



Fineract Platform Documentation

Version 1.14.0

Table of Contents

License	1
Preface	2
Introduction	3
Platform for Digital Financial Services	3
About	3
Contribute	3
Downloads	3
Resources	3
Deployment	4
Plugins	4
HTTPS	4
Docker Compose	5
Application Server	5
Fineract Instance types	6
Kubernetes	10
AWS	10
Google Cloud	10
Apache Software Foundation Infrastructure	10
Architecture	11
History	11
Resources	11
System Overview	11
Functional Overview	13
Principles	14
Design Overview	15
Persistence	21
Idempotency	24
Validation	25
Batch execution and jobs	26
Loan account locking	34
Technology	35
Modules	35
Introducing Business Date into Fineract - Community version	36
Reliable event framework	41
Introducing Advanced payment allocation	56
Features	61
Capitalized Income	61
Buy Down Fee	71

Approved amount modification on loans	83
Backdated interest modification	86
Interest Rate Modification For Progressive Loan.....	88
Contract Termination	93
Loan Charges.....	98
Journal Entry Aggregation.....	102
Loan Re-Aging.....	106
Fineract Development Environment.....	118
Git	118
GPG	118
Docker.....	118
Gradle.....	118
IDE.....	118
Kubernetes	119
Tools	119
Custom Modules.....	121
Introduction	121
Custom Services.....	123
Custom Business Steps	125
Custom Loan Transaction Processors.....	127
Custom Batch Jobs.....	134
Custom Database Migration	138
Deploying Custom Modules.....	139
Outlook.....	140
Resilience.....	141
Introduction Resilience	141
Command.....	143
Jobs	146
Loan	147
Savings	149
Security.....	152
HTTP Basic Authentication	152
OAuth	152
Two-factor authentication.....	154
Securing Fineract.....	154
Testing.....	159
Cucumber E2E Tests	159
Unit Testing	174
Integration Testing	174
Fineract Documentation Guide.....	190
File and Folder Layout	190

AsciiDoc	190
Diagrams	199
Editor	200
Antora.....	201
Releases	202
Configuration	202
Release Process.....	211
Maintenance Release Process	227
Publish Release Artifacts	228
Fineract SDKs	230
Generate Apache Fineract API Client	230
Frequently Asked Questions	232
Glossary	233
Appendix A: Fineract Application Properties	234
Tenant Database Properties	234
Hikari Connection Pool Properties	236
SSL Properties.....	239
Authentication Properties	240
Tomcat Properties	241
Kafka Properties.....	242
Metrics Properties	246
AWS Configuration Properties	247
Resilience4j Properties	247
Appendix B: Third Party Software	252

License

Copyright © 2015-2025 Apache Foundation

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

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.

Preface

Apache Fineract

Website: fineract.apache.org

Email: dev@fineract.apache.org

Version: 1.14.0

Built on: Fri Dec 19 09:28:41 PST 2025

For document authors and changelog, view code history for the `fineract-doc` directory in github.com/apache/fineract/.

Introduction

Platform for Digital Financial Services

See fineract.apache.org

About

See fineract.apache.org/#about

Contribute

See fineract.apache.org/#contribute

Downloads

See fineract.apache.org/#downloads

Resources

See fineract.apache.org/#resources

Deployment

Plugins

Apache Fineract is extensible through plugin JARs ([FINERACT-1177](#); based on [Spring Boot's support](#)). To launch Fineract with plugin JARs in `libs/*.jar`, use:

```
java -Dloader.path=libs/ -jar fineract-provider.jar
```

The Fineract "Docker" container image's `ENTRYPOINT` uses this, see our [Dockerfile](#). You could therefore build your customized Fineract distribution container image with your own [Dockerfile](#) using e.g. `FROM apache/fineract:latest` and then drop some plugin JARs into `/app/libs/`.

The WAR distribution does not directly support such plugins, but one could "explode" the WAR and drop JARs into `WEB-INF/lib`; if you know what you are doing, and feel nostalgic of the 1990s still using WARs, instead of the recommended modern Spring Boot distribution.

Here is a list of known 3rd-party plugin projects which can be dropped into `libs/`:

- github.com/vorburger/fineract-pentaho



The reporting module became our first module experiment out of necessity. We are currently developing a strategy to split up even more internals of Fineract into proper modules. Those that have an incompatible license will be hosted in a separate Git repository (probably on Github under the Mifos organisation). We'll send out an announcement as soon as we have more to say on this topic.

HTTPS

Because Apache Fineract deals with customer sensitive personally identifiable information (PII), it very strongly encourages all developers, implementors and end-users to always only use HTTPS. This is why it does not run on HTTP even for local development and enforces use of HTTPS by default.

For this purpose, Fineract includes a built-in default SSL certificate. This cert is intended for development on `localhost`, only. It is not trusted by your browser (because it's self signed).

For production deployments, we recommend running Fineract behind a modern managed cloud native web proxy which includes SSL termination with automatically rotating SSL certificates, either using your favourite cloud provider's respective solution, or locally setting up the equivalent using e.g. something like NGINX combined with [Let's Encrypt](#).

Such products, when correctly configured, add the conventional `X-Forwarded-For` and `X-Forwarded-Proto` HTTP headers, which Fineract (or rather the Spring Framework really) correctly respects since [FINERACT-914 was fixed](#).

Alternatively, you could replace the built-in default SSL certificate with one you obtained from a

Certificate Authority. We currently do not document how to do this, because we do not recommend this approach, as it's cumbersome to configure and support and less secure than a managed auto rotating solution.

The Fineract API client supports an insecure mode (`FineractClient.Builder#insecure()`), and API users such as mobile apps may expose Settings to let end-users accept the self signed certificate. This should always be used for testing only, never in production.

All SSL-related properties are tunable. SSL can be turned off by setting the environment variable `FINERACT_SERVER_SSL_ENABLED` to false. If you do that then please make sure to also change the server port to `8080` via the variable `FINERACT_SERVER_PORT`, just for the sake of keeping the conventions.

To use a different SSL keystore, set `FINERACT_SERVER_SSL_KEY_STORE` to a path to a different (not embedded) keystore. The password can be set via `FINERACT_SERVER_SSL_KEY_STORE_PASSWORD`. See the `application.properties` file and the [latest Spring Boot documentation](#) for details.

Docker Compose

TBD

Application Server

Tomcat

TBD

Undertow

TBD

Jetty

TBD

JBoss

TBD

Weblogic

TBD

Payara

TBD

Fineract Instance types

In cases where Fineract has to deal with high load, it can cause a performance problem for a single Fineract instance.

To overcome this problem, Fineract instances can be started in different instance types for better scalability and performance in a multi-instance environment:

Fineract instance types

- Read instance
- Write instance
- Batch instance

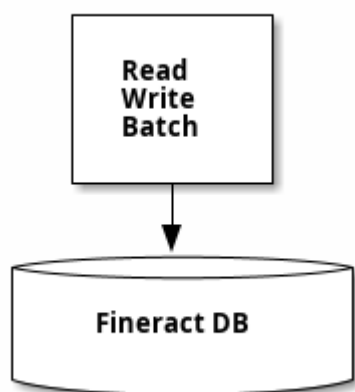
Each instance type comes with different restrictions. The specifics can be found in the table below.

Table 1. Instance types

	Read instance	Write instance	Batch instance
Using only read-only DB connection	Yes	No	No
Batch jobs are automatically scheduled or startable via API	No	No	Yes
Can receive events (business events, hook template events)	No	Yes	No
Can send events (business events, hook template events)	No	Yes	Yes
Read APIs supported	Yes	Yes	No
Write APIs supported	No	Yes	No
Batch job APIs supported	No	No	Yes
Liquibase migration initiated upon startup	No	Yes	No

Configuring instance types in single instance setup

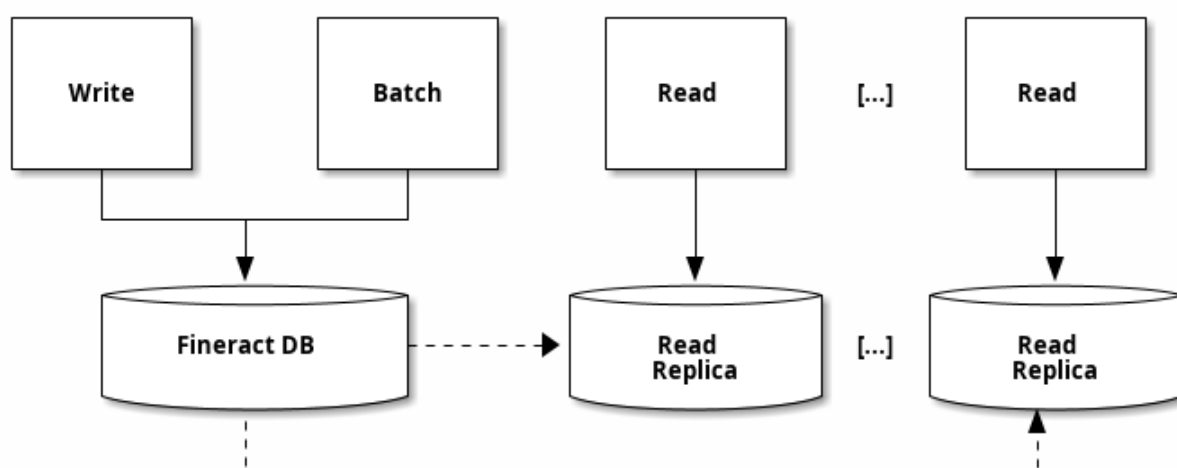
If Fineract is running as a single instance, then all of the 3 instance types should be enabled. In this case, there is no need to worry about the configuration, because this is the default behavior.



Configuring instance types in multi-instance setup

A common solution to dealing with the high load is to deploy 1 write and 1 batch instances and deploy multiple read instances with read replicas of the Fineract database.

In this case, the write instance and the database will be freed from part of the load, because read request will use the separated read instance and its read replica database.

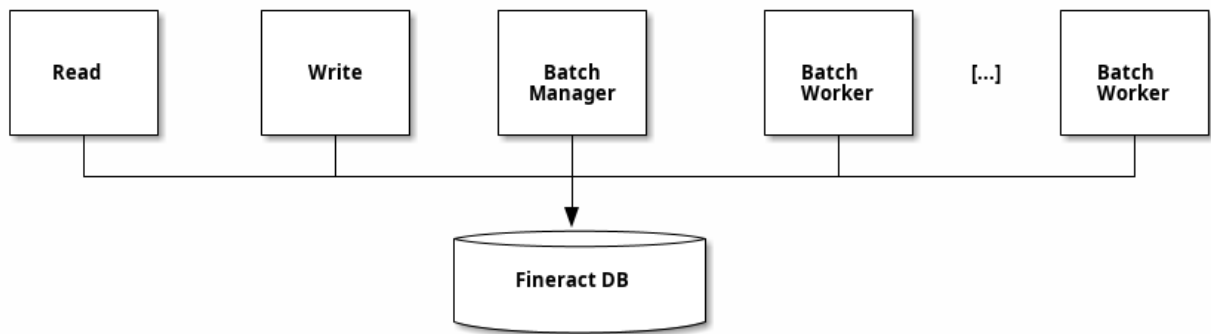


Also a common scenario when Close of Business jobs are running and Fineract has to deal with a high amount of processes.

(In a future release) Fineract (will be) is able to run this CoB jobs in batches.

In multi-instances environment these CoB jobs can run on multiple batch instances and they don't have any impact on the performance of the read and write processes.

The best practice is to deploy 1 master batch instance and multiple worker batch instances.



These solutions can be mixed with each other, based on the load of the Fineract deployment.

Configuring instance type via environment variables

The Fineract instance type is configurable via environment variables for the following 3 values:

Table 2. Environment variables

Instance type	Environment variable
Read instance	FINERACT_READ_MODE_ENABLED
Write instance	FINERACT_WRITE_MODE_ENABLED
Batch instance	FINERACT_BATCH_MODE_ENABLED

The environment variable values are booleans (true/false). The Fineract instance can be configured in any combination of these instance types, although if all 3 configurations are false, startup will fail. The default value for all 3 values is true.

The configured Fineract instance types are easily accessible via a single Spring bean, named `FineractProperties.FineractModeProperties` that has 4 methods: `isReadMode()`, `isWriteMode()`, `isBatchMode()`, `isReadOnlyMode()`

Liquibase Database Migration

Liquibase data migration is allowed only for write instances

APIs

Read APIs are allowed only for read and write instances

A Fineract instance is ONLY able to serve read API calls when it's configured as a read or write instance. In batch instance mode, it won't serve read API calls.

If it's a read or write instance, the read APIs will be served.

If it's a batch instance, the read APIs won't be served and a proper HTTP status code will be returned.

The distinction whether something is a read API can be decided based on the HTTP request method. If it's a GET, we can assume it's a read call.

Write APIs are allowed only for write instances

A Fineract instance is ONLY able to serve write API calls when it's configured as a write instance. In read or batch instance mode, it won't serve write API calls.

If the write APIs won't be served and a proper HTTP status code will be returned.

If it's a write instance, the write APIs will be served except the ones related to batch jobs.

The distinction whether something is a write API can be decided based on the HTTP request method. If it's non-GET, we can assume it's a write call. Also, the write APIs related to batch jobs (starting/stopping jobs) will not be served either.

Batch job APIs are allowed only for batch instances

A Fineract instance is ONLY able to serve batch API calls when it's configured as a batch instance. In read or write instance mode, it won't serve batch API calls.

If the batch APIs won't be served and a proper HTTP status code will be returned.

If it's a batch instance, the batch APIs will be served.

Batch jobs

Batch job scheduling is allowed only for batch instances

Batch jobs are scheduled only if the Fineract instance running as a batch instance

Read-only instance type restrictions

If the read mode is enabled, but the write mode and batch mode are disabled, Fineract instance runs in read-only mode.

Events are disabled for read-only instances

When a Fineract instance is running in read-only mode, all event receiving/sending will be disabled.

Read-only tenant connection support

With read separation, there's a possibility to use read-only database connections for read-only instances.

If the instance is read-only, the DataSource connection used for the tenant will be read-only.

If the instance is read-only and the configuration for the read-only datasource is not set, the application startup will fail.

Batch-only instance type restrictions

If the batch mode is enabled, but the read mode and write mode are disabled, Fineract instance runs in batch-only mode.

Receiving events is disabled for batch-only instances

When a Fineract instance is running as batch, event receiving will be disabled while sending events will be still possible since the batch jobs are potentially generating business events.

Kubernetes

In a scaled Kubernetes environment where multiple Fineract instances are deployed, doing the database migrations properly is essential.

Fineract provides a way to run only the Liquibase migrations instead of starting up the whole application server so that you can easily do the migrations before actually upgrading a Fineract instance.

The `FINERACT_LIQUIBASE_ENABLED` flag controls whether Liquibase is enabled or not. For regular read/write/batch manager/batch worker instances this should be disabled.

There's a special Spring profile that should be enabled for running Liquibase only. It can be done via `SPRING_PROFILES_ACTIVE` environment variable. The profile name is `liquibase-only`. At the end of the migration process, the application will exit.

For the instance running the Liquibase migrations, the profile should be activated.

AWS

TBD

Google Cloud

The www.fineract.dev demo server runs on Google Cloud.

[The Running Fineract.dev, SRE style presentation](#) given at *ApacheCon 2020* has some related background.

Apache Software Foundation Infrastructure

We can order a server from Apache's infrastructure team and deploy a demo instance...

TBD

Architecture

This document captures the major architectural decisions in platform. The purpose of the document is to provide a guide to the overall structure of the platform; where it fits in the overall context of an MIS solution and its internals so that contributors can more effectively understand how changes that they are considering can be made, and the consequences of those changes.

The target audience for this report is both system integrators (who will use the document to gain an understanding of the structure of the platform and its design rationale) and platform contributors who will use the document to reason about future changes and who will update the document as the system evolves.

History

The Idea

Fineract was an idea born out of a wish to create and deploy technology that allows the microfinance industry to scale. The goal is to:

- Produce a gold standard management information system suitable for microfinance operations
- Acts as the basis of a platform for microfinance
- Open source, owned and driven by member organisations in the community
- Enabling potential for eco-system of providers located near to MFIs

Timeline

- 2006: Project initiated by Grameen Foundation
- Late 2011: Grameen Foundation handed over full responsibility to open source community.
- 2012: Mifos X platform started. Previous members of project come together under the name of Community for Open Source Microfinance (COSM / OpenMF)
- 2013: COSM / OpenMF officially rebranded to Mifos Initiative and receive US 501c3 status.
- 2016: Fineract 1.x began incubation at Apache

Resources

- Project URL is github.com/apache/fineract
- Issue tracker is issues.apache.org/jira/projects/FINERACT/summary
- Download from fineract.apache.org/

System Overview

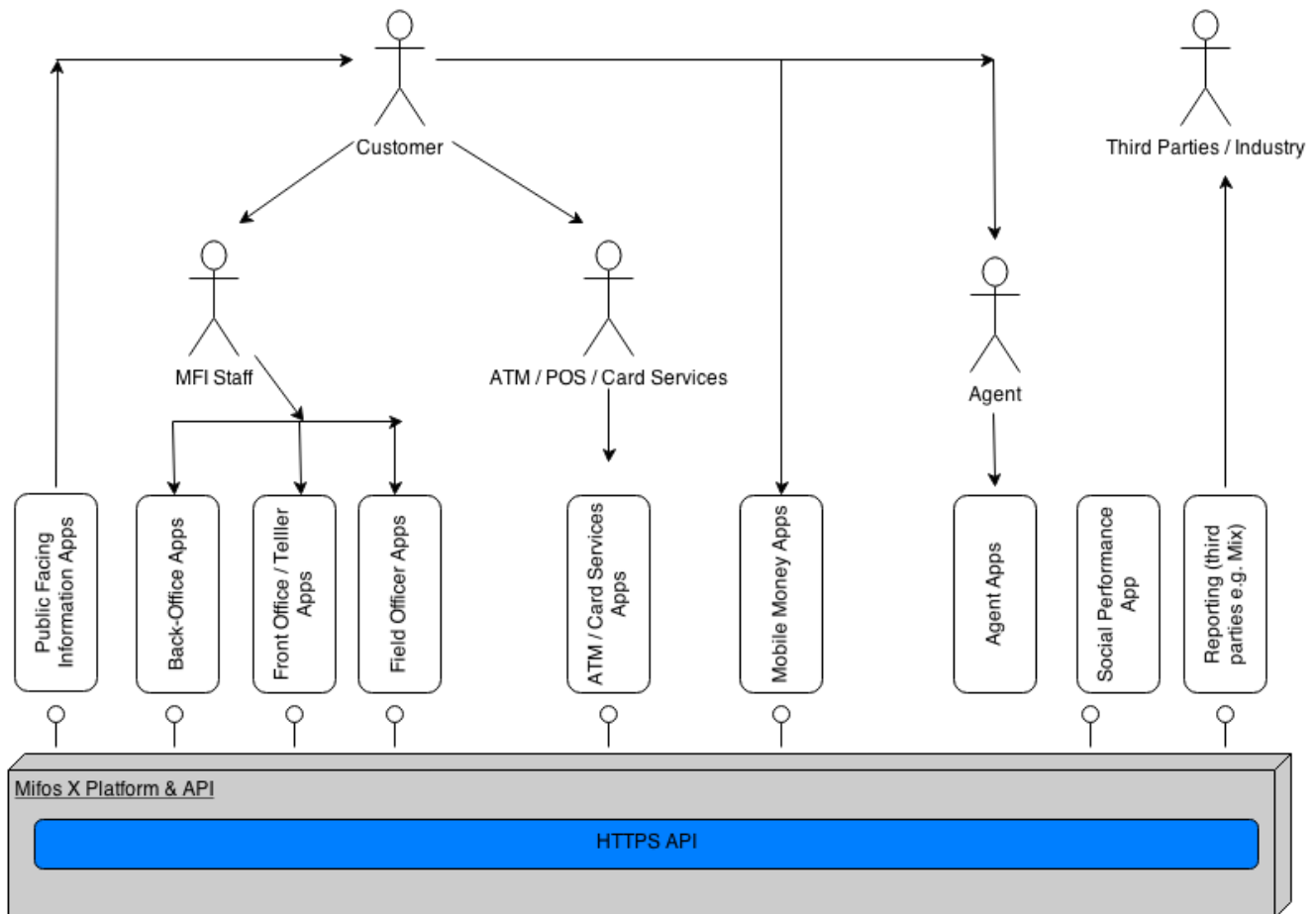


Figure 1. Platform System Overview

Financial institutions deliver their services to customers through a variety of means today.

- Customers can call direct into branches (teller model)
- Customers can organise into groups (or centers) and agree to meetup at a location and time with FI staff (traditional microfinance).
- An FI might have a public facing information portal that customers can use for variety of reasons including account management (online banking).
- An FI might be integrated into a ATM/POS/Card services network that the customer can use.
- An FI might be integrated with a mobile money operator and support mobile money services for customer (present/future microfinance).
- An FI might use third party agents to sell on products/services from other banks/FIs.

As illustrated in the above diagram, the various stakeholders leverage business apps to perform specific customer or FI related actions. The functionality contained in these business apps can be bundled up and packaged in any way. In the diagram, several of the apps may be combined into one app or any one of the blocks representing an app could be further broken up as needed.

The platform is the core engine of the MIS. It hides a lot of the complexity that exists in the business and technical domains needed for an MIS in FIs behind a relatively simple API. It is this API that frees up app developers to innovate and produce apps that can be as general or as bespoke as FIs need them to be.

Functional Overview

As ALL capabilities of the platform are exposed through an API, The API docs are the best place to view a detailed breakdown of what the platform does. See [online API Documentation](#).

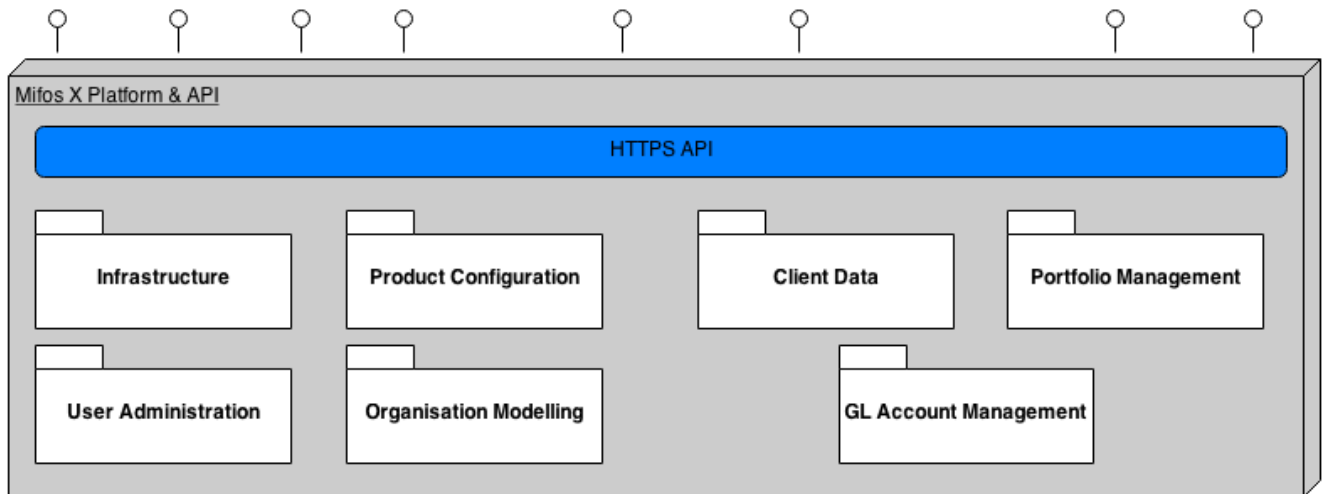


Figure 2. Platform Functional Overview

At a higher level though we see the capabilities fall into the following categories:

- Infrastructure
 - Codes
 - Extensible Data Tables
 - Reporting
- User Administration
 - Users
 - Roles
 - Permissions
- Organisation Modelling
 - Offices
 - Staff
 - Currency
- Product Configuration
 - Charges
 - Loan Products
 - Deposit Products
- Client Data
 - Know Your Client (KYC)
- Portfolio Management

- Loan Accounts
- Deposit Accounts
- Client/Groups
- GL Account Management
 - Chart of Accounts
 - General Ledger

Principles

RESTful API

The platform exposes all its functionality via a **practically-RESTful API**, that communicates using JSON.

We use the term **practically-RESTful** in order to make it clear we are not trying to be fully REST compliant but still maintain important RESTful attributes like:

- Stateless: platform maintains no conversational or session-based state. The result of this is ability to scale horizontally with ease.
- Resource-oriented: API is focussed around set of resources using HTTP vocabulary and conventions e.g GET, PUT, POST, DELETE, HTTP status codes. This results in a simple and consistent API for clients.

See online API Documentation for more detail.

Multi-tenanted

The Fineract platform has been developed with support for multi-tenancy at the core of its design. This means that it is just as easy to use the platform for Software-as-a-Service (SaaS) type offerings as it is for local installations.

The platform uses an approach that isolates an FI's data per database/schema (See Separate Databases and Shared Database, Separate Schemas).

Extensible

Whilst each tenant will have a set of core tables, the platform tables can be extended in different ways for each tenant through the use of Data tables functionality.

Command Query Separation

We separate **commands** (that change data) from **queries** (that read data).

Why? There are numerous reasons for choosing this approach which at present is not an attempt at full blown CQRS. The main advantages at present are:

- State changing commands are persisted providing an audit of all state changes.

- Used to support a general approach to **maker-checker**.
- State changing commands use the Object-Oriented paradigm (and hence ORM) whilst queries can stay in the data paradigm.

Maker-Checker

Also known as **four-eyes principal**. Enables apps to support a maker-checker style workflow process. Commands that pass validation will be persisted. Maker-checker can be enabled/disabled at fine-grained level for any state changing API.

Fine grained access control

A fine grained permission is associated with each API. Administrators have fine grained control over what roles or users have access to.

Package Structure

The intention is for platform code to be packaged in a vertical slice way (as opposed to layers).

Source code starts from github.com/apache/fineract/tree/develop/fineract-provider/src/main/java/org/apache/fineract

- accounting
- useradministration
- infrastructure
- portfolio
 - charge
 - client
 - fund
 - loanaccount
- accounting

Within each vertical slice is some common packaging structure:

- api - XXXApiResource.java - REST api implementation files
- handler - XXXCommandHandler.java - specific handlers invoked
- service - contains read + write services for functional area
- domain - OO concepts for the functional area
- data - Data concepts for the area
- serialization - ability to convert from/to API JSON for functional area

Design Overview



The implementation of the platform code to process commands through handlers whilst supporting maker-checker and authorisation checks is a little bit convoluted

at present and is an area pin-pointed for clean up to make it easier to on board new platform developers. In the mean time below content is used to explain its workings at present.

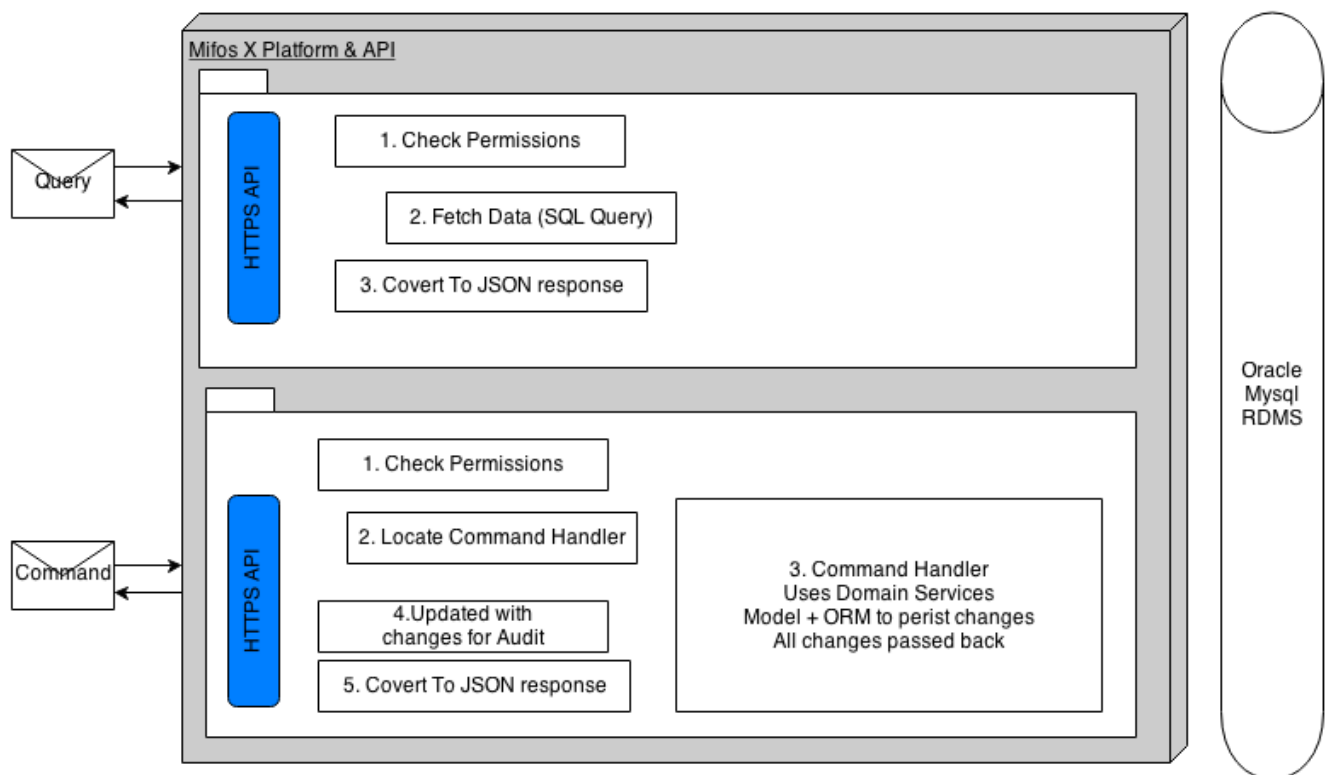


Figure 3. CQRS

Taking into account example shown above for the **users** resource.

- Query: GET /users
- HTTPS API: retrieveAll method on **org.apache.fineract.useradministration.api.UsersApiResource** invoked
- UsersApiResource.retrieveAll: Check user has permission to access this resources data.
- UsersApiResource.retrieveAll: Use 'read service' to fetch all users data ('read services' execute simple SQL queries against Database using JDBC)
- UsersApiResource.retrieveAll: Data returned to converted into JSON response
- Command: POST /users (Note: data passed in request body)
- HTTPS API: create method on **org.apache.fineract.useradministration.api.UsersApiResource** invoked

UsersApiResource.create

```

    return this.toApiJsonSerializer.serialize(result);
}

@PUT
@Path("/{userId}")
@Operation(summary = "Update a User", description = "Updates the user")

```

```

    @RequestBody(required = true, content = @Content(schema = @Schema(implementation =
UsersApiResourceSwagger.PutUsersUserIdRequest.class)))
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "OK", content = @Content
(schema = @Schema(implementation = UsersApiResourceSwagger.PutUsersUserIdResponse
.class))) })
    @Consumes({ MediaType.APPLICATION_JSON })
    @Produces({ MediaType.APPLICATION_JSON })
    public String update(@PathParam("userId") @Parameter(description = "userId") final
Long userId,
        @Parameter(hidden = true) final String apiRequestBodyAsJson) {

        final CommandWrapper commandRequest = new CommandWrapperBuilder() //
            .updateUser(userId) //
            .withJson(apiRequestBodyAsJson) //
            .build();

        final CommandProcessingResult result = this.
commandsSourceWritePlatformService.logCommandSource(commandRequest);

```

Create a CommandWrapper object that represents this create user command and JSON request body. Pass off responsibility for processing to PortfolioCommandSourceWritePlatformService.logCommandSource

```

        validateIsUpdateAllowed();

        final String json = wrapper.getJson();
        final JsonElement parsedCommand = this.fromApiJsonHelper.parse(json);
        JsonCommand command = JsonCommand.from(json, parsedCommand, this
.fromApiJsonHelper, wrapper.getEntityName(), wrapper.getEntityId(),
            wrapper.getSubentityId(), wrapper.getGroupId(), wrapper.getClientId(),
wrapper.getLoanId(), wrapper.getSavingsId(),
            wrapper.getTransactionId(), wrapper.getHref(), wrapper.getProductId(),
wrapper.getCreditBureauId(),
            wrapper.getOrganisationCreditBureauId(), wrapper.getJobName(),
wrapper.getLoanExternalId());

        return this.processAndLogCommandService.executeCommand(wrapper, command,
isApprovedByChecker);
    }

    @Override
    public CommandProcessingResult approveEntry(final Long makerCheckerId) {
        final CommandSource commandSourceInput = validateMakerCheckerTransaction
(makerCheckerId);
        validateIsUpdateAllowed();

        final CommandWrapper wrapper = CommandWrapper.fromExistingCommand
(makerCheckerId, commandSourceInput.getActionName(),
            commandSourceInput.getEntityName(), commandSourceInput.
getResourceId(), commandSourceInput.getSubResourceId(),

```

```

        commandSourceInput.getResourceGetUrl(), commandSourceInput
        .getProductId(), commandSourceInput.getOfficeId(),
        commandSourceInput.getGroupId(), commandSourceInput.getClientId(),
        commandSourceInput.getLoanId(),
        commandSourceInput.getSavingsId(), commandSourceInput.
        getTransactionId(), commandSourceInput.getCreditBureauId(),
        commandSourceInput.getOrganisationCreditBureauId(),
        commandSourceInput.getIdempotencyKey(),
        commandSourceInput.getLoanExternalId());
        final JsonElement parsedCommand = this.fromApiJsonHelper.parse
        (commandSourceInput.getCommandAsJson());
        final JsonCommand command = JsonCommand.fromExistingCommand(makerCheckerId,
        commandSourceInput.getCommandAsJson(), parsedCommand,
        this.fromApiJsonHelper, commandSourceInput.getEntityName(),
        commandSourceInput.getResourceId(),
        commandSourceInput.getSubResourceId(), commandSourceInput.
        getGroupId(), commandSourceInput.getClientId(),
        commandSourceInput.getLoanId(), commandSourceInput.getSavingsId(),
        commandSourceInput.getTransactionId(),
        commandSourceInput.getResourceGetUrl(), commandSourceInput
        .getProductId(), commandSourceInput.getCreditBureauId(),
        commandSourceInput.getOrganisationCreditBureauId(),
        commandSourceInput.getJobName(),
        commandSourceInput.getLoanExternalId());

        return this.processAndLogCommandService.executeCommand(wrapper, command,
        true);
    }

    @Transactional
    @Override
    public Long deleteEntry(final Long makerCheckerId) {

        validateMakerCheckerTransaction(makerCheckerId);
        validateIsUpdateAllowed();

        this.commandSourceRepository.deleteById(makerCheckerId);

        return makerCheckerId;
    }

    private CommandSource validateMakerCheckerTransaction(final Long makerCheckerId) {
        final CommandSource commandSource = this.commandSourceRepository.findById
        (makerCheckerId)
            .orElseThrow(() -> new CommandNotFoundException(makerCheckerId));
        if (!commandSource.isAwaitingApproval()) {
            throw new CommandNotAwaitingApprovalException(makerCheckerId);
        }
        AppUser appUser = this.context.authenticatedUser();
        String permissionCode = commandSource.getPermissionCode();
        appUser.validateHasCheckerPermissionTo(permissionCode);
    }

```

```

        if (!configurationService.isSameMakerCheckerEnabled() && !appUser
.isCheckerSuperUser()) {
            AppUser maker = commandSource.getMaker();
            if (maker == null) {
                throw new UnsupportedOperationException(permissionCode, "Maker user is
missing.");
            }
        }
    }

```

Check user has permission for this action. if ok, a) parse the json request body, b) create a JsonCommand object to wrap the command details, c) use CommandProcessingService to handle command

```

private final FineractRequestContextHolder fineractRequestContextHolder;
private final Gson gson = GoogleGsonSerializerHelper.createSimpleGson();

private CommandProcessingResult retryWrapper(Supplier<CommandProcessingResult>
supplier) {
    try {
        if (!BatchRequestContextHolder.isEnclosingTransaction()) {
            return retryConfigurationAssembler
.getRetryConfigurationForExecuteCommand().executeSupplier(supplier);
        }
        return supplier.get();
    } catch (RuntimeException e) {
        return fallbackExecuteCommand(e);
    }
}

@Override
public CommandProcessingResult executeCommand(final CommandWrapper wrapper, final
JsonCommand command,
        final boolean isApprovedByChecker) {
    return retryWrapper(() -> {
        // Do not store the idempotency key because of the exception handling
        setIdempotencyKeyStoreFlag(false);

        Long commandId = (Long) fineractRequestContextHolder.getAttribute
(COMMAND_SOURCE_ID, null);
        boolean isRetry = commandId != null;
        boolean isEnclosingTransaction = BatchRequestContextHolder
.isEnclosingTransaction();

        CommandSource commandSource = null;
        String idempotencyKey;
        if (isRetry) {
            commandSource = commandSourceService.getCommandSource(commandId);
            idempotencyKey = commandSource.getIdempotencyKey();
        } else if ((commandId = command.commandId()) != null) { // action on the
command itself
            commandSource = commandSourceService.getCommandSource(commandId);
            idempotencyKey = commandSource.getIdempotencyKey();
        } else {

```



```

        idempotencyKey = idempotencyKeyResolver.resolve(wrapper);
    }
    exceptionWhenTheRequestAlreadyProcessed(wrapper, idempotencyKey, isRetry);

    AppUser user = context.authenticatedUser(wrapper);
    if (commandSource == null) {
        if (isEnclosingTransaction) {
            commandSource = commandSourceService.getInitialCommandSource
(wrapper, command, user, idempotencyKey);
        } else {
            commandSource = commandSourceService.saveInitialNewTransaction
(wrapper, command, user, idempotencyKey);
            commandId = commandSource.getId();
        }
    }
    if (commandId != null) {
        storeCommandIdInContext(commandSource); // Store command id as a
request attribute
    }

    setIdempotencyKeyStoreFlag(true);

    return executeCommand(wrapper, command, isApprovedByChecker,
commandSource, user, isEnclosingTransaction);
});
}

private CommandProcessingResult executeCommand(final CommandWrapper wrapper, final
JsonCommand command,
        final boolean isApprovedByChecker, CommandSource commandSource, AppUser
user, boolean isEnclosingTransaction) {

    final CommandProcessingResult result;
    try {
        result = commandSourceService.processCommand(findCommandHandler(wrapper),
command, commandSource, user, isApprovedByChecker);
    } catch (Throwable t) { // NOSONAR
        RuntimeException mappable = ErrorHandler.getMappable(t);
        ErrorInfo errorInfo = commandSourceService.generateErrorInfo(mappable);
    }
}

```



if a RollbackTransactionAsCommandIsNotApprovedByCheckerException occurs at this point. The original transaction will of been aborted and we only log an entry for the command in the audit table setting its status as 'Pending'.

- Check that if maker-checker configuration enabled for this action. If yes and this is not a 'checker' approving the command - rollback at the end. We rollback at the end in order to test if the command will pass 'domain validation' which requires commit to database for full check.
- findCommandHandler - Find the correct Handler to process this command.
- Process command using handler (In transactional scope).

- `CommandSource` object created/updated with all details for logging to 'm_portfolio_command_source' table.
- In update scenario, we check to see if there were really any changes/updates. If so only JSON for changes is stored in audit log.

Persistence

TBD

Database support

Fineract supports multiple databases:

- MySQL compatible databases (e.g. MariaDB)
- PostgreSQL

The platform differentiates between these database types in certain cases when there's a need to use some database specific tooling. To do so, the platform examines the JDBC driver used for running the platform and tries to determine which database is being used.

The currently supported JDBC driver and corresponding mappings can be found below.

JDBC driver class name	Resolved database type
<code>org.mariadb.jdbc.Driver</code>	MySQL
<code>com.mysql.jdbc.Driver</code>	MySQL
<code>org.postgresql.Driver</code>	PostgreSQL

The actual code can be found in the `DatabaseTypeResolver` class.

Tenant database security

The tenant database schema password is stored in the `tenant_server_connections` table in the tenant database. The password and the read only schema password are encrypted using the `fineract.tenant.master-password` property. By default, the database property will be encrypted in the first start from a plain text.

When you want to generate a new encrypted password, you can use the `org.apache.fineract.infrastructure.core.service.database.DatabasePasswordEncryptor` class.

Database password encryption usage

```
java -cp fineract-provider.jar \
  -Dloader.main
=org.apache.fineract.infrastructure.core.service.database.DatabasePasswordEncryptor \
  org.springframework.boot.loader.PropertiesLauncher \
  <masterPassword> \
```

```
<plainPassword>
```

For example:

```
java -cp fineract-provider-0.0.0-48f7e315.jar \  
  
-Dloader.main=org.apache.fineract.infrastructure.core.service.database.DatabasePasswor  
dEncryptor \  
  org.springframework.boot.loader.PropertiesLauncher \  
  fineract-master-password \  
  fineract-tenant-password  
The encrypted password:  
VLwGl7vOP/q275ZTku+PNGWnGwW4mzzNHSNa09Pr67WT5/NZMpBr9tGYYiYsqwL1eRew2j1703/N1EFbL1XhSA  
==
```

Data-access layer

The data-access layer of Fineract is implemented by using JPA (Java Persistence API) with the EclipseLink provider.

Despite the fact that JPA is used quite extensively in the system, there are cases where the performance is a key element for an operation therefore you can easily find native SQLs as well.

The data-access layer of Fineract is compatible with different databases. Since a lot of the native queries are using specific database functions, a wrapper class - [DatabaseSpecificSQLGenerator](#) - has been introduced to handle these database specifics. Whenever there's a need to rely on new database level functions, make sure to extend this class and implement the specific functions provided by the database.

Fineract has been developed for 10+ years by the community and unfortunately there are places where entity relationships are configured with [EAGER](#) fetching strategy. This must not confuse anybody. The long-term goal is to use the [LAZY](#) fetching strategy for every single relationship. If you're about to introduce a new one, make sure to use [LAZY](#) as a fetching strategy, otherwise your PR will be rejected.

Database schema migration

As for every system, the database structure will and need to evolve over time. Fineract is no different. Originally for Fineract, Flyway was used until Fineract 1.6.x.

After 1.6.x, PostgreSQL support was added to the platform hence there was a need to make the data-access layer and the schema migration as database independent as possible. Because of that, from Fineract 1.7.0, Flyway is not used anymore but Liquibase is.

Some of the changesets in the Liquibase changelogs have database specifics into it but they only run for the relevant databases. This is controller by Liquibase contexts.

The currently available Liquibase contexts are:

- `mysql` - only set when the database is a MySQL compatible database (e.g. MariaDB)
- `postgresql` - only set when the database is a PostgreSQL database
- configured Spring active profiles
- `tenant_store_db` - only set when the database migration runs the Tenant Store upgrade
- `tenant_db` - only set when the database migration runs the Tenant upgrade
- `initial_switch` - this is a technical context and should **NOT** be used

The switch from Flyway (1.6.x) to Liquibase (1.7.x) was planned to be as smooth as possible so there's no need for manual work hence the behavior is described as following:

- If the database is empty, Liquibase will create the database schema from scratch
- If the database contains the latest Fineract 1.6.x database structure which was previously migrated with Flyway. Liquibase will seamlessly upgrade it to the latest version. Note: the Flyway related 2 database tables are left as they are and are not deleted.
- If the database contains an earlier version of the database structure than Fineract 1.6.x. Liquibase will **NOT** do anything and **will fail the application during startup**. The proper approach in this case is to first upgrade your application version to the latest Fineract 1.6.x so that the latest Flyway changes are executed and then upgrade to the newer Fineract version where Liquibase will seamlessly take over the database upgrades.

Troubleshooting

1. During upgrade from Fineract 1.5.0 to 1.6.0, Liquibase fails

After dropping the flyway migrations table (schema_version), Liquibase runs its own migrations which fails (in recreating tables which already exist) because we are aiming to re-use DB with existing data from Fineract 1.5.0.

Solution: The latest release version (1.6.0) doesn't have Liquibase at all, it still runs Flyway migrations. Only the develop branch (later to be 1.7.0) got switched to Liquibase. Do not pull the develop before upgrading your instance.

Make sure first you upgrade your instance (aka database schema with Fineract 1.6.0). Then upgrade with the current develop branch. Check if some migration scripts did not run which led to some operations failing due to slight differences in schema. Try with running the missing migrations manually.

Note: develop is considered unstable until released.

2. Upgrading database from MySQL 5.7 as advised to Maria DB 10.6, fails. If we use data from version 18.03.01 it fails to migrate the data. If we use databases running on 1.5.0 release it completes the startup but the system login fails.

Solution: A database upgrade is separate thing to take care of.

3. We are getting `SchemaUpgradeNeededException: Make sure to upgrade to Fineract 1.6 first and then to a newer version` error while upgrading to tag 1.6.

1.6 version shouldn't include Liquibase. It will only be released after 1.6. Make sure Liquibase is dropping `schema_version` table, as there is no Flyway it is not required. Drop Flyway and use Liquibase for both migrations and database independence. In case, if you still get errors, you can use git SHA `746c589a6e809b33d68c0596930fcaa7338d5270` and Flyway migration will be done to the latest.

```
TENANT_LATEST_FLYWAY_VERSION = 392;  
TENANT_LATEST_FLYWAY_SCRIPT_NAME =  
"V392__interest_recovery_conf_for_reschedule.sql";  
TENANT_LATEST_FLYWAY_SCRIPT_CHECKSUM = 1102395052;
```

Idempotency

Idempotency is the way to make sure your specific action is only executed once.

For example, if you have a button that is supposed to send a repayment, you don't want to repayment twice if the user clicks the button twice. Idempotency is a way to make sure that the action is only executed once.

There are two ways to use idempotency:

- HTTP Request with idempotency key header
- Batch request with batch item header

How it works

The `idempotency key` with `action name` and `entity name` is unique, and identify a specific command in the system.

If no idempotency key is assigned to the request, the system will generate one for you.

1. User send a request
2. The system checks there are already executed commands with the same `idempotency key` and `action name` and `entity name`
3. The action based on the result of the check
 - If the request is completed the system return with the already generated result
 - If not completed, return HTTP 409 response
 - If the request is not completed, we process the requests and store the results in the database

Idempotency in HTTP requests

To achieve idempotency in HTTP requests, you can use the HTTP header from `fineract.idempotency-key-header-name` configuration variables (default `Idempotency-Key`). This header is a unique identifier for the request. If you send the same request twice, the second request will be ignored and the response from the first request will be returned.

Idempotency in Batch requests

In batch requests, you can set the idempotency key for every batch item, in the batch item **header** fields. The header key is from `fineract.idempotency-key-header-name` configuration variables (default `Idemptency-Key`).

Result of the request

- When the request is already executed and completed, the system will return a `x-served-from-cache` header with the value `true` in the response and return the original request body.
- When the request is already executed but still not completed, the system will return to HTTP 409 error code
- When the request is not executed, the system runs it normally and stores the result in the date

Validation

Programmatic

Use the `DataValidatorBuilder`, e.g. like so:

```
new DataValidatorBuilder().resource("fileUpload")
    .reset().parameter("Content-Length").value(contentLength).notBlank()
    .integerGreaterThanNumber(0)
    .reset().parameter("FormDataContentDisposition").value(fileDetails).notNull()
    .throwValidationErrors();
```

Such code is often encapsulated in `*Validator` classes (if more than a few lines, and/or reused from several places; avoid copy/paste), like so:

```
public class YourThingValidator {

    public void validate(YourThing thing) {
        new DataValidatorBuilder().resource("yourThing")
            ...
            .throwValidationErrors();
    }
}
```

Declarative

[FINERACT-1229](#) is an open issue about adopting Bean Validation for *declarative* instead of *programmatic* (as above) validation. Contributions welcome!

