**Digital Regulatory Reporting**

The Global Digital Regulatory Reporting (DRR) program is a cross industry initiative to transform the reporting infrastructure so that all reporting entities implement rules consistently and cost effectively. DRR achieves this by delivering an open-source, functional expression of the reporting rules that can be used as machine-executable code in firms’ own implementations.

DRR is built as an extension of the CDM. It uses the CDM to represent the transaction data input into the reporting process. It conforms to the CDM’s design principles where data and functions are used to model reportable fields and logic, respectively.

*Background*

Financial institutions are subject to an array of global reporting obligations designed to provide transparency about their activities to regulators and ensure the safe functioning of financial markets, including the monitoring of systemic risk or market abuse. Derivatives transactions, following the global financial reform initiated in 2009, are now transparently reported to trade repositories across all G20 jurisdictions. More recently, similar reporting obligations have been extended to the Securities Financing markets.

Since they were first implemented, the regulatory community has progressed to harmonise global trade reporting rules under the BIS CPMI-IOSCO's working group. National regulators and reporting parties have been tasked with implementing these recommendations consistently across global markets.

DRR supports that next wave of derivatives reporting regulations, allowing firms to comply efficiently and cost effectively and regulators to achieve their policy goals. The CDM has demonstrated that it can serve as the basis for this new reporting paradigm. Previous implementations that led up to this DRR program include the [UK Digital Regulatory Reporting Pilot](https://www.isda.org/2019/05/21/isda-cdm-deployed-to-help-deliver-uk-digital-regulatory-reporting-pilot/) initiative in 2019 and the [G20 TechSprint](https://www.isda.org/2020/10/06/isda-and-regnosys-win-g-20-techsprint-for-regulatory-reporting/) in 2020.

*Scope and status*

DRR covers several trade & transaction reporting regimes that have been developed through various implementation initiatives, each at different stages of maturity.

The following table captures the list of regimes covered and their implementation stage (as of March 1st 2022).

|  |  |  |
| --- | --- | --- |
| **Regime** | **Initiative** | **Status** |
| CFTC Rewrite | Global DRR Program | In progress – Extensive coverage |
| EMIR Refit | Global DRR Program | In progress – Good coverage |
| EMIR | UK DRR Pilot | Proof-of-concept |
| MiFIR | UK DRR Pilot | Proof-of-concept |
| MAS | G20 TechSprint | Proof-of-concept |

*Process overview*



DRR consists of a sequence of 3 steps:

* **Translate**. This takes a firm’s transaction event data and translates it into a CDM object representing that transaction.
* **Report**. This takes the CDM transaction event object and applies the reporting logic (eligibility and field rules plus additional regulatory data guidelines) to produce a CDM report object.
* **Project**. This takes a CDM report object and applies additional mapping and projection rules to produce a report file in the format required by trade repositories and/or regulators (e.g. XML, ISO 20022).

DRR extends the CDM by modelling each of the above steps. The DRR output includes generated machine-executable code required by an implementor to run each step. Synthetic data samples for each step’s input and output are captured in a “Test Pack” that allows to validate any implementation against those tests.

*Using the DRR output*

Reporting firms can use the DRR output for their reporting implementation along 3 approaches that can be combined: Build/Benchmark/Buy.

* **Build**. A firm uses the open-source DRR model components and executable code artefacts to develop its own internal implementation.
  + They develop a run-time execution engine that sits on top of the reporting rules and deploy it on their infrastructure.
  + They integrate the DRR code artefacts into their software lifecycle management.
  + They develop their own translation from their internal data formats.
  + They use the Test Pack for quality assurance, running the input data through their implementation and comparing against the expected output.
* **Benchmark**. A firm uses the testing capabilities that are freely available under the Community Edition of the Rosetta Platform supporting the DRR and the CDM more broadly to validate their own reporting implementation.
  + These services are designed in the context of the program to support firms’ testing, proof-of-concept or benchmarking of their own implementations, but not production reporting systems – they can only accommodate limited volume and throughput.
  + These services cover all of *Translate*/*Report*/*Project* and are available both via a web interface or API.
  + The *Translate* and *Project* services only cover the formats that have been publicly developed and distributed in the CDM and DRR, not firms’ custom formats.
* **Buy**. A firm buys a reporting solution from a third-party vendor. That third-party vendor itself has followed the “Build” approach to develop their commercial product (rather than an internal reporting system) based on the DRR output.

