Integrated Clearing

CDML Developer Guide

Oct 23, 2014

ABSTRACT

A guide on the CDML framework and translators

1 Summary

This document briefly describes the CDML framework, its purpose and design, and how to write CCP EOD report translators to produce CDML documents that will be handled by the aforementioned framework.

It is aimed to developers and other personnel involved in creating and managing CDML content, and it requires some basic **XML, XPath, XSD, Java, Spring and Clearing** knowledge. It is also recommended to have a copy of the CDML schemas[1].

The CDML framework is still work in progress. You should expect changes in future Clearing releases, and have always the latest copy of all published documents.

1 *See 16 Related documentation*

2 Table of Contents

3 Changes

Mar 02, 2014	Eduardo Corral	Started initial version for the **CDML v2 specification**
Mar 17, 2014	Eduardo Corral	Updates after Alec Sullivan's feedback • Added CDML versioning section • Typos/grammar fixes
Oct 23, 2014	Eduardo Corral	Updated for 2.11.0/3.6.0 releases, including • **Updated CDML specification to v3** • Added ITD (intraday) flag • XML-to-XML translation producers

4 Definitions

CCP	Central Counterparty
CDML	Clearing Data Markup Language
CUP	Calypso Upgrade Package
EOD	End Of Day
IM	Initial Margin
ITD	Intraday
MCC	Margin Call Configuration (also *Collateral Config*)
OOTB	Out Of The Box
OS	Operating System
TV	Trade Valuation
VM	Variation Margin
XSD	XML Schema Definition

5 What is CDML?

One of the fundamental pillars of the Integrated Clearing solution is the **ITD and EOD CCP/member report processing**. All clearing members, and clearing clients, feed their systems with margin, trade, market data and other kinds of information provided by CCPs/members. This information takes different forms, splitted into several reports, and with different encoding standards.

CDML is a **collection of XML based formats** that aims to normalize this reporting, **in a CCP-agnostic fashion**. The final goal is to be able to record all relevant ITD and EOD CCP/member information in a clear and unified way, with no CCP/member-specific structures or terminology.

This document will focus on the **CCP and clearing member relationship**, but most concepts can also apply to the one between clearing members and client clearing customers.

6 The CDML flow: translation and processing

The CDML flow can be divided into two simple phases: **translation and processing**.

During translation, CCP-specific content is sorted, aggregated or split, and transformed into CCP-agnostic CDML content.

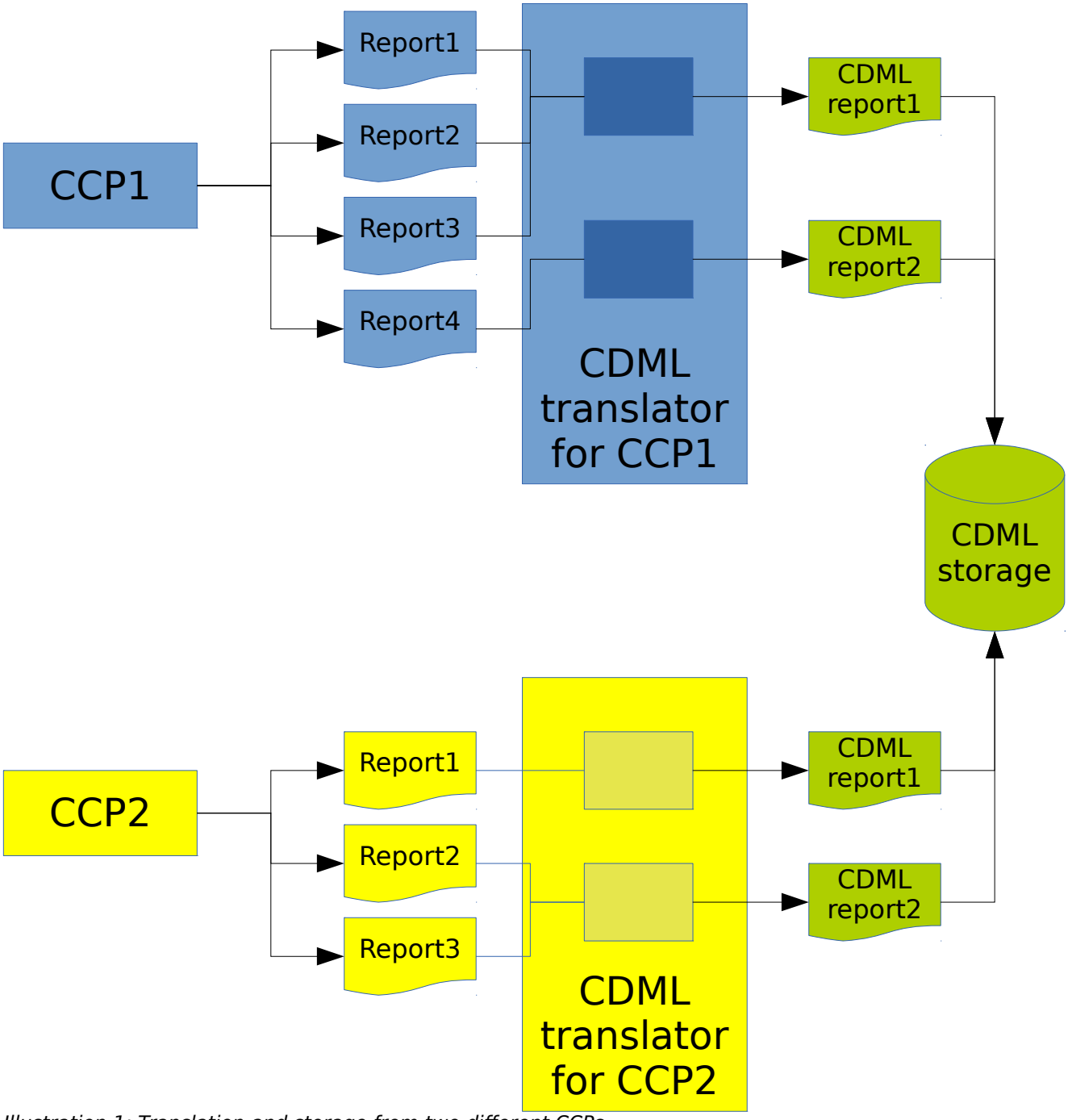


Illustration 1: Translation and storage from two different CCPs

Once CDML is stored, it is ready for processing. During this phase, the content is

analyzed, and downstream core Calypso objects, like trades or `PLMarks`, are created.

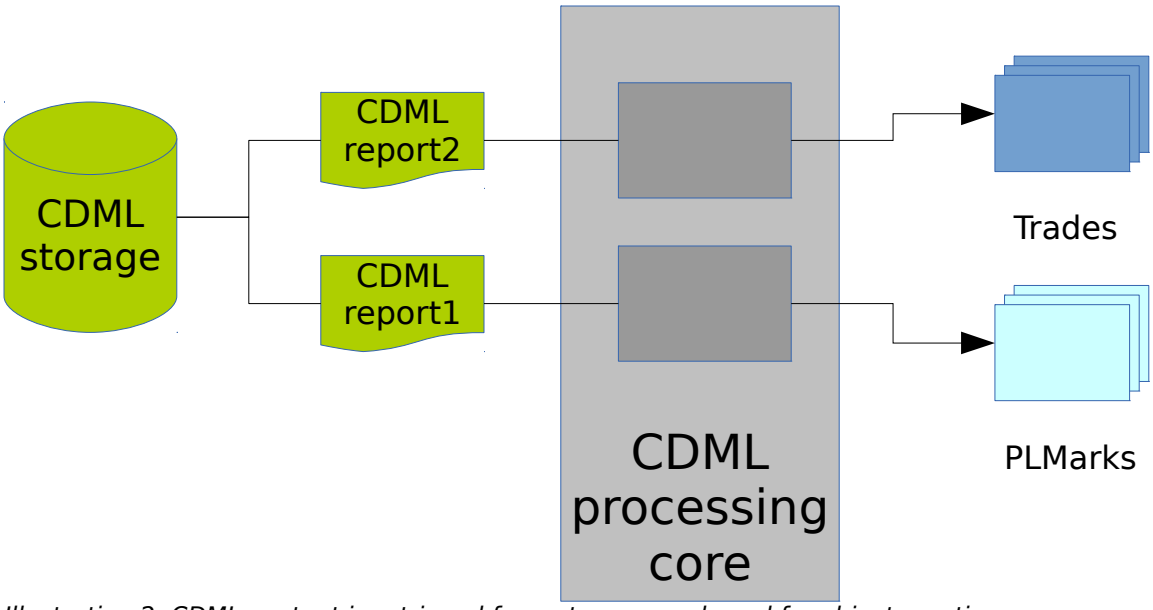


Illustration 2: CDML content is retrieved from storage, and used for object creation

7 CDML types

7.1 General characteristics

As mentioned before, CDML is a **collection of XML based formats**. Each of them is defined in **its own schema**, although they share **datatypes and a common structure**.

All CDML report types are capable of **simultaneously** representing information from **multiple CCPs, multiple members, clearing services (products) and accounts**.

All information contained in a single CDML report pertains to the **same business date**: such date is one of the fundamental characteristics of the report (see below).

7.2 Schema namespaces

The following are all the schemas currently defined within the CDML specification.

7.2.1 urn:cdml:schema:common:types

URI	urn:cdml:schema:common:types
Sample prefix[2]	cdml
Description	Common CDML datatypes namespace

The common namespace also defines the `CDMLReportType`, which is the **mandatory base type** for all CDML reports. From the schema itself

```
<complexType name="CDMLReportType">
   <annotation>
      <documentation>Base type for all CDML reports</documentation>
   </annotation>
   <sequence>
      <element name="reportDate" type="cdml:ReportDateType" minOccurs="1" maxOccurs="1">
         <annotation>
            <documentation>Business date on which the report data is valid</documentation>
         </annotation>
      </element>
      <element name="intraday" type="boolean" minOccurs="0" maxOccurs="1" default="false">
         <annotation>
            <documentation>Indicates if the data in this report has been generated intraday. The lack
             of this flag should be interpreted as it being false</documentation>
         </annotation>
      </element>
   </sequence>
   <attribute name="modelVersion" type="cdml:ReportModelVersionType" use="required">
      <annotation>
         <documentation>Report specification version number</documentation>
      </annotation>
   </attribute>
   <attribute name="version" type="cdml:ReportVersionType" use="optional">
```

2 *Namespace prefix to be used in this document. Prefixes are arbitrary, see http://en.wikipedia.org/wiki/XML_namespace#Namespace_declaration*

```
      <annotation>
         <documentation>Report content version number</documentation>
      </annotation>
   </attribute>
   <attribute name="generationDateTime" type="cdml:ReportGenerationDateTimeType" use="optional">
      <annotation>
         <documentation>Report generation timestamp</documentation>
      </annotation>
   </attribute>
</complexType>
```

All CDML reports then share

- A well defined, ISO 8601 **report date**, which represents the business date the report information is related to.
- ***NEW*** An optional **intraday** flag to signal downstream processes that the data gathered is from ITD sources.
- A **model version,** which is the CDML specification version. This attribute will be used for migration and verification purposes.
- An **optional version**, which is the iteration of the report content, for that specific type and date. It is recommended not to include it when generating CDML, as its usage is reserved to the CDML framework. See CDML storage.
- An **optional generation timestamp**. See CDML storage.

All of the following CDML snippets are to be considered sample data, and shouldn't be interpreted as actual production content.

7.2.2 urn:cdml:schema:margin:initialMargin

URI	urn:cdml:schema:margin:initialMargin
Sample prefix	cdml-im
Description	`initialMarginReport` related datatypes and root element definition

The `initialMarginReport` contains information about **margin requirements**, broken down by **CCP, member, service and account**. An arbitrary number of margin measures per account are supported, as well as native currency measure breakdown for each of the former.

7.2.2.1 initialMarginReport document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-im:initialMarginReport
   modelVersion="3"
   version="1"
   generationDateTime="2013-11-14T22:29:08.112Z"
   xmlns:cdml="urn:cdml:schema:common:types"
   xmlns:cdml-im="urn:cdml:schema:margin:initialMargin">
    <cdml:reportDate>2013-11-11</cdml:reportDate>
    <cdml:intraday>true</cdml:intraday>
    <cdml-im:initialMarginData>
        ...
    </cdml-im:initialMarginData>
    <cdml-im:initialMarginData>
        ...
    </cdml-im:initialMarginData>
    <cdml-im:initialMarginData>
        ...
    </cdml-im:initialMarginData>
</cdml-im:initialMarginReport>
```

Element/attribute	Description
initialMarginReport	Root element, defines the type of report
initialMarginReport/@modelVersion	CDML specification version (fixed)
initialMarginReport/@version	CDML instance version
initialMarginReport/@generationDateTime	ISO8601 datetime
initialMarginReport/reportDate	ISO8601 date
initialMarginReport/intraday	***NEW*** ITD flag
initialMarginReport/initialMarginData	Each initialMarginData element represents a single margin account/service combination at the CCP, and groups the margin measures for the given combination, at the given date

7.2.2.2 initialMarginData element

```
<cdml-im:initialMarginData>
    <cdml-im:CCP>LCH</cdml-im:CCP>
    <cdml-im:clearingService>IRD</cdml-im:clearingService>
    <cdml-im:memberId>CC1</cdml-im:memberId>
    <cdml-im:initialMarginAccountId>CALYPSOGIG3</cdml-im:initialMarginAccountId>
    <cdml-im:segregationAccount>C</cdml-im:segregationAccount>
    <cdml-im:measures>
        <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="5083145.176745">
            <cdml-im:measureBreakdown nativeCcy="USD" amount="125228.97" conversionFX="0.8"/>
            <cdml-im:measureBreakdown nativeCcy="EUR" amount="345876.23" conversionFX="1.0"/>
            <cdml-im:measureBreakdown nativeCcy="GBP" amount="3434878.3487" conversionFX="1.35"/>
        </cdml-im:measure>
        <cdml-im:measure requirementCcy="EUR" type="INITIAL_MARGIN" amount="4935687.34"/>
    </cdml-im:measures>
    <cdml-im:initialMarginPositionAccountData>
        <cdml-im:initialMarginPositionAccount>
            <cdml-im:positionAccountId>CALYPSOGIG3_1</cdml-im:positionAccountId>
            <cdml-im:measures>
                <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="4848268.853745"/>
            </cdml-im:measures>
        </cdml-im:initialMarginPositionAccount>
        <cdml-im:initialMarginPositionAccount>
```

```
            <cdml-im:positionAccountId>CALYPSOGIG3_2</cdml-im:positionAccountId>
            <cdml-im:measures>
                <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="234876.323"/>
            </cdml-im:measures>
        </cdml-im:initialMarginPositionAccount>
    </cdml-im:initialMarginPositionAccountData>
</cdml-im:initialMarginData>
```

Element/attribute	Description
`initialMarginData`	Represents a single CCP/service/member/margin account combination
`initialMarginData/CCP`	Short name of the CCP
`initialMarginData/clearingService`	Traditionally called *product type*, each service is normally an aggregation of similar cleared products than can be margined together (although it's not limited to this definition)
`initialMarginData/memberId`	FCM/SCM identifier at the CCP
`initialMarginData/initialMarginAccountId`	Identifier of the margin account at the CCP
`initialMarginData/segregationAccount`	Evolution of the segregation type concept, it represents a higher level grouping, compared to the margin account. Depending on the CCP this can result in just House/Client segregation, or in an actual account id.
`initialMarginData/measures`	Margin measures element
`initialMarginData/initialMarginPositionAccountData`	**Optional** margin position account breakdown

7.2.2.3 *measures element*

Measure breakdown is not mandatory, unless native margins are to be imported.

```
<cdml-im:measures>
    <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="5083145.176745">
        <cdml-im:measureBreakdown nativeCcy="USD" amount="125228.97" conversionFX="0.8"/>
        <cdml-im:measureBreakdown nativeCcy="EUR" amount="345876.23" conversionFX="1.0"/>
        <cdml-im:measureBreakdown nativeCcy="GBP" amount="3434878.3487" conversionFX="1.35"/>
    </cdml-im:measure>
    <cdml-im:measure requirementCcy="EUR" type="INITIAL_MARGIN" amount="4935687.34"/>
</cdml-im:measures>
```

Element/attribute	Description
measure	Single margin measure
measure/@requirementCcy	Currency the CCP issues the call in
measure/@type	Measure type
measure/@amount	Amount in the requirement currency
measure/measureBreakdown	Single currency component of the measure breakdown
measure/measureBreakdown/@nativeCcy	Native currency
measure/measureBreakdown/@amount	Amount in the native currency
measure/measureBreakdown/@conversionFX	**Multiplicative rate** to be applied to the native currency to obtain the amount in the requirement currency

7.2.2.4 initialMarginPositionAccountData element

Optional breakdown of margin data into separate position account level components. CCP, clearing service, member id and segregation account are implicit, as this element is **always nested within an `initialMarginData` one**.

```
<cdml-im:initialMarginPositionAccountData>
    <cdml-im:initialMarginPositionAccount>
        <cdml-im:positionAccountId>CALYPSOGIG3_1</cdml-im:positionAccountId>
        <cdml-im:measures>
            <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="4848268.853745"/>
        </cdml-im:measures>
    </cdml-im:initialMarginPositionAccount>
    <cdml-im:initialMarginPositionAccount>
        <cdml-im:positionAccountId>CALYPSOGIG3_2</cdml-im:positionAccountId>
        <cdml-im:measures>
            <cdml-im:measure requirementCcy="EUR" type="MAINTENANCE_REQUIREMENT" amount="234876.323"/>
        </cdml-im:measures>
    </cdml-im:initialMarginPositionAccount>
</cdml-im:initialMarginPositionAccountData>
```

Element/attribute	Description
`initialMarginPositionAccountData`	Single breakdown element allowed within an `intialMarginData` one
`initialMarginPositionAccountData/initialMarginPositionAccount`	A single position account element. Each `initialMarginPositionAccountData` can hold an arbitrary number of position account elements
`initialMarginPositionAccountData/initialMarginPositionAccount/positionAccountId`	Position account id
`initialMarginPositionAccountData/initialMarginPositionAccount/measures`	Measures for this position account. See 7.2.2.3

7.2.3 urn:cdml:schema:position:tradeValuation

URI	urn:cdml:schema:position:tradeValuation
Sample prefix	cdml-tv
Description	`tradeValuationReport` related datatypes and root element definition

The `tradeValuationReport` is a trade activity, flow and valuation report, in which the fundamental element information is the **cleared trade**.

7.2.3.1 tradeValuationReport document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-tv:tradeValuationReport
    modelVersion="1"
    version="17"
    generationDateTime="2013-11-13T23:07:21.872Z"
    xmlns:cdml="urn:cdml:schema:common:types"
    xmlns:cdml-tv="urn:cdml:schema:position:tradeValuation">
    <cdml:reportDate>2013-10-14</cdml:reportDate>
    <cdml-tv:trade>
        ...
    </cdml-tv:trade>
    <cdml-tv:trade>
        ...
    </cdml-tv:trade>
    <cdml-tv:trade>
        ...
    </cdml-tv:trade>
    ...
    <cdml-tv:trade>
        ...
    </cdml-tv:trade>
</cdml-tv:tradeValuationReport>
```

Element/attribute	Description
`tradeValuationReport`	Root element, defines the type of report
`tradeValuationReport/@modelVersion`	CDML specification version (fixed)
`tradeValuationReport/@version`	CDML instance version
`tradeValuationReport/@generationDateTime`	ISO8601 datetime
`tradeValuationReport/reportDate`	ISO8601 date
`tradeValuationReport/intraday`	***NEW*** ITD flag[3]
`tradeValuationReport/trade`	Each trade element represents a cleared trade reported by the CCP in the `reportDate`

7.2.3.2 trade element

```
<cdml-tv:trade>
    <cdml-tv:clearedTradeId>8984078249</cdml-tv:clearedTradeId>
    <cdml-tv:CCP>LCHFXCLEAR</cdml-tv:CCP>
    <cdml-tv:clearingService>NDF</cdml-tv:clearingService>
    <cdml-tv:memberId>CC3</cdml-tv:memberId>
    <cdml-tv:positionAccountId>CALYPSOGIG5</cdml-tv:positionAccountId>
    <cdml-tv:segregationAccount>CLIENT</cdml-tv:segregationType>
    <cdml-tv:tradeCashFlowData>
        <cdml-tv:flow settleCcy="USD" type="NPV_ADJUSTED" amount="-220273.13"/>
        <cdml-tv:flow settleCcy="USD" type="NPV_REV" amount="-549622.43"/>
        <cdml-tv:flow settleCcy="USD" type="PAI" amount="-576560.05"/>
        <cdml-tv:flow settleCcy="USD" type="CASH_DELIVERY" amount="0"/>
    </cdml-tv:tradeCashFlowData>
    <cdml-tv:tradeValuationData>
        <cdml-tv:valuation settleCcy="USD" type="NPV_ADJUSTED" amount="-220273.13"/>
        <cdml-tv:valuation settleCcy="USD" type="GAMMA" amount="24669.02"/>
        <cdml-tv:valuation settleCcy="USD" type="THETA" amount="0.207"/>
        <cdml-tv:valuation settleCcy="USD" type="RHO" amount="202005.74"/>
        <cdml-tv:valuation settleCcy="USD" type="PRICE" amount="29.83008655"/>
    </cdml-tv:tradeValuationData>
</cdml-tv:trade>
```

3 *Currently not used in TV processing*

Element/attribute	Description
trade	Single traded cleared at the CCP
trade/CCP	Short name of the CCP
trade/clearingService	See 7.2.2.2
trade/memberId	FCM/SCM identifier at the CCP
trade/positionAccountId	Identifier of the position account at the CCP
trade/segregationAccount	See 7.2.2.2
trade/tradeCashFlowData	Group of cash flow elements produced by the trade at the given reportDate
trade/tradeValuationData	Other trade valuation data

7.2.3.3 tradeCashFlowData element

```
<cdml-tv:tradeCashFlowData>
    <cdml-tv:flow settleCcy="USD" type="NPV_ADJUSTED" amount="-220273.13"/>
    <cdml-tv:flow settleCcy="USD" type="NPV_REV" amount="-549622.43"/>
    <cdml-tv:flow settleCcy="USD" type="PAI" amount="-576560.05"/>
    <cdml-tv:flow settleCcy="USD" type="CASH_DELIVERY" amount="0"/>
    <cdml-tv:flow settleCcy="USD" type="SPECIAL_FLOW" settleDate="2013-10-14" amount="0"/>
</cdml-tv:tradeCashFlowData>
```

Element/attribute	Description
tradeCashFlowData	Group of cash flow elements
tradeCashFlowData/flow	Single flow type/currency combination
tradeCashFlowData/flow/@settleCcy	Settle currency
tradeCashFlowData/flow/@type	Flow type
tradeCashFlowData/flow/@amount	Settlement amount in the given currency
tradeCashFlowData/flow/@settleDate	**Optional** settle date

7.2.3.4 *tradeValuationData element*

```
<cdml-tv:tradeValuationData>
    <cdml-tv:valuation settleCcy="USD" type="NPV_ADJUSTED" amount="-220273.13"/>
    <cdml-tv:valuation settleCcy="USD" type="GAMMA" amount="24669.02"/>
    <cdml-tv:valuation settleCcy="USD" type="THETA" amount="0.207"/>
    <cdml-tv:valuation settleCcy="USD" type="RHO" amount="202005.74"/>
    <cdml-tv:valuation settleCcy="USD" type="PRICE" amount="29.83008655"/>
</cdml-tv:tradeValuationData>
```

Element/attribute	Description
tradeValuationData	Group of valuation elements
tradeValuationData/valuation	Single valuation type/currency combination
tradeValuationData/valuation/@settleCcy	Settle currency
tradeValuationData/valuation/@type	Flow type
tradeValuationData/valuation/@amount	Settlement amount in the given currency

8 Generating content: CDMLProducer

There are several ways of **generating and storing** CDML content in the system. The only true requisite is to **abide by the latest CDML schemas**. The Clearing distribution ships with two main ways of achieving this: **loading CDML directly from the filesystem, and translating it from CCP sources**.

