Oracle FLEXCUBE Direct Banking

System Handbook – Volume IV – Business Services Layer
Release 12.0.2.0.0

Part No. E50108-01

September 2013

ORACLE®

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India
Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax:+91 22 6718 3001
www.oracle.com/financialservices/

Contents

1. Preface

1.1. Intended Audience

This System Handbook (Volume IV – Business Services Layer) is intended for the following audience:

- Application Architects
- End to End Designers
- Business Service Detailed Designers and Developers
- Implementation Partners

1.2. Documentation Accessibility

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

1.3. Access to OFSS Support

https://flexsupp.oracle.com/

1.4. Structure

This document, termed Oracle FLEXCUBE Direct Banking System Handbook, is a single reference for the product information which can be managed, configured, extended, by external parties, to implement, customize or rollout the product to a financial institution.

This is not an Implementation Guide but a System Handbook to explain low level details of how certain key features are implemented within the solution and how these could be extended, customized as appropriate to meet the requirements of the implementation.

This document is intended to provide a set of principles, guidelines and parameters for configuration and extending Oracle FLEXCUBE Direct Banking to meet the . As such, this document does not go into detail regarding the context and background of a number of design decisions but explains the extensibility features and provides insight into the design guidelines and principles for external parties to leverage and develop the required extensions in a non invasive way to the primary features and functionality of the application.

This document is segregated into five Volumes

1	Volume I – Core and Architecture
2	Volume II – Presentation Layer
3	Volume III – Channel Layer
4	Volume IV – Business Service Layer
5	Volume V – Host Interfacing Layer
6	Volume VI – Origination and Peer-to-Peer Payments

1.5. Related Information Sources

For more information on Oracle FLEXCUBE Direct Banking Release 12.0.2.0.0, refer to the following documents:

- Oracle FLEXCUBE Direct Banking System Handbook – Volume I
- Oracle FLEXCUBE Direct Banking System Handbook – Volume II
- Oracle FLEXCUBE Direct Banking System Handbook – Volume III
- Oracle FLEXCUBE Direct Banking System Handbook – Volume V
- Oracle FLEXCUBE Direct Banking System Handbook – Volume VI

2. About This Document

2.1. Glossary Of Terms

The following terms are some of the key terms used within the document for identifying the actor for the various actions mentioned within this document.

2.1.1. LICENSEE

The LICENSEE is the Financial Institution, Application Services Provider or the Bank which has licensed the Oracle FLEXCUBE Direct Banking application and shall rollout the solution to its customers as an internet and / or mobile banking channel.

2.1.2. IMPLEMENTER

The IMPLEMENTER is the Implementation Partner, Vendor, Application Service Provider or the LICENSEE themselves who is responsible for rolling out, configuring, extending or developing on Oracle FLEXCUBE Direct Banking.

2.2. TERMINOLOGY

The following terms and terminology is used within the documents to explain underlying processes, components, actions, actors etc.

Term	Definition
Business Service	A Business Service or a Transaction Service is a coarse-grained component that delivers a particular service contract. The Service Interfaces and that make up the contract are each implemented by their particular Service Endpoints.
POJO	A Plain Old Java Object (POJO) is exactly what it says. The term is used to differentiate these simple objects from more specific or complex types such as EJB classes. For example, when creating an EJB, a specific class must implement the SessionBean interface. However, that class will often delegate much of its functionality to one or more POJOs to aid maintainability and reuse of functionality.
Service Implementation or Service Endpoint	A Service Implementation is a concrete implementation of a Service Interface.
Service Interface	A Service Interface is a cohesive set of Service Methods that are grouped together in the anticipation that they will be commonly used together by a

	consumer. For example, the Service Interface for the FundsTransferService would contain a set of Service Methods that perform different types of immediate money transfer between two accounts.
Service Method	A Service Method takes the form of a Java method implemented by the Service Implementation and the Service Delegate. The consumer of the service will invoke one or more Service Methods to help perform part of a business process.
Extension Schema	The **Extension Schema** is a term used for the separate database schema as deployed by Oracle FLEXCUBE Direct Banking to allow IMPLEMENTERS to extend the Oracle FLEXCUBE Direct Banking application as per their needs.

2.3. Abbreviations

Acronyms	Description
FCDB / FC DB / FC Direct Banking / Direct Banking	Oracle FLEXCUBE Direct Banking
Java EE / JEE	Java Enterprise Edition
Java SE / JSE	Java Standard Edition
Java ME / JME	Java Mobile Edition
DBA	Database Administrator
XML	Extensible Markup Language
XSL	XML Stylesheets
TCP	Transmission Control Protocol
HTTP	Hypertext Transmission Protocol
HTTPS	Secured Hypertext Transmission Protocol
SSL	Secured Socket Layer
IDS	Intrusion Detection System

2.4. Conventions

- The diagrams and / or text in this document may contain colour to communicate or highlight additional information. However, the content of this document is retained when rendered without colour. Specific references to colour can be ignored if necessary.

- The technical terminology relating to the Oracle FLEXCUBE Direct Banking solution is aligned as much as possible to standard definitions or should be defined in the Glossary of Terms. Any deviations from standard terminology are either noted in the Terminology Section, or in context of usage.

- Some sections may contain additional notes and caveats included with the body text. For general and contextual information, these notes are contained within document footnotes. Any notes that have important implications or detailed recommendations are denoted by the information symbol (ⓘ). Important caveats are denoted with the warning symbol (💣).

3. Business Tier

The Business Tier is a collection of services which are one atomic transaction by default. Each service is a clearly defined REQUEST-RESPONSE pair which defines the transaction performed for the services.

Services are reusable and can be linked to each other within the same transaction context to support complex business functions. The business services provide a consistent experience to all channels within the solution and can be extended to adapt to specific variations to the requirements.

