

Yugandhar Microservice Platform (MSP)

Development and Customization Guide

Yugandhar Microservices Platform Release - V1.0.0

Date – 23/05/2018

+++++

Copyright [2017] [Yugandhar Microservices Platform]

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

About Yugandhar Project.....	4
About Yugandhar Microservice Platform	4
About this document.....	4
Before reading this document	4
Understanding Java Projects and Workspace	5
Project structure	5
yugandhar-microservices-platform-ews.....	5
yugandhar-microservices-platform-jeec	6
yugandhar-msp-eureka-server	7
Understanding Java classes	8
<entity>Component.java.....	8
<entity>ComponentRule.java	8
<entity> JPA Repository.java	9
<entity>Service.java	9
Entity Data Object Classes	9
Cache configuration.....	10
Spring boot properties	10
EWS:	10
JEEC:	10
Understanding different Components.....	10
Data Model	10
• Data entities	11
• Reference data entities (For storing list of values as key – value pairs)	11
• Configuration tables	11
• Audit log tables	11
Understanding Application Configuration.....	11
• Application Properties.....	11
• Error Codes registry	11
• Transactions Registry	11
• Multilingual support for reference tables	11
Extending Data Model.....	11

Optimistic Lock	12
Building Services	12
Service is a spring bean class which can be integrated with Yugandhar Microservices Platform request response framework so that set of business logic is executed as single unit of transaction.	12
Logically there are three types of entity services.....	12
Authorization Framework (Access Control).....	14
Pagination Framework.....	16
Request Parameters -	16
Response Parameters	16
Pluggable primary keys.....	17
Application Logging	19
YugandharPerfSummaryLogger	19
YugandharPerfErrorSummaryLogger	19
YugandharCommonLogger	19
YugandharCacheLogger	19
Audit Logs	19
List of Configuration properties as present in CONFIG_APP_PROPERTIES table	20
JMS Integration.....	21
Default Listener	21
Default Sender	21
Encryption Utility	22
Setting up the Development Environment (Workspace)	23
Code Generation using Freemarker templates and Hibernate tools	23
Invoking the transactions	23
Understanding Data Model	23

About Yugandhar Project

The Yugandhar Project is the umbrella project focused on building open source cloud ready solutions. The current offerings include Yugandhar Microservice Platform (MSP) and Yugandhar Open Master Data Management (MDM) Hub.

About Yugandhar Microservice Platform

Yugandhar Microservice Platform (also referred as Yugandhar MSP) provides framework for rapid development of your Microservice. This is an architecturally proven Springboot based application having all the basic components needed for a Microservice application to work which just needs to be extended as per your requirements.

Yugandhar Microservice platform codes with code generation templates for rapid development. Just design your data model and generate the code using Yugandhar templates, your base table services will be ready. To create the composite services, you may have to write custom code which would take minimal efforts. Your new Microservice would be ready in just few hours.

About this document

This is the development and customization guide of the Yugandhar Microservices Platform. This is intended for the use of developers.

Before reading this document

Refer the 'Yugandhar Microservices Platform Architectural Overview' before proceeding with this document.

Understanding Java Projects and Workspace




The Yugandhar Open Master Data Management (MDM) Hub codebase is shared on github at below location

1. Github public repository - <https://github.com/yugandharproject/yugandhar-microservices-platform>
2. Javadoc – <github repository>\resources\javadocs
3. Dbdocs – <github repository>\resources\dbdocs
4. Documentation -<github repository>\resources\documentation
5. Database setup scripts -<github repository>\resources\dbsetupscripts

You may choose to download the projects using 'clone and download' option available online or clone the repository using github client.

