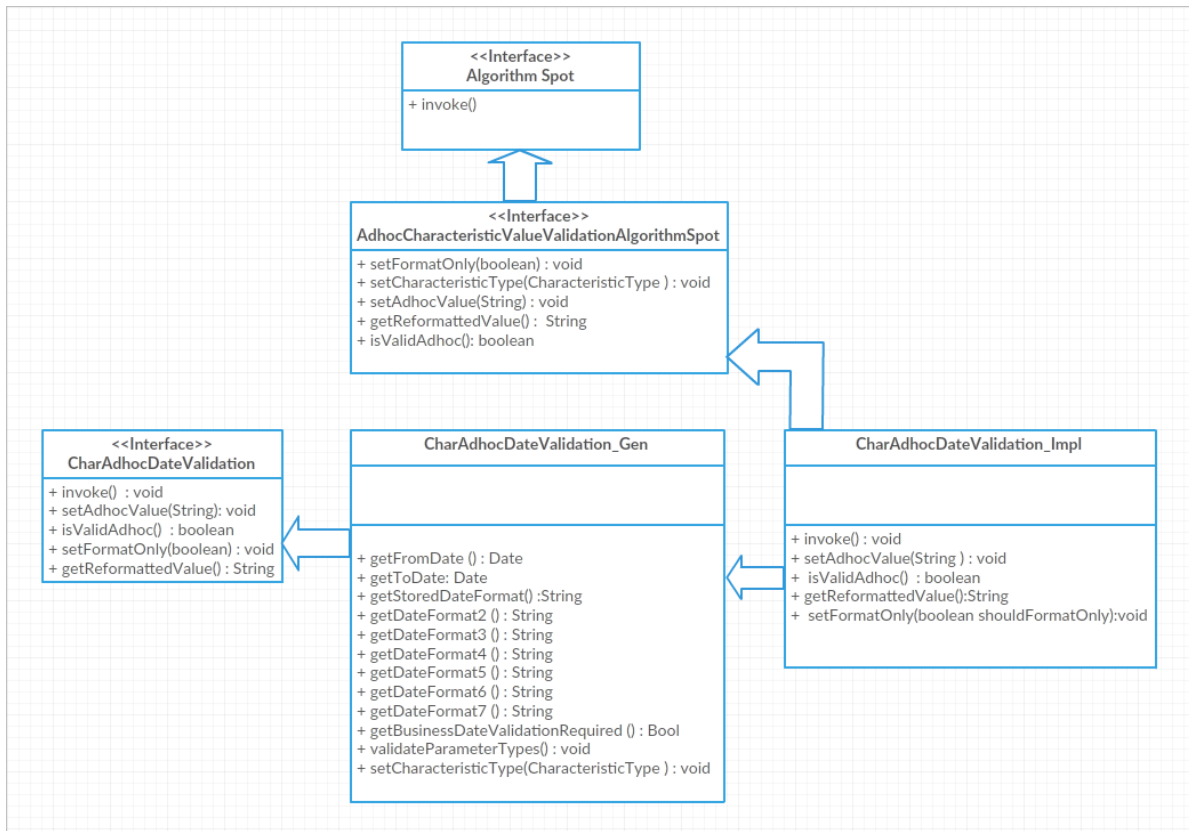
20.3 Algorithm Components

An algorithm requires a programmatic implementation. The Algorithm Type definition carries the program name, for example `com.splwg.ccb.domain.collection.caseType.CharAdhocDateValidation`. This name in fact specifies another interface which is generated from the implementation class. The implementation class name = the interface name + `Impl`, for example `com.splwg.ccb.domain.collection.caseType.CharAdhocDateValidation_Impl`.

The following diagram describes the Date Validation algorithm component. Remember:

- An interface is empty. It requires an implementation to perform appropriate tasks.
- The implementation for an algorithm spot is an Algorithm Component, that is Business Component.