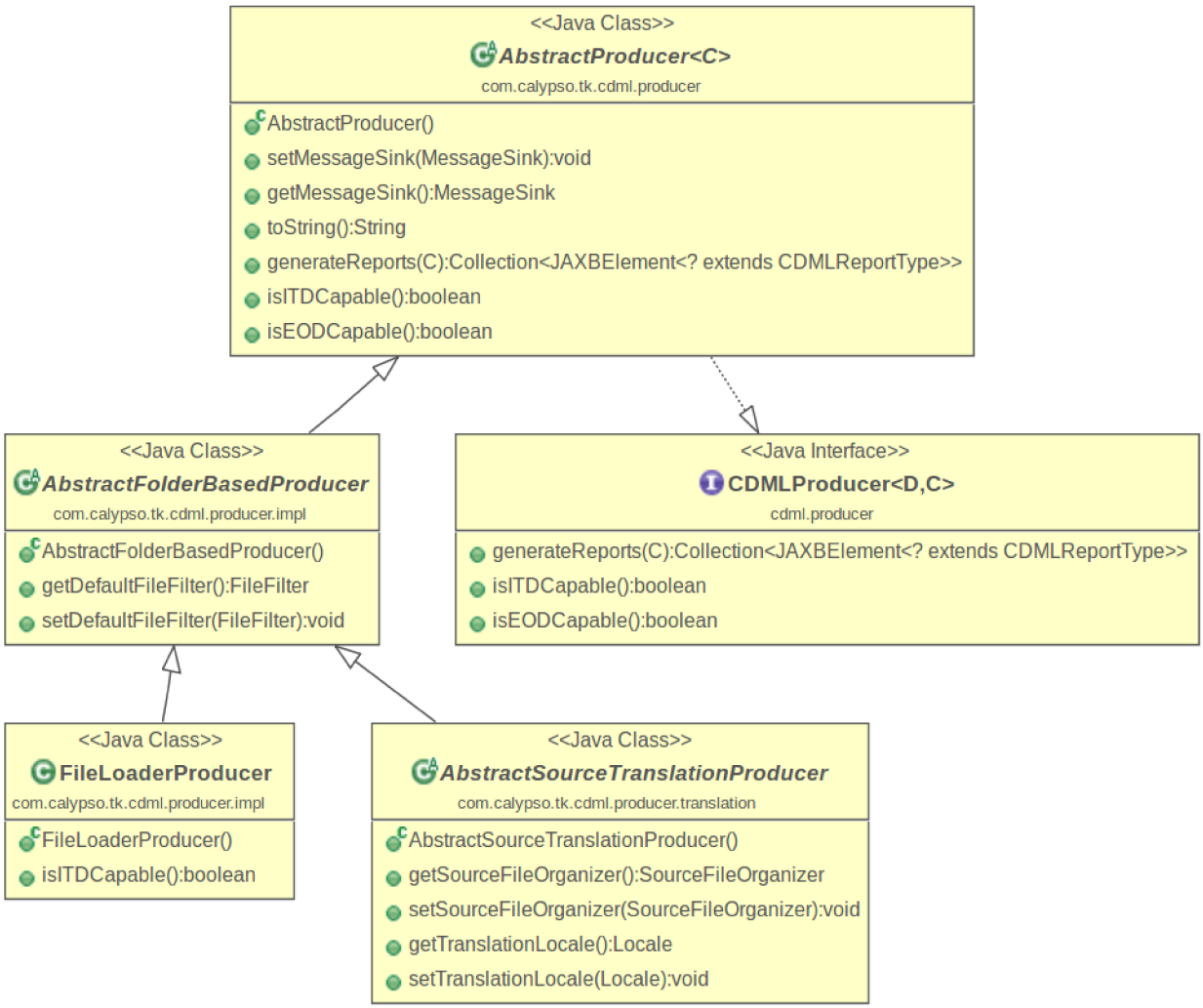


Illustration 3: The `CDMLProducer` hierarchy, and the two main implementations: translation and file loader

8.1 NEW EOD vs ITD

Introduced in 2.11.0/3.6.0, the `CDMLProducer` interface allows the implementation to announce itself as **either ITD-capable, EOD-capable, or both.** For backwards compatibility reasons, *previous implementations are EOD-capable only*. The CDML framework uses this information when discovering producer implementations. See 8.3.1.3 for more details.

8.2 Producer types

8.2.1 FileLoaderProducer

This is the simplest CDML producer implementation: it assumes the content to store is already in CDML format. See 8.3.1.2.

8.2.2 NEW AbstractSourceTranslationProducer

CDML translation is the process by which **CCP-specific content is transformed into the CCP-agnostic representation that is the CDML report**. Even when each translator deals with such CCP-specific content, most of the processing can be done in a CCP-agnostic fashion. `AbstractSourceTranslationProducer` , and its immediate child classes, implement as much as this processing, leaving the final details to the leafs of the class hierarchy.

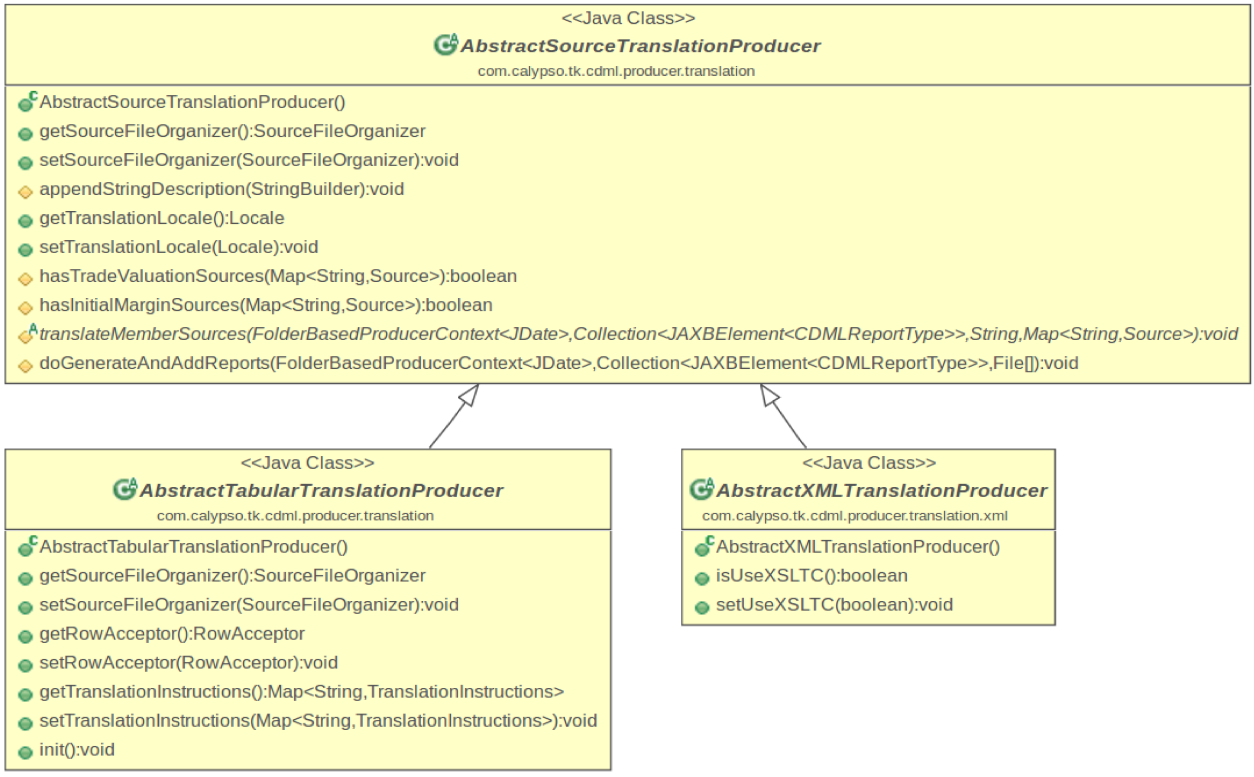


Illustration 4: `AbstractSourceTranslationProducer` class detail, including protected methods, and the two existing implementations: for XML, and table-based inputs

See 9 Writing a CDML translator for more details.

8.2.2.1.1 AbstractTabularTranslationProducer

This producer replaces the deprecated `AbstractTranslationProducer`[4]. It provides the basic functionality to handle row-and-column input files (e.g CSV).

Examples of this producer are the CME and LCH IRD implementations.

8.2.2.1.2 NEW AbstractXMLTranslationProducer

This producer handles the cases in which the input is an arbitrary number of XML files.

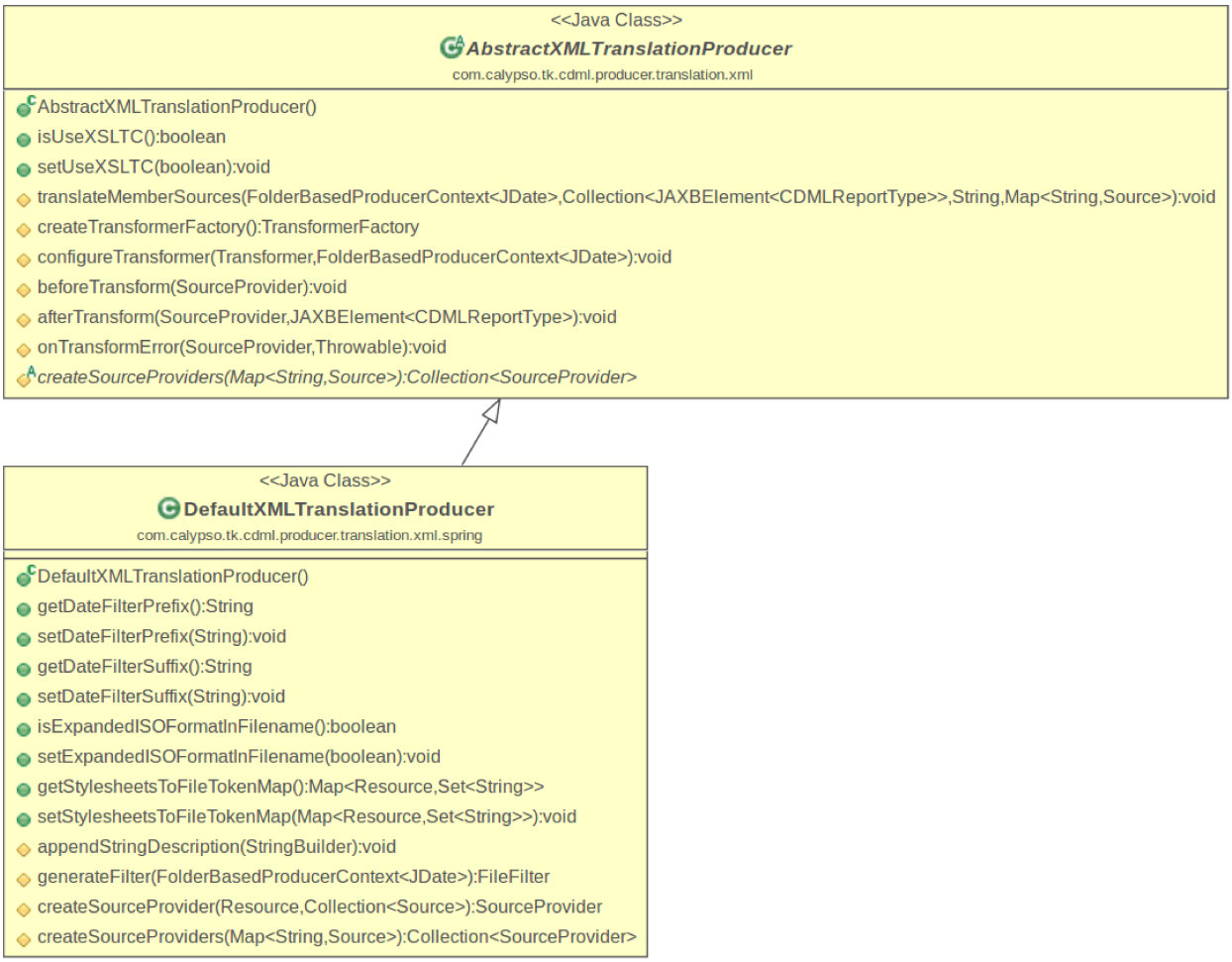


Illustration 5: `AbstractXMLTranslationProducer` and the default implementation

The CDML framework provides a default implementation that only requires information on how to map and group input XML files, to XSL templates. See 9.3 for more details.

4 *To be removed in 2.12.0/3.7.0*

8.3 The CDML_TRANSLATE_TO_CDML ScheduledTask

The CDML_TRANSLATE_TO_CDML `ScheduledTask` is the central driver of CDML loading, translating, and storing. It is a **file based ST**, and it can be used for all CCPs, clearing services, POs and CDML report types **simultaneously:** there is no need to configure multiple instances of it.

The ST valuation date determines the CDML `reportDate` in any downstream operation: from filtering EOD files, to populating the CDML output.

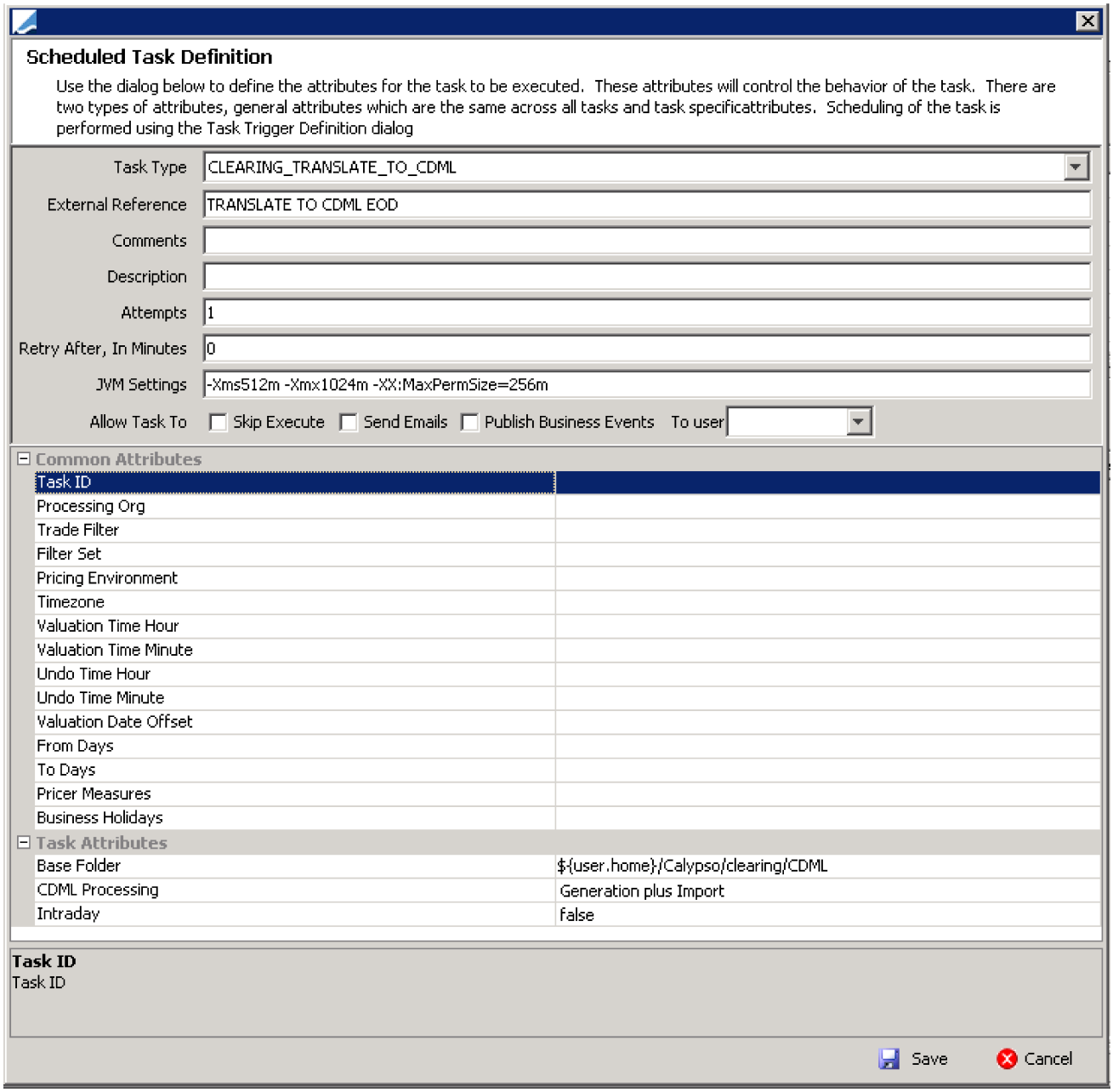


Illustration 6: Sample translate CDML `ScheduledTask`, configured for EOD: for backwards compatibility purposes, all new and existing tasks come with `Intraday=false`. User must configure it to `true` to run in ITD mode

8.3.1 Arguments

8.3.1.1 Base Folder

OS folder that will be used

- to load already generated CDML content
- as base folder for the CCP-specific CDML translators

8.3.1.2 CDML Processing

Processing mode, allows two values

- ***Import Only*** : assume there are CDML files[5] in Base Folder, and load them, using the FileLoaderProducer.
- ***Generation plus Import*** : locate CCP-specific content in the *Base Folder* subfolders, and transform it into CDML. See CCP subfolders and CDMLProducer discovery mechanism for details. Implementations of NEW AbstractSourceTranslationProducer will be used in this mode.

8.3.1.3 NEW Intraday flag

This attribute is used to populate the CDML report `intraday` element, as well as to decide if a given CDML producer should be used at all, given its ability to produce EOD or ITD content.

- When configured as `false` (default setting), only EOD-capable translators will be used.
- When configured as `true`, only ITD-capable translators will be used.

See 8.3.3 CDMLProducer discovery mechanism for more info.

8.3.2 CCP subfolders

Each subfolder in *Base Folder* is assumed to be a **CCP[6] folder**

5 *Files with .cdml or .xml extension*
6 *LE with "CCP" role*

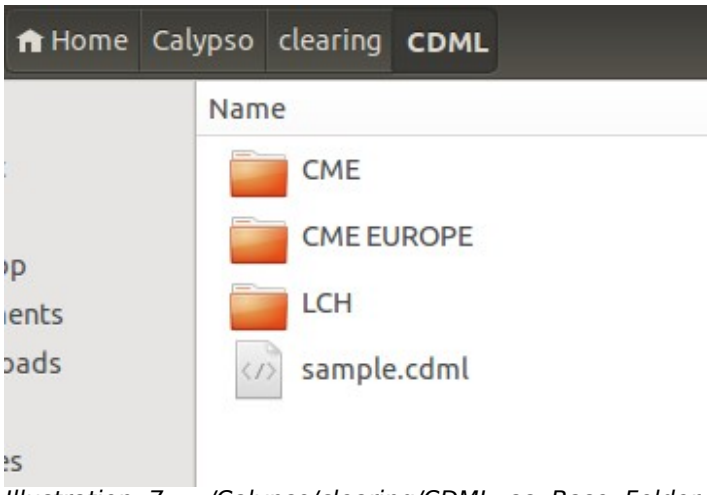


Illustration 7: ~/Calypso/clearing/CDML as Base Folder, with 3 CCP subfolders

In the screenshot above, each of the “CDML” subfolders is **named after an existing CCP LE in the system**. Folders not named after a CCP LE are ignored.

8.3.2.1 NEW Arbitrary CCP subfolders

To support the case of multiple POs clearing on a CCP that doesn't produce files with different names, the task now also reads content from **any first-level (non recursive) CCP subfolder**. Although **the naming of this subfolders is arbitrary**, it is recommended to do so after the firm id at the CCP.

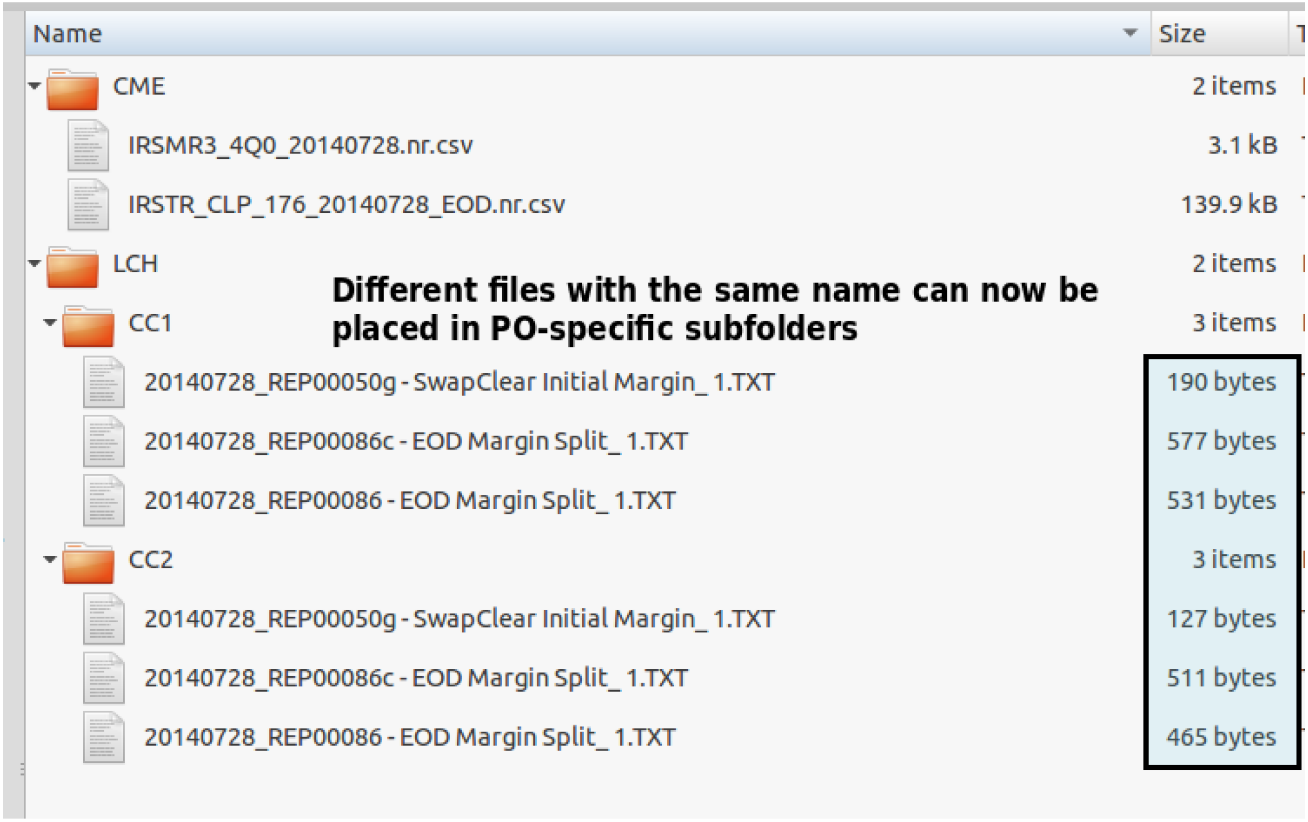


Illustration 8: CCP folder setup for two POs, showing different approaches: for CME, files are named differently, after the PO firm id (4Q0 and 176); LCH requires separate subfolders to avoid name conflicts

8.3.3 CDMLProducer discovery mechanism

This mechanism is still under development, and will probably be changed in future releases.

The OOTB Clearing distribution doesn't ship any *fully configured*[7] NEW AbstractSourceTranslationProducer implementation[8]. Clearing users have to contribute their own translators to be able to produce CDML from CCP content (see Writing a CDML translator). Still, the CDML framework needs a way to locate and run these implementations.

The **`Clearing.CDML.producerNames`** domain is used to register such implementations, using an arbitrary name that identifies the producer.

