**Productkeylib**

**modules using productkeylib**

**productkeygenportal**

emscontactbean

emsresultstatusbean

emsupcontactbean

getcontactbymail

updatecontactbymail

pkgeventlog

pkgsendmail

emsfeaturebean

productbean

serialutil

feature.getfeature

chkduplicatekey

pkgchkappsnum

entitlement

getfeaturelist

generatelicense

commonutil

setupconfig

setupregion

activationbean

fileloadutil

searchgeneratedlicenses

**adminweb**

emsontactsbean

resultstatus

emsupcontactbean

updatecontactbymail

pkgsendmail

pkgeventlog

featurebean

resultstatus

getproduct

getcontactbymail

updateproductbyid

contactbean

featurebean

getfeature

updatefeaturebyid

eventlog

productbeN

serialutil

chkduplicatekey

chkappsnum

generatelicense

createnewcontact

deleteproductbyid

revokelicense

searchgeneratedlicense

createfeature

updatecontact

fileloadutil

loginbymail

setup

config

setup.region

activationbean

**keygenerator**

xsd.setup

common.bean

contactbean

entitlementbean

productbean

Featurebean

resultstatus

common.util

fileloadutil

serialutil

getcontactbymail

getfeature

chkduplicatekey

generatelicense

mail.pkgsendmail

product

eventlog

PKGEnTool

Not using

pkwebclientserver-master

not using

pkwebclientwebui-master

not using

SpeedCTO-master

not using

speedctotester-master

not using

SharpJapanLKGProductKey-master

not using

productkeylib is a library where all the utility functions are written

and whenever they need those functionalities they were making use of those classes calling from jar file

Understanding of Code Components in `sh.model.dto` and `sh.sqlstatements`

1. PrdDto Class (Product Data Transfer Object)

Overview:

The `PrdDto` class represents a product data transfer object (DTO). It is used for holding product-related information and facilitating the transfer of product data across different layers of an application. It includes fields such as product ID, name, description, version, and other attributes related to product configuration and state.

Key Methods and Logic:

-setPrdInfo(EmsProductBean productBean, ListTypeValue listTypeValue, String emsAccount):

- This method creates and returns a new `PrdDto` object based on the information from an `EmsProductBean` (a bean representing a product in the system), a `ListTypeValue` (which holds a family ID), and an account (`emsAccount`).

- The method constructs the `descr` (description) field by concatenating the status, start date, end date, and country flag.

- It sets fixed values for certain attributes like `lifecycle` (set to 1, representing DRAFT), `isEnabled` (set to true), `isDeployed` (set to false), and others based on the product's attributes.

- setPrdInfoOnModel(Product product, PrdNm prdNm, String emsAccount):

- Similar to the previous method, this method creates a `PrdDto` from a `Product` (representing a product model), a `PrdNm` (product name object), and the `emsAccount`.

- It extracts various properties from the `Product` (e.g., `productName`, `description`, `refId1`, `refId2`, etc.) and sets the appropriate values on the `PrdDto`.

- This method also handles the conversion of the fixed quantity from `BigInteger` to `Integer`.

- getXmlValue(XPath xpath, Element root, String valuePath):

- This utility method extracts a specific value from an XML element based on an XPath expression.

- It uses the `XPath` API to compile the expression and evaluate it against the XML `root` element.

- If the XPath expression fails, it logs the exception and returns an empty string.

Fields:

The class contains several fields such as:

- `prdId`, `prdName`, `ver`, `descr`: Basic product details.

- `isEnabled`, `isDeployed`: Flags indicating product state.

- `refId1`, `refId2`: Reference identifiers for the product.

- `familyId`, `fixedQuantity`, `activationType`: Configuration and product-specific data.

- `createDate`, `modifiedDate`: Timestamps for product creation and modification.

These fields are initialized through the constructors and retrieved via getter methods.

Use Cases:

- The `PrdDto` class is likely used for transferring product data within an application, such as between layers (e.g., from the database to the business logic layer or from a service to a front-end view).

---

2. ConnectSqlServer Class

Overview:

The `ConnectSqlServer` class is responsible for constructing the JDBC connection URL required to connect to a Microsoft SQL Server database. It reads the connection details from a configuration file and uses these values to build the connection URL.

Key Method and Logic:

- getConnectionUrl():

- This method loads the database connection properties (e.g., database endpoint, user, password) from a configuration file (via `FileLoadUtil.loadProperties(FILE\_NAME\_APPLICATION)`).

- It constructs the JDBC connection URL in the format:

```

jdbc:sqlserver://<db-endpoint>;databaseName=<db-name>;user=<username>;password=<password>

```

- The method returns the constructed connection URL as a string.

- If an error occurs (e.g., properties file not found or an I/O exception), it returns an empty string.

Properties and Dependencies

- FileLoadUtil: A utility class used to load properties from a configuration file. The `FILE\_NAME\_APPLICATION` likely points to a configuration file containing database connection information.

- SQLSRV\_DB\_ENDPOINT, SQLSRV\_DB\_USER, SQLSRV\_DB\_PASSWORD: Constants that represent keys in the properties file, holding the SQL Server's connection details.