Project structure

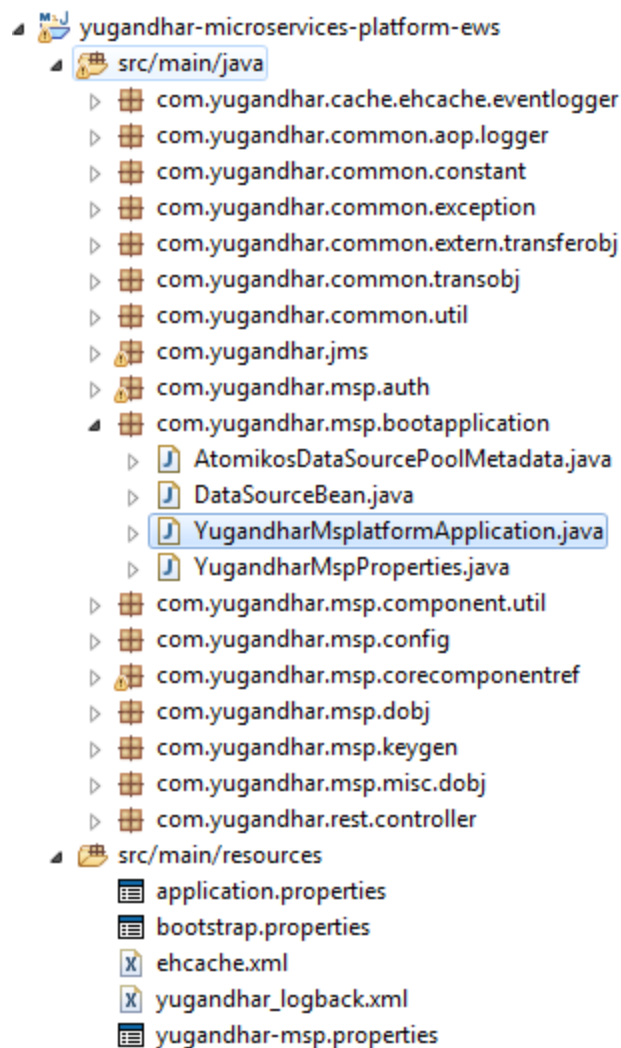
The Yugandhar Microservices Platform currently has three projects as shown in below screenshot

- ▷  yugandhar-microservices-platform-ews
- ▷  yugandhar-microservices-platform-jeec
- ▷  yugandhar-msp-eureka-server

The users must use ews project if embedded web Server needs to be used in production environment. If you plan to deploy a war file on JEE container (Web Server) then choose the jeec.