7 *Although the XML implementation is Java code-complete, it requires CCP-specific XSL templates and configuration*

8 *This could change in the future, to either include licensed implementations, or sample ones. It doesn't change the resolution mechanism, though*

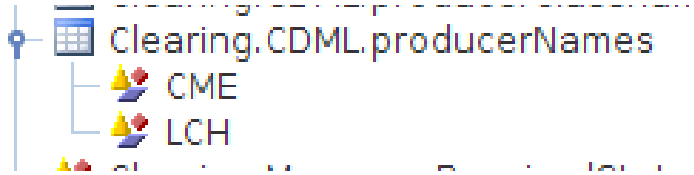


Illustration 9: `Clearing.CDML.producerNames` domain with two producer implementations registered, "CME" and "LCH"

The producer name **does not** need to be the name of a CCP LE, or other object on the system. Its only purpose is **to form the name of an XML classpath resource** that will be used to instantiate the producer itself. The classpath resource name will be formed like

```
<producer name>.CDMLProducer.xml
```

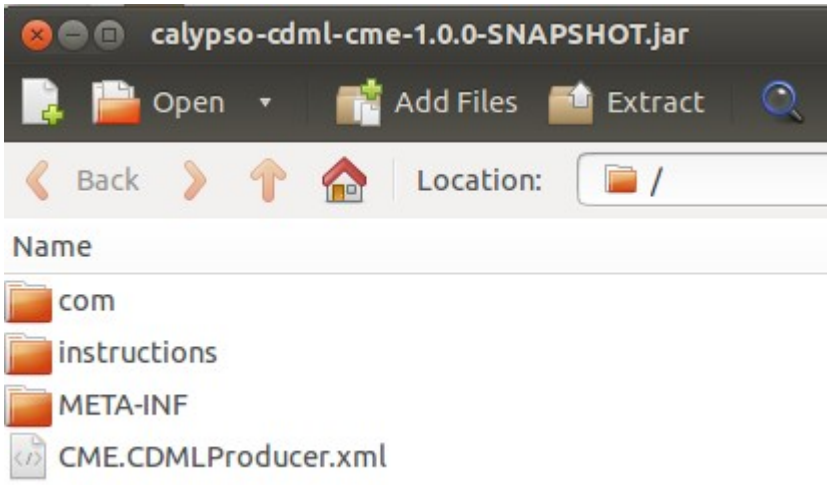


Illustration 10: A sample CME CDML implementation, with the CME.CDMLProducer.xml resource at root level

The XML resource is interpreted as a **Spring application context**, with a single requirement: **it must contain a `CDMLProducer` implementation bean named after the producer**.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:util="http://www.springframework.org/schema/util"
   xsi:schemaLocation="
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.0.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
   default-init-method="init" default-destroy-method="destroy">

   <bean name="CME" class="com.calypso.tk.cdml.producer.translation.CMEProducer">
      <property name="sourceFileOrganizer">
         <bean class="com.calypso.tk.cdml.producer.translation.source.CMESourceFileOrganizer">
            <property name="stripQuotes" value="true" />
            <property name="contentDelimiter" value="," />
         </bean>
      </property>
      <property name="translationInstructions">
         <bean class="com.calypso.tk.cdml.producer.translation.instructions.factory.TranslationInstructionsLoaderBean">
            <property name="resourcePath" value="instructions/CMEProducer.translationInstructions.properties" />
            <property name="instructionIds">
               <set>
                  <value>IRSMR3</value>
                  <value>IRSTR</value>
               </set>
            </property>
         </bean>
      </property>
   </bean>

</beans>
```

Illustration 11: The CME.CDMLProducer.xml Spring file, with a "CME" bean. The CMEProducer implements CDMLProducer

Once a producer is discovered and loaded, the scheduled task will perform a last check on the producer capabilities, matching the `Intraday` task attribute versus the former's EOD/ITD capabilities, as stated in 8.3.1.3.

8.3.4 CDML storage

Once CDML is generated, it is stored for later usage, with a `CDMLBackend` implementation.

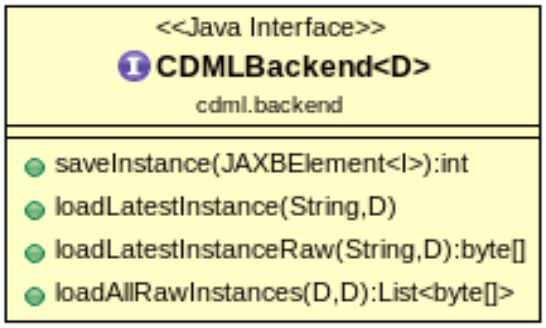


Illustration 12: The `CDMLBackend` interface

The current implementation **stores all CDML in DB**, in the `CDML_INSTANCE` table.

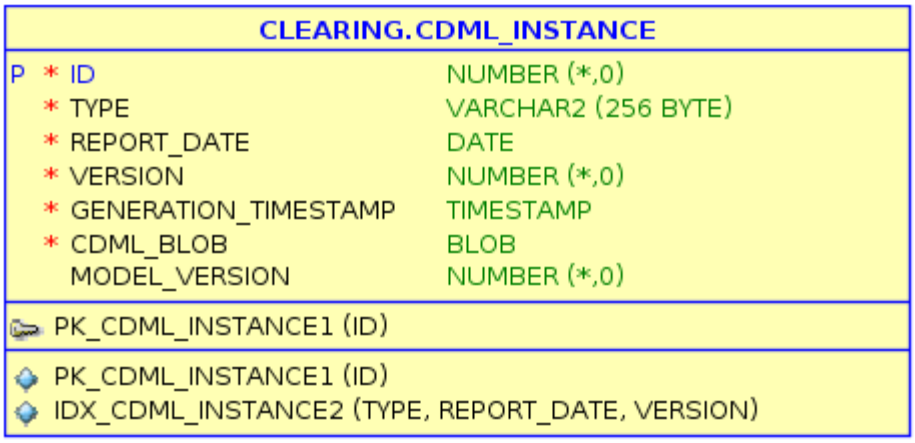


Illustration 13: `CDML_INSTANCE` table

The **report type, date and version combination is unique**: every time a CDML report is saved

- If there is no previous row for the report type and date combination, one in inserted, and **version is set to 1.**
- If there's a previous version, the **version is incremented.**

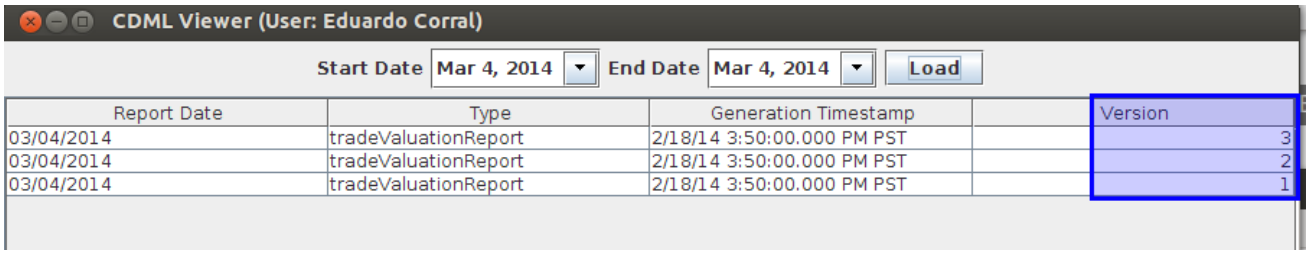


Illustration 14: `CDMLViewer` showing 3 `tradeValuationReport` instances for the same day

When using CDML for downstream processing, **only the latest version of the given report type and date should be considered**. See 10 Processing CDML for more details.

8.3.5 ST execution summary

So, what happens when the ST is run?

- CDML content is **produced**
  - Either is loaded from *Base Folder.*
  - ...or produced by CDML translators from CCP folders and subfolders.
- CDML content is **merged**
  - As seen in CDML storage, **only one report type per date is allowed**.
  - But each `CDMLProducer` implementation (see Generating content:

CDMLProducer) can generate an arbitrary number of CDML report instances.

    - Merging ensures that **only one CDML instance of each type remains** at the end.

- The framework sets the current model version ond generation datetime automatically.
- CDML content is **stored**.

9 Writing a CDML translator

As seen in Generating content: CDMLProducer, the goal of the CDML translator author is **to create an `AbstractSourceTranslationProducer` implementation capable of producing CDML from CCP-specific content**.

This chapter will describe various use cases, covering different strategies.

Chapter 9.2 focuses on **tabular translators**. Both presented cases can be summarized in

- Create the **translation instructions properties** (see 9.2.1.1)
- Creating the **source file organizer** (see 9.1)
- Creating the **translation processors**, one per output type (see 9.2.1.4)
- Create the actual **producer implementation** (see 8.2.2)
- Creating the **Spring resource file** (see 8.3.3 and 9.2.1.6)

This design is still under development. Changes to it can occur to cover new cases, or to simplify existing ones.

The CME use case (9.2.1) contains, not only the CME translation specifics, but also a description of the common classes used by all tabular translators.

It is highly recommended not to skip it, regardless the interest in the CME case.

Chapter 9.3 introduces **XML based translators**. These are simpler, in terms of number classes involved, and can be implemented with as little as

- The Spring resource file
- A set of XSL templates (one template per desired CDML report type)

9.1 NEW Source and SourceFileOrganizer

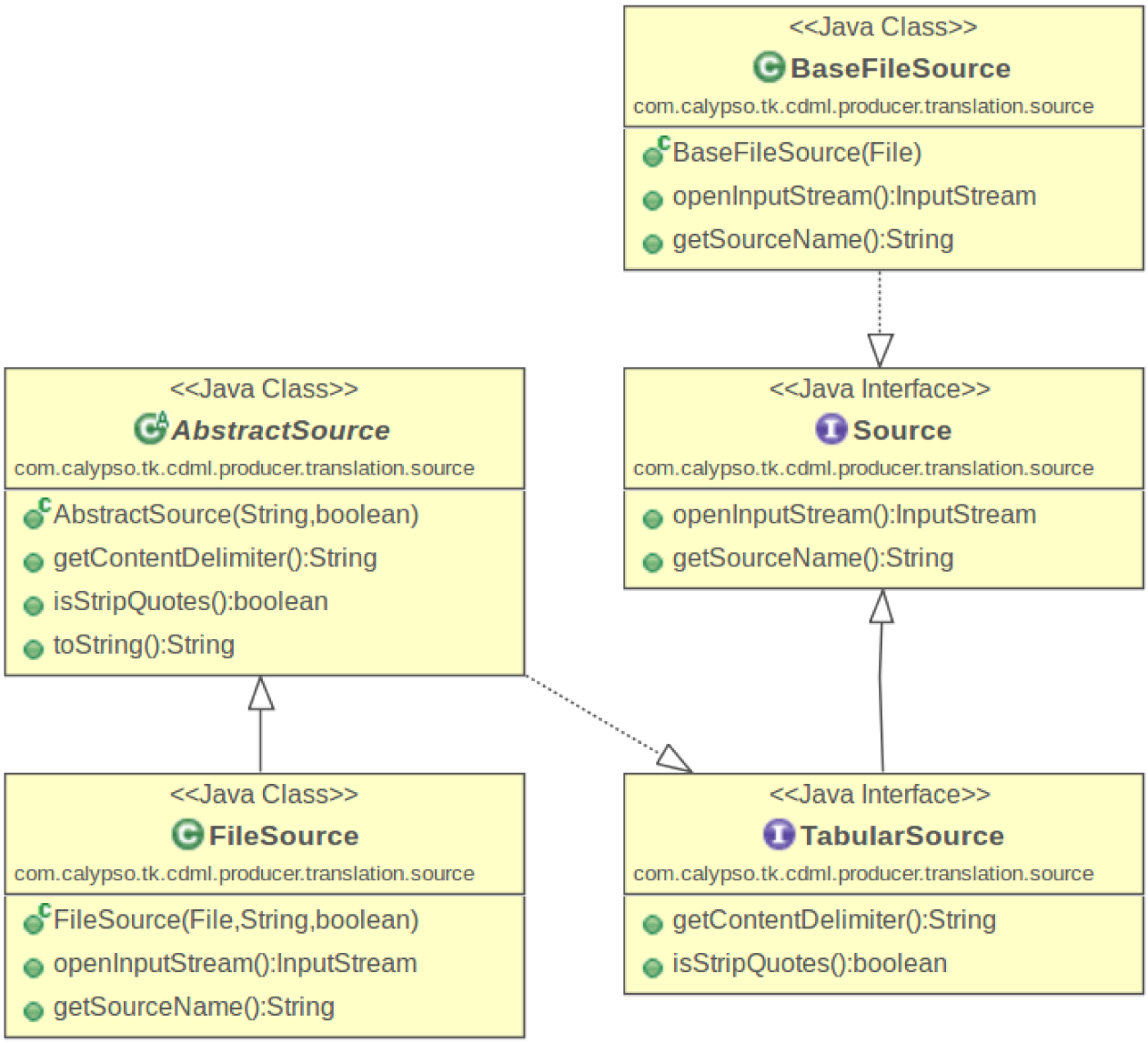


Illustration 15: `Source` hierarchy: `Source` was splitted into `Source` and `TabularSource`, to avoid table-like details in XML-to-XML translation. `FileSource` is still the main implementation for tabular designs, kept for backwards compatibility reasons; `BaseFileSource` is a file-based reimplementation, with no table details.

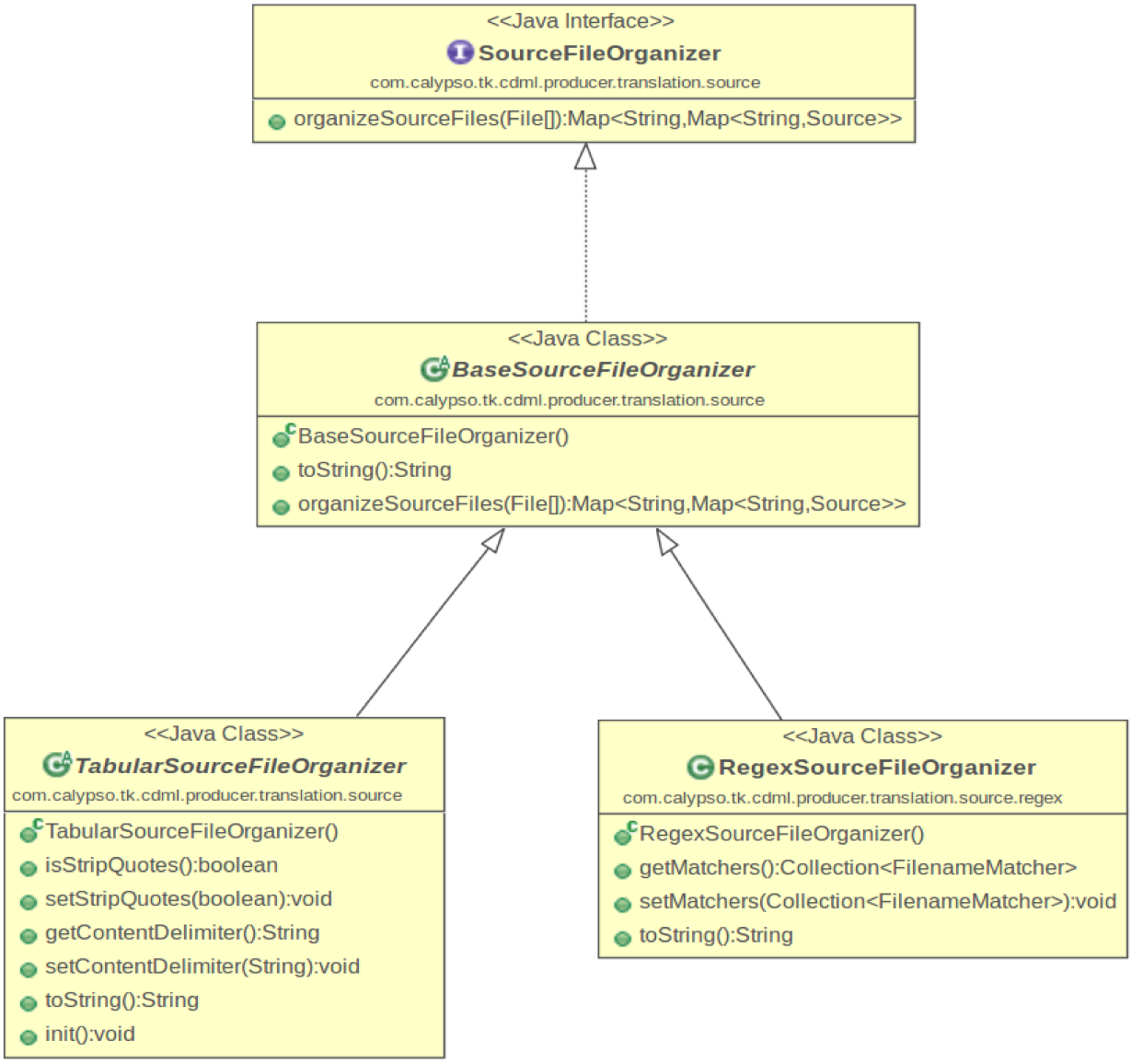


Illustration 16: `SourceFileOrganizer` hierarchy. `AbstractSourceFileOrganizer` has been deprecated, and split in two (`Base` and `Tabular`). `RegexSourceFileOrganizer` is not XML specific, although the latter is the only implementation using it at this moment

A `Source` is a lightweight abstraction of the actual contents of a single CCP report. Main implementations today, `FileSource` and `BaseFileSource`, are file-based, meaning that they provide access to a filesystem resource[9].

A `SourceFileOrganizer` classifies these sources by **purpose** (trade valuation, margin) and **clearing member**. As seen in Illustration 4, all `AbstractTranslationProducer` implementations are configured with a `SourceFileOrganizer` implementation.

9 *There is a `ClasspathSource` implementation used in unit testing*

This classification (organization) of input sources into groups by purpose and member is shared by XML-to-XML and table-based translators.

9.2 Tabular translators

9.2.1 Use case: CME IRD translator

With the CME IRD translator, the goal is to produce the `initialMarginReport` out of the IRSMR3 EOD file, and the `tradeValuationReport` out of the IRSTR EOD file.

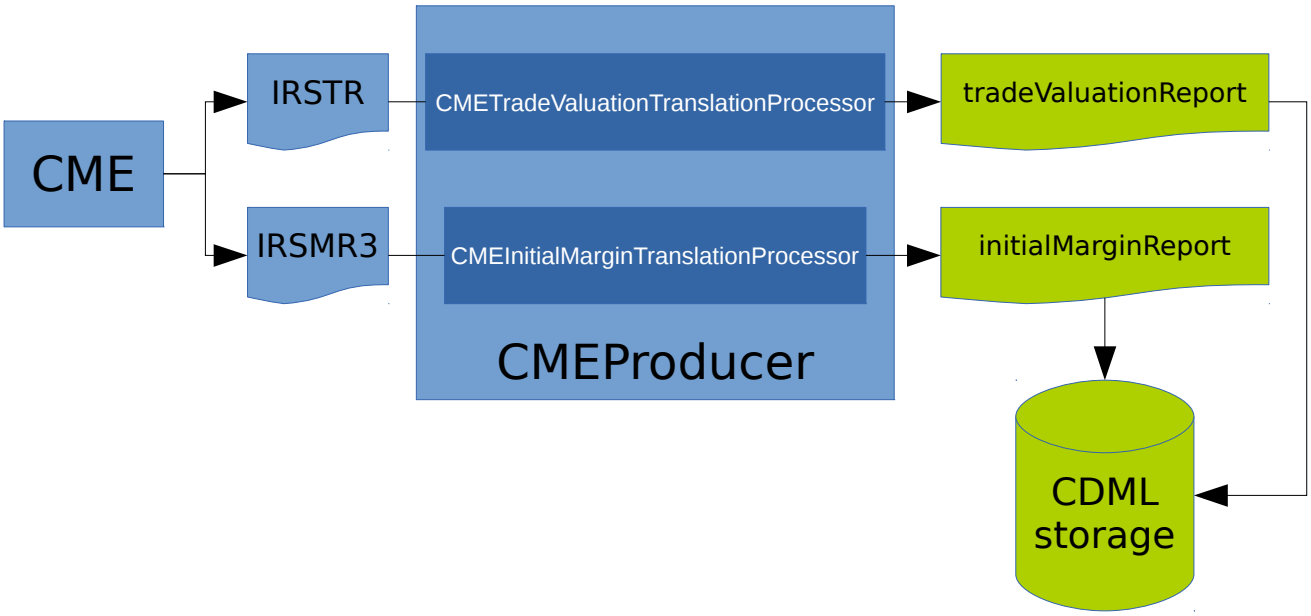


Illustration 17: CME translation flow summary: each of CME's EOD IRD files results in a different CDML report type

9.2.1.1 Translation instructions

The translation instructions are groups of properties, organized by id, in a properties file, that govern

- **how many different file sources** we're going to handle, and **how we'll identify them**, beyond the plain file name.
- **which input report columns we'll parse.**
- **which column we'll use for indexing**, if applicable.

The id is used, among other things, to reference them during the actual translation. Three type of keys are recognized

Key	Optional? (default value)	Description
`indexColumnName`	No	Will used as index when grouping rows. See 9.2.1.3.1 SourceParser.
`indexColumnOptional`	Yes (false)	Instructs the framework not to fail in case the column in `indexColumnName` is not found in the input report. See 9.2.1.3.1 SourceParser.
`columns`	No	Comma-separated list of input columns to pull from the input report. The columns not listed **will be ignored**.[10]

The actual property in the file looks like