4. Business Service Layer – Extensions and Configurations

4.1. Validation Engine

The Validation Services or the Validation Engine as it is commonly referred is the key feature of Oracle FLEXCUBE Direct Banking. This engine provides support for request and data field validations to be put in place for any requests that are processed by Oracle FLEXCUBE Direct Banking. This includes the web based requests originated as part of the Oracle FLEXCUBE Direct Banking internet and mobile banking solution or any other requests processed as a part of business integration services offered by Oracle FLEXCUBE Direct Banking.

The Validation Engine provides a pluggable and configurable mechanism to verify for data integrity checks and validations to be performed before the request is passed on to the actual business processing layer which completes the transaction. This ensures that data elements are strongly typed and validated against a pre-defined set of rules as well as the ability to plug-in custom validation rules for verification of any specific field values.

The Validation Engine also acts as a security mechanism by which data could be introspected for specific data patterns that can results in SQL injection or Cross Scripting attacks or any other security threats. The Validation Engine segregates the validation functions from the business processing layer allows flexibility and extensibility to modify the required validation rules without impacting or with a minimal impact to the business processing layer.

ⓘ The Validation Engine is one of the Core Services in the Business Tier and provides extensibility features for validations to be adapted, modified, updated and custom configured for specific installations and implementation of Oracle FLEXCUBE Direct Banking.

The Validation Engine allows definition of Data Types to be used for validations. All data elements should be defined as part of a Data Dictionary with an appropriate Data Type association.

The key features of the Validation Engine are

- Data Type validations allowing each incoming field to conform to a predefined data type.
- Data Length validations using the minimum and maximum lengths defined for the data field.
- Data Integrity checks disallowing invalid characters in the input (e.g: Special Characters allowed for String values)
- Data Input Format Checks by allows certain data to be imported only in the predefined format (e.g: Date Formats etc.)
- Providing parsed values to the processing components. The processing components have an option of using some the pre-parsed fields, if required. (e.g: Date objects etc). This also allows for data enrichment using the validation engine if required.
- Provides pre-processing capability before the request is handed over to the business processing components.
- Validations for all data fields before error is returned allowing all errors to be returned in a single validation call rather than returning separate errors.
- Validations against enumeration of values to allow only a certain set of valid values for a given data field. The Validation Engine supports enumerations to check for a list of valid values.
- One or more Custom Validation can be plugged-in to allow for additional functional validations over and above the required data verification checks.

4.1.1. Data Types

Data Types identify the types of the various data field elements in a request. The data types are defined in the database associated with an appropriate Java Validator component. The addition of a new data type allows extending the default capabilities of the solution requiring lesser site specific components for validations.

The standard data types supported by the Oracle FLEXCUBE system are defined in the `DATA_DICTIONARY_TYPES` table.

Oracle FLEXCUBE @ supports a list of standard data types to be used and have the flexibility to support custom data types if new data types

E.G: The Account Number can be treated as a new data type with specific validator associated with the data type to only validate for the rules for account numbers. Similarly, Credit Card can be a new data type defined as part of the data dictionary types.

Each validator also returns a typed object, appropriate for the data type, after a successful validation which is the parsed form of the data that has been received.

Example:

For a DATE data field, the DATE validator shall return an instance of `java.lang.Date` representing the date value after validating the incoming string date value.

The following data types are currently supported in Oracle FLEXCUBE @.

ALPHABETS (A)

This data type allows values which contain only Alphabets in upper case and lower case. Special characters and Numeric digits are not allowed for this data type.

The FIELDFORMAT field is not used for data fields with the ALPHABETS data type.

STRING (S)

This data type allows values which contain Alphabets and Numeric Digits. Special characters are not allowed, by default, for this data type.

The FIELDFORMAT field for the specific data element definition can be used to indicate the special characters to be allowed with the Upper Alphabets and the Numeric Digits. The special characters including space should be listed a continuous string of characters in the FIELDFORMAT field for the data field definition in TXN_DATA database table.

The validator for the STRING data type returns the actual value as a String value as the typed object.

NUMERIC (N)

This data type allows all numeric values for a given data field. This includes all integer and floating point values.

The FIELDFORMAT field is not used for data fields with the NUMERIC data type.

The validator for the NUMERIC data type returns the actual value as a java.math.BigDecimal value as the typed object.

FLOATING POINT (N)

This data type allows all numeric values for a given data field. This includes all integer and floating point values.

The functionality is the same as the NUMERIC data type but the only difference is the FLOATING POINT data type validator returns an instance of java.lang.Double as a typed object than a java.math.BigDecimal instance returned by the Numeric Validator.

The FIELDFORMAT field is not used for data fields with the FLOATING POINT data type.

The validator for the FLOATING POINT data type returns the actual value as a java.lang.Double value as the typed object.

INTEGER (N)

This data type allows all numeric integer values for a given data field.

The FIELDFORMAT field is not used for data fields with the INTEGER data type.

The validator for the INTEGER data type returns the actual value as a java.lang.Long value as the typed object.

EMAIL (E)

This data types is used to validate for data fields used for email addresses. The validator checks for the required valid fields (e.g: @ and .) at appropriate locations and allows only characters allowed as per the SMTP specification in the email address.

The FIELDFORMAT field is not used for data fields with the EMAIL data type.

The validator for the EMAIL data type returns the actual value as a String value as the typed object.

DATE (D)

The date type supports validations for date fields. The default incoming date format is expected to be dd/MM/yyyy.

The FIELDFORMAT field for a specific data element can be used to override the default format and provide the required incoming date format.

The validator for the DATE data type returns the actual value as a java.util.Date instance as a typed object.

TIMESTAMP (T)

The date type supports validations for date time fields. The default incoming date format is expected to be dd/MM/yyyy.