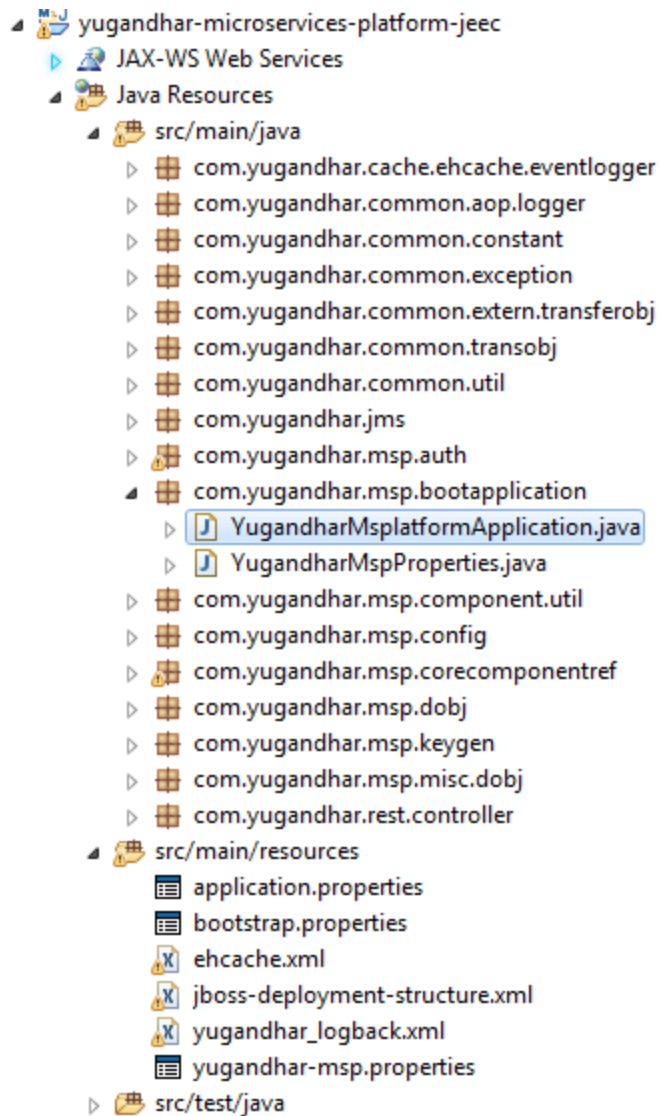
yugandhar-microservices-platform-ews

This is spring boot application project for Embedded Web Server (EWS). The below screenshot shows the structure of this project, the class YugandharMsplatformApplication has the spring boot initialization configuration which is used in conjunction with yugandhar-msp. properties and application.properties files. ehcache.xml and yugandhar_logback.xml files are used to configure ehcache and logback respectively.



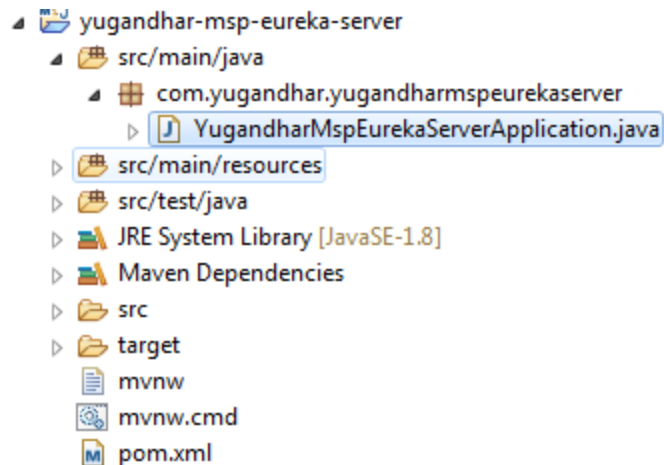
yugandhar-microservices-platform-jeec

This is spring boot application project for Java Enterprise Edition Container (JEEC) flavor of Yugandhar Microservices platform. The below screenshot shows the structure of this project, the class `YugandharMsplatformApplication` has the spring boot initialization configuration which is used in conjunction with `yugandhar-msp.properties` and `application.properties` files. `ehcache.xml` and `yugandhar_logback.xml` files are used to configure ehcache and logback respectively.



yugandhar-msp-eureka-server

This is the Yugandhar Eureka Server for service discovery; it's a small application having all the necessary configurations to run as Eureka server.



Understanding Java classes

The below java classes are present for each of the entities present out of the box in Yugandhar MSP, also the same will get generated by Yugandhar templates.

<entity>Component.java

The Component beans handle the entity level data CRUD (Create, Update, Retrieve, Delete) operations. This bean refers and makes use of Entity Data Objects (DO), entity manager and JPA repository classes.

Component Bean primary have below methods

- Persist() –This method is used to create a new record in a database entity.
- merge() –This method is used to update existing record in a database entity based on primary key.
- findById() –This method is used to retrieve existing record in a database based on primary key.
- findByBusinessKey() – This method is used to search a database table based on business key e.g. firstname and lastName from Person table.

<entity>ComponentRule.java

The component rule class used to write the business rules for a specific entity transaction based on the method from component bean is invoked. The entity transaction can have rules. The business rules are defined at below levels based on the transaction

- Rule cross points for Persist() method:
 - prevalidatePersist
 - preExecutePersist
 - postExecutePersist

- Rule cross points for Merge() method
 - PrevalidateMerge
 - preExecuteMerge
 - postExecuteMerge
- Rule cross points for FindById()
 - postExecuteFindById
 - prevalidateFindById
- Rule cross points for FindByBusinessKey() method
 - postExecuteFindByBusinessKey
 - prevalidateFindByBusinessKey

This rule class is carved out separately so that the users make the changes in the rule class instead of the Component class itself. This way any future changes in the component class (due to additional features introduced in later releases of code generation templates) will not mess up with the changes of users.