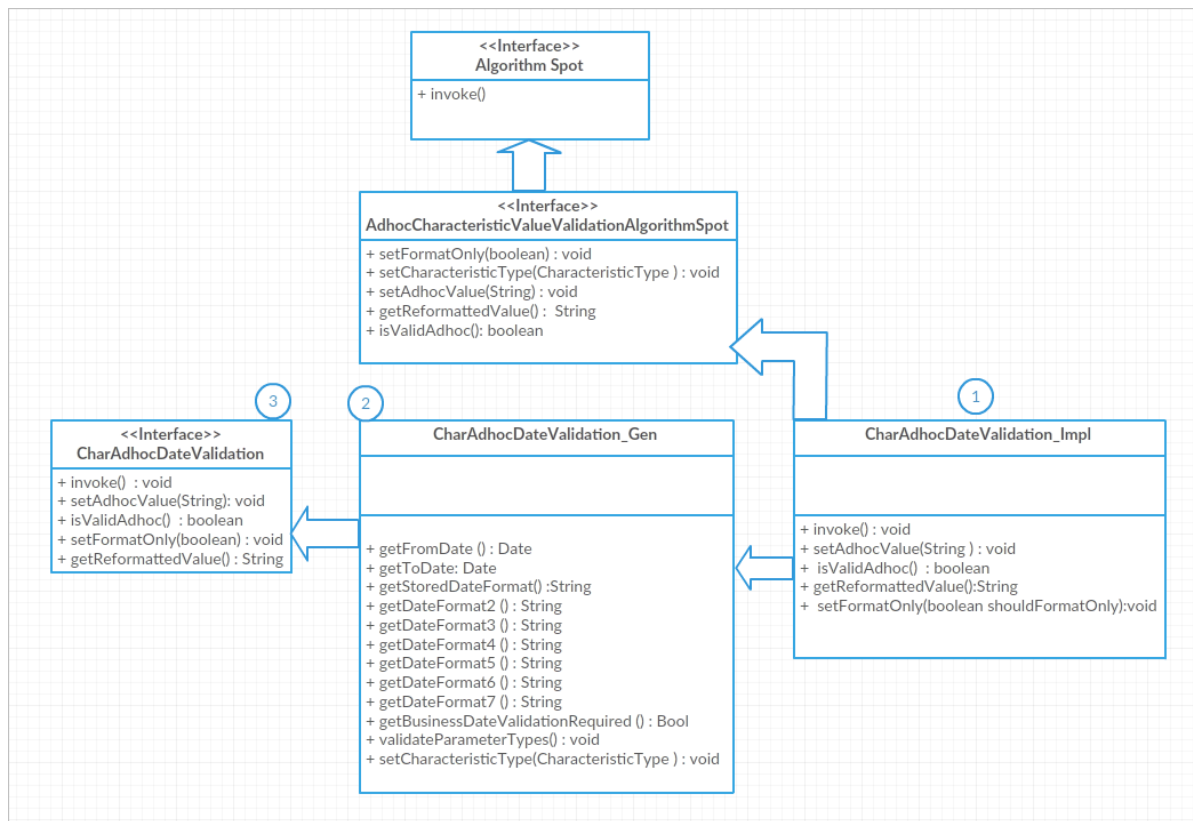
Figure 19–4 Example: Data Validation Algorithm Component



Algorithm Components Example - CharAdhocDateValidation

The following diagram presents an example of the CharAdhocDateValidation algorithm component.

Figure 19–5 Example: CharAdhocDateValidation



The annotations marked in the above diagram are explained as follows:

- The implementation class (`CharAdhocDateValidation_Impl`) is hand-coded - it can be customized.
- The component interface is generated by the artifact generator - a customized version will be generated for a custom impl class. The `_Gen` class (`CharAdhocDateValidation_Gen`) has the methods for the soft parameters (as specified on the Algorithm Type definition). Note: These are generated from the annotations in the `_Impl` class.
- An algorithm is invoked via its component interface (`CharAdhocDateValidation`).