The `FIELDFORMAT` field for a specific data element can be used to override the default format and provide the required incoming date time format.

This data type is exactly similar to the `DATE` data type but differs only in the instance of the typed object returned by the validator. The validator for the `TIMESTAMP` data type returns the actual value as a `java.sql.Timestamp` instance while the DATE validator returns an instance of java.util.Date.

UPPERCASE ALPHABETS (UA)

This data type allows values which contain only Alphabets in UpperCase. Special characters and Numeric digits are not allowed for this data type.

The FIELDFORMAT field is not used for data fields with the UPPERCASE ALPHABETS data type.

The validator for the UPPERCASE ALPHABETS data type returns the actual value as a String value, as a typed object.

UPPERCASE STRING (US)

This data type allows string values which contain Alphabets in UpperCase only along with numeric digits. Special characters are not allowed by default for this data type.

The other properties are the same as the STRING validator wherein the `FIELDFORMAT` field can be used to indicate the special characters to be allowed with the Upper Alphabets and the Numeric Digits.

The validator for the UPPERCASE STRING data type returns the actual value as a String value, as a typed object.

POSITIVE NUMBER (PN)

This data type supports only positive integral values (Numeric Values > 0). Numeric values less than zero shall result in a validation error.

The `FIELDFORMAT` field for a given data field of base type POSITIVE NUMBER can be used to identify the maximum value that can be accepted by the data field. The integer portion of the value shall be used, if the value is provided as a decimal.

Example:

If the FIELDFORMAT field for a Reference Number data field indicated as a POSITIVE NUMBER contains the values 9999, the value greater than 9999 would result in a validation error.

The validator for the POSITIVE NUMBER data type returns the actual value as a `java.lang.Long` value, as a typed object.

FREETEXT (FT)

This data type allows all characters to be accepted for any given data field including all special characters. The validator associated with this data type is a flow through validator which does not perform any specific data validations. The `FIELDFORMAT` field is not used for data fields with the `FREETEXT` data type. The validator for the `FREETEXT` data type returns the actual value as a String value, as a typed object.

Please refer Appendix A for the Data Types and their corresponding Validators.

4.1.2. Data Dictionary

The Data Dictionary is the master set of data elements used with the application. The Data Dictionary identifies the data type of the base data element and also the attributes like the minimum and maximum length of the data.

The Data Dictionary should be defined prior to defining the Validation Templates. Data Dictionary elements can be reused for multiple fields hence applying common validations to fields across various requests

4.1.3. Validation Templates

The Validation Templates identify the request and its associated fields for any given request. The Validation Templates are defined in the `TXN_DATA_MASTER` and the `TXN_DATA` database tables.

The template identifies the request data fields and tags them to the Data Dictionary and the Data Types defined. The template also identifies whether a field is mandatory in the request or not and also whether a field requires data validation or not.

The Validation Template Identifier or the Request Identifier should be mapped to the ATS request definition in the `MSTATS` table for the Validation Engine to invoke the validation against the Validation Template defined. The `REQ_TEMPLATE_ID` in `the MSTATS` table should contain the required Validation Template Identifier for any given request for the Validation Engine to be used.

Please refer Appendix B for sample definition process for a Validation Template.

4.1.4. Validation Configurations

The Validation Engine component drives the complete validation for a given request. The request identifier and the required data values are passed to the Validation Engine to be validated.

The Validation Engine loads the required Validation Template to be used and checks for the data fields defined in the Validation Template. The Validation Engine checks whether a given field is mandatory or required validation before the validation is invoked on the required fields. The engine also checks for any custom validators defined for the data fields and use the same, if required. All errors returned by the validators are accumulated and returned to the caller. The engine indicates a success or failure of the validation and provides details on the errors.

The validation engine works on a map of values presented to it or on an XML document. The values are indexed by the Reference Field Name defined in the validation template. The reference Field Name is not mandatory and the Data Field Name shall be used in case the Reference Field Name is not provided in the template.

The Validation Engine is invoked before the transaction is passed to the business processing layer.

4.1.5. Validators

All Validator components extend the validator interface and provide the required validation logic for the data type and data field validations. Each data type defined in the Data Types section has a corresponding standard validator associated with the same.

The validator uses the configuration and the runtime (data values) passed to it to perform the required validations on the data field. The validators use the `FIELDFORMAT` field from the Data Dictionary definition or from the Data Element level (if overridden). The interpretation of the `FIELDFORMAT` field in the definition is as per the data type and the validator for the data types interprets that field for completing the validations.

CUSTOM VALIDATORS

The Validation Services support the mechanism of extending the default validators or developing custom validators which may be used for specific installations (site specific validators).

The custom validators can be plugged-in for individual data fields defined for the request in the `TXN_DATA` table using the `CUSTOM_VALIDATOR` field. The fully qualified class name of the Validator should be maintained in this location. The Validation Engine uses the Java Reflection API to instantiate the custom validator classes execute the same. Similarly, the custom validators can be plugged-in for the entire request using the `CUSTOM_VALIDATOR` field in the `TXN_DATA_MASTER` database table, for the required Request Identifier.

The basic data type validations are ignored while using a custom validator but the basic field length validations are still performed before passing the request / data field to the custom validator.

The guidelines governing any validation using custom validators are the same as the guidelines for defining standard or generic validations. Custom validators should typically be maintained in the site specific locations of the version control system.

4.1.6. Error Handling

The Validation Engine provides support for complete validations of all data elements in a request before a validation error is complete. This mechanism allows all the validation errors to be returned to the caller which can be rectified.

The default validation error returned is a generic error message as follows

```
Invalid Value for request [{0}], field name [{1}].
```

Where

- {0} is the Request Identifier
- {1} is the data field name (Reference Field Name) for which the validation has failed

The error code used is 113 in case of the default validation error being returned to the caller.