<entity> JPA Repository.java

The JPA repository is implementation of `org.springframework.data.jpa.repository.JpaRepository` interface used primarily to search the data from the single table based business keys.

<entity>Service.java

This is a Service bean of spring framework and the request from request Processor would always come to Service bean for further processing. The Service bean is a logical layer between RequestProcessor and Component bean introduced to create base as well as composite transaction and infuse any additional business logic so to insulate Component beans from having any business logic.

Entity Data Object Classes

The entity data objects are used by hibernate to map the DO to database entities. We have divided the DOs in AbstractDO and Child DO. The Yugandhar Template will generate the AbstractDO and ChildDO and the thumb rule is that the custom attributes and methods must to be added to Child DOs and Abstract DO be kept intact as generated by the template. This would keep the code clean and any further enhancement by Yugandhar team in the templates in future releases can be seamlessly incorporated in your code.

Cache configuration

Yugandhar Microservices Platform cache the Reference Data values (List of values stored as key-value pairs) so to improve transaction response time. It uses ehcache which is an open source, jsr 107 compliant cache framework that boosts performance, offloads your database, and simplifies scalability. It's the most widely-used Java-based cache because it's robust, proven, full-featured, and integrates with other popular libraries and frameworks. The ehcache configuration file is kept in /src/main/resources/ehcache.xml of the yugandhar-microservices-platform project. The default ehcache expiry time is configured as 1296000 seconds (15 days) in the ehcache.xml configuration file. This may be revisited as per the requirements.

Ehcache is configured in Spring boot project application.properties file using below property

```
#ehcache  
spring.cache.jcache.config=classpath:ehcache.xml
```

Visit www.ehcache.org for understanding ehcache in further detail.

Spring boot properties

EWS: The Yugandhar Microservices Platform uses application.properties and yugandhar-msp.properties present at / yugandhar-microservices-platform-ews/src/main/resources folder. The Datasource, logging, ehcache, JTA transaction manager and Json configuration are currently configured in these property files. The properties are self-explanatory and should be enabled/disabled as needed. The data source userid and password can be plain text or encrypted, to encrypt a string use the below command and place the encrypted password at appropriate place. Refer Encryption utility section for more details.

JEEC: The JEEC flavor user application server datasources, ActiveMQ implementation and JTA transaction manager instance. It gets all the required details using application server JNDI, the configuration of required server configuration is provided in environment setup guide. There are very few configurations in properties file for JEEC in comparison to EWS.

Understanding different Components

Data Model

Yugandhar MSP has below types of entities in the database

- **Data entities**
- **Reference data entities (For storing list of values as key – value pairs)**
- **Configuration tables**
- **Audit log tables**

All the above entities are covered in comprehensive details in Data Model Guide.

Understanding Application Configuration

- **Application Properties**

The application properties are stored in CONFIG_APP_PROPERTIES table.

- **Error Codes registry**

The error codes are stored in CONFIG_ERRORCODE_REGISTRY table. This table have CONFIG_LANGUAGE_CODE_KEY attribute which is used to store multilingual support for the error code. By default Yugandhar Microservices Platform take the language code from the header object (txnHeader.requesterLanguage attribute) of the request message and retrieve the error code on language code – error code combination.

- **Transactions Registry**

Every transaction is registered in CONFIG_TXN_REGISTRY table so to get picked by the request processor. The transaction name, class and method needs to be registered in the table.

- **Multilingual support for reference tables**

The list of languages considered for multilingual support is configured in CONFIG_LANGUAGE_CODE. This table should not be confused with REF_LANGUAGE_CODE which is used to store the LOV of worldwide languages available or served by enterprise.

The application configuration entities are covered in more detail in Data Model guide.

Extending Data Model

Introduction of New data Object is done when a completely new database entity needs to be created in database. The details of this step are covered in Code Generation Guide.