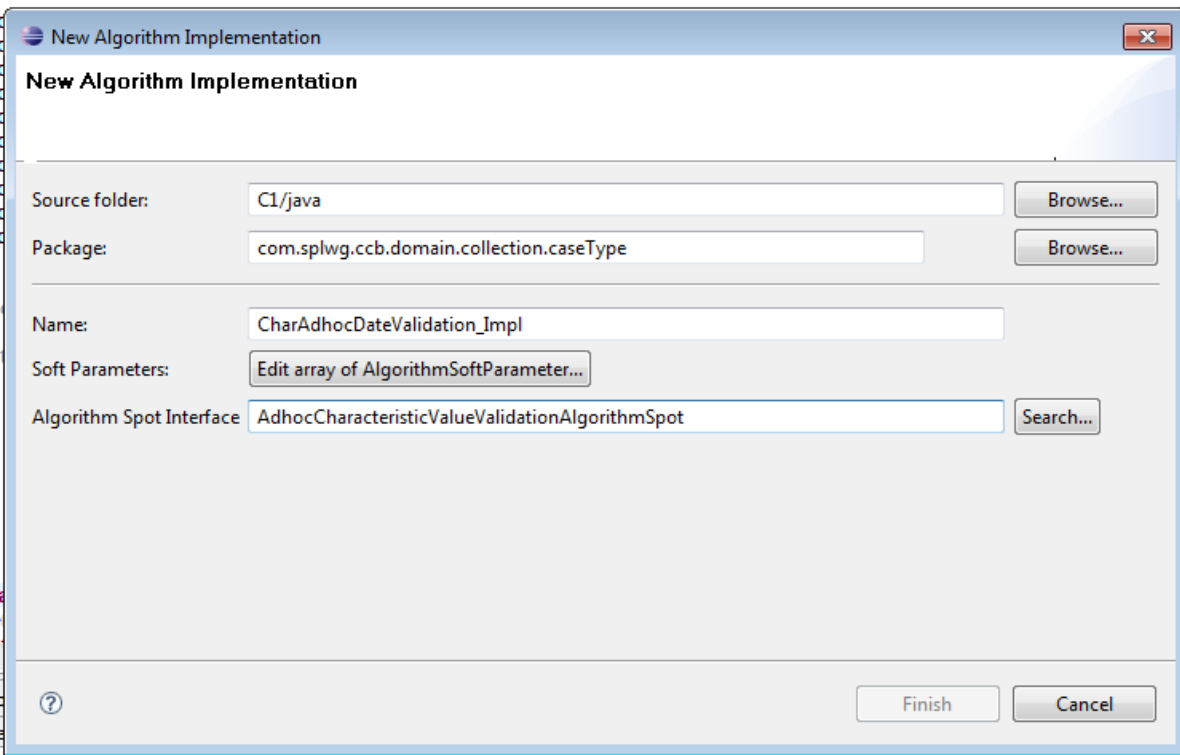
Algorithm Implementation Class:

The base versions of all algorithms are provided. To create a new one, it is easiest to duplicate the appropriate base one if it exists and modify it.

The basic Java elements of a new algorithm are:

- An `_Impl` class, the hand-coded implementation class that contains the logic
- A `_Gen` class, the implementation class for the "soft" parameters, generated by the artifact generator
- A component interface class, generated by the AG
- A message method, if required

Figure 19–6 New Algorithm Implementation



Adhoc Characteristic Date Validation Example:

The following diagram presents an example of Adhoc Characteristic Date Validation.

