# Low-Level Design (LLD) - OFAC Batch Modernization (Spring Boot 3, Java 17)

This document provides a detailed Low-Level Design (LLD) for the modernization of legacy Clipper OFAC Batch Jobs into a Spring Boot 3 command-line batch application.

## 1. Purpose

The OFAC Batch application enhances the processes for extracting and processing OFAC (Office of Foreign Assets Control) records.   
It replaces legacy Java 1.6 and shell-based AIX batch jobs with a modern Spring Boot 3 Java 17-based batch framework that integrates with DB2, writes extract files, and runs on Kubernetes.

## 2. Architecture Overview

The project follows a layered modular batch architecture:

- Application Layer: Controls the overall flow and initialization.  
- Service Layer: Handles business logic for record processing.  
- DAO Layer: Interacts with DB2 for data fetching and updates.  
- Utility Layer: Provides common reusable functionalities (date formatting, file I/O, decryption).  
- Config Layer: Manages properties, datasource configuration, and environment-based credentials.

## 3. Component Overview

1. OfacProcessRecordsApplication – Entry point of the batch.  
2. OfacConnect – Handles DB connection setup.  
3. OfacDbRoutines – Executes queries and updates.  
4. OfacResendRecordExtract – Main business logic for record extraction and file generation.  
5. CommonDateRoutines – Utility for date handling.  
6. OfacPrimerData – Handles primer/reset resend date logic.

## 4. Data Flow

1. Spring Boot starts via OfacProcessRecordsApplication.  
2. Parses input arguments (environment, resend flag, full/delta flag, fileDate).  
3. OfacConnect loads DB credentials.  
4. DataSource creates DB2 connection.  
5. OfacDbRoutines runs SQL queries to fetch OFAC data.  
6. OfacResendRecordExtract builds record extracts and writes to file system.  
7. Updates DB extract flags and closes connection.  
8. Logs and exits with success/failure code.

## 5. Class Responsibilities

OfacProcessRecordsApplication: Initializes Spring Boot, validates arguments, and triggers the extraction flow.  
OfacConnect: Reads and decrypts DB credentials from configuration files.  
OfacDbRoutines: Executes queries, fetches records, and performs updates.  
OfacResendRecordExtract: Coordinates extraction logic and writes data to files.  
CommonDateRoutines: Provides reusable date formatting methods.  
OfacPrimerData: Executes primer job and resend date reset operations.

## 6. Configuration

Configuration is handled via application.yml and OFACBatch.properties files.  
It supports multiple environments such as DEV, TEST, and PROD.  
Passwords are stored encrypted, and DB credentials are decrypted at runtime.

## 7. Error Handling

- Invalid arguments trigger usage message and exit.  
- Database connectivity failures are logged and the application exits with code 1.  
- File write or SQL execution errors trigger cleanup and exit with non-zero code.

## 8. Logging Design

- Uses SLF4J with Logback.  
- INFO for batch start/end, DEBUG for query and counts, ERROR for exceptions.  
- Logs stored in container logs and persistent file system if required.

## 9. Deployment

- Run locally: java -jar clipper-ofac-springboot.jar DEV RESENDTRUE FULL 20250101  
- In Kubernetes: Deployed as Job or CronJob using container image.  
- Containerized using Eclipse Temurin JRE 17.  
- Configurable via environment variables or mounted ConfigMap.

## 10. Extensibility and Future Enhancements

- Support for S3 file uploads can be added by replacing FileWriter with AWS SDK integration.  
- Extend OfacDbRoutines to support stored procedures.  
- Integrate monitoring using Micrometer and Prometheus.  
- Add parallel processing or Spring Batch partitioning for large datasets.