Optimistic Lock

To avoid concurrent update of the same record, Microservices Platform use optimistic lock based on VERSION attribute of the database. Microservices Platform relies heavily on Hibernate Optimistic lock feature.

Optimistic locking assumes that multiple transactions can complete without affecting each other, and therefore transactions can proceed without locking the impacted data resources. Before making a commit each transaction verifies that no other transaction has modified its data. If the check reveals conflicting modifications, the committing transaction rolls back. When your application uses long transactions or conversations that span several database transactions, you can store versioning data, so that if the same entity is updated by two conversations, the last to commit changes is informed of the conflict, and does not override the other conversation's work. This approach guarantees some isolation, but scales well and works particularly well in Read-Often Write-Sometimes situations. * (The definition is as per Hibernate website)

So while updating a record in database through APIs, the version attribute needs to be provided in the request having the value matching with the value in the database. e.g. For updating REF_CURRENCY table having idpk x, provide the currencyDO.version attribute the same as that in the database for idpk x.

Building Services

Service is a spring bean class which can be integrated with Yugandhar Microservices Platform request response framework so that set of business logic is executed as single unit of transaction.

Logically there are three types of entity services

1. Base entity services
Base entity services perform CRUD (create, retrieve, update and delete) operations on single database entity.
2. Composite Services
Composite entity services perform CRUD (create, retrieve, update and delete) operations on multiple database entities as single transaction. Also composite services can have composition of create/ retrieve/ update or delete in single services. e.g. searchAuthUsers service perform a search on different tables in database.

Samples –

1. Base entity Service – Refer
com.yugandhar.msp.config.ConfigErrorcodeRegistryComponent.findById()
method which is registered as transaction.

2. Composite Service – Refer OOTB service class
com.yugandhar.msp.auth.SearchAuthRolesService

After writing the code you need to register this as a transaction in Configuration transaction registry CONFIG_TXN_REGISTRY table. You may use below sql to register the transaction

```
Insert into <SCHEMA_NAME>.CONFIG_TXN_REGISTRY
(ID_PK, VERSION, TXNSERVICE_NAME, TXNSERVICE_CLASS,
TXNSERVICE_CLASSMETHOD, DESCRIPTION,CREATED_TS, UPDATED_TS,
UPDATED_BY_USER, UPDATED_TXN_REF_ID)
Values
(YUG_REGISTRY_SEQ.nextval, 0, '<name of transaction>', '<fully qualified name of the
class which have the method> , '<method name>',
'<description of the service>', CURRENT_TIMESTAMP,CURRENT_TIMESTAMP,'<user
name>', '<transaction id>');
COMMIT;
```

The Microservices Platform request response framework invokes the given method using Spring ReflectionUtils framework.

The TxnTransferObj is the default transfer object of MDM request and response. The TxnTransferObj encapsulates TxnPayload object which is the wrapper for all the objects. So if you are introducing a whole new object as request or response then you need to define this in TxnPayload object. Please refer Code Generation guide to understand how to define an object in TxnPayload object.

Authorization Framework (Access Control)

Authorization framework is used to provide access to the user or group to a particular transaction service. The authorization framework consists of below data entities

- AUTH_ROLES_REGISTRY - This Table stores the Authorization roles
- AUTH_USER_REGISTRY - This table stores the user names
- AUTH_USER_ROLE_ASSOC - This table stores the user to role association.
- AUTH_USERROLE_ACCESSCONTROL - This table stores the user/role to txn mapping so that execution of the transaction is controlled based on the transaction being invoked from the request. PROFILE_TYPE can be either USER or ROLE, mention the IDPK of the AUTH_ROLES_REGISTRY if PROFILE_TYPE is role else mention the IDPK of AUTH_USER_REGISTRY if PROFILE_TYPE is USER
- CONFIG_APP_PROPERTIES - the property com_yugandhar_authorization_framework_enabled must be set to true in this table. The authorization framework can be disabled altogether if the value is set to false. There might be a need to disable the authorization framework in non-production environment or in production for some specific business scenario. If so, this property must be set to false so that the framework is disabled.