Batch execution and jobs

Just like any financial system, Fineract also has batch jobs to achieve some processing on the data that's stored in the system.

The batch jobs in Fineract are implemented using [Spring Batch](#). In addition to the Spring Batch ecosystem, the automatic scheduling is done by the [Quartz Scheduler](#) but it's also possible to trigger batch jobs via regular APIs.

Glossary

Job	A Job is an object that encapsulates an entire batch process.
Step	A Step is an object that encapsulates an independent phase of a Job.
Chunk oriented processing	Chunk oriented processing refers to reading the data one at a time and creating 'chunks' that are written out within a transaction boundary.
Partitioning	Partitioning refers to the high-level idea of dividing your data into so called partitions and distributing the individual partitions among Workers. The splitting of data and pushing work to Workers is done by a Manager.
Remote partitioning	Remote partitioning is a specialized partitioning concept. It refers to the idea of distributing the partitions among multiple JVMs mainly by using a messaging middleware.
Manager node	The Manager node is one of the objects taking a huge part when using partitioning. The Manager node is responsible for dividing the dataset into partitions and keeping track of all the divided partitions' Worker execution. When all Workers nodes are done with their partitions, the Manager will mark the corresponding Job as completed.
Worker node	A Worker node is the other important party in the context of partitioning. The Worker node is the one executing the work needed for a single partition.

Batch jobs in Fineract

Types of jobs

The jobs in Fineract can be divided into 2 categories:

- Normal batch jobs
- Partitionable batch jobs

Most of the jobs are normal batch jobs with limited scalability because Fineract is still passing through the evolution on making most of them capable to process a high-volume of data.

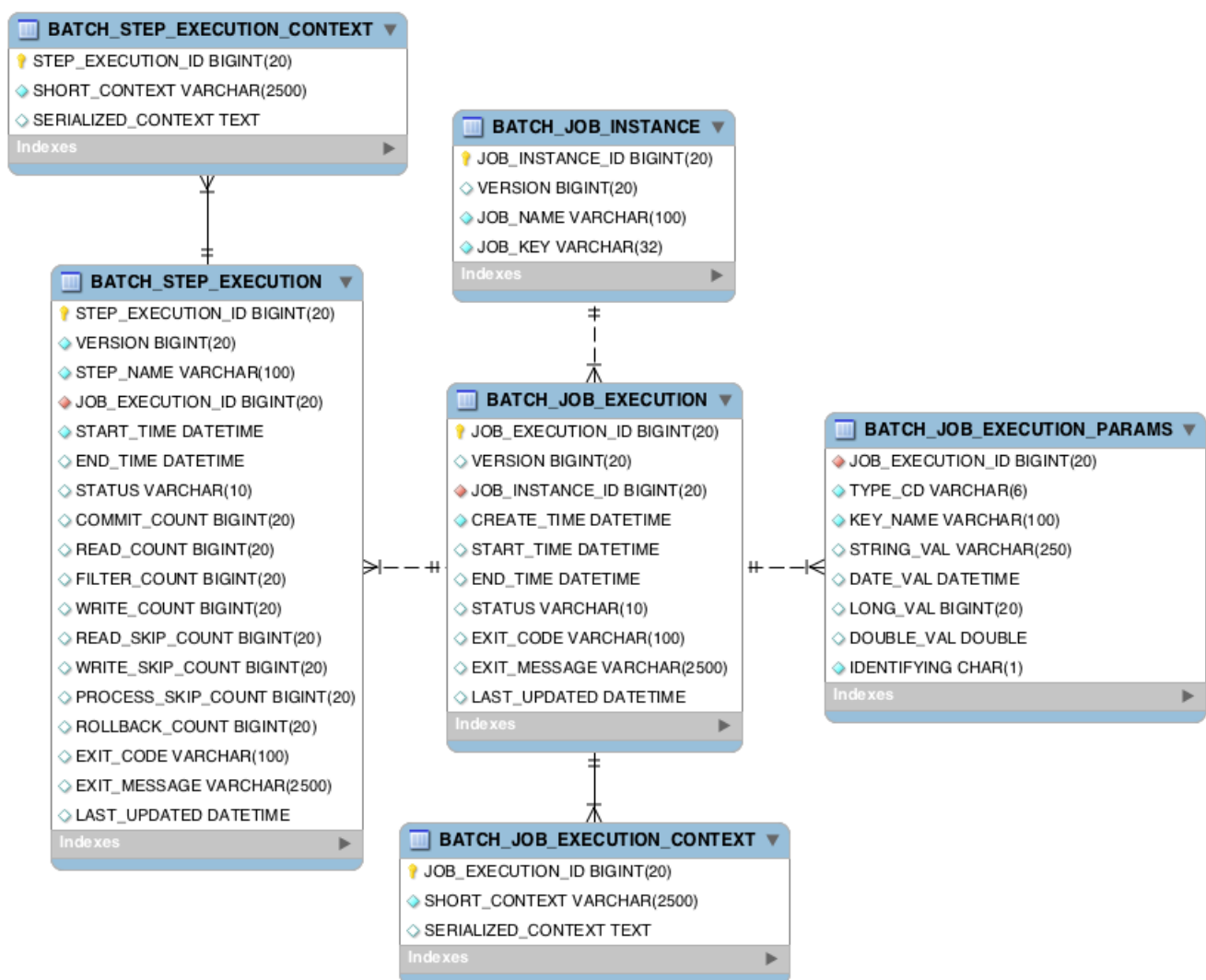
List of jobs

Job name	Active by default	Partitionable	Description
LOAN_CLOSE_OF_BUSINESS	No	Yes	TBD

Batch job execution

State management

State management for the batch jobs is done by the Spring Batch provided state management. The data model consists of the following database structure:



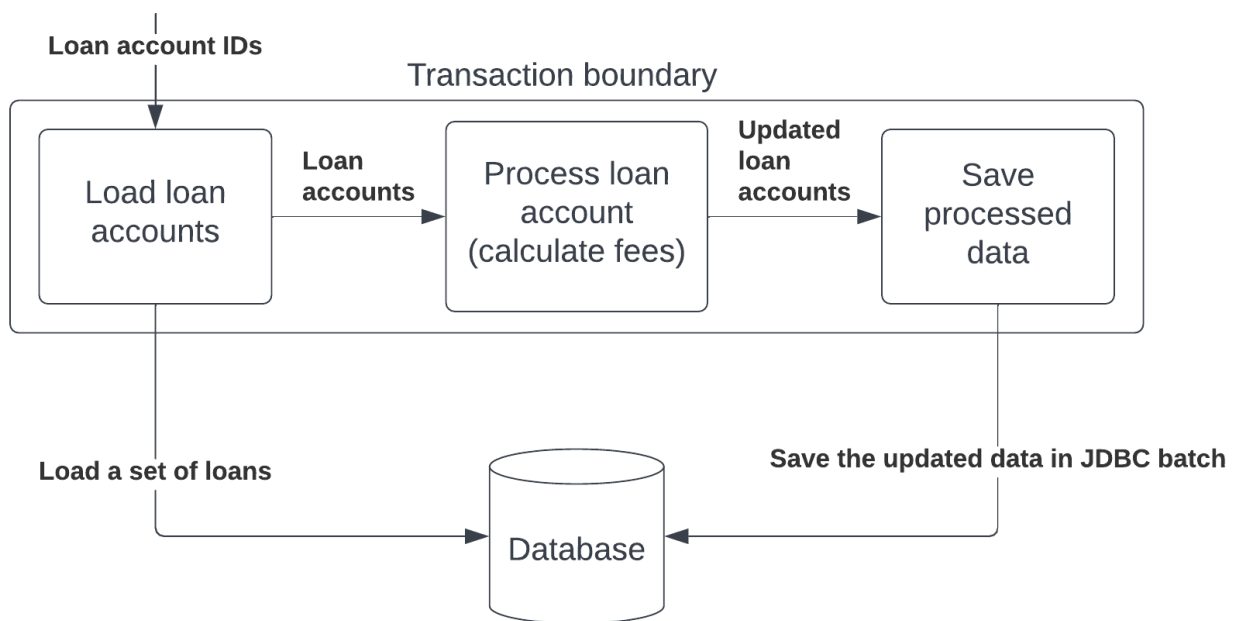
The corresponding database migration scripts are shipped with the Spring Batch core module under the `org.springframework.batch.core` package. They are only available as native scripts and

are named as `schema-.sql` where is the short name of the database platform. For MySQL it's called `schema-mysql.sql` and for PostgreSQL it's called `schema-postgresql.sql`. When Fineract is started, the database dependent schema SQL script will be picked up according to the datasource configurations.

Chunk oriented processing

Chunking data has not been easier. Spring Batch does a really good job at providing this capability.

In order to save resources when starting/committing/rollbacking transactions for every single processed item, chunking shall be used. That way, it's possible to mark the transaction boundaries for a single processed chunk instead of a single item processing. The image below describes the flow with a very simplistic example.



In addition to not opening a lot of transactions, the processing could also benefit from JDBC batching. The last step - writing the result into the database - collects all the processed items and then writes it to the database; both for MySQL and PostgreSQL (the databases supported by Fineract) are capable of grouping multiple DML (INSERT/UPDATE/DELETE) statements and sending them in one round-trip, optimizing the data being sent over the network and granting the possibility to the underlying database engine to enhance the processing.

Remote partitioning

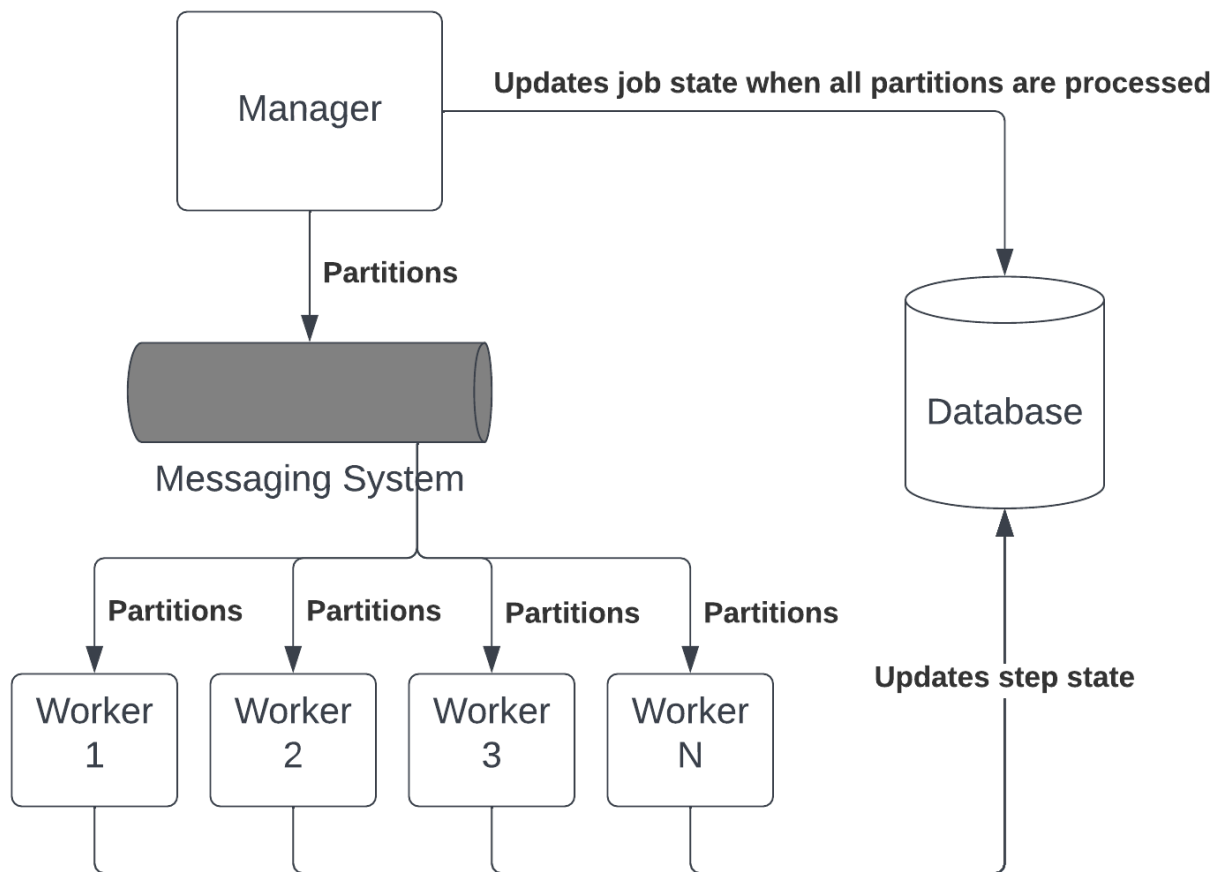
Spring Batch provides a really nice way to do remote partitioning. The 2 type of objects in this setup is a manager node - who splits and distributes the work - and a number of worker nodes - who picks up the work.

In remote partitioning, the worker instances are receiving the work via a messaging system as soon as the manager splits up the work into smaller pieces.

Remote partitioning could be done 2 ways in terms of keeping the job state up-to-date. The main

difference between the two is how the manager is notified about partition completions.

One way is that they share the same database. When the worker does something to a partition - for example picks it up for processing - it updates the state of that partition in the database. In the meantime, the manager regularly polls the database until all partitions are processed. This is visualized in the below diagram.



An alternative approach to this - when the database is not intended to be shared between manager and workers - is to use a messaging system (could be the same as for distributing the work) and the workers could send back a message to the manager instance, therefore notifying it about failure/completion. Then the manager can simply keep the database state up-to-date.

Even though the alternative solution decouples the workers even better, we thought it's not necessary to add the complexity of handling reply message channel to the manager.

Also, please note that the partitioned job execution is multitenant meaning that the workers will receive which tenant it should do the processing for.

Supported message channels

For remote partitioning, the following message channels are supported by Fineract:

- Any JMS compatible message channels (ActiveMQ, Amazon MQ, etc)
- Apache Kafka

Fault-tolerance scenarios

There are multiple fault tolerance use-cases that this solution must and will support:

1. If the manager fails during partitioning
2. If the manager completes the partitioning and the partition messages are sent to the broker but while the manager is waiting for the workers to finish, the manager fails
3. If the manager runs properly and during a partition processing a worker instance fails

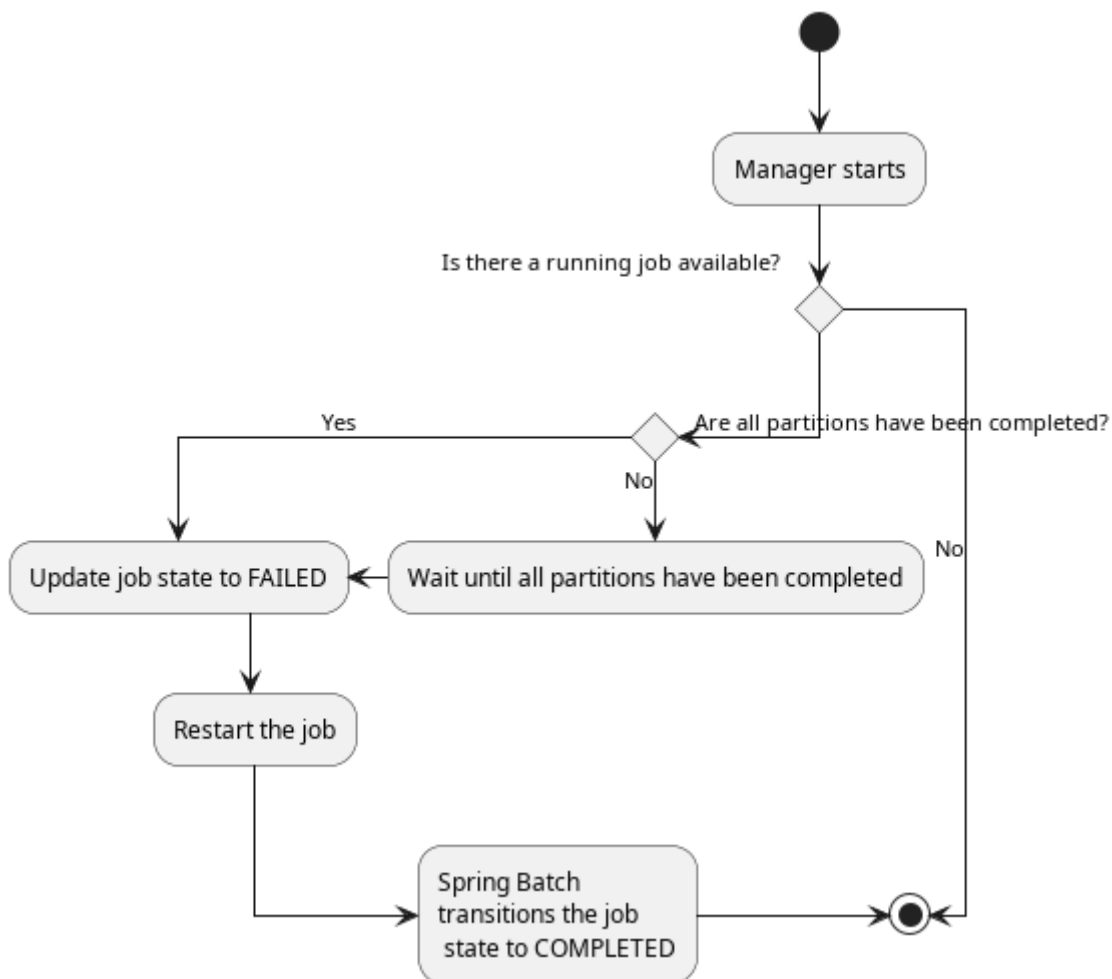
In case of scenario 1), the simple solution is to re-trigger the job via API or via the Quartz scheduler.

In case of scenario 2), there's no out-of-the-box solution by Spring Batch. Although there's a custom mechanism in place that'll resume the job upon restarting the manager. There are 2 cases in the context of this scenario:

- If all the partitions have been successfully processed by workers
- If not all the partitions have been processed by the workers

In the first case, we'll simply mark the stuck job as **FAILED** along with its partitioning step and instruct Spring Batch to restart the job. The behavior in this case will be that Spring Batch will spawn a new job execution but will notice that the partitions have all been completed so it's not going to execute them once more.

In the latter case, the same will happen as for the first one but before marking the job execution as **FAILED**, we'll wait until all partitions have been completed.



In case of scenario 3), another worker instance will take over the partition since it hasn't been finished.

Configurable batch jobs

There's another type of distinction on the batch jobs. Some of them are configurable in terms of their behavior.

The currently supported configurable batch jobs are the following:

- LOAN_CLOSE_OF_BUSINESS

The behavior of these batch jobs are configurable. There's a new terminology we're introducing called **business steps**.

Business steps

Business steps are a smaller unit of work than regular Spring Batch Steps and the two are not meant to be mixed up because there's a large difference between them.

A Spring Batch Step's main purpose is to decompose a bigger work into smaller ones and making sure that these smaller Steps are properly handled within a single database transaction.

In case of a business step, it's a smaller unit of work. Business steps live **within** a Spring Batch Step. Fundamentally, they are simple classes that are implementing an interface with a single method

that contains the business logic.

Here's a very simple example:

```
public class MyCustomBusinessStep implements BusinessStep<Loan> {
    @Override
    public Loan process(Loan loan) {
        // do something
    }
}
```

```
public class LoanCOBItemProcessor implements ItemProcessor<Loan, Loan> {
    @Override
    public Loan process(Loan loan) {
        List<BusinessStep<Loan>> bSteps = getBusinessSteps();
        Loan result = loan;
        for (BusinessStep<Loan> bStep : bSteps) {
            result = bStep.process(result);
        }
        return result;
    }
}
```

Business step configuration

The business steps are configurable for certain jobs. The reason for that is because we want to allow the possibility for Fineract users to configure their very own business logic for generic jobs, like the Loan Close Of Business job where we want to do a formal "closing" of the loans at the end of the day.

All countries are different with a different set of regulations. However in terms of behavior, there's no all size fits all for loan closing.

For example in the United States of America, you might need the following logic for a day closing:

1. Close fully repaid loan accounts
2. Apply penalties
3. Invoke IRS API for regulatory purposes

While in Germany it should be:

1. Close fully repaid loan accounts
2. Apply penalties
3. Do some fraud detection on the account using an external service
4. Invoke local tax authority API for regulatory purposes

These are just examples, but you get the idea.

The business steps are configurable through APIs:

Retrieving the configuration for a job:

```
GET /fineract-provider/api/v1/jobs/{jobName}/steps?tenantIdentifier={tenantId}
HTTP 200

{
  "jobName": "LOAN_CLOSE_OF_BUSINESS",
  "businessSteps": [
    {
      "stepName": "APPLY_PENALTY_FOR_OVERDUE_LOANS",
      "order": 1
    },
    {
      "stepName": "LOAN_TAGGING",
      "order": 2
    }
  ]
}
```

Updating the business step configuration for a job:

```
PUT /fineract-provider/api/v1/jobs/{jobName}/steps?tenantIdentifier={tenantId}

{
  "businessSteps": [
    {
      "stepName": "LOAN_TAGGING",
      "order": 1
    },
    {
      "stepName": "APPLY_PENALTY_FOR_OVERDUE_LOANS",
      "order": 2
    }
  ]
}
```

The business step configuration for jobs are tracked within the database in the `m_batch_business_steps` table.

Inline Jobs

Some jobs that work with business entities have a corresponding job that can trigger the job with a list of specified entities.

When the Inline job gets triggered then the corresponding existing job will run in real time with the

given entities as a dataset.

List of Inline jobs

Inline Job name	Corresponding Job
LOAN_COB	LOAN_CLOSE_OF_BUSINESS

Triggering the Inline Loan COB Job:

```
POST /fineract-provider/api/v1/jobs/LOAN_COB/inline?tenantIdentifier={tenantId}

{
  "loanIds": [
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
  ]
}
```

In this case the Loan COB job will work only with the given loan IDs.

Global Configuration for enabling/disabling jobs

Some jobs can be enabled/disabled with global configuration.

If the job is disabled with the global configuration then it cannot be scheduled and cannot be triggered via API.

List of jobs with global configuration

Job name	Application property	Environment variable	Default value
LOAN_CLOSE_OF_BUSINESS	fineract.job.loan-cob-enabled	FINERACT_JOB_LOAN_COB_ENABLED	true

Loan account locking

Keeping a consistent state of loan accounts become quite important when we start talking about doing a business day closing each day for loans.

There are 2 concepts for loan account locking:

1. Soft-locking loan accounts
2. Hard-locking loan accounts

Soft-locking simply means that when the Loan COB has been kicked off but workers not yet processing the chunk of loan accounts (i.e. the partition is waiting in the queue to be picked up) and during this time a real-time write request (e.g. a repayment/disbursement) comes in through the API, we simply do an "inlined" version of the Loan COB for that loan account. From a practical standpoint this will mean that before doing the actual repayment/disbursement on the loan account on the API, we execute the Loan COB for that loan account, kind of like prioritizing it.

Hard-locking means that when a worker picks up the loan account in the chunk, real-time write requests on those loan accounts will be simply rejected with an **HTTP 409**.

The locking is strictly tied to the Loan COB job's execution but there could be other processes in the future which might want to introduce new type of locks for loans.

The loan account locking is solved by maintaining a database table which stores the locked accounts, it's called **m_loan_account_locks**.

When a loan account is present in the table above, it simply means there's a lock applied to it and whether it's a soft or hard lock can be determined by the **lock_owner** column.

And when a loan account is locked, loan related write API calls will be either rejected or will trigger an inline Loan COB execution. There could be a corner case here when the Loan COB fails to process some loan accounts (due to a bug, inconsistency, etc) and the loan accounts stay locked. This is an intended behavior to mark loans which are not supposed to be used until they are "fixed".

Since the fixing might involve making changes to the loan account via API (for example doing a repayment to fix the loan account's inconsistent state), we need to allow those API calls. Hence, the lock table includes a **bypass_enabled** column which disables the lock checks on the loan write APIs.

Technology

- Java: www.oracle.com/technetwork/java/javase/downloads/index.html
- JAX-RS using Jersey
- JSON using Google GSON
- Spring I/O Platform: spring.io/platform
 - Spring Framework
 - Spring Boot
 - Spring Security
 - Spring Data (JPA) backed by EclipseLink
- MySQL: www.oracle.com/us/products/mysql/overview/index.html
- PostgreSQL

TBD

Modules

We are currently working towards a fully modular codebase and will publish more here when we are ready.



Even if we are not quite there yet with full modularity you can already create your own custom modules to extend Fineract. Please see chapter [Custom Modules](#).

Introducing Business Date into Fineract - Community version

Business date as a concept does not exist as of now in Fineract. It would be business critical to add such a functionality to support various banking functionalities like “Closing of Business day”, “Having Closing of Business day relevant jobs”, “Supporting logical date management”.

Glossary

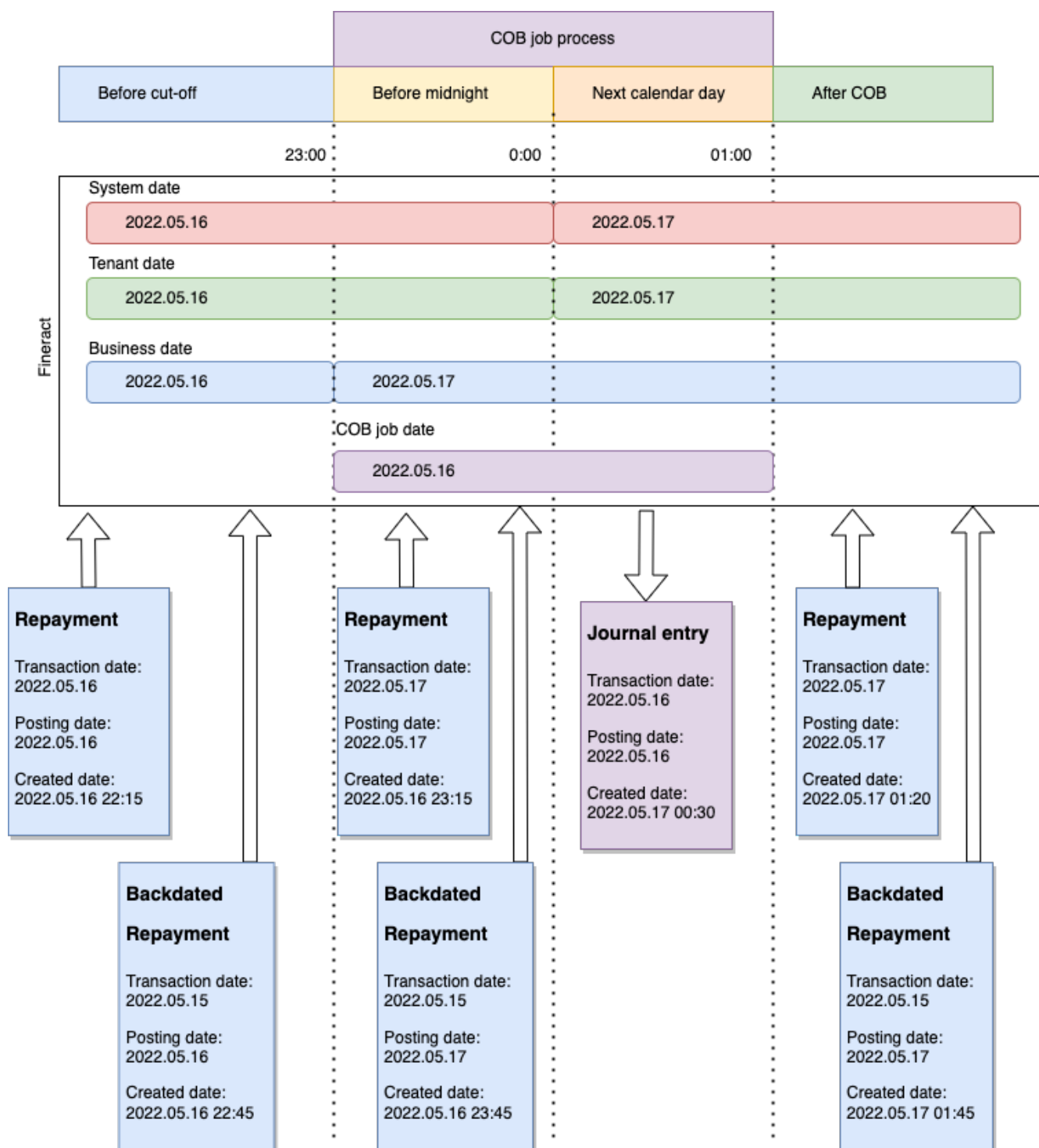
*COB	Close of Business; concept of closing a business day
*Business day	Timeframe that logically group together actions on a particular business date
*Business date	Logical date; its value is not tied to the physical calendar. Represents a business day
*Cob date	Logical date; Represents the business date for actions during COB job execution
*Created date	When the transaction was created (audit purposes). Date + time
*Last modified date	When the transaction was last modified (audit purposes). Date + time
*Submitted on date / Posting date	When the transaction was posted. Tenant date or business date (depends on whether the logical date concept was introduced or not)
*Transaction date / Value date	The date on which the transaction occurred or to be accounted for

Current behaviour

- Fineract support 3 types of dates:
 - System date
 - Physical/System date of the running environment
 - Tenant date
 - Timezoned version of the above system date
 - User-provided date
 - Based on the provided date (as string) and the provided date format
- There is no support of logical date concept
 - Independent from the system / tenant date
- Jobs are scheduled against system date (CRON), but aligned with the tenant timezone.
- During the job execution all the data and transactions are using the actual tenant date

- It could happen some transactions are written for 17th of May and other for 18th of May, if the job was executed around midnight
- There is no support of COB
 - No backdated transactions by jobs
 - There is no support to logically group together transactions and store them with the same transaction date which is independent of the physical calendar of the tenant
- All the transactions and business logic are tied to a physical calendar

Business date



Design

By introducing the business day concept we are not tied anymore to the physical calendar of the system or the tenant. We got the ability to define our own business day boundaries which might end 15 minutes before midnight and any incoming transactions after the cutoff will be accounted for the following business day.

It is a logical date which makes it possible to separate the business day from the physical calendar:

- Close a business day before midnight
- Close a business day at midnight
- Close a business day after midnight

Closing a Business Day could be a longer process (see COB jobs) meanwhile some processes shall still be able to create transactions for that business day (COB jobs), but others are meant to create the transactions for the next (incoming transactions): Business date concept is there to sort that out.

Business date concept is essential when:

- Having COB jobs:
 - When the COB was triggered:
 - All the jobs which processing the data must still accounted for actual business day
 - All the incoming transactions must be accounted to the next business day
- Business day is ending before / after midnight (tenant date / system date)
- Testing purposes:
 - Since the transactions and job execution is not tied anymore to a physical calendar, we can easily test a whole loan lifecycle by altering the business date
- Handling disruption of service: For any unseen reason the system goes down or there are any disruption in the workflow, the “missed days” can easily be processed one by one as nothing happened
 - There is a disruption at 2022-06-02
 - The issue is fixed by 2022-06-05
 - The COB flow can be executed for 2022-06-03 and when it is finished for 2022-06-04 and after when the time arrives for 2022-06-05

This logical date is manageable via:

- Job
- API

To maintain such separation from physical calendar we need to introduce the following new dates:

- Business date
- COB date

- Can be calculated based on the actual business date
 - Depend on COB date strategy (see below)

Business date

The - logical - date of the actual business day, eg: 2022-05-06

- It does not support time parts
- It can be managed manually (via API call) or automatically (via scheduled job)
- All business actions during the business day shall use this date:
 - Posting / submitted on date of transactions
 - Submitted on date of actions
 - (Regular) jobs
- It will be used in every situation where the transaction date / value date is not provided by the user or the user provided date shall be validated.
 - Opening date
 - Closing date
 - Disbursal date
 - Transaction/Value date
 - Posting/Submitted date
 - Reversal date
- Will not be use for audit purposes:
 - Created on date
 - Updated on date

COB date

The - logical - date of the business day for job execution, eg: 2022-05-05

- It can be calculated based on the business date
 - COB date = business date - 1 day
 - Automatically modified alongside with the business date change
- It does not support time parts
- It is automatically managed by business date change
 - Configurable
- It is used only via COB job execution
 - When we create / modify any business data during the COB job execution, the COB date is to be used:
 - Posting date of transactions

- Submitted on date of actions
- Transaction / value date of any actions

Some basic example

Apply for a loan

#1

Tenant date: 2022-05-23 14:22:12

Business date: 2022-05-22

Submitted on date: 2022-05-23

Outcome: **FAIL**

Message: **The date on which a loan is submitted cannot be in the future.**

Reason: Even the tenant date is 2022-05-23, but the business date was 2022-05-22 which means anything further that date must be considered as a future date.

#2

Tenant date: 2022-05-23 14:22:12

Business date: 2022-05-22

Submitted on date: 2022-05-22

Outcome: **SUCCESS**

Loan application details:

- Submitted on date: 2022-05-22

Repayment for a loan

#1

Tenant date: 2022-05-25 11:22:12

Business date: 2022-05-24

Transaction date: 2022-05-25

Outcome: **FAIL**

Message: **The transaction date cannot be in the future.**

Reason: Even the physical date is 2022-05-25, but the business date was 2022-05-24 which means anything further that date must be considered as a future date.

#2

Tenant date: 2022-05-25 11:22:12

Business date: 2022-05-24

Transaction date: 2022-05-23

Outcome: **SUCCESS**

Loan transaction details:

- Submitted on date: 2022-05-24
- Transaction date: 2022-05-23
- Created on date: 2022-05-25 11:22:12

Changes in Fineract

We shall modify at all the relevant places where the tenant date was used:

- With very limited exceptions all places where the tenant date is used we need to modify to use the business date.
- Replace system date with tenant date or business date (exceptions may apply)
- Add missing Value dates and Posting dates to entities
- Having a generic naming conventions for JPA fields and DB fields
- Renaming the fields accordingly
- Evaluate value date (transaction date) and posting date (submitted on date), created on date usages
- Jobs to be checked and modified accordingly
- Native queries to be checked and modified accordingly
- Reports to be checked and modified accordingly
- Every table where update is supported the AbstractAuditableCustom should be implemented
- Amend Transactions and Journal entries date handling to fit for business date concept
- For audit fields we shall introduce timezoned datetimes and store them in database accordingly
 - Storing DATETIME fields without Timezone is potential problem due to the daylight savings
 - Also, some external libs (like Quartz) are using system timezone and Fineract will using Tenant timezone for audit fields. To be able to distinct them in DB we shall use DATETIME with TIMESTAMP column types and use timezoned java time objects in the application

Reliable event framework

Fineract is capable of generating and raising events for external consumers in a reliable way. This section is going to describe all the details on that front with examples.

Framework capabilities

ACID (transactional) guarantee

The event framework must support ACID guarantees on the business operation level.

Let's see a simple use-case:

1. A client applies to a loan on the UI
2. The loan is created on the server
3. A loan creation event is raised

What happens if step 3 fails? Shall it fail the original loan creation process?

What happens if step 2 fails but step 3 still gets executed? We're raising an event for a loan that hasn't been created in reality.

Therefore, raising an event is tied to the original business transaction to ensure the data that's getting written into the database along with the respective events are saved in an all-or-nothing fashion.

Messaging integration

The system is able to send the raised events to downstream message channels. The current implementation supports the following message channels:

- ActiveMQ

Ordering guarantee

The events that are raised will be sent to the downstream message channels in the same order as they were raised.

Delivery guarantee

The framework supports the at-least-once delivery guarantee for the raised events.

Reliability and fault-tolerance

In terms of reliability and fault-tolerance, the event framework is able to handle the cases when the downstream message channel is not able to accept events. As soon as the message channel is back to operational, the events will be sent again.

Selective event producing

Whether or not an event must be sent to downstream message channels for a particular Fineract instance is configurable through the UI and API.

Standardized format

All the events sent to downstream message channels are conforming a standardized format using Avro schemas.

Extendability and customizations

The event framework is capable of being easily extended with new events for additional business operations or customizing existing events.

Ability to send events in bulk

The event framework makes it possible to sort of queue events until they are ready to be sent and send them as a single message instead of sending each event as a separate, individual one.

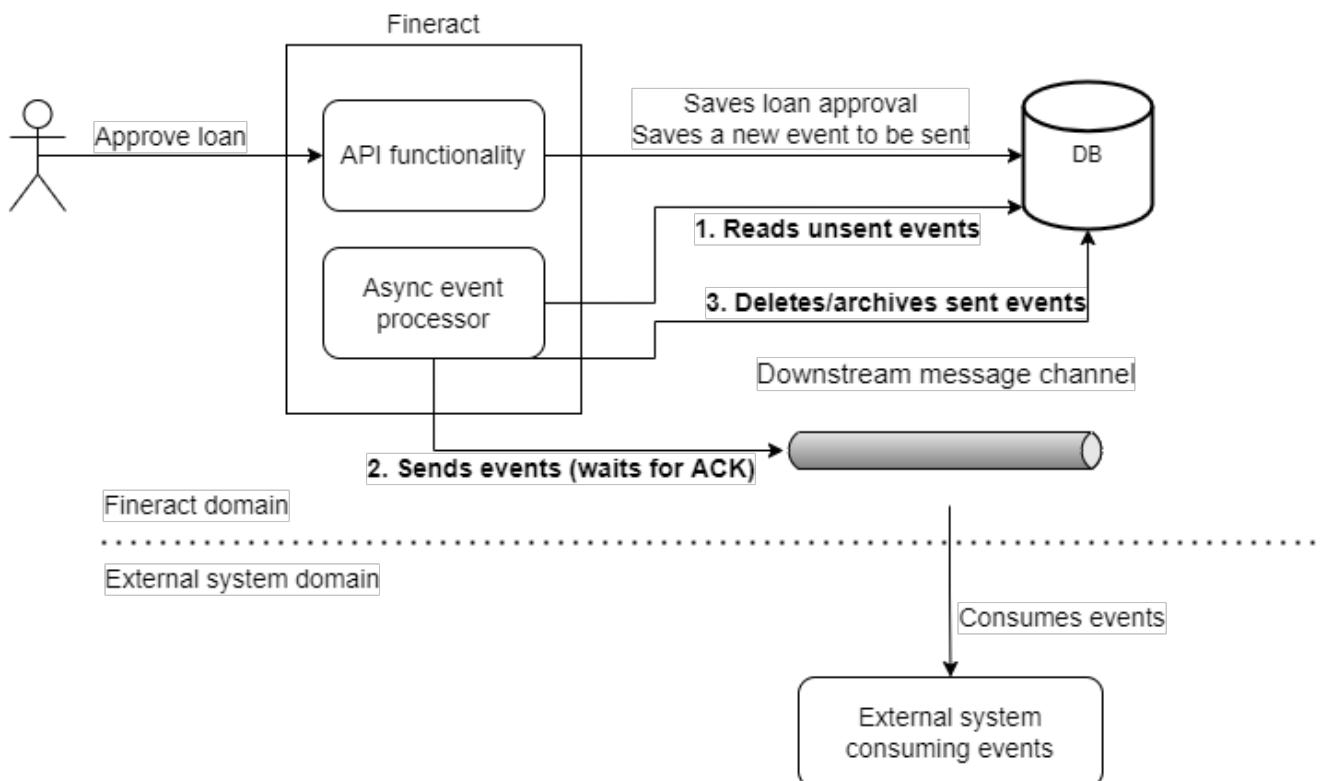
For example during the COB process, there might be events raised in separate business steps which needs to be sent out but they only need to be sent out at the end of the COB execution process instead of one-by-one.

Architecture

Intro

On a high-level, the concept looks the following. An event gets raised in a business operation. The event data gets saved to the database - to ensure ACID guarantees. An asynchronous process takes the saved events from the database and puts them onto a message channel.

The flow can be seen in the following diagram:



Foundational business events

The whole framework is built upon an existing infrastructure in Fineract; the Business Events.

As a quick recap, Business Events are Fineract events that can be raised at any place in a business operation using the `BusinessEventNotifierService`. Callbacks can be registered when a certain type of Business Event is raised and other business operations can be done. For example when a Loan gets disbursed, there's an interested party doing the Loan Arrears Aging recalculation using the Business Event communication.

The nice thing about the Business Events is that they are tied to the original transaction which means if any of the processing on the subscriber's side fail, the entire original transaction will be rolled back. This was one of the requirements for the Reliable event framework.

Event database integration

The database plays a crucial part in the framework since to ensure transactionality, - without doing proper transaction synchronization between different message channels and the database - the framework is going to save all the raised events into the same relational database that Fineract is using.

Database structure

The database structure looks the following

Name	Type	Description	Example
<code>id</code>	number	Auto incremented ID. Not null.	<code>1</code>
<code>type</code>	text	The event type as a string. Not null.	<code>LoanApprovedBusinessEvent</code>
<code>schema</code>	text	The fully qualified name of the schema that was used for the data serialization, as a string. Not null.	<code>org.apache.fineract.avro.loan.v1.LoanAccountDataV1</code>
<code>data</code>	BLOB (MySQL/MariaDB), BYTEA (PostgreSQL)	The event payload as Avro binary. Not null.	

<code>created_at</code>	timestamp	UTC timestamp when the event was raised. Not null.	<code>2022-09-06 14:20:10.148627 +00:00</code>
<code>status</code>	text	Enum text representing the status of the external event. Not null, indexed.	<code>TO_BE_SENT, SENT</code>
<code>sent_at</code>	timestamp	UTC timestamp when the event was sent.	<code>2022-09-06 14:30:10.148627 +00:00</code>
<code>idempotency_key</code>	text	Randomly generated UUID upon inserting a row into the table for idempotency purposes. Not null.	<code>68aed085-8235-4722-b27d-b38674c19445</code>
<code>business_date</code>	date	The business date to when the event was generated. Not null, indexed.	<code>2022-09-05</code>

The above database table contains the unsent events which later on will be sent by an asynchronous event processor.

Upon successfully sending an event, the corresponding statuses will be updated.

Avro schemas

For serializing events, Fineract is using Apache Avro. There are 2 reasons for that:

- More compact storage since Avro is a binary format
- The Avro schemas are published with Fineract as a separate JAR so event consumers can directly map the events into POJOs

There are 3 different levels of Avro schemas used in Fineract for the Reliable event framework which are described below.

Standard event schema

The standard event schema is for the regular events. These schemas are used when saving a raised event into the database and using the Avro schema to serialize the event data into a binary format.

For example the OfficeDataV1 Avro schema looks the following:

▼ `OfficeDataV1.avsc`

```

{
  "name": "OfficeDataV1",
  "namespace": "org.apache.fineract.avro.office.v1",
  "type": "record",
  "fields": [
    {
      "default": null,
      "name": "id",
      "type": [
        "null",
        "long"
      ]
    },
    {
      "default": null,
      "name": "name",
      "type": [
        "null",
        "string"
      ]
    },
    {
      "default": null,
      "name": "nameDecorated",
      "type": [
        "null",
        "string"
      ]
    },
    {
      "default": null,
      "name": "externalId",
      "type": [
        "null",
        "string"
      ]
    },
    {
      "default": null,
      "name": "openingDate",
      "type": [
        "null",
        "string"
      ]
    },
    {
      "default": null,
      "name": "hierarchy",
      "type": [
        "null",

```

```

        "string"
    ],
    },
    {
        "default": null,
        "name": "parentId",
        "type": [
            "null",
            "long"
        ]
    },
    {
        "default": null,
        "name": "parentName",
        "type": [
            "null",
            "string"
        ]
    },
    {
        "default": null,
        "name": "allowedParents",
        "type": [
            "null",
            {
                "type": "array",
                "items": "org.apache.fineract.avro.office.v1.OfficeDataV1"
            }
        ]
    }
]
}

```

Event message schema

The event message schema is just a wrapper around the standard event schema with extra metadata for the event consumers.

Since Avro is strongly typed, the event content needs to be first serialized into a byte sequence and that needs to be wrapped around.

This implies that for putting a single event message onto a message queue for external consumption, data needs to be serialized 2 times; this is the 2-level serialization.

1. Serializing the event
2. Serializing the already serialized event into an event message using the message wrapper

The message schema looks the following:

```

{
  "name": "MessageV1",
  "namespace": "org.apache.fineract.avro",
  "type": "record",
  "fields": [
    {
      "name": "id",
      "doc": "The ID of the message to be sent",
      "type": "long"
    },
    {
      "name": "source",
      "doc": "A unique identifier of the source service",
      "type": "string"
    },
    {
      "name": "type",
      "doc": "The type of event the payload refers to. For example
LoanApprovedBusinessEvent",
      "type": "string"
    },
    {
      "name": "category",
      "doc": "The category of event the payload refers to. For example LOAN",
      "type": "string"
    },
    {
      "name": "createdAt",
      "doc": "The UTC time of when the event has been raised; in
ISO_LOCAL_DATE_TIME format. For example 2011-12-03T10:15:30",
      "type": "string"
    },
    {
      "name": "businessDate",
      "doc": "The business date when the event has been raised; in
ISO_LOCAL_DATE format. For example 2011-12-03",
      "type": "string"
    },
    {
      "name": "tenantId",
      "doc": "The tenantId that the event has been sent from. For example
default",
      "type": "string"
    },
    {
      "name": "idempotencyKey",
      "doc": "The idempotency key for this particular event for consumer de-
duplication",
      "type": "string"
    }
  ]
}

```

```

    },
    {
      "name": "dataschema",
      "doc": "The fully qualified name of the schema of the event payload. For
example org.apache.fineract.avro.loan.v1.LoanAccountDataV1",
      "type": "string"
    },
    {
      "name": "data",
      "doc": "The payload data serialized into Avro bytes",
      "type": "bytes"
    }
  ]
}

```

Bulk event schema

The bulk event schema is used when multiple events are supposed to be sent together. This schema is used also when serializing the data for the database storing but the idea is quite simple. Have an array of other event schemas embedded into it.

Since Avro is strongly typed, the array within the bulk event schema is an array of `MessageV1` schemas. That way the consumers can decide which events they want to deserialize and which don't.

This elevates the regular 2-level serialization/deserialization concept up to a 3-level one:

1. Serializing the standard events
2. Serializing the standard events into a bulk event
3. Serializing the bulk event into an event message

Versioning

Avro is quite strict with changes to an existing schema and there are a number of compatibility modes available.

Fineract keeps it simple though. Version numbers - in the package names and in the schema names - are increased with each published modification; meaning that if the `OfficeDataV1` schema needs a new field and the `OfficeDataV1` schema has been published officially with Fineract, a new `OfficeDataV2` has to be created with the new field instead of modifying the existing schema.

This pattern ensures that a certain event is always deserialized with the appropriate schema definition, otherwise the deserialization could fail.

Code generation

The Avro schemas are described as JSON documents. That's hardly usable directly with Java hence Fineract generates Java POJOs from the Avro schemas. The good thing about these POJOs is the fact that they can be serialized/deserialized in themselves without any magic since they have a `toByteBuffer` and `fromByteBuffer` method.

From POJO to ByteBuffer:

```
LoanAccountDataV1 avroDto = ...  
ByteBuffer buffer = avroDto.toByteBuffer();
```

From ByteBuffer to POJO:

```
ByteBuffer buffer = ...  
LoanAccountDataV1 avroDto = LoanAccountDataV1.fromByteBuffer(buffer);
```



The ByteBuffer is a stateful container and needs to be handled carefully. Therefore Fineract has a built-in ByteBuffer to byte array converter; `ByteBufferConverter`.

Downstream event consumption

When consuming events on the other side of the message channel, it's critical to know which events the system is interested in. With the multi-level serialization, it's possible to deserialize only parts of the message and decide based on that whether it makes sense for a particular system to deserialize the event payload more.