```
<instructions id>.<key>=<property value>
```

For instance

```
IRSTR.indexColumnName=Cleared Trade ID
IRSTR.indexColumnOptional=true
IRSTR.columns=Value Date,Cleared Trade ID,Firm ID,Position Account ID,ORIGIN,Currency,Cleared Date,NPV Adjusted,Previ
IRSMR3.indexColumnName=A/C ID
IRSMR3.columns=Firm Id,A/C ID,Seg Type,Currency,Breakout Currency,Breakout Margin,FX Rate,Settle Initial Margin,Settle
```

Illustration 18: CME translation instructions (content truncated)

As seen in Illustration 18, the CME translation instructions declare two ids

- *IRSTR* : in this case the index column is `Cleared Trade ID`, although it's optional, because the IRSTR parsing doesn't need to group rows. See StreamingSourceParser for more info.
- *IRSMR3* : the *A/C ID* column is mandatory, and will be used for row grouping.

10 *In some cases, incoming CCP files can have more than a hundred columns, most of them of little use. Although there is no apparent upper limit to the number of columns to pull from them, it is recommended to include only those which will be actually used during translation.*

9.2.1.2 Source and source file organizer

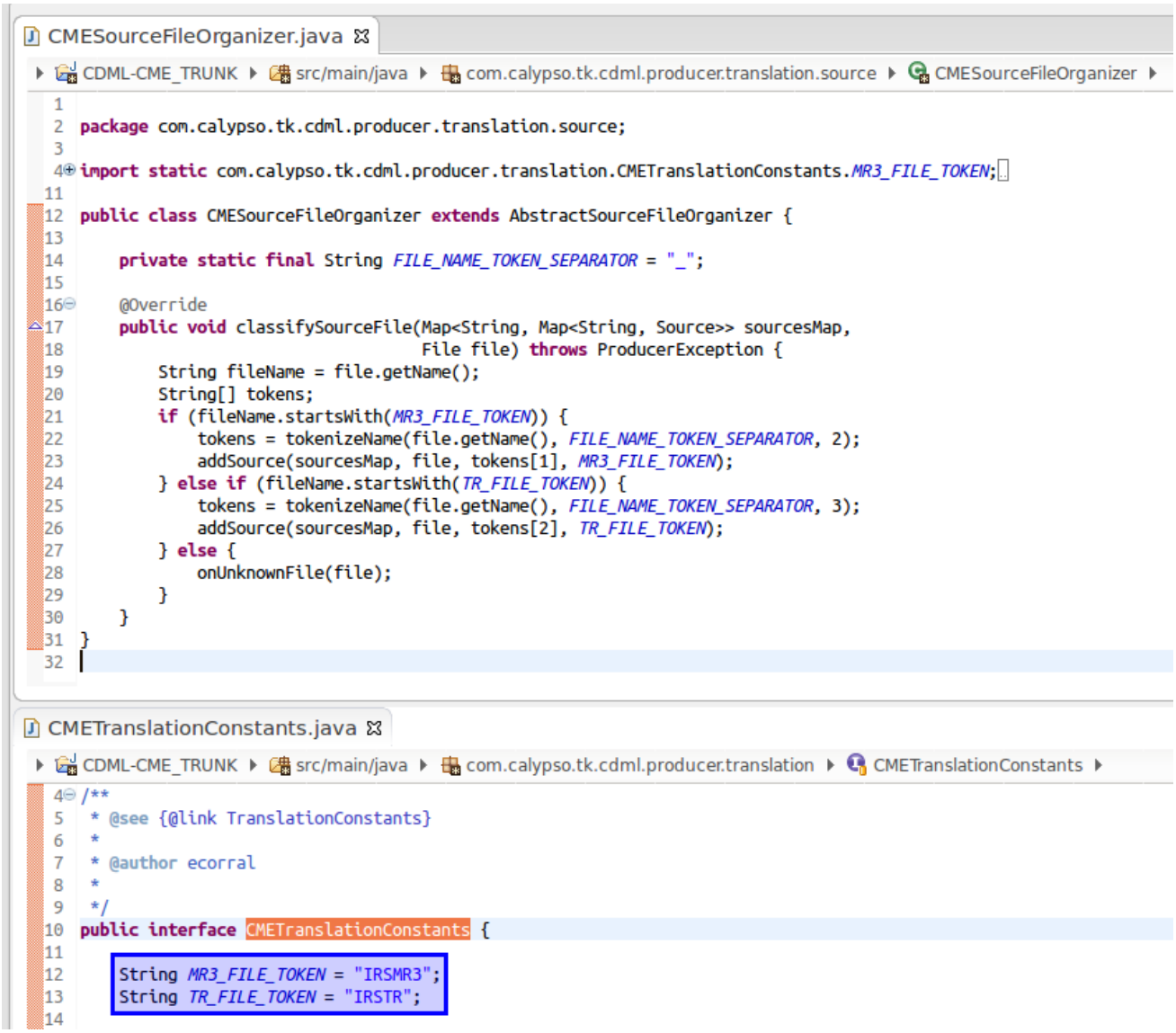


Illustration 19: The `CMESourceFileOrganizer`, showing the file token constant detail in `CMETranslationConstants`

Illustration 19 shows the CME implementation

- The raw file names are tokenized, using a separator (`FILE_NAME_TOKEN_SEPARATOR`).
- CME IRD EOD report file names include member and date information (e.g. *IRSMR3_4Q0_20130118.nr.csv*).
- The file type or purpose (TR or margin) and member mnemonic present in the file name are used to classify the file, invoking `AbstractSourceFileOrganizer.addSource()`.
- Other files are ignored. The default behavior is to log an exception, and keep

processing. Ideally, only files that can contribute to the final content should reach this stage: see 9.2.1.5.3 Providing parser implementations for more info.

9.2.1.3 Parser implementations

Before jumping into the translation process, it is important to understand how the parsing of the CCP EOD files works, and the current `Parser` implementations.

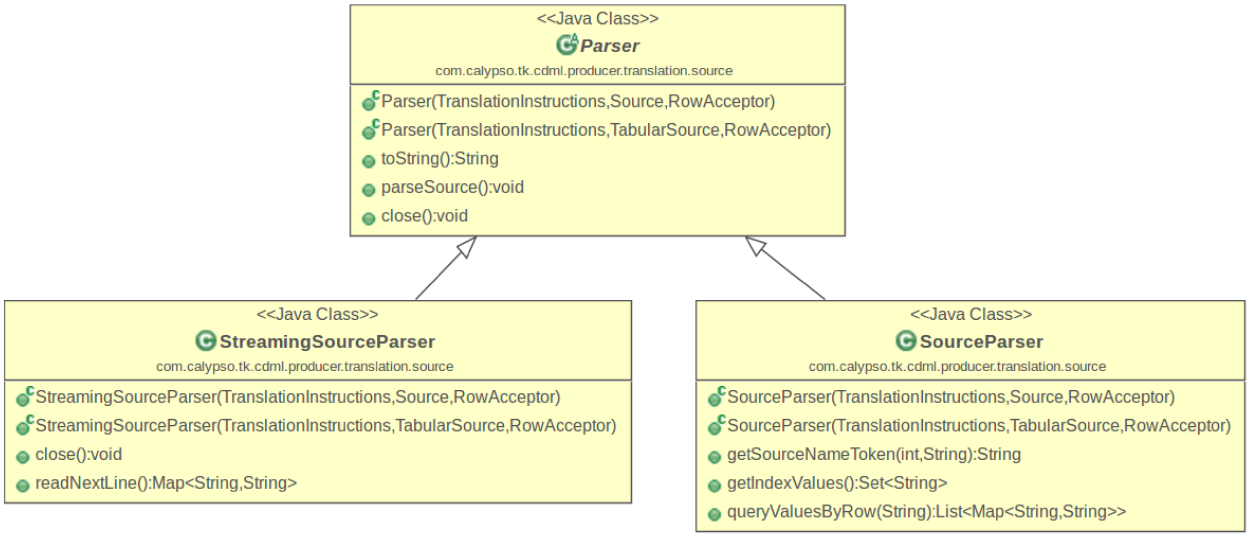


Illustration 20: `Parser` implementations

Parsers are **stateful** objects that need to be configured with the **source** to parse, a **row acceptor** (see 9.2.1.5.2 Row acceptor), and the **translation instructions** for the given source. As such, **they cannot be reused for multiple sources**, even if those are of the same kind.

The main `Parser` method is `parseSource()`: this method must be called before the source can be used, and causes it to be fully scanned, or, at least, prepared for it, depending on the implementation.

See 9.2.1.5.3 Providing parser implementations for more info.

9.2.1.3.1 SourceParser

This is the **default implementation**. Upon `parseSource()` invocation, the whole input source is read, and its contents translated into rows and columns. This implementation is suitable for **small to mid-size files, and/or files that need to group rows by index column** (see 9.2.1.1 Translation instructions).

A	B	C	D	E	F	G	H	I	J
Date	Firm Id	A/C ID	Seg Type	Currency	Maintenance Requirement	Initial Requirement	Breakout Currency	Settle Initial Margin	Settle Maintenance Margin
1/18/2013	123	A--A	CUST	USD	110473.21	121520.54	USD	0	0
1/18/2013	123	A--A	CUST	USD	110473.21	121520.54	EUR	364.28	331.16
1/18/2013	123	A--A	CUST	USD	110473.21	121520.54	AUD	0	0
1/18/2013	123	A--A	CUST	USD	110473.21	121520.54	GBP	121156.26	110142.05
1/18/2013	123	BBBB	CUST	USD	124513.44	136964.79	EUR	90769.09	82517.36
1/18/2013	123	BBBB	CUST	USD	124513.44	136964.79	USD	38059.61	34599.65
1/18/2013	123	BBBB	CUST	USD	124513.44	136964.79	GBP	8136.08	7396.44
1/18/2013	123	CCC	HOUS	USD	169569.85	169569.85	AUD	611.82	611.82
1/18/2013	123	CCC	HOUS	USD	169569.85	169569.85	GBP	39130.96	39130.96
1/18/2013	123	CCC	HOUS	USD	169569.85	169569.85	JPY	105.41	105.41
1/18/2013	123	CCC	HOUS	USD	169569.85	169569.85	EUR	72163.37	72163.37
1/18/2013	123	CCC	HOUS	USD	169569.85	169569.85	USD	57558.3	57558.3

*Illustration 21: Edited CME IRSMR3 file (reordered columns), with three account ids (AAAA, BBBB, CCC). CME provides margin breakout per currency, so it makes sense to group rows by **A/C ID**, as seen in CME's IRSMR3 translation instructions.*

Once parsed, information can be retrieved using the following methods

Method name	Input	Output	Description
`getIndexValues`	N/A	Set of index values	Get all different index column values. In the example from Illustration 21, that would be `[AAAA, BBBB, CCC]`
`queryValuesByRow`	Index value	List of rows that contain the index value	Retrieve all rows that contain the passed index value, as a map of column name/raw column value. **Further cell data type conversion is deferred**: processors need to interpret (read: further parse/convert) the data, if applicable.

9.2.1.3.2 StreamingSourceParser

This is an on-demand parser, which defers the actual reading of the input source until new rows are needed, thus saving memory. This implementation is recommended for **large input files, and/or files that don't require row grouping**.

A	B	C	D	E	F	G	H	I	J	K	L	M
Value Date	Position Account ID	Cleared Trade ID	Platform ID	Client ID	CME Swap Indicator	Currency	Effective Date	Maturity Date	Cleared Date	Status	Notional	Direction
1/25/2013	AAAA	734282	9150180	9150180-2	USD3L-20130221-20140221-1.25	USD	1/12/2013	1/12/2014	1/18/2013	CLEARED	3250000	R
1/25/2013	AAAA	753384	9279586	9279586-2	ZCSUSD-20130312-20140312-3.45	USD	1/12/2013	1/12/2014	1/8/2013	CLEARED	2450000	R
1/25/2013	AAAA	730362	9118956	9118956-2	BSPAUD-20130215-20180215	AUD	1/15/2013	1/15/2018	1/14/2013	CLEARED	2450000	R
1/25/2013	AAAA	730362	9118956	9118956-2	BSPAUD-20130215-20180215	AUD	1/15/2013	1/15/2018	1/14/2013	CLEARED	2450000	R

*Illustration 22: Truncated CME IRSTR file. The information unit is the cleared trade, and each row represents a different one, so it doesn't make sense to do any grouping: in terms of trade information, **rows are unrelated to each other**.*

Main methods are

Method name	Input	Output	Description
`readNextLine`	N/A	Next row from the source file, or `null`, of there's no more content to read	Obtain the next row from the input file, as a map of column name/raw column value. **Further cell data type conversion is deferred**: processors need to interpret (read: further parse/convert) the data, if applicable.

9.2.1.4 Translation processors

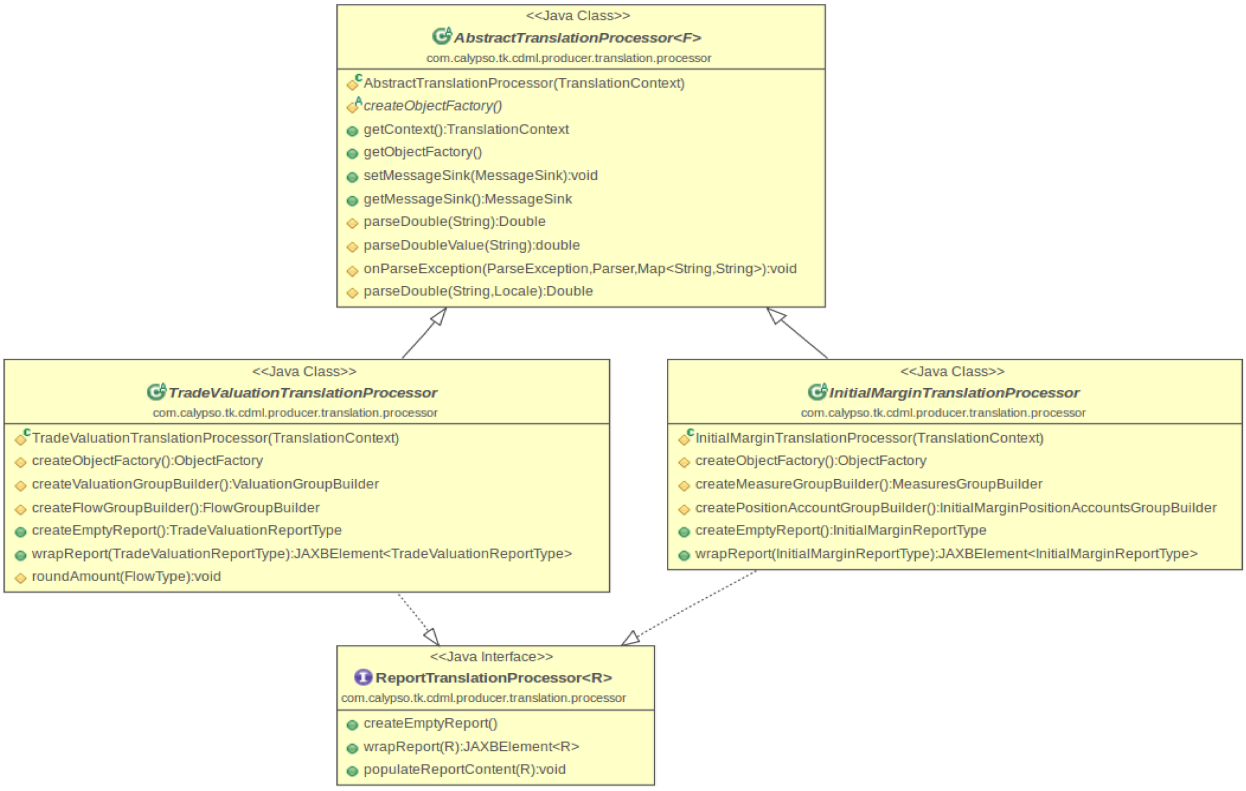


Illustration 23: Trade valuation and initial margin `ReportTranslationProcessor` hierarchies

CDML producers normally delegate the heavy lifting of the translation process to the `ReportTranslationProcessor` implementations. As seen in Illustration 23, two partial implementations are included in the clearing distribution. CDML translator writers need to provide the final piece, `populateReportContent()`, which performs the actual translation. The process can be summarized as

- Obtaining the appropriate parser or parsers for the type of report to be generated (initial margin or trade valuation).
- Iterate over index values, or rows, and produce an `<initialMarginData>` or `<trade>` element, depending on the translator, for each of them.
- Populate the new element with information in the row/s provided by the parser.

9.2.1.4.1 CMEInitialMarginTranslationProcessor

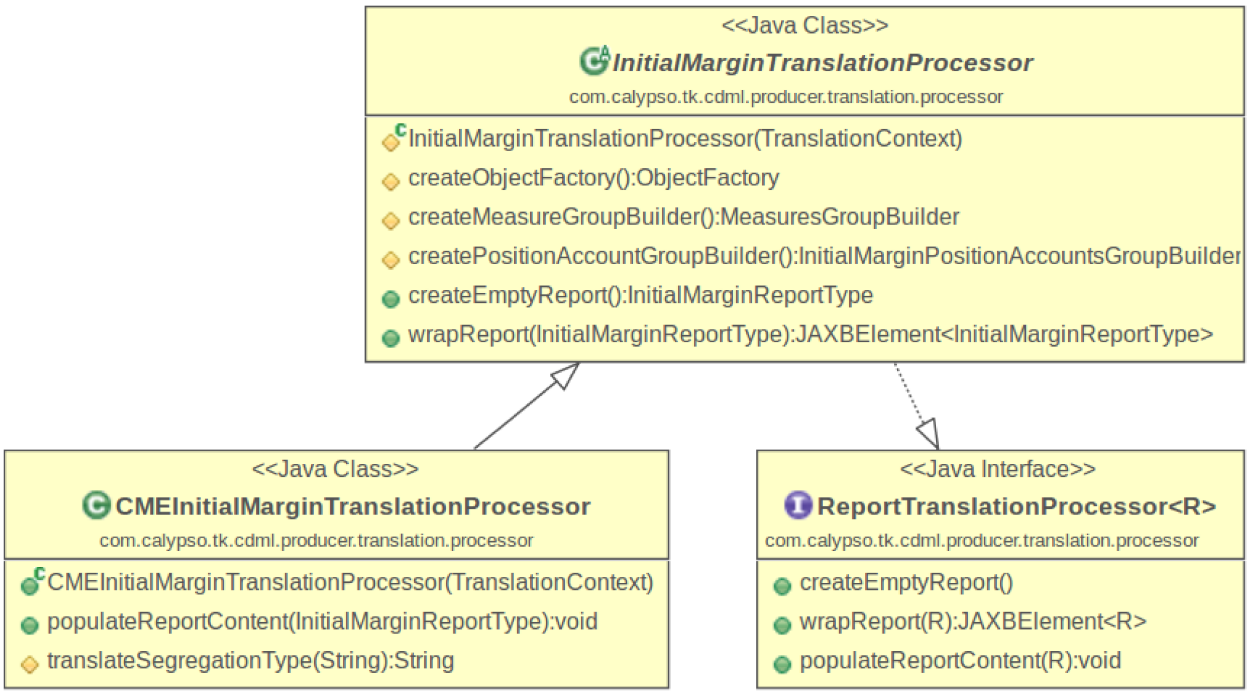
The CME IM translation processor uses **the default** `SourceParser` to retrieve all rows per account id, as seen in Illustration 21.

```
@Override
public void populateReportContent(InitialMarginReportType imReportType) throws ProducerException {
    TranslationContext context = getContext();
    SourceParser parsedData = (SourceParser) context.getParser(MR3_FILE_TOKEN);
    Set<String> indexValues = parsedData.getIndexValues();
    // Index values is the account
    for(String indexValue : indexValues) {
        List<Map<String,String>> rows = parsedData.queryValuesByRow(indexValue);
        if (!rows.isEmpty()) {
```

Illustration 25: Main CME IM translator loop, showing how the parser is extracted from the context, and the iteration over the index values

Every index value represents a margin account, so a new `<initialMarginData>` element is created

```
InitialMarginDataType imData = getObjectFactory().createInitialMarginDataType();
imReportType.getInitialMarginDataElements().add(imData);
imData.setCCP(context.getCCPCode());
imData.setClearingService(IRD_CLEARING_SERVICE);
imData.setMemberId(sampleRow.get(MR3_FIRM_ID_COLUMN));
imData.setInitialMarginAccountId(indexValue);
imData.setSegregationAccount(translateSegregationType(sampleRow.get(SEG_TYPE_COLUMN)));
```



```
String requirementCcy = sampleRow.get(REQUIREMENT_CURRENCY_COLUMN);
MeasuresGroupBuilder measuresGroupBuilder = createMeasureGroupBuilder();
imData.setMeasuresGroup(measuresGroupBuilder.getElement());

MeasureTypeBuilder maintenanceRequirementBuilder =
        new MeasureTypeBuilder(getObjectFactory(),
                               MAINTENANCE_REQUIREMENT_MEASURE,
                               0.0d,
                               requirementCcy);
measuresGroupBuilder.addMeasureFromBuilder(maintenanceRequirementBuilder);
```

Illustration 27: The group builder is attached to the current IM element, and the single measure builder to the group one. ***Note the 0.0 value****: the actual value will be aggregated after*

In the case of CME, currency breakdown is provided, and so such detail is added to the CDML output

```
double totalMaintenanceMargin = 0.0d;
double totalInitialMargin = 0.0d;

for (Map<String,String> breakdownRow : rows) {
    String nativeCcy = breakdownRow.get(BREAKOUT_CURRENCY_COLUMN);
    String rawFXRate = breakdownRow.get(FX_RATE_COLUMN);
    double fxRate = parseDouble(rawFXRate);
    if (Util.isZero(fxRate, Util.EPSILON)) {
        throw new ProducerException(String.format(MISSING_FX_RATE_ERROR_MSG,
                                                  indexValue,
                                                  nativeCcy,
                                                  FX_RATE_COLUMN));
    }

    // Settle margins are reported in requirement ccy, rate is always
    // multiplicative to go from native to requirement ccy, which is what
    // CDML records

    double maintMargin = parseDouble(breakdownRow.get(MAINTENANCE_MARGIN_COLUMN));
    totalMaintenanceMargin += maintMargin;

    maintenanceRequirementBuilder.addBreakdown(nativeCcy,
                                               parseDouble(breakdownRow.get(BREAKOUT_MARGIN_COLUMN)),
                                               fxRate);

    double initialMargin = parseDouble(breakdownRow.get(INITIAL_MARGIN_COLUMN));
    totalInitialMargin += initialMargin;
}

maintenanceRequirementBuilder.getElement().setAmount(totalMaintenanceMargin);
initialMarginBuilder.getElement().setAmount(totalInitialMargin);
```

Illustration 28: Currency breakdown loop, within a single IM element. The list of rows was provided by the parser. Note how the aggregated amounts are updated after the currency iteration

9.2.1.4.2 CMETradeValuationTranslationProcessor

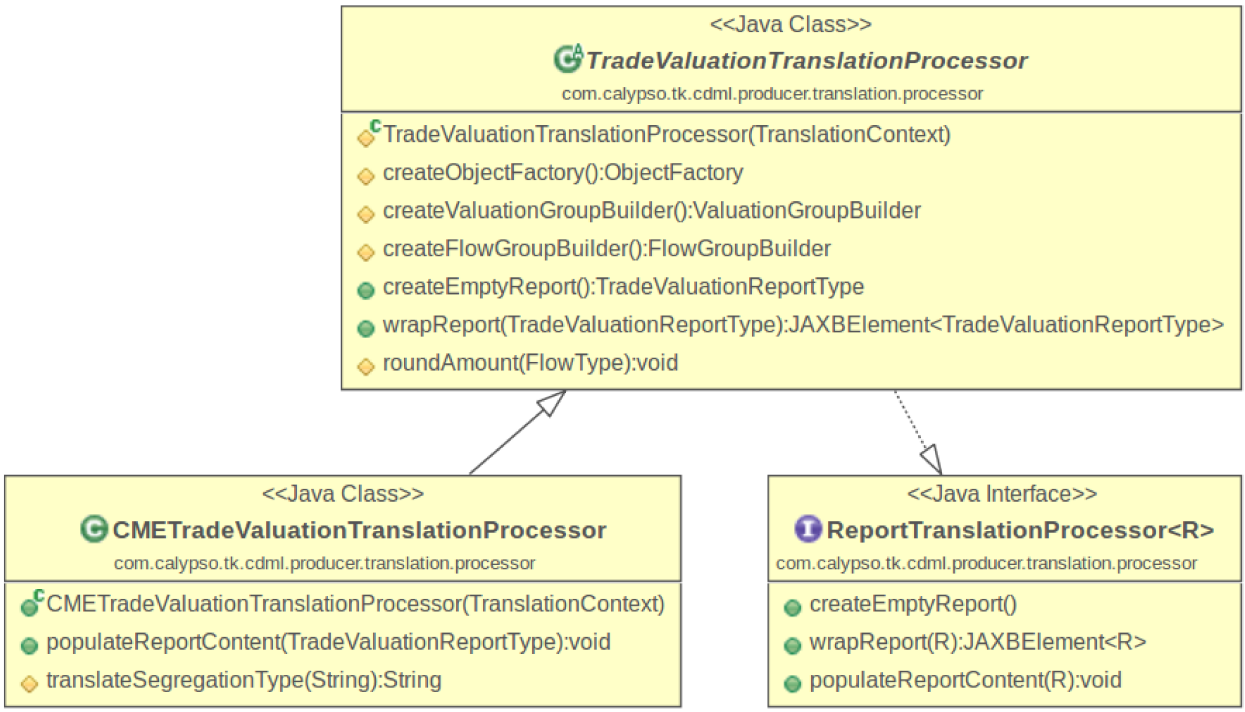


Illustration 29: `CMETradeValuationTranslationProcessor` hierarchy detail

CME TV processor uses the `StreamingSourceParser` instead of the default one: there's no benefit in grouping rows, as the output information unit (the `<trade>` element) corresponds to the input one (the cleared trade row). Also, the TR is a report that can grow substantially, and it can be dangerous, in terms of resources, to keep it all in memory.

As with the IM one, the first step is to obtain the parser, and then iterate over the input. In this case, though, the exit condition (no more rows present) has to be checked

```
@Override
public void populateReportContent(TradeValuationReportType tradeValuationReport) throws ProducerException {
    TranslationContext context = getContext();
    Parser trParser = context.getParser(TR_FILE_TOKEN);
    if (!(trParser instanceof StreamingSourceParser)) {
        throw new UnsupportedOperationException(PARSER_NOT_SUPPORTED_ERROR_MSG);
    }
    StreamingSourceParser parser = (StreamingSourceParser) trParser;
    // Index value is the cleared trade ID
    DateFormat valueDateFormat = new SimpleDateFormat(VALUE_DATE_COLUMN_FORMAT);
    Map<String, String> row = null;
    try {
        while((row = parser.readNextLine()) != null) {
```

Illustration 30: CME TV translator obtaining the streaming parser, and detail on the row parsing loop setup

As stated before, every row represents a <trade> element

```
while((row = parser.readNextLine()) != null) {
    String tradeId = row.get(TRADE_ID_COLUMN);
    String rawValueDate = row.get(VALUE_DATE_COLUMN);
    try {
        JDate valueDate = JDate.valueOf(valueDateFormat.parse(rawValueDate));
        JDate reportDate = toJDate(tradeValuationReport.getReportDate());

        TradeType trade = getObjectFactory().createTradeType();
        tradeValuationReport.getTrades().add(trade);

        trade.setCCP(context.getCCPCode());
        trade.setClearedTradeId(row.get(TRADE_ID_COLUMN));
        trade.setClearingService(IRD_CLEARING_SERVICE);
        trade.setMemberId(row.get(TR_FIRM_ID_COLUMN));
        trade.setPositionAccountId(row.get(POSITION_ACCOUNT_ID_COLUMN));
        trade.setSegregationAccount(translateSegregationType(row.get(ORIGIN_COLUMN)));
```

*Illustration 31: Creating the `<trade>` element from the just read row. **Note the date processing**: the parser provides raw `String` data, the translator adds the last parsing step by interpreting the column as a date*

Flows and valuations can be added manually, or with the helper builders