Before invoking the transaction, the request processor performs the authorization based on txnHeader.userName or txnHeader.userRole and txnHeader.transactionServiceName attributes.

The searchAuthAccessControl service is used to perform the authorization based on below rule

1. If username and userRole are present in the request then authorization if all of the below conditions are satisfied
 - ✓ The given user must be present in AUTH_USER_REGISTRY table.
 - ✓ The given role must be present in AUTH_ROLES_REGISTRY table.
 - ✓ The user must be associated with the given role in AUTH_USER_ROLE_ASSOC table
 - ✓ The role has access to the transaction invoked i.e. AUTH_USERROLE_ACCESSCONTROL have entry for given role and transaction service name.
2. If userRole is present and username is null in the request then role is authorized to perform the given transaction if all the below conditions are satisfied
 - ✓ The given role must be present in AUTH_ROLES_REGISTRY table.

- ✓ The role has access to the transaction invoked i.e. AUTH_USERROLE_ACCESSCONTROL have entry for given role and transaction service name.
3. If username is present and userRole is null in the request then user is authorized to perform the given transaction if all the below conditions are satisfied
- ✓ The given user must be present in AUTH_USER_REGISTRY table.
 - ✓ The user has access to the transaction invoked i.e. AUTH_USERROLE_ACCESSCONTROL have entry for given user or role and transaction service name.

Sample header objects are as below

Sample 1 - username and roleName both are present in header	"txnHeader": { "userName": "rakesh", "userRole": "admin", "transactionServiceName": "createDemoService" }
Sample 2- Only roleName present in header	"txnHeader": { "userRole": "admin", "transactionServiceName": " createDemoService " }
Sample 3 - Only username is present in header	"txnHeader": { "userName": "rakesh", "transactionServiceName": " createDemoService " }

Note – As Yugandhar Microservices Platform is built on SOA framework, authentication (user credentials validation) framework is not provided. The application heavily depends on Application Server features (e.g. LDAP or webseal integration, SSL handshake etc) for user authentication. It is considered that the user or role name coming in the request is valid and authenticated.

The rules for authorization are defined in com.yugandhar.mdm.auth.searchAuthAccessControlService class. So if there is a need to change the above mentioned rules then write a new service for authorization and replace the TXNSERVICE_CLASS and TXNSERVICE_CLASSMETHOD with appropriate values as per new rule class.

TXNSERVICE_NAME	TXNSERVICE_CLASS	TXNSERVICE_CLASSMETHOD
searchAuthAccessControl	com.yugandhar.mdm.auth.searchAuthAccessControlService	process

Pagination Framework

The Pagination framework supports paged retrieval of the information. The `txnPayload` object has below attributes which needs to be provided while invoking the transaction.

Request Parameters -

`paginationIndexOfCurrentSlice`: used to provide the requested page number

`paginationPageSize`: Input parameter to define the size of the page (e.g. 25 elements or 100 elements)

Response Parameters

The response returns additional information about the total number of pages, elements on current slice and total elements.

- **`paginationIndexOfCurrentSlice`**: Same as that of Input parameter
- **`paginationPageSize`**: Same as that of the request parameter
- **`paginationInfoElementsOnCurrentSlice`**: Output information on elements database rows on the current slice or page.
- **`paginationInfoTotalElements`**: Output information on total number of elements database rows) for given search/retrieve/find criteria. This attribute is returned only in case of retrieve transaction response and NOT in search response as search result may potentially be very large and it will degrade the transaction response time if total number of elements is to be calculated.
- **`paginationInfoTotalPages`**: Output information on total number of pages for given search/retrieve/find criteria. This attribute is returned only in case of retrieve transaction response and NOT in search response as search result may potentially be very large and it will degrade the transaction response time if total number of elements is to be calculated.