This default error message only provided information that a particular field in the Validation Template for the given Request Identifier has an invalid value but does not provide information on the details of the error. The Validation Services provide support for distinct error codes to be used for specific field validation failures. The ERRORCODE field in the TXN_DATA table which defines the Validation Templates should define the appropriate error code to be used by the Validation Engine in case a specific data field validation fails. The Validation Engine shall use this error code, if defined, to lookup the appropriate error message from the Oracle FLEXCUBE @ application messages table and return the same to the caller as the error description.

The actual error code returned by the validator, if different from the default error code of 113, or the error code defined in the TXN_DATA table for the given data field shall be returned to the caller in case of non default error codes used.

ⓘ All application error messages, which require to be looked up by the Validation Engine, should be

defined for the FLEXCUBE@ Application ID “A1”.

4.1.7. Generic Validators

Data Type	**Name**	**Validator Class Name**
A	Alphabets	com.iflex.fcat.xjava.data.AlphaValidator
S	String	com.iflex.fcat.xjava.data.StringValidator
N	Numeric	com.iflex.fcat.xjava.data.NumericValidator
F	Floating Point	com.iflex.fcat.xjava.data.DoubleValidator
I	Whole Numbers	com.iflex.fcat.xjava.data.IntegerValidator
E	Email Address	com.iflex.fcat.xjava.data.EmailValidator
D	Date	com.iflex.fcat.xjava.data.DateValidator
T	Timestamp	com.iflex.fcat.xjava.data.TimestampValidator
UA	UpperCase Alphabets	com.iflex.fcat.xjava.data.UpperAlphaValidator
US	UpperCase String	com.iflex.fcat.xjava.data.UpperStringValidator
PN	Positive Numeric	com.iflex.fcat.xjava.data.PositiveNumericValidator
FT	Free Text	com.iflex.fcat.xjava.data.FreeTextValidator

ALPHA VALIDATOR

This class provides validations for string values containing only alphabets. This validates both lower case and upper case alphabets.

STRING VALIDATOR

This class provides validations for string values containing alphabets, numeric and the special characters given in the formatfield column in table `data_dictionary/txn_data`.

NUMERIC VALIDATOR

This class provides validations for string values containing integers or doubles.

DOUBLE VALIDATOR

This class provides validations for string values containing double values.

INTEGER VALIDATOR

This class provides validations for string values containing integers.

EMAIL VALIDATOR

Validates email as per RFC 2822 token definitions for valid email

DATE VALIDATOR

This class provides validations for string values containing date values. The expected input date format should be specified in the `formatfield` column in table `data_dictionary/txn_data`.

TIMESTAMP VALIDATOR

This class provides validations for string values containing date values. The expected input date format should be specified in the `formatfield` column in table `data_dictionary/txn_data`.

UPPERALPHA VALIDATOR

This class provides validations for string values containing only alphabets. This validates both lower case and upper case alphabets.

UPPERSTRING VALIDATOR

This class provides validations for string values containing alphabets, numeric and the special characters given in the `formatfield` column in table `data_dictionary/txn_data`.

POSITIVE NUMERIC VALIDATOR

This class provides validations for string values containing positive integers or doubles.

4.2. EXTENTION User LifeCycle Handler

Oracle FLEXCUBE Direct Banking provides capability to extended processing behavior of certain milestones in user's lifecycle.

The handler should implement the interface `com.iflex.fcat.services.apps.UserLifeCycleExtendedHandler` and provided concrete implementation for below mentioned callback methods.

The handler can be optionally configured in table MSTENTITYUSERTYPES.USERLIFECYCLEHANDLER for each user type.

ⓘ Refer the Java Documentation for additional documentation on the component.

A default handler `com.iflex.fcat.services.apps.handlers.DefaultUserLifeCycleAlertHandler` provides out-of-box implementation of the above interface. This handler should be explicitly configured in the table if the extension is needed.

Extension available	Implementation in DefaultUserLifeCycleAlertHandler
Create User	Generate alert
Create Channel User	Generate alert containing channel user id
Modify User	Generate alert
Modify Channel User	Generate alert containing modified channel user
Create User – Login Password generation	Generate alert containing generated password
Create User – Transaction Pin generation	Generate alert containing generated pin
Change Login password	Generate alert containing changed password
Change Transaction pin	Generate alert containing changed pin
Reset Login password	Generate alert containing generated password
Reset Transaction pin	Generate alert containing generated pin
Change activation code	Generate alert containing activation code

4.3. Services Definitions

4.3.1. Service Principles

The Single Stack application architecture already partitions functionality into layers to manage complexity. For example, all common business functionality is separated into the EBS layer. Adopting an independent service approach adds:

- The potential for reuse of business functionality between clients

- A defined point at which to offer out this functionality
- Reduced dependencies due to clear packaging

This document embodies the set of overarching principles through which these benefits are realised:

- A clear business interface at the centre of the model
- Remote access through Remote Facades for different types of client
- Consistently applied packaging and structuring principles
- The requirement for Service instances to be stateless to aid scalability/availability characteristics

ACQUIRING TRANSACTIONS

When a service is invoked, depending on the service type (transactional, singleton), the service is invoked with transactional context. A service call from another service will be part of same transaction if same transaction context is used. If a service is invoked from another service with "null" transaction context, the invoked service will be processed under different transaction context.

Following API is used to invoke a service within another service.

```
ServiceManager.invokeService (
      String              p_service_name
,     RequestDTO          p_request
,     UserTransaction     p_user_transaction
,     TransactionContext  p_context
,     boolean             p_api_call
)
```

p_context in the above call denotes the transaction context. Transaction context is available to the service endpoint. When invoking a service from with another service endpoint, if current transaction context is passed in the above API call, the service being invoked will be part of the original transaction.