Figure 19–7 Adhoc Characteristic Date Validation

```

/**
 ①  @AlgorithmComponent (softParameters = { @AlgorithmSoftParameter (name = fromDate,
      type = date)
      *   , @AlgorithmSoftParameter (name = toDate, type = date)
      *   , @AlgorithmSoftParameter (name = storedDateFormat, required =
      *   true, type = string)
      *   , @AlgorithmSoftParameter (name = dateFormat2, type = string)
      *   , @AlgorithmSoftParameter (name = dateFormat3, type = string)
      *   , @AlgorithmSoftParameter (name = dateFormat4, type = string)
      *   , @AlgorithmSoftParameter (name = dateFormat5, type = string)
      *   , @AlgorithmSoftParameter (name = dateFormat6, type = string)
      *   , @AlgorithmSoftParameter (name = businessDateValidationRequired,
      *   type = boolean)})
  */
public class CharAdhocDateValidation_Impl
    extends CharAdhocDateValidation_Gen
    implements AdhocCharacteristicValueValidationAlgorithmSpot {
    ②
    ③
    ④

```

The annotations marked in the above diagram are explained as follows:

1. The soft parameters expected by the algorithm - these correspond with the Algorithm Type parameter definitions
2. Has an Algorithm Component name (as specified on the Algorithm Type) + "_Impl"
3. Extends the "_Gen" class - the "_Gen" class is generated by the Artifact Generator
4. Implements the base Algorithm Spot class for the algorithm type

Algorithm Spot interface methods that are implemented in _Impl class:

The following diagrams present the Algorithm Spot interface methods that are implemented in _Impl class.

Figure 19–8 Algorithm Spot Interface Methods

```

/**
 * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setAdhocValue(java.lang.String)
 */
1 public void setAdhocValue(String value) {
    adhocValue = value;
}

/**
 * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#isValidAdhoc()
 */
2 public boolean isValidAdhoc() {
    return isValid;
}

/**
 * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#getReformattedValue()
 */
3 public String getReformattedValue() {
    return newFormattedValue;
}

/**
 * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setFormatOnly(boolean)
 */
4 public void setFormatOnly(boolean shouldFormatOnly) {
    formatOnly = shouldFormatOnly;
}

```

Figure 19–9 Algorithm Spot Interface Methods (continued)

```

/**
 * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#invoke()
 */
5 public void invoke() {
    logger.debug("Executing adhoc date validation");
    if (adhocValue == null || adhocValue.trim().length() == 0) {
        isValid = true;
        newFormattedValue = "";
        return;
    }
    initializeFormats();
    DateTime date = parseDateFromAnyFormat();
    if (date == null) {
        isValid = false;
    }

    for (Iterator iter = validFormats.iterator(); iter.hasNext();) {
        DateFormat format = (DateFormat) iter.next();
        if (format != null) {
            notNullValidFormats.add(format);
        }
    }
    ServerMessage message = CustomMessageRepository.invalidFormatForDate(adhocValue, notNullValidFormats,
        notNullValidFormats.size());
    addError(message);

    return;
}
isValid = true;

if (!formatOnly && adhocValue.length() <= Date.TO_STRING_SIZE) {
    validateDateInRange(date.getDate());
}

if (getBusinessDateValidationRequired().isTrue()) {
    if (!DateUtility.isWorkingDay(date.getDate()))
        addError(CustomMessageRepository.businessDateValidationFailed(date.getDate().toString()));
}

newFormattedValue = storedFormat.format(date);

```

The annotations marked in the above diagrams are explained as follows:

1. This method is invoked by the business component to set the hard parameters. This sets the char value to validate. It is stored here for use later.
2. This returns a true/false to indicate the validity of the date characteristics.
3. This returns the reformatted value.
4. This method set the required format.
5. The invoke () method is called to validate and format the date.