Sample request xml

```
{ "txnHeader": {
    "requesterLanguage": "1",
    "userName": "admin",
    "userRole": "admin",
    "txnMessageId": "12312311115999",
    "transactionServiceName": "findAllRefCurrencyByLanguageCodeBase"
  },
  "txnPayload": {
    "paginationIndexOfCurrentSlice": 1,
    "paginationPageSize": 25,
    "refCurrencyDO": {
      "configLanguageCodeKey": "1"
    }
  }
}
```

At the entity manager level, Yugandhar Microservices Platform uses the jpa repository in retrieve transactions and entity manager query parameters for search

transactions. The findAllxxx transactions generated by Yugandhar templates supports the pagination by default.

Refer the code of below method to understand how to write the code for pagination.
com.yugandhar.mdm.composite.service.
com.yugandhar.msp.corecomponentref.RefCurrencyComponent.findAllRefCurrencyByLanguageCodeBase()

Pluggable primary keys

Pluggable primary key feature enables creating the record in the database with given primary key in the request. Generation of the primary key in Microservices Platform happens as per below logic

1. If primaryKeyDO is present for a given DO then verify that no record having given idpk is present in the database and then create a new record in database.
2. If primaryKeyDO is not present then generate the primary key based on default primary key generator and create a record in database.

The primary key generator is externalized in YugandharKeygenerator class. You may override the default generateKey() method if unique key generation logic needs to be customized.

Snippet: Create legal entity transaction with user provided idpk

```
    "transactionServiceName": " createAuthRolesRegistryBase "
  },
  "txnPayload": {
    "authRolesRegistryDO": {
      "primaryKeyDO": {
        "idPk": "77776663666AAAA211"
      },
      "roleName": "TESTROLE",
      "description": "TESTROLE"
    }
  }
```

Snippet: Create legal entity transaction with auto generated idpk

```
    "transactionServiceName": " createAuthRolesRegistryBase "
  },
  "txnPayload": {
    "authRolesRegistryDO": {
      "idPk": null,
```

```
"roleName": "TESTROLE",  
"description": "TESTROLE"
```

```
}
```

By default, every out of the box persistent data Object (DO) has the primaryKeyDO. This is also applicable for DOs generated using Yugandhar free-marker template generated code.

Application Logging

The Microservices Platform application logging is configured using logback logging framework using below spring boot properties (application.properties file)

```
# logging
logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n
logging.level.org.hibernate.SQL=info
logging.level.org.hibernate.type.descriptor.sql=trace
logging.level.com.yugandhar.*=INFO
logging.config= classpath:yugandhar_logback.xml
#logging.file= #
```

Currently below loggers are defined in yugandhar_logback.xml

YugandharPerfSummaryLogger – This logger logs the performance summary of every transaction having success response.

YugandharPerfErrorSummaryLogger - This logger logs the performance summary of every transaction having failure response

YugandharCommonLogger – This is the common logger which logs the application processing.

YugandharCacheLogger - This logger logs the ehcache and caching framework related logs.

You may change the log levels, log file name pattern, appender type etc using logback xml.

Audit Logs

The Audit log framework consists of database tables to store the insert, update and delete operations performed on every single record of the database. Refer 'Understanding Audit Log table structure' of the data model guide for complete coverage of the functionality.

List of Configuration properties as present in CONFIG_APP_PROPERTIES table

Property Name	Value	Description
com_yugandhar_pagination_default_pagesize_search	25	default page size for the data table search results
com_yugandhar_pagination_referencelov_default_pagesize	50	default page size for the reference lov
com_yugandhar_pagination_datatables_default_pagesize	50	default page size for the data table find/retrieve results
com_yugandhar_phonetic_algorithm_class_method	encode	method name to invoke from the algorithm class, by default encode method from commons-codec distribution is invoked
com_yugandhar_phonetic_algorithm_class	org.apache.commons.codec.language.Nysiis	phonetic algorithm class, default uses the apache nysiis implementation from commons-codec distribution
com_yugandhar_match_le_Fuzzy_LePerson_RuleClassMethod	process	LE Matching rule class method-name for fuzzy matching of Person type LE
com_yugandhar_phonetic_framework_enabled	TRUE	phonetic framework is enabled by default, set it to false to disable
com_yugandhar_authorization_framework_enabled	TRUE	Enables or disable the authorization framework, must be enabled in production environment. In test or developement environment this can be disabled. Valid values are (true, false)
com_yugandhar_dateFormat	yyyy-MM-dd'T'HH:mm:ss.SSSZ	default date format