To fetch database connection within a service endpoint,

1) Database Connection as part of the transaction: To fetch database connection which should be part of transaction, getConnection method should be invoked on the object

TransactionContext. The method getConnection takes connection Identifier. The connection identifiers are registered in property file fcat.properties.

```
public InformationResponseDTO getInformation(

       TransactionContext         p_context

,      InformationResponseDTO     p_request

) {


//Connection part of the transaction

   Connection l_con = p_context.getConnection(AppConstants.FCAT_APP_ID);

}
```

The above example shows how to fetch database connection that will be part of the transaction.

Connection fetched as part of transaction should not be explicitly managed in the service endpoint (i.e. commit, rollback, close). The transaction is managed after the service endpoint has completed based on the return code.

2) Database connection independent of the transaction: This scenario occurs when the service endpoint needs to perform database operation which should be independent of the transaction & managed by the endpoint. To fetch the connection, use open connection using JFConnection.openConnection with "AP" as the identifier. AP connection management is handled by application itself and does not rely on application server.

```
public InformationResponseDTO getInformation(

       TransactionContext         p_context

,      InformationResponseDTO     p_request

) {


//connection independent of transaction & needs to be explicitly managed

Connection l_con = JFConnection.openConnection ("AP");
```

```
}
```

MULTI PHASE TRANSACTIONS

The Business Tier of Oracle FLEXCUBE Direct Banking supports the use of Multi Phase transactions using the JTA features supported within the Java EE Server.

All Business Services extending the `TransactionService` interface can be executed in a multi phase context. It is assumed that the underlying services are leveraging on XA enabled and compliant resources for this feature to be utilized. The

The `ISMULTIPHASE` columns of the `MSTSERVICES` table can be used to indicate a single-phase transaction context or a multi phase transaction.

The Business Tier is implemented as the Stateless Session Bean named `ServiceEndpointEJB` and the *Bean Managed Transaction Management* technique is used to leverage the `SessionContext` of the Session Bean for multi-phase transactions.

Acquiring Out of Context JDBC Connections within a Multi Phase Transaction

Nested transactions are generally not supported within the JTA context by many Java EE Application Servers and hence Oracle FLEXCUBE Direct Banking provides a facility to support opening single-phase JDBC connections from within the multi-phase context. This is usually required if some aspect of the transaction require to be handled independently irrespective of the actual transaction outcome or status.

Example:

`JFSequenceGenerator.getNextSequence` *call has to commit the incremented sequence number even though the transaction has been rolled back. A different single-phase transaction context is used.*

To handle above scenario, open connection using `JFConnection.openConnection` with "AP" as the identifier. AP connection management is handled by application itself and does not rely on application server.

4.3.2. Defining a New Service

Each 'Service', within the Oracle FLEXCUBE Direct Banking Services Architecture, defines an interface or a method within an existing interface that provides the business definitions for that function. The business definitions always define and exchange of DTOS (Request DTO and Response DTO) that require to be implemented by the implementation component or the endpoint

The endpoint provides the implementations for the business definitions to access data from various 'host data sources' using appropriate interface mechanisms.

INTERFACES

The Business Definitions are represented as Java Interfaces. The business definitions method signatures that provide the required business definitions for data exchange.

DATA TRANSFER OBJECTS

The Data Transfer Objects or DTOs are the data carriers within and external to the system. The DTOs provide transparent access to the underlying data (either incoming in the request or outgoing in the response).

The DTOs within the Services Architecture support specific data types as indicated below. All DTOs should define only the following data types.

Data Types

- All Primitive Java Types are supported as data types.
- Arrays of all Primitive Java Types are supported
- Other than primitive data types, only `java.util.Date` and `java.lang.String` instances are allowed to be fined.
- DTOs can contain other DTOs extending from the `BaseDTO`, `RequestDTO` or `ResponseDTO`.

ENDPOINTS

Service Endpoints are implementation components for the given service interface definitions. The endpoints are the plug-in components which adapt to the changing business requirements by host, region, implementation specific parameters etc.

Steps to create a new service

1) Create a new Interface with definition of all the services to implement.
2) The new Interface should extend `com.iflex.fcat.services.apps.interfaces.TransactionService` if the expected service will be transactional. If the expected service will be singleton, extend `com.iflex.fcat.services.apps.interfaces.SingletonService`
3) Create required Request/Response DTOs.
4) New RequestDTOs should extend `com.iflex.fcat.services.RequestDTO`
5) New ResponseDTOs should extend `com.iflex.fcat.services.ResponseDTO`
6) Create a new endpoint implementing the above create Interface.
7) Implement the services in the endpoint.
8) A service is made available by register it table `MSTSERVICES`.
9) Add validation information for the service in table `TXN_DATA_MASTER` and `TXN_DATA`.
10) Add service mapping information if needed in table `MSTSERVICESMAP`.

4.3.3. Extending a Service

Steps to extend a service

1) Create a new endpoint implementing the original interface.
2) Implement the new service.
3) Register the service with same service name but different version.
11) Add/Update mapping in table `MSTSERVICESMAP` to include new service version.

4.4. Authorization Engine

Oracle FLEXCUBE Direct Banking allows managing of customer mandates via the Authorization Engine. The Oracle FLEXCUBE Direct Banking Framework allows the Authorization Engine to be exposed as a service using the endpoints.

Authorization engine allows the customer to manage the various authorization rules as well as cater to zero authorizations.

For an engine to be configured for a transaction the following steps need to be followed.

- Transaction Authorization Flags in MSTCHANNELATS
- Engine Flags and Engine Mapping
- Flavors of Authorization Engines
- Configure Auth DTO mapper in MSTSERVICES
- Configure Modules to the transaction in MSTINITAUTHMODULES
- Action status matrix in MSTACTIONCONFIG
- Rules

4.4.1. Transaction Authorization Flags in MSTCHANNELATS

Special handling is required for verify requests at time of initiation to pass through authorization engine.