Use Cases:

- This class is used to retrieve the connection URL necessary for establishing a JDBC connection to an SQL Server. It can be used in scenarios where database operations need to be performed, such as querying or updating product data.

Summary of the System's Design and Purpose

- PrdDto Class:

- The `PrdDto` class is designed to encapsulate product data and provide a standardized format for transferring product-related information. It is flexible and can be populated either from an `EmsProductBean` or a `Product` object, making it versatile for different data sources.

- The class is heavily used in scenarios where product data is created, modified, or transferred between different layers

ConnectSqlServer Class:

- The `ConnectSqlServer` class facilitates connection management to the Microsoft SQL Server by constructing the necessary JDBC URL based on configurable parameters. This class is essential for ensuring that the system can securely and reliably connect to the SQL Server to perform database operations.

Recommendations for Improvements

- Logging and Error Handling in `ConnectSqlServer`:

- It would be helpful to enhance error handling by adding specific error messages in case of connection URL construction failure. Using a logging mechanism to capture the details of any issues with property loading or URL creation could improve traceability.

-Validation of Properties in `ConnectSqlServer`:

- Adding validation to ensure that the required properties (`SQLSRV\_DB\_ENDPOINT`, `SQLSRV\_DB\_USER`, `SQLSRV\_DB\_PASSWORD`) are loaded correctly from the configuration file before attempting to construct the URL would help prevent failures.

- Use of Constants in `PrdDto:

- The `PrdDto` class uses several fixed values Consider making these constants configurable or moving them to a centralized constants file, which would make future changes to default values easier to manage.

Productkeygenerator

Overview of ProductKeyLib and Data Flow in the System

Introduction

ProductKeyLib is a library designed to manage product-related data, handle product database operations, and interact with external systems like SQL Server. It involves multiple components and utilities, including data transfer objects (DTOs), SQL connection handlers, utility classes, and helper methods. The library facilitates the creation, updating, and retrieval of product data, ensuring that products are properly handled in the system.

This document provides an overview of the system’s key components, particularly how the product data flows through the system, starting from creation to its use within the system. We will take Product as an example to demonstrate the data flow, illustrating where it is created, how it is processed, and what functionalities are involved in handling product data.

1. Key Components of ProductKeyLib

- PrdDto (Product Data Transfer Object): This class serves as a container for product data. It holds information like product name, version, description, family ID, fixed quantity, and more.

- ConnectSqlServer: Handles the creation of the SQL connection URL for connecting to the Microsoft SQL Server database.

- FileLoadUtil: A utility class that loads configuration properties such as database connection details from property files.

- Common Utilities: Various helper classes and constants are used throughout the system to process and format data (e.g., CommonUtil, CommonProcess).

2. Flow of Product Data

Let’s break down the Product data flow step by step, highlighting the interactions between different components in the library.

Step 1: Product Creation

- Product Data Source: A Product object is created based on input data or retrieved from an external system. This object may represent a product from a service or a database. For example, the Product object could be part of an incoming API response or a result from a database query. It contains key fields such as productName, productVersion, description, refId1, refId2, and others.

Step 2: Populating the PrdDto Object

- Using PrdDto.setPrdInfoOnModel(Product product, PrdNm prdNm, String emsAccount):

Once a Product object is available, it is passed into the PrdDto.setPrdInfoOnModel() method. This method extracts product-related data and stores it in a PrdDto object. Example: If we have a product named "Laptop", the method will set its name (prdName), version (ver), description (descr), references (refId1, refId2), and other attributes in the PrdDto object. The PrdDto also gets system-generated values like globalId, createDate, and modifiedDate. It also handles setting default or fixed values like lifecycle, isEnabled, and activationType.

Step 3: Interaction with the Database (SQL Server)

- Creating a Database Connection: The ConnectSqlServer class is responsible for connecting to the database. When the application needs to save the product data into the database, it retrieves the database connection URL through the ConnectSqlServer.getConnectionUrl() method. This method loads the database connection details from a properties file (e.g., database endpoint, username, password), and constructs the JDBC URL needed for SQL Server.

- Database Interaction: The PrdDto object containing the product data can then be used in database operations. For example, an insert or update SQL query might use the values stored in PrdDto to save the product information into the database. In the background, the system would interact with SQL Server to store or update product details using SQL queries. The connection string provided by ConnectSqlServer is essential for this interaction.

Step 4: Managing Product Data in Business Logic

- Product Data Handling: The PrdDto object is not only used for database storage but also serves as a carrier for product data between different system layers. For instance, the product data might be passed to the front-end application (e.g., a web or mobile app) where it can be displayed to users or used in further business logic (e.g., activating a product, showing details on a product page).

- Product Lifecycle Management: Product data is handled according to its lifecycle. For instance, products may go through different stages (e.g., draft, active, archived). These stages are represented in fields like lifecycle and isDeployed. The system may use the lifecycle field to control visibility or actions that can be taken on the product.

3. Example Flow for a Product

Scenario: Adding a New Product ("Laptop")

1. Step 1: Product Data Creation

- A new product, "Laptop", is created by an external system or user. The product data contains information like:

- productName: "Laptop"

- productVersion: "1.0"

- description: "A powerful laptop."

