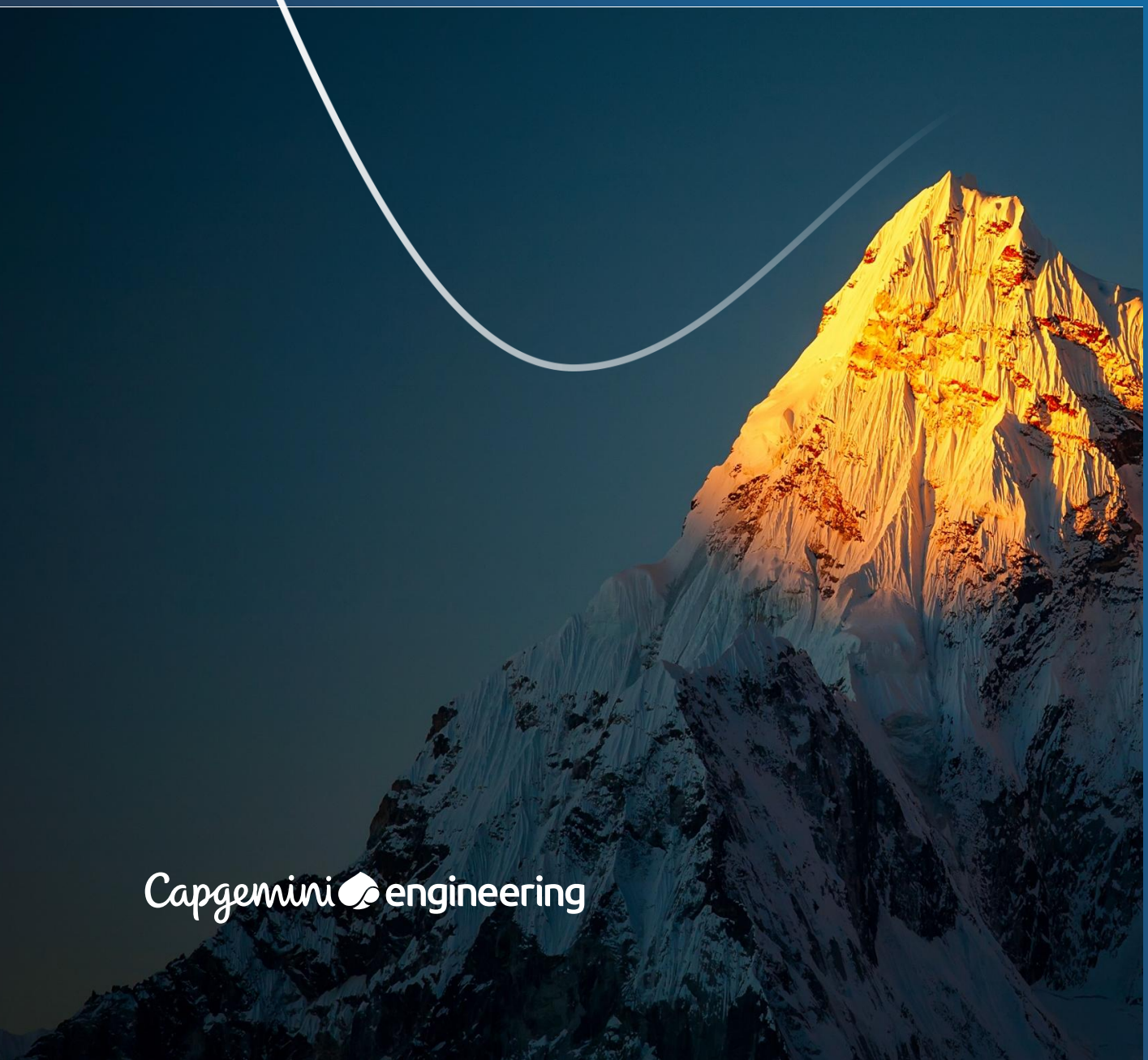
TransCelerate SDR RI *System Maintenance Guide V2.1*



Capgemini engineering

Content

Introduction

Purpose of this Document

This System Maintenance Guide is intended to provide technical direction for developers or DevOps engineers working with the SDR Reference Implementation[1] solution. It includes activities such as:

- Building, configuring, and running the application locally
- Building, configuring, and running the application in containers
- Running unit tests
- Maintenance activities

Relationship to Other Documentation

This document is not an exhaustive overview of the SDR RI system, its design, or its implementation. We defer to other key documents for additional detail, rather than repeat them here:

- DDF SDR RI Solution Architecture

We have, however, consolidated into this document technical steps from the README.md files of the three separate GitHub repositories that make up the SDR RI Solution:

- https://github.com/transcelerate/ddf-sdr-platform
- https://github.com/transcelerate/ddf-sdr-api
- https://github.com/transcelerate/ddf-sdr-ui

We have done this to create a central maintenance guide, rather than one split across repositories, and to allow us to record corrections and additional detail to that recorded in the READMEs.

Changes for Release V5.0 (September 2025)

SDR Release V5.0 marks a fundamental shift from previous versions by eliminating Azure dependencies from its architecture, more easily enabling platform-agnostic deployment capabilities across various environments.

[1] The SDR RI is not a commercial product, rather it is TransCelerate's attempt to illustrate what might be possible in implementing the USDM developed by CDISC. To the extent that the SDR Reference Implementation incorporates or relies on any specific branded products or services, this resulted out of the practical necessities associated with making a reference implementation available to demonstrate the SDR's capabilities. TransCelerate does not endorse any particular software, system, or service. And the use of specific brands of products or services by TransCelerate and its collaboration partners in developing the SDR Reference Implementation should not be viewed as any endorsement of such products or services. Users can use the USDM for any purpose they choose and can build their own implementations of the SDR using the resources available on GitHub.

Infrastructure

Overview

The DDF SDR Platform GitHub repository provides layered and environment-specific Docker Compose configurations which aims to repeatably and reliably run a multi-container SDR Reference Implementation (RI). This includes core containerized services such as:

- A MongoDB container service as the document database
- A .NET API container service with CDISC Rules Engine tool for USDM conformance validation
- An Angular UI container service

The full architecture of this platform is described in more detail in existing documentation and is not repeated here:

- DDF SDR RI Solution Architecture

Database

Prerequisites

The following prerequisites must be installed to the environment first.

- MongoDB Community Server for local development

  or

- Docker Engine and Docker Compose for running Docker Compose service

Required Collections

The following Mongo database collections are used by the SDR API. If starting from a clean database, they must be created in the database prior to starting the API:

- StudyDefinitions
- ChangeAudit

Restoring from Mongo Dump

Exports of the demo databases have been placed in TransCelerate SharePoint. These consist of metadata.json files and .bson data files (one of each per database collection).

These can be restored to a target Mongo DB instance using the mongorestore database tool, which is available as part of the MongoDB Database Tools. Install mongotools and place the mongorestore executable in the same directory as the database snapshot files.

Restoration can then be completed one collection at a time using

```
mongorestore --host HOST:PORT --authenticationDatabase admin -u USERNAME -p PASSWORD --db SDR --collection COLLECTION --writeConcern="{w:0}" --ssl COLLECTION.bson
```

Where

- HOST
- PORT
- USERNAME
- PASSWORD

can all be obtained from the Connection Settings of the Mongo Database, and

- COLLECTION

is the name of the collection being restored (e.g. Study)

Running Container

Environment-specific Configuration Variables

The `.env` file contains environment-specific configuration variables used by Docker Compose and the containerized applications. This file should never be committed to the repository as it may contain sensitive information; hence it is excluded in the `.gitignore` file.

Copy the provided `.env.template` file to create your own `.env` file. Edit the `.env` file and replace the placeholder values with your actual configuration values.

Run MongoDB Container Service

MongoDB container service can be run using the base docker-compose.yml. To start the service using Docker Compose, use the following command:

```
docker compose --env-file .env up -d
```

API

Prerequisites

The following prerequisites must be installed to the environment first.

- .NET 8
- Visual Studio 2022 or Visual Studio Code IDE
- Docker Engine for running Docker container
- A local download of the CDISC Core Rules Engine executable

Building and Running Locally

Copy the `appsettings.json` file to create an `appsettings.Development.json` file in the root folder of TransCelerate.SDR.WebApi project.