Sr. No	Table	Column(if any)	Description
1	mstchannelats	ISONLYVALIDATEREQUEST	This should be set to 'Y'
		AUTHREQUIRED	This should be set to 'Y'
		ISONLYAUTHVALIDATE	This should be set to 'Y'

4.4.2. Engine Flags and Engine Mapping

The name of the engine for the transaction is picked from a combination of userType, Channel ID and TXN ID from .

Sr. No	Table	Column(if any)	Description
1	Mstcustomerprofile	TypeAuth	This will indicate which engine is going to be for the customer

2	Mstusertypetxn	initauthid	This will indicate which engine is going to be picked up the usertype+channel+txn

Engine configured against customer has higher preference to engine configured against txn.

Engine flags are actually the heart of the engine and can be explained as below

Sequential Flag

This will make the engine a sequential one i.e. the user who will authorize a transaction will be in a sequential manner.

Module Check

This will inform the engine to validate the Authorization Data and will iterate though the various Modules. Each module will display Errors, Warning or Result message which each screen will need to display on screen

Auto Authorized

This will inform the engine to Auto Authorize the transaction i.e. the transaction would move from the Initiation to the Authorization State and will iterate though the various Modules. Each module will display Errors, Warning or Result message which each screen will need to display on screen

ISINITALSOAUTH

This will be used when initiator is also the authorizer. This is currently not implemented.

For initiator to be the authorizer also following other flags need to be checked

Sr. No	Table	Column(if any)	Description

1	Mstcustomerprofile	isinitailalsoauth	Need to be checked Y

4.4.3. Flavors of Authorization Engines

List Based Sequential / Non Sequential Authorization

The engine will have the facility to provide the customer to have their mandates executed in a sequential or a non sequential order

Zero Authorization

The engine will have the facility to provide the customer to have their mandates executed in a zero authorization mode but the transaction initiated with zero auth will not be seen in the initiators Transaction History transaction and will be directly seen in his Account Activity.

Retail Authorization

The engine will have the facility to provide the customer to have their mandates executed in a zero authorization mode but the transaction will still be seen in his Transaction History transaction.

Configurable User Based Engine

The engine will have the facility to provide the customer to have their mandates executed by specific users instead of groups.

engine will have the facility to configure the financial or non financial transaction

Null Checker

The engine will indicate that engine needs to be picked from the USERTYPE/TXNID/CHANNELID combination. If transaction has Null Checker engine Configured, Customer Specific engine will be ignored. Null Checker will ensure that the transaction will have their mandates executed but will not be captured in transaction history.

Note : *Behavior of the engine can be changed by making changes in engine flags. New engine can also be configured with different combinations of engine flag.*

4.4.4. Authorization Mappers

These mappers need to be mapped for transactions where the transaction need to be validated against the values given in the Module. Each transaction and a request id needs to have a DTO Mapper as part of the Service Definition. Currently there are currently 2 Categories of Mappers viz.

- Admin Mappers
- Specific Transaction Mappers

The Auth DTO Mapper needs to implement
com.iflex.fcat.services.apps.auth.authdtomapper.AuthorizationHostDTOMapper

The following methods need to be overridden:

doMapping (BaseDTO)-- This method should always be overridden to map data from transaction DTO into auth DTO. This is the first entry point from a transaction to Authorization Engine. The whole Authorization flow will get started with the data populated from this method

Sample mapper Code:

```
((AuthorizationRequestDTO )p_dto ).authData a.idCustomer = Primary customer

((AuthorizationRequestDTO )p_dto ).authData.namAccount = Should be debit
account

((AuthorizationRequestDTO )p_dto ).authData.idTxnCustId = Customer of the debit
account in question

((AuthorizationRequestDTO )p_dto ).authData.numAmount = Transaction amount

((AuthorizationRequestDTO )p_dto ).authData.codcurrency = Ttransaction currency

((AuthorizationRequestDTO )p_dto ).authData.valudate = Transaction date .

 ((AuthorizationRequestDTO )p_dto ).authData.codBranch = Cod branch of debit
account
```

doMapping(BaseDTO , JDBCResultSet)—This method is called by the Authorization engine to populate data into the Auth DTO from the DB entry made in ADMINTXNUNAUTHDATA for that transaction. This method is called only for a non auto authorized transactions.

The Auth DTO Mapper is configured in MSTSERVICES.

Sr. No	Table	Column(if any)	Description
• 1	• mstservices	• AUTHDTOMAPPER	• Specify the mapper to be used when initiating the transaction.

4.4.5. Modules

The engines will have the facility to execute a modules or a specific set of functional validation which can be a configured at a transaction level or for all transactions. They are essentially bits of components which can be invoked for a certain type of transaction or can be removed by making changes in the DB. This gives the flexibility to the engine to invoke a module on a particular transaction or not. Some of the generic modules are listed below.

Customer Access Modules

This module will ensure that the user has access to the customer/group/admin for which the transaction is being performed. This customer would be of the following types

- Primary Customer
- Secondary Customer
- Group

Transaction Access Module

This module will ensure that the user has access to the transaction he is performing which are as follows

- View
- Initiate

- Authorize

Account Access Module

This module will ensure that the user has access to the accounts for which the transaction is being performed

List Checker Module

This module will ensure that the appropriate filters get added both during view / initiating as well as authorization

Value Date Check Module

This module will ensure that the value date for the transaction is a valid business date based on the host calendar as well as the appropriate cutoff are taken care of.

Important: For transaction cutoff Value date check module refers to Db : TXNCUTOFFTIME , TXNCUTOFFTIMEDETAILS

For Hostprocessing date : System refers to entity specific views .