The next 3 sections focus on the “Build” approach. They detail the artefacts included in the DRR output for each of the Translate/Report/Project steps and how reporting firms can use them to develop an implementation.

In each case, details of how reporting firms can use the equivalent “Benchmark” option in Rosetta for testing purposes are also provided.

***Translate***

Why

* To transform an internal messaging format into a CDM transaction event object. This CDM object is required as an input to the DRR reporting rules.

What

* Model-to-model mappings are available as compact “synonyms” in the CDM distribution *for public models only*.
* The list of supported public models can be found at: [https://docs.rosetta-technology.io/cdm/documentation/source/documentation.html#mapping-synonym](https://docs.rosetta-technology.io/cdm/documentation/source/documentation.html)
* Custom adaptations of public models would require these synonyms to be extended.

Where

* Synonyms are contained in the CDM distribution in *.rosetta* format. All synonym files in the distribution are contained in the *cdm.synonym.\** namespaces and prefixed accordingly.

How

Build

* Synonyms provide a functional specification for how to perform model-to-model mapping.
  + The semantics of synonyms is documented at: <https://docs.rosetta-technology.io/dsl/documentation.html#mapping-component>
* Synonyms are distributed as machine-readable artefacts so that implementors can use them as input to code-generate their translation’s executable code (in the same way as the CDM’s executable code is generated).
  + A guide for writing a code generator is at: <https://docs.rosetta-technology.io/dsl/codegen-readme.html>
* Depending on their internal model, implementors have three approaches to build their translation:
  + If they use a public model, they should write their own code generator to generate the translation’s executable code from the machine-readable synonym specification.
  + If they use a custom adaptation of a public model, they should extend that translation’s implementation by either:
    - extending the synonyms first (as documented at: <https://docs.rosetta-technology.io/dsl/documentation.html#synonym-source>), then code-generating from them, or
    - directly extending the executable code that they generate from the public model synonyms.
  + If they use a fully bespoke model that is not adapted from a public one, they will not be able to use the public synonyms and will need to create their own custom translation implementation.

Benchmark

* An ingestion service that translates *from a publicly available model only* into CDM is available on the Rosetta Platform and documented here: <https://docs.rosetta-technology.io/core/4-api-export.html#ingestion-service>

***Report***

Why

* To generate a reportable output object in CDM format based on a CDM transaction event input.

What

* A report is defined by 3 components: what (report fields), whether (eligibility) and when (timing). The “what” is represented by a CDM data type whose attributes are the reportable fields.
* Each reportable field is associated with a `reporting rule` component representing the logic to extract or compute that field from a CDM transaction event object.
* The DRR distribution contains a library component that takes the name of the report and a CDM transaction event object as input and returns a report object. That library is available as a Java JAR and compiled with Java 11, which is required for use.

Where

* The report and rule definitions are available as *.rosetta* files in the DRR distribution. The files are contained in the *cdm.regulation.\** namespaces and prefixed accordingly – e.g. *cdm.regulation.cftc.rewrite*.
* The distribution also contains a library called `rosetta-reports` to help execute the DRR rules. As this library is protected by copyright, it is only intended to guide implementors in their own build but should not directly support production systems.
* Work contemplated for future stage:
  + Aligning the code generation of DRR rules (`reporting rule`) onto that of Rosetta functions (`func`) would facilitate implementors’ build of their own execution engine, by providing a single way of using the CDM’s business logic as executable code.