Whether events are important can be decided based on:

- the `type` attribute in the message
- the `category` attribute in the message
- the `dataschema` attribute in the message

These are the main attributes in the message wrapper one can use to decide whether an event message is useful.

If the event needs to be deserialized, the next step is to find the corresponding schema definition. That's going to be sent in the `dataschema` attribute within the message wrapper. Since the attribute contains the fully-qualified name of the respective schema, it can be easily resolved to a Class object. Based on that class, the payload data can be easily deserialized using the `fromByteBuffer` method on every generated schema POJO.

Message ordering

One of the requirements for the framework is to provide ordering guarantees. All the events have to conform a happens-before relation.

For the downstream consumers, this can be verified by the `id` attribute within the messages. Since it's going to be a strictly-monotonic numeric sequence, it can be used for ordering purposes.

Event categorization

For easier consumption, the terminology event category is introduced. This is nothing else but the bounded context an event is related to.

For example the `LoanApprovedBusinessEvent` and the `LoanWaiveInterestBusinessEvent` are both related to the Loan bounded contexts.

The category in which an event resides in is included in the message under the `category` attribute.

The existing event categories can be found under the [Event categories](#) section.

Asynchronous event processor

The events stored in the database will be picked up and sent by a regularly executed job.

This job is a Fineract job, scheduled to run for every minute and will pick a number of events in order. Those events will be put onto the downstream message channel in the same order as they were raised.

Purging events

The events database table is going to grow continuously. That's why Fineract has a purging functionality in place that's gonna delete old and already sent events.

It's implemented as a Fineract job and is disabled by default. It's called TBD.

Usage

Using the event framework is quite simple. First, it has to be enabled through properties or environment variable.

The respective options are the following:

- the `fineract.events.external.enabled` property
- the `FINERACT_EXTERNAL_EVENTS_ENABLED` environment variable

These configurations accept a boolean value; `true` or `false`.

The key component to interact with is the `BusinessEventNotifierService#notifyPostBusinessEvent` method.

Raising events

Raising events is really easy. An instance of a `BusinessEvent` interface is needed, that's going to be the event. There are plenty of them available already in the Fineract codebase.

And that's pretty much it. Everything else is taken care of in terms of event data persisting and later on putting it onto a message channel.

An example of event raising:

```
@Override
public CommandProcessingResult createClient(final JsonCommand command) {
    ...
    businessEventNotifierService.notifyPostBusinessEvent(new
```



```
ClientCreateBusinessEvent(newClient));  
...  
return ...;  
}
```



The above code is copied from the `ClientWritePlatformServiceJpaRepositoryImpl` class.

Example event message content

Since the message is serialized into binary format, it's hard to represent in the documentation therefore here's a JSON representation of the data, just as an example.

```
{  
  "id": 121,  
  "source": "a65d759d-04f9-4ddf-ac52-34fa5d1f5a25",  
  "type": "LoanApprovedBusinessEvent",  
  "category": "Loan",  
  "createdAt": "2022-09-05T10:15:30",  
  "tenantId": "default",  
  "idempotencyKey": "abda146d-68b5-48ca-b527-16d2b7c5daef",  
  "dataschema": "org.apache.fineract.avro.loan.v1.LoanAccountDataV1",  
  "data": "..."  
}
```



The source attribute refers to an ID that's identifying the producer service. Fineract will regenerate this ID upon each application startup.

Raising bulk events

Raising bulk events is really easy as well. The 2 key methods are:

- `BusinessEventNotifierService#startExternalEventRecording`
- `BusinessEventNotifierService#stopExternalEventRecording`

First, you have to start recording your events. This recording will be applied for the current thread. And then you can raise as many events as you want with the regular `BusinessEventNotifierService#notifyPostBusinessEvent` method, but they won't get saved to the database immediately. They'll get "recorded" into an internal buffer.

When you stop recording using the method above, all the recorded events will be saved as a bulk event to the database; and serialized appropriately.

From then on, the bulk event works just like any of the event. It'll be picked up by the processor to send it to a message channel.

Event categories

TBD

Selective event producing

TBD

Customizations

The framework provides a number of customization options:

- Creating new events (that's already given by the Business Events)
- Creating new Avro schemas
- Customizing what data gets serialized for existing events

In the upcoming sections, that's what going to be discussed.

Creating new events

Creating new events is super easy. Just create an implementation of the `BusinessEvent` interface and that's it.

From then on, you can raise those events in the system, although you can't publish them to an external message channel. If you have the event framework enabled, it's going to fail with not finding the appropriate serializer for your business event.



There are existing serializers which might be able to handle your new event. For example the `LoanBusinessEventSerializer` is capable of handling all `LoanBusinessEvent` subclasses so there's no need to create a brand new serializer.

The interface looks the following:

`BusinessEvent.java`

```
public interface BusinessEvent<T> {  
  
    T get();  
  
    String getType();  
  
    String getCategory();  
  
    Long getAggregateRootId();  
}
```

Quite simple. The `get` method should return the data you want to pass within the event instance. The `getType` method returns the name of the business event that's gonna be saved as the `type` into the database.



Creating a new business event only means that it can be used for raising an event. To make it compatible with the event framework and to be sent to a message channel, some extra work is needed which are described below.

Creating new Avro schemas and serializers

First let's talk about the event serializers because that's what's needed to make a new event compatible with the framework.

The serializer has a special interface, `BusinessEventSerializer`.

`BusinessEventSerializer.java`

```
public interface BusinessEventSerializer {  
  
    <T> boolean canSerialize(BusinessEvent<T> event);  
  
    Class<? extends GenericContainer> getSupportedSchema();  
  
    <T> ByteBufferSerializable toAvroDTO(BusinessEvent<T> rawEvent);  
  
}
```

An implementation of this interface shall be registered as a Spring bean, and it'll be picked up automatically by the framework.



You can look at the existing serializers for implementation ideas.

New Avro schemas can be easily created. Just create a new Avro schema file in the `fineract-avro-schemas` project under the respective bounded context folder, and it will be picked up automatically by the code generator.

BigDecimal support in Avro schemas

Apache Avro by default doesn't support complex types like a `BigDecimal`. It has to be implemented using a custom snippet like this:

```
{  
    "logicalType": "decimal",  
    "precision": 27,  
    "scale": 8,  
    "type": "bytes"  
}
```

It's a 20 precision and 8 scale `BigDecimal`.

Obviously it's quite challenging to copy-paste this snippet to every single `BigDecimal` field, so there's a customization in place for Fineract.

The type `BigDecimal` is supported natively, and you're free to use it like this:

```
{
  "default": null,
  "name": "principal",
  "type": [
    "null",
    "BigDecimal"
  ]
}
```



This `BigDecimal` type will be simple replaced with the `BigDecimal` snippet showed above during the compilation process.

Custom data serialization for existing events

In case there's a need some extra bit of information within the event message that the default serializers are not providing, you can override this behavior by registering a brand-new custom serializer (as shown above).

Since there's a priority order of serializers, the only thing the custom serializer need to do is to be annotated by the `@Order` annotation or to implement the `Ordered` interface.

An example custom serializer with priority looks the following:

```
@Component
@RequiredArgsConstructor
@Order(Ordered.HIGHEST_PRECEDENCE)
public class CustomLoanBusinessEventSerializer implements BusinessEventSerializer {
    ...

    @Override
    public <T> boolean canSerialize(BusinessEvent<T> event) {
        return ...;
    }

    @Override
    public <T> byte[] serialize(BusinessEvent<T> rawEvent) throws IOException {
        ...
        ByteBuffer buffer = avroDto.toByteBuffer();
        return byteBufferConverter.convert(buffer);
    }

    @Override
    public Class<? extends GenericContainer> getSupportedSchema() {
        return ...;
    }
}
```



All the default serializers are having `Ordered.LOWEST_PRECEDENCE`.

Appendix A: Properties and environment variables

Property name	Environment variable	Default value	Description
<code>fineract.events.external.enabled</code>	<code>FINERACT_EXTERNAL_EVENTS_ENABLED</code>	<code>false</code>	Whether the external event sending is enabled or disabled.

Introducing Advanced payment allocation

Since the first repayment strategy got introduced, many followed, but there was one thing common in them:

- They were hard coding the allocation rules for each transaction type.

By introducing the "Advanced payment allocation" the idea was to have a repayment strategy which was:

- supporting dynamic configuration of the allocation rules for transaction types
- supporting configuration of more fine-grained allocation rules for future installments
- supporting reprocessing of transactions and charges in chronological order

Glossary

*Advanced payment allocation	Ability to configure allocation rules dynamically for transactions
*Payment allocation	Rule that defines which outstanding balance to be paid of first on which installment
*Re-amortization	Transaction amount to be divided into equal portions by the number of future installments and those installments to be paid by these portions.

Capabilities

- Payment allocation should be configurable for transactions:
 - Repayment
 - Goodwill credit
 - Payout refund
 - Merchant refund
 - Charge adjustments
 - etc.

- Can be configured for Loan products
 - Payment allocation rule changes on the loan product will affect only the newly created Loan accounts.
- Chronological reprocess order
 - Transactions (including disbursements) and charges are (re)processed and allocated in chronological order
- Support re-amortization between future installments
 - Transaction amount to be divided into equal portions (based on the number of future installments) and to repay each future installment by the calculated portion.
 - It's not hard coded, but usually the principal portion needs to be allocated first, but if there are still unprocessed amounts, the rest of the outstanding balances are to be allocated based on the rest of the rules
- Main allocation rules (installment level)
 - Past Due Installment(s):
 - Oldest first
 - Due Installment(s):
 - Normal installment takes priority over Down-payment installment (if applicable)
 - Future Installment(s):
 - Available allocation orders:
 - Next installment first
 - Last installment first
 - Re-amortization*
- Secondary allocation rules
 - Penalty
 - Fee
 - Interest
 - Principal

Configuration

Advanced repayment allocation rules can be configured for the Loan product if "Advanced payment allocation" got selected as repayment strategy.

There will be a (always required) "DEFAULT" transaction type configuration which acts as fallback ruleset, if there are no configured rules for a specific transaction type.

New repayment strategy

- Name: Advanced payment allocation
- Code: advanced-payment-allocation-strategy

- Order: 8

Allocation rules

- Past due penalty
- Past due fee
- Past due principal
- Past due interest
- Due penalty
- Due fee
- Due principal
- Due interest
- In advance penalty
- In advance fee
- In advance principal
- In advance interest

Future installment allocation rules:

- Next installment
- Last installment
- Re-amortization

Example Request

```
{
  ...
  "paymentAllocation": [
    {
      "transactionType": "DEFAULT",
      "paymentAllocationOrder": [
        {
          "paymentAllocationRule": "DUE_PAST_PENALTY",
          "order": 1
        },
        {
          "paymentAllocationRule": "DUE_PAST_FEE",
          "order": 2
        },
        {
          "paymentAllocationRule": "DUE_PAST_INTEREST",
          "order": 3
        },
        ...
      ]
    }
  ]
}
```

```

        {
            "paymentAllocationRule": "IN_ADVANCE_INTEREST",
            "order": 14
        },
        {
            "futureInstallmentAllocationRule": "NEXT_INSTALLMENT"
        }
    ],
    ...
}

```

The above request configures the "DEFAULT" allocation rules:

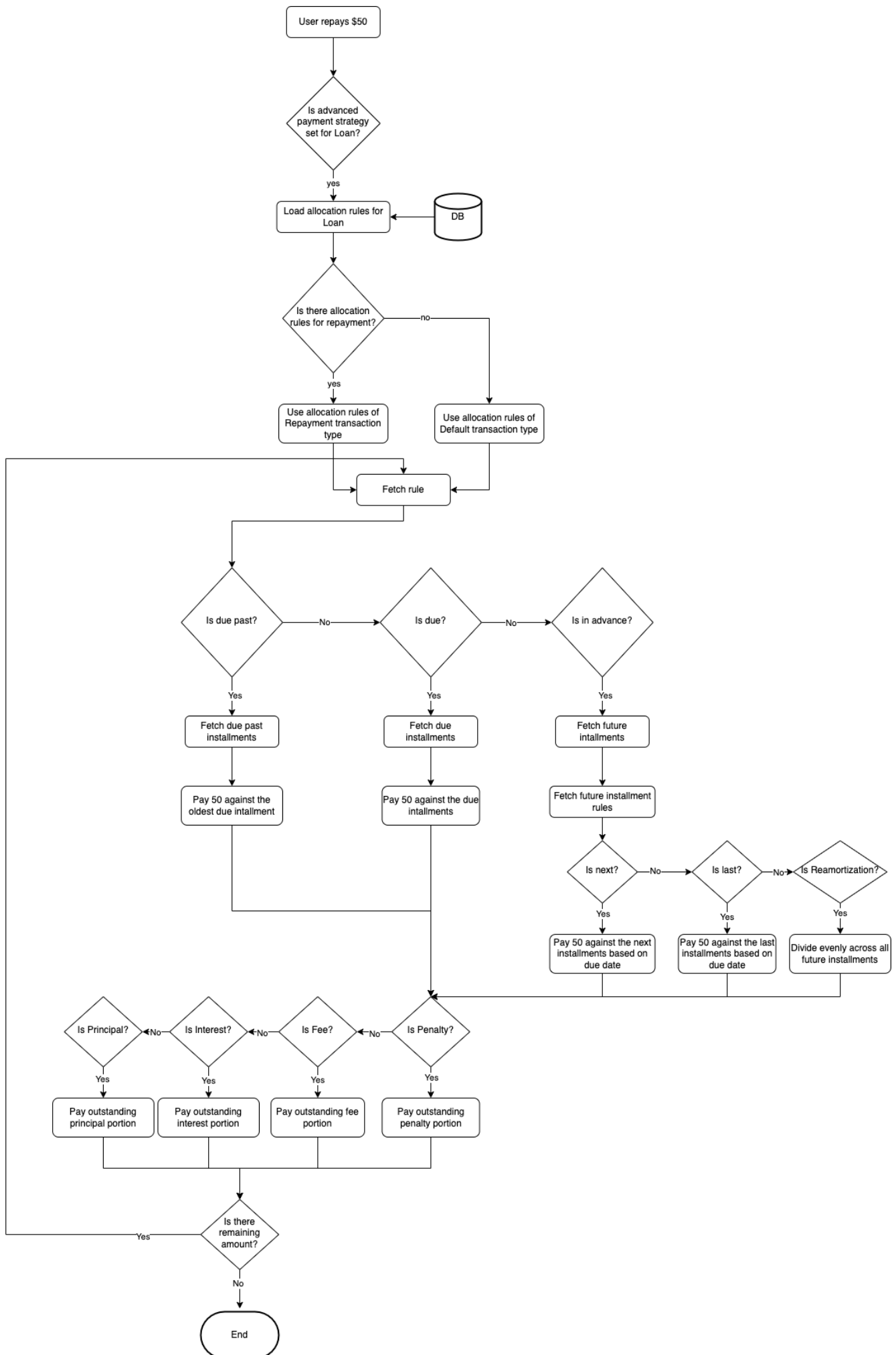
- First the already due penalties to be paid
- Second the already due fees to be paid
- Last the future interests to be paid

Also for future installments set the allocation rules as

- First future installment by due date to be paid first

High level design

Flow of advanced payment allocation processing



Features

This section covers specific features and functionality available in Apache Fineract.

Capitalized Income

Overview

Capitalized Income (formerly referred to as Origination Fee) in Apache Fineract is a fee-based or interest-based income item that is added to the principal of a loan and amortized over time. It is designed to be applied at disbursement and is treated as part of the loan's principal for repayment and interest calculations.

Purpose

This functionality enables financial institutions to:

- Recognize deferred income (fees or interest) systematically over the loan term
- Align accounting practices with regulatory requirements
- Improve income recognition accuracy

Supported Loan Type



Capitalized Income is only supported for:

- Progressive Loan Schedules
- Advanced Payment Allocation Strategy

Other loan schedule types and transaction processing strategies are not supported.

Configuration at Loan Product Level

Capitalized income must be configured on the loan product.

The configuration options include:

- **Enable Capitalized Income:** Boolean toggle (default: disabled)
- **Calculation Mode:** Only "Flat" is currently supported
 - Later "Percentage based" can be introduced
- **Amortization Strategy:** Only "EQUAL_AMORTIZATION" is supported
 - Daily equal portions are recognized over the life of the loan
 - Later other strategies can be introduced
- **Income Type:** Specifies allocation rule. Defines which "balance category" to be used.



In Fineract, balance of a transaction is either: Principal, Fee, Penalty, Interest or overpayment

Options:

- * FEE (default)
- * INTEREST

GL Mapping

Required GL Account mappings when Capitalized Income is enabled:

- **Deferred Income (Liability):** `deferredIncomeLiabilityAccountId` - mandatory when enabled
- **Income from Capitalization (Income):** `incomeFromCapitalizationAccountId` - mandatory when enabled



Both GL accounts become mandatory when `enableIncomeCapitalization` is set to `true`.

Configuration Dependencies

When `enableIncomeCapitalization` is set to `true`, all following parameters become mandatory:

- `capitalizedIncomeCalculationType` - must be "FLAT"
- `capitalizedIncomeStrategy` - must be "EQUAL_AMORTIZATION"
- `capitalizedIncomeType` - must be "FEE" or "INTEREST"
- `deferredIncomeLiabilityAccountId` - must reference a valid GL account
- `incomeFromCapitalizationAccountId` - must reference a valid GL account

Behavior and Calculations

- Capitalized income is added via API on or after the first disbursement date
- It is treated as a principal portion, recalculating the repayment schedule accordingly
- Interest and amortization schedules are updated to include the capitalized income amount
- Validated using formula: $(\text{Total Disbursed} + \text{Current Capitalized Income} + \text{New Transaction Amount}) \leq \text{Max Amount}$, where Max Amount depends on loan product configuration: if `allowApprovedDisbursedAmountsOverApplied = true` uses `getOverAppliedMax(loan)`, otherwise uses `getApprovedPrincipal()`

Daily Amortization

- Recognized daily using the configured strategy
- Recognized portions move from Deferred Income to Income from Capitalization

Special Handling

- **Preclosure:** Remaining balance recognized in full on the preclosure date

- **Charge-off:** Amortization stops and remaining balance is charged off

Transaction Types Introduced

- Capitalized Income
- Capitalized Income Amortization
- Capitalized Income Adjustment
- Capitalized Income Amortization Adjustment

Capitalized Income Transaction

The Capitalized Income transaction in Apache Fineract performs the following actions:

- Adds a specified amount to the loan principal
 - Considered a deferred income item (such as a fee or interest)
 - Booked as part of the loan's principal
 - Added post-disbursement and only if the loan type supports it (currently: Progressive Loans)
- Creates a distinct loan transaction
 - Separately tracked with its own transaction type ("Capitalized Income")
 - Not merged with disbursements or repayments
- Updates the loan schedule
 - Recalculates amortization and interest schedule to include the added amount in the outstanding principal
- Triggers accounting entries
 - Debits "Loan Portfolio" (Asset)
 - Credits "Deferred Income" (Liability)
 - Does not recognize income upfront
- Initiates daily amortization
 - Source for daily income recognition through "Capitalized Income Amortization" transactions
 - Progressively converts the deferred amount to recognized income

Accounting Entries

Scenario	Debit
Credit	Capitalized Income
Loan Portfolio (Asset)	Deferred Income (Liability)

Capitalized Income Amortization

A Capitalized Income Amortization transaction in Apache Fineract does the following:

- **Recognizes Deferred Income Over Time:** Transfers a portion of the capitalized income (originally posted as a liability) into recognized income (posted as interest or fee income), based on a configured daily amortization strategy.
- **Daily Posting:** The system automatically creates this transaction each day from the date of capitalized income until the loan maturity or until the full amount is amortized. This is handled by a background job during the COB (Close of Business) process.
- **Uses Equal Amortization:** The default and only supported strategy is Equal Amortization, which divides the total capitalized income evenly over the remaining number of days until the loan matures.

Accounting Entries

Scenario	Debit
Credit	Daily amortization
Deferred Income (Liability)	Income from Capitalization (Income)

Stops on Events

- **Preclosure:** Triggers final amortization for remaining unrecognized income
- **Charge-off:** Halts further amortization; the remaining deferred income is charged off



Reversal Handling: If the original Capitalized Income transaction is reversed, all associated amortization transactions are also reversed via "Capitalized Income Amortization Adjustment" transactions.

Capitalized Income Adjustment

A Capitalized Income Adjustment transaction in Apache Fineract serves to reduce the balance of an existing capitalized income transaction.

Purpose

- Correct overcharged or misposted capitalized income amounts
- Reflect fee waivers or negotiated reductions
- Support backdated corrections if needed

Transaction Behavior

- It is a credit-type transaction, reducing the capitalized income balance
- Treated similarly to other credit transactions and follows a defined allocation strategy
- Can be backdated, but not dated before the original capitalized income transaction

Validation Rules

- The adjustment amount must not exceed the remaining amount (original capitalized income amount minus total previous adjustments)

- Adjustment is linked to a specific Capitalized Income transaction (by ID)
- Multiple adjustments can be made against the same original transaction
- Adjustments can be reversed if needed

Accounting Entries

Scenario	Debit	Credit
Adjustment \leq unrecognized balance	Deferred Income (Liability)	Loan Portfolio (Asset)
Adjustment $>$ unrecognized balance	Deferred Income (Liability)	Loan Portfolio (Asset)

Business Event Triggers

- Triggers "Capitalized Income Adjustment" event
- Updates loan balance and possibly loan status depending on impact

Impact

- Reduces amortization basis
- May modify future amortization amounts
- Repayment schedule is not affected directly unless recalculated manually

Capitalized Income Amortization Adjustment

A Capitalized Income Amortization Adjustment in Apache Fineract is a special transaction type used to reverse previously recognized income from capitalized income amortization.

Purpose

- Automatically generated when a Capitalized Income transaction is reversed or when backdated Capitalized Income Adjustment affects amortization balances
- Reverses all already recognized portions (amortized income) linked to the original Capitalized Income transaction

When It Occurs

- Created by daily amortization (COB) or final amortization (triggered on loan closure, charge-off, or by any backdated transaction that affects capitalized income balances)
- Reverses previously recognized income when amortization needs to be adjusted
- Restores Deferred Income balances and reverses income recognition

Accounting Entries

Transaction Type	Debit	Credit
Capitalized Income Amortization Adjustment	Income from Capitalization (Income)	Deferred Income (Liability)

Key Characteristics

- **System-Generated Only:** Cannot be created manually by API or UI
- **Ensures Accounting Integrity:** Keeps amortized and unrecognized balances aligned after reversals
- **Non-monetary transaction - does not trigger balance changed or status update events**

Business Events

- Triggers a new business event: Capitalized Income Amortization Adjustment

API Endpoints

Configure Capitalized Income on Loan Product

- **Endpoint:** `/loanproducts`
- **Method:** `POST`

```
{
  ...
  "enableIncomeCapitalization": true, // Mandatory
  "capitalizedIncomeCalculationType": "FLAT", // Mandatory when enabled
  "capitalizedIncomeStrategy": "EQUAL_AMORTIZATION", // Mandatory when enabled
  "capitalizedIncomeType": "FEE", // Mandatory when enabled
  "deferredIncomeLiabilityAccountId": 123, // Mandatory when enabled
  "incomeFromCapitalizationAccountId": 456 // Mandatory when enabled
}
```

Add Capitalized Income

- **Endpoint:** `/loans/{loanId}/transactions?command=capitalizedIncome`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions?command=capitalizedIncome`
- **Method:** `POST`

```
{
  "transactionDate": "2025-05-01", // Mandatory
  "dateFormat": "yyyy-MM-dd", // Mandatory
  "locale": "en", // Mandatory
  "transactionAmount": 100.0, // Mandatory
  "paymentTypeId": 1, // Optional
  "note": "Capitalized income fee", // Optional
}
```

```

    "externalId": "CINCOME-001" // Optional
  }

```

Get Capitalized Income Amortization Info

- **Endpoint:** `/loans/{loanId}/deferredincome`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/deferredincome`
- **Method:** GET

Response Body

```

{
  "capitalizedIncomeData": [
    {
      "amount": 50.0, // Total capitalized income amount
      "amortizedAmount": 1.1, // Amount already amortized
      "unrecognizedAmount": 48.9, // Amount not yet amortized
      "amountAdjustment": 0.0, // Any adjustments made
      "chargedOffAmount": 0.0 // Amount charged off (if applicable)
    }
  ]
}

```

Add Capitalized Income Adjustment

- **Endpoint:**
`/loans/{loanId}/transactions/{capitalizedIncomeTransactionId}?command=capitalizedIncomeAdjustment`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions/{capitalizedIncomeTransactionId}?command=capitalizedIncomeAdjustment`
- **Method:** POST

```

{
  "transactionDate": "2025-05-01", // Mandatory
  "dateFormat": "yyyy-MM-dd", // Mandatory
  "locale": "en", // Mandatory
  "transactionAmount": 50.0, // Mandatory
  "paymentTypeId": 1, // Optional
  "note": "Capitalized income fee", // Optional
  "externalId": "CINCOMEADJ-001" // Optional
}

```

Response Body


```
{
  "resourceId": 1,
  "resourceExternalId": "CINCOMEADJ-001"
}
```

Capitalized Income Template API (to retrieve limits)

- **Endpoint:** `/loans/{loanId}/transactions/template?command=capitalizedIncome`
- **Alternative** **Endpoint:** `/loans/external-id/{loanExternalId}/transactions/template?command=capitalizedIncome`
- **Method:** GET

```
{
  "paymentTypeOptions": [], // List of available payment types
  "currency": {...},        // Currency configuration
  "date": [2025, 5, 29],    // Return the current date
  "amount": 0               // Return the maximum amount that can be capitalized
                             (approved amount - disbursed amount - capitalized income)
}
```

Capitalized Income Adjustment Template API (to retrieve limits)

- **Endpoint:** `/loans/{loanId}/transactions/template?command=capitalizedIncomeAdjustment&transactionId={capitalizedIncomeTransactionId}`
- **Alternative** **Endpoint:** `/loans/external-id/{loanExternalId}/transactions/template?command=capitalizedIncomeAdjustment&transactionId={capitalizedIncomeTransactionId}`
- **Method:** GET

```
{
  "paymentTypeOptions": [], // List of available payment types
  "currency": {...},        // Currency configuration
  "date": [2025, 5, 29],    // Return the current date
  "amount": 0               // Return the maximum amount that can be adjusted
                             (capitalized income - adjustment)
}
```

Accounting Entries

Transaction Type	Debit	Credit
Capitalized Income	Loan Portfolio (Asset)	Deferred Income (Liability)

Transaction Type	Debit	Credit
Capitalized Income Amortization	Deferred Income (Liability)	Income from Capitalization (Income)
Capitalized Income Adjustment	Deferred Income (Liability)	Loan Portfolio (Asset)
Capitalized Income Amortization Adjustment	Income from Capitalization (Income)	Deferred Income (Liability)

Business Events

Triggered for Capitalized Income

- `LoanCapitalizedIncomeTransactionCreatedBusinessEvent`
- `LoanBalanceChangedBusinessEvent`

Daily Amortization

- `LoanCapitalizedIncomeAmortizationTransactionCreatedBusinessEvent`
- `LoanCapitalizedIncomeAmortizationAdjustmentTransactionCreatedBusinessEvent`

Capitalized Income Adjustment

- `LoanCapitalizedIncomeAdjustmentTransactionCreatedBusinessEvent`
- `LoanBalanceChangedBusinessEvent`

Reversal

- `LoanAdjustTransactionBusinessEvent`

Database Structure

Configuration

Stored on Loan Product (`m_product_loan`)

Field	Data Type	Description
<code>enable_income_capitalization</code>	BOOLEAN	Enable capitalized income feature (default: <code>false</code>)
<code>capitalized_income_calculation_type</code>	VARCHAR	Calculation method (ENUM: <code>FLAT</code>)
<code>capitalized_income_strategy</code>	VARCHAR	Amortization strategy (ENUM: <code>EQUAL_AMORTIZATION</code>)
<code>capitalized_income_type</code>	VARCHAR	Income type (ENUM: <code>FEE</code> , <code>INTEREST</code>)

Stored on Loan (m_loan)

Field	Data Type	Description
enable_income_capitalization	BOOLEAN	Enable capitalized income feature (default: false)
capitalized_income_calculation_type	VARCHAR	Calculation method (ENUM: FLAT)
capitalized_income_strategy	VARCHAR	Amortization strategy (ENUM: EQUAL_AMORTIZATION)
capitalized_income_type	VARCHAR	Income type (ENUM: FEE , INTEREST)

Balances**On Loan**

Field	Data Type	Description
capitalized_income_derived	DECIMAL(19,6)	Total capitalized income amount (nullable)
capitalized_income_adjustment_derived	DECIMAL(19,6)	Total adjustment amount (nullable)

Capitalized Income Balance (m_loan_capitalized_income_balance)

Each capitalized income has its own balance (1 row for each transaction)

Field	Data Type	Description
id	BIGINT	Unique identifier (Primary Key)
version	BIGINT	Version for optimistic locking
loan_id	BIGINT	Associated loan ID (Foreign Key, NOT NULL)
loan_transaction_id	BIGINT	Associated loan transaction ID (Foreign Key, NOT NULL)
amount	DECIMAL(19,6)	Capitalized income transaction amount (NOT NULL)
date	DATE	Capitalized income transaction date (NOT NULL)
unrecognized_amount	DECIMAL(19,6)	Amortization - not yet recognized amount (NOT NULL)
charged_off_amount	DECIMAL(19,6)	Charged-off balance (nullable)
amount_adjustment	DECIMAL(19,6)	Total adjustment amount (nullable)

Field	Data Type	Description
created_by	BIGINT	Audit field - user who created the record
created_on_utc	DATETIME	Creation timestamp (UTC)
last_modified_by	BIGINT	Last modifier user ID
last_modified_on_utc	DATETIME	Last modification timestamp (UTC)

Constraints

• Foreign Key Constraints:

- loan_id references m_loan(id)
- loan_transaction_id references m_loan_transaction(id)
- created_by references m_appuser(id)
- last_modified_by references m_appuser(id)

Notes



- Capitalized income transactions support backdating
- Adjustment transactions must not predate the original capitalized income
- No automatic reversal is supported; must be handled manually via dedicated transactions
- Proper GL accounts must be set for Deferred Income and Income from Capitalization to enable this functionality

Buy Down Fee

Overview

Buy Down Fee is a specialized fee mechanism in Apache Fineract that allows financial institutions to collect upfront fees from borrowers to reduce their effective interest rate over the loan term. This feature is particularly designed for 0% interest "buy down" loans where a merchant fee is collected and amortized into interest/fee income over the life of the loan.

The key characteristic of Buy Down Fee is that the amortized fee is **NOT visible to the customer** and **NOT affecting the repayment schedule** - it operates as a background process for proper revenue recognition while maintaining transparency in customer-facing loan terms.

Purpose

This functionality enables financial institutions to:

- **Interest Rate Reduction:** Borrowers can reduce their effective interest rate by paying an

upfront fee

- **Merchant Fee Support:** Enables 0% interest loan products with merchant-paid fees
- **Revenue Recognition:** Provides controlled amortization of fee income over the loan term
- **Customer Transparency:** Fee amortization is invisible to customers, maintaining clean loan presentation
- **Accounting Integration:** Proper journal entries and accounting treatment for fee transactions

Supported Loan Type



Buy Down Fee is only supported for loans that have **all** of the following:

- **Advanced Payment Allocation Strategy** (transaction processing strategy)
- **Progressive Loan Schedule** (loan schedule type)

Other transaction processing strategies or loan schedule types are not supported.

Configuration at Loan Product Level

Buy Down Fee must be configured on the loan product.

The configuration options include:

- **Enable Buy Down Fee:** Boolean toggle (`enableBuyDownFee`) (default: disabled)
- **Calculation Mode:** Only "Flat" is currently supported (`buyDownFeeCalculationType`)
- **Amortization Strategy:** Only "EQUAL_AMORTIZATION" is supported (`buyDownFeeStrategy`)
 - Daily equal portions are recognized over the life of the loan
- **Income Type:** Specifies allocation rule (`buyDownFeeIncomeType`)

Options:

* FEE

* INTEREST

GL Mapping

Required GL Account mappings when Buy Down Fee is enabled:

- **Buy Down Expense Account:** `buyDownExpenseAccountId` - mandatory when enabled
- **Deferred Income Liability Account:** `deferredIncomeLiabilityAccountId` - mandatory when enabled
- **Income from Buy Down Account:** `incomeFromBuyDownAccountId` - mandatory when enabled



All GL accounts become mandatory when `enableBuyDownFee` is set to `true`.

Configuration Dependencies

When `enableBuyDownFee` is set to `true`, all following parameters become mandatory:

- `buyDownFeeCalculationType` - must be "FLAT"
- `buyDownFeeStrategy` - must be "EQUAL_AMORTIZATION"
- `buyDownFeeIncomeType` - must be "FEE" or "INTEREST"
- `buyDownExpenseAccountId` - must reference a valid GL account
- `deferredIncomeLiabilityAccountId` - must reference a valid GL account
- `incomeFromBuyDownAccountId` - must reference a valid GL account

Validation Rules

Product Level Validations

- Buy Down Fee can only be enabled for Progressive Loan products
- When `enableBuyDownFee` is `true`, all related parameters become mandatory
- Calculation type must be `FLAT` (other types not yet supported)
- Strategy must be `EQUAL_AMORTIZATION` (other strategies not yet supported)
- Income type must be either `FEE` or `INTEREST`
- Both expense and income GL accounts must be provided and valid
- GL accounts must have correct account types (EXPENSE and INCOME respectively)
- `deferredIncomeLiabilityAccountId` mapping requirements:
 - Must be a valid LIABILITY type GL account
 - Represents temporary holding of not-yet-recognized income
 - Used to track unamortized Buy Down Fee portion
 - Cannot be zero or null when Buy Down Fee is enabled

Transaction Level Validations

- Buy Down Fee transactions can only be added to active loans
- Transaction amount must be positive (greater than zero)
- Transaction date cannot be before the first disbursement date
- Loan must have Buy Down Fee enabled in its product configuration
- Transaction date cannot be in the future
- Client/Group must be active
- Loan must be disbursed
- Multiple Buy Down Fee transactions per loan are supported

Adjustment Validations

- Original Buy Down Fee transaction must exist
- Adjustment amount cannot exceed remaining balance (amount - previous adjustments)
- Adjustment date cannot be before original transaction date
- Cannot reverse Buy Down Fee transaction if it has linked adjustments

Buy down fee adjustments are related to the buy down fee transaction (they have relation with type ADJUSTMENT between them), and there can be more than one adjustment to the same buy down fee transaction.

Error Responses

Common Error Codes

- `buy.down.fee.not.enabled`: Buy Down Fee not enabled for loan product
- `cannot.be.before.first.disbursement.date`: Invalid transaction date
- `cannot.be.more.than.remaining.amount`: Adjustment exceeds balance
- `loan.transaction.not.found`: Referenced transaction not found

Error Response Example

```
{
  "developerMessage": "Buy down fee is not enabled for this loan product",
  "httpStatusCode": "400",
  "defaultUserMessage": "Buy down fee is not enabled for this loan product",
  "userMessageGlobalisationCode": "buy.down.fee.not.enabled",
  "errors": [
    {
      "developerMessage": "Buy down fee is not enabled for this loan product",
      "defaultUserMessage": "Buy down fee is not enabled for this loan product",
      "userMessageGlobalisationCode": "buy.down.fee.not.enabled",
      "parameterName": null
    }
  ]
}
```

Configuration Error Messages

- **"Buy Down Fee calculation type is required"**: Provide `buyDownFeeCalculationType` when enabling Buy Down Fee
- **"Buy Down Fee strategy is required"**: Provide `buyDownFeeStrategy` when enabling Buy Down Fee
- **"Buy Down Fee income type is required"**: Provide `buyDownFeeIncomeType` when enabling Buy Down Fee

- **"Buy Down expense account is required"**: Provide valid `buyDownExpenseAccountId`
- **"Deferred income liability account is required"**: Provide valid `deferredIncomeLiabilityAccountId`
- **"Income from Buy Down account is required"**: Provide valid `incomeFromBuyDownAccountId`
- **"Buy Down fees can only be added to active loans"**: Ensure loan status is ACTIVE before adding Buy Down Fee transactions

Behavior and Calculations

- Buy Down Fee transactions can only be added to active loans
- Transaction amount must be positive (greater than zero)
- Transaction date cannot be before the first disbursement date
- Loan must have Buy Down Fee enabled in its product configuration

Daily Amortization

- Recognized daily using the configured strategy
- Recognized portions move from Deferred Income to Income from Buy Down

Special Handling

- **Preclosure**: Remaining balance recognized in full on the preclosure date
- **Charge-off**: Amortization stops and remaining balance is charged off

Transaction Types Introduced

- Buy Down Fee
- Buy Down Fee Amortization
- Buy Down Fee Adjustment
- Buy Down Fee Amortization Adjustment

Buy Down Fee Transaction

The Buy Down Fee transaction in Apache Fineract performs the following actions:

- Creates a distinct loan transaction
 - Separately tracked with its own transaction type ("Buy Down Fee")
 - Not merged with disbursements or repayments
- Triggers accounting entries
 - Debits "Buy Down Expense Account" (Expense)
 - Credits "Deferred Income Liability Account" (Liability)
 - Does not recognize income upfront

- Initiates daily amortization
 - Source for daily income recognition through "Buy Down Fee Amortization" transactions
 - Progressively converts the deferred amount to recognized income

Accounting Entries

Scenario	Debit
Credit	Buy Down Fee
Buy Down Expense Account	Deferred Income Liability Account

Buy Down Fee Amortization

A Buy Down Fee Amortization transaction in Apache Fineract does the following:

- **Recognizes Deferred Income Over Time:** Transfers a portion of the buy down fee (originally posted as a liability) into recognized income, based on a configured daily amortization strategy.
- **Daily Posting:** The system automatically creates this transaction each day from the date of buy down fee until the loan maturity or until the full amount is amortized. This is handled by a background job during the COB (Close of Business) process.
- **Uses Equal Amortization:** The default and only supported strategy is Equal Amortization, which divides the total buy down fee evenly over the remaining number of days until the loan matures.

Accounting Entries

Scenario	Debit
Credit	Daily amortization
Deferred Income Liability Account	Income from Buy Down Account

Stops on Events

- **Preclosure:** Triggers final amortization for remaining unrecognized income
- **Charge-off:** Halts further amortization; the remaining deferred income is charged off using Charge-off Expense Account

Buy Down Fee Adjustment

A Buy Down Fee Adjustment transaction in Apache Fineract serves to reduce the balance of an existing buy down fee transaction.

Purpose

- Correct overcharged or misposted buy down fee amounts
- Reflect fee waivers or negotiated reductions
- Support backdated corrections if needed

Transaction Behavior

- It is a credit-type transaction, reducing the buy down fee balance
- Can be backdated, but not dated before the original buy down fee transaction

Validation Rules

- The adjustment amount cannot exceed remaining balance (amount - previous adjustments)
- Adjustment date cannot be before original transaction date
- Adjustment date cannot be before disbursement date
- Adjustment date cannot be in the future
- Cannot reverse Buy Down Fee transaction if it has linked adjustments
- Loan must be in Active, Closed, or Overpaid status
- Buy Down Fee must be enabled on the loan product
- Loan must use Progressive Schedule

Accounting Entries

Scenario	Debit	Credit
Buy Down Fee Adjustment	Deferred Income Liability Account	Buy Down Expense Account

Buy Down Fee Amortization Adjustment

A Buy Down Fee Amortization Adjustment in Apache Fineract is a special transaction type used to reverse previously recognized income from buy down fee amortization.

Purpose

- Automatically generated when a Buy Down Fee transaction is reversed
- Reverses all already recognized portions (amortized income) linked to the original Buy Down Fee transaction

When It Occurs

- **Trigger:** Only initiated during the reversal of a Buy Down Fee transaction
- Reverses all amortization that has occurred up to that point
- Restores Deferred Income balances and reverses income recognition

Accounting Entries

Transaction Type	Debit	Credit
Buy Down Fee Amortization Adjustment	Income from Buy Down Account	Deferred Income Liability Account

Key Characteristics

- **System-Generated Only:** Cannot be created manually by API or UI
- **Ensures Accounting Integrity:** Keeps amortized and unrecognized balances aligned after reversals
- **Links to Original Amortization:** Maintains traceability by referencing the reversed Buy Down Fee transaction

API Endpoints

Configure Buy Down Fee on Loan Product

- **Endpoint:** `/loanproducts`
- **Method:** `POST`

```
{
  ...
  "enableBuyDownFee": true,           // Mandatory
  "buyDownFeeCalculationType": "FLAT", // Mandatory when enabled
  "buyDownFeeStrategy": "EQUAL_AMORTIZATION", // Mandatory when enabled
  "buyDownFeeIncomeType": "FEE",      // Mandatory when enabled
  "buyDownExpenseAccountId": 123,      // Mandatory when enabled
  "deferredIncomeLiabilityAccountId": 456, // Mandatory when enabled
  "incomeFromBuyDownAccountId": 789    // Mandatory when enabled
}
```

Add Buy Down Fee

- **Endpoint:** `/loans/{loanId}/transactions?command=buyDownFee`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions?command=buyDownFee`
- **Method:** `POST`

```
{
  "transactionDate": "2025-05-01", // Mandatory
  "dateFormat": "yyyy-MM-dd",      // Mandatory
  "locale": "en",                  // Mandatory
  "transactionAmount": 100.0,       // Mandatory
  "paymentTypeId": 1,              // Optional
  "note": "Buy down fee",          // Optional
  "externalId": "BUYDOWN-001"      // Optional
}
```

Response Body

```
{
  "resourceId": 1,
```

```
}
  "resourceExternalId": "BUYDOWN-001"
}
```

Get Buy Down Fee Amortization Info

- **Endpoint:** `/loans/{loanId}/buydown-fees`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/buydown-fees`
- **Method:** GET

Response Body

```
[
  {
    "id": 1,
    "loanId": 123,
    "transactionId": 456,
    "buyDownFeeDate": "2025-05-01",
    "buyDownFeeAmount": 100.0,
    "amortizedAmount": 5.0,
    "notYetAmortizedAmount": 95.0,
    "adjustedAmount": 0.0,
    "chargedOffAmount": 0.0
  }
]
```

Add Buy Down Fee Adjustment

- **Endpoint:**
`/loans/{loanId}/transactions/{buyDownFeeTransactionId}?command=buyDownFeeAdjustment`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions/{buyDownFeeTransactionId}?command=buyDownFeeAdjustment`
- **Method:** POST

```
{
  "transactionDate": "2025-05-01", // Mandatory
  "dateFormat": "yyyy-MM-dd", // Mandatory
  "locale": "en", // Mandatory
  "transactionAmount": 50.0, // Mandatory
  "paymentTypeId": 1, // Optional
  "note": "Buy down fee adjustment", // Optional
  "externalId": "BUYDOWNADJ-001" // Optional
}
```

Response Body

```
{
  "resourceId": 1,
  "resourceExternalId": "BUYDOWNADJ-001"
}
```

Buy Down Fee Template API (to retrieve limits)

- **Endpoint:** `/loans/{loanId}/transactions/template?command=buyDownFee`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions/template?command=buyDownFee`
- **Method:** GET

```
{
  "paymentTypeOptions": [], // List of available payment types
  "currency": {...},        // Currency configuration
  "date": [2025, 5, 29],    // Return the current date
  "amount": 0               // Return the maximum amount that can be applied
}
```

Buy Down Fee Adjustment Template API (to retrieve limits)

- **Endpoint:** `/loans/{loanId}/transactions/template?command=buyDownFeeAdjustment`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}/transactions/template?command=buyDownFeeAdjustment`
- **Method:** GET

```
{
  "paymentTypeOptions": [], // List of available payment types
  "currency": {...},        // Currency configuration
  "date": [2025, 5, 29],    // Return the current date
  "amount": 0               // Return the maximum amount that can be adjusted
}
```

Database Structure

Configuration

Loan Product Table (m_product_loan)

Field	Data Type	Description
<code>enable_buy_down_fee</code>	BOOLEAN	Enable buy down fee feature (default: false)

Field	Data Type	Description
buy_down_fee_calculation_type	VARCHAR	Calculation method (ENUM: FLAT)
buy_down_fee_strategy	VARCHAR	Amortization strategy (ENUM: EQUAL_AMORTIZATION)
buy_down_fee_income_type	VARCHAR	Income type (ENUM: FEE , INTEREST)

Balances

Buy Down Fee Balance Table (**m_loan_buy_down_fee_balance**)

Field	Data Type	Description
id	BIGINT	Primary Key (auto-increment)
version	BIGINT	Version for optimistic locking (NOT NULL)
loan_id	BIGINT	Foreign Key to m_loan.id (NOT NULL)
loan_transaction_id	BIGINT	Foreign Key to m_loan_transaction.id (NOT NULL)
amount	DECIMAL(19,6)	Buy down fee transaction amount (NOT NULL)
date	DATE	Buy down fee transaction date (NOT NULL)
unrecognized_amount	DECIMAL(19,6)	Not yet amortized amount (NOT NULL)
charged_off_amount	DECIMAL(19,6)	Charged-off balance (nullable)
amount_adjustment	DECIMAL(19,6)	Total adjustment amount (nullable)
created_by	BIGINT	User who created the record (NOT NULL)
created_on_utc	DATETIME	Creation timestamp in UTC (NOT NULL)
last_modified_by	BIGINT	Last modifier user ID (NOT NULL)
last_modified_on_utc	DATETIME	Last modification timestamp in UTC (NOT NULL)

Constraints and Indexes

- **Primary Key:** `id`
- **Foreign Keys:**
 - `loan_id` → `m_loan(id)`
 - `loan_transaction_id` → `m_loan_transaction(id)`
 - `created_by` → `m_appuser(id)`
 - `last_modified_by` → `m_appuser(id)`

Related Transaction Types

Buy Down Fee operations are stored in `m_loan_transaction` table with these transaction types:

- `BUY_DOWN_FEE` - Initial buy down fee creation
- `BUY_DOWN_FEE_ADJUSTMENT` - Adjustment to existing buy down fee
- `BUY_DOWN_FEE_AMORTIZATION` - Daily amortization transaction
- `BUY_DOWN_FEE_AMORTIZATION_ADJUSTMENT` - Adjustment to amortization

Accounting Entries

Transaction Type	Debit	Credit
Buy Down Fee	Buy Down Expense Account	Deferred Income Liability Account
Buy Down Fee Amortization	Deferred Income Liability Account	Income from Buy Down Account
Buy Down Fee Adjustment	Deferred Income Liability Account	Buy Down Expense Account
Buy Down Fee Amortization Adjustment	Income from Buy Down Account	Deferred Income Liability Account

Business Events

Triggered for Buy Down Fee

- `LoanBuyDownFeeTransactionCreatedBusinessEvent`
- `LoanBalanceChangedBusinessEvent`

Daily Amortization

- `LoanBuyDownFeeAmortizationTransactionCreatedBusinessEvent`
- `LoanBuyDownFeeAmortizationAdjustmentTransactionCreatedBusinessEvent`

Buy Down Fee Adjustment

- `LoanBuyDownFeeAdjustmentTransactionCreatedBusinessEvent`
- `LoanBalanceChangedBusinessEvent`

Reversal

- `LoanAdjustTransactionBusinessEvent`

Available Disbursement Calculation

Buy Down Fee does not affect the available disbursement amount calculation:

$$\begin{aligned} \text{Available Disbursement} &= \text{Approved Loan Amount} \\ &\quad - \text{Total Disbursed Amount} \\ &\quad - \text{Total Capitalized Income} \end{aligned}$$

Buy Down Fee transactions are separate from the loan disbursement logic and do not reduce the available disbursement amount.