Ex : fcat_vw_process_date

The signature of the view has to be like :

create or replace view

fcc_vw_process_date **as**
select
today **as DATPROCESS**,
prev_working_day **as DATLAST** ,

next_working_day **as DATNEXT**
from <<>> **where** <<>>

Day 0 setup details:

It requires an entry in mstquery with idquery as

<<IDentity>>.HOST_PROCESSING_DATE :

Default look up will be on HOST_PROCESSING_DATE.

For Txn Cutoff check the Value date check module takes current system time from Db function convert_time which will return timestamp in entity time zone considering the daylight saving.

The function signature is:

```
create or replace function convert_time (tz in varchar2)
return timestamp with time zone
as
retval timestamp with time zone;
l_temp varchar2(10);
begin
select SESSIONTIMEZONE into l_temp from dual ;
retval:=from_tz(cast(current_timestamp as timestamp), (l_temp)) at time zone tz   ;
return retval;
end;
```

Where tz is the entity time zone name.

Note: Time zone name if used , oracle handle daylight saving and returns offset considering daylight saving . For example

London : corresponds to GMT +0:00 , if tz is specified as +0:00 , daylight saving will be ignored.

If tz is specified as "Europe/London" , during daylight saving offset returned will be +1:00 (depending on the saving value).

Grace period can be configured for a customer in MSTCUSTOMERPROFILE. If Grace Period is configured then the value date check module checks the Authorization date against Value date of the transaction + Grace Period for that customer.

Daily Limit Check Module

This module will ensure that the initiate as well as the transactions to be authorized is within the daily limit for that particular transaction.

Sr. No	Table	Column(if any)	Description
1	Mstmodule		Module ID, name and class are configured.
2	Mstinitauthmodules		Modules are mapped against engine and idtxn

Note : New modules can be configured as per requirements. Like BulkAccessCheckModule introduced to meet Bulk specific requirements.

Engines can be configured to use desired modules.

4.4.6. Action Status Matrix

The action Status Matrix will give the view for all the various actions maintained in the system.

Sr. No	Table	Column(if any)	Description
1	Mstinitauthstatus	INITAUTHID	Engine for with the Action is maintained
		FLGACTION	Action to be performed (I= Initiate M= Modifiy D= Delete A= Authorized RM=Reject for Modify /Send Back R= reject CI = Cancel Initiate VI = verify for Initiate V0 = Verify Only VA=Verify for Authorization TP = Template DF = Draft)
		FLGCURRSTATUS	Current Status of the transaction .
		FLGNEXTSTATUS	Next Status of the transaction if the action is performed

			successfully .
		ISMODULECHKREQD	Module Check required or not for the given action.
		ISPOSTAUTHREQD	Determines whether any Db operation is required or not.
		AUTHOPERATION	Determines which Db operation to be performed (99 = no DB operation 100=insert 102=update entire row 103=update only status 104=update only status 105=insert and update refidfcatref).
		COPYTOHISTORY	Determines where History tables should be updated or not .

4.4.7. Rules

Rules get applied with transaction is initiated. And Authorization takes place specific using the rules.

Rules determine details like who all can authorize the transaction, how many authorizer are required for specific transaction.

Rules are configured for customer:

- For a specific Account or All

- For a specific Branch or All
- For a range of Amount
- For a specific Transaction id or All
- Auto Authorization based on amounts for a rule

Sr. No	Table	Column(if any)	Description
1	mstcatrule		Rule is defined
2	mstcatruledetail	ISLIST and IDLIST IDSEQ	When Numer of auth is one or more , details of users from which lists can authorize the transaction is and the sequence (in case of sequential engine) is maintained here. When ISLIST flag is 'N' and then IDLIST should contain user ID of the user who will authorize the transaction. When ISLIST is 'Y' , IDLIST should have a valid list id , and users mapped to that id can authorize the transaction. Indicates the Sequence in which the uses will authorize the transaction , when sequential engine is applied.

4.4.8. AUTHORIZATION SUPPORT FOR NID (Non Intrusive Development)

For Non Intrusive development Authorization Engine has given some data exchange points between the Transaction Service and Authorization Engine which are as follows.

AUTHDTOMapper.doReverseMappingForRequest (RequestDTO, RequestDTO)-- This method should be overridden in the authdtomapper if the transaction wants the transaction request DTO to be populated with some values that authorization engine is generating **before call to actual service.**

AUTHDTOMapper.doReverseMappingForResponse (ResponseDTO, ResponseDTO)-- This method should be overridden in the authdtomapper if the transaction wants the auth response to be populated with some values that is fetched from transaction service response (it can be some value that is fetched from host). Auth Engine calls this method **after the actual transactions service is invoked.**

AuthorizationHelper. udpateUnauthMsg (RequestDTO, Connection)--This method can be called by any transaction endpoint service to update the blob 'MSG' in ADMINTXNUNAUTHDATA table. This method takes the request DTO passed and directly updates the entire blob with it. This method can be used by transactions for ' NID - Non Intrusive development' Auth mapping requirement. This is to be used **inside actual service call.**

4.4.9. HOST ERROR/WARNING TO AUTH STATUS MAPPING

Error /Warning returned by host are mapped to valid Auth staus and this mapping is maintained in msthostappdatamap.

Example : Host error code xxx0011 is Auth status SFR

A1	**	HOSTAPPSTATUS	26	xxx0011	Y

4.4.10. Configuration for dry run

Special handling is required to configure DRY run for a request

DRY run can be used when authorization is not required for any transaction , but module check is required. DRY run uses NULL_CHECKER as engine.