Generated artifacts that are based on the `_Impl` class annotation:

- The `_Gen` class has the methods for the soft parameters
- The `_Impl` class calls these methods to get the soft parameter values, as set on the Algorithm definition

Figure 19–10 Generated Artifacts

```

* Generated by com.splwg.tools.artifactgen.ArtifactGenerator[]
package com.splwg.ccb.domain.collection.caseType;

import com.splwg.base.api.BusinessComponent[];

/**
 * Interface for the charAdhocDateValidation component
 *
 * @author Generated by com.splwg.tools.artifactgen.ArtifactGenerator
 */
public interface CharAdhocDateValidation extends BusinessComponent
    , AdhocCharacteristicValueValidationAlgorithmSpot
    {

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#invoke()[]
    void invoke();

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setAdhocValue(java.lang.String)[]
    void setAdhocValue(java.lang.String value);

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#isValidAdhoc()[]
    boolean isValidAdhoc();

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#getReformattedValue()[]
    java.lang.String getReformattedValue();

    /**
     *
     * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setFormatOnly(boolean)
     */
    void setFormatOnly(boolean shouldFormatOnly);
}

```

- The component interface defines the required methods for the `_Impl` class as viewed from the application (the business component).

Figure 19–11 Generated Artifacts

```

/* Generated by com.splwg.tools.artifactgen.ArtifactGenerator[]
package com.splwg.ccb.domain.collection.caseType;

import com.splwg.base.api.BusinessComponent[];

/**
 * Interface for the charAdhocDateValidation component
 *
 * @author Generated by com.splwg.tools.artifactgen.ArtifactGenerator
 */
public interface CharAdhocDateValidation extends BusinessComponent
    , AdhocCharacteristicValueValidationAlgorithmSpot
    {

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#invoke()[]
    void invoke();

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setAdhocValue(java.lang.String[])
    void setAdhocValue(java.lang.String value);

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#isValidAdhoc()[]
    boolean isValidAdhoc();

    * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#getReformattedValue()[]
    java.lang.String getReformattedValue();

    /**
     *
     * @see com.splwg.base.domain.common.characteristicType.AdhocCharacteristicValueValidationAlgorithmSpot#setFormatOnly(boolean)
     */
    void setFormatOnly(boolean shouldFormatOnly);
}

```

The steps to create a new algorithm Impl class are:

1. Determine the Algorithm Spot interface name. The Javadocs can be used for this.
2. Create the "_Impl" class, implementing the appropriate Algorithm Spot interface.
3. Add default implementations for all the Algorithm Spot methods (For example, using the Eclipse *Source, Override/implement Methods...* menu item).
4. Code the annotation.
5. Run the Artifact Generator to create the "_Gen" and component interface classes.

In Eclipse, you must refresh the project after this.

The steps to create a new algorithm (Admin UI) are:

1. Create the Algorithm Type and attached with algorithm component.

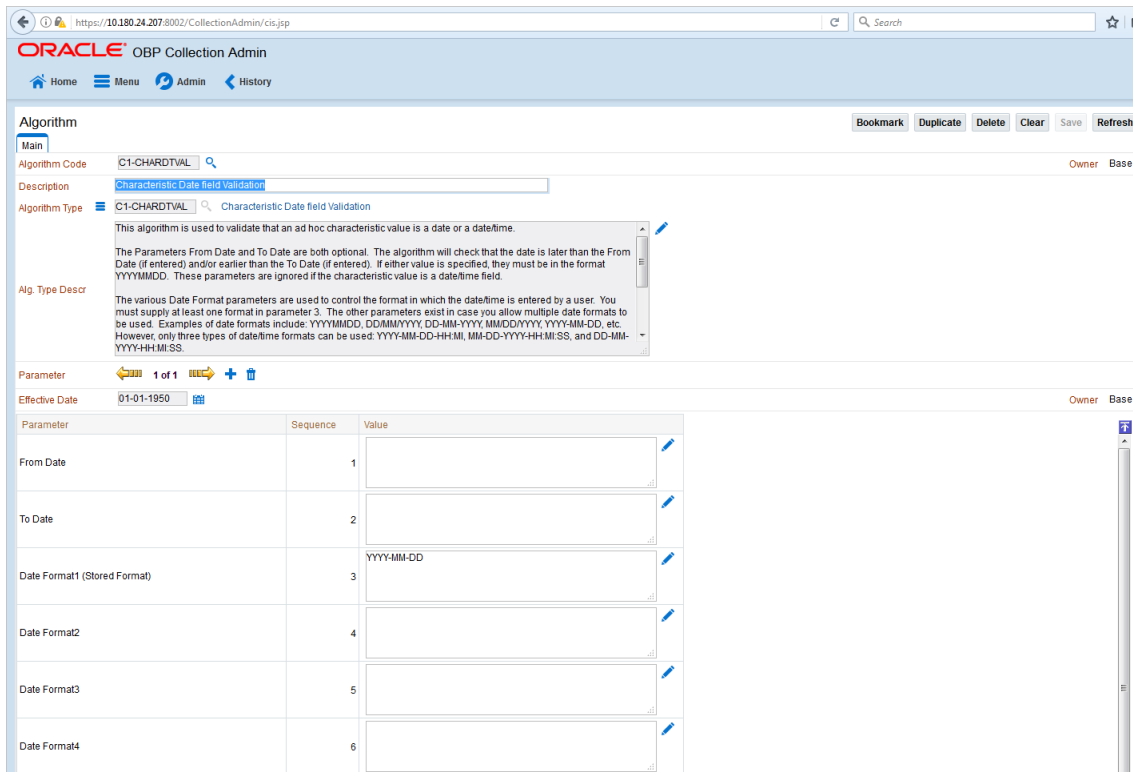
Figure 19–12 Create Algorithm Type

The screenshot shows the Oracle OBP Collection Admin interface for creating an Algorithm Type. The page title is 'Algorithm Type' and the current type is 'C1-CHARDTVAL'. The 'Description' field contains the text: 'Characteristic Date field Validation'. Below this, a 'Detailed Description' field provides more information: 'This algorithm is used to validate that an ad hoc characteristic value is a date or a datetime. The Parameters From Date and To Date are both optional. The algorithm will check that the date is later than the From Date (if entered) and/or earlier than the To Date (if entered). If either value is specified, they must be in the format YYYYMMDD. These parameters are ignored if the characteristic value is a datetime field. The various Date Format parameters are used to control the format in which the datetime is entered by a user. You must supply at least one format in parameter 3. The other parameters exist in case you allow multiple date formats to be used. Examples of date formats include: YYYYMMDD, DDMMYYYY, DD-MM-YYY, MMDDYYYY, YYYY-MM-DD, etc. However, only three types of datetime formats can be used: YYYY-MM-DD-HH:MM:SS, MM-DD-YYYY-HH:MM:SS, and DD-MM-YYYY-HH:MM:SS.' The 'Algorithm Entity' is set to 'Characteristic Type - Adhoc Validation' and the 'Program Type' is 'Java'. The 'Program Name' is 'com.spilwg.ccb.domain.collection.caseType.CharAdhocDateValidation'. Below the form is a table of parameters:

Sequence	Parameter	Required	Owner
1	From Date	<input type="checkbox"/>	Base
2	To Date	<input type="checkbox"/>	Base
3	Date Format1 (Stored Format)	<input checked="" type="checkbox"/>	Base
4	Date Format2	<input type="checkbox"/>	Base
5	Date Format3	<input type="checkbox"/>	Base
6	Date Format4	<input type="checkbox"/>	Base
7	Date Format5	<input type="checkbox"/>	Base
8	Date Format6	<input type="checkbox"/>	Base
9	Business Date Validation Required	<input type="checkbox"/>	Base

2. Attach the algorithm to the Algorithm Type and test.

Figure 19–13 Attach Algorithm



20.4 List of Algorithm Spots

The detailed list of algorithm spots which can be used for extending and customizing the product are listed in the OBP Host Extensibility Guide - Collections Algorithm Spots document.