```
FlowGroupBuilder flowGroupBuilder = createFlowGroupBuilder();
trade.setTradeCashFlowData(flowGroupBuilder.getElement());

String settleCcy = row.get(SETTLE_CURRENCY_COLUMN);
double npvAdjusted = parseDouble(row.get(NPV_ADJUSTED_COLUMN));

flowGroupBuilder.addFlow(NPV_FLOW, npvAdjusted, settleCcy);
if (valueDate.before(reportDate)) {
    flowGroupBuilder.addFlow(NPV_REV_FLOW, npvAdjusted, settleCcy);
} else {
    flowGroupBuilder.addFlow(NPV_REV_FLOW, parseDouble(row.get(PREVIOUS_NPV_ADJUSTED_COLUMN)), settleCcy);
```

Illustration 32: `FlowGroupBuilder` and flow adding detail

NEW *AggregatingFlowGroupBuilder*

Every time the `FlowGroupBuilder.addFlow()` method is invoked, a new `<flow>` element will be added to the output. Sometimes this is not the desired behavior: in some cases, the expected output is to aggregate a column across several rows, into a single output flow. For those cases, `AggregatingFlowGroupBuilder` was created.

`AggregatingFlowGroupBuilder.addFlow()` method checks if there is already an existing flow, and adds up the value, instead of blindly creating a new one.

9.2.1.5 Orchestrating the process: the CDMLProducer implementation

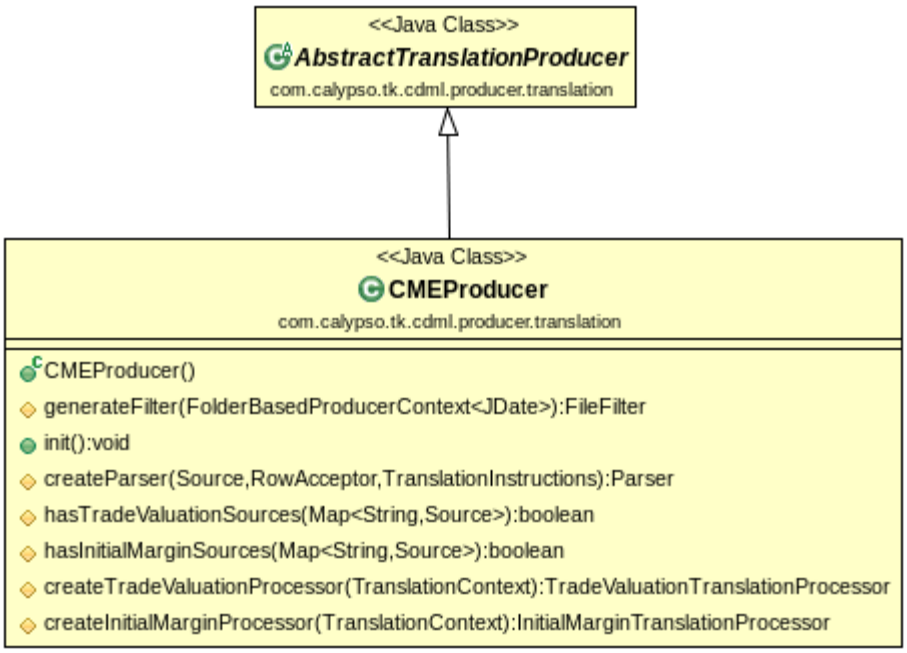


Illustration 33: `CMEProducer` as an `AbstractTranslationProducer` implementation. See Illustration 3 for an extended hierarchy

As the object being instantiated by the Spring context, the `CDMLProducer` implementation **integrates all previously mentioned concepts**, driving the translation process.

Its main responsibilities are the ones **linked to the specifics of a given CCP**, like

- Providing the **source files filtering expression**
- Creating the source parsers, when **other than the default one** must be used
- Verifying the input folder has **enough source files to generate the output** CDML report/s

9.2.1.5.1 Filtering sources

The `generateFilter()` method is one of the mandatory methods to implement, as an `AbstractFolderBasedProducer` child class. The latter will use the resulting `FileFilter` to **ignore any file non related to CDML generation, or to the business date being processed**.

```
// Matcher is IRSTR/IRSMR3 + member mnemonic + optional extra mnemonic for
// TR + date + optional EOD + optional .nr + suffix
private static final String REPORT_MATCHER_PREFIX = "IRS(MR3|TR)_[a-zA-Z0-9]{3}(_[a-zA-Z0-9]{3}){0,1}_";
private static final String REPORT_MATCHER_SUFFIX = "(_EOD){0,1}(\\.nr){0,1}\\.csv";

@Override
protected FileFilter generateFilter(FolderBasedProducerContext<JDate> context) {
    return new FilenameRegexpFileFilter(REPORT_MATCHER_PREFIX
            + toISOString(context.getReportDate(), false)
            + REPORT_MATCHER_SUFFIX);
}
```

Illustration 34: `CMEProducer generateFilter()` implementation. Note how the report (business) date is added to the resulting filter

9.2.1.5.2 Row acceptor

In most producer implementations, the row acceptor will be configured in the Spring context (see 9.2.1.6), if there is need for any at all. If not configured, the default implementation will be used.

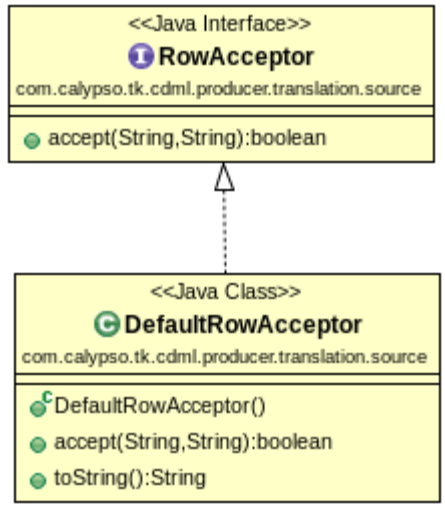


Illustration 35: `DefaultRowAcceptor`

`DefaultRowAcceptor` will accept all rows, **as long as they're not empty**.

9.2.1.5.3 Providing parser implementations

Producer implementations can also override the default parser creation

```
/**
 * For backwards compatibility, default implementation creates a {@link SourceParser}
 * @param source
 * @param rowAcceptor
 * @param instr
 * @return
 */
protected Parser createParser(Source source,
                              RowAcceptor rowAcceptor,
                              TranslationInstructions instr) {
    return new SourceParser(instr, source, getRowAcceptor());
}
```

Illustration 36: Default `createParser()` implementation

in case the streaming parsed is preferred/needed.

```
@Override
protected Parser createParser(Source source,
                              RowAcceptor rowAcceptor,
                              TranslationInstructions instr) {
    return TR_FILE_TOKEN.equals(instr.getId())
                                          ? new StreamingSourceParser(instr,
                                                                      source,
                                                                      rowAcceptor)
                                          : super.createParser(source,
                                                               rowAcceptor,
                                                               instr);
}
```

Illustration 37: `CMEProducer createParser()` override: TR processing requires a `StreamingSourceParser`. See Illustration 30

9.2.1.5.4 Source file presence check

Not an actual requisite, these checks **improve the user's understanding of the missing sources problem**.

```
@Override
protected boolean hasTradeValuationSources(Map<String, Source> memberSourceMap) {
    boolean hasThem = memberSourceMap.containsKey(TR_FILE_TOKEN);
    if (!hasThem) {
        getMessageSink().onMessage(MessageSinkFactory.createWarning(MISSING_TR_ERROR_MSG));
    }
    return hasThem;

}

@Override
protected boolean hasInitialMarginSources(Map<String, Source> memberSourceMap) {
    boolean hasThem = memberSourceMap.containsKey(MR3_FILE_TOKEN);
    if (!hasThem) {
        getMessageSink().onMessage(MessageSinkFactory.createWarning(MISSING_MR3_ERROR_MSG));
    }
    return hasThem;
}
```

Illustration 38: CME `hasTradeValuation` and `hasInitialMarginSources` checks

In case either of the methods returns `false`, the TV or IM CDML generation is skipped.

9.2.1.6 The Spring context

As stated in 8.3.3, **all CDML translator implementations must provide a Spring file named after the producer**, containing a `CDMLProducer` bean, also named after the producer.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:util="http://www.springframework.org/schema/util"
   xsi:schemaLocation="
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.0.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
   default-init-method="init" default-destroy-method="destroy">

   <bean name="CME" class="com.calypso.tk.cdml.producer.translation.CMEProducer">
      <property name="sourceFileOrganizer">
         <bean class="com.calypso.tk.cdml.producer.translation.source.CMESourceFileOrganizer">
            <property name="stripQuotes" value="true" />
            <property name="contentDelimiter" value="," />
         </bean>
      </property>
      <property name="translationInstructions">
         <bean class="com.calypso.tk.cdml.producer.translation.instructions.factory.TranslationInstructionsLoaderBean">
            <property name="resourcePath" value="instructions/CMEProducer.translationInstructions.properties" />
            <property name="instructionIds">
               <set>
                  <value>IRSMR3</value>
                  <value>IRSTR</value>
               </set>
            </property>
         </bean>
      </property>
   </bean>

</beans>
```

*Illustration 39: **`CME.CDMLProducer.xml`** redux. Note the bean, and the file, are named after the producer, "CME". This name is arbitrary.*

A breakdown of this configuration follows

```
<bean name="CME" class="com.calypso.tk.cdml.producer.translation.CMEProducer">
```

Bean name matching the file name prefix. Using the `com.calypso.tk.cdml.producer.translation` package is not required, **although it's recommended** to follow the same code organization convention.

```
<property name="sourceFileOrganizer">
   <bean class="com.calypso.tk.cdml.producer.translation.source.CMESourceFileOrganizer">
      <property name="stripQuotes" value="true" />
      <property name="contentDelimiter" value="," />
   </bean>
</property>
```

CME `SourceFileOrganizer` implementation. Note `stripQuotes` is set to `true`, as CME escapes their files with them. Also, CME publishes CSVs (Comma Separated Values), so comma is the separator.

```
<property name="translationInstructions">
   <bean class="com.calypso.tk.cdml.producer.translation.instructions.factory.TranslationInstructionsLoaderBean">
      <property name="resourcePath" value="instructions/CMEProducer.translationInstructions.properties" />
      <property name="instructionIds">
         <set>
            <value>IRSMR3</value>
            <value>IRSTR</value>
         </set>
      </property>
   </bean>
</property>
```

Translation instructions map, loaded by a `TranslationInstructionsLoaderBean`: the values in `instructionIds` are both the instruction ids, and the keys in the resulting map. The **classpath location and naming of the properties file**, `instructions/CMEProducer.translationInstructions.properties`, **is arbitrary**, although it is recommended to follow the same approach

```
instructions/<producer class name>.translationInstructions.properties
```

9.2.2 Use case: LCH IRD translator

As in the CME IRD case, LCH IRD translator needs to produce both the `initialMarginReport` and `tradeValuationReport`. But in the LCH case the translation is more complex, mainly because the information is scattered across a variety of files.

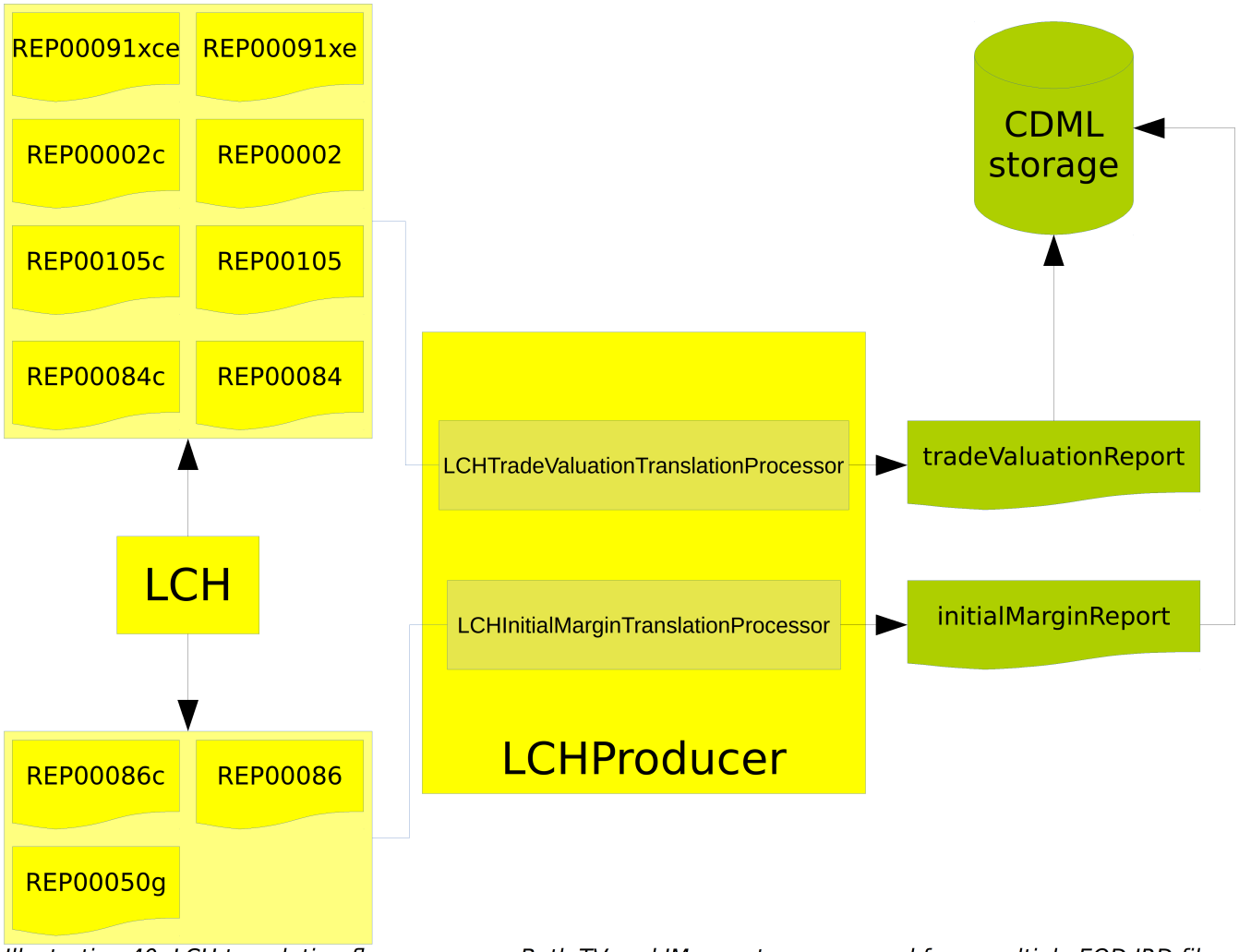


Illustration 40: LCH translation flow summary. Both TV and IM reports are sourced from multiple EOD IRD files

9.2.2.1 Translation instructions

The translation instructions for LCH IRD include both indexed and non-indexed files, depending on their purpose and structure. Translation ids are chosen to match LCH file naming, dropping date, description and suffix. This is similar to the CME approach, and it simplifies the handling to be done by the source organizer, producer and translation processors.

```
REP00086c.indexColumnName=ClientAccountId
REP00086c.columns=MbrMnemonic,Account,ClientAccountId,ReportingCCY,ConversionExchangeRate,InitialMargin,Liqui

REP00086.indexColumnName=Account
REP00086.columns=MbrMnemonic,Account,ClientAccountId,ReportingCCY,ConversionExchangeRate,InitialMargin,Liquid

REP00050g.indexColumnName=ClientAccountId
REP00050g.columns=TotalInitialMargin(SoR),ClientAccountId,ClientShortName

REP00091xce.indexColumnName=LchMatchedTradeRef
REP00091xce.indexColumnOptional=true
REP00091xce.columns=LchMatchedTradeRef,MbrMnemonic,Account,ClientShortName,Currency,CurrentNPV,NPVPreviousCOB

REP00091xe.indexColumnName=LchMatchedTradeRef
REP00091xe.indexColumnOptional=true
REP00091xe.columns=LchMatchedTradeRef,MbrMnemonic,SdMnemonic,Account,Currency,CurrentNPV,NPVPreviousCOB,Trade

REP00002c.indexColumnName=LchMatchedTradeRef
REP00002c.indexColumnOptional=true
REP00002c.columns=LchMatchedTradeRef,Amount,StateName,Currency

REP00002.indexColumnName=LchMatchedTradeRef
REP00002.indexColumnOptional=true
REP00002.columns=LchMatchedTradeRef,Amount,StateName,Currency

REP00105c.indexColumnName=LchMatchedTradeRef
REP00105c.columns=LchMatchedTradeRef,Currency,CouponStateName,FRASettlementAmount,PaymentDate,TradePV01

REP00105.indexColumnName=LchMatchedTradeRef
REP00105.columns=LchMatchedTradeRef,Currency,CouponStateName,FRASettlementAmount,PaymentDate,TradePV01

REP00084c.indexColumnName=LchMatchedTradeRef
REP00084c.indexColumnOptional=true
REP00084c.columns=LchMatchedTradeRef,Currency,TradePV01

REP00084.indexColumnName=LchMatchedTradeRef
REP00084.indexColumnOptional=true
REP00084.columns=LchMatchedTradeRef,Currency,TradePV01
```

Illustration 41: LCH translation instructions (content truncated). Note some of the reports are indexed (e.g. REP00086c), while others are to be parsed in a streaming, non-indexed fashion (e.g. REP00091xce). Also, note the different choice in index columns for some of the reports. E.g. for REP00086c `ClientAccountId` *is used, while for REP00086 is* `Account`*.*

9.2.2.2 Source and source file organizer

LCH source file organizing needs to deal with the fact that there's no member information included in the file name.

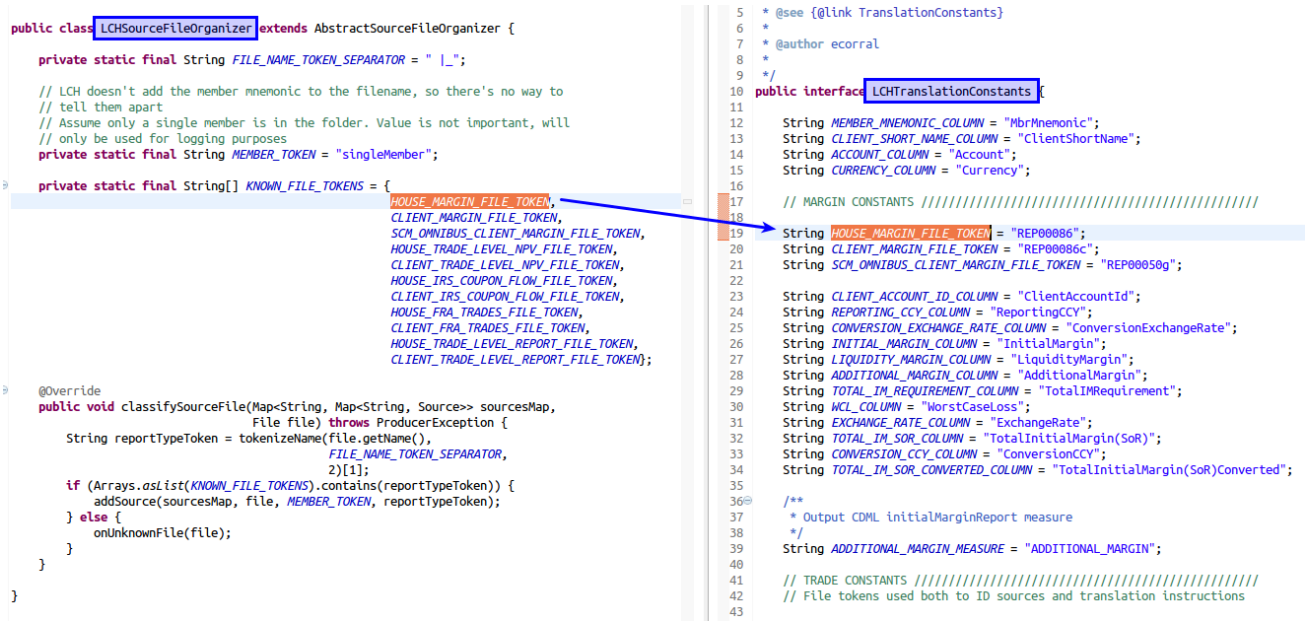


Illustration 42: General view of the `LCHSourceFileOrganizer`, with a partial `LCHTranslationConstants` view. `LCHTranslationConstants` groups all the hardcoded constant values shared by the classes that compose the LCH translator

A more detailed breakdown of the organizer code follows

```
private static final String MEMBER_TOKEN = "singleMember";
```

As stated before, there's no member information available in the filename: an arbitrary one is chosen.

Since 2.11/3.6, it is possible to place files for different members in separate CCP sub-folders (see 8.3.2.1 for more info).

9.2.2.3 Translation processors

9.2.2.3.1 LCHInitialMarginTranslationProcessor

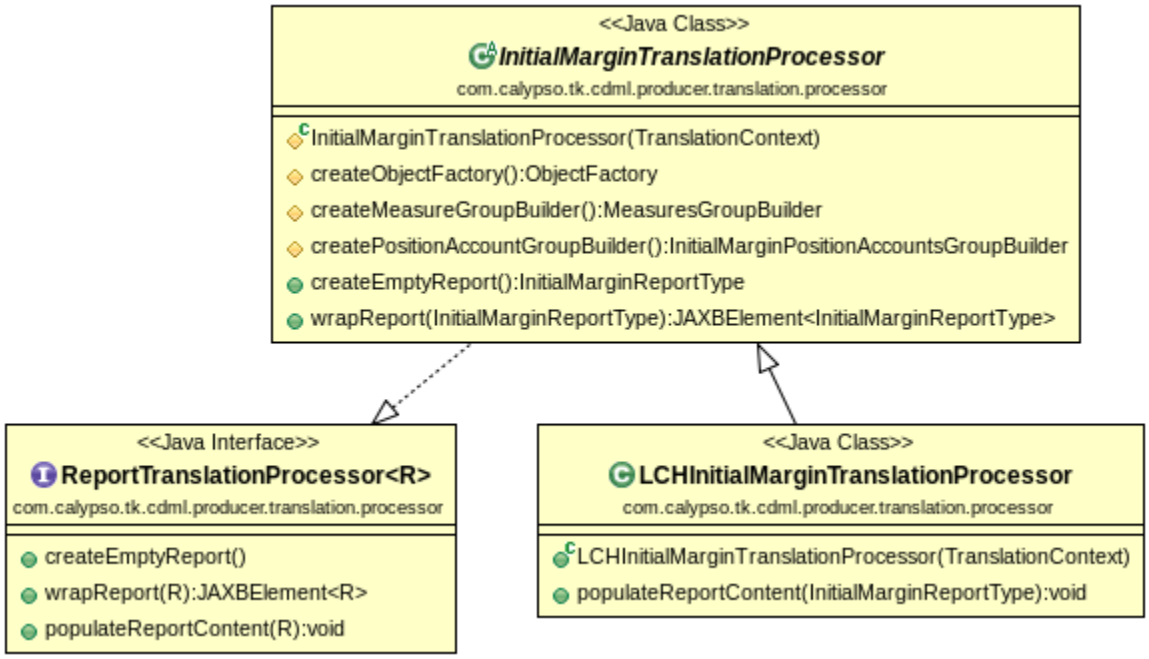


Illustration 43: LCHInitialMarginTranslatorProcessor hierarchy detail

LCH IM translation processor requires the default `SourceParser` for the three source files to parse, identified by the tokens

- `REP00086c`
- `REP00086`
- `REP00050g`

The `populateReportContent()` implementation is a simple delegation

```
@Override
public void populateReportContent(InitialMarginReportType initialMarginReport) throws ProducerException {
    SourceParser houseParser = (SourceParser) getContext().getParser(HOUSE_MARGIN_FILE_TOKEN);
    if (houseParser != null) {
        populateReportContent(initialMarginReport, houseParser, null, true);
    }

    SourceParser clientParser = (SourceParser) getContext().getParser(CLIENT_MARGIN_FILE_TOKEN);
    if (clientParser != null) {
        populateReportContent(initialMarginReport,
                              clientParser,
                              (SourceParser) getContext().getParser(SCM_OMNIBUS_CLIENT_MARGIN_FILE_TOKEN),
                              false);
    }
}
```

*Illustration 44: `LCHInitialMarginTranslationProcessor populateReportContent()` implementation. Note how the existence of parsers is not guaranteed. If parser is not in the context, it means that **its source wasn't found**. Even with one of the sources missing, content can still be generated with the other one: **translators should always try to generate as much content as possible with the existing sources***