Sr. No	Table	Column(if any)	Description
1	mstchannelats	ISONLYVALIDATEREQUEST	This should be set to 'N'
		AUTHREQUIRED	This should be set to 'Y'
		ISONLYAUTHVALIDATE	This should be set to 'D'

4.4.11. Authorization dashboard tabs

Authorization Dashboard Menu consist of following tabs

- Initiated Transactions
- Transactions to Authorize
- View Transactions
- View Drafts/Templates

The transactions to be rendered in these tabs is configured using

FCAT_VW_MSTVIEWSTATUSMAP: View on MSTVIEWSTATUSMAP to map roletype with viewtype and status

IDVIEW	STATUS	TYPEROLE
D	6,19	I
A	1,2	A
M	1,28	N
C	5	N
I	1,2,3,4,5,7,9,10,11,23,25,26,28,34,35,36,38,39,40,41,42,43,44,45,46,47,55	I
V	1,2,3,4,5,7,9,10,11,23,25,26,28,34,35,36,38,39,40,41,42,43,44,45,46,47,55	V
U	25,11	N
R	3	N

The Status Description of transactions under these tabs is configured in APPLDATA against

(DATANAME ='STATUS_DESC_NG').

The Search Criteria query for Auth transactions is also configurable in APPLDATA against

(DATANAME ='FILTER_QUERY')

The Buttons on the Authorization Details screen is configured in APPLDATA against

(DATANAME = 'BUTTON_MAP_<<Transaction Status>>_<<

IdView>>_<<USERTYPE>>')

Example: BUTTON_MAP_1_A_INA

This would mean Button map for transaction in cod Status =1 , under tab transaction to authorize and user type INA.

4.4.12. TRANSACTION SPECIFIC AUTH VIEW XSL

Every Transaction requiring authorization needs to create an Auth View xsl which will take take from authviewresponsedto and populate the transaction specific details. The design of the screen should be exactly similar to the confirm screen of that transaction.

This auth view xsl needs to be imported in txnimport.xsl file, along with an entry as follows:

```
<xsl:if
test="//faml/response/authviewresponsedto/currentstatics/authorizationstatisticsdto/id
txn = 'DFT' ">

             <xsl:apply-templates
select="//faml/response/authviewresponsedto/currentstatics/authorizationstatisticsdto/
authinfo/domesticfundstransferrequestdto" />

</xsl:if>
```

Wherein the string in the apply-templates tag is the template name inside the transaction specific auth view xsl.

4.4.13. CONFIGURE MODIFICATION OF A TRANSACTION THROUGH AUTHORIZATION

All transactions (Except admin transactions) configured for Authorization can be modified by the initiator.

To Configure Modification through authorization following hidden variables need to be added in transaction specific xsls(Prepare screen, verify screen and Confirm screen xsls for a transaction)

```
   <input type="hidden" name="fldauthaction"  value=""/>

   <input type="hidden" name="fldnextaction" value = "{//faml/request/fldnextaction}"
/>

   <input type="hidden" name="fldreferenceno"
value="{//faml/request/fldreferenceno}"/>

   <input type="hidden" name="fldstatebit" value="{//faml/request/fldstatebit}"/>
```

Following changes to be done in Transaction specific service xsl :

To be added after Service Name :

```
<serviceAction>
```

```
      <xsl:value-of select="//faml/request/ fldnextaction "/>
</serviceAction>
<referenceNo>
      <xsl:value-of select="//faml/request/fldreferenceno"/>
</referenceNo>
<stateBit>
      <xsl:value-of select="//faml/request/fldstatebit"/>
</stateBit>
```

An Entry needs to be made in MSTCHANNELATS for idrequest= RR<<idtxn>>99:

This entry will be same as the entry for rendering the prepare screen for that transaction.

4.5. Transaction Release Workflow

Traditionally, the system submits the transaction to host after it is authorized in Oracle FLEXCUBE Direct Banking. However there can be a requirement for certain transactions to be processed manually in Oracle FLEXCUBE Direct Banking before sending them to host. There can also be a requirement to pass the transaction through a series of statuses after authorization.

In order to achieve this, generic workflow has been developed in Oracle FLEXCUBE Direct Banking.

Below are some of the key features of this workflow:

- Provides a generic framework in which the transaction can move from one status to another based on certain action by Oracle FLEXCUBE Direct Banking user. This workflow will come into picture only after the transaction is fully authorized in FCDB.
- FCDB will not attach any special meaning to any of the 'Statuses' configured in the workflow, but will treat them only as a state in which the transaction resides during its lifecycle.
- Transactions configured for 'Release' will go in 'Pending for Processing' state after they are fully authorized in FCDB.
- A separate transaction will be available, from where the 'Releaser' can manually release the transaction.
- The 'Releaser' must have the corresponding transaction (to be released) mapped to his/her role along with authorization access.

- The system supports cross user type release functionality. Transactions initiated and authorized by a corporate user can be released by a bank administrator user. This is configurable for each type of transaction.
- The status transaction flow can be extended to any level and status call can be configured at any status

ⓘ Refer the **Oracle_FLEXCUBE_Direct_Banking_Transaction_Release_Workflow** document for further details on configurations of the Transaction Release Workflow.

4.6. Security Question Authentication

Oracle FLEXCUBE Direct Banking provides SECURITY QUESTION AUTHENTICATION mechanism to authenticate transactions. To introduce SECURITY QUESTION AUTHENTICATION, admin will need to enter the corresponding handler entry in mstentityusertypes table and take server restart. During authentication process, authentication will happen as per the sequence of authenticators maintained in mstentityusertypes for the applicable entity-usertype combination. It should be noted that authenticators are maintained at ENTITY-USERTYPE level ie, authentication sequence shall remain same for all the transactions of any given ENTITY-USERTYPE combination.

Admin can update the questions already maintained in day zero to reflect their choice of question. Once server is restarted post update of day zero questions, updated questions will start appearing in maintenance screen. Above activities need to be carried out on Day one.

It should be noted that number of security questions in appldata table should be sufficient if not more to span every possible entity-usertype-set combination.