Notes



- Buy down fee transactions support backdating
- Adjustment transactions must not predate the original buy down fee
- No automatic reversal is supported; must be handled manually via dedicated transactions
- Proper GL accounts must be set for Buy Down Expense, Deferred Income Liability, and Income from Buy Down to enable this functionality

Approved amount modification on loans

Overview

In Apache Fineract, after a loan is disbursed, it is possible to alter the principal amount that the loan was approved with. This means that the amount to be disbursed can be fine-tuned throughout the loan lifecycle. The approved loan amount can either be modified directly or indirectly through different endpoints.

Supported Loan Type

Approved amount modifications are supported on all loan types.

Business Events

- Triggers a new business event: Loan Approved Amount Changed

API Endpoints

Modifying approved amount on loans

Fineract supports the direct modification of the approved amount on loans

- **Endpoint:** `/loans/<loan_id>/approved-amount`
- **Alternative Endpoint:** `/loans/external-id/<loan_external_id>/approved-amount`
- **Method:** `PUT`

```
{
  "amount": 1000.0,
  "locale": "en"
}
```

Response Body

```
{
  "changes": {
    "locale": "en",
    "newApprovedAmount": 1000.0,
    "oldApprovedAmount": 1500.0
  },
  "clientId": 6,
  "groupId": 10,
  "officeId": 2,
  "resourceExternalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7",
  "resourceId": 3
}
```

Validations

- The approved amount of the loan cannot be lower than the `total principal disbursed + total expected principal + total principal from capitalized income transactions`.
- The approved amount of the loan cannot be set higher, than the proposed amount of the loan or if `allow approved/disbursed over applied amount` configuration is enabled then the calculated threshold.

Modifying available disbursement amount on loans

Fineract supports the indirect modification of the approved amount on loans. This is called modifying the available disbursement amount.



Available disbursement amount is only a calculated value used by this endpoint to indirectly update the approved amount of the loan. It is not stored anywhere.

The approved amount is calculated as: `total principal disbursed + total expected principal +`

total principal from capitalized income + "amount" from the request.

- **Endpoint:** `/loans/<loan_id>/available-disbursement-amount`
- **Alternative Endpoint:** `/loans/external-id/<loan_external_id>/available-disbursement-amount`
- **Method:** PUT

```
{
  "amount": 100.0,
  "locale": "en"
}
```

Response Body

```
{
  "changes": {
    "locale": "en",
    "newApprovedAmount": 100.0,
    "oldApprovedAmount": 1000.0,
    "newAvailableDisbursementAmount": 100.0,
    "oldAvailableDisbursementAmount": 1000.0
  },
  "clientId": 6,
  "groupId": 10,
  "officeId": 2,
  "resourceExternalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7",
  "resourceId": 3
}
```

Validations

- The available disbursement amount cannot be lower than 0.
- The approved amount of the loan cannot be set higher, than the proposed amount of the loan or if `allow approved/disbursed over applied amount` configuration is enabled then the calculated threshold. This means that the new available disbursement amount cannot be higher than `maximumLoanPrincipalThreshold - total principal disbursed - total expected principal - total principal from capitalized income`

Approved amount history

Modifying the approved amount of the loan through either endpoint also creates a history entry that can be used to observe the changes overtime.

- **Endpoint:** `/loans/<loan_id>/approved-amount`
- **Alternative Endpoint:** `/loans/external-id/<loan_external_id>/approved-amount`
- **Method:** GET

Response Body

```
[
  {
    "loanId": 152,
    "externalLoanId": "9e058913-3de8-4f6e-9e09-4b2067c4bb91",
    "newApprovedAmount": 800.000000,
    "oldApprovedAmount": 1000.000000,
    "dateOfChange": "2025-08-05T16:35:43.427229+02:00"
  },
  {
    "loanId": 152,
    "externalLoanId": "9e058913-3de8-4f6e-9e09-4b2067c4bb91",
    "newApprovedAmount": 600.000000,
    "oldApprovedAmount": 800.000000,
    "dateOfChange": "2025-08-05T16:35:43.543779+02:00"
  },
  {
    "loanId": 152,
    "externalLoanId": "9e058913-3de8-4f6e-9e09-4b2067c4bb91",
    "newApprovedAmount": 400.000000,
    "oldApprovedAmount": 600.000000,
    "dateOfChange": "2025-08-05T16:35:43.603855+02:00"
  }
]
```

Backdated interest modification

In the previous implementation we were only allowing the interest rate modification from current date and from now on we will allow Interest modification backdated on progressive loans as well.



Only available on progressive loans.

Functionality

Validations updated to allow backdated interest change, even on charged-off or otherwise closed loan. Making progressive loans more flexible.

1. Interest rate can be modified from backdate any date from first disbursement date
2. Interest will be affected from the applied date itself.
3. Backdate can be done on already paid Installments as well
4. Repayment schedule will be recalculated with New EMI and Interest from the Interest applied schedule date
 - Backdated PAID Installments and Unpaid/Partial Paid Installment EMI would be changed as per the new calculated EMI
5. Installments paid Interest amount will be reverse replayed as per the new Interest rate from the

applied date

- Transactions will be reversed replayed if there is any change in allocations
- Accrual adjustments will be done during reverse replay (during the COB process)

6. No of Installments will remain the same, only EMI and Interest would get affected

7. Backdated Interest modification allowed on the loan that is charged off

- If the repayment that was made before the charge-off is reversed and replayed due to backdated Interest modification, then the accounting entry of the reversed transaction and replayed transaction should follow standard accounting rules and not charge-off accounting rules

8. Backdated interest modification allowed on the loan that is overpaid, and CBR is complete.

- Any action that triggers the recalculation (ex: reversal of backdated transaction) on the CBR loan will result in treating CBR as a credit transaction during reverse-replay. Same logic to be applied if the backdated interest modification is allowed on CBR loan accounts.

9. Asset transfer (externalization)

- If the repayment that was made before the asset owner change got reversed and replayed due to backdated Interest modification, then, the accounting entry for the reversed transaction and replayed transaction include the tag of the current asset owner

10. Since backdated Interest modification is allowed on CBR/overpaid loans, we can keep the modification open on closed loans as well

- system will do the chronological reverse-replay when the backdated Interest is changed on closed loans, schedule and transaction will be allocated accordingly

API endpoints

Create reschedule loans request (create reschedule)

- **Endpoint:** `/rescheduleloans`
- **Method:** `POST`

Example interest rate change request

```
{
  "loanId": 1, // Mandatory
  "newInterestRate": 1, // Mandatory for interest rate change type
  "reschedule": {
    "rescheduleFromDate": "2024-01-01", // Mandatory
    "rescheduleReasonId": 54, // Mandatory
    "submittedOnDate": "2024-01-01", // Mandatory
    "dateFormat": "yyyy-MM-dd", // Mandatory
    "locale": "en" // Mandatory
  }
}
```

Example interest rate change response

```
{
  "resourceId": 1,
  "loanId": 1,
  "clientId": 1,
  "officeId": 1
}
```

Update reschedule loans request (approve reschedule)

- **Endpoint:** `/rescheduleloans/<reschedule_Id>`
- **Method:** `POST`

Example reschedule approve request

```
{
  "approvedOnDate": "2024-01-01", // Mandatory for approval
  "submittedOnDate": "2024-01-01", // Mandatory
  "dateFormat": "yyyy-MM-dd", // Mandatory
  "locale": "en" // Mandatory
}
```

Example reschedule approve response

```
{
  "changes": {
    "approvedByUserId": 1,
    "approvedOnDate": "2024-01-01",
    "dateFormat": "yyyy-MM-dd",
    "locale": "en"
  },
  "clientId": 186,
  "loanId": 188,
  "officeId": 1,
  "resourceId": 35
}
```

Interest Rate Modification For Progressive Loan

Overview

The Original Interest Rate Modification feature allows updating the interest rate for **active loans**. The updated interest rate can be applied only for active loans and effective date should be in the future.

This capability is introduced to support flexible interest rate management in loan lifecycles, reflecting changes due to inflation, risk reassessment, or customer-specific conditions.

New Progressive Loan Interest Change Modification feature can be applied for overpaid, charged off or even backdated cases, which makes it much more usable.

Scope and Limitations

- Only supported for **progressive** loan types.
- Loans must be **disbursed**
- Interest rate modifications apply from a specified **applied date**, which can be backdated from the **original disbursement date** onward.
- **Paid EMIs (Equal Monthly Installments) and interest amounts may be affected** by backdated interest rate changes.
- The modified interest rate affects **EMI amounts**
- Installment counts are not affected
- **Reversals/backdated repayments** are allowed after modification.

Feature Behavior

- When a new interest rate is applied, the system recalculates EMI values starting from the **applied date**.
- If the applied date is in the past, previously paid installments will be reprocessed under the new interest rate.
- The updated interest rate is effective **from the applied date itself**.
- Repayment schedule is updated using the current recalculation strategy.

Configuration

Interest calculation and interest recalculation strategies are **inherited from the loan product** configuration at the time of loan application.

This means:

- The interest calculation method (e.g., declining balance, flat) and
- The interest recalculation strategy

are **fixed per loan account** once the loan is created.

As a result, **no further changes** to these configurations are possible after loan creation. Any interest rate modifications must operate within the originally defined calculation and recalculation strategies.

API Specification

Endpoint

POST `DomainName/api/v1/rescheduleloans`

related API-s

Retrieve a Loan Reschedule Request

GET `DomainName/api/v1/rescheduleloans/{requestId}`

Retrieve a Preview of The New Loan Repayment Schedule

GET `DomainName/api/v1/rescheduleloans/{requestId}?command=previewLoanReschedule`

Reject a Loan Reschedule Request

POST `DomainName/api/v1/rescheduleloans/{requestId}?command=reject`

Approve a Loan Reschedule Request

POST `DomainName/api/v1/rescheduleloans/{requestId}?command=approve`

Request Payload

The following fields are accepted in the request body for interest rate modification. All date fields must follow the format specified by the `dateFormat` field, and parsing is performed using the specified `locale`.

Field	Type	Description
<code>loanId</code>	Long	Identifier of the loan to be modified. Must refer to a disbursed progressive loan.
<code>newInterestRate</code>	BigDecimal	Required. The new interest rate to be applied. Must be zero or positive, and comply with the loan product's min/max rate constraints.
<code>dateFormat</code>	String	Required when any date fields are provided. Defines the expected format of date values (e.g., <code>yyyy-MM-dd</code> , <code>dd-MM-yyyy</code> , etc.).
<code>locale</code>	String	Required. Specifies the locale (e.g., <code>en</code> , <code>fr</code> , <code>in</code>) used for parsing numbers and dates.
<code>submittedOnDate</code>	String	Optional. The date the request is submitted. If provided, must match the specified <code>dateFormat</code> .
<code>rescheduleFromDate</code>	String	Required. The date from which the new interest rate becomes effective. Must be on or after the loan disbursement date. Format must match <code>dateFormat</code> .
<code>rescheduleReasonComment</code>	String	Optional. A free-text comment describing the reason for the interest rate change.
<code>rescheduleReasonId</code>	Long	Optional. ID referencing a predefined rescheduling reason (e.g., from a dropdown or lookup table).



The following fields are **not applicable** in the context of an interest rate change

request:

- `adjustedDueDate`
- `extraTerms`
- `graceOnPrincipal`
- `graceOnInterest`

These parameters are reserved for other loan rescheduling operations.

newInterestRate

When processing an interest rate modification request, the system validates the `newInterestRate` parameter as follows:

- The value must be a valid `BigDecimal` parsed with the appropriate locale.
- The interest rate must be **zero or positive**.
- If defined on the loan product, the new interest rate must satisfy the following boundaries:
 - It must be **greater than or equal to** the product-level `minNominalInterestRatePerPeriod`.
 - It must be **less than or equal to** the product-level `maxNominalInterestRatePerPeriod`.

These boundaries are enforced using the product's configured range at the time the loan was applied. If no minimum or maximum is set on the product, only the zero-or-positive constraint is enforced.

Example

Example Create Request

POST `DomainName/api/v1/rescheduleloans`

POST `rescheduleloans`
Content-Type: `application/json`

```
{
  "loanId": 1,
  "graceOnPrincipal": null,
  "graceOnInterest": null,
  "extraTerms": null,
  "rescheduleFromDate": "04 December 2014",
  "dateFormat": "dd MMMM yyyy",
  "locale": "en",
  "recalculateInterest": null,
  "submittedOnDate": "04 September 2014",
  "newInterestRate": 28,
  "rescheduleReasonId": 1
}
```


Response

```
{
  "loanId": 1,
  "resourceId": 2
}
```

Example Approval

POST `DomainName/api/v1/rescheduleloans/{requestId}?command=approve`

POST `rescheduleloans/2?command=approve`
Content-Type: `application/json`

```
{
  "locale": "en",
  "dateFormat": "dd MMMM yyyy",
  "approvedOnDate": "11 September 2014"
}
```

```
{
  "loanId": 1,
  "resourceId": 2,
  "changes": {
    "locale": "en",
    "dateFormat": "dd MMMM yyyy",
    "approvedOnDate": "11 September 2014",
    "approvedByUserId": 3
  }
}
```

Developer Notes

The core concept is that the `AdvancedPaymentScheduleTransactionProcessor` processes transactions in order of their effective date, allowing it to handle backdated transaction cases.

The transaction processor uses the `EMICalculator` to manage interest rate changes over time, ensuring that changes only affect future transactions relative to the actual processing transaction. The `ProgressiveLoanInterestScheduleModel` is responsible for holding and calculating interest for future installments.

The underlying principle is to split repayment periods into smaller interest periods, enabling the calculation of interest for partial repayment periods. This approach makes it easier to adjust interest rates for specific interest periods as needed.

Contract Termination

Overview

Contract Termination in Apache Fineract is a loan management feature that allows financial institutions to terminate loan contracts. When applied to loans with unpaid installments, this functionality accelerates the maturity date and makes the outstanding loan balance immediately due as of the termination date.

Purpose

This functionality enables financial institutions to:

- Terminate loan contracts when required by business rules
- Accelerate payment schedules by making outstanding balances immediately due
- Maintain proper loan status tracking and accounting
- Support charge-off and recovery operations on terminated loans

Supported Loan Type



Contract Termination is only supported for:

- Progressive Loan Schedules
- Active loan accounts

Other loan schedule types and inactive loan states are not supported.

Business Rules

Eligibility Requirements

Contract termination can only be applied when:

- **Loan Status:** The loan must be in **Active** status
- **Schedule Type:** Only **Progressive** loan schedule type is supported
- **Not Charged Off:** Loan must not be in charged-off state
- **Not Already Terminated:** Loan must not already have contract termination applied

Termination Date Rules

- Contract termination can only be done as of the current business date
- Backdated termination is not allowed
- No future-dated termination permitted

Schedule Impact

When contract termination is applied:

- **Maturity Acceleration:** If unpaid installments exist, the loan maturity date is accelerated to the termination date
- **Interest Calculation:** Interest is calculated only until the contract termination date
- **Maturity Date Updated:** The loan maturity date is updated to the contract termination date
- **Principal Outstanding:** The full outstanding principal balance becomes due
- **Delinquency Bucketing:** Continues as per the new accelerated schedule

Post-Termination Operations

After contract termination:

- **Charge-off Allowed:** Terminated loans can be charged off
- **Charge-backs Allowed:** Terminated loans support charge-back transactions
- **Future Installments:** All installments scheduled after termination date are removed from the schedule
- **Accrual Activities:** Accrual and accrual activity transactions stop after termination
- **Backdated Payments:** Backdated payments and reversals are allowed
- **Contract Termination Reversal:** The termination can be undone/reversed

Special Handling

Post-Maturity Termination

If contract termination is done after the original maturity date:

- No schedule acceleration occurs (installments and maturity date remain unchanged)
- Interest calculation follows normal rules up to the original maturity date
- The loan remains in its current state without forced acceleration

Contract Termination Reversal

- Contract termination can be undone/reversed
- Schedule will be recalculated and reapplied accordingly
- All associated transactions are properly reversed and replayed

Transaction Types

Contract Termination Transaction

The Contract Termination transaction in Apache Fineract performs the following actions:

- **Accelerates Payment Schedule:** Makes all outstanding amounts immediately due

- **Creates Distinct Transaction:** Tracked separately with transaction type "Contract Termination"
- **Updates Loan Sub-Status:** Changes loan sub-status to `CONTRACT_TERMINATION` (value: 900)
- **Triggers Schedule Recalculation:** Updates repayment schedule with accelerated terms
- **No Accounting Entries:** Contract termination itself does not generate accounting entries
- **Stops Accrual Activity:** Interest accrual and accrual activities cease after termination

Transaction Behavior

- Transaction date is set to current business date
- Amount represents total outstanding balance as calculated by loan summary
- Triggers loan reprocessing for interest-bearing loans with recalculation enabled
- Non-monetary transaction (excluded from monetary transaction queries)

Accrual Transactions

During contract termination, the system may generate:

- Final accrual transaction up to termination date
- Accrual adjustment transaction if needed
- Associated journal entries
- Relevant business events for accrual processing

Contract Termination Undo

The Contract Termination Undo transaction reverses a previous contract termination:

- **Reverses Termination Transaction:** Marks the original termination as reversed
- **Removes Sub-Status:** Restores loan to previous sub-status state
- **Recalculates Schedule:** Regenerates original repayment schedule
- **Reprocesses Transactions:** Re-runs transaction processing logic
- **Triggers Business Events:** Notifies system of balance and status changes

API Endpoints

Apply Contract Termination

- **Endpoint:** `/loans/{loanId}?command=contractTermination`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}?command=contractTermination`
- **Method:** `POST`

```
{
  "note": "Contract terminated due to default",           // Optional
  "externalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7"  // Optional
}
```

```
}
```

Response Body

```
{
  "entityId": 1,
  "entityExternalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7",
  "officeId": 1,
  "clientId": 1,
  "loanId": 1,
  "changes": {
    "subStatus": 900
  }
}
```

Undo Contract Termination

- **Endpoint:** `/loans/{loanId}?command=undoContractTermination`
- **Alternative Endpoint:** `/loans/external-id/{loanExternalId}?command=undoContractTermination`
- **Method:** `POST`

```
{
  "note": "Reversing contract termination", // Optional
  "reversalExternalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7" // Optional
}
```

Response Body

```
{
  "entityId": 1,
  "entityExternalId": "95174ff9-1a75-4d72-a413-6f9b1cb988b7",
  "officeId": 1,
  "clientId": 1,
  "groupId": null,
  "loanId": 1,
  "changes": {
    "subStatus": null
  }
}
```

Business Events

Triggered for Contract Termination

- `LoanTransactionContractTerminationPostBusinessEvent` - After termination processing

- `LoanBalanceChangedBusinessEvent` - After termination processing
- `LoanAdjustTransactionBusinessEvent` - During termination transaction processing

Triggered for Contract Termination Undo

- `LoanUndoContractTerminationBusinessEvent` - Before and after undo operation
- `LoanBalanceChangedBusinessEvent` - After schedule recalculation
- `LoanAdjustTransactionBusinessEvent` - During reversal processing

Database Impact

Loan Sub-Status

Updated on Loan (`m_loan`)

Field	Data Type	Description
<code>sub_status_enum</code>	<code>SMALLINT</code>	Set to 900 (<code>CONTRACT_TERMINATION</code>) when terminated, reset to NULL when undone

Transaction Records

Loan Transaction (`m_loan_transaction`)

Field	Data Type	Description
<code>transaction_type_enum</code>	<code>SMALLINT</code>	Set to 38 (<code>CONTRACT_TERMINATION</code>)
<code>transaction_date</code>	<code>DATE</code>	Current business date
<code>amount</code>	<code>DECIMAL(19,6)</code>	Total outstanding balance
<code>is_reversed</code>	<code>BOOLEAN</code>	Set to TRUE when undone

Validation Rules

Contract Termination Validation

- Loan must be in active status (`loan.isOpen()`)
- Loan product must use Progressive schedule type
- Loan must not be charged off (`!loan.isChargedOff()`)
- Loan must not already be terminated (`!loan.isContractTermination()`)
- Client or group must be active

Contract Termination Undo Validation

- Original contract termination transaction must exist
- Transaction must not be already reversed
- Proper permissions required for reversal operations

Integration Points

Charge Operations

- Charge-off operations can be performed on terminated loans
- Charge-back transactions are supported on terminated loans
- Charge adjustments follow normal business rules

Delinquency Management

- Delinquency bucketing continues based on accelerated schedule
- Delinquency calculations use the new due dates from termination

Accounting Integration

- No direct accounting entries for contract termination transaction
- Potential accrual transactions generated up to termination date
- Journal entries created for final accrual transactions
- Interest accrual stops after termination
- Business events triggered for accrual processing

Notes



- Contract termination is irreversible through normal business processes once additional transactions occur
- Proper authorization and audit trails are maintained for all termination activities
- Integration with external systems should account for accelerated payment schedules
- Only Progressive loan schedule type supports this functionality due to schedule recalculation requirements

Loan Charges

Overview

Loan charge products can be created with different charge time types and charge calculation types. The created charge product can be associated with loan products or individual loan accounts to

automate fee and penalty application throughout the loan lifecycle.

Benefits

- Automates charge application at disbursement, specified dates, or installment schedules
- Supports multiple calculation methods including flat amounts and various percentage-based calculations
- Handles multi-disbursement loans with distinct first disbursement vs. tranche disbursement behavior
- Provides penalty automation for overdue installments
- Enables flexible fee structures for different loan products

Charge Time Types and Calculation Types

The following charge time types and corresponding calculation types are supported for loans:

Charge Time Type	Description	Available Charge Calculation Types
Disbursement	Charged at the time of first disbursement. For multi-disbursement loans, it applies only to the first disbursement.	<ul style="list-style-type: none">• Flat: Fixed amount defined at charge creation• % amount: Percentage charged on approved loan amount• % interest: Percentage charged on total interest amount as per first disbursement• % loan amount + interest: Percentage charged on the disbursed amount plus total interest as per first disbursement• % disbursement amount: Percentage charged on actual disbursement amount

Charge Time Type	Description	Available Charge Calculation Types
Specified Due Date	Charge applied on a specific date defined by the user within the loan lifecycle.	<ul style="list-style-type: none"> • Flat: Fixed amount defined at charge creation • % amount: Percentage charged on approved loan amount • % interest: Percentage charged on total interest amount as per first disbursement • % loan amount + interest: Percentage charged on the disbursed amount plus total interest as per first disbursement
Installment Fees	Charged with each repayment installment throughout the loan term.	<ul style="list-style-type: none"> • Flat: Fixed amount per installment • % amount: Percentage of approved amount divided across installments • % loan amount + interest: Percentage of principal plus interest for each installment
Overdue Fees	Penalties automatically applied when installments become overdue.	<ul style="list-style-type: none"> • Flat: Fixed penalty amount per overdue installment • % amount: Percentage penalty based on overdue amount
Tranche Disbursement	Charged for each disbursement tranche in multi-disbursement loans.	<ul style="list-style-type: none"> • Flat: Fixed amount per disbursement tranche • % disbursement amount: Percentage charged on each disbursement tranche amount

API Endpoints

Charge Product Management

- **Endpoint:** [/v1/charges](#)
- **Methods:** [GET](#) (list all charges), [POST](#) (create charge)

- **Purpose:** Manage charge product definitions
- **Endpoint:** `/v1/charges/{chargeId}`
- **Methods:** `GET` (retrieve), `PUT` (update), `DELETE` (delete)
- **Purpose:** Individual charge product operations
- **Endpoint:** `/v1/charges/template`
- **Methods:** `GET`
- **Purpose:** Retrieve charge creation template with dropdown options

Loan Charge Management

- **Endpoint:** `/v1/loans/{loanId}/charges`
- **Alternative:** `/v1/loans/external-id/{loanExternalId}/charges`
- **Methods:** `GET` (list loan charges), `POST` (add charge to loan)
- **Purpose:** Manage charges on specific loans
- **Endpoint:** `/v1/loans/{loanId}/charges/{chargeId}`
- **Alternative:** `/v1/loans/external-id/{loanExternalId}/charges/external-id/{chargeExternalId}`
- **Methods:** `GET` (retrieve), `PUT` (update), `DELETE` (remove), `POST` (execute commands)
- **Purpose:** Individual loan charge operations
- **Endpoint:** `/v1/loans/{loanId}/charges/template`
- **Methods:** `GET`
- **Purpose:** Get template for adding charges to loans

Charge Commands

Loan charges support command-based operations via `POST` requests with `command` query parameter:

- `POST /v1/loans/{loanId}/charges/{chargeId}?command=waive` - Waive charge
- `POST /v1/loans/{loanId}/charges/{chargeId}?command=pay` - Pay charge directly

Configuration

Charge Product Setup

1. **Charge Time Type:** Select when the charge should be applied (disbursement, specified date, installments, overdue, or tranche)
2. **Charge Calculation Type:** Choose calculation method (flat amount or percentage-based)
3. **Amount/Percentage:** Define the charge amount or percentage value
4. **Currency:** Must match the loan product currency
5. **Penalty Flag:** Mark charges as penalties for reporting purposes
6. **Active Status:** Enable/disable charge availability

Loan Product Association

- Associate default charges to loan products during product configuration
- Default charges are automatically applied when loans are created from the product
- Individual charges can be added to specific loans regardless of product defaults

Validation Rules

- Charge currency must match loan currency
- Specified due dates must fall within the loan term
- Percentage values must be within valid ranges (0-100)
- Overdue charges require penalty flag to be enabled
- Disbursement charges can only be added before loan disbursement

Journal Entry Aggregation

Overview

The Journal Entry Aggregation Job is a Spring Batch-based solution designed to efficiently aggregate journal entries in the Fineract system. This job processes journal entries in configurable chunks, improving performance and resource utilization when dealing with large volumes of financial transactions.

Key Features

Chunk-based Processing

- Processes journal entries in configurable batch sizes
- Reduces memory footprint by working with manageable data subsets
- Improves performance through efficient batch processing

Tracking and Deduplication

- Tracks processed date ranges to prevent duplicate aggregations
- Uses `JournalEntryAggregationTracking` to maintain execution history
- Skips already processed date ranges in subsequent runs

Configurable Exclude Recent N Days

- Excludes the last N days (from business date) from processing
- Default `Exclude Recent N Days` can be customized via application properties

How It Works

Job Flow

Job Initialization

- Determines the date range to process based on last execution
- Sets up execution context with date boundaries

Data Reading

- Fetches unaggregated journal entries within the target date range
- Groups entries by GL account, product, office, and other dimensions

Processing

- Aggregates debit and credit amounts for each group
- Handles external asset owner mappings
- Processes data in configurable chunk sizes

Tracking

- Records successful aggregation runs
- Maintains execution history for future reference

Configuration

Job Parameters

- **aggregatedOnDate**: (Optional) Specific date to process (defaults to business date)
- **chunkSize**: (Optional) Number of records to process in each chunk

Application Properties

```
# Exclude Recent N days from aggregation
fineract.job.journal-entry-aggregation.exclude-recent-N-days=1

# Chunk size for batch processing
fineract.job.journal-entry-aggregation.chunk-size=1000
```

Usage

Manual Execution

Trigger the job manually through the Fineract API:

```
POST /jobs/short-name/JRNL_AGG
Content-Type: application/json
```

```
{  
}
```

Scheduled Execution

Configure the job to run on a schedule by adding to your scheduler configuration.

Monitoring

Monitor job execution through:

- Job execution logs
- `JOURNAL_ENTRY_AGGREGATION_TRACKING` table
- Spring Batch job execution tables

Best Practices

Chunk Size Tuning

- Larger chunks improve throughput but increase memory usage
- Monitor memory usage and adjust chunk size accordingly

Scheduling

- Schedule during off-peak hours for large datasets
- Consider running more frequently with smaller `Exclude Recent N Days` values

Error Handling

- Failed jobs can be restarted from the last successful chunk
- Review job execution logs for any processing issues

Performance Considerations

- **Indexing:** Ensure proper indexes exist on `aggregated_on_date`, `office_id`, and other filtering columns
- **Partitioning:** Consider partitioning large journal entry tables by date for better performance
- **Batch Window:** Allocate sufficient time for the job to complete during maintenance windows

Database Schema

`m_journal_entry_aggregation_summary` Table

This table stores the aggregated journal entry amounts, grouped by various dimensions for efficient reporting and analysis.

Column	Type	Nullable	Description
id	BIGINT	No	Primary key
gl_account_id	BIGINT	No	Reference to acc_gl_account
product_id	BIGINT	Yes	Reference to the product (if applicable)
office_id	BIGINT	No	Reference to m_office
entity_type_enum	SMALLINT	No	Type of entity (e.g., loan, savings)
submitted_on_date	DATE	No	The date of the business date when entry was submitted
aggregated_on_date	DATE	No	The date when aggregation was performed
debit_amount	DECIMAL(19,6)	No	Sum of debit amounts
credit_amount	DECIMAL(19,6)	No	Sum of credit amounts
external_owner_id	BIGINT	Yes	Reference to external owner (if applicable)
job_execution_id	BIGINT	No	Reference to batch job execution
created_date	TIMESTAMP	No	Record creation timestamp
last_modified_date	TIMESTAMP	Yes	Last modification timestamp

The table is designed to support efficient querying of aggregated financial data by:

- * Date ranges (using **submitted_on_date** and **aggregated_on_date**)
- * Organizational structure (using **office_id**)
- * Financial dimensions (using **gl_account_id** and **product_id**)
- * Entity types (using **entity_type_enum**)

m_journal_entry_aggregation_tracking Table

This table maintains a history of aggregation job executions, tracking which date ranges have been processed to prevent duplicate aggregations.

Column	Type	Nullable	Description
id	BIGINT	No	Primary key
job_execution_id	BIGINT	No	Reference to Spring Batch job execution

Column	Type	Nullable	Description
aggregated_on_date_from	DATE	No	Start date of the aggregation period
aggregated_on_date_to	DATE	No	End date of the aggregation period
submitted_on_date	DATE	No	The date of the business date when entry was submitted
status	VARCHAR(20)	No	Status of the aggregation (e.g., COMPLETED, FAILED)
error_message	TEXT	Yes	Error details if the job failed
start_time	TIMESTAMP	No	When the aggregation started
end_time	TIMESTAMP	Yes	When the aggregation completed
records_processed	INT	Yes	Number of records processed
created_date	TIMESTAMP	No	Record creation timestamp
last_modified_date	TIMESTAMP	Yes	Last modification timestamp

Key aspects of the tracking table:

- * Tracks the exact date ranges processed in each job execution
- * Maintains job status and error information for debugging
- * Records performance metrics (processing time, record counts)
- * Used by the job to determine which date ranges need processing in subsequent runs

Indexes are created on frequently queried columns to ensure optimal performance for reporting and analysis.

This aggregation job provides a robust, scalable solution for processing journal entries while maintaining data integrity and providing clear audit trails of all aggregation activities.

Loan Re-Aging

Overview

Re-aging also known as Settlement Plan is to assist customers who are in duress around repaying their loans;

The goal is to create a new set of Installments for the same loan account from the principal outstanding pending on the loan account

For example, for a customer who is severely behind on payments, they could agree to a settlement plan where instead of having to pay now, it could be spread over the next 12 months, paying back the remaining 750 euro balance but over 10 equal payments.

Introduction

Purpose

This document specifies the functional and business requirements for the Re-Aging feature for Progressive loan products in Fineract.

It defines rules for interest handling, schedule regeneration, charge/fee mapping, special installments, and reversal logic to ensure consistent behavior across different interest handling methods.

Scope

The scope of this document includes:

- * Re-Aging for Progressive, interest-bearing loans
- * Handling of Default, Equal Amortization scenarios
- * Support for backdated transactions
- * Schedule regeneration and installment remapping
- * Special collected installment creation
- * Reversal and replay of Re-Age transactions
- * User-driven configuration of re-aged schedule (start date, number of installments, period type/frequency)

This document does **not** cover:

- * Non-Progressive loan strategies
- * Charge-off loan accounts
- * Interest-bearing loans outside the scope of Fineract Progressive product definition
- * External API documentation (covered separately)

Applicability

The requirements described apply to:

- * Progressive loan products
- * Non-interest-bearing and Interest-bearing accounts with or without interest recalculation enabled
- * Loan accounts where re-aging may occur at any date \geq last repayment date

Definitions and Key Concepts

Re-Age: The process of recalculating and adjusting the remaining loan schedule, including principal, interest, fees, and penalties, starting from a specified start date.

Transaction Date: The accounting/event date when re-aging is applied.

Start Date: The due date of the first re-aged installment (re-age start date).

Special Collected Installment: A single installment created to capture paid portions from installments with due-date after the transaction date.

N+1 Installment: Any additional installment added to the schedule before or after re-aging to balance principal, interest, and fees.

Interest Handling: The method applied during re-aging — Default, Equal Amortization.

Chargeback: A partial or full reversal of a previously posted repayment, adjusting principal, interest, and/or fees back to their prior state.

General Re-Aging Rules

Eligibility Criteria

- Re-Aging is allowed only for Progressive loan products.
- The re-age transaction date must be **greater than or equal to the last repayment date**.
- Re-Aging can be applied **anywhere in the loan lifecycle**, including before maturity.

Transaction Date vs Start Date

- **Transaction Date**: The accounting/event date when re-aging is applied.
- **Start Date**: The due date of the first re-aged installment.
- In backdated scenarios, **transaction date = start date**.
- In non-backdated scenarios, **transaction date ≤ start date**.
- The re-aged schedule begins from the start date, irrespective of past due installments.

Partial Payments Handling

- Any installment with a **partially paid amount** before the transaction date:
 - The **paid portion remains** allocated to the original installment.
 - The **unpaid portion** is included in the re-aged schedule.
- Installments fully paid before transaction date remain unchanged.

Down-Payment Handling

- Down-payments are **not affected** by re-aging.
- They remain linked to the original installment and are excluded from the re-age calculations.

Chargebacks

- Charge-back amounts (principal, interest, fees) are processed according to the selected interest handling scenario:
 - Equal Amortization: split across new installments.
 - Default: principal re-aged, interest moved to first new installment.
- Only **unpaid** are affected.

Special Collected Installments

- Any installment with **due-date > transaction date** that has payments posted before the transaction date:
 - Paid portions are **merged into a single special collected installment**.
 - Due date = transaction date.
 - GL postings must remain identical to original payments.

- The special installment does not participate in interest/principal redistribution.
- Supports reverse and replay logic for backdated transactions.

Reversal and Replay Rules

- Re-Age transactions can be **reversed only if they are the latest transaction** on the loan account.
- Any backdated repayment or reversal triggers **reverse + replay**:
 - Reverts schedule to original state before re-age.
 - Removes special collected installment if it exists.
 - Restores charge/fee mapping.
- Backdated transaction posting or reversals triggers **reverse + replay**

User Input Parameters

Users must provide the following inputs when applying re-aging:

- * **Start Date**: first due date of the re-aged schedule.
- * **Number of Re-aged Installments**: total installments to generate.
- * **Period Type**: Daily, Weekly, Monthly, or Yearly.
- * **Period Frequency**: number of units per period type.
- * **Interest Handling Method**: Default, Equal Amortization.
- * These inputs drive the schedule generation, replacement of future installments, and calculation of N+1 installment.
- * In case of non-interest-bearing loans interest handling strategy is omitted

Implementation details

Implementation core is located in AdvancedPaymentScheduleTransactionProcessor.

the entrypoint of the feature is handleReAge.

handleReAge function is the purpose of configure the different re-age strategies and handle both interest bearing and non interest bearing scenarios.

There is no real handling, and all the different specific handlers are described in the related interest handling scenarios dev notes chapters.

Non-Interest-Bearing Scenario

Background and Context

Re-aging functionality for **Non-Interest-Bearing Progressive loans** existed only **after maturity date**.

This means Fineract now supports re-aging before maturity date.

This enhancement introduces the ability to:

- * Apply re-aging at **any point after the first disbursement**
- * Apply re-aging **before or after maturity**
- * Override installment schedule when necessary
- * Handle partially paid installments via **special installment creation**

Principal-only amortization — no future interest calculation involved.

General Re-Aging Rules

- Re-aging date **may be before last repayment date**
- Only **outstanding principal + outstanding fees/penalties** are redistributed
- Re-aging transaction remains **non-monetary**
- GL entries are not created

Fees/Penalties behavior:

- * Fees/penalties **retain their due dates**
- * Re-mapped to appropriate new installment when impacted by re-aging

Schedule Modification Requirements

When the re-aging transaction occurs before maturity date or overlaps future repayment periods:

- **Installments before the re-aging transaction date**
 - Only **paid portions** are retained
 - Any **unpaid portion** is included in re-aging redistribution
- **Installments on or after the re-aging transaction date**
 - Replaced with newly generated re-aged installments
 - New due dates and installment structure are based on re-aging configuration (period count, period type, frequency)

Handling of N+1 Installments (Post-Maturity Items)

N+1 installments exist to hold items that occur **after** the core repayment term — typically post-maturity fees and penalties (and in some contexts, post-maturity interest; interest is out of scope for this document but noted here for context).

Business rule:

- * After re-aging, an existing N+1 installment is **kept only if its due date is strictly after the new maturity date** produced by the re-aged schedule.
- * If an N+1 installment's due date is **on or before** the new maturity date, it is **removed** and any underlying items are handled as part of the regenerated re-aged schedule.

Special Installment Handling

Special installments are created when **the re-aging transaction date** is before or on previous maturity date.

In this case:

- * Any installment occurring after the re-aging date that has a paid portion is **collected and converted into special installments** to preserve historical repayment accuracy.
- * Transactions already posted remain mapped to these special installments.
- * All general ledger entries remain unchanged — no reversals are introduced because the payments were already consumed by the loan.

This ensures:

- * Historical repayment activity is represented accurately.
- * Only unpaid obligations are restructured in the new re-aged repayment plan.

Developer's Note

Implementation is found in `AdvancedPaymentScheduleTransactionProcessor` Entry point for non-interest-bearing loans is `handleReAgeWithCommonStrategy`.

This implementation also contains solution for interest-bearing but not `EMICalculator` compatible loan schedules.

Implementation-wise it has the same logic, the only exception is the interest related calculations ends in zero amounts.

Core Logic steps:

- determine outstanding balances
- update transaction amounts to outstanding balances
- handle reversal if total amount is zero
- calculate `EqualAmortizationValues` for attributes recalculated by re-age
- handle charges if needed
- collect `paidInAdvanceBalances` - to move them into the special installment if required
- create special installment for early repaid balances.
- create first re-aged installment
 - try to merge or insert it depending on the existing schedule
 - add charge mapping
- create other re-aged installments if needed
 - try to merge or insert it depending on the existing schedule
 - add charge mapping

Note: merging the re-aged installments depends on the existing installment due-date and index.

Also, it keeps the charge mapping correct, in case charges (fees and penalties) should not be affected by re-age.

Interest Handling Scenarios

Scenario #1 — Equal Amortization

Core Logic

- Outstanding unpaid amounts (principal, interest, fees/penalties \leq transaction date) are **split equally** across the newly generated installments.
- No new interest accrues after re-age — interest recalculation and declining balance logic are **turned off**.

- User-selected interest scope determines which interest is included:
 - **Outstanding Payable Interest:** only interest due to date.
 - **Outstanding Full Interest:** all accrued interest, including future installments if applicable.

Outstanding Principal, Interest, Fees/Penalties

- **Principal:** only unpaid portion included in equal split.
- **Interest:** split according to user-selected interest scope.
- **Fees/Penalties:** unpaid portions with due-date \leq transaction date are included in the equal split; due-date $>$ transaction date remain mapped to original installments.

Special Collected Installment

Same to non-interest-bearing scenario

Schedule Generation & N+1 Installment Adjustment

- Re-aged installments start from the **user-provided start date**.
- The number of installments, period type, and frequency are **user-defined**.
- N+1 installment may be adjusted to balance rounding differences in principal/interest/fees.
- Fully paid installments before start date remain unchanged; partially paid installments retain the paid portion, with unpaid portion included in re-age.

GL / Transaction Recording

- A **non-monetary re-age transaction** is posted with amount = total outstanding principal + interest + fees/penalties included in equal split.
- Special collected installment GL postings match original repayments.
- Reversal of re-age removes non-monetary transaction and restores original schedule and mappings.

Developer's Note

Entrypoint:

`AdvancedPaymentScheduleTransactionProcessor:handleReAgeEqualAmortizationEMICalculator`

Core Logic steps:

- determine outstanding balances
- update transaction amounts to outstanding balances
- handle reversal if total amount is zero
- calculate EqualAmortizationValues for non progressive model handled attributes
- handle charges
- collect paidInAdvanceBalances - to move them into the special installment if required
- update model with EMICalculator - see corresponding steps in next chapter

- remove installments after transaction date
 - keep N+1 installments and update it's from date if it has a due date after the new maturity date, otherwise it is removed
- create special and re-aged installments according to model and update by EqualAmortizationValues calculated values
- update installments index

ProgressiveEMICalculator

Entry point: reAgeEqualAmortization

- **Determine original maturity and check if transaction is after it**
- **Calculate outstanding principal, interest for re-age**
- **Calculate already paid balances since transaction date**
 - Also sets moved credited principal & interest balances to handle not paid credited portions post re-age correctly which will be added back to fist re-age repayment period
- **Accelerate maturity date**
 - this step is actually modifies existing model by removing interest and repayment periods from re-age transaction date
- **Close existing repayment periods**
 - Marks **outstanding interest as moved due to re-aging**.
 - Sets EMI of each period to **total paid amount**.
 - Stops further unrecognised interest calculation.
- **Handle early repaid installments**
 - If any principal or interest has been paid beyond the transaction date:
 - Preserves paid amounts in a **special collected repayment period**.
- **Update model for re-aged periods**
 - Generates empty repayment periods according to the number of new installments.
- **Calculate EMI for new periods**
 - Splits principal, interest, and fees/penalties equally across new installments.
 - Ensures **rounding differences** are applied to the last installment.
- **Recalculate outstanding balance**
- **Add zero-interest period**
 - Ensures interest rate after re-age start is handled correctly.
 - If transaction is posted before original maturity date, we apply 0 interest from the transaction date
 - If transaction is posted after maturity date, we should add zero interest period from the original maturity date.

- **Calculate last unpaid repayment period EMI**
 - Ensures final installment balances the loan fully.

Scenario #2 — Default Behavior

Core Logic

- Overdue principal and charge-back principal are **re-aged** across new installments.
- Overdue interest and charge-back interest are moved **100% to the first new re-aged installment**.
- Future interest continues to accrue according to loan product rules (only past-due interest may be included in re-age calculation if applicable).
- Overdue fees and penalties are moved to the **first new re-aged installment**.
- Future fees and penalties remain on their original installments and are adjusted externally (not part of re-age logic).

Overdue vs Future Interest Handling

- Only **past-due interest** may be included in the re-age calculation.
- Overdue interest is **not split** — it goes entirely to the first re-aged installment.
- Future interest is calculated normally per the interest model.
- Charge-back interest follows the same rule: moved entirely to the first re-aged installment.

Overwriting Existing Future Installments

- Existing installments with due-date \geq start date are replaced by **re-aged installments**.
- New installments are generated according to user-provided **start date, number of installments, period type, and frequency**.
- If re-age reduces total installments, any installments beyond new maturity date are removed.
- N+1 installments are recalculated to balance principal, interest, and fees.

Special Collected Installment

Same to non-interest-bearing scenario

Fees/Penalties Handling

- Overdue fees and penalties are moved to the **first re-aged installment**.
- Fees/penalties with due-date $>$ start date remain on their original installments; mappings adjusted if installment numbering changes.
- Recalculation of future fees and penalties is handled outside re-age logic.

Schedule Generation & N+1 Installment Adjustment

- Schedule starts from **start date** provided by user.

- Number of installments, period type, and frequency are **user-defined**.
- Fully paid installments before start date remain unchanged.
- Partially paid installments: paid portion retained, unpaid portion included in re-aged principal.

Reversal / Backdated Transaction Handling

- Re-age can be reversed **only if it is the latest transaction**.
- Any backdated repayment or reversal triggers **reverse + replay**:
 - Removes special collected installment.
 - Restores original installment schedule.
 - Restores original charge/fee mappings and GL postings.
 - Re-applies re-age if necessary.

Schedule & Installment Mapping

Charge/Fee → Installment Mapping

- All fees and penalties are linked to specific installments in the schedule.
- After re-aging, the mapping must be updated to point fees/penalties to the correct new installment IDs.
- For fees/penalties with due-date > start date:
 - Preserve original due-date.
 - Adjust installment mapping only if installment IDs are shifted due to schedule regeneration.
- Overdue fees/penalties moved to first re-aged installment must have their mapping updated to the new installment ID.

Due Date Recalculation Rules

- Re-aged installments start from **user-provided start date**.
- Subsequent installment due-dates are calculated based on:
 - **Period Type**: Daily, Weekly, Monthly, Yearly
 - **Period Frequency**: number of periods between installments
- Fully paid installments before start date remain unchanged.
- Partial payments are preserved for principal and fees; only unpaid portions are re-aged.
- Any installment after the new maturity date is deleted if the re-age reduces total number of installments.

N+1 Installment Adjustment

- After generating re-aged installments, the **N+1 installment** is adjusted to balance rounding differences in principal, interest, and fees.
- Applies to all interest handling scenarios.

- Ensures total outstanding principal, interest, and fees are fully allocated across the new schedule.

Edge Cases

- Special collected installment:
 - Created if transaction date is before or on maturity date.
 - Merges all paid portions into a single installment.
 - GL postings must match original payments.
 - Does not participate in principal/interest redistribution.
- Reversal of re-age:
 - Only allowed if it is the latest transaction.
 - Removes special collected installment.
 - Restores original installment schedule, charge/fee mappings, and GL postings.
- Backdated re-age:
 - Transaction date = start date.
 - Reverse + replay logic applied to rebuild schedule correctly.

Validation Rules

Transaction Date Validation

- Transaction date cannot be before the start of the loan lifecycle.
- For backdated re-age, transaction date = start date.

Start Date Validation

- Start date cannot be earlier than the loan disbursement date.
- Start date drives the first installment of the re-aged schedule.

Re-Age Eligibility

- Re-aging is allowed only for Progressive loan strategy.
- Charge-off loan accounts are **not eligible** for re-aging.

Summary Comparison Table For Interest Handling Scenarios

Feature	Equal Amortization	Default Behavior	Notes
Outstanding Principal	Equally split across new installments	Re-aged according to schedule	Only unpaid portion is considered; paid portion preserved

Feature	Equal Amortization	Default Behavior	Notes
Outstanding Interest	Equally split (full or payable) among new installments	Overdue interest moved 100% to first re-aged installment; future interest continues normally	—
Charge-back Principal	Equally split	Re-aged according to schedule	—
Charge-back Interest	Equally split	Moved 100% to first re-aged installment	—
Overdue Fees / Penalties	Equally split	Moved to first re-aged installment	Future fees handled externally, not part of re-age
Special Collected Installment	Created if installments after transaction date are partially paid; due date = transaction date; GL postings preserved	Same as Equal Amortization	Not included in principal/interest redistribution
User Input Parameters	Start date, number of installments, period type/frequency, interest handling method	Same as Equal Amortization	Inputs drive new schedule generation

Fineract Development Environment

TBD

Git

TBD

GPG

TBD

Committers

Please make sure to provide your GPG fingerprint in your Apache committer profile at id.apache.org.

Docker

TBD

Docker Compose

TBD

Podman

TBD

Rancher Docker Desktop

TBD

Gradle

TBD

IDE

TBD

IntelliJ

TBD

Eclipse

TBD

VSCode

TBD

Kubernetes

TBD

Minikube

TBD

Microk8s

TBD

K3d

TBD

Helm Charts

TBD

Tools

TBD

SDKMAN

We recommend using SKDMAN to manage the following developer tools:

- JDK
- Spring Boot CLI
- Gradle (if you need a global installation)
- AsciiDoctorJ

TBD

Brew

MacOS

TBD

Linux

TBD

Custom Modules



Currently, modules are a proof of concept feature in Fineract.