The actual implementation occurs in the following method

```
private void populateReportContent(InitialMarginReportType initialMarginReport,
                                   SourceParser eodMarginParser,
                                   SourceParser osaMarginParser,
                                   boolean isHouseParser) throws ProducerException {

    for (String accountId : eodMarginParser.getIndexValues()) {
        // No ccy breakdown in LCH: each account has an already aggregated
        // number (86/86c)
        // So we don't expect more than 1 row per account
        Map<String, String> eodMarginRow = eodMarginParser.queryValuesByRow(accountId)
                .get(0);
```

Illustration 45: Actual `populateReportContent()` implementation, showing the loop detail, and the extraction of the single row for the loop index

As stated in the code comment, because of the structure of LCH REP00086/c margin reports, it is not expected to encounter more than one row per margin account: we can assume only the account aggregation is left, and proceed as such.

```
InitialMarginDataType initialMarginData = getObjectFactory().createInitialMarginDataType();
initialMarginReport.getInitialMarginDataElements().add(initialMarginData);

initialMarginData.setCCP(getContext().getCCPCode());
initialMarginData.setClearingService(IRD_CLEARING_SERVICE);
initialMarginData.setMemberId(eodMarginRow.get(MEMBER_MNEMONIC_COLUMN));
// locateHouseInitialMarginAccountId needs the 3 above to be set
String initialMarginAccountId = isHouseParser ? locateHouseInitialMarginAccountId(initialMarginData) : accountId;
// Still can be null if not found. Better add something as fallback
initialMarginData.setInitialMarginAccountId(initialMarginAccountId == null ? accountId : initialMarginAccountId);
initialMarginData.setSegregationAccount(eodMarginRow.get(ACCOUNT_COLUMN));

MeasuresGroupBuilder measuresGroupBuilder = createMeasureGroupBuilder();
initialMarginData.setMeasuresGroup(measuresGroupBuilder.getElement());

String requirementCcy = eodMarginRow.get(REPORTING_CCY_COLUMN);

// Maintenance
double maintenanceRequirement = parseDouble(eodMarginRow.get(TOTAL_IM_REQUIREMENT_COLUMN));
MeasureTypeBuilder maintenanceRequirementBuilder = new MeasureTypeBuilder(getObjectFactory(),
                                                                          MAINTENANCE_REQUIREMENT_MEASURE,
                                                                          maintenanceRequirement,
                                                                          requirementCcy);
measuresGroupBuilder.addMeasureFromBuilder(maintenanceRequirementBuilder);

// IM
double initialMargin = parseDouble(eodMarginRow.get(INITIAL_MARGIN_COLUMN));
MeasureTypeBuilder initialMarginBuilder = new MeasureTypeBuilder(getObjectFactory(),
                                                                 INITIAL_MARGIN_MEASURE,
                                                                 initialMargin,
                                                                 requirementCcy);
measuresGroupBuilder.addMeasureFromBuilder(initialMarginBuilder);
```

Illustration 46: Bulk of the `initialMarginDataType` creation, and addition of two measures to the measures group

REP00086/c doesn't provide a true currency breakdown. But in the case of LCH LLC, the native margins in GBP can be calculated, and included in CDML, with a dummy, single currency breakdown

```
String gbpCcy = CurrencyType.GBP.name();
if (!gbpCcy.equals(requirementCcy)) {
    double conversionRate = parseDouble(eodMarginRow.get(CONVERSION_EXCHANGE_RATE_COLUMN));
    if (!Util.isZero(conversionRate, Util.EPSILON)) {
        maintenanceRequirementBuilder.addBreakdown(gbpCcy, maintenanceRequirement / conversionRate, conversionRate);
        initialMarginBuilder.addBreakdown(gbpCcy, initialMargin / conversionRate, conversionRate);
        liquidityMarginBuilder.addBreakdown(gbpCcy, liquidityMargin / conversionRate, conversionRate);
        additionalMarginBuilder.addBreakdown(gbpCcy, additionalMargin / conversionRate, conversionRate);
    }
}
```

Illustration 47: Calculating original margins in GBP, using the conversion rate from the report

The result would look like this

```
<cdml-im:measures>
    <cdml-im:measure requirementCcy="USD" type="MAINTENANCE_REQUIREMENT" amount="-1366413552.33">
        <cdml-im:measureBreakdown nativeCcy="GBP" amount="-819094564.3987532" conversionFX="1.6682"/>
    </cdml-im:measure>
    <cdml-im:measure requirementCcy="USD" type="INITIAL_MARGIN" amount="-1192572443.92">
        <cdml-im:measureBreakdown nativeCcy="GBP" amount="-714885771.4422731" conversionFX="1.6682"/>
    </cdml-im:measure>
    <cdml-im:measure requirementCcy="USD" type="LIQUIDITY_MARGIN" amount="-173841108.42">
        <cdml-im:measureBreakdown nativeCcy="GBP" amount="-104208792.96247452" conversionFX="1.6682"/>
    </cdml-im:measure>
    <cdml-im:measure requirementCcy="USD" type="ADDITIONAL_MARGIN" amount="0">
        <cdml-im:measureBreakdown nativeCcy="GBP" amount="0" conversionFX="1.6682"/>
    </cdml-im:measure>
</cdml-im:measures>
```

See 10.3.1 Requirement vs. native margins for more info.

9.2.2.3.2 Populating the report from different sources: the OSA/ISA accounts case

Sometimes it is convenient to report upon margin at two different breakdown levels, provided the CCP supports it: margin account, and position account level. With the same translator output, it's just a configuration matter to produce downstream content at these two levels.

As seen in 7.2.2.4, CDML is capable of encoding both these levels by using the `initialMarginPositionAccountData` element. This is the approach taken with LCH REP00086c and REP00050g: the former will report at margin account level, the latter will offer a position account breakdown.

```
                                                                    requirementCurrency);

            // Additional
            MeasureTypeBuilder additionalMarginBuilder = createAndAddMeasure(measuresGroupBuilder,
                                                                              eodMarginRow,
                                                                              ADDITIONAL_MARGIN_MEASURE,
                                                                              ADDITIONAL_MARGIN_COLUMN,
                                                                              requirementCurrency);

            computeAndAddMeasureBreakdowns(requirementCurrency,
                                           CONVERSION_EXCHANGE_RATE_COLUMN,
                                           eodMarginRow,
                                           initialMarginBuilder,
                                           maintenanceRequirementBuilder,
                                           liquidityMarginBuilder,
                                           additionalMarginBuilder);

            if (osaMarginParser != null) {
                populateInitialMarginPositionAccounts(initialMarginData,
                                                      osaMarginParser);
            }

            initialMarginReport.getInitialMarginDataElements()
                    .add(initialMarginData);

        } catch (ParseException pe) {
            onParseException(pe, eodMarginParser, eodMarginRow);
        }

    } // End initialMarginAccountId checks

} // End margin account loop
```

Only run when REP00050g is available

Illustration 48: End of the main IM translation loop, showing how the position account margin level population is only invoked when the information is available. This is standard CDML practice: don't fail, but produce as much as possible with the available information

```
private void populateInitialMarginPositionAccounts(InitialMarginDataType initialMarginData,
                                                   SourceParser osaMarginParser) {
    List<Map<String, String>> osaAccountRows = osaMarginParser.queryValuesByRow(initialMarginData.getInitialMarginAccountId());
    if (!osaAccountRows.isEmpty()) {

        // Only 1 measure, 1 currency (GBP, as it's SCM only) per short name
        // Number is aggregated across all currencies and indexes

        InitialMarginPositionAccountsGroupBuilder positionAccountsBuilder = createPositionAccountGroupBuilder();
        initialMarginData.setInitialMarginPositionAccountsGroup(positionAccountsBuilder.getElement());
        // To allow easy access to the measure w/o having to dig into the
        // element hierarchy
        Map<String, MeasureType> measuresByShortName = new HashMap<String, MeasureType>();

        for (Map<String, String> osaAccountRow : osaAccountRows) {
            String clientShortName = osaAccountRow.get(CLIENT_SHORT_NAME_COLUMN);
            MeasureType measure = measuresByShortName.get(clientShortName);

            if (measure == null) {
                MeasuresGroupBuilder groupBuilder = positionAccountsBuilder.addPositionAccount(clientShortName);
                MeasureTypeBuilder measureBuilder = new MeasureTypeBuilder(getObjectFactory(),
                                                                           MAINTENANCE_REQUIREMENT_MEASURE,
                                                                           0.0d,
                                                                           CurrencyType.GBP.name());
                groupBuilder.addMeasureFromBuilder(measureBuilder);

                // Measure itself
                measure = measureBuilder.getElement();
                measuresByShortName.put(clientShortName, measure);
            }

            try {
                measure.setAmount(measure.getAmount()
                        + parseDoubleValue(osaAccountRow.get(TOTAL_IM_SOR_COLUMN)));
            } catch (ParseException pe) {
                onParseException(pe, osaMarginParser, osaAccountRow);
            }
        }
    }
}
```

Crossreferencing happens here: only rows related to the current initial margin element are considered

Each crossreferenced row will result in a separate initialMarginPositionAccount element

Illustration 49: Detail of the population of position level information for a given margin account

9.2.2.3.3 LCHTradeValuationTranslationProcessor

LCH TV generation is a complex case, because of its exceptions

- Sourced from several reports, including separate reports for FRAs and IRS
- Account ids coming from LCH need to be translated using the system configuration to produce usable CDML[11]

That said, **most of the parsing is driven by a single report**[12], REP00091xce/xe, which contains trade level NPV information. The processor will parse such report in a streaming fashion, and will enrich the information using the other aforementioned reports.

It is worth detailing the main steps here

11 *As a rule of thumb, CDML should be generated only with information coming from the CCP files. Sometimes, though, that information needs to be translated or enriched.* ***These cases should be the exceptions****, not the norm*

12 *Or report pair, House and Client*

```
private void populateReportContent(TradeValuationReportType tradeValuationReport,
                                   Parser tradeLevelNPVParser,
                                   boolean isHouseParser) {
    if (!(tradeLevelNPVParser instanceof StreamingSourceParser)) {
        throw new UnsupportedOperationException(String.format(PARSER_NOT_SUPPORTED_ERROR_MSG,
                                                              tradeLevelNPVParser));
    }
    StreamingSourceParser parser = (StreamingSourceParser) tradeLevelNPVParser;
```

First check is to make sure the appropriate parser has been created (see 9.2.2.4.3 Parser implementations), which in this case is a `StreamingSourceParser`[13]. Main loop then begins

```
Map<String, String> row = null;
try {
    while ((row = parser.readNextLine()) != null) {

        TradeType trade = getObjectFactory().createTradeType();
        // Don't add it yet, wait to see if we can locate the position account id
        trade.setCCP(getContext().getCCPCode());
        trade.setClearedTradeId(row.get(MATCHED_TRADE_REF_COLUMN));
        trade.setClearingService(IRD_CLEARING_SERVICE);
        trade.setMemberId(row.get(MEMBER_MNEMONIC_COLUMN));
        trade.setSegregationAccount(row.get(ACCOUNT_COLUMN));

        String positionAccountId = locatePositionAccountId(trade, row.get(isHouseParser ? SD_MNEMONIC_COLUMN : CLIENT_SHORT_NAME_COLUMN));
        if (positionAccountId != null) {
            trade.setPositionAccountId(positionAccountId);
            // Now add
            tradeValuationReport.getTrades().add(trade);
```

Note how the `TradeType` object is created, **but not added to the output collection**. Then it's used as a DTO object for `locatePositionAccountId()`, which will find the actual ***POSITIONACCOUNTID***[14]. Note also the different choice of column for House and Client reports.

Only if such account is found in the system the trade is added the the output collection.

---

13 *For the same reasons than in the CME case: trade level reports can be massive, and there's no need to group rows using an index*

14 *This is accomplished using the clearing accounts configured with an `LCHAccountName` attribute, and matching them with it*

```
FlowGroupBuilder flowGroupBuilder = createFlowGroupBuilder();
trade.setTradeCashFlowData(flowGroupBuilder.getElement());

ValuationGroupBuilder valuationGroupBuilder = createValuationGroupBuilder();
trade.setTradeValuationData(valuationGroupBuilder.getElement());

// NPV
double npv = parseDouble(row.get(CURRENT_NPV_COLUMN));
flowGroupBuilder.addFlow(NPV_FLOW, npv, settleCcy);
valuationGroupBuilder.addValuation(NPV_FLOW, npv, settleCcy);
valuationGroupBuilder.addValuation(RAW_NPV_FLOW, npv, settleCcy);

// Previous NPV
double previousNPV = parseDouble(row.get(NPV_PREV_COB_COLUMN));
flowGroupBuilder.addFlow(NPV_REV_FLOW, -previousNPV, settleCcy);
valuationGroupBuilder.addValuation(NPV_ADJUSTED_PREV_FLOW, previousNPV, settleCcy);
valuationGroupBuilder.addValuation(RAW_NPV_PREV_FLOW, previousNPV, settleCcy);

// Variation
double variation = npv - previousNPV;
roundAmount(flowGroupBuilder.addFlow(VARIATION_FLOW, variation, settleCcy));
valuationGroupBuilder.addValuation(VARIATION_FLOW, variation, settleCcy);

// PAI
double pai = parseDouble(row.get(TRADE_LEVEL_PAI_COLUMN));
flowGroupBuilder.addFlow(PAI_FLOW, pai, settleCcy);
valuationGroupBuilder.addValuation(PAI_FLOW, pai, settleCcy);

// Upfront Fee
double upfrontFee = parseDouble(row.get(CONSIDERATION_COLUMN));
flowGroupBuilder.addFlow(UPFRONT_FEE_FLOW, upfrontFee, settleCcy);
valuationGroupBuilder.addValuation(UPFRONT_PAYMENT_FLOW, upfrontFee, settleCcy);
```

As stated before, most of the information is taken from REP00091xce/xe. Show above is the sourcing of NPV, Previous NPV, Variation[15], PAI and Upfront Fees flows. Same amount is reused for the corresponding valuation elements.

15 *Synthetic flow: not directly provided by LCH, although the computation is trivial, given the presence of NPV and Previous NPV*

```
// Coupon
Amount coupon = getCoupon(trade.getClearedTradeId(), isHouseParser);
if (coupon != null) {
    flowGroupBuilder.addFlow(COUPON_FLOW, coupon.amount, coupon.currency);
}
```

Coupon is not an always present flow, so it's existence is validated. See 9.2.2.3.4 Parsing coupons for more info on obtaining coupons. The same happens with FRA payments

```
// FRA payment
Amount fraPayment = getFRAPayment(tradeValuationReport, trade.getClearedTradeId(), isHouseParser);
if (fraPayment != null) {
    flowGroupBuilder.addFlow(FRA_PAYMENT_FLOW, fraPayment.amount, fraPayment.currency);
}
```

Finally, the trade element is completed with the remaining valuations

```
// Accrued payleg
valuationGroupBuilder.addValuation(ACCRUAL_PAYLEG_FLOW, parseDouble(row.get(ACCRUED_PAYLEG_COUPON_COLUMN)), settleCcy);

// Accrued recleg
valuationGroupBuilder.addValuation(ACCRUAL_RECLEG_FLOW, parseDouble(row.get(ACCRUED_RECLEG_COUPON_COLUMN)), settleCcy);

// DV01/PV01
Amount pv01 = getPV01(trade.getClearedTradeId(), isHouseParser);
if (pv01 != null) {
    // Spec doesn't define what happens if missing
    valuationGroupBuilder.addValuation(DV01_FLOW, pv01.amount, pv01.currency);
}
```

9.2.2.3.4 Parsing coupons

Coupon information is sourced from a separate report, REP00002/c. These reports detail the coupon flow information **for every registered IRS trade**. That means a sizable amount of report rows: a streaming approach is desirable in this case, with minimum column pulling.

```
REP00002c.indexColumnName=LchMatchedTradeRef
REP00002c.indexColumnOptional=true
REP00002c.columns=LchMatchedTradeRef,Amount,StateName,Currency
```

Illustration 50: REP00002c translation instructions detail. Note only the minimum amount of columns are pulled from the report

When a coupon amount for a given trade id is requested, lazy parsing of the House or Client reports occurs

```
private Amount getCoupon(String lchTradeRefId, boolean isHouseParser) throws IOException {
    Map<String, Amount> cachedCoupons = null;

    // The coupon maps set to null indicate parsing hasn't yet happened, or ar least tried

    if (isHouseParser) {
        if (readHouseCoupons == null) {
            readHouseCoupons = new HashMap<String, Amount>();
            parseCoupons(readHouseCoupons, getContext().getParser(HOUSE_IRS_COUPON_FLOW_FILE_TOKEN));
        }
        cachedCoupons = readHouseCoupons;
    } else {
        if (readClientCoupons == null) {
            readClientCoupons = new HashMap<String, Amount>();
            parseCoupons(readClientCoupons, getContext().getParser(CLIENT_IRS_COUPON_FLOW_FILE_TOKEN));
        }
        cachedCoupons = readClientCoupons;
    }

    return cachedCoupons.get(lchTradeRefId);
}
```

Raw parsed reports will be transformed to maps of {trade ids – `Amount` instances}. From there, obtaining the coupon is trivial. `Amount` is a simple intermediate structure to ease mapping

```
private class Amount {
    String currency;
    double amount;
}
```

The actual parsing and map creation occurs in `parseAmounts()`, which is also used for PV01/DV01

```
private void parseAmounts(Map<String, Amount> cachedAmounts,
                          Parser amountParser,
                          String amountColumn,
                          Filter<Map<String, String>> rowFilter) throws IOException {
    Assert.notNull(cachedAmounts);
    Assert.notNull(rowFilter);
    Assert.hasText(amountColumn);
    if (amountParser != null) {
        if (!(amountParser instanceof StreamingSourceParser)) {
            throw new UnsupportedOperationException(String.format(PARSER_NOT_SUPPORTED_ERROR_MSG,
                                                                  amountParser));
        }
        StreamingSourceParser parser = (StreamingSourceParser) amountParser;

        Map<String, String> row = null;
        while ((row = parser.readNextLine()) != null) {
            if (rowFilter.accept(row)) {
                String tradeRef = row.get(MATCHED_TRADE_REF_COLUMN);
                Amount couponAmount = cachedAmounts.get(tradeRef);
                if (couponAmount == null) {
                    couponAmount = new Amount();
                    couponAmount.amount = 0.0d;
                    couponAmount.currency = row.get(CURRENCY_COLUMN);
                    cachedAmounts.put(tradeRef, couponAmount);
                }
                couponAmount.amount += parseDouble(row.get(amountColumn));
            }
        }
    }
}
```

Coupons are aggregated per trade, **provided the row matches the right criteria**. For coupons, that is

```
private void parseCoupons(Map<String, Amount> cachedCoupons, Parser couponParser) throws IOException {
    parseAmounts(cachedCoupons, couponParser, AMOUNT_COLUMN, new Filter<Map<String, String>>() {
        @Override
        public boolean accept(Map<String, String> row) {
            return PAYMENT_INSTRUCTED_COLUMN_VALUE.equals(row.get(STATE_COLUMN));
        }
    });
}
```

Which means that **only coupon legs[16] being scheduled for payment as of the report date will be aggregated**. For DV01/PV01, which is **published everyday**, no filtering is needed

```
private void parsePV01(Map<String, Amount> cachedPV01, Parser pv01Parser) throws IOException {
    parseAmounts(cachedPV01, pv01Parser, TRADE_PV01_COLUMN, new Filter<Map<String, String>>() {
        @Override
        public boolean accept(Map<String, String> row) {
            return true;
        }
    });
}
```

16 *Pay and receive legs, normally*

9.2.2.4 LCHProducer

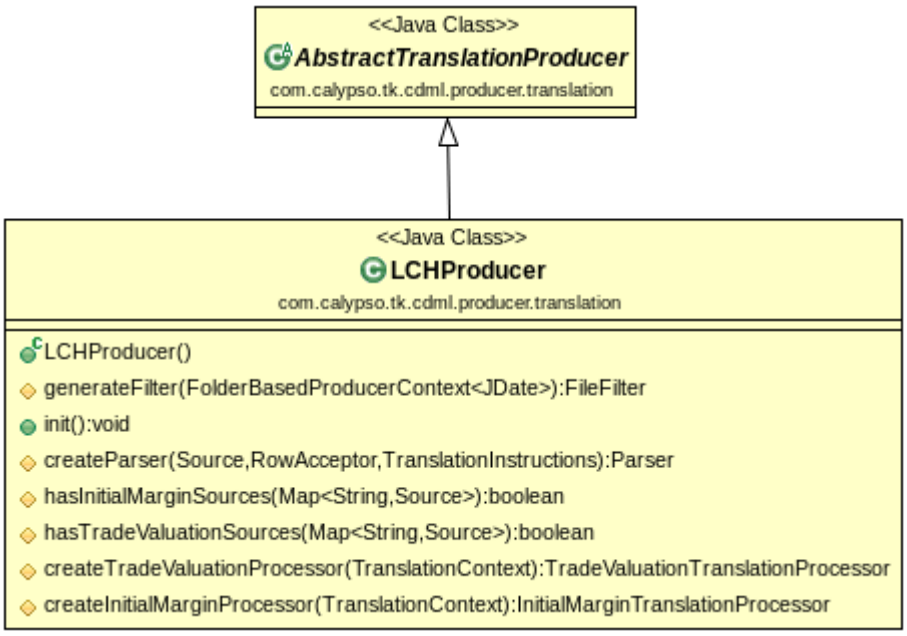


Illustration 51: `LCHProducer` hierarchy detail. See Illustration 3 for an extended hierarchy

9.2.2.4.1 Filtering sources

```
// Matcher is date + separator + report type + free text +  TXT suffix
private static final String REPORT_MATCHER_SUFFIX = "_REP00(086|086c|050g|091xe|091xce|002|002c|105|105c|084|084c) -.*.(TXT|txt)";

@Override
protected FileFilter generateFilter(FolderBasedProducerContext<JDate> context) {
    return new FilenameRegexpFileFilter(toISOString(context.getReportDate(),
                                                    false)
            + REPORT_MATCHER_SUFFIX);
}
```

Illustration 52: LCH source filter expression. Date is prepended, and a simple report token list follows

9.2.2.4.2 Row acceptor

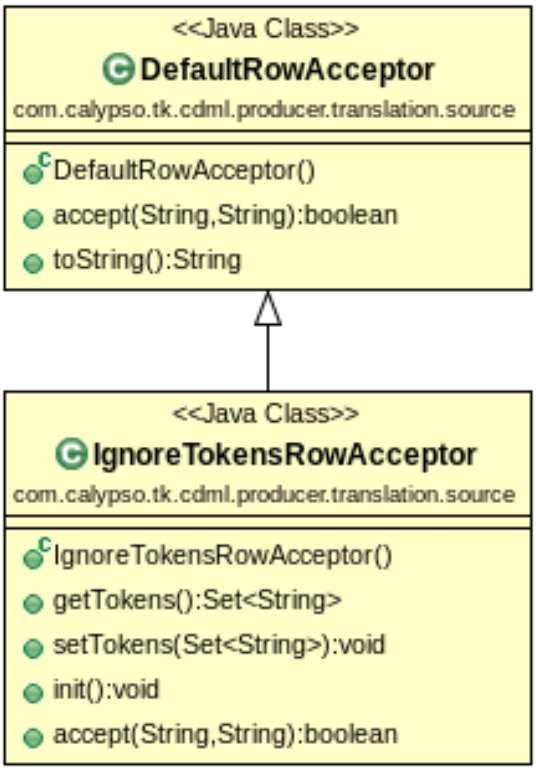


Illustration 53: `IgnoreTokensRowAcceptor` will drop empty rows, and those which contain one of the configured tokens

LCH reports can contain header and footer information that need to be ignored when generating CDML content.

	A	B	C	D
	Report Creation Date/Time: 20140219 01:32			
	CobDate	Party_A_MbrMnemonic	Party_A_SdMnemonic	Party_A_SdTradeId
	2014/02/18	CC1	MEGACALPCC	12118884-2
	Number of Rows: 1.			