Edit the file to provide details for local development. Key fields to update are:

- `ConnectionStrings`: `DefaultConnection` should be a full connection string (including required authentication details); `DatabaseName` should be `SDR`.
- `StudyHistory.DateRange`: Set to a numeric value, e.g. `"30"`
- `"ApiVersionUsdmVersionMapping": "{\"SDRVersions\":[{\"apiVersion\":\"v3\",\"usdmVersions\":[\"2.0\"]},{\"apiVersion\":\"v4\",\"usdmVersions\":[\"3.0\"]},{\"apiVersion\":\"v5\",\"usdmVersions\":[\"4.0\"]}]}",`
- `"CdiscRulesEngine"`
- `"CdiscRulesEngineRelativeBinary"`
- `"CdiscRulesEngineRelativeCache"`

Where the last three are the CDISC Rules Engine root directory path, core binary file, and relative cache directory path respectively - for example:

`"CdiscRulesEngine": "C:\\core-windows\\core"`

`"CdiscRulesEngineRelativeBinary": "core.exe"` (or `"core"` on Linux)

`"CdiscRulesEngineRelativeCache": "resources\\cache"`

The API can then be run from:

- Visual Studio with the 'Run (F5)' command
- Visual Studio Code with `dotnet build` and `dotnet run` in the TransCelerate.SDR.WebApi directory

The Swagger endpoint will open in a browser window.

Building and Running Container

To build and run the API container using Docker, use the following commands:

Build the image from the Dockerfile in the DDF-SDR-API directory with build arguments:

```
docker build -t ddf-sdr-api:latest \
  --build-arg ASPNETCORE_ENVIRONMENT={ENVIRONMENT} \
  --build-arg CdiscRulesEngine_LATEST_RELEASE_URL={CDISC_RULES_ENGINE_LATEST_RELEASE_URL} \
  --build-arg CdiscRulesEngine_LATEST_RELEASE_ZIP={CDISC_RULES_ENGINE_LATEST_RELEASE_ZIP} \
  --build-arg CdiscRulesEngine={CDISC_RULES_ENGINE} \
  --build-arg CdiscRulesEngineRelativeBinary={CDISC_RULES_ENGINE_RELATIVE_BINARY} \
  --build-arg CdiscRulesEngineRelativeCache={CDISC_RULES_ENGINE_RELATIVE_CACHE}
  .
```

Where

- ENVIRONMENT

can be `Development` or `Production`, and

- CDISC_RULES_ENGINE
- CDISC_RULES_ENGINE_RELATIVE_BINARY
- CDISC_RULES_ENGINE_RELATIVE_CACHE

are the CDISC Rules Engine root directory path, core binary file, and relative cache directory path respectively

For example:

```
CdiscRulesEngine=/app/cdisc-rules-engine
CdiscRulesEngineRelativeBinary=core
CdiscRulesEngineRelativeCache=resources/cache
```

As of September 2025, the latest CDISC Rules Engine release v0.12.0 does not perform well with large and nested JSON datasets. As a result, the SDR uses a custom-built binary with performance improvements from a forked CDISC Rules Engine repository.

- CDISC_RULES_ENGINE_LATEST_RELEASE_URL
- CDISC_RULES_ENGINE_LATEST_RELEASE_ZIP
- CDISC_RULES_ENGINE_RELATIVE_BINARY

should point to this forked repository until CDISC releases an updated binary with the performance improvements. After the official release, these environment variables no longer need to be passed in as default values are set to the CDISC Rules Engine official release within the Dockerfile.

For example:

```
CdiscRulesEngine_LATEST_RELEASE_URL=https://api.github.com/repos/transcelerate/cdisc-rules-engine/releases/latest
```

```
CdiscRulesEngine_LATEST_RELEASE_ZIP=cdisc-core-ubuntu.zip
```

```
CdiscRulesEngineRelativeBinary=cdisc-core-20250903
```