Introduction

Creating customizations for Fineract services is easy. The method described here will work both with our future module guidelines (aka "clean room" modules) and with the intermediary solution we will put in place to avoid major refactorings.

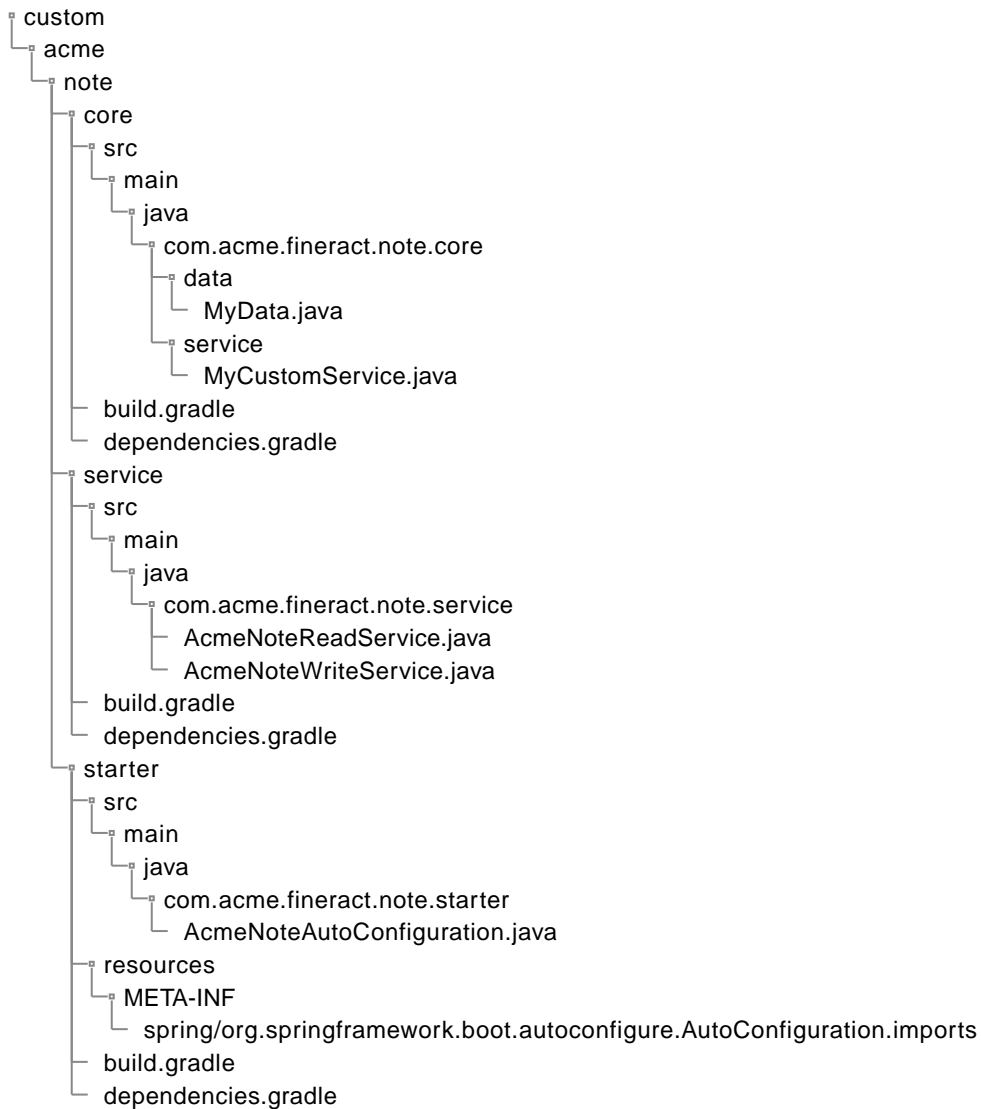
The folder structure for modules is based on a convention that ensures that your extensions don't clash with Fineract's internals. This is to make sure that your downstream forks of Fineract are easy to sync. In the past we had all kinds of strategies to add custom code - including editing existing sources in `fineract-provider`. This is not recommended.



At the moment the only service(s) we prepared to be overridden/replaced are `org.apache.fineract.portfolio.note.service.NoteReadPlatformService` and `org.apache.fineract.portfolio.note.service.NoteWritePlatformService`. Please reach out on the developer mailing list if you need other services.

The recommended folder structure is very simple. If you follow this recommendation you'll get some additional benefits, e. g. you don't even have to edit `settings.gradle` to include your new custom modules. Your modules will also be automatically included in a custom Fineract Docker image build that you can use for your production deployments.

Let's assume your company/org is called "ACME Inc." and you are trying to (fully/partially) replace an existing Fineract service, let's say those in `org.apache.fineract.portfolio.note`. The recommended folder structure would then look something like this:



As soon as we can publish Fineract module JARs to Maven Central you'll have more freedom to setup your projects (including to setup separate Git repos). But for now please follow these instructions:

1. Create a folder under **custom** and name it according to your company/organisation (e. g. **acme** if your company is **ACME Inc.**); this way your custom modules can't clash even with other companies' modules
2. Under your company folder create a folder for the **category** or **domain** your module is targeting; e. g. "loan", "client", "account" etc.
3. Finally, setup **library** folders for the actual modules you want to create; usually that will be to replace/extend some existing service, so there could be a **service** folder, maybe even a **core** folder, e. g. if you want to add additional DTOs etc.; we have also an example for COB business steps
4. Per **category/domain** you should have a **starter** library; means: a Spring Boot auto-configuration setup that makes including your module in Fineract easier ("hands-free"); the necessary parts for a auto-configuration library are a Spring Java configuration class (annotated with **@Configuration**) and a text file at **META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports** in your starter resource folder:

```
com.acme.fineract.portfolio.note.starter.AcmeNoteAutoConfiguration
```

Please make sure that your module libraries have proper `build.gradle` files:

```
description = 'ACME Fineract Note Service'

group = 'com.acme.fineract'

base {
    archivesName = 'acme-fineract-note-service'
}

apply from: 'dependencies.gradle'
```



You don't need to edit `settings.gradle` to add your modules/libraries. If you follow above convention they'll get included automatically.

5. The `dependency.gradle` file could look something like this:

```
dependencies {
    implementation(project(':fineract-core'))
    implementation(project(':fineract-provider'))
    compileOnly('org.springframework.boot:spring-boot-autoconfigure')
}
```



We've included by default some basic and useful dependencies for all custom modules, like Slf4j, Lombok, the usual testing frameworks (JUnit, Cucumber, Mockito etc.)



Do not include your custom module in `fineract-provider`'s `dependency.gradle` file. This creates a circular dependency and will fail your build.

Custom Services



We are still trying to figure out which internal services make most sense to be pluggable. Please join the discussion and let us know if you have a specific requirement.

Note Service

The Note service is responsible for ... TBD



We chose the note service because it's interface is very simple and has not many cross dependencies.

Interfaces

Note Read Service Interface

```
package org.apache.fineract.portfolio.note.service;

import java.util.List;
import org.apache.fineract.portfolio.note.data.NoteData;

public interface NoteReadPlatformService {

    NoteData retrieveNote(Long noteId, Long resourceId, Integer noteTypeId);

    List<NoteData> retrieveNotesByResource(Long resourceId, Integer noteTypeId);
}
```

Note Write Service Interface

```
package org.apache.fineract.portfolio.note.service;

import java.util.List;
import org.apache.fineract.portfolio.note.data.NoteData;

public interface NoteReadPlatformService {

    NoteData retrieveNote(Long noteId, Long resourceId, Integer noteTypeId);

    List<NoteData> retrieveNotesByResource(Long resourceId, Integer noteTypeId);
}
```

Auto Start Configuration

The rules to replace the Note services are very simple. If you provide an alternative implementation of the services then the default implementations will **not** be loaded.

Note Auto Starter Configuration

```
package org.apache.fineract.portfolio.note.starter;

import org.apache.fineract.portfolio.client.domain.ClientRepositoryWrapper;
import org.apache.fineract.portfolio.group.domain.GroupRepository;
import org.apache.fineract.portfolio.loanaccount.domain.LoanRepositoryWrapper;
import org.apache.fineract.portfolio.loanaccount.domain.LoanTransactionRepository;
import org.apache.fineract.portfolio.note.domain.NoteRepository;
import org.apache.fineract.portfolio.note.serialization.NoteCommandFromApiJsonDeserializer;
import org.apache.fineract.portfolio.note.service.NoteReadPlatformService;
import org.apache.fineract.portfolio.note.service.NoteReadPlatformServiceImpl;
import org.apache.fineract.portfolio.note.service.NoteWritePlatformService;
import
```

```

org.apache.fineract.portfolio.note.service.NoteWritePlatformServiceJpaRepositoryImpl;
import org.apache.fineract.portfolio.savings.domain.SavingsAccountRepository;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;

@Configuration
public class NoteAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public NoteReadPlatformService noteReadPlatformService(JdbcTemplate jdbcTemplate)
    {
        return new NoteReadPlatformServiceImpl(jdbcTemplate);
    }

    @Bean
    @ConditionalOnMissingBean
    public NoteWritePlatformService noteWritePlatformService(NoteRepository
noteRepository, ClientRepositoryWrapper clientRepository,
        GroupRepository groupRepository, LoanRepositoryWrapper loanRepository,
LoanTransactionRepository loanTransactionRepository,
        NoteCommandFromApiJsonDeserializer fromApiJsonDeserializer,
SavingsAccountRepository savingsAccountRepository) {
        return new NoteWritePlatformServiceJpaRepositoryImpl(noteRepository,
clientRepository, groupRepository, loanRepository,
        loanTransactionRepository, fromApiJsonDeserializer,
savingsAccountRepository);
    }
}

```

Custom Business Steps

It is very easy to add your own business steps to Fineract's default steps:

1. Create a custom module (e. g. `custom/acme/steps`, follow the instructions on how to create a custom module)
2. Create a class that implements interface `org.apache.fineract.cob.COBBusinessStep`
3. Provide the custom database migration to add the necessary information about your business step in table `m_batch_business_steps`

Business Step Interface

```

package org.apache.fineract.cob;

import org.apache.fineract.infrastructure.core.domain.AbstractPersistableCustom;

public interface COBBusinessStep<T> extends AbstractPersistableCustom<Long>> {

```

```

    T execute(T input);

    String getEnumStyledName();

    String getHumanReadableName();
}

```

Business Step Implementation

Custom Business Step Implementation Example

```

package com.acme.fineract.loan.cob;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.fineract.cob.loan.LoanCOBBusinessStep;
import org.apache.fineract.portfolio.loanaccount.domain.Loan;
import org.apache.fineract.portfolio.loanaccount.domain.LoanAccountDomainService;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.stereotype.Component;

@Slf4j
@Component
@RequiredArgsConstructor
public class AcmeNoopBusinessStep implements LoanCOBBusinessStep, InitializingBean {

    private static final String ENUM_STYLED_NAME = "ACME_LOAN_NOOP";

    private static final String HUMAN_READABLE_NAME = "ACME Loan Noop";

    // NOTE: just to demonstrate that dependency injection is working
    private final LoanAccountDomainService loanAccountDomainService;

    @Override
    public void afterPropertiesSet() throws Exception {
        log.warn("Acme COB Loan: '{}'", getClass().getCanonicalName());
    }

    @Override
    public Loan execute(Loan input) {
        return input;
    }

    @Override
    public String getEnumStyledName() {
        return ENUM_STYLED_NAME;
    }

    @Override

```

```

public String getHumanReadableName() {
    return HUMAN_READABLE_NAME;
}
}

```

As you can see this implementation is very simple and doesn't do much. There are some simple conventions though that you should follow implementing your own business steps:

1. Make sure the value returned by method `getEnumStyledName()` is unique; it's a good idea to choose a prefix that reflects the name of your organization (in this example `ACME_`)
2. You have more freedom for the value returned by `getHumanReadableName()`, but it's a good idea to keep this value as unique as possible

Business Step Database Migration

Business Step Database Migration Example

```

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd">
    <changeSet author="acme" id="1">
        <insert tableName="m_batch_business_steps">
            <column name="job_name" value="LOAN_CLOSE_OF_BUSINESS"/>
            <column name="step_name" value="ACME_LOAN_NOOP"/>
            <column name="step_order" value="5"/>
        </insert>
    </changeSet>
</databaseChangeLog>

```



See also chapter about batch jobs in this documentation.

Custom Loan Transaction Processors

Fineract has 7 built-in loan transaction processors:

1. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.CreocoreLoanRepaymentScheduleTransactionProcessor`
2. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.EarlyPaymentLoanRepaymentScheduleTransactionProcessor`
3. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.FineractStyleLoanRepaymentScheduleTransactionProcessor`
4. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.HeavensFamilyLoanRepaymentScheduleTransactionProcessor`
5. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.InterestPrincipalPenaltyFeesOrderLoanRepaymentScheduleTransactionProcessor`

6. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.PrincipalInterestPenaltyFeesOrderLoanRepaymentScheduleTransactionProcessor`
7. `org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.RBILoanRepaymentScheduleTransactionProcessor`

Default Loan Transaction Processor configuration

```
import org.apache.fineract.portfolio.loanaccount.service.LoanChargeService;
import
org.apache.fineract.portfolio.loanaccount.service.ProgressiveLoanInterestRefundService
Impl;
import
org.apache.fineract.portfolio.loanaccount.service.schedule.LoanScheduleComponent;
import org.apache.fineract.portfolio.loanproduct.calc.EMICalculator;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Conditional;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Lazy;

@Configuration
public class LoanAccountAutoStarter {

    @Bean
    @Conditional(CreocoreLoanRepaymentScheduleTransactionProcessorCondition.class)
    public CreocoreLoanRepaymentScheduleTransactionProcessor
creocoreLoanRepaymentScheduleTransactionProcessor(
        final ExternalIdFactory externalIdFactory, final LoanChargeValidator
loanChargeValidator,
        final LoanBalanceService loanBalanceService) {
        return new CreocoreLoanRepaymentScheduleTransactionProcessor(
externalIdFactory, loanChargeValidator, loanBalanceService);
    }

    @Bean
    @Conditional(EarlyRepaymentLoanRepaymentScheduleTransactionProcessorCondition
.class)
    public EarlyPaymentLoanRepaymentScheduleTransactionProcessor
earlyPaymentLoanRepaymentScheduleTransactionProcessor(
        final ExternalIdFactory externalIdFactory, final LoanChargeValidator
loanChargeValidator,
        final LoanBalanceService loanBalanceService) {
        return new EarlyPaymentLoanRepaymentScheduleTransactionProcessor
(externalIdFactory, loanChargeValidator, loanBalanceService);
    }

    @Bean
    @Conditional(MifosStandardLoanRepaymentScheduleTransactionProcessorCondition.
class)
    public FineractStyleLoanRepaymentScheduleTransactionProcessor
```

```

fineractStyleLoanRepaymentScheduleTransactionProcessor(
    final ExternalIdFactory externalIdFactory, final LoanChargeValidator
loanChargeValidator,
    final LoanBalanceService loanBalanceService) {
    return new FineractStyleLoanRepaymentScheduleTransactionProcessor
(externalIdFactory, loanChargeValidator, loanBalanceService);
}

@Bean
@Conditional(HeavensFamilyLoanRepaymentScheduleTransactionProcessorCondition.
class)
public HeavensFamilyLoanRepaymentScheduleTransactionProcessor
heavensFamilyLoanRepaymentScheduleTransactionProcessor(
    final ExternalIdFactory externalIdFactory, final LoanChargeValidator
loanChargeValidator,
    final LoanBalanceService loanBalanceService) {
    return new HeavensFamilyLoanRepaymentScheduleTransactionProcessor
(externalIdFactory, loanChargeValidator, loanBalanceService);
}

```

All default processor implementations are enabled by default, but can also be prevented from being loaded into memory by a simple configuration in `application.properties`. Use the environment variables you see below in your Kubernetes and Docker Compose deployments to override the default behavior.

Default Loan Transaction Processor Application Properties

```

fineract.tenant.config.max-pool-size=${FINERACT_CONFIG_MAX_POOL_SIZE:-1}
fineract.tenant.config.rounding-mode=${FINERACT_CONFIG_ROUNDING_MODE:6}

fineract.mode.read-enabled=${FINERACT_MODE_READ_ENABLED:true}
fineract.mode.write-enabled=${FINERACT_MODE_WRITE_ENABLED:true}
fineract.mode.batch-worker-enabled=${FINERACT_MODE_BATCH_WORKER_ENABLED:true}
fineract.mode.batch-manager-enabled=${FINERACT_MODE_BATCH_MANAGER_ENABLED:true}

```

Implement Processors

Loan Transaction Processor Interface

```

package org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor;

import java.time.LocalDate;
import java.util.List;
import java.util.Set;
import org.apache.fineract.organisation.monetary.domain.MonetaryCurrency;
import org.apache.fineract.organisation.monetary.domain.Money;
import org.apache.fineract.portfolio.loanaccount.domain.ChangedTransactionDetail;
import org.apache.fineract.portfolio.loanaccount.domain.LoanCharge;
import
org.apache.fineract.portfolio.loanaccount.domain.LoanRepaymentScheduleInstallment;

```

```

import org.apache.fineract.portfolio.loanaccount.domain.LoanTransaction;

public interface LoanRepaymentScheduleTransactionProcessor {

    String getCode();

    String getName();

    boolean accept(String s);

    /**
     * Provides support for processing the latest transaction (which should be the
     * latest transaction) against the loan
     * schedule.
     *
     * @return ChangedTransactionDetail
     */
    ChangedTransactionDetail processLatestTransaction(LoanTransaction loanTransaction,
    TransactionCtx ctx);

    /**
     * Provides support for passing all {@link LoanTransaction}'s so it will
     * completely re-process the entire loan
     * schedule. This is required in cases where the {@link LoanTransaction} being
     * processed is in the past and falls
     * before existing transactions or and adjustment is made to an existing in which
     * case the entire loan schedule
     * needs to be re-processed.
     */
    ChangedTransactionDetail reprocessLoanTransactions(LocalDate disbursementDate,
    List<LoanTransaction> repaymentsOrWaivers,
        MonetaryCurrency currency, List<LoanRepaymentScheduleInstallment>
    repaymentScheduleInstallments, Set<LoanCharge> charges);

    Money handleRepaymentSchedule(List<LoanTransaction> transactionsPostDisbursement,
    MonetaryCurrency currency,
        List<LoanRepaymentScheduleInstallment> installments, Set<LoanCharge>
    loanCharges);

    /**
     * Used in interest recalculation to introduce new interest only installment.
     */
    boolean isInterestFirstRepaymentScheduleTransactionProcessor();
}

```

Custom Loan Transaction Processor Example

```

package com.acme.fineract.loan.processor;

import org.apache.fineract.infrastructure.core.service.ExternalIdFactory;

```

```

import
org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.FineractStyleLoanRepaymentScheduleTransactionProcessor;
import org.apache.fineract.portfolio.loanaccount.serialization.LoanChargeValidator;
import org.apache.fineract.portfolio.loanaccount.service.LoanBalanceService;
import org.springframework.stereotype.Component;

@Component
public class AcmeLoanRepaymentScheduleTransactionProcessor extends
FineractStyleLoanRepaymentScheduleTransactionProcessor {

    public static final String STRATEGY_CODE = "acme-standard-strategy";

    public static final String STRATEGY_NAME = "ACME Corp.: standard loan transaction
processing strategy";

    public AcmeLoanRepaymentScheduleTransactionProcessor(final ExternalIdFactory
externalIdFactory,
        final LoanChargeValidator loanChargeValidator, final LoanBalanceService
loanBalanceService) {
        super(externalIdFactory, loanChargeValidator, loanBalanceService);
    }

    @Override
    public String getCode() {
        return STRATEGY_CODE;
    }

    @Override
    public String getName() {
        return STRATEGY_NAME;
    }
}

```

The example implementation doesn't do much. We are just overriding one of the default processor implementations

`org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.impl.FineractStyleLoanRepaymentScheduleTransactionProcessor` and give the custom processor its own lookup code and name (descriptive text for display in UIs, e. g. when configuring a loan product). As usual it is a good idea to follow some simple conventions:

1. Make sure the value returned by `getCode()` is unique. Prefixing it with characters that reflect your organization name (here `acme-`) is a good idea.
2. You have more freedom for the descriptive text returned by `getName()`, but it is still a good idea to keep the value unique to avoid confusion.

Method `getCode()`

Lookup value that is used to pick a loan transaction processor (see processor factory).

Method `getName()`

Descriptive text about the loan transaction processor that is mostly used in user interfaces.

Method `handleTransaction()`

TBD

Method `handleWriteOff()`

TBD

Method `handleRepaymentSchedule()`

TBD

Method `isInterestFirstRepaymentScheduleTransactionProcessor()`

TBD

Method `handleRefund()`

TBD

Method `handleChargeback()`

TBD

Method `processTransactionsFromDerivedFields()`

TBD

Override Processor Factory

The processor factory has no reference to any specific implementation of the loan transaction processor interface. All available implementations will be injected here (internal default and custom implementations). Processor instances can be looked up via method `determineProcessor()`. You can pass either the code of the processor or the processor's name to look it up. If a matching processor can't be found then the factory function will either return the default instance or fails with an exception depending on the configuration in `application.properties`.



It is preferable to use the processor code to lookup processor instances. Lookups via processor names are only done in the import service via Excel sheets (should be fixed).

Loan Transaction Processor Factory Implementation

```
package org.apache.fineract.portfolio.loanaccount.domain;

import java.util.List;
import java.util.Optional;
```

```

import lombok.RequiredArgsConstructor;
import
org.apache.fineract.portfolio.loanaccount.domain.transactionprocessor.LoanRepaymentSch
eduleTransactionProcessor;
import
org.apache.fineract.portfolio.loanaccount.exception.LoanTransactionProcessingStrategyN
otFoundException;
import
org.apache.fineract.portfolio.loanproduct.data.TransactionProcessingStrategyData;
import org.springframework.beans.factory.annotation.Value;

@RequiredArgsConstructor
public class LoanRepaymentScheduleTransactionProcessorFactory {

    private final LoanRepaymentScheduleTransactionProcessor
defaultLoanRepaymentScheduleTransactionProcessor;

    private final List<LoanRepaymentScheduleTransactionProcessor> processors;

    @Value("${fineract.loan.transactionprocessor.error-not-found-fail}")
    private Boolean errorNotFoundFail;

    public LoanRepaymentScheduleTransactionProcessor determineProcessor(final String
transactionProcessingStrategy) {

        Optional<LoanRepaymentScheduleTransactionProcessor> processor = processors
.stream()
            .filter(p -> p.accept(transactionProcessingStrategy)).findFirst();

        if (processor.isEmpty() && Boolean.TRUE.equals(errorNotFoundFail)) {
            throw new LoanTransactionProcessingStrategyNotFoundException
(transactionProcessingStrategy);
        } else {
            return processor.orElse(defaultLoanRepaymentScheduleTransactionProcessor);
        }
    }

    public List<TransactionProcessingStrategyData> getStrategies() {
        return processors.stream().map(p -> new TransactionProcessingStrategyData(
null, p.getCode(), p.getName())).toList();
    }
}

```

This is the default factory auto-configuration.

Loan Transaction Processor Factory Auto-Configuration

```

}

@Bean
@Conditional

```

```
(InterestPrincipalPenaltiesFeesLoanRepaymentScheduleTransactionProcessorCondition.class)

public InterestPrincipalPenaltyFeesOrderLoanRepaymentScheduleTransactionProcessor
interestPrincipalPenaltyFeesOrderLoanRepaymentScheduleTransactionProcessor(
    final ExternalIdFactory externalIdFactory, final LoanChargeValidator
    loanChargeValidator,
    final LoanBalanceService loanBalanceService) {
```

If you need then you can override this, e.g. because you want to set a different default processor then you can do so in your custom module's auto-configuration.

Custom Loan Transaction Processor Factory Auto-Configuration Example

```
@Bean
public LoanRepaymentScheduleTransactionProcessorFactory
loanRepaymentScheduleTransactionProcessorFactory(
    AcmeLoanRepaymentScheduleTransactionProcessor
    defaultLoanRepaymentScheduleTransactionProcessor,
    List<LoanRepaymentScheduleTransactionProcessor> processors) {
    return new LoanRepaymentScheduleTransactionProcessorFactory
    (defaultLoanRepaymentScheduleTransactionProcessor, processors);
```

Processor Lookup Failure Configuration Property

Custom Batch Jobs

Fineract provides extension points to define custom batch jobs using module system. Using this approach custom batch jobs can be defined and configured along with Fineract's default batch jobs to extend or customize batch processing.

The batch jobs in Fineract are implemented using [Spring Batch](#). In addition to the Spring Batch ecosystem, automatic scheduling is done by [Quartz Scheduler](#) but it's also possible to trigger batch jobs via regular APIs.

For defining custom job:

1. Create custom module (e. g. `custom/acme/loan/job`), follow the instructions on how to create a custom module.
2. Create job configuration to register job, job steps, tasklet with job builder factory. (e. g. `com.acme.fineract.loan.job.AcmeNoopJobConfiguration`)
3. Create tasklet for job execution functionality. (e.g. `com.acme.fineract.loan.job.AcmeNoopJobTasklet`)
4. Provide the custom database migration to add necessary information about your job in table `job`. (e.g. `custom/acme/loan/job/src/main/resources/db/custom-changelog/0001_acme_loan_job.xml`)
5. New job name should be registered along with default jobs so that it can be scheduled at

startup. For registering job name with Fineract job scheduler, create an enum with job name details (e.g. `com.acme.fineract.loan.job.AcmeJobName`) and a job name provider configuration which is accessed by Fineract job scheduler at startup to retrieve job name (e.g. `com.acme.fineract.loan.job.AcmeJobNameConfig`).

Job Configuration

Job Configuration Example

```
package com.acme.fineract.loan.job;

import lombok.RequiredArgsConstructor;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.job.builder.JobBuilder;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.repository.JobRepository;
import org.springframework.batch.core.step.builder.StepBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.PlatformTransactionManager;

@Configuration
@RequiredArgsConstructor
public class AcmeNoopJobConfiguration {

    private final JobRepository jobRepository;
    private final PlatformTransactionManager transactionManager;
    private final AcmeNoopJobTasklet tasklet;

    @Bean
    protected Step acmeNoopJobStep() {
        return new StepBuilder(AcmeJobName.ACME_NOOP_JOB.name(), jobRepository)
            .tasklet(tasklet, transactionManager).build();
    }

    @Bean
    public Job acmeNoopJob() {
        return new JobBuilder(AcmeJobName.ACME_NOOP_JOB.name(), jobRepository).start(
            acmeNoopJobStep()).incrementer(new RunIdIncrementer())
            .build();
    }
}
```

Tasklet Definition

Job Tasklet Example

```
package com.acme.fineract.loan.job;
```

```

import lombok.extern.slf4j.Slf4j;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.stereotype.Component;

@Slf4j
@Component
public class AcmeNoopJobTasklet implements Tasklet {

    @Override
    public RepeatStatus execute(StepContribution contribution, ChunkContext
chunkContext) throws Exception {
        log.info("Acme custom job execution");
        return RepeatStatus.FINISHED;
    }
}

```

Database Migration Script for Job

Database Migration Script Example

```

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd">
    <changeSet author="acme" id="1">
        <insert tableName="job">
            <column name="name" value="Acme Noop Job"/>
            <column name="display_name" value="Acme Noop Job"/>
            <column name="cron_expression" value="0 1 0 1/1 * ? *"/>
            <column name="create_time" valueDate="${current_datetime}"/>
            <column name="task_priority" valueNumeric="5"/>
            <column name="group_name"/>
            <column name="previous_run_start_time"/>
            <column name="job_key" value="Acme Noop Job _ DEFAULT"/>
            <column name="initializing_errorlog"/>
            <column name="is_active" valueBoolean="false"/>
            <column name="currently_running" valueBoolean="false"/>
            <column name="updates_allowed" valueBoolean="true"/>
            <column name="scheduler_group" valueNumeric="0"/>
            <column name="is_misfired" valueBoolean="false"/>
            <column name="node_id" valueNumeric="1"/>
            <column name="is_mismatched_job" valueBoolean="true"/>
        </insert>
    </changeSet>
    <changeSet author="acme" id="2">
        <update tableName="job">

```

```

        <column name="short_name" value="ACM_NOOP"/>
        <where>name='Acme Noop Job'</where>
    </update>
</changeSet>
</databaseChangeLog>

```

Job Name Configuration

Job Name Enum Example

```

package com.acme.fineract.loan.job;

public enum AcmeJobName {

    ACME_NOOP_JOB("Acme Noop Job"); //

    private final String name;

    AcmeJobName(final String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return this.name;
    }
}

```

Job Name Provider Configuration Example

```

package com.acme.fineract.loan.job;

import java.util.List;
import org.apache.fineract.infrastructure.jobs.service.jobname.JobNameData;
import org.apache.fineract.infrastructure.jobs.service.jobname.JobNameProvider;
import org.apache.fineract.infrastructure.jobs.service.jobname.SimpleJobNameProvider;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AcmeJobNameConfig {

    @Bean
    public JobNameProvider acmeJobNameProvider() {
        return new SimpleJobNameProvider(List.of(new JobNameData(AcmeJobName
.ACME_NOOP_JOB.name(), AcmeJobName.ACME_NOOP_JOB.toString())));
    }
}

```

Gradle Build Files

Please make sure that your module libraries have proper `build.gradle` and `dependencies.gradle` files:

Example (build.gradle)

```
description = 'ACME Fineract Loan Job'

group = 'com.acme.fineract'

base {
    archivesName = 'acme-fineract-loan-job'
}

apply from: 'dependencies.gradle'
```

Example (dependencies.gradle)

```
dependencies {
    implementation(project(':fineract-core'))
    implementation(project(':fineract-loan'))
    implementation(project(':fineract-provider'))
    implementation('org.springframework.batch:spring-batch-integration')
    implementation('org.springframework.boot:spring-boot-starter-data-jpa')
}
```

Deployment

Custom modules can be deployed using docker image. See chapter about deploying custom modules in this documentation.

Example command to build docker image

```
./gradlew :custom:docker:jibDockerBuild
```



See also chapter about batch jobs in this documentation.

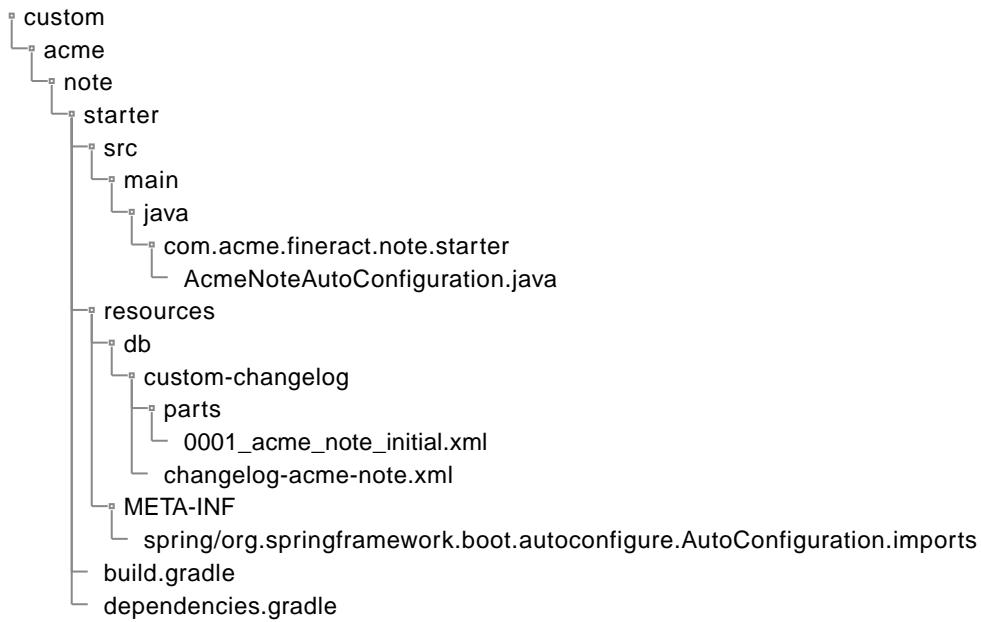
Custom Database Migration

If database migrations are needed as part of your customizations then you can add your own migration scripts. This is again based on conventions:

1. Create folders `db/custom-changelog` in one of your `resources` folders; we recommend using the resources folder in your starter library, but actually any of your custom libs will do.
2. Under `db/custom-changelog` create an XML changelog file, e. g. `changelog-acme-note.xml`; you are free to choose a name for this file, but we recommend being consistent to avoid classpath

conflicts.

3. Under `db/custom-changelog` create a folder `parts` for your specific changelogs



And here an example migration script:

```
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd">
  <changeSet author="acme" id="1">
    <createTable tableName="acme_note_dummy">
      <column autoIncrement="true" name="id" type="BIGINT">
        <constraints nullable="false" primaryKey="true"/>
      </column>
      <column name="name" type="VARCHAR(100)">
        <constraints unique="true"/>
      </column>
      <column name="description" type="VARCHAR(500)"/>
    </createTable>
  </changeSet>
</databaseChangeLog>
```



By default, custom database migration changelogs are executed in context `tenant_db`. That makes sure your changes will be applied to the tenant database (read: main database and not the tenant store database). In theory you could also target the tenant configuration database, but it's not recommended to do that.

Deploying Custom Modules

Custom modules (better: the JAR files) only need to be dropped in Fineract's `libs` folder if you run Fineract from the Spring Boot JAR file. Dynamic loading of external JARs is provided since Fineract

version 1.5.0. For your convenience we've created a separate Docker image module that automatically includes your custom modules (see [custom/docker](#)). You can build this Docker image with

```
./gradlew :custom:docker:jibDockerBuild
```

The Docker image with included custom modules is called [fineract-custom](#).



We'll provide soon a way to customize the Docker image parameters (image name, JVM implementation, JVM args, ports etc.).

Outlook

If this **proof of concept** is accepted we could prepare more of Fineract's internal services to be replaceable. This approach works already very well even if we don't have proper JAR libraries published on Maven Central. It's an important goal to separate customized code from Fineract's internals to have soon real modules.

Resilience

Introduction Resilience

Fineract had handcrafted retry loops in place for the longest time. A typical retry code would have looked like this:

Legacy retry code

```
@Override
@SuppressWarnings("AvoidHidingCauseException")
@SuppressFBWarnings(value = {
    "DMI_RANDOM_USED_ONLY_ONCE" }, justification = "False positive for random
object created and used only once")
public CommandProcessingResult logCommandSource(final CommandWrapper wrapper) {

    boolean isApprovedByChecker = false;
    // check if is update of own account details
    if (wrapper.isUpdateOfOwnUserDetails(this.context.authenticatedUser(wrapper
).getId())) {
        // then allow this operation to proceed.
        // maker checker doesn't mean anything here.
        isApprovedByChecker = true; // set to true in case permissions have
        // been maker-checker enabled by
        // accident.
    } else {
        // if not user changing their own details - check user has
        // permission to perform specific task.
        this.context.authenticatedUser(wrapper).validateHasPermissionTo(wrapper
.getTaskPermissionName());
    }
    validateIsUpdateAllowed();

    final String json = wrapper.getJson();
    CommandProcessingResult result = null;
    JsonCommand command;
    int numberOfRetries = 0; ①
    int maxNumberOfRetries = ThreadLocalContextUtil.getTenant().getConnection
().getMaxRetriesOnDeadlock();
    int maxIntervalBetweenRetries = ThreadLocalContextUtil.getTenant
().getConnection().getMaxIntervalBetweenRetries();
    final JsonElement parsedCommand = this.fromApiJsonHelper.parse(json);
    command = JsonCommand.from(json, parsedCommand, this.fromApiJsonHelper,
wrapper.getEntityName(), wrapper.getEntityId(),
    wrapper.getSubentityId(), wrapper.getGroupId(), wrapper.getClientId(),
wrapper.getLoanId(), wrapper.getSavingsId(),
    wrapper.getTransactionId(), wrapper.getHref(), wrapper.getProductId(),
wrapper.getCreditBureauId(),
    wrapper.getOrganisationCreditBureauId(), wrapper.getJobName());
    while (numberOfRetries <= maxNumberOfRetries) { ②
```

```

        try {
            result = this.processAndLogCommandService.executeCommand(wrapper,
                command, isApprovedByChecker);
            numberOfRetries = maxNumberOfRetries + 1; ③
        } catch (CannotAcquireLockException |
ObjectOptimisticLockingFailureException exception) {
            log.debug("The following command {} has been retried {} time(s)",
                command.json(), numberOfRetries);
            /**
             * Fail if the transaction has been retired for maxNumberOfRetries
             */
            if (numberOfRetries >= maxNumberOfRetries) {
                log.warn("The following command {} has been retried for the max
allowed attempts of {} and will be rolled back",
                    command.json(), numberOfRetries);
                throw exception;
            }
            /**
             * Else sleep for a random time (between 1 to 10 seconds) and continue
             */
            try {
                int randomNum = RANDOM.nextInt(maxIntervalBetweenRetries + 1);
                Thread.sleep(1000 + (randomNum * 1000));
                numberOfRetries = numberOfRetries + 1; ④
            } catch (InterruptedException e) {
                throw exception;
            }
        } catch (final RollbackTransactionAsCommandIsNotApprovedByCheckerException
e) {
            numberOfRetries = maxNumberOfRetries + 1; ③
            result = this.processAndLogCommandService.logCommand(e
                .getCommandSourceResult());
        }

        return result;
    }
}

```

① counter

② while loop

③ increment to abort

④ increment

For better code quality and readability we introduced [Resilience4j](#):

Annotation based retry

```

private final CommandProcessingService processAndLogCommandService;
private final SchedulerJobRunnerReadService schedulerJobRunnerReadService;
private final ConfigurationDomainService configurationService;

```

```

@Override
public CommandProcessingResult logCommandSource(final CommandWrapper wrapper) {
    boolean isApprovedByChecker = false;

    // check if is update of own account details
    if (wrapper.isChangeOfOwnUserDetails(this.context.authenticatedUser(wrapper
).getId())) {
        // then allow this operation to proceed.
        // maker checker doesnt mean anything here.
        isApprovedByChecker = true; // set to true in case permissions have
        // been maker-checker enabled by
        // accident.
    } else {
        // if not user changing their own details - check user has
        // permission to perform specific task.
        this.context.authenticatedUser(wrapper).validateHasPermissionTo(wrapper
.getTaskPermissionName());
    }
    validateIsUpdateAllowed();

    final String json = wrapper.getJson();
    final JsonElement parsedCommand = this.fromApiJsonHelper.parse(json);
    JsonCommand command = JsonCommand.from(json, parsedCommand, this
.fromApiJsonHelper, wrapper.getEntityName(), wrapper.getEntityId(),
        wrapper.getSubentityId(), wrapper.getGroupId(), wrapper.getClientId(),
wrapper.getLoanId(), wrapper.getSavingsId(),
        wrapper.getTransactionId(), wrapper.getHref(), wrapper.getProductId(),
wrapper.getCreditBureauId(),
        wrapper.getOrganisationCreditBureauId(), wrapper.getJobName(),
wrapper.getLoanExternalId());

```

Command

CommandProcessingService

TBD

Retry-able service function executeCommand

```

public static final String IDEMPOTENCY_KEY_ATTRIBUTE = "IdempotencyKeyAttribute";
public static final String COMMAND_SOURCE_ID = "commandSourceId";
private final PlatformSecurityContext context;
private final ApplicationContext applicationContext;
private final ToApiJsonSerializer<Map<String, Object>> toApiJsonSerializer;
private final ToApiJsonSerializer<CommandProcessingResult>
toApiResultJsonSerializer;
private final ConfigurationDomainService configurationDomainService;
private final CommandHandlerProvider commandHandlerProvider;
private final IdempotencyKeyResolver idempotencyKeyResolver;

```

```

private final CommandSourceService commandSourceService;
private final RetryConfigurationAssembler retryConfigurationAssembler;

private final FineractRequestContextHolder fineractRequestContextHolder;
private final Gson gson = GoogleGsonSerializerHelper.createSimpleGson();

private CommandProcessingResult retryWrapper(Supplier<CommandProcessingResult>
supplier) {
    try {
        if (!BatchRequestContextHolder.isEnclosingTransaction()) {
            return retryConfigurationAssembler
.getRetryConfigurationForExecuteCommand().executeSupplier(supplier);
        }
        return supplier.get();
    } catch (RuntimeException e) {
        return fallbackExecuteCommand(e);
    }
}

@Override
public CommandProcessingResult executeCommand(final CommandWrapper wrapper, final
JsonCommand command,
        final boolean isApprovedByChecker) {
    return retryWrapper(() -> {
        // Do not store the idempotency key because of the exception handling
        setIdempotencyKeyStoreFlag(false);

        Long commandId = (Long) fineractRequestContextHolder.getAttribute
(COMMAND_SOURCE_ID, null);
        boolean isRetry = commandId != null;
        boolean isEnclosingTransaction = BatchRequestContextHolder
.isEnclosingTransaction();

        CommandSource commandSource = null;
        String idempotencyKey;
        if (isRetry) {
            commandSource = commandSourceService.getCommandSource(commandId);
            idempotencyKey = commandSource.getIdempotencyKey();
        } else if ((commandId = command.commandId()) != null) { // action on the
command itself
            commandSource = commandSourceService.getCommandSource(commandId);
            idempotencyKey = commandSource.getIdempotencyKey();
        } else {
            idempotencyKey = idempotencyKeyResolver.resolve(wrapper);
        }
        exceptionWhenTheRequestAlreadyProcessed(wrapper, idempotencyKey, isRetry);

        AppUser user = context.authenticatedUser(wrapper);
        if (commandSource == null) {
            if (isEnclosingTransaction) {
                commandSource = commandSourceService.getInitialCommandSource

```

```

(wrapper, command, user, idempotencyKey);
        } else {
            commandSource = commandSourceService.saveInitialNewTransaction
(wrapper, command, user, idempotencyKey);
            commandId = commandSource.getId();
        }
    }
    if (commandId != null) {
        storeCommandIdInContext(commandSource); // Store command id as a
request attribute
    }

    setIdempotencyKeyStoreFlag(true);

    return executeCommand(wrapper, command, isApprovedByChecker,
commandSource, user, isEnclosingTransaction);
});
}

private CommandProcessingResult executeCommand(final CommandWrapper wrapper, final
JsonCommand command,
        final boolean isApprovedByChecker, CommandSource commandSource, AppUser
user, boolean isEnclosingTransaction) {

    final CommandProcessingResult result;
    try {
        result = commandSourceService.processCommand(findCommandHandler(wrapper),
command, commandSource, user, isApprovedByChecker);
    } catch (Throwable t) { // NOSONAR
        RuntimeException mappable = ErrorHandler.getMappable(t);
        ErrorInfo errorInfo = commandSourceService.generateErrorInfo(mappable);
        Integer statusCode = errorInfo.getStatusCode();
    }
}

```

Fallback function `fallbackExecuteCommand`

```

    Retry persistenceRetry = retryConfigurationAssembler
.getRetryConfigurationForCommandResultPersistence();

    try {
        CommandSource finalCommandSource = commandSource;
        AtomicInteger attemptNumber = new AtomicInteger(0);
        CommandSource savedCommandSource = persistenceRetry.executeSupplier(() ->
{
            // Critical: Refetch on retry attempts (not on first attempt)
            CommandSource currentSource = finalCommandSource;
            attemptNumber.getAndIncrement();
        }
    }
}

```

Retry configuration for `executeCommand`

```
fineract.loan.transactionprocessor.interest-principal-penalties-fees.enabled
```

```

=${FINERACT_LOAN_TRANSACTIONPROCESSOR_INTEREST_PRINCIPAL_PENALTIES_FEES_ENABLED:true}
fineract.loan.transactionprocessor.principal-interest-penalties-fees.enabled
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_PRINCIPAL_INTEREST_PENALTIES_FEES_ENABLED:true}
fineract.loan.transactionprocessor.rbi-india.enabled
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_RBI_INDIA_ENABLED:true}
fineract.loan.transactionprocessor.due-penalty-fee-interest-principal-in-advance-
principal-penalty-fee-interest.enabled
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_DUE_PENALTY_FEE_INTEREST_PRINCIPAL_IN_ADVANCE_PR
INCIPAL_PENALTY_FEE_INTEREST_ENABLED:true}
fineract.loan.transactionprocessor.due-penalty-interest-principal-fee-in-advance-
penalty-interest-principal-fee.enabled
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_DUE_PENALTY_INTEREST_PRINCIPAL_FEE_IN_ADVANCE_PE
NALTY_INTEREST_PRINCIPAL_FEE_ENABLED:true}
fineract.loan.transactionprocessor.advanced-payment-strategy.enabled
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_ADVANCED_PAYMENT_STRATEGY_ENABLED:true}

```

Jobs

SchedulerWritePlatformService



This service has a typo and should be called `SchedulerWritePlatformService`.