Illustration 54: REP00105c report, highlighting the creation datetime header and row count footer

`IgnoreTokensRowAcceptor` is configured by the Spring context, and set into the producer.

```
<property name="rowAcceptor">
   <bean class="com.calypso.tk.cdml.producer.translation.source.IgnoreTokensRowAcceptor">
      <property name="tokens">
         <set>
            <value>Report Creation Date/Time</value>
            <value>Totals</value>
            <value>Number of Rows</value>
         </set>
      </property>
   </bean>
</property>
```

Illustration 55: Detail of the acceptor Spring configuration for LCH

9.2.2.4.3 Parser implementations

`createParser()` overrides the default creation when dealing with trade level information reports.

```
@Override
protected Parser createParser(Source source,
                              RowAcceptor rowAcceptor,
                              TranslationInstructions instr) {
    return CLIENT_TRADE_LEVEL_NPV_FILE_TOKEN.equals(instr.getId())
            || HOUSE_TRADE_LEVEL_NPV_FILE_TOKEN.equals(instr.getId())
            || CLIENT_IRS_COUPON_FLOW_FILE_TOKEN.equals(instr.getId())
            || HOUSE_IRS_COUPON_FLOW_FILE_TOKEN.equals(instr.getId())
            || CLIENT_TRADE_LEVEL_REPORT_FILE_TOKEN.equals(instr.getId())
            || HOUSE_TRADE_LEVEL_REPORT_FILE_TOKEN.equals(instr.getId())
                                                        ? new StreamingSourceParser(instr,
                                                                                    source,
                                                                                    rowAcceptor)
                                                        : super.createParser(source,
                                                                             rowAcceptor,
                                                                             instr);
}
```

Illustration 56: Parser override for LCH: trade level and coupon reports are better handled with streaming parsers

9.2.2.4.4 Spring context

```
<bean name="LCH" class="com.calypso.tk.cdml.producer.translation.LCHProducer">
   <property name="sourceFileOrganizer">
      <bean class="com.calypso.tk.cdml.producer.translation.source.LCHSourceFileOrganizer">
         <property name="stripQuotes" value="true" />
         <property name="contentDelimiter" value="\t" />
      </bean>
   </property>
   <property name="rowAcceptor">
      <bean class="com.calypso.tk.cdml.producer.translation.source.IgnoreTokensRowAcceptor">
         <property name="tokens">
            <set>
               <value>Report Creation Date/Time</value>
               <value>Totals</value>
               <value>Number of Rows</value>
            </set>
         </property>
      </bean>
   </property>
   <property name="translationInstructions">
      <bean class="com.calypso.tk.cdml.producer.translation.instructions.factory.TranslationInstructionsLoaderBean">
         <property name="resourcePath" value="instructions/LCHProducer.translationInstructions.properties" />
         <property name="instructionIds">
            <set>
               <value>REP00086</value>
               <value>REP00086c</value>
               <value>REP00050g</value>
               <value>REP00091xce</value>
               <value>REP00091xe</value>
               <value>REP00002c</value>
               <value>REP00002</value>
               <value>REP00105c</value>
               <value>REP00105</value>
               <value>REP00084c</value>
               <value>REP00084</value>
            </set>
         </property>
      </bean>
   </property>
</bean>
```

Illustration 57: `LCHProducer` bean detail. Note the `rowAcceptor` setup, as well as the tab separator (`contentDelimiter`)

9.3 NEW XML translators

As of now, only one XML translator exists. The following is subject to change if new, unsupported XML-to-XML cases are encountered.

With the lessons learned from previous translators, the XML-to-XML support was built to reduce, even further, the complexity of the **CCP-specific** details of a given translator. The only current example, EUREX, **doesn't require any CCP-specific Java code**. As we'll see, Spring configuration and XSLT files are all that is required.

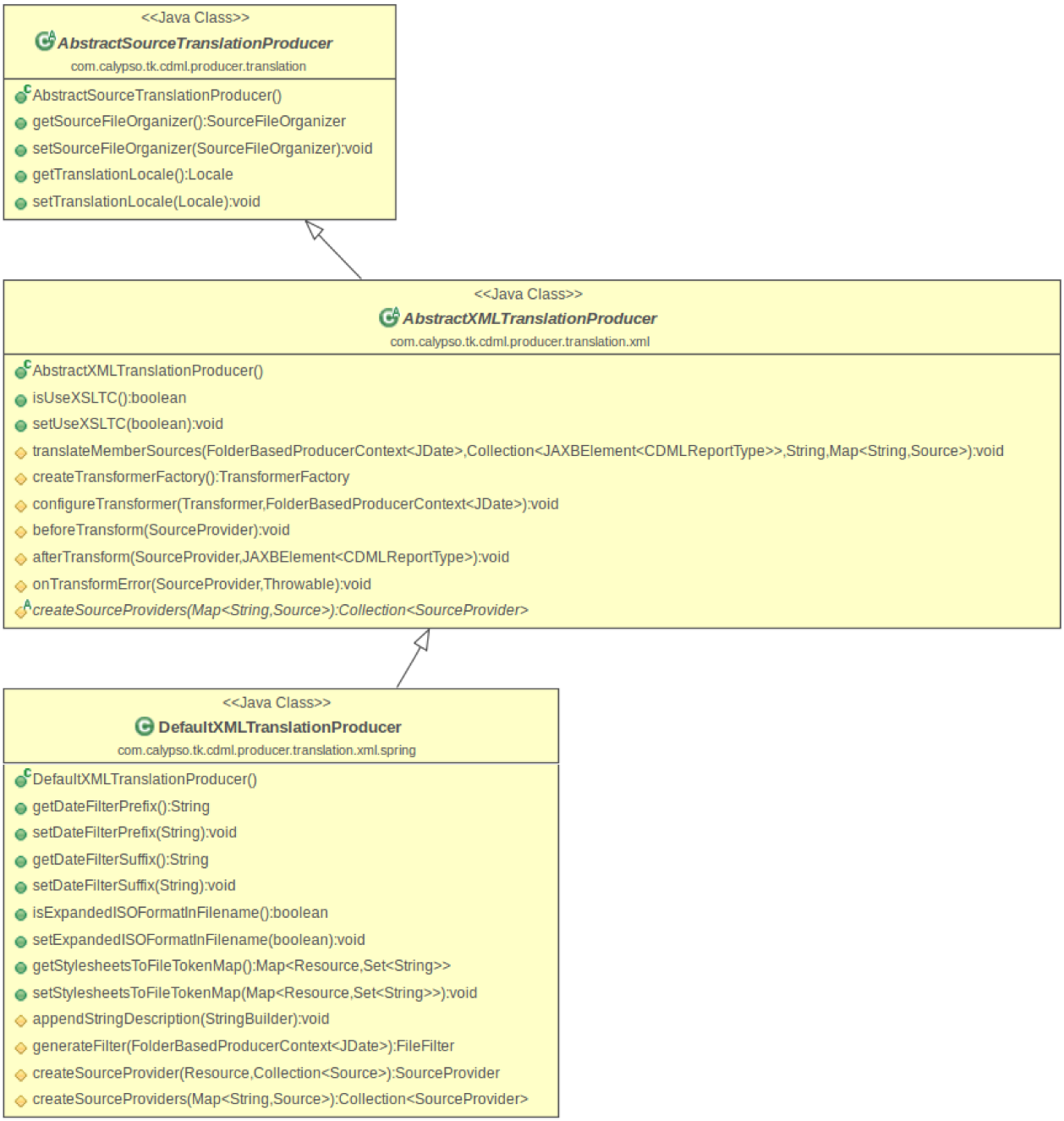


Illustration 58: XML translation producers, including the default implementation.

9.3.1 From input files, to javax.xml.transform.Source instances

The XML-to-XML translator implementation makes use of the **Java API for XML Processing (JAXP)**. The idea is to provide a pair of resources for each CDML report that is to be produced

- The content `javax.xml.transform.Source`[17] : this is the CCP content to be transformer
- The transformation stylesheet `javax.xml.transform.Source` : these are the instructions by which the CCP content is translated to CDML

This idea is captured in the new `SourceProvider` interface

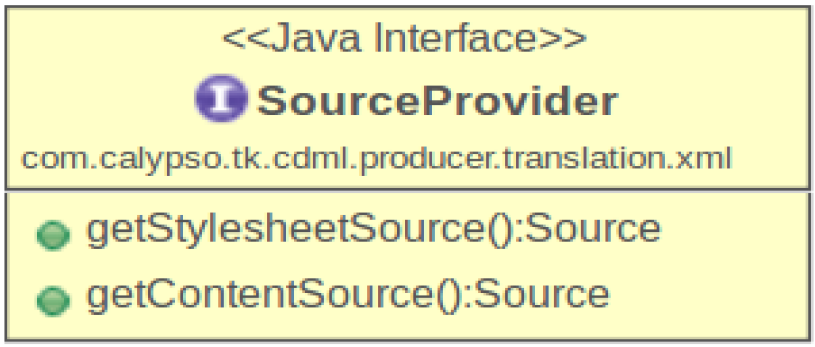


Illustration 59: `SourceProvider` interface

As seen in 9.1, every translation producer is configured with a `SourceFileOrganizer` that performs an initial classification of the input files. Initially, the idea was to separate files per member, and then per purpose (margin, valuation). The problem comes when **more than one input file is required to populate a single output CDML file**. This case (e.g. EUREX, LCH, COMDER) is more common than the "*1 input CCP file to 1 output CDML report*" one (e.g. CME). The result is that the translator implementations need to host the **extra code** to group input files by purpose to produce a single output file.

XML translators take that concept further with a **second classification**, whose goal is to produce a **collection of `SourceProvider` implementations**, and save the producer implementations from reimplementing it.

17 *Not to be mistaken with the CDML organizer `Source`*

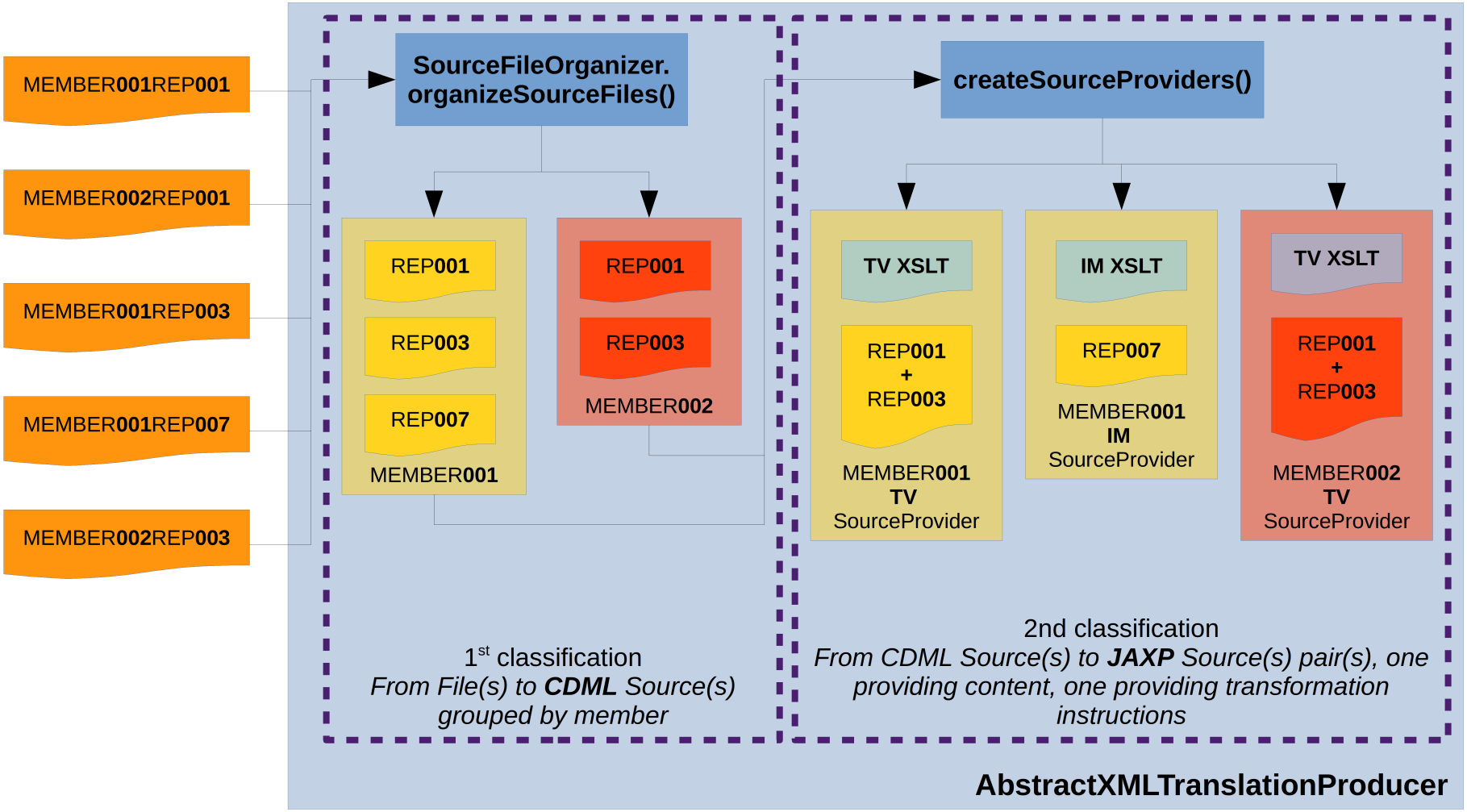


Illustration 60: Two phase classification: `AbstractXMLTranslationProducer` delegates onto its `SourceFileOrganizer` to separate reports per member, "tag" them with a key or token, and wrap them into a CDML `Source`. Then the producer implementation performs a second classification, in which sources are grouped by purpose, producing a collection of `SourceProviders`. Each `SourceProvider` is then ready to produce an IM or TV report using the attached XSLT instructions

9.3.2 DefaultXMLTranslationProducer and ResourceMergingSourceProvider

`DefaultXMLTranslationProducer` is the only current `AbstractXMLTranslationProducer` implementation. It uses a **map of stylesheets** to group the input files into the expected `SourceProvider` implementations. See 9.3.3.1 for an example of this map.

The grouped input files, along with the selected stylesheet, are configured in a `ResourceMergingSourceProvider`. This provider **concatenates all input XML files into a single merged one**[18], which will be the content `Source` used in the XML-to-XML transformation.

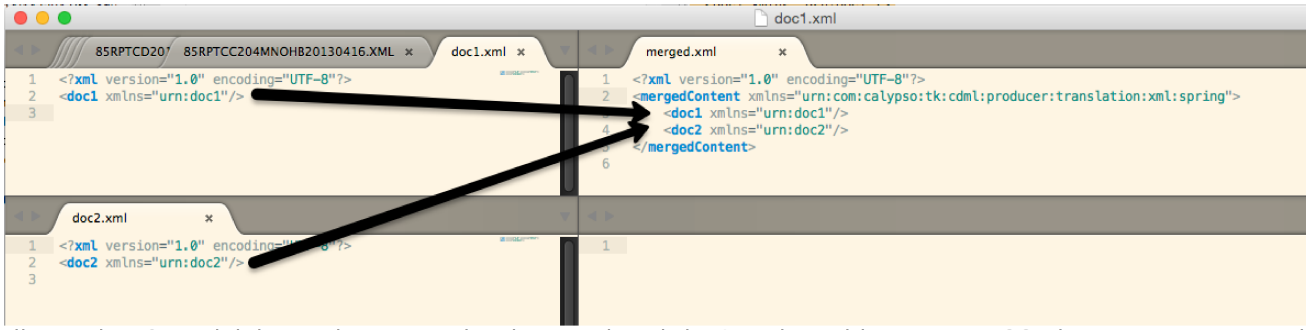


*Illustration 61: Trivial merging example: doc1.xml and doc2.xml would represent CCP inputs concatenated into merged.xml. In real life, **no merged file will be created**: content is merged, and then transformed in memory.*

The merged approach was chosen over the multiple, separate content Sources, for the following reasons

- **JAXP API doesn't allow multiple input content Sources**: only one can be passed on each transformation
- The `document()` XPath function needs to be used to reference the remaining content sources. This function is implemented differently in different XSLT processors, and requires absolute paths to the content sources. **These paths are not known on compile time**
- Having to choose one of the input files over the others as the one passed to JAXP is an arbitrary choice, better to be avoided

9.3.3 Use case: EUREX IRD translator

As stated before, the EUREX translator only requires

- The XSL files for TV and IM
- The Spring configuration file

18 *No merging is done if only one input file is provided*

9.3.3.1 Spring file

A commented breakdown of the file follows.

```
<bean name="EUREX" class="com.calypso.tk.cdml.producer.translation.xml.spring.DefaultXMLTranslationProducer">
```

Producer name is “EUREX”, and the code is provided by core clearing.

```
<property name="dateFilterPrefix" value="(84|85)RPT.*" />
```

`dateFilterPrefix` and `dateFilterSuffix` are **regular expression fragments** used by `generateFilter()` to narrow down the set of input files to be considered by later stages. See 9.2.1.5.1 for more info.

```
<property name="sourceFileOrganizer">
   <bean class="com.calypso.tk.cdml.producer.translation.source.regex.RegexSourceFileOrganizer">
      <property name="matchers">
         <list>
            <bean
               class="com.calypso.tk.cdml.producer.translation.source.regex.FilenameMatcher">
               <!-- The regular expression to tokenize the EUREX report names -->
               <!-- 2 digits for the environment -->
               <!-- Then RPTXYYY -->
               <!-- Then member id, 5 characters -->
               <!-- Then date in compact ISO format -->
               <!-- Then optional time as hhmm for ITD reports -->
               <constructor-arg value="[0-9]{2}+(RPTC[A-Z][\d]{3}+)([\S]{5}+)[\d]{8}+([\d]{4}+){0,1}?\.XML" />
               <!-- Report type capturing group index -->
               <constructor-arg value="1" />
               <!-- Member id capturing group index -->
               <constructor-arg value="2" />
            </bean>
         </list>
      </property>
   </bean>
</property>
```

Generic regex organizer. The regular expression serves two purposes:

- Further narrow down the set of accepted files, beyond `generateFilter()`
- Extract the member id and report type from the input file name

This is an alternative to the code shown in 9.2.1.2 and 9.2.2.2. In the configuration above, two capturing groups are defined, their boundaries delimited by (), which will serve to extract the **report type or token** (first capturing group) and

member id (second capturing group).

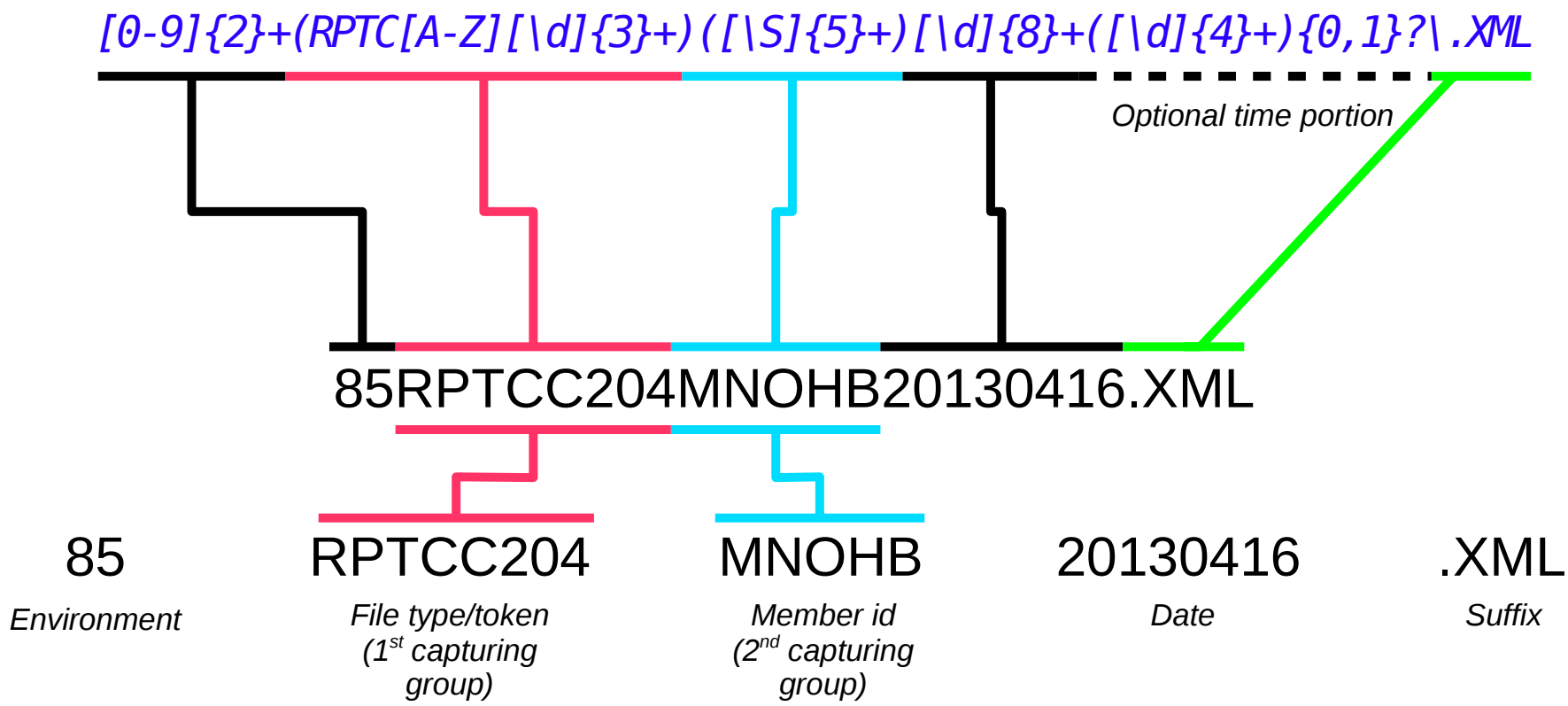


Illustration 62: Sample regex breakdown of an EUREX filename. Only the first and second capturing groups are used, as seen in the bean configuration. The remaining portions are useful to validate only the expected files are transformed, but such information is not used in dowstream processing

```
<property name="stylesheetsToFileTokenMap">
   <map>
      <entry key="classpath:stylesheet/cdml/EUREXTradeValuationReport.xslt">
         <set>
            <value>RPTCC203</value>
            <value>RPTCI280</value>
            <value>RPTCD200</value>
            <value>RPTCB202</value>
         </set>
      </entry>

      <entry key="classpath:stylesheet/cdml/EUREXInitialMarginReport.xslt">
         <set>
            <value>RPTCC204</value>
         </set>
      </entry>
   </map>
</property>
```

Mapping definition. With the file tokens extracted with the regular expressions, files will be grouped around a stylesheet, and together will populate the `SourceProvider`.

Note the keys to the map are the actual Spring classpath resources pointing to the XSLT files containing the TV and IM instructions, respectively.

9.3.3.2 XSLT instructions

The XSL transformation **must produce valid CDML content** (as of today, TV or IM). Any approach is allowed, as long as this condition is met.