Run the container:

```
docker run \
  -e "ConnectionStrings__DefaultConnection=mongodb://{MONGO_ROOT_USER}:{MONGO_ROOT_PASS}@mongodb:27018/{MONGO_DATABASE}?authSource=admin" \
  -e "ConnectionStrings__DatabaseName={MONGO_DATABASE}" \
  -e "StudyHistory__DateRange=30" \
  -e "ApiVersionUsdmVersionMapping={API_VERSION_USDM_VERSION_MAPPING}" \
  -e "ASPNETCORE_ENVIRONMENT={ENVIRONMENT}" \
  -v cdisc_rules_engine_cache:{CDISC_RULES_ENGINE}/{CDISC_RULES_ENGINE_RELATIVE_CACHE} \
  -p 8080:80 \
  --name ddf-sdr-api \
  ddf-sdr-api:latest
```

Where

- MONGO_DATABASE

should be `SDR`,

- API_VERSION_USDM_VERSION_MAPPING

should be `{"SDRVersions":[{"apiVersion":"v3","usdmVersions":["2.0"]},{"apiVersion":"v4","usdmVersions":["3.0"]},{"apiVersion":"v5","usdmVersions":["4.0"]}]}`, and

- CDISC_RULES_ENGINE
- CDISC_RULES_ENGINE_RELATIVE_CACHE

should be the same values as above

The `StudyHistory__DateRange` value will be used to restrict the historical data (last 30/60/90 days) in the study history API endpoint response, if no date filters are passed in request. Keep this value as "-1" to disable this restriction.

Updating CDISC Rules Engine Rule Cache

CDISC Rules are released continuously on the CDISC Library (https://www.cdisc.org/cdisc-library). These rules are cached and persisted in a Docker volume attached to the API Docker Container where the CDISC Rules Engine also resides. In order to update this rules cache, run the update cache command using docker exec:

```
docker exec -e CDISC_LIBRARY_API_KEY={CDISC_LIBRARY_API_KEY} ddf-sdr-api core update-cache
```

A CDISC Library API Key is required to perform this update cache. To obtain an API key, please follow the instructions found here: https://wiki.cdisc.org/display/LIBSUPRT/Getting+Started%3A+Access+to+CDISC+Library+API+using+API+Key+Authentication. Please note it can take up to an hour after signing up to have an API key issued

Running Tests

Unit Tests

There are two options for running the unit tests for the SDR API solution:

Option 1: Using Visual Studio IDE

1. Open the SDR API solution in Visual Studio
2. Navigate to the UnitTesting project in the Solution Explorer
3. Right-click on the UnitTesting project and select “Run Tests”

Alternatively, the Test Explorer window can be used to run the same unit tests.

Option 2: Using Visual Studio Code **(Requires C# Dev Kit Extension)**

1. Open the SDR API src folder in VS Code
2. Navigate to the Testing view and “Run Tests”

Test Coverage

Two options for running test coverage for the API are as follows:

Option 1: Using Visual Studio’s Built-in code coverage feature **(Requires Enterprise Edition)**

1. Open the SDR API solution in Visual Studio
2. On the “Test” menu, select “Analyze Code Coverage for All Tests”
3. After tests run, a “Code Coverage Results” window will appear showing coverage statistics

Alternatively, the Test Explorer window can be used to run the same unit tests.

Option 2: Using Visual Studio Code **(Requires C# Dev Kit Extension)**

1. Open the SDR API src folder in VS Code
2. Navigate to the Testing view and “Run Tests with Coverage”

User Interface

Prerequisites

The following prerequisites must be installed to the environment first.

- Node JS
- NPM
- Angular CLI
- Visual Studio Code IDE
- Docker Engine for running Docker container

These are the versions we tested with:

- Node 23.6.0
- NPM 11.3.0
- Angular CLI 19.2.7

Building and Running Locally

From the SDR-WebApp directory, install required package dependencies:

```
npm install
```

Copy `SDR-WebApp\src\environments\environment.ts` to create an `environment.development.ts` file and configure API endpoint and environment name.

The web application can then be run locally with

```
ng serve
```

Building and Running Container

To build and run the UI container using Docker, use the following commands:

Build the image from the Dockerfile in the DDF-SDR-UI directory:

```
docker build -t ddf-sdr-ui:latest .
```

Run the container:

```
docker run \
  -e "PRODUCTION={IS_PRODUCTION}" \
  -e "APP_API_BASE_URL={API_BASE_URL}" \
  -e "envName={ENVIRONMENT}" \
```

```
  -p 4200:80 \
  --name ddf-sdr-ui \
  ddf-sdr-ui:latest
```

Where

- IS_PRODUCTION

can be `true` or `false`

- ENVIRONMENT

can be `Dev` or `Prod`, and

- API_BASE_URL

Points to the URL to target (e.g. `http://localhost:8080` if running against a locally developed API instance).