TBD

Retry-able service function `processJobDetailForExecution`

```

@Transactional
@Override
@Retry(name = "processJobDetailForExecution", fallbackMethod =
"fallbackProcessJobDetailForExecution")
public boolean processJobDetailForExecution(final String jobKey, final String
triggerType) {
    boolean isStopExecution = false;
    final ScheduledJobDetail scheduledJobDetail = this
.scheduledJobDetailsRepository.findByJobKeyWithLock(jobKey);
    if (scheduledJobDetail.isCurrentlyRunning() || (triggerType.equals
(SchedulerServiceConstants.TRIGGER_TYPE_CRON)
&& scheduledJobDetail.getNextRunTime().after(new Date())) {
        isStopExecution = true;
    }
    final SchedulerDetail schedulerDetail = retrieveSchedulerDetail();
    if (triggerType.equals(SchedulerServiceConstants.TRIGGER_TYPE_CRON) &&
schedulerDetail.isSuspended()) {
        scheduledJobDetail.setTriggerMisfired(true);
        isStopExecution = true;
    } else if (!isStopExecution) {
        scheduledJobDetail.setCurrentlyRunning(true);
        scheduledJobDetail.setMismatchedJob(false);
    }
}

```

```

        this.scheduledJobDetailsRepository.save(scheduledJobDetail);
        return isStopExecution;
    }

```

Fallback function `fallbackProcessJobDetailForExecution`

```

@SuppressWarnings("unused")
public boolean fallbackProcessJobDetailForExecution(Exception e) {
    return false;
}

```

Retry configuration for `processJobDetailForExecution`

```

fineract.loan.transactionprocessor.error-not-found-fail
=${FINERACT_LOAN_TRANSACTIONPROCESSOR_ERROR_NOT_FOUND_FAIL:true}

# Comma separated list of loan statuses which will be recorded on change. There are
two extra values: "NONE" and "ALL".
# "NONE" disables the feature and no entries will be created, "ALL" enables the
feature for all loan statuses.
fineract.loan.status-change-history-statuses
=${FINERACT_LOAN_STATUS_CHANGE_HISTORY_STATUSES:NONE}

```

Loan

LoanWritePlatformService

TBD

Retry-able service function `recalculateInterest`

```

        if (externalId.isEmpty() && TemporaryConfigurationServiceContainer
.isExternalIdAutoGenerationEnabled()) {
            externalId = ExternalId.generate();
        }

        changes.put(CLOSED_ON_DATE, command.stringValueOfParameterNamed
(TRANSACTION_DATE));
        changes.put(WRITTEN_OFF_ON_DATE, command.stringValueOfParameterNamed
(TRANSACTION_DATE));
        changes.put("externalId", externalId);

        if (DateUtils.isBefore(writtenOffOnLocalDate, loan.getDisbursementDate())) {
            final String errorMessage = "The date on which a loan is written off
cannot be before the loan disbursement date: "
                + loan.getDisbursementDate().toString();
            throw new InvalidLoanStateTransitionException("writeoff",
"cannot.be.before.submittal.date", errorMessage,
                writtenOffOnLocalDate, loan.getDisbursementDate());
        }
    }

```



```

        if (DateUtils.isDateInTheFuture(writtenOffOnLocalDate)) {
            final String errorMessage = "The date on which a loan is written off
cannot be in the future.";
            throw new InvalidLoanStateTransitionException("writeoff",
"cannot.be.a.future.date", errorMessage, writtenOffOnLocalDate);
        }

        final LoanTransaction loanTransaction = LoanTransaction.writeoff(loan, loan
.getOffice(), writtenOffOnLocalDate, externalId);
        LocalDate lastTransactionDate = loan.getLastUserTransactionDate();
        if (DateUtils.isAfter(lastTransactionDate, writtenOffOnLocalDate)) {
            final String errorMessage = "The date of the writeoff transaction must
occur on or after previous transactions.";
            throw new InvalidLoanStateTransitionException("writeoff",
"must.occur.on.or.after.other.transaction.dates", errorMessage,
writtenOffOnLocalDate);
        }

        if (loan.isInterestBearingAndInterestRecalculationEnabled()
            && DateUtils.isBeforeBusinessDate(loanTransaction.
getTransactionDate())) {

```

Fallback function `fallbackRecalculateInterest`

```

        loanTransactionProcessingService.processLatestTransaction(loan
.getTransactionProcessingStrategyCode(), loanTransaction,
            new TransactionCtx(loan.getCurrency(), loan
.getRepaymentScheduleInstallments(), loan.getActiveCharges(),
                new MoneyHolder(loan.getTotalOverpaidAsMoney()), null));
    }
    loanBalanceService.updateLoanSummaryDerivedFields(loan);
    loanLifecycleStateMachine.transition(LoanEvent.WRITE_OFF_OUTSTANDING, loan);
    changes.put(PARAM_STATUS, LoanEnumerations.status(loan.getLoanStatus()));
    return Optional.of(loanTransaction);
}

private void closeDisbursements(final Loan loan, final ScheduleGeneratorDTO
scheduleGeneratorDTO) {
    if (loan.isDisbursementAllowed() && loan.atLeastOnceDisbursed()) {

```

Retry configuration for `recalculateInterest`

```

fineract.content.regex-whitelist-enabled
=${FINERACT_CONTENT_REGEX_WHITELIST_ENABLED:true}
fineract.content.regex-whitelist=${FINERACT_CONTENT_REGEX_WHITELIST:.*\\.pdf$,.*
\\.doc,.*\\.docx,.*\\.xls,.*\\.xlsx,.*\\.jpg,.*\\.jpeg,.*\\.png}
fineract.content.mime-whitelist-enabled
=${FINERACT_CONTENT_MIME_WHITELIST_ENABLED:true}
fineract.content.mime-whitelist

```

```
=${FINERACT_CONTENT_MIME_WHITELIST:application/pdf,application/msword,application/vnd.  
openxmlformats-officedocument.wordprocessingml.document,application/vnd.ms-  
excel,application/vnd.openxmlformats-  
officedocument.spreadsheetml.sheet,image/jpeg,image/png}  
fineract.content.filesystem.enabled=${FINERACT_CONTENT_FILESYSTEM_ENABLED:true}
```

Savings

SavingsAccountWritePlatformService

TBD

Retry-able service function `postInterest`

```
        postInterestOnDate = transactionDate;  
    }  
  
    savingsAccountData = this.savingsAccountInterestPostingService  
.postInterest(mc, today, isInterestTransfer,  
        isSavingsInterestPostingAtCurrentPeriodEnd,  
financialYearBeginningMonth, postInterestOnDate, backdatedTxnsAllowedTill,  
        savingsAccountData);  
  
    if (!backdatedTxnsAllowedTill) {  
        List<SavingsAccountTransactionData> transactions = savingsAccountData  
.getSavingsAccountTransactionData();  
        for (SavingsAccountTransactionData accountTransaction : transactions)  
{  
            if (accountTransaction.getId() == null) {  
                savingsAccountData.setNewSavingsAccountTransactionData  
(accountTransaction);  
                selectAccountId(accountTransaction, savingsAccountData);  
            }  
        }  
        savingsAccountData.setExistingTransactionIds(existingTransactionIds);  
        savingsAccountData.setExistingReversedTransactionIds  
(existingReversedTransactionIds);  
    }  
    return savingsAccountData;  
}  
  
public void selectAccountId(SavingsAccountTransactionData accountTransaction,  
SavingsAccountData savingsAccountData) {  
    SavingsAccountTransactionType transactionType = SavingsAccountTransactionType  
.fromInt(accountTransaction.getTransactionType().getId().intValue());  
    if (transactionType.isOverDraftInterestPosting()) {  
        if (MathUtil.isGreaterThanZero(accountTransaction.getRunningBalance())) {  
            accountTransaction.setAccountDebit(savingsAccountData  
.getGlAccountIdForSavingsControl());  
        }  
    }  
}
```

```

        accountTransaction.setAccountCredit(savingsAccountData
.getGlaAccountIdForInterestReceivable());
    } else {
        accountTransaction.setAccountDebit(savingsAccountData
.getGlaAccountIdForOverdraftPortfolio());
        accountTransaction.setAccountCredit(savingsAccountData
.getGlaAccountIdForInterestReceivable());
    }
} else {
    accountTransaction.setAccountDebit(savingsAccountData
.getGlaAccountIdForInterestPayable());
    accountTransaction.setAccountCredit(savingsAccountData
.getGlaAccountIdForSavingsControl());
}
}

@Override
public CommandProcessingResult reverseTransaction(final Long savingsId, final Long
transactionId,
        final boolean allowAccountTransferModification, final JsonCommand command)
{

```

Fallback function fallbackPostInterest

```

        .isSavingsInterestPostingAtCurrentPeriodEnd();
        final Integer financialYearBeginningMonth = this.configurationDomainService
.retrieveFinancialYearBeginningMonth();

        // Get Savings account from savings charge
        final SavingsAccount account = savingsAccountCharge.savingsAccount();
        this.savingAccountAssembler.assignSavingAccountHelpers(account);
        final Set<Long> existingTransactionIds = new HashSet<>();
        final Set<Long> existingReversedTransactionIds = new HashSet<>();
        Pageable sortedByDateAndIdDesc = PageRequest.of(0, 1, Sort.by("dateOf", "id"
).descending());

        List<SavingsAccountTransaction> savingsAccountTransaction = this
.savingsAccountTransactionRepository
        .findBySavingsAccountIdAndLessThanDateOfAndReversedIsFalse(account
.getId(), transactionDate, sortedByDateAndIdDesc);

```

Retry configuration for postInterest

```

fineract.content.filesystem.rootFolder=${FINERACT_CONTENT_FILESYSTEM_ROOT_FOLDER:${use
r.home}/.fineract}
fineract.content.s3.enabled=${FINERACT_CONTENT_S3_ENABLED:false}
fineract.content.s3.bucketName=${FINERACT_CONTENT_S3_BUCKET_NAME:}
fineract.content.s3.accessKey=${FINERACT_CONTENT_S3_ACCESS_KEY:}
fineract.content.s3.secretKey=${FINERACT_CONTENT_S3_SECRET_KEY:}

```

```
fineract.content.s3.region=${FINERACT_CONTENT_S3_REGION:}
```

Security

Fineract is **secure by design**. It is designed and built from the ground up to accept, manage, and present data securely. This chapter will detail its various security-related features and settings, along with best practices for secure deployment.

If you believe you have found a security vulnerability in Fineract itself, [let us know privately](#).

Your task as bank CTO, sysadmin, vendor, or other entity responsible for hosting Fineract securely is to thoroughly consider these sections and thoughtfully apply them in your work. While a Fineract release *is* secure by design, it is *not* sufficient for a sysadmin to simply start it up and hope for the best. Careful steps must be taken to ensure a deployment is and remains secure despite software environment changes, attacks, staff transitions, and anything else that may arise.

We'll first cover the various supported authentication schemes and will continue on to recommendations for securing a Fineract deployment.



The HTTP Basic and OAuth authentication schemes are mutually exclusive. You can't enable them both at the same time. Fineract checks these settings on startup and will fail if more than one authentication scheme is enabled.

HTTP Basic Authentication

By default Fineract is configured with a HTTP Basic Authentication scheme, so you actually don't have to do anything if you want to use it. But if you would like to explicitly choose this authentication scheme then there are two ways to enable it:

1. Use environment variables (best choice if you run with Docker Compose):

```
FINERACT_SECURITY_BASICAUTH_ENABLED=true  
FINERACT_SECURITY_OAUTH_ENABLED=false
```

2. Use JVM parameters (best choice if you run the Spring Boot JAR):

```
java -Dfineract.security.basicauth.enabled=true -Dfineract.security.oauth2.enabled=false -jar fineract-provider.jar
```

OAuth

Fineract has basic OAuth support based on Spring Boot Security.

This can be enabled at runtime in one of two ways:

1. Use environment variables (best choice if you run with Docker Compose):

```
FINERACT_SECURITY_BASICAUTH_ENABLED=false  
FINERACT_SECURITY_OAUTH_ENABLED=true  
FINERACT_SERVER_OAUTH_RESOURCE_URL=http://localhost:9000/realms/fineract
```

2. Use JVM parameters (best choice if you run the Spring Boot JAR):

```
java -Dfineract.security.basicauth.enabled=false -Dfineract  
.security.oauth2.enabled=true -jar fineract-provider.jar
```

Here's how to test OAuth with [Keycloak](#).

The steps required for other OAuth providers will be similar.

Set up Keycloak

1. From terminal, run: `docker run -p 9000:8080 -e KC_BOOTSTRAP_ADMIN_USERNAME=admin -e KC_BOOTSTRAP_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:26.2.5 start-dev`
2. Go to URL 'http://localhost:9000/admin' and login with admin/admin
3. Click 'Manage realms', then 'Create realm'
4. Enter name `fineract` for the realm name
5. Click on tab 'Users' on the left, then 'Create new user' with username `mifos`, email `test@example.com`, First name `Mifos`, Last name `User`
6. Click on tab 'Credentials' at the top, and set password to `password`, turning 'temporary' setting to off
7. Click on tab 'Clients' on the left, and create client with ID `community-app`
8. In Settings tab, set 'Valid redirect URIs' to `localhost`, enable 'Client authentication', check 'Direct access grants'
9. Click 'Save' and a 'Credentials' tab will appear
10. In Credentials tab, copy string in field 'secret' as this will be needed in the step to request the access token

Finally we need to change Keycloak configuration so that it uses the username as a subject of the token:

1. Choose client `community-app` in the tab 'Clients'
2. Click on tab 'Client scopes', then `community-app-dedicated`
3. Go to tab 'Mappers', click 'Configure a new mapper' and choose 'User Property'
4. Enter `usernameInSub` as 'Name'
5. Enter `username` into the field 'Property' and `sub` into the field 'Token Claim Name'

You are now ready to test out OAuth:

Retrieve an access token from Keycloak

```
curl --request POST \
  "$FINERACT_SERVER_OAUTH_RESOURCE_URL/protocol/openid-connect/token" \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'username=mifos' \
  --data-urlencode 'password=password' \
  --data-urlencode 'client_id=community-app' \
  --data-urlencode 'grant_type=password' \
  --data-urlencode 'client_secret=<enter the client secret from credentials tab>'
```

The reply should contain a field `access_token`. Copy the field's value and use it in the API call below:

Invoke APIs and pass **Authorization: bearer ... header**

```
curl --insecure \
  'https://localhost:8443/fineract-provider/api/v1/offices' \
  --header 'Fineract-Platform-TenantId: default' \
  --header 'Authorization: bearer <enter the value of the access_token field>'
```



See also fineract.apache.org/docs/legacy/#authentication_oauth

Two-factor authentication

You can enable 2FA authentication. Depending on how you start Fineract add the following:

1. Use environment variables (best choice if you run with Docker Compose):

```
FINERACT_SECURITY_2FA_ENABLED=true
```

2. Use JVM parameter (best choice if you run the Spring Boot JAR):

```
java -Dfineract.security.2fa.enabled=true -jar fineract-provider.jar
```

Securing Fineract

This section covers best practices in securing the use of Fineract.

Security hardening is a continuum and Fineract is adaptable to your security needs. There's no one way to correctly deploy it and the open source project offers no warranty. It's up to you to deploy and maintain it carefully, according to your organization's needs and compliance requirements.

Since Fineract is a financial application with PII (personally identifiable information), it is vital that it is secured in production whenever it is setup. If you are a small financial entity, bank, credit

union, microfinance organization, or non-banking financial institution, the project urges you to identify and work with the vendors that work regularly with Fineract and are regular contributors to the security fixes. In this way, we encourage a community of contributions that keep the overall solution secure. Open source gets the benefit of many people reviewing the code and suggesting issues and solutions - let's ensure that virtuous cycle can work by supporting those working on security at Fineract.

Members of the Security team can be reached at [security AT fineract.apache.org](mailto:security@fineract.apache.org). The reporting mechanisms for vulnerabilities and exploits are there, not on the public dev list.

See apache security practices for more information. security.apache.org

Also, we recommend you familiarize yourself with the OWASP foundation and the "Cheat Sheet" series cheatsheetseries.owasp.org

Tips for securing the Fineract infrastructure

Run it isolated and/or disconnected

In the world of Microfinance or small banking operations (in some geographies), it is possible that you can run Fineract on a private network, or isolated from the internet by being hosted locally and securing all connections. This could involve establishing a VPN with limited ports open, and only accepting connections within that VPN. At the far end of this spectrum, is running it isolated and air-gapped as a backend accounting system, where there is no internet connection on that device. In such scenarios, you are limiting the vectors of attack to just those employees you give access to. You are also limiting the functionality to accounting and basic operations, so this is rarely appropriate. Even in these scenarios, it is important that you establish reviews of logs and accounts on a periodic basis to determine if any internal fraud is occurring. Such things should be part of your operational manual. There are a number of resources available for this topic, please find them online. For Fineract in particular, be mindful of the set up of approvals and the access you give to each person or role in your organization

Running it connected but behind a firewall

It should be clear that running it on the internet directly, without API monitoring and filtering, is a bad idea. This is especially true if your Fineract instance is connected to a payment mechanism of any kind. Imagine an exploit being used to gain access and then to send funds from an account to an outside merchant or bank. An attacker could drain an account before you can detect the issue. And, then it will depend on the payment scheme rules whether any of those funds are recoverable.

There are multiple ways to enhance the security that is built into Fineract, but none of them are bulletproof and so you must have defense in depth. One key thing is to run the Fineract instance behind an API gateway, and to prevent certain API patterns or calls that are likely to be fraudulent. The important thing is to recognize that while you may not be a target institution now, at any moment this can change, and your IP will be listed on the dark web as a potential target for exploit. Your IT team must also have ways to quickly turn off services to limit the damage.

It is recommended to run it with at least API Gateway, WAF (Web Application Firewall) and SQL Injection filtering tool if connecting to the internet. Fineract must be hardened to run in

production.

Fraud prevention

Even if you have secured your infrastructure, you will need the ability to monitor fraudulent traffic, and then to stop that fraud in real time. Fraud can occur even when your infrastructure is good, but a user account has been accessed through a phishing attack or similar vector. And, in the world of real time payments and multiple payment types and channels, there is a need for additional real time monitoring, and inline processing of potential fraudulent transactions. Detection and blocking needs to occur in real time and then resolution can occur more leisurely with additional manual and help desk interventions.

If you are a small institution and you are getting into this situation, you should consider having holds on all payment and transfers built into your process, until you can enable effective tools.

There are a number of fraud prevention tools that are available in market from fraud prevention vendors. When looking for solution providers, the ideal scenario is a vendor with longevity and a track record in your market for detecting recent fraudulent activities. Pattern detection is a key part of this, and for that, it is important to be able to get enough data from your systems to identify the anomalies.

There are a number of GitHub projects that cover algorithms for conducting ML on transactions to detect anomalies. There is also an open source project, Tazama, which provides a kind of framework for your own logic and algorithms: github.com/frmscoe/docs.

Fraud exploits are on the rise, supercharged by AI tooling. AI tooling can also be used to fight these trends, but it is an ever escalating area of concern for financial providers globally.

Self-service APIs

It is recommended that you leave the Self Service APIs disabled to avoid any potential exploits there. Apps should not be developed to use those APIs.

There is a way to run those APIs endpoint (re-written but consistent) in a separate isolated component, where there is a way to control the ingress and egress of data. Once that component is linked up with authenticated users with a fully designed authorization scheme, then the APIs can be accessed. This is an area of exploration by the project. Currently, Fineract should not be run in a way that allows access to those APIs. We strongly advise against using any APP that connects to those APIs without revising the architecture as described, except in a test or demo environment.

User Education and Training

Educating and training your team is another limb of your organizational cybersecurity defense. Equipped with engaging security awareness training sessions, end-users can be prepared with both knowledge and skills on how to identify potential security threats and react to them. You can get more information from some of the resources offered in the course during CISA Training: www.cisa.gov

Regular Security Audits and Compliance Checks

Know your compliance surface. Regularly conduct routine security audits and compliance checks. This can be helpful in finding all the vulnerabilities and their fix prior to exploitation, thereby helping to reduce the exposure window. A combined automated tool with manual expert reviews provides complete coverage. There are multiple vendors available that scan for compliance with existing security standards. We don't recommend any vendor in particular, but for pointers you can look at owasp.org/www-community/Vulnerability_Scanning_Tools

Key Management and Data Encryption Strategies

Implement strong data encryption strategies to protect sensitive information. Key management should be something that your IT team does, utilizing best practices. Just like a physical key, you should keep it in a secure location with limited access and take special care not to copy it to digital locations that can be scanned or found, including email systems. Make sure you have procedures in place.

You would probably want to encrypt the data at rest with AES-256 and [in transit](#) via TLS 1.3. Create and maintain binding standards for encryption in your organization. And remember, key management to encryption is the key. Every cloud providers provides key management services that help you manage and secure your keys.

Examples:

- aws.amazon.com/kms/
- cloud.google.com/docs/security/key-management-deep-dive
- learn.microsoft.com/en-us/azure/security/fundamentals/key-management
- github.com/getsops/sops
- github.com/Infisical/infisical

Secure Coding Practices

Secure code by following secure coding practices and standards, such as OWASP's top ten, for any kind of vulnerability at the code level. Use tools like SonarQube for finding security problems in your source code through static application security testing (SAST) prior to deploying an application. Note that SonarQube has already been integrated into our automation build process.

Apache Software Foundation has an account with SonarQube and Fineract scans can be found in that account.

Multi-factor Authentication (MFA)

Enhance your security layers with MFA (or 2FA: two-factor authentication). One such approach, built on three things: something the user knows (like a password), something the user has (like a security token), and something the user is (biometric verification, for example). When MFA is used, it adds another layer of security. Solutions such as Duo Security may be a good implementation for MFA.

Leverage Community Support

You should stay engaged with the Fineract community to stay on top of security updates, patches, and best practices. Also, look for the possibility of collaboration with cybersecurity firms that would help you increase the capability of your threat detection and response system. Such relationships may avail specialized skills, technologies, and intelligence that may strengthen the security posturing of your organization.

Testing

TBD

Cucumber E2E Tests

Apache Fineract's E2E test suite provides comprehensive coverage of business functionality using Cucumber BDD (Behavior-Driven Development) framework. These tests serve as both functional validation and living documentation of the system's capabilities.

Overview

Architecture

- **fineract-e2e-tests-runner**: Contains all Cucumber feature files and test scenarios
- **fineract-e2e-tests-core**: Contains step definitions, test utilities, and supporting code
- **Framework**: Cucumber with Java, using Gherkin syntax for readable test specifications
- **Prerequisites**: Running Apache Fineract instance (typically on port 8443 with HTTPS)

Test Organization

- Feature files located in: `fineract-e2e-tests-runner/src/test/resources/features/`
- Step definitions in: `fineract-e2e-tests-core/src/test/java/org/apache/fineract/test/stepdef/`
- Tests are tagged with TestRail IDs for traceability (e.g., `@TestRailId:C16`)
- Special tags include `@Smoke` for quick validation tests

Prerequisites

Required Software

- **Java 21**: Apache Fineract requires Java 21 (Azul Zulu JDK recommended)
- **Database**: MariaDB 11.5.2, PostgreSQL 17.4, or MySQL 9.1
- **Git**: For source code management
- **Gradle 8.14.3**: Included via wrapper

Database Setup

Before running E2E tests, ensure the databases are created:

```
# Create required databases
./gradlew createdB -PdbName=fineract_tenants
./gradlew createdB -PdbName=fineract_default
```

Configuration

Connection Configuration

E2E tests connect to the running Fineract instance. Default configuration in `fineract-e2e-tests-core/src/test/resources/fineract-test-application.properties`:

```
# Connection details to running backend
fineract-test.api.base-url=${BASE_URL:https://localhost:8443}
fineract-test.api.username=${TEST_USERNAME:mifos}
fineract-test.api.password=${TEST_PASSWORD:password}
fineract-test.api.tenant-id=${TEST_TENANT_ID:default}
```

To override defaults, use environment variables:

```
export BASE_URL=http://localhost:8080
export TEST_USERNAME=mifos
export TEST_PASSWORD=password
export TEST_TENANT_ID=default
```

Test Data Initialization



Many E2E tests require pre-configured test data (loan products, charges, configurations). This initialization is controlled by the `fineract-test.initialization.enabled` property.

Default Behavior:

- By default, initialization is **DISABLED** (`fineract-test.initialization.enabled=false`)
- Without initialization, tests will fail with errors like:

```
java.lang.IllegalArgumentException: Loan product
[LP2_ADV_CUSTOM_PMT_ALLOC_PROGRESSIVE_LOAN_SCHEDULE_HORIZONTAL] not found
```

How to Enable Initialization:

Method 1 - Environment Variable (Recommended):

```
cd fineract-e2e-tests-runner
INITIALIZATION_ENABLED=true ../gradlew cucumber
```

Method 2 - System Property:

```
cd fineract-e2e-tests-runner
```

```
../gradlew cucumber -DINITIALIZATION_ENABLED=true
```

Method 3 - Modify Properties File:

Edit `fineract-e2e-tests-core/src/test/resources/fineract-test-application.properties`:

```
fineract-test.initialization.enabled=true
```

What Initialization Creates:

- 100+ loan products with specific configurations
- Various charge types (NSF fees, processing fees, etc.)
- Payment allocation rules
- Interest calculation configurations
- Advanced payment allocation strategies
- Progressive loan schedule configurations

When to Use Initialization:

- First-time test execution on a fresh database
- After database reset/recreation
- When running tests that require specific loan products
- Testing new features that depend on pre-configured products



Initialization takes additional time (2-5 minutes) as it creates extensive test data. Consider running it once and reusing the database for multiple test runs.

Business Date Configuration

CRITICAL: The Business Date feature must be enabled in the database for many E2E tests to function correctly.

Default Behavior:

- Business Date is **DISABLED** by default in fresh Fineract installations
- Without Business Date enabled, tests fail with:

```
{
  "errors": [
    {
      "defaultUserMessage": "Business date functionality is not enabled",
      "developerMessage": "Business date functionality is not enabled",
      "userMessageGlobalisationCode": "business.date.is.not.enabled"
    }
  ]
}
```

How to Enable Business Date:

Method 1 - Via SQL (Direct Database):

```
mysql -u root -pmysql fineract_default -e \  
"UPDATE c_configuration SET enabled = 1 WHERE name = 'enable-business-date';"
```

Method 2 - Via API (After Fineract is Running):

```
curl -X PUT https://localhost:8443/fineract-  
provider/api/v1/configurations/name/enable-business-date \  
-H "Authorization: Basic bWlmb3M6cGFzc3dvcmQ=" \  
-H "Content-Type: application/json" \  
-d '{"enabled": true}'
```

Verification:

```
mysql -u root -pmysql fineract_default -e \  
"SELECT * FROM c_configuration WHERE name LIKE '%business%';"
```

Running E2E Tests

Complete Workflow

Step 1: Start Fineract

```
# Start Fineract in background  
./gradlew bootRun
```

Wait for Fineract to be fully started. You can verify by checking:

```
curl -k https://localhost:8443/actuator/health
```

Step 2: Enable Business Date (if needed)

```
mysql -u root -pmysql fineract_default -e \  
"UPDATE c_configuration SET enabled = 1 WHERE name = 'enable-business-date';"
```

Step 3: Run E2E Tests

Navigate to the E2E tests module:

```
cd fineract-e2e-tests-runner
```

Run All E2E Tests:

```
# First run with initialization
INITIALIZATION_ENABLED=true ../gradlew cucumber

# Subsequent runs without initialization (faster)
../gradlew cucumber
```

Run Specific Feature File:

```
../gradlew cucumber -Pcucumber.features="src/test/resources/features/Loan.feature"
```

Run Tests by Tag:

```
# Run only smoke tests
../gradlew cucumber -Pcucumber.tags="@Smoke"

# Run specific TestRail test
../gradlew cucumber -Pcucumber.tags="@TestRailId:C16"

# Run multiple tags
../gradlew cucumber -Pcucumber.tags="@Smoke and @TestRailId:C16"
```

Run Tests with Custom Configuration:

```
BASE_URL=http://localhost:8080 \
TEST_USERNAME=admin \
TEST_PASSWORD=admin123 \
INITIALIZATION_ENABLED=true \
../gradlew cucumber
```

Gradle Command Options

Basic Cucumber Task

```
../gradlew cucumber
```

Feature File Selection

```
# Single feature
../gradlew cucumber -Pcucumber.features="src/test/resources/features/Client.feature"

# Multiple features
../gradlew cucumber -Pcucumber.features
```



```
= "src/test/resources/features/Client.feature:src/test/resources/features/Loan.feature"

# Specific scenario by line number
../gradlew cucumber -Pcucumber.features="src/test/resources/features/Loan.feature:45"
```

Tag-Based Execution

```
# Single tag
../gradlew cucumber -Pcucumber.tags="@Smoke"

# Multiple tags (AND)
../gradlew cucumber -Pcucumber.tags="@Smoke and @TestRailId:C16"

# Multiple tags (OR)
../gradlew cucumber -Pcucumber.tags="@Smoke or @TestRailId:C16"

# Exclude tags
../gradlew cucumber -Pcucumber.tags="not @ignore"

# Complex tag expression
../gradlew cucumber -Pcucumber.tags="@Smoke and not @ignore"
```

Report Generation

```
# Generate HTML report
../gradlew cucumber -Dcucumber.plugin="pretty,html:build/cucumber-
reports/cucumber.html"

# Generate JSON report
../gradlew cucumber -Dcucumber.plugin="json:build/cucumber-reports/cucumber.json"

# Multiple report formats
../gradlew cucumber -Dcucumber.plugin="pretty,html:build/cucumber-
reports/cucumber.html,json:build/cucumber-reports/cucumber.json"

# Generate Allure report (comprehensive visual reporting)
../gradlew cucumber allureReport
```

After running tests with Allure, the report is available at:

```
fineract-e2e-tests-runner/build/reports/allure-report/index.html
```



Allure provides rich visual reports with test history, statistics, and detailed execution information. Open the `index.html` file in a browser to view the interactive report.

Clean and Run

```
# Clean previous test results and run
../gradlew clean cucumber
```

Advanced Execution Scenarios

Running Against Different Environment

```
# Against staging environment
BASE_URL=https://staging.example.com:8443 \
TEST_USERNAME=staging_user \
TEST_PASSWORD=staging_pass \
../gradlew cucumber
```

Running with External Event Verification

```
# Enable external event verification (requires ActiveMQ)
ACTIVEMQ_BROKER_URL=tcp://localhost:61616 \
ACTIVEMQ_BROKER_USERNAME=admin \
ACTIVEMQ_BROKER_PASSWORD=admin \
ACTIVEMQ_TOPIC_NAME=fineract-events \
EVENT_VERIFICATION_ENABLED=true \
../gradlew cucumber
```

Running with TestRail Integration

```
TESTRAIL_ENABLED=true \
TESTRAIL_BASEURL=https://testrail.example.com \
TESTRAIL_USERNAME=test@example.com \
TESTRAIL_PASSWORD=testrail_password \
TESTRAIL_RUN_ID=123 \
../gradlew cucumber
```

Test Development

Feature Coverage

The E2E tests cover the following functional domains:

Client Management

- Client creation and management
- Address management
- Document management

- Family member tracking

Loan Management

- Loan application and approval
- Disbursement
- Repayment processing
- Charges and fees
- Advanced features: chargeback, charge-off, re-aging, re-amortization
- Specialized loans: down payment, merchant-issued refund

Savings Account Management

- Account opening and activation
- Deposits and withdrawals
- Interest calculation
- Account closure

Accounting

- Journal entry validation
- Asset externalization
- GL account mapping

Operational Processes

- Close of Business (COB)
- Inline COB
- Business date management
- Batch API operations

Writing New E2E Tests

When writing new Cucumber tests:

1. **Create Feature File:** Add new `.feature` file in `fineract-e2e-tests-runner/src/test/resources/features/`
2. **Use Gherkin Syntax:**

Feature: Loan Disbursement

Scenario: Successful loan disbursement

Given A client named "John Doe"

When Admin creates a loan for client with amount "1000"

And Admin approves the loan

```
And Admin disburses the loan on business date
Then Loan status is "Active"
And Loan outstanding balance is "1000"
```

3. **Implement Step Definitions:** Add corresponding step definitions in `fineract-e2e-tests-core/src/test/java/org/apache/fineract/test/stepdef/`
4. **Add Tags:** Tag scenarios appropriately:

```
@TestRailId:C1234 @Smoke
Scenario: Critical loan test
```

5. **Verify with Gradle:**

```
cd fineract-e2e-tests-runner
../gradlew cucumber -Pcucumber.features
="src/test/resources/features/YourNewFeature.feature"
```

Troubleshooting

Common Test Failures

Connection Issues

Symptom: Tests fail with connection refused errors

Solutions:

```
# Verify Fineract is running
curl -k https://localhost:8443/actuator/health

# Check if port is in use
netstat -tulpn | grep 8443

# Check logs (logs go to console/stdout)
# If running in background, redirect output:
# ./gradlew bootRun > build/bootRun.log 2>&1 &
# tail -f build/bootRun.log
```

Business Date Issues

Symptom: Tests fail with "Business date functionality is not enabled"

Solutions:

```
# Enable Business Date
mysql -u root -pmysql fineract_default -e \
```

```
"UPDATE c_configuration SET enabled = 1 WHERE name = 'enable-business-date';"
```

```
# Verify
```

```
mysql -u root -pmysql fineract_default -e \  
"SELECT * FROM c_configuration WHERE name = 'enable-business-date';"
```

Data Dependencies

Symptom: Tests fail due to missing products or charges

Solutions:

```
# Run with initialization enabled  
cd fineract-e2e-tests-runner  
INITIALIZATION_ENABLED=true ../gradlew cucumber
```

Authentication Failures

Symptom: 401 or 403 errors

Solutions:

```
# Verify credentials  
TEST_USERNAME=mifos TEST_PASSWORD=password ../gradlew cucumber  
  
# Check user permissions in database  
mysql -u root -pmysql fineract_default -e \  
"SELECT * FROM m_appuser WHERE username = 'mifos';"
```

Debugging Tips

Enable Detailed Logging

```
# Run with verbose output  
../gradlew cucumber -Dcucumber.plugin="pretty" --info  
  
# Save output to file  
../gradlew cucumber > test-output.log 2>&1
```

Check Database State

```
# Check loan products after initialization  
mysql -u root -pmysql fineract_default -e \  
"SELECT id, product_name FROM m_product_loan LIMIT 20;"  
  
# Check configurations  
mysql -u root -pmysql fineract_default -e \  
"
```

```
"SELECT name, enabled FROM c_configuration WHERE name LIKE '%enable%';"
```

View Test Reports

After test execution, reports are available in:

```
fineract-e2e-tests-runner/build/cucumber-reports/  
fineract-e2e-tests-runner/build/reports/tests/
```

Best Practices

Test Organization

- Keep feature files focused on specific business domains
- Use descriptive scenario names
- Include business context in scenario descriptions
- Tag tests appropriately for organization and execution

Test Isolation

- Each test should be independent
- Don't rely on state from other tests
- Clean up test data in `@After` hooks
- Use unique identifiers for test entities

Test Data Management

- Use initialization for complex test data setup
- Create minimal required data in test scenarios
- Document test data dependencies
- Reuse database for multiple test runs when possible

Performance Optimization

- Run initialization once per database
- Use tags to run subset of tests during development
- Run full suite in CI/CD pipelines
- Consider parallel execution for large test suites

Cucumber Cheatsheet

Cucumber is a test framework based on Behavior-Driven Development (BDD). Tests are written in plain text with very basic syntax rules. These rules form a mini language that is called Gherkin.

A specification resembles spoken language. This makes it ideal for use with non-technical people that have domain specific knowledge. The emphasis of Cucumber lies on finding examples to describe your test cases. The few keywords and language rules are easy to explain to anyone (compared JUnit for example).

Keywords

The Gherkin language has the following keywords:

- **Feature**
- **Rule**
- **Scenario Outline** or **Scenario Template**
- **Example** or **Scenario**
- **Examples** or **Scenarios**
- **Background**
- **Given**
- **And**
- **But**
- **When**
- **Then**

There are a couple of additional signs used in Gherkin:

- **|** is as column delimiters in **Examples** tables
- with **@** you can assign any kind of tags to categorize the specs (or e.g. relate them to certain Jira tickets)
- **#** is used to indicate line comments



The tag **@ignore** is used to skip tests. This is a somewhat arbitrary choice (we could use any other tag to indicate temporarily disabled tests).

Each non-empty line of a test specification needs to start with one of these keywords. The text blocks that follows the keywords are mapped to so called step definitions that contain the actual test code.

A typical Cucumber test specification written in Gherkin looks like this:

```
Feature: Template Service
```

```
@template
```

```
Scenario Outline: Verify that mustache templates have expected results
```

```
  Given A mustache template file <template>
```

```
  Given A JSON data file <json>
```

```
  When The user merges the template with data
```

Then The result should match the content of file <result>

Examples:

template	json	result
hello.mustache	hello.json	hello.txt
loan.mustache	loan.json	loan.html
array.loop.mustache	array.json	array.loop.txt
array.index.mustache	array.json	array.index.txt

The corresponding step definitions would look like this:

```
package org.apache.fineract.template.service;

import static org.junit.jupiter.api.Assertions.assertEquals;

import com.google.common.reflect.TypeToken;
import com.google.gson.Gson;
import com.google.gson.JsonElement;
import com.google.gson.JsonParser;
import io.cucumber.java8.En;
import java.io.IOException;
import java.lang.reflect.Type;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.apache.commons.io.IOUtils;
import org.apache.fineract.template.domain.Template;
import org.apache.fineract.template.domain.TemplateMapper;
import org.springframework.beans.factory.annotation.Autowired;

public class TemplateServiceStepDefinitions implements En {

    @Autowired
    private TemplateMergeService tms;

    private String template;

    private Map<String, Object> data;

    private String result;

    public TemplateServiceStepDefinitions() {
        Given("/^A mustache template file (.*)$/", (String file) -> {
            template = IOUtils.resourceToString("templates/" + file, StandardCharsets
.UTF_8,
                TemplateServiceStepDefinitions.class.getClassLoader());
        });

        Given("/^A JSON data file (.*)$/", (String file) -> {
```



```

        data = parse(IOUtils.resourceToString("templates/" + file,
StandardCharsets.UTF_8,
        TemplateServiceStepDefinitions.class.getClassLoader()));
    });

    When("The user merges the template with data", () -> {
        result = compile(template, data);
    });

    Then("/^The result should match the content of file (.*)$/", (String file) ->
{
        String expected = IOUtils.resourceToString("results/" + file,
StandardCharsets.UTF_8,
        TemplateServiceStepDefinitions.class.getClassLoader());
        assertEquals(expected, result);
    });
}

private String compile(String templateText, Map<String, Object> scope) throws
IOException {
    List<TemplateMapper> mappers = new ArrayList<>();
    Template template = new Template("TemplateName", templateText, null, null,
mappers);
    return tms.compile(template, scope);
}

private Map<String, Object> parse(String data) {
    Gson gson = new Gson();
    Type ssMap = new TypeToken<Map<String, Object>>() {}.getType();
    JsonElement json = JsonParser.parseString(data);
    return gson.fromJson(json, ssMap);
}
}

```



This example is an actual test specification that you can find in the `fineract-provider` module.

Feature

This keyword is used to group scenarios and to group related scenarios. All Gherkin specifications must start with the word **Feature**.

Descriptions

A description is any non-empty line that doesn't start with a keyword. Descriptions can be placed under the keywords:

- **Feature**
- **Rule**

- Background
- Example/Scenario
- Scenario Outline

Rule

Rule is used to group multiple related scenarios together.

Example/Scenario

This is the important part of the specification as it should describe the business logic in more detail with the usage of steps (usually **Given**, **When**, **Then**)

Steps

TBD

Given

TBD

When

TBD

Then

TBD

And, But

TBD

Background

TBD

Scenario Outline

TBD

Examples/Tables

TBD

Outlook

As a proof of concept we've converted all unit tests in **fineract-provider** into Cucumber tests. The more interesting part starts when we'll attack the integration tests with over 400 mostly business logic related tests. These tests fit very well in Cucumber's test specification structure (a lot of *if-then-else* or in Gherkin: *Given-When-Then*). Migrating all tests will take a while, but we would already recommend trying to implement tests as Cucumber specifications. It should be relatively easy to

convert these tests into the new syntax.

Hopefully this will motivate even more people from the broader Fineract community to participate in the project by sharing their domain specific knowledge as Cucumber specifications. Specifications are written in English (although not a technical requirement).



Have a look at the specifications in [fineract-provider](#) for an initial inspiration. For more information please see cucumber.io/docs

Unit Testing

TBD

Integration Testing

Integration tests in Apache Fineract validate the complete API layer and business logic by making HTTP calls to a running Fineract instance. These tests ensure that different components work together correctly and that the API behaves as expected.

Overview

Architecture

- **Location:** [integration-tests/src/test/java/org/apache/fineract/integrationtests](#)
- **Framework:** JUnit 5 with REST Assured for HTTP communication
- **Base Class:** Most tests extend [BaseLoanIntegrationTest](#) or [IntegrationTest](#)
- **Client Library:** Uses Fineract client library for type-safe API interactions
- **Prerequisites:** Running Apache Fineract instance (default: [localhost:8443](#))

Key Characteristics

- Tests run against a live Fineract instance
- Validates end-to-end API functionality
- Tests business logic, validation rules, and workflows
- Includes accounting verification and data integrity checks
- Uses real database transactions
- Tests can be run individually or as a suite

Prerequisites

Required Software

- **Java 21:** Apache Fineract requires Java 21 (Azul Zulu JDK recommended)
- **Database:** MariaDB 11.5.2, PostgreSQL 17.4, or MySQL 9.1

- **Git:** For source code management
- **Gradle 8.14.3:** Included via wrapper
- **12GB RAM:** Recommended for test execution

Database Setup

Before running integration tests, ensure the databases are created:

```
# Create required databases
./gradlew createDB -PdbName=fineract_tenants
./gradlew createDB -PdbName=fineract_default
```

Fineract Instance

Integration tests run fineract instance from cargo plugin by default.

Configuration

Default Connection Settings

Integration tests use the following default connection settings:

```
BACKEND_PROTOCOL=https
BACKEND_HOST=localhost
BACKEND_PORT=8443
BACKEND_USERNAME=mifos
BACKEND_PASSWORD=password
BACKEND_TENANT=default
```

Override Configuration

To override default values, set environment variables:

```
# Set custom connection details
export BACKEND_PROTOCOL=http
export BACKEND_HOST=localhost
export BACKEND_PORT=8080
export BACKEND_USERNAME=admin
export BACKEND_PASSWORD=admin123
export BACKEND_TENANT=default
```

Running Integration Tests

Complete Workflow

Step 1: Start Fineract

```
# Start Fineract in background
./gradlew bootRun &

# Wait for startup (manual check)
curl -k https://localhost:8443/actuator/health
```

Expected response:

```
{"status": "UP"}
```

Step 2: Run Integration Tests

Navigate to the project root and execute tests:

```
# Run all integration tests
./gradlew :integration-tests:test

# Run with clean build
./gradlew clean :integration-tests:test
```

Running Specific Tests

Run Single Test Class

```
# Run entire test class
./gradlew :integration-tests:test --tests ClientLoanIntegrationTest

# Run with verbose output
./gradlew :integration-tests:test --tests ClientLoanIntegrationTest --info
```

Run Specific Test Method

```
# Run single test method
./gradlew :integration-tests:test --tests ClientLoanIntegrationTest.testLoanSchedule

# Run multiple specific tests
./gradlew :integration-tests:test --tests ClientLoanIntegrationTest.testLoanSchedule \
--tests ClientLoanIntegrationTest.testLoanRepayment
```

Run Tests by Pattern

```
# Run all loan-related tests
./gradlew :integration-tests:test --tests "*Loan*"

```

```
# Run all client-related tests
./gradlew :integration-tests:test --tests "*Client*"

# Run all accounting tests
./gradlew :integration-tests:test --tests "*Accounting*"

# Run all COB tests
./gradlew :integration-tests:test --tests "*COB*"
```

Advanced Test Execution

Run with Test Filtering

```
# Run tests excluding specific packages
./gradlew :integration-tests:test --tests "*" \
  --exclude "*Deprecated*"

# Run only fast tests (custom tag)
./gradlew :integration-tests:test --tests "*Fast*"
```

Parallel Execution

```
# Run tests in parallel
./gradlew :integration-tests:test --parallel --max-workers=4

# Set custom thread count
./gradlew :integration-tests:test --parallel --max-workers=8
```



Some integration tests may have dependencies on shared state. Use parallel execution carefully and ensure tests are properly isolated.