JMS Integration

Microservices Platform provides uses Active MQ as messaging queue. The Microservices Platform EWS comes with own ActiveMQ setup, on the other hand JEEC depends heavily on Jboss server resources to integrate with MQ Server and uses JNDI to connect to MQ server.

The package 'com.yugandhar.jms' have the below listener and sender along with some sample and test classes.

Default Listener

Microservices Platform has default MQ Listener defined in YugDefaultRequestQueueListener class which listens to queue YUG.DEFAULT.REQUEST using JMS configuration. For the JEEC version, a jms destination 'java:jboss/com/yugandhar/default/requestQueue' is defined in jboss server configuration which creates connectivity with queue and listener. The connection factory 'yugJNDIDestJmsListenerContainerFactory' is also defined in jboss configuration

Default Sender

Default message sender for outbound messages is defined in YugJMSMessageSender class. The default response queue of Microservices Platform is YUG.DEFAULT.RESPONSE. For the JEEC version, this queue is mapped to jms destination 'java:jboss/com/yugandhar/default/responseQueue' defined on jboss application server.

The below entries in the jboss server defines the default request and response queues

```
<jms-queue name="YUG.DEFAULT.RESPONSE" entries="java:jboss/com/yugandhar/default/responseQueue"/>
<jms-queue name="YUG.DEFAULT.REQUEST" entries="java:jboss/com/yugandhar/default/requestQueue"/>
<pooled-connection-factory name="YugandharDefaultPooledConnectionFactory"
entries="java:jboss/com/yugandhar/DefaultPooledConnectionFactory"
connectors="yugConnectorInvm" statistics-enabled="true"/>
```

Refer section '4.3 Configure RedHat JBOSS EAP' of Development Environment Setup document to understand more about the JEEC configuration for JBoss.

Encryption Utility

The Yugandhar-msp.properties of EWS provides a functionality to have encrypted username and password for datasource. On the other hand, JEEC relies on the application server to encrypt the credentials.

A utility to encrypt a string is provided in the Microservices platform. You may use this utility to encrypt and decrypt any other properties as required.

YugandharEncoderDecoder Arguments:

- Required: operation name valid values 'encode/decode'
- Required: input String valid values: any UTF-8 complaint String

Encoder example: YugandharEncoderDecoder encode mypassword

Decoder example: YugandharEncoderDecoder decode bXlwYXNzd29yZA==

Follow below steps to encrypt a password

- **Create a Jar file:** Package the ews as yugandhar-microservice-platform-ews.jar using maven or eclipse export as jar functionality.
- **To Encode:** Open the command line, change the current directory where above jar is present. Execute below command to encrypt the password

```
java -cp yugandhar-microservice-platform-ews.jar  
com.yugandhar.common.util.YugandharEncoderDecoder encode mypassword
```

Result will be displayed as below:

Encoded String:bXlwYXNzd29yZA==

- **To decode:** Open the command line and provide below command

```
java -cp yugandhar-microservice-platform-ews.jar  
com.yugandhar.common.util.YugandharEncoderDecoder decode  
bXlwYXNzd29yZA==
```

Decoded string will be displayed as below

Decoded String:mypassword

Setting up the Development Environment (Workspace)

Please refer 'Development Environment Setup Guide' document.

Code Generation using Freemarker templates and Hibernate tools

Please refer 'Code Generation Guide' document.

Invoking the transactions

Please refer 'API and Transaction Reference Guide' Document.

Understanding Data Model

Please refer Data Model Guide and DB Doc.