How

Build (test only)

* The DRR Java code containing the reporting rules can be either:
  + downloaded from the Rosetta application – see: <https://docs.rosetta-technology.io/core/1-workspace.html#download-workspace>, or
  + added as a code dependency (using maven or gradle) – e.g. in maven:

<dependency>

<groupId>com.regnosys.drr</groupId>

<artifactId>rosetta-source</artifactId>

<version>LATEST</version>

</dependency>

In particular, this dependency gives access to the generated Java class representing a specific report object, called a “Blueprint” in Rosetta – e.g. `CFTCPart45BlueprintReport`.

* The `rosetta-reports` library, which gives access to the `ReportApi` Java class required to execute a report, can be added as a code dependency – e.g. using maven:

<dependency>

<groupId>com.regnosys</groupId>

<artifactId>rosetta-reports</artifactId>

<version>LATEST</version>

</dependency>

* To execute a report for a particular DRR version requires the following steps:
  + Create a new `ReportApi` with the following parameters:
    - body – the body of the report that this class will generate, e.g. "CFTC"
    - corpusList – a list of corpus for the report that this class generates, e.g. List.of("Part45")
    - name – the name of the report to run, e.g. "CFTCPart45"
    - blueprintClass – the class type of the report to run, e.g. “CFTCPart45BlueprintReport.class”
    - initialisationGuiceModule – the Guice module used to initialise the report runner. The convention for the CDM is: “CdmRuntimeModule.class”.
  + Create an input CDM object representing the transaction (always a `RosettaModelObject` sub-type, e.g. `WorkflowStep`) by either:
    - converting JSON to Java by using a Jackson Object Mapper (the `RosettaObjectMapper` utility helps set things up),
    - creating a `WorkflowStep` by Ingesting FpML or other external models into the CDM,
    - creating a `WorkflowStep` using Java code, or
    - extracting a `WorkflowStep` from a CDM native implementation.
  + Run the report based on that input CDM object (`inputData`), to return either:
    - a structured object, according to the report’s specified data type

public <T extends RosettaModelObject> T runStructuredReport(RosettaModelObject inputData)

* + - a table, i.e. a set of key-value pairs

public Map<String, String> runTabularReport(RosettaModelObject inputData)

* Below is a full example using the “CFTC Part 45” reporting regime:

package com.regnosys.drr.examples;

import cdm.event.workflow.WorkflowStep;

import cdm.regulation.cftc.rewrite.CFTCPart45TransactionReport;

import cdm.regulation.cftc.rewrite.blueprint.CFTCPart45BlueprintReport;

import com.regnosys.rosetta.common.serialisation.RosettaObjectMapper;

import com.regnosys.rosetta.reports.api.ReportApi;

import org.isda.cdm.CdmRuntimeModule;

import java.io.File;

import java.io.IOException;

import java.util.List;

public class ReportApiTest {

public static void main(String[] args) throws IOException {

// 1. Create the Report Api

ReportApi reportApi = new ReportApi(

"CFTC",

List.of("Part45"),

"CFTCPart45",

CFTCPart45BlueprintReport.class,

CdmRuntimeModule.class);

// 2. Deserialise a Workflow Step JSON into Java

WorkflowStep inputData = RosettaObjectMapper.getNewRosettaObjectMapper().readValue(new File("FILE-LOCATION-OF-WORKFLOW-STEP-JSON"), WorkflowStep.class);

// 3. Run the Api and product a CFTCPart45TransactionReport

CFTCPart45TransactionReport report = reportApi.runStructuredReport(inputData);

// process the result

System.out.println(report);

}

}

* Note: A similar code example is planned to be contributed as a demo project into the DRR distribution and will be available to download.

Benchmark

* A reporting service that packages all the above steps is available on the Rosetta Platform and documented here: <https://docs.rosetta-technology.io/core/4-api-export.html#regulation-report-service>

***Project***

Why

* To convert a CDM reportable output object into a message that is ready to be sent to a Trade Repository.

What/Where/How

* This component has not been built yet.