Run with Custom JVM Arguments

```
# Increase heap size for large test suites
./gradlew :integration-tests:test -Xmx4g

# Enable debugging
./gradlew :integration-tests:test --debug-jvm
```

Generate Test Reports

```
# Run tests and generate HTML reports
./gradlew :integration-tests:test

# Reports are generated at:
```

```
# integration-tests/build/reports/tests/test/index.html
```

Continuous Execution

```
# Watch for changes and re-run tests
./gradlew :integration-tests:test --continuous

# Run specific test continuously
./gradlew :integration-tests:test --continuous --tests ClientLoanIntegrationTest
```

Test Execution Examples

Basic Loan Workflow Test

```
# Test complete loan lifecycle
./gradlew :integration-tests:test --tests LoanApplicationTest
```

Progressive Loan Tests

```
# Run all progressive loan tests
./gradlew :integration-tests:test --tests "*Progressive*"
```

Accounting Integration Tests

```
# Run accounting-related tests
./gradlew :integration-tests:test --tests AccountingScenarioIntegrationTest
```

Business Date Tests

```
# Run business date functionality tests
./gradlew :integration-tests:test --tests BusinessDateTest
```

Charge-Off Tests

```
# Run charge-off related tests
./gradlew :integration-tests:test --tests "*ChargeOff*"
```

Test Structure

BaseLoanIntegrationTest Overview

`BaseLoanIntegrationTest` is the comprehensive base test class for loan-related integration tests. It provides:

Pre-configured Loan Product Creation

```
// Create standard loan products
createOnePeriod30DaysLongNoInterestPeriodicAccrualProduct()
create4IProgressive() // Progressive loan products
create4IProgressiveWithCapitalizedIncome() // With capitalized income
createOnePeriod30DaysPeriodicAccrualProductWithAdvancedPaymentAllocation()
```

Transaction Management

```
// Validate loan transactions
verifyTransactions(loanId,
    transaction(100.0, "Disbursement", "01 January 2024"),
    transaction(50.0, "Repayment", "15 January 2024")
);

// Verify accounting journal entries
verifyJournalEntries(loanId, expectedEntries);

// Create transaction test data
Transaction txn = transaction(amount, type, date);
```

Loan Lifecycle Operations

```
// Disburse loan
disburseLoan(loanId, BigDecimal.valueOf(100), "01 January 2024");

// Undo disbursement
undoDisbursement(loanId);

// Re-age loan
reAgeLoan(loanId, reAgeRequest);

// Re-amortize loan
reAmortizeLoan(loanId, reAmortizeRequest);

// Execute Close of Business
executeInlineCOB(loanId);
```

Business Date Management

```
// Execute code at specific business date
runAt("01 January 2024", () -> {
    Long loanId = applyAndApproveProgressiveLoan(...);
    disburseLoan(loanId, BigDecimal.valueOf(100), "01 January 2024");
});
```



```
// Execute over date range
runFromToInclusive("01 January 2024", "31 January 2024", () -> {
    // Operations for each date in the range
});

// Execute without bypass privileges
runAsNonByPass(() -> {
    // Test operations with regular user permissions
});
```

Verification Methods

```
// Validate repayment schedule
verifyRepaymentSchedule(loanId, expectedSchedule);

// Check loan status
verifyLoanStatus(loanId, "ACTIVE");

// Verify outstanding amounts
verifyOutstanding(loanId, expectedOutstanding);

// Check arrears status
verifyArrears(loanId, expectedArrears);
```

Common Test Patterns

Test Setup Pattern

```
@BeforeEach
public void setup() {
    Utils.initializeRESTAssured();
    this.requestSpec = new RequestSpecBuilder()
        .setContentType(ContentType.JSON)
        .build();
    this.requestSpec.header("Authorization", "Basic " +
        Utils.loginIntoServerAndGetBase64EncodedAuthenticationKey());
    this.responseSpec = new ResponseSpecBuilder()
        .expectStatusCode(200)
        .build();
}
```

Date-Specific Operations

```
runAt("01 January 2024", () -> {
    // Create client
    Long clientId = clientHelper.createClient(...);
```

```

// Apply for loan
Long loanId = applyLoan(clientId, productId, amount);

// Approve loan
approveLoan(loanId, "01 January 2024");

// Disburse loan
disburseLoan(loanId, amount, "01 January 2024");
});

```

Structured Verification

```

// Verify transactions
verifyTransactions(loanId,
    transaction(100.0, "Disbursement", "01 January 2024"),
    transaction(50.0, "Capitalized Income", "01 January 2024"),
    transaction(0.55, "Capitalized Income Amortization", "01 January 2024")
);

// Verify journal entries using convenience methods
verifyJournalEntries(loanId,
    debit(account.getLoansReceivable(), 100.0),
    credit(account.getSuspenseClearingAccount(), 100.0)
);

// Or using full journalEntry method with Account objects
verifyJournalEntries(loanId,
    journalEntry(100.0, account.getLoansReceivable(), "DEBIT"),
    journalEntry(100.0, account.getSuspenseClearingAccount(), "CREDIT")
);

```

Progressive Loan Testing

```

// Create progressive loan product
Long productId = create4IProgressive();

// Apply and approve
Long loanId = applyAndApproveProgressiveLoan(clientId, productId,
    amount, numberOfRepayments, interestRate);

// Test advanced features
testCapitalizedIncome(loanId);
testDownPayment(loanId);
testAdvancedPaymentAllocation(loanId);

```

Writing Integration Tests

Test Development Guidelines

1. Create Test Class

Create a new test class in `integration-tests/src/test/java/org/apache/fineract/integrationtests/`:

```
package org.apache.fineract.integrationtests;

import org.apache.fineract.integrationtests.common.Utils;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;

public class MyNewFeatureIntegrationTest extends BaseLoanIntegrationTest {

    @BeforeEach
    public void setup() {
        Utils.initializeRESTAssured();
        this.requestSpec = new RequestSpecBuilder()
            .setContentType(ContentType.JSON)
            .build();
        this.requestSpec.header("Authorization", "Basic " +
            Utils.loginIntoServerAndGetBase64EncodedAuthenticationKey());
        this.responseSpec = new ResponseSpecBuilder()
            .expectStatusCode(200)
            .build();
    }

    @Test
    public void testMyNewFeature() {
        // Test implementation
    }
}
```

2. Use Helper Classes

Leverage existing helper classes:

```
// Client operations
ClientHelper clientHelper = new ClientHelper(requestSpec, responseSpec);
Long clientId = clientHelper.createClient(...);

// Loan operations
LoanTransactionHelper loanHelper = new LoanTransactionHelper(requestSpec,
responseSpec);
Long loanId = loanHelper.applyLoan(...);

// Account operations
```

```

AccountHelper accountHelper = new AccountHelper(requestSpec, responseSpec);

// Business date operations
BusinessDateHelper businessDateHelper = new BusinessDateHelper();
businessDateHelper.updateBusinessDate(...);

// COB operations
InlineLoanCOBHelper cobHelper = new InlineLoanCOBHelper(requestSpec, responseSpec);
cobHelper.executeInlineCOB(loanId);

```

3. Follow Best Practices

- **Self-Contained Tests:** Each test should be independent
- **Clear Setup:** Use `@BeforeEach` for test initialization
- **Date Management:** Use `runAt()` for consistent date-based testing
- **Comprehensive Verification:** Verify transactions, schedules, and accounting
- **Helper Methods:** Use provided helper classes rather than direct API calls
- **Error Testing:** Test both positive and negative scenarios
- **Cleanup:** Clean up test data when necessary

4. Test Complex Scenarios

```

@Test
public void testLoanWithMultipleDisbursements() {
    runAt("01 January 2024", () -> {
        // Create client
        Long clientId = clientHelper.createClient(ClientHelper
            .defaultClientCreationRequest());

        // Create multi-disbursement loan product
        Long productId = createMultiDisbursementProduct();

        // Apply for loan
        Long loanId = applyAndApproveProgressiveLoan(clientId, productId,
            BigDecimal.valueOf(1000), 12, BigDecimal.valueOf(10));

        // First disbursement
        disburseLoan(loanId, BigDecimal.valueOf(500), "01 January 2024");

        // Verify first disbursement
        verifyTransactions(loanId,
            transaction(500.0, "Disbursement", "01 January 2024")
        );
    });

    runAt("15 January 2024", () -> {
        // Second disbursement
    });
}

```

```

        disburseLoan(loanId, BigDecimal.valueOf(500), "15 January 2024");

        // Verify both disbursements
        verifyTransactions(loanId,
            transaction(500.0, "Disbursement", "01 January 2024"),
            transaction(500.0, "Disbursement", "15 January 2024")
        );

        // Verify outstanding balance
        verifyOutstanding(loanId, BigDecimal.valueOf(1000));
    });
}

```

5. Run and Verify

```

# Run your new test
./gradlew :integration-tests:test --tests MyNewFeatureIntegrationTest

# Run with verbose output for debugging
./gradlew :integration-tests:test --tests MyNewFeatureIntegrationTest --info

# Run specific test method
./gradlew :integration-tests:test --tests MyNewFeatureIntegrationTest.testMyNewFeature

```

Troubleshooting

Common Test Failures

Connection Issues

Symptom: Tests fail with connection refused errors

Solutions:

```

# Verify Fineract is running
curl -k https://localhost:8443/actuator/health

# Check if port is available
netstat -tulpn | grep 8443

# Check Fineract logs (logs go to console/stdout)
# If running in background with output redirection:
# tail -f build/bootRun.log

# Restart Fineract if needed
pkill -f bootRun
./gradlew bootRun &

```

Authentication Failures

Symptom: Tests fail with 401 or 403 errors

Solutions:

```
# Check default credentials
mysql -u root -pmysql fineract_default -e \
  "SELECT username, password FROM m_appuser WHERE username = 'mifos';"

# Reset credentials if needed
mysql -u root -pmysql fineract_default -e \
  "UPDATE m_appuser SET password = '5jdQ3dNQXHPzCuBbZVdQZ2XnVlPc3l2l' \
  WHERE username = 'mifos';"

# Verify connection settings
echo "Protocol: ${BACKEND_PROTOCOL:-https}"
echo "Host: ${BACKEND_HOST:-localhost}"
echo "Port: ${BACKEND_PORT:-8443}"
```

Data Inconsistency

Symptom: Tests fail due to unexpected data state

Solutions:

```
# Reset database
mysql -u root -pmysql -e "DROP DATABASE fineract_default;"
mysql -u root -pmysql -e "DROP DATABASE fineract_tenants;"

# Recreate databases
./gradlew createDB -PdbName=fineract_tenants
./gradlew createDB -PdbName=fineract_default

# Restart Fineract
pkill -f bootRun
./gradlew bootRun &
```

Test Timeout

Symptom: Tests hang or timeout

Solutions:

```
# Increase test timeout
./gradlew :integration-tests:test -Dtest.timeout=600

# Check for database locks
mysql -u root -pmysql fineract_default -e "SHOW PROCESSLIST;"
```

```
# Kill long-running queries
mysql -u root -pmysql fineract_default -e "KILL <process_id>;"
```

Memory Issues

Symptom: OutOfMemoryError during test execution

Solutions:

```
# Increase heap size
./gradlew :integration-tests:test -Xmx4g -Xms2g

# Run fewer tests in parallel
./gradlew :integration-tests:test --max-workers=2

# Clean build directory
./gradlew clean
```

Debugging Tips

Enable Detailed Logging

```
# Run with debug output
./gradlew :integration-tests:test --debug

# Run with info level
./gradlew :integration-tests:test --info

# Save output to file
./gradlew :integration-tests:test --info > test-output.log 2>&1
```

Check Test Reports

After test execution, detailed reports are available:

```
# HTML report
integration-tests/build/reports/tests/test/index.html

# XML reports (for CI/CD)
integration-tests/build/test-results/test/

# Gradle scan (upload for detailed analysis)
```

Generate Gradle build scan:

```
./gradlew :integration-tests:test --scan
```

Database State Verification

```
# Check loan status
mysql -u root -pmysql fineract_default -e \
  "SELECT id, account_no, loan_status_id, principal_amount \
  FROM m_loan ORDER BY id DESC LIMIT 10;"

# Check transactions
mysql -u root -pmysql fineract_default -e \
  "SELECT loan_id, transaction_type_enum, amount, transaction_date \
  FROM m_loan_transaction WHERE loan_id = <loan_id>;"

# Check journal entries
mysql -u root -pmysql fineract_default -e \
  "SELECT entry_date, account_id, type_enum, amount \
  FROM acc_gl_journal_entry WHERE loan_id = <loan_id>;"

# Check configurations
mysql -u root -pmysql fineract_default -e \
  "SELECT name, enabled FROM c_configuration \
  WHERE name LIKE '%business%' OR name LIKE '%enable%';"
```

API Response Debugging

Add logging to test methods:

```
Response response = loanHelper.applyLoan(...);
System.out.println("Response: " + response.asString());
System.out.println("Status Code: " + response.getStatusCode());

// Or use logger
log.info("Response: {}", response.asString());
```

Isolate Failing Tests

```
# Run only the failing test
./gradlew :integration-tests:test --tests FailingTest --info

# Run with rerun-tasks option
./gradlew :integration-tests:test --tests FailingTest --rerun-tasks

# Run with fail-fast to stop on first failure
./gradlew :integration-tests:test --fail-fast
```


Best Practices

Test Organization

- Extend appropriate base classes (`BaseLoanIntegrationTest`, `IntegrationTest`)
- Use descriptive test method names that explain what is being tested
- Group related tests in the same test class
- Use `@BeforeEach` for setup and `@AfterEach` for cleanup
- Follow existing naming conventions

Test Isolation

- Each test should be independent and not rely on other tests
- Create fresh test data for each test
- Clean up test data after test execution
- Use unique identifiers to avoid conflicts
- Don't share mutable state between tests

Performance Optimization

- Reuse Fineract instance across test runs
- Use `runAt()` for efficient date management
- Minimize unnecessary API calls
- Use bulk operations when appropriate
- Consider parallel execution for independent tests
- Run subset of tests during development

Code Quality

- Follow existing code patterns and conventions
- Use helper methods instead of duplicating code
- Add comments for complex business logic
- Verify both positive and negative scenarios
- Include edge cases in test coverage
- Document test assumptions and prerequisites

Comprehensive Verification

- Always verify transaction creation
- Check accounting journal entries
- Validate repayment schedules

- Verify loan status transitions
- Test charge applications
- Validate business date handling
- Check error messages for validation failures

Maintenance

- Update tests when API changes
- Remove deprecated test methods
- Keep test data realistic
- Document complex test scenarios
- Review and refactor tests regularly
- Keep tests aligned with current best practices

Fineract Documentation Guide

TBD

File and Folder Layout

The general rules are

- keep things as flat as possible (avoid sub-folders as much as possible)
- DRY (don't repeat yourself): don't copy and paste code pieces, use AsciiDoc's include feature and reference files/-sections from the project folder
- images are located in `fineract-doc/src/docs/en/images` (or sub-folders)
- diagrams are located in `fineract-doc/src/docs/en/diagrams` (or sub-folders)
- specific chapters are located in `fineract-doc/src/docs/en/chapters`
- every chapter has its own folder and at least one `index.adoc` file
- it's recommended to keep the chapters flat (i. e. no sub-folders in the chapter folders)
- it's recommended to create one file per chapter section; like that you can re-arrange sections very easily in the `index.adoc` file



These rules are not entirely set in stone and could be modified if necessary. If you see any issues then please report them on the [mailing list](#) or [open a Jira ticket](#).

AsciiDoc

Cheatsheet

You can find the definitive manual on AsciiDoc syntax at [AsciiDoc documentation](#). To help people get started, however, here is a simpler cheat sheet.

AsciiDoc vs AsciiDoctor (format vs tool)

When we refer to *AsciiDoc* then we mean the language or format that this documentation is written in. AsciiDoc is a markup language similar to Markdown (but more powerful and expressive) designed for technical documentation. You don't need necessarily any specialized editors or tools to write your documentation in AsciiDoc, a plain text editor will do, but there are plenty of choices that give you a better experience (in this documentation we describe the basic usage with AsciiDoc plugins for IntelliJ, Eclipse and VSCode).

AsciiDoctor on the other hand is the command line tool we use to transform documents written in AsciiDoc into HTML and PDF (Epub3 and Docbook are also available). There are three variants available:

- AsciiDoctor (written in Ruby)
- AsciiDoctor.js (written in JavaScript, often used for browser previews)

- AsciiDoctorJ (Java lib that integrates the Ruby implementation via JRuby, e. g. the AsciiDoctor Gradle plugin is based on that)



Sometimes you will still find documentation related to the original incarnation of AsciiDoc/tor (written in Python). The format evolved quite a bit since then and the tools try to maintain a certain degree of backward compatibility, but there is no guarantee. We prefer to use the latest language specs as documented [here](#).

Basic AsciiDoc Syntax

Bold

Put asterisks around text to make it **bold**.



More info at docs.asciidoctor.org/asciidoc/latest/text/bold

Italics

Use underlines on either side of a string to put text into *italics*.



More info at docs.asciidoctor.org/asciidoc/latest/text/italic

Headings

Equal signs (=) are used for heading levels. Each equal sign is a level. Each page can **only** have one top level (i.e., only one section with a single =).

Levels should be appropriately nested. During the build, validation occurs to ensure that level 3s are preceded by level 2s, level 4s are preceded by level 3s, etc. Including out-of-sequence heading levels (such as a level 3 then a level 5) will not fail the build, but will produce an error.



More info at docs.asciidoctor.org/asciidoc/latest/sections/titles-and-levels/

Code Examples

Use backticks ` for text that should be monospaced, such as code or a class name in the body of a paragraph.



More info at docs.asciidoctor.org/asciidoc/latest/text/monospace/

Longer code examples can be separated from text with **source** blocks. These allow defining the syntax being used so the code is properly highlighted.

Example Source Block

```
[source,xml]
<field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false" />
```

If your code block will include line breaks, put 4 hyphens (----) before and after the entire block.



More info at docs.asciidoctor.org/asciidoc/latest/verbatim/source-blocks/

Source Block Syntax Highlighting

The HTML output uses Rouge to add syntax highlighting to code examples. This is done by adding the language of the code block after the `source`, as shown in the above example source block (`xml` in that case).

Rouge has a long selection of lexers available. You can see the full list at github.com/rouge-ruby/rouge/wiki/List-of-supported-languages-and-lexers. Use one of the valid short names to get syntax highlighting for that language.

Ideally, we will have an appropriate lexer to use for all source blocks, but that's not possible. When in doubt, choose `text`, or leave it blank.

Importing Code Snippets from Other Files

The build system has the ability to "include" snippets located in other files — even non-AsciiDoc files such as `*.java` source code files.

We've configured a global attribute called `{rootdir}` that you can use to reference these files consistently from Fineract's project root folder.

Snippets are bounded by `tag` comments placed at the start and end of the section you would like to import. Opening tags look like: `// tag::snippetName[]`. Closing tags follow the format: `// end::snippetName[]`.

Snippets can be inserted into an `.adoc` file using an `include` directive, following the format: `include::{rootdir}/<directory-under-root-folder>/<file-name>[tag=snippetName]`.



You could also use relative paths to reference include files, but it is preferred to always use the root folder as a starting point. Like this you can be sure that the preview in your editor of choice works.

For example, if we wanted to highlight a specific section of the following Cucumber test definition (more on that in section Cucumber Testing) `ClasspathDuplicatesStepDefinitions.java` file located under `fineract-provider/src/test/java/org/apache/fineract/infrastructure/classpath/`.

```
[source,java,indent=0]
----
include::{rootdir}/fineract-
provider/src/test/java/org/apache/fineract/infrastructure/classpath/ClasspathDuplicate
sStepDefinitions.java[tag=then]
----
```

For more information on the `include` directive, see the documentation at docs.asciidoctor.org/asciidoc/latest/directives/include.

Block Titles

Titles can be added to most blocks (images, source blocks, tables, etc.) by simply prefacing the title with a period (.). For example, to add a title to the source block example above:

```
.Example ID field
[source,xml]
<field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false" />
```



More info at docs.asciidoctor.org/asciidoc/latest/blocks/add-title

Links

Link to Sites on the Internet

When converting content to HTML, Asciidoctor will automatically render many link types (such as [http:](#) and [mailto:](#)) without any additional syntax. However, you can add a name to a link by adding the URI followed by square brackets:

```
http://fineract.apache.org/[Fineract Website]
```



More info at docs.asciidoctor.org/asciidoc/latest/macros/url-macro

Link to Other Pages/Sections of the Guide

A warning up front, linking to other pages can be a little painful. There are slightly different rules depending on the type of link you want to create, and where you are linking from. The build process includes a validation for *internal* or *inter-page* links, so if you can build the docs locally, you can use that to verify you constructed your link properly. With all the below examples, you can add text to display as the link title by putting the display text in brackets after the link, as in:

```
xref:indexing-guide:schema-api.adoc#modify-the-schema[Modify the Schema]
```

You can also use the title of the Page or Section you are linking to by using an empty display text. This is useful in case the title of the page or section changes. In that case you won't need to change the display text for every link that refers to that page/section.

See an example below:

```
xref:indexing-guide:schema-api.adoc#modify-the-schema[]
```

Link to a Section on the Same Page

To link to an anchor (or section title) on the *same page*, you can simply use double angle brackets

(<< >>) around the anchor/heading/section title you want to link to. Any section title (a heading that starts with equal signs) automatically becomes an anchor during conversion and is available for deep linking.

Example

If I have a section on a page that looks like this (from `process.adoc`):

```
== Steps

Common parameters for all steps are:
```

To link to this section from another part of the same `process.adoc` page, I simply need to put the section title in double angle brackets, as in:

```
See also the <<Steps>> section.
```

The section title will be used as the display text; to customize that add a comma after the the section title, then the text you want used for display.



More info at docs.asciidoctor.org/asciidoc/latest/macros/xref/#internal-cross-references

Link to a Section with an Anchor ID

When linking to any section (on the same page or another one), you must also be aware of any pre-defined anchors that may be in use (these will be in double brackets, like `[[]]`).

When the page is converted, those will be the references your link needs to point to.

Example

Take this example from `configsets-api.adoc`:

```
[[configsets-create]]
== Create a ConfigSet
```

To link to this section, there are two approaches depending on where you are linking from:

- From the same page, simply use the anchor name: `<<configsets-create>>`.
- From another page, use the page name and the anchor name: `xref:configuration-guide:configsets-api.adoc#configsets-create[]`.

Link to Another Page

To link to *another page* or a section on another page, you must refer to the full filename and refer to the section you want to link to.

When you want to refer the reader to another page without deep-linking to a section, Asciidoctor

allows this by merely omitting the # and section id.

Example

To construct a link to the `process.adoc` page, we need to refer to the file name (`process.adoc`), as well as the module that the file resides in (`release/`).

It's preferred to also always use the page name to give the reader better context for where the link goes.

As in:

For more about upgrades, see `xref:release:process.adoc[Fineract Release Process]`.

Link to Another Page in the same folder

If the page that contains the link and the page being linked to reside in the same module, there is no need to include the module name after `xref:`

Example

To construct a link to the `process-step01.adoc` page from `process.adoc` page, we do not need to include the module name because they both reside in the `upgrade-notes` module.

For more information on the first step of the release process, see the section `\xref:process-step01.adoc[]`.

Link to a Section on Another Page

Linking to a section is the same conceptually as linking to the top of a page, you just need to take a little extra care to format the anchor ID in your link reference properly.

When you link to a section on another page, you must make a simple conversion of the title into the format of the section ID that will be created during the conversion. These are the rules that transform the sections:

Example

TBD



More info at docs.asciidoctor.org/asciidoc/latest/macros/inter-document-xref

Ordered and Unordered Lists

AsciiDoc supports three types of lists:

- Unordered lists
- Ordered lists
- Labeled lists

Each type of list can be mixed with the other types. So, you could have an ordered list inside a

labeled list if necessary.

Unordered Lists

Simple bulleted lists need each line to start with an asterisk (*). It should be the first character of the line, and be followed by a space.



More info at docs.asciidoctor.org/asciidoc/latest/lists/unordered

Ordered Lists

Numbered lists need each line to start with a period (.). It should be the first character of the line, and be followed by a space. This style is preferred over manually numbering your list.



More info at docs.asciidoctor.org/asciidoc/latest/lists/ordered

Description Lists

These are like question & answer lists or glossary definitions.

Each line should start with the list item followed by double colons (::), then a space or new line. Labeled lists can be nested by adding an additional colon (such as :::, etc.). If your content will span multiple paragraphs or include source blocks, etc., you will want to add a plus sign (+) to keep the sections together for your reader.



We prefer this style of list for parameters because it allows more freedom in how you present the details for each parameter. For example, it supports ordered or unordered lists inside it automatically, and you can include multiple paragraphs and source blocks without trying to cram them into a smaller table cell.



More info at docs.asciidoctor.org/asciidoc/latest/lists/description

Images

There are two ways to include an image: inline or as a block. Inline images are those where text will flow around the image. Block images are those that appear on their own line, set off from any other text on the page. Both approaches use the `image` tag before the image filename, but the number of colons after `image` define if it is inline or a block. Inline images use one colon (`image:`), while block images use two colons (`image::`). Block images automatically include a caption label and a number (such as **Figure 1**). If a block image includes a title, it will be included as the text of the caption. Optional attributes allow you to set the alt text, the size of the image, if it should be a link, float and alignment. We have defined a global attribute `{imagesdir}` to standardize the location for all images (`fineract-doc/src/docs/en/images`).



More info at docs.asciidoctor.org/asciidoc/latest/macros/images

Tables

Tables can be complex, but it is pretty easy to make a basic table that fits most needs.

Basic Tables

The basic structure of a table is similar to Markdown, with pipes (|) delimiting columns between rows:

```
|===  
| col 1 row 1 | col 2 row 1 |  
| col 1 row 2 | col 2 row 2 |  
|===
```

Note the use of `|===` at the start and end. For basic tables that's not exactly required, but it does help to delimit the start and end of the table in case you accidentally introduce (or maybe prefer) spaces between the rows.

Header Rows

To add a header to a table, you need only set the `header` attribute at the start of the table:

```
[options="header"]  
|===  
| header col 1 | header col 2 |  
| col 1 row 1 | col 2 row 1 |  
| col 1 row 2 | col 2 row 2 |  
|===
```

Defining Column Styles

If you need to define specific styles to all rows in a column, you can do so with the attributes.

This example will center all content in all rows:

```
[cols="2*^" options="header"]  
|===  
| header col 1 | header col 2 |  
| col 1 row 1 | col 2 row 1 |  
| col 1 row 2 | col 2 row 2 |  
|===
```

Alignments or any other styles can be applied only to a specific column. For example, this would only center the last column of the table:

```
[cols="2*,^" options="header"]  
|===  
| header col 1 | header col 2 |  
| col 1 row 1 | col 2 row 1 |  
| col 1 row 2 | col 2 row 2 |  
|===
```



Many more examples of formatting:

- Columns: docs.asciidoctor.org/asciidoc/latest/tables/add-columns/
- Cells and rows: docs.asciidoctor.org/asciidoc/latest/tables/add-cells-and-rows/

More Options

Tables can also be given footer rows, borders, and captions. You can determine the width of columns, or the width of the table as a whole.

CSV or DSV can also be used instead of formatting the data in pipes.



More info at docs.asciidoctor.org/asciidoc/latest/tables/build-a-basic-table/

Admonitions (Notes, Warnings)

AsciiDoc supports several types of callout boxes, called "admonitions":

- NOTE
- TIP
- IMPORTANT
- CAUTION
- WARNING

It is enough to start a paragraph with one of these words followed by a colon (such as **NOTE:**). When it is converted to HTML, those sections will be formatted properly - indented from the main text and showing an icon inline.

You can add titles to admonitions by making it an admonition block. The structure of an admonition block is like this:

```
.Title of Note
[NOTE]
====
Text of note
=====
```

In this example, the type of admonition is included in square brackets (**[NOTE]**), and the title is prefixed with a period. Four equal signs give the start and end points of the note text (which can include new lines, lists, code examples, etc.).



More info at docs.asciidoctor.org/asciidoc/latest/blocks/admonitions/

STEM Notation Support

We have set up the Ref Guide to be able to support STEM notation whenever it's needed.

The [AsciiMath](#) syntax is supported by default, but LaTeX syntax is also available.

To insert a mathematical formula inline with your text, you can simply write:

```
stem:[a//b]
```

MathJax.js will render the formula as proper mathematical notation when a user loads the page. When the above example is converted to HTML, it will look like this to a user: a/b

To insert LaTeX, preface the formula with `latexmath` instead of `stem`:

```
latexmath:[tp \leq 1 - (1 - \sin^{\{rows\}})^{\{bands\}}]
```

Long formulas, or formulas which should to be set off from the main text, can use the block syntax prefaced by `stem` or `latexmath`:

```
[stem]
++++
sqrt(3x-1)+(1+x)^2 < y
++++
```

or for LaTeX:

```
[latexmath]
++++
[tp \leq 1 - (1 - \sin^{\{rows\}})^{\{bands\}}]
++++
```



More info at docs.asciidoctor.org/asciidoc/latest/stem/stem

Diagrams

TBD



docs.asciidoctor.org/diagram-extension/latest

PlantUML Cheatsheet

TBD



plantuml.com

C4 Cheatsheet

TBD



c4model.com and github.com/plantuml-stdlib/C4-PlantUML

Archimate Cheatsheet

TBD



www.opengroup.org/archimate-forum/archimate-overview and plantuml.com/archimate-diagram

Vega Cheatsheet

TBD



vega.github.io/vega-lite/

Editor

TBD

IntelliJ IDEA

Asciidoc

See: plugins.jetbrains.com/plugin/7391-asciidoc

PlantUML

See: plugins.jetbrains.com/plugin/7017-plantuml-integration

Eclipse

Asciidoc

See: marketplace.eclipse.org/content/asciidoctor-editor

PlantUML

See: plantuml.com/eclipse

VSCode

Asciidoc

See: marketplace.visualstudio.com/items?itemName=asciidoctor.asciidoctor-vscode

PlantUML

See: marketplace.visualstudio.com/items?itemName=jebbs.plantuml

Antora

See: antora.org/

TBD

Releases

This chapter explains how we make the [source code](#) into an official release available on fineract.apache.org.

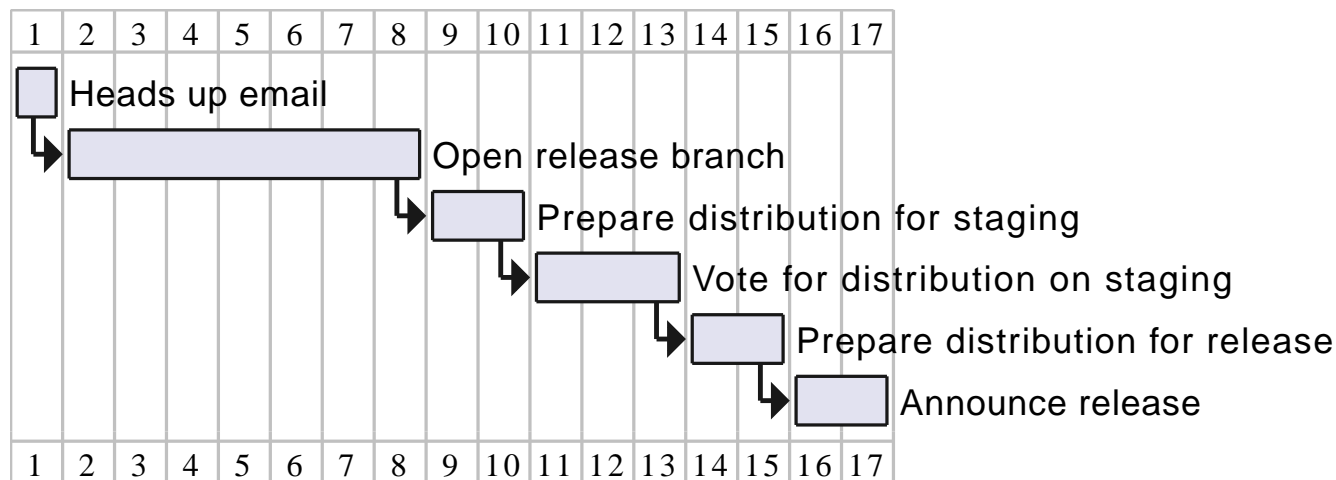


Figure 4. Release Schedule

Configuration

Before you can start using the Fineract release plugin to create releases you have to configure and setup a couple of things first.

- All official communication concerning releases happens on the [mailing list](#). Every release manager needs to be a member of and engaging on the mailing list for credibility.
- Make sure you have edit permissions on the [Apache Confluence Wiki](#)
- You need full permissions on [Apache JIRA](#) to be able to move issues to the next release
- Git committer privileges to be allowed to create tags and the release branch, and to upload release candidates to ASF's distribution dev (staging) area
- Familiarity with building Fineract locally and creating release distributions is required
- You need to be a member of the PMC to be able to upload release artifacts to ASF's distribution release area; this task can be delegated though
- A general Familiarity with PGP/GPG is recommended (at least to setup your keypairs), but the release plugin does most of the heavy lifting
- Make sure to read the release plugin documentation for troubleshooting
- Read, understand, and follow everything listed at www.apache.org/dev/#releases. It helps to pair with someone who has previously done a release.

Secrets

TBD

Infrastructure Team

A couple of secrets for third party services are automatically configured by the infrastructure team at The Apache Foundation for the Fineract Github account. At the moment this includes environment variables for:

- Github token (e. g. to publish Github Pages, use the Github API in Github Actions)
- Docker Hub token (to publish our Docker images)
- Sonar Cloud token (for our code quality reports)

See also:

- infra.apache.org/github-pages.html
- cwiki.apache.org/confluence/display/INFRA/Github+Actions+to+DockerHub
- github.com/apache/jmeter-site-preview
- github.com/apache/fineract-site
- github.com/apache/systemds-website/blob/main/.asf.yaml

Lastpass

It seems that Apache has some kind of org account or similar. Popped up a couple of times in the infrastructure documentation.

TBD

1Password

Other Fineract development related secrets, e. g. for deployments of demo systems on Google Cloud, AWS etc. are managed in a team account at 1Password. At the moment the following committers are members of the 1Password team account:

- [Ed Cable](#)
- [Michael Vorburger](#)
- [Petri Tuomola](#)
- [Arnold Galovics](#)
- [Aleksandar Vidakovic](#)



If you need access or have any questions related to those secrets then please reach out to one of the team members.

GPG

Generate GPG key pairs if you don't already have them and publish them. Please use your Apache email address when creating your GPG keypair. If you already have configured GPG and associated your keypair with a non-Apache email address then please consider creating a separate one just for all things related to Fineract (or Apache in general).

Instructions:

1. Check your GPG version:

Input GPG version

```
gpg --version
```

Output GPG version

```
gpg (GnuPG) 2.4.4
libgcrypt 1.10.3
Copyright (C) 2024 g10 Code GmbH
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/aleks/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```



The insecure hash algorithm SHA1 is still supported in version 2.4.4. SHA1 is obsolete and you don't want to use it to generate your signature.

2. Generate your GPG key pair:

Input generate GPG key pair

```
gpg --full-gen-key
```

Output generate GPG key pair (step 1: key type selection)

```
Please select what kind of key you want:
(1) RSA and RSA
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
(9) ECC (sign and encrypt) *default*
(10) ECC (sign only)
(14) Existing key from card
Your selection?
```

Choose the default.

Output generate GPG key pair (step 2: elliptic curve selection)

```
Please select which elliptic curve you want:
(1) Curve 25519 *default*
(4) NIST P-384
(6) Brainpool P-256
Your selection?
```

Again, choose the default.

Output generate GPG key pair (step 3: validity selection)

```
Please specify how long the key should be valid.
  0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 2y
```

2 years for the validity of your keys should be fine. You can always update the expiration time later on.

Output generate GPG key pair (step 4: confirmation)

```
Key expires at Sun 16 Apr 2024 08:10:24 PM UTC
Is this correct? (y/N) y
```

Confirm if everything is correct.

Output generate GPG key pair (step 5: provide user details)

```
GnuPG needs to construct a user ID to identify your key.

Real name: Aleksandar Vidakovic
Email address: aleks@apache.org
Comment:
```

Provide your user details for the key. This is important because this information will be included in our key. It's one way of indicating who is owner of this key. The email address is a unique identifier for a person. You can leave Comment blank.

Output generate GPG key pair (step 6: user ID selection)

```
You selected this USER-ID:
"Aleksandar Vidakovic <aleks@apache.org>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 0
```

Select **Okay**.

After the selection of your user ID GPG will ask for a passphrase to protect your private key. Maybe time to open your password manager and generate a secure one and save it in your vault. Once you've confirmed your password GPG will start to generate your keys.



Don't lose your private key password. You won't be able to unlock and use your private key without it.

Output generate GPG key pair (step 7: gpg key pair generation)

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

Generating the GPG keys will take a while.

Output generate GPG key pair (step 8: gpg key pair finished)

```
gpg: key 7890ABCD marked as ultimately trusted ①
gpg: directory '/home/aleks/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/aleks/.gnupg/openpgp-
revocs.d/ABCDEFGHJKLMNOPQRSTUVWXYZ1234567890ABCD.rev' ②
public and secret key created and signed.

gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: PGP
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2024-04-16
pub ed25519/7890ABCD 2022-04-16 [SC] [expires: 2024-04-16] ③
Key fingerprint = ABCD EFGH IJKL MNOP QRST UVWX YZ12 3456 7890 ABCD ④
uid [ultimate] Aleksandar Vidakovic <aleks@apache.org> ⑤
sub cv25519/4FGHIJ56 2022-04-16 [E] [expires: 2024-04-16] ⑥
```

- ① GPG created a unique identifier in hexadecimal format for your public key. When someone wants to download your public key, they can refer to it either with your email address or this hex value. The hex value is sometimes prefixed with **0x** as is commonly done with hexadecimal numbers.
- ② GPG created a revocation certificate and its directory. If your private key is compromised, you need to use your revocation certificate to revoke your key.
- ③ The public key uses the Ed25519 ECC (Elliptic Curve Cryptography) algorithm and shows the expiration date of 16 Apr 2024. The public key ID **7890ABCD** matches the last 8 characters of key fingerprint. The **[SC]** indicates this key is used to sign (prove authorship) and certify (issue subkeys for encryption, signature and authentication operations).
- ④ The key fingerprint (**ABCD EFGH IJKL MNOP QRST UVWX YZ12 3456 7890 ABCD**) is a hash of your public key.

⑤ Your name and your email address are shown with information about the subkey.

⑥ This Curve25519 subkey is used for encryption.

Now you can find that there are two files created under `~/.gnupg/private-keys-v1.d/` directory. These two files are binary files with `.key` extension.

3. Export your public key:

```
gpg --armor --export aleks@apache.org > pubkey.asc
```

4. Export Your Private Key:

```
gpg --export-secret-keys --armor aleks@apache.org > privkey.asc
```

5. Protect Your Private Key and Revocation Certificate

Your private key should be kept in a safe place, like an encrypted flash drive. Treat it like your house key. Only you can have it and don't lose it. And you must remember your passphrase, otherwise you can't unlock your private key.

You should protect your revocation certificate. Anyone in possession of your revocation certificate, could immediately revoke your public/private key pair and generate fake ones.



Please contact a PMC member to add your GPG public key in Fineract's Subversion repository. This is necessary to be able to validate published releases.

6. Upload your GPG key to a keyserver:

```
gpg --send-keys 0xYZ1234567890ABCD
```

Before doing this, make sure that your default keyserver is `hkp://keyserver.ubuntu.com/`. You can do this by changing the default keyserver in `~/.gnupg/dirmngr.conf`:

```
keyserver hkp://keyserver.ubuntu.com/
```

Alternatively you can provide the keyserver with the send command:

```
gpg --keyserver keyserver.ubuntu.com --send-keys 0xYZ1234567890ABCD
```

Another option to publish your key is to submit an armored public key directly at keyserver.ubuntu.com/. You can create the necessary data with this command by providing the email address that you used when you created your key pair:

```
gpg --armor --export aleks@apache.org
```

Output:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
  
mQINBF8iGq0BEADGRqeSs0oNDc1sV3L9sQ34KhmoQrACnMYGztx33TD98aWplul+  
jm8uGtMmBus4DJJJap1bVQ1oMehw2mscmDHpfJjLNZ/q+vUqbExx1/CER7XvLryN  
<--- snip --->  
2nHBuBftxDRpDHQ+O5XYwSDSTDMmthPjx0vJGBH4K1k08XK99e01A6/oYLV2SMKp  
gXXeWjafxBmHT1cM8hoBZBYzgTu9nK5UnllWunfaHXiCBG4oQQ==  
=85/F  
-----END PGP PUBLIC KEY BLOCK-----
```

Email

Official communication related to releases needs to be done with an Apache email address. The Apache Foundation doesn't provide any real email inboxes anymore and just relays emails to your configured private account (GMail etc.).



At the moment we are supporting only GMail accounts. Please let us know if you have other configuration recipes for other email providers.

GMail

You can configure your GMail account and add another profile to use the Apache relay server if you need to send official messages. Please follow these instructions:

TBD.

See also: [Send mail from another address without "on behalf of"](#)

To be able to send emails via SMTP with your GMail account you probably need to create an app password. Please follow these instructions:

1. Go to your Google Account.
2. Select Security.
3. Under "Signing in to Google," select App Passwords. You may need to sign in. If you don't have this option, it might be because:
4. 2-Step Verification is not set up for your account.
5. 2-Step Verification is only set up for security keys.
6. Your account is through work, school, or other organization.
7. You turned on Advanced Protection.
8. At the bottom, choose Select app and choose the app you're using and then Select device and

choose the device you're using and then Generate.

9. Follow the instructions to enter the App Password. The App Password is the 16-character code in the yellow bar on your device.
10. Tap Done.

See also: [Google Support: Sign in with App Passwords](#) for more details.

Gradle

TBD

User Properties

There are a couple of properties that contain committer/release manager related secrets. Please add the following properties to your personal global Gradle properties (you will find them at `~/.gradle/gradle.properties` in your home folder).

```
fineract.config.gnupg.keyName=ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ABCD①
fineract.config.gnupg.password=*****
fineract.config.gnupg.publicKeyring=~/.gnupg/pubring.kbx②
fineract.config.gnupg.secretKeyring=~/.gnupg/secring.gpg
fineract.config.smtp.username=aleks@gmail.com ③
fineract.config.smtp.password=*****
fineract.config.name=Aleksandar Vidakovic
fineract.config.email=aleks@apache.org
fineract.config.username=aleks ④
fineract.config.password=*****
```

- ① Make sure you use the full GPG key name (you can list yours via `gpg --list-secret-keys --keyid -format=long`)
- ② GnuPG has its own kbx format to store the public key ring. At the moment we are only supporting this format
- ③ Currently we only have instructions for GMail
- ④ Apache committer credentials



Never add any personal secrets in the project `gradle.properties`. Double check that you are not accidentally committing them to Git!

Release Plugin

Creating Apache Fineract releases was a very manual and tedious procedure before we created the Gradle release plugin. It was easy - even with documentation - to forget a detail. Some ideas are borrowed from the excellent [JReleaser](#) tool. Unfortunately at the moment we can't use it for the full release process. Being an Apache project we have certain requirements that are not fully covered by [JReleaser](#).

```

config {
    username = "${findProperty('fineract.config.username')}}"
    password = "${findProperty('fineract.config.password')}}"

    doc {
        url = 'git@github.com:apache/fineract-site.git'
        directory = "${System.getProperty("java.io.tmpdir")}/fineract-site"
        branch = "asf-site"
    }
    git {
        dir = "${projectDir.absolutePath}/.git"
        sections = [
            [
                section: "user",
                name: "name",
                value: "${findProperty('fineract.config.name')}}",
            ],
            [
                section: "user",
                name: "email",
                value: "${findProperty('fineract.config.email')}}",
            ],
            [
                section: "user",
                name: "signingkey",
                value: "${findProperty('fineract.config.gnupg.keyName')}}",
            ],
            [
                section: "commit",
                name: "gpgsign",
                value: "true",
            ],
        ]
    }
    template {
        templateDir = "${projectDir}/buildSrc/src/main/resources"
    }
    gpg {
        keyName = "${findProperty('fineract.config.gnupg.keyName')}}"
        publicKeyring = "${findProperty('fineract.config.gnupg.publicKeyring')}}"
        secretKeyring = "${findProperty('fineract.config.gnupg.secretKeyring')}}"
        password = "${findProperty('fineract.config.gnupg.password')}}"
    }
    smtp {
        host = 'smtp.gmail.com'
        username = "${findProperty('fineract.config.smtp.username')}}"
        password = "${findProperty('fineract.config.smtp.password')}}"
        tls = true
        ssl = true
    }
}

```

```

    }
    subversion {
        username = "${findProperty('fineract.config.username')}}"
        password = "${findProperty('fineract.config.password')}}"
        revision = 'HEAD'
    }
    jira {
        url = 'https://issues.apache.org/jira/rest/api/2/'
        username = "${findProperty('fineract.config.username')}}"
        password = "${findProperty('fineract.config.password')}}"
    }
    confluence {
        url = 'https://cwiki.apache.org/confluence/rest/api/'
        username = "${findProperty('fineract.config.username')}}"
        password = "${findProperty('fineract.config.password')}}"
    }
}

```

Release Process



Fineract release plugin Gradle tasks are experimental and incomplete.

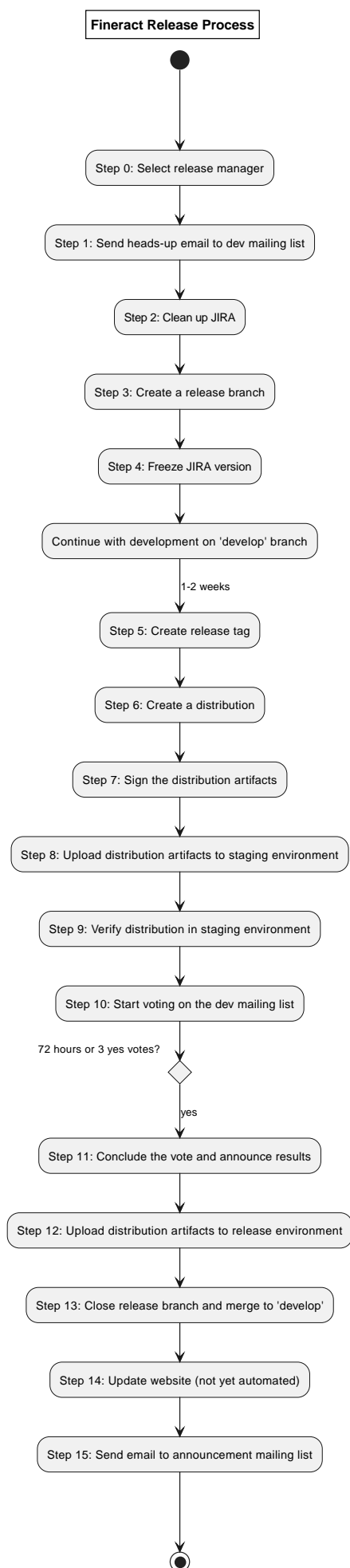


Figure 5. Release Process Diagram

Step 1: Heads-Up Email

Description

The RM should, if one doesn't already exist, first create a new release umbrella issue in JIRA. This issue is dedicated to tracking (a summary of) any discussion related to the planned new release. An example of such an issue is [FINERACT-873](#).

The RM then creates a list of resolved issues & features through an initial check in JIRA for already resolved issues for the release, and then setup a timeline for release branch point. The time for the day the issue list is created to the release branch point must be at least two weeks in order to give the community a chance to prioritize and commit any last minute features and issues they would like to see in the upcoming release.

The RM must then send the pointer to the umbrella issue along with the tentative timeline for branch point to the developer lists. Any work identified as release related that needs to be completed should be added as a sub tasks of the umbrella issue to allow all developers and users to see the overall release progress in one place. The umbrella issue shall also link to any issues still requiring clarification whether or not they will make it into the release.

The RM should then inform users when the git branch is planned to be created, by sending an email based on this template:

```
[FINERACT] [PROPOSAL] □ New release ${project['fineract.release.version']}
```

Hello everyone,

... based on our release process
(https://fineract.apache.org/docs/current/#_releases), I will create a
release/\${project['fineract.release.version']} branch off develop in our git
repository at <https://github.com/apache/fineract> on
\${project['fineract.releaseBranch.date']}.

The release tracking umbrella issue for tracking all activity in JIRA is FINERACT-
\${project['fineract.release.issue']!'0000'}
([https://issues.apache.org/jira/browse/FINERACT-](https://issues.apache.org/jira/browse/FINERACT-${project['fineract.release.issue']!'0000'})
\${project['fineract.release.issue']!'0000'}).

If you have any work in progress that you would like to see included in this release,
please add "blocking" links to the release JIRA issue.

I am the release manager for this release.

Cheers,

\${project['fineract.config.name']}

Gradle Task

Command

```
./gradlew fineractReleaseStep1 -Pfineract.release.issue=1234 -  
Pfineract.releaseBranch.date="Monday, April 25, 2022" -Pfineract.release.version  
=1.14.0
```

Step 2: Clean Up JIRA

Description

Before a release is done, make sure that any issues that are fixed have their fix version setup correctly.

```
project = FINERACT and resolution = fixed and fixVersion is empty
```

Move all unresolved JIRA issues which have this release as Fix Version to the next release

```
project = FINERACT and fixVersion = 1.14.0 and status not in ( Resolved, Done,  
Accepted, Closed )
```

You can also run the following query to make sure that the issues fixed for the to-be-released version look accurate:

```
project = FINERACT and fixVersion = 1.14.0
```

Finally, check out the output of the JIRA release note tool to see which tickets are included in the release, in order to do a sanity check.

Gradle Task

Command

```
./gradlew fineractReleaseStep2 -Pfineract.release.version=1.14.0
```



This task is not yet automated!

Step 3: Create Release Branch

Description

Communicate with the community. You do not need to start a new email thread on the developer mailing list to notify that you are about to branch, just do it ca. 2 weeks after the initial email, or later, based on the discussion on the initial email.

You do not need to ask committers to hold off any commits until you have branched finished, as it's always possible to fast-forward the branch to latest develop, or cherry-pick last minute changes to it. People should be able to continue working on the develop branch on bug fixes and great new features for the next release while the release process for the current release is being worked through.

1. Clone fresh repository copy

```
git clone git@github.com:apache/fineract.git
cd fineract
```

2. Check that current HEAD points to commit on which you want to base new release branch. Checkout a particular earlier commit if not.

```
git log ①
```

① Check current branch history. HEAD should point to commit that you want to be base for your release branch

3. Create a new release branch using the version number

```
git checkout -b release/1.14.0
```

4. Push new branch to Apache Fineract repository

```
git push origin release/1.14.0
```

5. Start new release notes page under [Fineract Releases](#). The change list can be swiped from the JIRA release note tool (use the "text" format for the change log). See JIRA Cleanup above to ensure that the release notes generated by this tool are what you are expecting.

6. Send an email announcing the new release branch on the earlier email thread

```
[FINERACT] [ANNOUNCE] □ ${project['fineract.release.version']} release branch

Hello everyone,

... as previously announced, I've created the branch for our upcoming
${project['fineract.release.version']} release. The branch name is
release/${project['fineract.release.version']}.
```

You can continue working and merging PRs into the develop branch for future releases, as always.

I started the DRAFT release notes at [https://cwiki.apache.org/confluence/display/FINERACT/\\${project\['fineract.release.version'\]}+-+Apache+Fineract](https://cwiki.apache.org/confluence/display/FINERACT/${project['fineract.release.version']}+-+Apache+Fineract) . Please help me by filling in "Summary of changes". Does anyone see anything else missing?

Does anyone have any last minute changes for the release branch, or are we good to go and actually cut the release based on this branch as it is?

I'll initiate the final stage of actually creating the release on `${project['fineract.release.date']}` if nobody objects.

Cheers,

`${project['fineract.config.name']}`

Gradle Task

Command

```
./gradlew fineractReleaseStep3 -Pfineract.release.date="Monday, May 10, 2022"
-Pfineract.release.version=1.14.0
```

Step 4: Freeze JIRA

Description

You first need to close the release in JIRA so that the about to be released version cannot be used as "fixVersion" for new bugs anymore. Go to JIRA "Administer project" page and follow "Versions" in left menu. Table with list of all releases should appear, click on additional menu on the right of your release and choose "Release" option. Submit release date and you're done.

Gradle Task

Command

```
./gradlew fineractReleaseStep4
```



This task is not yet automated!

Step 5: Create Release Tag

Description

Next, you create a git tag from the HEAD of the release's git branch.

```
git checkout -b release/1.14.0 ①
git tag -a 1.14.0 -m "Fineract 1.14.0 release" -s ②
git push origin tag 1.14.0
```

① Ensure all tests pass for this commit both in CI and locally.

② -s is optional but recommended: GPG signatures on tags are useful for trust and integrity.



It is important to create so called annotated tags (vs. lightweight) for releases.

Gradle Task

Command

```
./gradlew fineractReleaseStep5 -Pfineract.release.version=1.14.0
```

Step 6: Create Distribution

Description

Create source and binary tarballs.

```
./gradlew clean srcDistTar binaryDistTar
```

Check that `fineract-provider/build/classes/java/main/git.properties` exists. If so, continue. If not, you're likely encountering [this bug](#), and you need to **re-run the command above** to create proper source and binary tarballs. That `git.properties` file is supposed to end up at `BOOT-INF/classes/git.properties` in `fineract-provider-1.14.0.jar` in the binary release tarball. Its contents are displayed at the `/fineract-provider/actuator/info` endpoint. It may be possible to fix this heisenbug entirely by [modifying our git properties gradle plugin config](#) in `fineract-provider/build.gradle`, perhaps by changing where `git.properties` is written.