Running Tests

Unit Tests

The User Interface (UI) unit tests use Karma test runner to execute the tests in actual browsers. Karma is a test runner for JavaScript applications. Karma test runner launches browsers, executes test scripts, and reports results. The tests can be executed locally within the *SDR-WebApp* directory via Angular CLI with the command:

```
ng test --watch=false --browsers ChromeHeadless
```

Alternatively, the JUnit reporter can be used to generate a test result file named `unit-test-results.xml` in the `SDR-WebApp/testresults/junit` directory with the command:

```
ng test --reporters junit --watch=false --browsers ChromeHeadless
```

Currently, there are 196 unit tests passing successfully.

Test Coverage

A test coverage summary, reporting how much of the code is being executed by the unit tests, can also be produced by executing the following command:

```
ng test --watch=false --browsers ChromeHeadless --code-coverage
```

The generated report is written to `SDR-WebApp/testresults/coverage` directory.

End-to-End Tests

There is no end-to-end (e2e) testing implemented.

Maintenance Activities

Updates to USDM Version

Execute the following steps to update the SDR to support newer versions of the CDISC USDM:

1. Obtain required deliverables from the CDISC GitHub – most important are:
    - UML Delta corresponding to the intended version update.
    - Latest Core Rules (each delta will be a single tab in the document).
    - API JSON (deltas can be generated with a JSON diff tool)
2. Make the required changes to the API solution. This typically will require creating / editing:
    - Entity classes in the `src\TransCelerate.SDR.Core\Entities` directory.
    - Corresponding DTO classes in `src\TransCelerate.SDR.Core\DTO`.
    - Entity / DTO mapping in `src\TransCelerate.SDR.WebApi\Mappers`.
    - Rules Validators in `src\TransCelerate.SDR.RuleEngine`.
        - Note: Rules Validators update is only applicable for SDR versions prior to V5.0. SDR RI V5.0 integrates CDISC Rules Engine for USDM V4.0 conformance validation. The integration utilities and classes are also located in the `src\TransCelerate.SDR.RuleEngine`.
    - Updates to services in `src\TransCelerate.SDR.Service\Services`.
3. Determine if any API Endpoint changes are required
    - In most cases, we expect the changes reflected in the API JSON will be solely due to changes in the underlying model structure (rather than changes to endpoint structure or functionality), so there may not be much work to do here.
4. Create updated test data and test the API for correctness (e.g. using Postman)
5. Evaluate the UI for any issues.
    - We don't generally expect much in the way of UI changes – but it is possible that you may identify defects, representation improvements etc. as a result of model changes made to the API.

About Capgemini Engineering

World leader in engineering and R&D services, Capgemini Engineering combines its broad industry knowledge and cutting-edge technologies in digital and software to support the convergence of the physical and digital worlds. Coupled with the capabilities of the rest of the Group, it helps clients to accelerate their journey towards Intelligent Industry. Capgemini Engineering has 65,000 engineer and scientist team members in over 30 countries across sectors including Aeronautics, Space, Defense, Naval, Automotive, Rail, Infrastructure & Transportation, Energy, Utilities & Chemicals, Life Sciences, Communications, Semiconductor & Electronics, Industrial & Consumer, Software & Internet.

Capgemini Engineering is an integral part of the Capgemini Group, a global business and technology transformation partner, helping organizations to accelerate their dual transition to a digital and sustainable world, while creating tangible impact for enterprises and society. It is a responsible and diverse group of 340,000 team members in more than 50 countries. With its strong over 55-year heritage, Capgemini is trusted by its clients to unlock the value of technology to address the entire breadth of their business needs. It delivers end-to-end services and solutions leveraging strengths from strategy and design to engineering, all fueled by its market leading capabilities in AI, generative AI, cloud and data, combined with its deep industry expertise and partner ecosystem. The Group reported 2024 global revenues of €22.1 billion.

Get the future you want | www.capgemini.com