- refId1: "XYZ123"

- refId2: "ABCD456"

2. Step 2: Populating the PrdDto Object

- The product data is passed to the PrdDto.setPrdInfoOnModel() method. The method creates a PrdDto object with the product's details and other system data, such as:

- prdName: "Laptop"

- ver: "1.0"

- descr: "A powerful laptop."

- familyId: 123 (from the ListTypeValue object)

- lifecycle: 1 (draft state)

- activationType: 5 (SAOT)

- Other metadata such as creation and modification timestamps.

3. Step 3: Saving Product Data to Database

- The ConnectSqlServer.getConnectionUrl() method is invoked to retrieve the database connection string. A SQL query is executed to insert the new product's data into the database (e.g., INSERT INTO products ...).

4. Step 4: Business Logic and Display

- Once stored in the database, the product can be accessed for various operations, such as:

- Displaying product details on a website or in an app.

- Managing the product lifecycle (e.g., transitioning from draft to active).

- Activating or deactivating the product based on business rules.

4. Summary of Data Flow and Functionality

- PrdDto: Acts as the data container for product information. It provides a structured way to hold product details and is used for both database storage and transfer across different parts of the application.

- ConnectSqlServer: Handles database connections and ensures that the system can interact with the SQL Server database to store, retrieve, and manipulate product data.

- Data Flow:

1. Product Data is created (via Product or external input).

2. It is transferred to a PrdDto object where product-related details are populated.

3. The PrdDto object is then used to interact with the database through ConnectSqlServer to store or update the product in the database.

4. The data can be further used in business logic or displayed to users.

Conclusion

The ProductKeyLib plays a crucial role in handling product-related data, providing a framework for storing, retrieving, and transferring product information. The flow from product creation, through DTO population, to database interaction, and business logic processing, ensures a smooth and efficient process for managing products in the system.

The `PrdNm` class in the `sh.model` package is a Java model class that represents the product data. It defines various properties related to a product, such as the product name, ID, version, description, external IDs, activation type, lifecycle, and other attributes. This class plays a key role in managing and transferring product data within the system.

Key Role of the 'PrdNm' Class

The `PrdNm` class serves as a Data Transfer Object (DTO) that is used to encapsulate product information for operations like storing, retrieving, and updating product data. This model is often used in combination with other components (such as database access or business logic) to transfer data between layers of the application.

Data Flow Involving the `PrdNm` Class

Let’s revisit the example of the `Product` object and explain how the `PrdNm` class is involved in the data flow:

1. Data Initialization:

- The `PrdNm` class holds various attributes, such as `prdId`, `prdName`, `ver`, `descr`, `refId1`, `refId2`, `externalId`, `isDeployed`, `isEnabled`, `lifecycle`, `activationType`, etc.

- When a new product is created, an instance of `PrdNm` is populated with values representing the product. For instance, a product named "Laptop" could have attributes like `prdName = "Laptop"`, `ver = "1.0"`, `activationType = 1` (indicating activation status), and `familyId = 123`.

2. Setting Product Information:

- The `setPrdInfoOnModel` method, as mentioned earlier in `PrdDto`, likely takes an instance of `PrdNm` to populate product-specific details. It could set properties like the product name, version, and other attributes from the `PrdNm` object.

- Example: The `prdName` ("Laptop"), `ver` ("1.0"), and `descr` ("A powerful laptop.") would be transferred from `PrdNm` to `PrdDto`.

3. Database Interaction:

- The `PrdNm` object is used when inserting or updating product records in the database. For example, the `prdId`, `prdName`, `familyId`, and other product-related fields are included in an SQL statement to store the product details.

- The data in the `PrdNm` object is passed to database interaction layers (such as the `ConnectSqlServer` class), where SQL queries are executed to persist or retrieve product data.

4. Business Logic:

- The `PrdNm` object can be used for business logic processing, such as managing the lifecycle state of the product (e.g., `isDeployed` and `isEnabled` properties).

- It may also be used to track relationships with other entities (e.g., family or category through the `familyId` property).

5.Displaying Product Information:

- Once the product is stored or retrieved from the database, the `PrdNm` object can be used in the application's user interface (UI) or API responses. For example, when showing product details to the user, the `prdName`, `descr`, and `ver` attributes are used to present information in the UI.

Where and How `PrdNm` Is Imported and Used:

- In the DTO Layer:

- The `PrdNm` class is imported into the `PrdDto` (Product Data Transfer Object) class and possibly other classes where product data needs to be handled. It serves as the container for product attributes that are transferred through different parts of the system.

- In the Business Logic:

- The `PrdNm` object can be used directly in business logic to determine actions on products, such as activating/deactivating products (`isEnabled`), managing product quantity (`fixedQuantity`), or tracking lifecycle (`lifecycle`).

In the Database Interaction Layer:

- The `PrdNm` class's attributes are used when interacting with the database (via SQL queries). For instance, the `prdId`, `prdName`, `descr`, and `familyId` would be used in SQL statements to store or retrieve data from a product table in the database.