Look in `fineract-war/build/distributions/` for the tarballs.

Do some sanity checks. The source tarball and the code in the release branch (at the commit with the release tag) should match.

```
cd /fineract-release-preparations
tar -xvzf path/to/apache-fineract-src-1.14.0.tar.gz
git clone git@github.com:apache/fineract.git
cd fineract/
git checkout tags/1.14.0
cd ..
diff -r fineract apache-fineract-src-1.14.0
```

Make sure the code compiles and tests pass on the uncompressed source. You should at the very least do exactly what you will ask the community to do in [Step 9: Verify Distribution Staging](#).

Ideally you'd build code and docs and run every possible test and check, but [running everything has complex dependencies, caches, and takes many hours](#). It is rarely done in practice offline / local / on developer machines. But please, go ahead and run the test and doc tasks, and more! Grab a cup of coffee and run everything you can. See the various builds in [.github/workflows/](#) and try the same things on your own. We should all hammer on a release candidate as much as we can to see if it breaks and fix it if so. All that of course improves our final release.



We don't release any artifacts to Apache's Maven repository.

Gradle Task

Command

```
./gradlew fineractReleaseStep6
```



This task doesn't work. Build release artifacts manually as indicated above.

Step 7: Sign Distribution

Description

Release source and binary tarballs must be checksummed and signed. In order to sign a release you will need a PGP key. You should get your key signed by a few other people. You will also need to receive their keys from a public key server. See the [Apache release policy](#) for more details.

```
# sign
gpg --armor --output apache-fineract-src-1.14.0.tar.gz.asc \
  --detach-sig apache-fineract-src-1.14.0.tar.gz
gpg --armor --output apache-fineract-bin-1.14.0.tar.gz.asc \
  --detach-sig apache-fineract-bin-1.14.0.tar.gz

# hash
gpg --print-md SHA512 apache-fineract-src-1.14.0.tar.gz \
  > apache-fineract-src-1.14.0.tar.gz.sha512
gpg --print-md SHA512 apache-fineract-bin-1.14.0.tar.gz \
  > apache-fineract-bin-1.14.0.tar.gz.sha512
```

Gradle Task

Command

```
./gradlew fineractReleaseStep7
```

Step 8: Upload Distribution Staging

Description

Next we'll stage the release candidate. Create a new folder and add these files:

- apache-fineract-bin-1.14.0.tar.gz
- apache-fineract-bin-1.14.0.tar.gz.sha512
- apache-fineract-bin-1.14.0.tar.gz.asc
- apache-fineract-src-1.14.0.tar.gz
- apache-fineract-src-1.14.0.tar.gz.sha512
- apache-fineract-src-1.14.0.tar.gz.asc

These files (or "artifacts") comprise the release candidate. Upload these files to [ASF's distribution dev/staging area](#) like so:

```
# this is a remote operation
svn mkdir https://dist.apache.org/repos/dist/dev/fineract/1.14.0
# create local svn-tracked folder "1.14.0"
svn checkout https://dist.apache.org/repos/dist/dev/fineract/1.14.0
# prepare to upload
cp path/to/new/folder/* 1.14.0/
cd 1.14.0/
# actual upload occurs here
svn add * && svn commit
```



You will need your ASF Committer credentials to be able to access the Subversion host at dist.apache.org.

Gradle Task

Command

```
./gradlew fineractReleaseStep8 -Pfineract.release.version=1.14.0
```



This task is inefficient. Follow `svn mkdir` and other manual steps above.

Step 9: Verify Distribution Staging

Description

Following are the typical things we need to verify before voting on a release candidate. And the release manager should verify them too before calling out a vote.

Make sure release artifacts are hosted at dist.apache.org/repos/dist/dev/fineract

- Release candidate files should match filenames mentioned earlier, and will be moved without renaming if/when the release vote passes.

- Verify signatures and hashes. You may have to import the public key of the release manager to verify the signatures. (`gpg --import KEYS` or `gpg --recv-key <key id>`)
- Git tag matches the released bits (`diff -rf`)
- Can compile docs and code successfully from source
- Verify DISCLAIMER, NOTICE and LICENSE (year etc)
- All files have correct headers (Rat check should be clean - `./gradlew rat`)
- No jar files in the source artifacts
- All tests pass both in CI and locally

Artifact verification

```
# source tarball signature and checksum verification steps
# we'll check the source tarball first
src=apache-fineract-src-1.14.0.tar.gz

# upon success: prints "Good signature" and returns successful exit code
# upon failure: prints "BAD signature" and returns error exit code
gpg --verify $src.asc

# upon success: prints nothing and returns successful exit code
# upon failure: prints checksum differences and returns error exit code
gpg --print-md SHA512 $src | diff - $src.sha512

# binary tarball signature and checksum verification steps and outputs are similar
bin=apache-fineract-bin-1.14.0.tar.gz
gpg --verify $bin.asc
gpg --print-md SHA512 $bin | diff - $bin.sha512
```

Look for **Good signature** in the `gpg` output:

```
$ gpg --verify $bin.asc
gpg: assuming signed data in 'apache-fineract-bin-1.14.0.tar.gz'
gpg: Signature made Sat 11 Oct 2025 05:46:42 PM PDT
gpg:                using EDDSA key 250775BDB5FE7D53E4AF95C00E895A1A7A090CFC
gpg: Good signature from "Adam Monsen <haircut@gmail.com>" [unknown]
```

That's the most important part.

You may see this warning:

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                There is no indication that the signature belongs to the owner.
```

You may choose to ignore it. To squelch this warning, you must extend your [web of trust](#), by, for example, [signing the release manager's key](#).

Now it's time to build and run the release candidate.

Build from source

```
tar -xzf $src
cd apache-fineract-src-1.14.0
./gradlew build -x test -x doc
cd ..
```

Run from binary

Before running Fineract you must first start a [supported relational database server](#) and ensure the `fineract_default` and `fineract_tenants` databases exist. Detailed steps for database preparation are left as an exercise for the reader. You can find ideas on how to prepare your database in the `build-mariadb.yml`, `build-mysql.yml`, and `build-postgresql.yml` files in source control, and in [Database Setup](#).

Finally, start your Fineract server:

```
tar -xvzf apache-fineract-bin-1.14.0.tar.gz
cd apache-fineract-bin-1.14.0
export FINERACT_SERVER_SSL_ENABLED=false
export FINERACT_SERVER_PORT=8080
export BACKEND_PROTOCOL=http
export BACKEND_PORT=$FINERACT_SERVER_PORT
# assumes reachable, healthy mariadb with default username, password, and port
java -jar fineract-provider-1.14.0.jar
```

Alternatively, you can run it in Tomcat:

```
cat << 'EndOfRcenv' >> rcenv
FINERACT_SERVER_SSL_ENABLED=false
FINERACT_SERVER_PORT=8080
BACKEND_PROTOCOL=http
BACKEND_PORT=$FINERACT_SERVER_PORT
EndOfRcenv
# assumes reachable, healthy mariadb with default username, password, and port
docker run --rm -it -v "$(pwd):/usr/local/tomcat/webapps" \
  --net=host --env-file=rcenv tomcat:jre21
```

Confirm the following:

1. localhost:8080/fineract-provider/actuator/health works
2. localhost:8080/fineract-provider/actuator/info displays the expected information
3. API calls work against localhost:8080/fineract-provider/api/v1

Gradle Task

Command

```
./gradlew fineractReleaseStep9 -Pfineract.release.version=1.14.0
```



This task is not yet automated!

Step 10: Start Vote

Description

Voting has to be done on dev@fineract.apache.org. You can close the vote after voting period expires (72 hours) and you accumulate sufficient votes (minimum 3 x +1 PMC votes).

```
[FINERACT] [VOTE] [] ${project['fineract.release.version']} for release
```

Hello everyone,

We are proud to present Apache Fineract `${project['fineract.release.version']}`, with the artifacts below up for a vote. Releases are important for a number of reasons: They put a stamp of approval on a set of code changes and they build momentum for future improvements.

Release notes and ChangeLog:

[https://cwiki.apache.org/confluence/display/FINERACT/\\${project\['fineract.release.version'\]}+-+Apache+Fineract](https://cwiki.apache.org/confluence/display/FINERACT/${project['fineract.release.version']}+-+Apache+Fineract)

Source and binary artifacts:

[https://dist.apache.org/repos/dist/dev/fineract/\\${project\['fineract.release.version'\]}/](https://dist.apache.org/repos/dist/dev/fineract/${project['fineract.release.version']}/)

Tagged as `${project['fineract.release.version']}`

Committer PGP keys, including the release signing key:

<https://dist.apache.org/repos/dist/dev/fineract/KEYS>

Note that this release candidate contains source and binary artifacts.

This vote will be open for 72 hours:

[] +1 approve

[] +0 no opinion

[] -1 disapprove (and reason why)

Please indicate if yours is a binding vote, and "Verified: YES/NO/PARTIAL".

Verified: YES ... Verified integrity and signatures of release artifacts locally, built from source, ran jar/war: Did everything mentioned in the current release

candidate verification guidance (https://fineract.apache.org/docs/rc/#_artifact_verification). If you did more than that, please specify. "Verified: YES" is required for binding votes.

Verified: NO ... No testing performed on release candidate, e.g. relying on testing performed by other contributors and/or output of GitHub Actions, while exercising your right to vote.

Verified: PARTIAL ... Please specify.

Cheers,

`${project['fineract.config.name']}`

Gradle Task

Command

```
./gradlew fineractReleaseStep10 -Pfineract.release.version=1.14.0
```

Step 11: Finish Vote

Description

Upon receiving 3 x +1 from the PMC, or after 72 hours (whichever one comes first), reply to the voting thread and add the prefix "[RESULT]" to the subject line with the results, as follows:

```
[FINERACT] [VOTE] [RESULT]  [] ${project['fineract.release.version']} for release
```

```
<#if (project['fineract.vote'].approve.binding?size +  
project['fineract.vote'].approve.nonBinding?size >  
project['fineract.vote'].disapprove.binding?size +  
project['fineract.vote'].disapprove.nonBinding?size)>
```

Voting is now closed and has passed with the following tally,

```
Binding +1s: ${project['fineract.vote'].approve.binding?size}  
Non binding +1s: ${project['fineract.vote'].approve.nonBinding?size}  
<#else>
```

Voting is now closed and has not passed with the following tally,

```
Binding +1s: ${project['fineract.vote'].approve.binding?size}  
Non binding +1s: ${project['fineract.vote'].approve.nonBinding?size}
```

```
Binding -1s: ${project['fineract.vote'].disapprove.binding?size}  
Non binding -1s: ${project['fineract.vote'].disapprove.nonBinding?size}  
</#if>
```

Here are the detailed results:

```
<#list project['fineract.vote'].approve.binding>
Binding +1s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#list project['fineract.vote'].approve.nonBinding>
Non binding +1s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#list project['fineract.vote'].disapprove.binding>
Binding -1s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#list project['fineract.vote'].disapprove.nonBinding>
Non binding -1s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#list project['fineract.vote'].noOpinion.binding>
Binding +0s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#list project['fineract.vote'].noOpinion.nonBinding>
Non binding +0s:
    <#items as item>
-   ${item.name}
    </#items>
</#list>
```

```
<#if (project['fineract.vote'].approve.binding?size +
project['fineract.vote'].approve.nonBinding?size >
project['fineract.vote'].disapprove.binding?size +
project['fineract.vote'].disapprove.nonBinding?size)>
Thanks to everyone who voted! I'll now continue with the rest of the release process.
</else>
```

Thanks to everyone who voted! Looks like we have to repeat the vote.

</#if>

```
${project['fineract.config.name']}
```

Gradle Task

Command

```
./gradlew fineractReleaseStep11 -Pfineract.release.version=1.14.0
```

Step 12: Upload Distribution Release

Description

Move the release candidate from the dev area to the release area using a Subversion server-side copy.

```
svn mv https://dist.apache.org/repos/dist/dev/fineract/1.14.0  
https://dist.apache.org/repos/dist/release/fineract/
```



This must be done by a Fineract PMC member.

You will now get an automated email from the Apache Reporter Service ([no-reply@reporter.apache.org](mailto:reply@reporter.apache.org)), subject "Please add your release data for 'fineract'" to add the release data (version and date) to the database on reporter.apache.org/addrelease.html?fineract (requires PMC membership).

Gradle Task

Command

```
./gradlew fineractReleaseStep12 -Pfineract.release.version=1.14.0
```



This task is not yet automated!

Step 13: Close Release Branch

Description

As discussed in [FINERACT-1154](#), now that everything is final, please do the following to remove the release branch (and just keep the tag), and make sure that everything on the release tag is merged to develop and that e.g. `git describe` works:

```
git checkout develop  
git merge release/1.14.0 ①
```

```
git push origin develop
git branch -D release/1.14.0
git push origin :release/1.14.0
git describe ②
```

- ① This merge is necessary for posterity: It's how we're able to preserve and trace lineage from releases to descendent commit. Note this is a traditional merge. This is for simplicity, and is an exception to our otherwise [flat git commit history](#).
- ② The output must refer to the most recent release. For example, if your working copy is checked out to the `develop` branch, the current commit is `0762a012e`, and the latest release tag (28 commits ago) was `1.12.1`, the output of `git describe` would be `1.12.1-28-g0762a012e`.

Gradle Task

Command

```
./gradlew fineractReleaseStep13 -Pfineract.release.version=1.14.0
```



This task is not yet automated!

Step 14: Update website

Description

Finally update the fineract.apache.org website with the latest release details. The website's HTML source code is available at github.com/apache/fineract-site.



This step is not yet automated. We are working on a static site generator setup.

Gradle Task

Command

```
./gradlew fineractReleaseStep14 ①
```

- ① Currently doing nothing. Will trigger in the future the static site generator and publish on Github.



This task is not yet automated!

Step 15: Announcement Email

Description

Send an email to announce@apache.org (sender address must be @apache.org):

```
[ANNOUNCE] Apache Fineract ${project['fineract.release.version']} Release
```

The Apache Fineract project is pleased to announce the release of Apache Fineract `${project['fineract.release.version']}`. The release is available for download from <https://fineract.apache.org/#downloads>

Apache Fineract is an open-source core banking platform providing a flexible, extensible foundation for a wide range of financial services. By making robust banking technology openly available, it lowers barriers for institutions and innovators to reach underserved and unbanked populations.

This release addressed `${project['fineract.release.issues']?size}` issues.

Readme:

[https://github.com/apache/fineract/blob/\\${project\['fineract.release.version'\]}/README.md](https://github.com/apache/fineract/blob/${project['fineract.release.version']}/README.md)

Release page:

[https://cwiki.apache.org/confluence/display/FINERACT/\\${project\['fineract.release.version'\]}+-+Apache+Fineract](https://cwiki.apache.org/confluence/display/FINERACT/${project['fineract.release.version']}+-+Apache+Fineract)

List of fixed issues:

[https://issues.apache.org/jira/secure/ReleaseNote.jspa?version=\\${project\['fineract.release.versionId'\]}&styleName=Html&projectId=\\${project\['fineract.release.projectId'\]}](https://issues.apache.org/jira/secure/ReleaseNote.jspa?version=${project['fineract.release.versionId']}&styleName=Html&projectId=${project['fineract.release.projectId']})

For more information on Apache Fineract please visit project home page: <https://fineract.apache.org>

The Apache Fineract Team

Gradle Task

Command

```
./gradlew fineractReleaseStep15 -Pfineract.release.version=1.14.0
```

Maintenance Release Process



This is a first attempt to introduce maintenance releases. Some details might change as soon as we get more experience with the process and feedback from the community. The numbers here are still more or less arbitrary, and we'll adapt as necessary.

Rules

- hotfix releases are reserved for critical (BLOCKER) bugs and security issues. Probably we'll have some kind of voting process in place, e. g. "minimum 3 x +1 votes from PMC members"
- we will support (for now to start) two minor versions back counting from the last release; this

would mean that once 1.8.0 is out we would support 1.8.x and 1.7.x, but not 1.6.x and older; this rule is tentative, we'll see then what we do in the future when we have more feedback.

- guaranteed backward compatibility with the last minor release; i. e. "1.6.1" is a drop-in replacement for "1.6.0"
- NO new features, tables, data, REST endpoints
- NO major (or "minor" framework upgrades); i. e. if we used Spring Boot "2.6.1" in version "1.6.0" of Fineract we can upgrade dependencies to "2.6.10" (unless it breaks something of course), but not to "2.7.2" of Spring Boot



The rest of the release process is the same as for normal releases. In the future we might have smaller time windows for reviews.

JIRA

- Continuously update the [JIRA umbrella issue](#) to make sure we catch all ticket changes.
- List tickets that have discrepancies, e. g. tickets still open while associated PR merged, ticket on wrong version (i. e. associated PR already merged before with another release).

Publish Release Artifacts



More on releases at the ASF see www.apache.org/legal/release-policy.html#distribute-raw-artifact

Requirements

You need to have your GPG keypairs properly set up. The JAR release artifacts (currently only `fineract-client`) are signed with a Gradle plugin just before being uploaded to the Maven repository. Please make sure that the following properties are set in your private `gradle.properties` file in your home folder:

```
signing.keyId=7890ABCD
signing.password=*****
signing.secretKeyRingFile=~/.gnupg/secring.gpg
```

This is quite similar to the Fineract release plugin properties for GPG. In one of the next release we'll merge these two setups to avoid this duplicated configuration.

Maven Repository

We are using the ASF's official [Nexus Maven repository](#) to publish our snapshot and release artifacts.



Find more information at infra.apache.org/publishing-maven-artifacts.html

NPM Registry

For convenience we will be using Github Packages to publish Fineract's Typescript API client.

TBD

Docker Hub

TBD

Fineract SDKs

TBD

Generate Apache Fineract API Client

Apache Fineract supports client code generation using [OpenAPI Generator](#). It uses [OpenAPI Specification Version 3.0.3](#).

Fineract SDK Java API Client

The `fineract-client.jar` will eventually be available on Maven Central (watch [FINERACT-1102](#)). Until it is, you can quite easily build the latest and greatest version locally from source, see below.

The `FineractClient` is the entry point to the *Fineract SDK Java API Client*. `Calls` is a convenient and recommended utility to simplify the use of the `retrofit2.Call` type which all API operations return. This permits you to use the API like the `FineractClientDemo` illustrates:

```
import org.apache.fineract.client.util.FineractClient;
import static org.apache.fineract.client.util.Calls.ok;

    FineractClient fineract = FineractClient.builder().baseUrl
("https://demo.fineract.dev/fineract-provider/api/v1/").tenant("default")
    .basicAuth("mifos", "password").build();
    List<StaffData> staff = Calls.ok(fineract.staff.retrieveAll16(1L, true, false,
"ACTIVE"));
    String name = staff.get(0).getDisplayName();
    log.info("Display name: {}", name);
```

Generate API Client

The API client is built as part of the standard overall Fineract Gradle build. The client JAR can be found in `fineract-client/build/libs` as `fineract-client.jar`.

If you need to save time to incrementally work on making small changes to Swagger annotations in an IDE, you can execute e.g. the following line in root directory of the project to exclude non-require Gradle tasks:

```
./gradlew -x compileJava -x compileTest -x spotlessJava -x enhance resolve
prepareInputYaml :fineract-client:buildJavaSdk
```

Validate OpenAPI Spec File

The `resolve` task in `build.gradle` file will generate the OpenAPI Spec File for the project. To make sure Swagger Codegen generates a correct library, it is important for the OpenAPI Spec file to be valid. Validation is done automatically by the OpenAPI code generator Gradle plugin. If you still

have problems during code generation please use [Swagger OpenAPI Validator](#) to validate the spec file.

Frequently Asked Questions

See [Fineract FAQ wiki page](#).

Glossary

TBD

Appendix A: Fineract Application Properties

TBD

Tenant Database Properties

Table 3. Tenant Database Properties

Name	Env Variable	Default Value	Description
fineract.tenant.host	FINERACT_DEFAULT_TENANTDB_HOSTNAME	localhost	This property sets the hostname of the default tenant database.
fineract.tenant.port	FINERACT_DEFAULT_TENANTDB_PORT	3306	This property sets the port of the default tenant database.
fineract.tenant.username	FINERACT_DEFAULT_TENANTDB_UID	root	This property sets the username of the default tenant database.
fineract.tenant.password	FINERACT_DEFAULT_TENANTDB_PWD	mysql	This property sets the password of the default tenant database.
fineract.tenant.parameters	FINERACT_DEFAULT_TENANTDB_CONNECTION_PARAMS		This property sets the connection parameters of the default tenant database. eg. whether ssl is enabled or not
fineract.tenant.timezone	FINERACT_DEFAULT_TENANTDB_TIMEZONE	Asia/Kolkata	This property sets the timezone of the default tenant
fineract.tenant.identifier	FINERACT_DEFAULT_TENANTDB_IDENTIFIER	default	This property sets the unique identifier for the tenant within fineract
fineract.tenant.name	FINERACT_DEFAULT_TENANTDB_NAME	fineract_default	This property sets the database name of the default tenant
fineract.tenant.description	FINERACT_DEFAULT_TENANTDB_DESCRIPTION	Default Demo Tenant	This property sets the description of the default tenant
fineract.tenant.master-password	FINERACT_DEFAULT_TENANTDB_MASTER_PASSWORD	fineract	The password used to encrypt sensitive tenant data within the database

Name	Env Variable	Default Value	Description
fineract.tenant.encryption	FINERACT_DEFAULT_TENANTDB_ENCRYPTION	AES/CBC/PKCS5Padding	This property sets the symmetric encryption algorithm used to encrypt sensitive tenant data within the database e.g tenant database password
spring.liquibase.enabled	FINERACT_LIQUIBASE_ENABLED	true	If set to true, liquibase will be enabled and the instance running this configuration will run migrations
fineract.tenant.read-only-name	FINERACT_DEFAULT_TENANTDB_RO_NAME		For read only configuration, set this to the name of the read only tenant database
fineract.tenant.read-only-host	FINERACT_DEFAULT_TENANTDB_RO_HOSTNAME		For read only configuration, set this to the hostname of the read only tenant database
fineract.tenant.read-only-port	FINERACT_DEFAULT_TENANTDB_RO_PORT		For read only configuration, set this to the port of the read only tenant database
fineract.tenant.read-only-username	FINERACT_DEFAULT_TENANTDB_RO_UID		For read only configuration, set this to the username of the read only tenant database
fineract.tenant.read-only-password	FINERACT_DEFAULT_TENANTDB_RO_PWD		For read only configuration, set this to the password of the read only tenant database
fineract.tenant.read-only-parameters	FINERACT_DEFAULT_TENANTDB_RO_CONNECTION_PARAMETERS		For read only configuration, set this to the connection parameters of the read only tenant database

Hikari Connection Pool Properties

Table 4. Hikari Connection Pool Properties

Name	Env Variable	Default Value	Description
spring.datasource.hikari.driverClassName	FINERACT_HIKARI_DRIVER_SOURCE_CLASS_NAME	org.mariadb.jdbc.Driver	The correct driver name for the database that will be used with fineract.
spring.datasource.hikari.jdbcUrl	FINERACT_HIKARI_JDBC_URL	jdbc:mariadb://localhost:3306/fineract_tenants	The database connection string for the database with tenant information that will be used with fineract.
spring.datasource.hikari.username	FINERACT_HIKARI_USERNAME	root	The username for the database with tenant information that will be used with fineract
spring.datasource.hikari.password	FINERACT_HIKARI_PASSWORD	mysql	The password for the database with tenant information that will be used with fineract
spring.datasource.hikari.minimumIdle	FINERACT_HIKARI_MINIMUM_IDLE	3	The minimum number of connections in hikari pool that will be maintained when the system is idle
spring.datasource.hikari.maximumPoolSize	FINERACT_HIKARI_MAXIMUM_POOL_SIZE	10	The maximum number of connections that hikari can create in the pool.
spring.datasource.hikari.idleTimeout	FINERACT_HIKARI_IDLE_TIMEOUT	60000	The maximum time in milliseconds that a connection is allowed to sit idle in the pool.
spring.datasource.hikari.connectionTimeout	FINERACT_HIKARI_CONNECTION_TIMEOUT	20000	The maximum time in milliseconds that hikari will wait for a connection to be established.
spring.datasource.hikari.connectionTestquery	FINERACT_HIKARI_TEST_QUERY	SELECT 1	The query that will be used to test the database connection.

Name	Env Variable	Default Value	Description
spring.datasource.hikari. autoCommit	FINERACT_HIKARI_AUTO_COMMIT	true	If set to true, the connections in the pool will be in auto-commit mode.
spring.datasource.hikari. dataSourceProperties['cachePrepStmts']	FINERACT_HIKARI_DS_PROPERTIES_CACHE_PREP_STMTS	true	If set to true, hikari caches compiled SQL statements to avoid the overhead of re-parsing and re-compiling SQL queries.
spring.datasource.hikari. dataSourceProperties['prepStmtCacheSize']	FINERACT_HIKARI_DS_PROPERTIES_PREP_STMT_CACHE_SIZE	250	The maximum number of prepared statements that hikari can cache.
spring.datasource.hikari. dataSourceProperties['prepStmtCacheSqlLimit']	FINERACT_HIKARI_DS_PROPERTIES_PREP_STMT_CACHE_SQL_LIMIT	2048	This property sets the upper limit for the size of individual SQL queries that can be stored in the cache. If a SQL query exceeds this limit in terms of character length, it will not be cached, even if caching is enabled.
spring.datasource.hikari. dataSourceProperties['useServerPrepStmts']	FINERACT_HIKARI_DS_PROPERTIES_USE_SERVER_PREP_STMTS	true	This property determines if the connection should leverage server-side prepared statements rather than client-side ones.
spring.datasource.hikari. dataSourceProperties['useLocalSessionState']	FINERACT_HIKARI_DS_PROPERTIES_USE_LOCAL_SESSION_STATE	true	This property allows the connection pool to locally track changes to session-specific properties (like character sets or time zones) rather than sending these queries to the database repeatedly.

Name	Env Variable	Default Value	Description
spring.datasource.hikari.dataSourceProperties['rewriteBatchedStatements']	FINERACT_HIKARI_DS_PROPERTIES_REWRITE_BATCHED_STATEMENTS	true	This property, when set to true, allows the JDBC driver to rewrite batched SQL statements into a more efficient single query format before sending them to the database.
spring.datasource.hikari.dataSourceProperties['cacheResultSetMetadata']	FINERACT_HIKARI_DS_PROPERTIES_CACHE_RESULT_SET_METADATA	true	This property, when set to true, enables the caching of metadata for ResultSet objects. This metadata includes details such as column names, types, and other relevant schema information.
spring.datasource.hikari.dataSourceProperties['cacheServerConfiguration']	FINERACT_HIKARI_DS_PROPERTIES_CACHE_SERVER_CONFIGURATION	true	When set to true, this property allows the JDBC driver to cache the server configuration settings, which include properties such as session state, character sets, and other configuration details relevant to the database server.
spring.datasource.hikari.dataSourceProperties['elideSetAutoCommits']	FINERACT_HIKARI_DS_PROPERTIES_ELIDE_SET_AUTO_COMMITS	true	When set to true, this property prevents the JDBC driver from issuing a SET autocommit command on the database connection during its initialization.

Name	Env Variable	Default Value	Description
spring.datasource.hikari.dataSourceProperties['maintainTimeStats']	FINERACT_HIKARI_DS_PROPERTIES_MAINTAIN_TIME_STATS	false	When set to true, this property enables HikariCP to track and maintain statistics regarding various timing metrics related to connection pool operations, such as connection acquisition times.
spring.datasource.hikari.dataSourceProperties['logSlowQueries']	FINERACT_HIKARI_DS_PROPERTIES_LOG_SLOW_QUERIES	true	When set to true, this property enables HikariCP to log SQL queries that exceed a specified execution time threshold, allowing developers and administrators to identify and analyze performance issues related to slow-running queries.
spring.datasource.hikari.dataSourceProperties['dumpQueriesOnException']	FINERACT_HIKARI_DS_PROPERTIES_DUMP_QUERIES_IN_EXCEPTION	true	When set to true, this property instructs HikariCP to log the SQL statements that caused exceptions during execution. This includes capturing the query text and any associated parameters.

SSL Properties

Table 5. SSL Properties

Name	Env Variable	Default Value	Description
server.ssl.enabled	FINERACT_SERVER_SSL_ENABLED	true	When set to true, SSL (Secure Sockets Layer) or TLS (Transport Layer Security) will be enabled for the server.

Name	Env Variable	Default Value	Description
server.ssl.protocol	FINERACT_SERVER_SSL_PROTOCOL	TLS	This property allows you to define specific SSL/TLS protocol version the server will use when establishing secure connections. Common protocols include TLSv1.2, TLSv1.3, etc.
server.ssl.ciphers	FINERACT_SERVER_SSL_CIPHERS	TLS_RSA_WITH_AES_128_CBC_SHA256	This property allows you to control the cipher suites that fineract will accept for secure connections
server.ssl.enabled-protocols	FINERACT_SERVER_SSL_PROTOCOLS	TLSv1.2	This property allows you to define a list of SSL/TLS protocol versions that the server will support when establishing secure connections
server.ssl.key-store	FINERACT_SERVER_SSL_KEY_STORE	classpath:keystore.jks	The property is used to specify the location of the SSL key store file that contains the server's private key and the associated certificate
server.ssl.key-store-password	FINERACT_SERVER_SSL_KEY_STORE_PASSWORD	openmf	The property defines the password for the keystore specified under property server.ssl.key-store

Authentication Properties

Table 6. Authentication Properties

Name	Env Variable	Default Value	Description
fineract.security.basic-auth.enabled	FINERACT_SECURITY_BASICAUTH_ENABLED	true	When set to true, the supported authentication method will be basic authentication.

Name	Env Variable	Default Value	Description
fineract.security.oauth2.enabled	FINERACT_SECURITY_OAUTH_ENABLED	false	When set to true, the supported authentication method will be OAuth.
fineract.security.2fa.enabled	FINERACT_SECURITY_2FA_ENABLED	false	Set the value to true enable two-factor authentication. For this to work as expected, ensure that you have set the correct email/sms configuration
spring.security.oauth2.resourceserver.jwt.issuer-uri	FINERACT_SERVER_OAUTH_RESOURCE_URL	localhost:9000/auth/realms/fineract	If OAuth is enabled and a custom resource server (different from what is provided) is required, set the issuer-uri here.

Tomcat Properties

Table 7. Tomcat Properties

Name	Env Variable	Default Value	Description
server.tomcat.accept-count	FINERACT_SERVER_TOMCAT_ACCEPT_COUNT	100	The property specifies the maximum number of concurrent connection requests that embedded Tomcat can queue. If this limit is reached, incoming connection requests will be rejected.
server.tomcat.accesslog.enabled	FINERACT_SERVER_TOMCAT_ACCESSLOG_ENABLED	false	If set to true, tomcat will log access requests to file
server.tomcat.max-connections	FINERACT_SERVER_TOMCAT_MAX_CONNECTIONS	8192	Sets the maximum number of simultaneous connections Tomcat can handle.

Name	Env Variable	Default Value	Description
server.tomcat.max-http-form-post-size	FINERACT_SERVER_TOMCAT_MAX_HTTP_FORM_POST_SIZE	2MB	The property in sets the maximum size of HTTP POST requests that Tomcat can handle
server.tomcat.max-keep-alive-requests	FINERACT_SERVER_TOMCAT_MAX_KEEP_ALIVE_REQUESTS	100	The property specifies the maximum number of HTTP requests that can be sent over a single persistent connection (HTTP Keep-Alive) before Tomcat closes the connection
server.tomcat.threads.max	FINERACT_SERVER_TOMCAT_THREADS_MAX	200	The property sets the maximum number of threads that Tomcat can use to process requests
server.tomcat.threads.min-spare	FINERACT_SERVER_TOMCAT_THREADS_MIN_SPARE	10	The property specifies the minimum number of spare (idle) threads that Tomcat should maintain

Kafka Properties

Table 8. Kafka related properties for Remote Spring Batch Jobs

Name	Env Variable	Default Value	Description
fineract.remote-job-message-handler.kafka.enabled	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_ENABLED	false	Enables or disables Kafka for remote job execution. If Kafka is enabled then JMS shall be disabled.
fineract.remote-job-message-handler.kafka.topic.auto-create	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_TOPIC_AUTO_CREATE	true	Enables topic auto creation. In case the auto creation of the topic is disabled please make sure that the replica and the partition count is properly configured.

Name	Env Variable	Default Value	Description
fineract.remote-job-message-handler.kafka.topic.name	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_TOPIC_NAME	job-topic	Name of the topic where partitioned tasks are sent to
fineract.remote-job-message-handler.kafka.topic.replicas	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_TOPIC_REPLICAS	1	Number of the replicas
fineract.remote-job-message-handler.kafka.topic.partitions	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_TOPIC_PARTITIONS	10	Number of partitions
fineract.remote-job-message-handler.kafka.bootstrap-servers	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_BOOTSTRAP_SERVERS	localhost:9092	Comma separated list of bootstrap servers
fineract.remote-job-message-handler.kafka.consumer.group-id	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_CONSUMER_GROUPID	fineract-consumer-group-id	Group ID of the Consumer
fineract.remote-job-message-handler.kafka.consumer.extra-properties-key-value-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_CONSUMER_EXTRA_PROPERTIES_SEPARATOR	=	Defines key and value separator for consumer, e.g.: key=value
fineract.remote-job-message-handler.kafka.consumer.extra-properties-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_CONSUMER_EXTRA_PROPERTIES_SEPARATOR		Defines item separator for consumer, e.g.: key1=value1 key2=value2
fineract.remote-job-message-handler.kafka.consumer.extra-properties	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_CONSUMER_EXTRA_PROPERTIES		#holds list of key value pairs using the above defined separators for consumer: key1=value1 key2=value2 ... keyn=valuen
fineract.remote-job-message-handler.kafka.producer.extra-properties-key-value-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_PRODUCER_EXTRA_PROPERTIES_KEY_VALUE_SEPARATOR	=	Defines key and value separator for producer, e.g.: key=value

Name	Env Variable	Default Value	Description
fineract.remote-job-message-handler.kafka.producer.extra-properties-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_PRODUCER_EXTRA_PROPERTIES_SEPARATOR		Defines item separator for producer, e.g.: key1=value1 key2=value2
fineract.remote-job-message-handler.kafka.producer.extra-properties	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_PRODUCER_EXTRA_PROPERTIES		#holds list of key value pairs using the above defined separators for producer: key1=value1 key2=value2 ... keyn=valuen
fineract.remote-job-message-handler.kafka.admin.extra-properties-key-value-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_ADMIN_EXTRA_PROPERTIES_KEY_VALUE_SEPARATOR	=	Defines key and value separator for admin, e.g.: key=value
fineract.remote-job-message-handler.kafka.admin.extra-properties-separator	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_ADMIN_EXTRA_PROPERTIES_SEPARATOR		Defines item separator for admin, e.g.: key1=value1 key2=value2
fineract.remote-job-message-handler.kafka.admin.extra-properties	FINERACT_REMOTE_JOB_MESSAGE_HANDLER_KAFKA_ADMIN_EXTRA_PROPERTIES		#holds list of key value pairs using the above defined separators for admin: key1=value1 key2=value2 ... keyn=valuen

Table 9. Kafka related Properties for External Events

Name	Env Variable	Default Value	Description
fineract.events.external.producer.kafka.enabled	FINERACT_EXTERNAL_EVENTS_KAFKA_ENABLED	false	Enables disables Kafka for External Events. If Kafka is enabled then JMS shall be disabled.
fineract.events.external.producer.kafka.timeout-in-seconds	FINERACT_EXTERNAL_EVENTS_KAFKA_TIMEOUT_IN_SECONDS	10	Timeout for Kafka confirming the messages written in the topic

Name	Env Variable	Default Value	Description
fineract.events.external. producer.kafka.topic. auto-create	FINERACT_EXTERNAL_Events_KAFKA_TOPIC_AUTO_CREATE	true	Enables topic auto creation. In case the auto creation of the topic is disabled please make sure that the replica and the partition count is properly configured.
fineract.events.external. producer.kafka.topic. name	FINERACT_EXTERNAL_Events_KAFKA_TOPIC_NAME	external-events	Name of the topic where external events are sent to
fineract.events.external. producer.kafka.topic. replicas	FINERACT_EXTERNAL_Events_KAFKA_TOPIC_REPLICAS	1	Number of the replicas
fineract.events.external. producer.kafka.topic. partitions	FINERACT_EXTERNAL_Events_KAFKA_TOPIC_PARTITIONS	10	Number of partitions
fineract.events.external. producer.kafka.bootstr ap-servers	FINERACT_EXTERNAL_Events_KAFKA_BOOTSTRAP_SERVERS	localhost:9092	Comma separated list of Kafka bootstrap servers
fineract.events.external. producer.kafka.produc er.extra-properties- separator	FINERACT_EXTERNAL_Events_KAFKA_PRODUCER_EXTRA_PROPERTIES_SEPARATOR		Defines item separator for producer,e.g.: key=value
fineract.events.external. producer.kafka.produc er.extra-properties-key- value-separator	FINERACT_EXTERNAL_Events_KAFKA_PRODUCER_EXTRA_PROPERTIES_KEY_VALUE_SEPARATOR	=	Defines key and value separator for producer client
fineract.events.external. producer.kafka.produc er.extra-properties	FINERACT_EXTERNAL_Events_KAFKA_PRODUCER_EXTRA_PROPERTIES	linger.ms=10 batch.size=16384	Defines the extra properties for external event producer clients. Optimization for sending out large volume of messages. Increases Batch buffer size and batching time window.
fineract.events.external. producer.kafka.admin. extra-properties- separator	FINERACT_EXTERNAL_Events_KAFKA_ADMIN_EXTRA_PROPERTIES_SEPARATOR		Defines item separator for admin client.

Name	Env Variable	Default Value	Description
fineract.events.external-producer.kafka.admin.extra-properties-key-value-separator	FINERACT_EXTERNAL_EVENTS_KAFKA_ADMIN_EXTRA_PROPERTIES_KEY_VALUE_SEPARATOR	=	Defines key and value separator for admin client
fineract.events.external-producer.kafka.admin.extra-properties	FINERACT_EXTERNAL_EVENTS_KAFKA_ADMIN_EXTRA_PROPERTIES		Defines the extra properties for external event admin clients

Metrics Properties

For further understanding of the configurations properties related to metrics, refer to [Springboot metrics docs](#)

Table 10. Metrics Properties

Name	Env Variable	Default Value	Description
management.info.git.mode		FULL	Mode for displaying Git information in the <code>/info</code> endpoint.
management.endpoints.web.exposure.include	FINERACT_MANAGEMENT_ENDPOINT_WEB_EXPOSURE_INCLUDE	health,info,prometheus	Comma-separated list of endpoints that should be exposed over the web.
management.tracing.enabled	FINERACT_MANAGEMENT_METRICS_TAGS_APPLICATION	fineract	Whether tracing is enabled.
management.metrics.distribution.percentiles-histogram.http.server.requests	FINERACT_MANAGEMENT_METRICS_DISTRIBUTION_HTTP_SERVER_REQUESTS	false	Whether to publish percentile histograms for HTTP server requests.
management.otlp.metrics.export.url	FINERACT_MANAGEMENT_OTLP_METRICS_EXPORT_URL	tempo:4318/v1/traces	URL to export OTLP metrics.
management.otlp.metrics.export.aggregationTemporality	FINERACT_MANAGEMENT_OTLP_METRICS_EXPORT_AGGREGATION_TEMPORALITY	cumulative	Aggregation temporality for OTLP metrics export.
management.prometheus.metrics.export.enabled	FINERACT_MANAGEMENT_PROMETHEUS_ENABLED	false	Whether to enable Prometheus metrics export.

Name	Env Variable	Default Value	Description
spring.cloud.aws.cloudwatch.enabled	FINERACT_MANAGEMENT_CLOUDWATCH_ENABLED	false	Whether to enable AWS CloudWatch integration.
management.metrics.export.cloudwatch.enabled	FINERACT_MANAGEMENT_CLOUDWATCH_ENABLED	false	Whether to enable CloudWatch metrics export.
management.metrics.export.cloudwatch.namespace	FINERACT_MANAGEMENT_CLOUDWATCH_NAMESPACE	fineract	Namespace for CloudWatch metrics.
management.metrics.export.cloudwatch.step	FINERACT_MANAGEMENT_CLOUDWATCH_STEP	1m	Step size for CloudWatch metrics export.

AWS Configuration Properties

For further understanding of the configuration properties related to AWS, refer to [Spring Cloud AWS documentation](#).

Table 11. AWS Configuration Properties

Name	Env Variable	Default Value	Description
spring.cloud.aws.endpoint	FINERACT_AWS_ENDPOINT		The AWS service endpoint.
spring.cloud.aws.region.static	FINERACT_AWS_REGION_STATIC	us-east-1	The static region for AWS services.
spring.cloud.aws.credentials.access-key	FINERACT_AWS_CREDENTIALS_ACCESS_KEY		The AWS access key.
spring.cloud.aws.credentials.secret-key	FINERACT_AWS_CREDENTIALS_SECRET_KEY		The AWS secret key.
spring.cloud.aws.credentials.instance-profile	FINERACT_AWS_CREDENTIALS_INSTANCE_PROFILE	false	Whether to use the instance profile for credentials.
spring.cloud.aws.credentials.profile.name	FINERACT_AWS_CREDENTIALS_PROFILE_NAME		The name of the AWS credentials profile.
spring.cloud.aws.credentials.profile.path	FINERACT_AWS_CREDENTIALS_PROFILE_PATH		The path to the AWS credentials profile.

Resilience4j Properties

For a deeper understanding of resilience4j, refer to the [Official website](#)

Table 12. Resilience4j Properties

Name	Env Variable	Default Value	Description
fineract.retry.instances.executeCommand.max-attempts	FINERACT_COMMAND_PROCESSING_RETRY_MAX_ATTEMPTS	3	The number of attempts that resilience4j will attempt to execute a command after a failed execution. Refer to org.apache.fineract.commands.service.SynchronousCommandProcessingService#executeCommand for more details
fineract.retry.instances.executeCommand.wait-duration	FINERACT_COMMAND_PROCESSING_RETRY_WAIT_DURATION	1s	The fixed time value that the retry instance will wait before the next attempt can be made to execute a command
fineract.retry.instances.executeCommand.enable-exponential-backoff	FINERACT_COMMAND_PROCESSING_RETRY_ENABLE_EXponential_BACKOFF	true	If set to true, the wait-duration will increase exponentially between each retry to execute a command
fineract.retry.instances.executeCommand.exponential-backoff-multiplier	FINERACT_COMMAND_PROCESSING_RETRY_EXponential_BACKOFF_MULTIPLIER	3	The multiplier for exponential backoff, this is useful only when enable-exponential-backoff is set to true
fineract.retry.instances.executeCommand.retry Exceptions	FINERACT_COMMAND_PROCESSING_RETRY_EXCEPTIONS	org.springframework.dao.ConcurrencyFailureException,org.eclipse.persistence.exceptions.OptimisticLockException,jakarta.persistence.OptimisticLockException,org.springframework.orm.jpa.JpaOptimisticLockingFailureException,org.apache.fineract.infrastructure.core.exception.IdempotentCommandProcessUnderProcessingException	This property specifies the list of exceptions that the execute command retry instance will retry on

Name	Env Variable	Default Value	Description
resilience4j.retry.instances.processJobDetailForExecution.max-attempts	FINERACT_PROCESS_JOB_DETAIL_RETRY_MAX_ATTEMPTS	3	The number of attempts that resilience4j will attempt to process job details for execution. Refer to org.apache.fineract.infrastructure.jobs.service.JobRegisterServiceImpl#processJobDetailForExecution for more details
resilience4j.retry.instances.processJobDetailForExecution.wait-duration	FINERACT_PROCESS_JOB_DETAIL_RETRY_WAIT_DURATION	1s	The fixed time value that the retry instance will wait before the next attempt can be made
resilience4j.retry.instances.processJobDetailForExecution.enable-exponential-backoff	FINERACT_PROCESS_JOB_DETAIL_RETRY_ENABLE_EXPONENTIAL_BACKOFF	true	If set to true, the wait-duration will increase exponentially between each retry to process job detail
resilience4j.retry.instances.processJobDetailForExecution.exponential-backoff-multiplier	FINERACT_PROCESS_JOB_DETAIL_RETRY_EXPONENTIAL_BACKOFF_MULTIPLIER	2	The multiplier for exponential backoff, this is useful only when enable-exponential-backoff is set to true
resilience4j.retry.instances.recalculateInterest.max-attempts	FINERACT_PROCESS_RECALCULATE_INTEREST_RETRY_MAX_ATTEMPTS	3	The number of attempts that resilience4j will attempt to run recalculate interest. Refer to org.apache.fineract.portfolio.loanaccount.service.LoanWritePlatformServiceJpaRepositoryImpl#recalculateInterest for more details
resilience4j.retry.instances.recalculateInterest.wait-duration	FINERACT_PROCESS_RECALCULATE_INTEREST_RETRY_WAIT_DURATION	1s	The fixed time value that the retry instance will wait before the next attempt can be made

Name	Env Variable	Default Value	Description
resilience4j.retry.instances.recalculateInterest.enable-exponential-backoff	FINERACT_PROCESS_RECALCULATE_INTEREST_RETRY_ENABLE_EXPONENTIAL_BACKOFF	true	If set to true, the wait-duration will increase exponentially between each retry to recalculate interest
resilience4j.retry.instances.recalculateInterest.exponential-backoff-multiplier	FINERACT_PROCESS_RECALCULATE_INTEREST_RETRY_EXPONENTIAL_BACKOFF_MULTIPLIER	2	The multiplier for exponential backoff, this is useful only when enable-exponential-backoff is set to true
resilience4j.retry.instances.recalculateInterest.retryException	FINERACT_PROCESS_RECALCULATE_INTEREST_RETRY_EXCEPTIONS	org.springframework.dao.ConcurrencyFailureException,org.eclipse.persistence.exceptions.OptimisticLockException,jakarta.persistence.OptimisticLockException,org.springframework.orm.jpa.JpaOptimisticLockingFailureException	This property specifies the list of exceptions that the recalculateInterest retry instance will retry on
resilience4j.retry.instances.postInterest.max-attempts	FINERACT_PROCESS_POST_INTEREST_RETRY_MAX_ATTEMPTS	3	The number of attempts that resilience4j will attempt to run post interest. Refer to org.apache.fineract.portfolio.loanaccount.service.LoanWritePlatformServiceJpaRepositoryImpl#postInterest for more details
resilience4j.retry.instances.postInterest.wait-duration=	FINERACT_PROCESS_POST_INTEREST_RETRY_WAIT_DURATION	1s	The fixed time value that the retry instance will wait before the next attempt can be made
resilience4j.retry.instances.postInterest.enable-exponential-backoff	FINERACT_PROCESS_POST_INTEREST_RETRY_ENABLE_EXPONENTIAL_BACKOFF	true	If set to true, the wait-duration will increase exponentially between each retry to post interest

Name	Env Variable	Default Value	Description
resilience4j.retry.instances.postInterest.exponential-backoff-multiplier	FINERACT_PROCESS_POST_INTEREST_RETRY_EXPONENTIAL_BACKOFF_MULTIPLIER	2	The multiplier for exponential backoff, this is useful only when enable-exponential-backoff is set to true
resilience4j.retry.instances.postInterest.retryExceptions	FINERACT_PROCESS_POST_INTEREST_RETRY_EXCEPTIONS	org.springframework.dao.ConcurrencyFailureException,org.eclipse.persistence.exceptions.OptimisticLockException,jakarta.persistence.OptimisticLockException,org.springframework.orm.jpa.JpaOptimisticLockingFailureException	This property specifies the list of exceptions that the post interest retry instance will retry on

Appendix B: Third Party Software

TBD