Translator writers can make use of the following special import namespaces to bring templates that will populate the report headers

- `urn:cdml:transform:initialMarginReportBase` for IM
- `urn:cdml:transform:tradeValuationReportBase` for TV

For each of those, the following XSL templates must then be implemented, respectively

- `initialMarginReportContentTemplate` for IM
- `tradeValuationReportContentTemplate` for TV

A sample IM XSL would begin like this

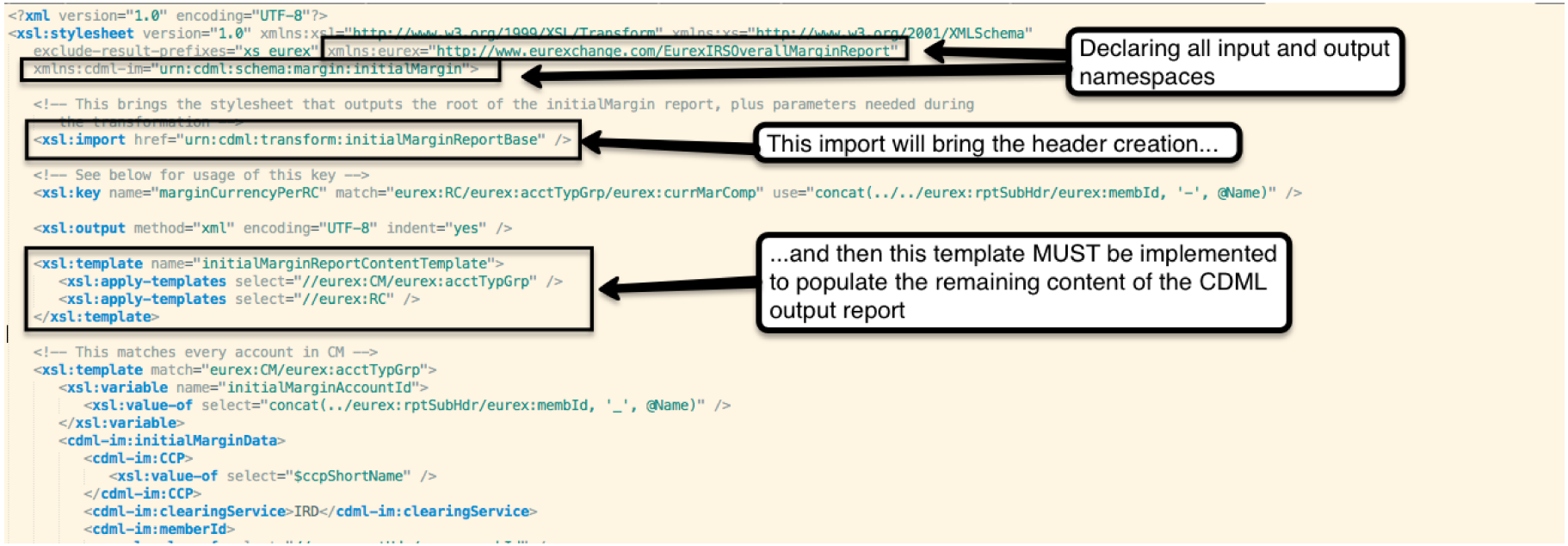


Illustration 63: Start of the stylesheet that produces the CDML IM report. Having the special urn:cdml:transform:initialMarginReportBase import saves the writer from re-implementing the header output, at the cost of needing to have a template named "initialMarginReportContentTemplate"

As seen in 9.3.2, when multiple files are required to generate a single CDML report, those are merged into a single one. The stylesheet **needs to reference all the input and output namespaces**, so fully qualified XPath expressions can be used.


```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   exclude-result-prefixes="xs r203 r200 r202 r280"
   xmlns:r203="http://www.eurexchange.com/EurexIRSVariationMarginReport"
   xmlns:r200="http://www.eurexchange.com/EurexIRSCashForecastReport"
   xmlns:r202="http://www.eurexchange.com/EurexIRSFullInventoryReport"
   xmlns:r280="http://www.eurexchange.com/EurexIRSSettlementReport"
   xmlns:cdml-tv="urn:cdml:schema:position:tradeValuation"
   xmlns:cdml="urn:cdml:schema:common:types">

   <!-- This brings the stylesheet that outputs the root of the tradeValuation report, plus parameters needed during
   the transformation -->
   <xsl:import href="urn:cdml:transform:tradeValuationReportBase" />

   <xsl:output method="xml" encoding="UTF-8" indent="yes" />

   <xsl:template name="tradeValuationReportContentTemplate">

      <xsl:apply-templates select="//r203:CCPTrade" />

   </xsl:template>

   <xsl:template match="r203:CCPTrade">
      <xsl:variable name="tradeId" select="@Id" />

      <cdml-tv:trade>
         <cdml-tv:clearedTradeId>
            <xsl:value-of select="$tradeId" />
         </cdml-tv:clearedTradeId>
         <cdml-tv:CCP>
            <xsl:value-of select="$ccpShortName" />
         </cdml-tv:CCP>
         <cdml-tv:clearingService>IRD</cdml-tv:clearingService>
         <cdml-tv:memberId>
            <xsl:value-of select="../../../../r203:rptHdr/r203:membId" />
         </cdml-tv:memberId>
         <cdml-tv:positionAccountId>
            <xsl:value-of select="../../r203:rptSubHdr/r203:membId" />
         </cdml-tv:positionAccountId>
         <cdml-tv:segregationAccount>
            <xsl:choose>
               <xsl:when test="../@Name = 'P' and local-name(../..) = 'CM'">
                  <xsl:value-of select="'H'" />
               </xsl:when>
               <xsl:otherwise>
                  <xsl:value-of select="'C'" />
               </xsl:otherwise>
            </xsl:choose>
         </cdml-tv:segregationAccount>

         <!-- FLOWS -->
```

Illustration 64: TV EUREX stylesheet. Each input file comes with its own namespace, and declared here. Later, in the transformation instructions, the namespaces can be used to select the portion of input merged XML file needed to generate a subset of the output

10 Processing CDML

This section describes some general concepts of the CDML processing code, and how it's kicked off by the *CLEARING_PROCESS_FROM_CDML* `ScheduledTask`.

This is not intended to be a formal specification of the CDML processing rules. For such information, visit the shared *CDML Processing Rules* and *CDML Scheduled Tasks* shared Box folders at https://calypso.box.com/s/khx18p3sfew9ttj4uvdx and https://calypso.box.com/s/d9r9unlqsvrjklb8g4xk, respectively

10.1 The CLEARING_PROCESS_FROM_CDML ScheduledTask

The CLEARING_PROCESS_FROM_CDML `ScheduledTask` is the **BO processing task** that **replaces** the previous object-creating Clearing ones, such as (but not limited to)

- **CLEARING_BO_MARGIN**
- **CLEARING_SETTLEMENT**
- **CLEARING_MARKS**

It takes CDML content **previously stored**, and updates the system state by creating Calypso core objects

- Collateral Exposure trades
- ClearingTransfer trades
- PLMarks

Given that CDML is aimed to be a fully descriptive report, and the way it's stored (see 8.3.4 CDML storage), **there's is no need to create separate instances of the scheduled task for different CCPs, POs, or products**: all the information is contained in the same CDML report instance.

The ST valuation date determines the CDML report instance/s to be loaded from persistence.

10.1.1 Arguments

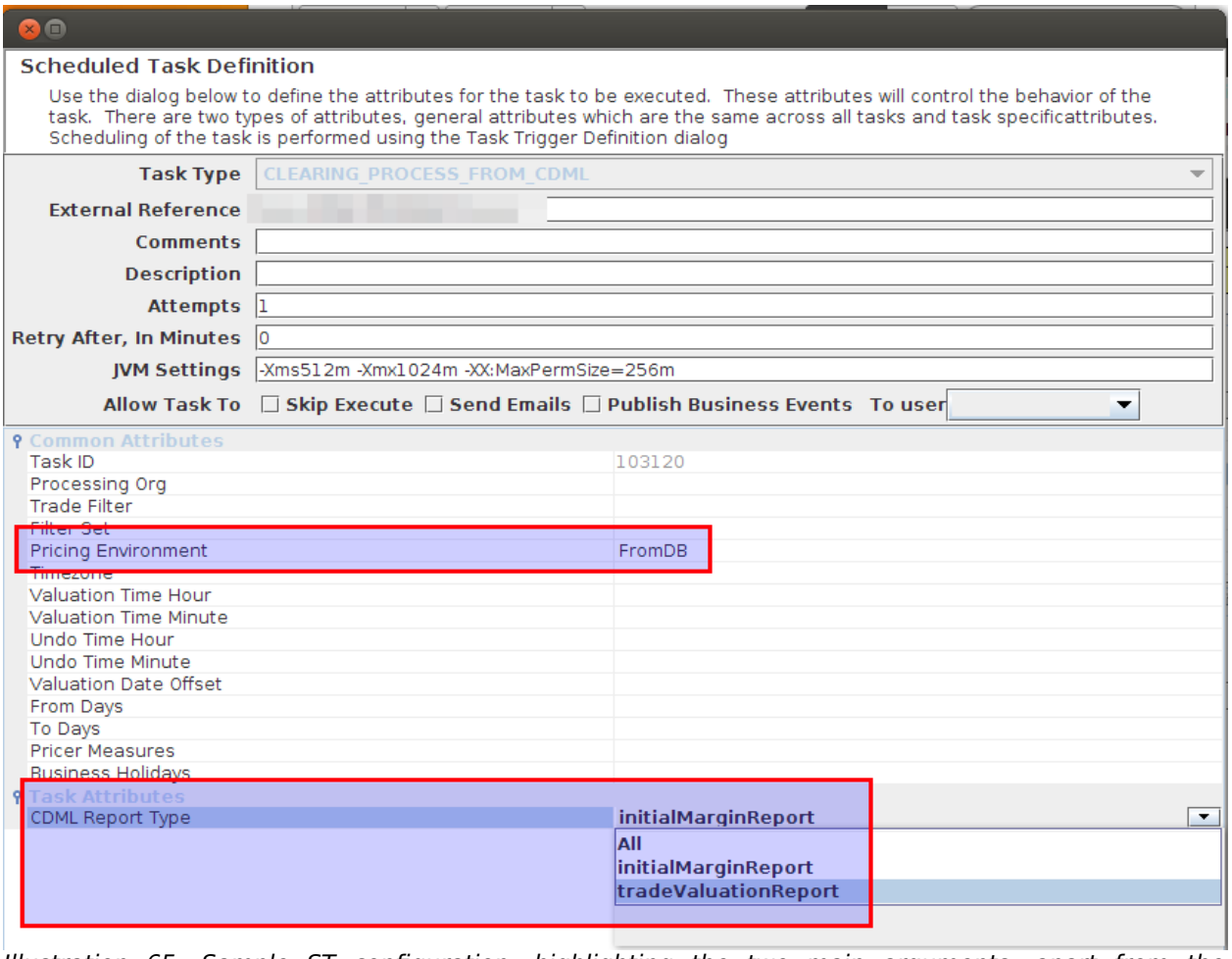


Illustration 65: Sample ST configuration, highlighting the two main arguments, apart from the valuation datetime: CDML Report Type and Pricing Environment

10.1.1.1 CDML Report Type

Users can choose to process all CDML content for the given day, or focus only in one report type. As of now, the available reports are

- `initialMarginReport`
- `tradeValuationReport`

See 7 CDML types for more info.

10.1.1.2 Pricing Environment

Given that the pricing environment is a Calypso concept, and it's not available directly in CDML, the ST requires this argument to be configured: in some cases, like the **creation of product trade `PLMarks`**, there is no way to obtain it from any related object.

In other cases in which a Calypso core object proves to be a better source, such as

Collateral Exposures `PLMarks`[19], it will be ignored.

10.1.2 Execution summary

The scheduled task responsibilities are minimal

- Obtain the CDML report instance/s for the given valuation date
- Instantiate and configure the appropiate `CDMLConsumer` for each report loaded
- Invoke the consumer's `onReport()`

19 *MCCs store both ITD and EOD pricing environment references*

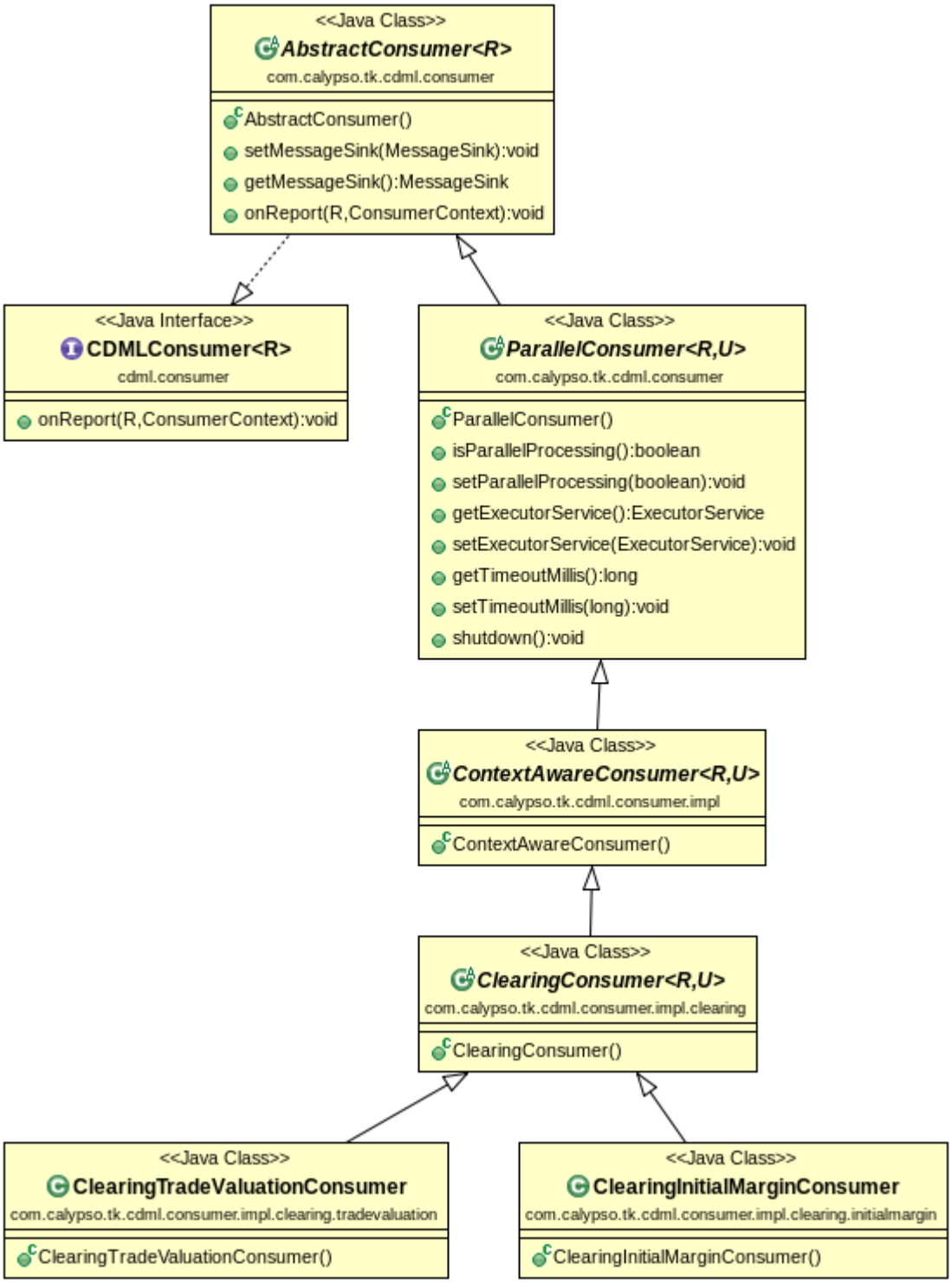


Illustration 66: `CDMLConsumer` hierarchy, showing only public methods

There are no customization points so far in the `CDMLConsumer` hierarchy, mainly because **there's is no need to**: at this point, all content is CDML, and encoded equally, regardless the CCP, product/service, member or account.

10.2 Processing the tradeValuationReport

The trade valuation CDML processing can be summarized with the following steps

- Group `<trade>` elements by
  - **CCP**
  - **Clearing Service**[20]
  - **Member Id** (PO at the given CCP)
  - **Position Account Id**
  - **Currency**. See 10.2.1 Trades with flows in multiple currencies
- **For every of these groups**
  - Create a `ClearingTransfer` trade, **in memory**, with the aforementioned details
  - Aggregate the `<trade>` elements flows, if matching the group currency, and attach them to the `ClearingTransfer` as fees
    - **Fee date** is calculated following **clearing standard practices**[21], except for the cases in which a specific `settleDate` is present in the input CDML. See 10.2.2 Flow/fee settle date override
  - Transfer settlement amount is computing by adding all fee values[22]
  - A lookup is performed, to verify if an existing transfer already exists
    - If the transfer already exists, and the economic details have not changed[23], nothing happens: the new transfer is discarded
    - If the transfer already exists, and the economic details have changed, **it is canceled, and the new one saved**, containing the latest information
    - If the transfer does not exist, the new one is saved
  - **`ClearingTransfer PLMarks`** are created or adjusted, sourced from the transfer fees
  - **Cleared trade `PLMarks`** are created or adjusted, sourced from the `<valuation>` elements in each `<trade>`

10.2.1 Trades with flows in multiple currencies

CDML allows the same cleared `<trade>` element to contain flows in more than one currency. Given that the transfer trade grouping includes the currency, this means that **a single cleared trade can contribute to one or more clearing transfer trades**:

20 *Today this is equivalent to the product type (e.g. IRD, NDF, CDX), but **Clearing Service is aimed to be a super set of it,** not bound to products only*
21 *Using currency default's settle lag configuration*
22 *MTM flows will cancel each other*
23 *That is, flows/fees haven't changed*

its flows will be separated to match the latter's settle currency.

For instance, the following unlikely cleared trade

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-tv:tradeValuationReport modelVersion="2" version="1" generationDateTime="2014-03-
13T00:00:00-07:00" xmlns:cdml="urn:cdml:schema:common:types" xmlns:cdml-
im="urn:cdml:schema:margin:initialMargin" xmlns:cdml-
tv="urn:cdml:schema:position:tradeValuation">
    <cdml:reportDate>2014-03-13</cdml:reportDate>
    <cdml-tv:trade>
        <cdml-tv:clearedTradeId>1329237</cdml-tv:clearedTradeId>
        <cdml-tv:CCP>CME</cdml-tv:CCP>
        <cdml-tv:clearingService>IRD</cdml-tv:clearingService>
        <cdml-tv:memberId>4Q0</cdml-tv:memberId>
        <cdml-tv:positionAccountId>AAAA</cdml-tv:positionAccountId>
        <cdml-tv:segregationAccount>C</cdml-tv:segregationAccount>
        <cdml-tv:tradeCashFlowData>
            <cdml-tv:flow settleCcy="JPY" type="NPV_ADJUSTED" amount="36243675"/>
            <cdml-tv:flow settleCcy="JPY" type="NPV_REV" amount="-36271779"/>
            <cdml-tv:flow settleCcy="JPY" type="VARIATION" amount="-28104"/>
            <cdml-tv:flow settleCcy="JPY" type="PAI" amount="72"/>
            <cdml-tv:flow settleCcy="USD" type="VARIATION" amount="120.12"/>
            <cdml-tv:flow settleCcy="USD" type="PAI" amount="15.95"/>
            <cdml-tv:flow settleCcy="JPY" type="UPFRONT_FEE" amount="0"/>
            <cdml-tv:flow settleCcy="JPY" type="COUPON" amount="0"/>
        </cdml-tv:tradeCashFlowData>
    </cdml-tv:trade>
</cdml-tv:tradeValuationReport>
```

contains flows in JPY and USD. Two trades need to be created, one for each currency

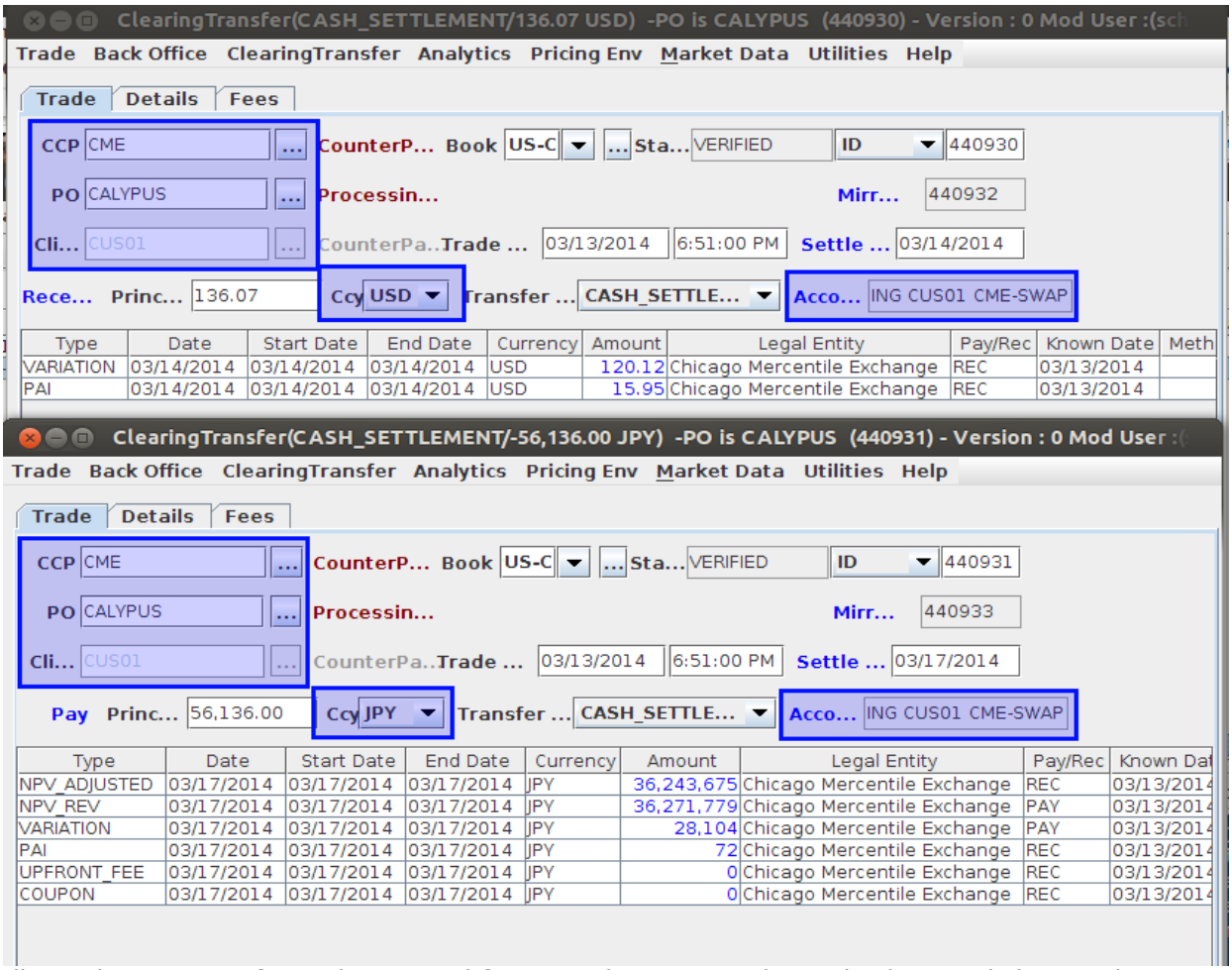


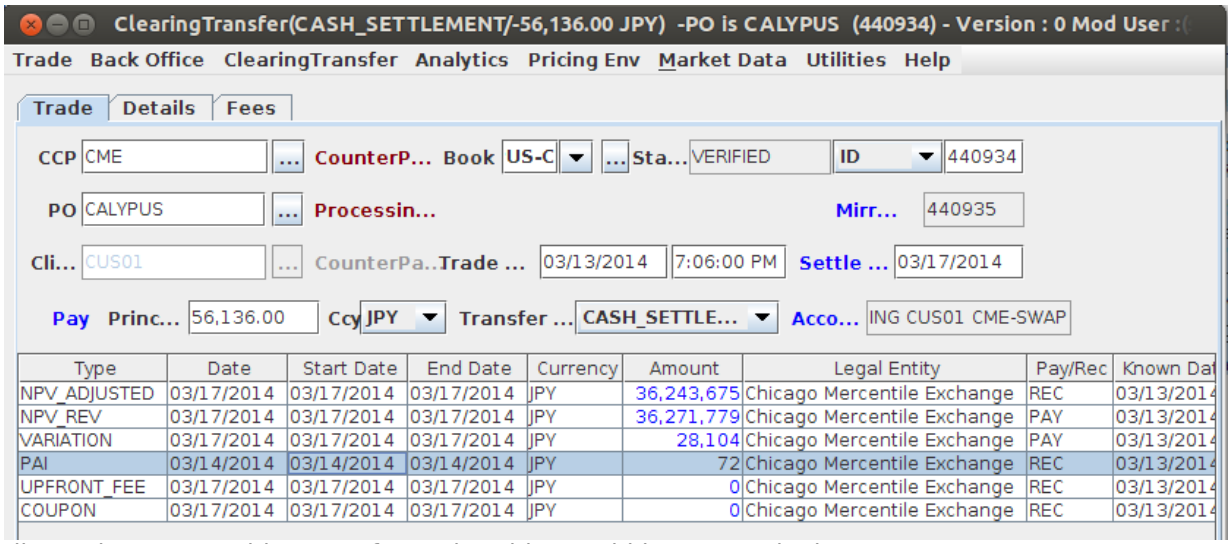
Illustration 67: Transfer trades created for JPY and USD. Note the trade characteristics are the same, except for the currency and flows. Settle dates also change, due to the fact that JPY has a different settle lag

10.2.2 Flow/fee settle date override

There are some cases in which the standard settle date calculation logic is not valid, and needs to be overridden. This can be signaled by populating the `flow/@settleDate` attribute

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-tv:tradeValuationReport modelVersion="2" version="1" generationDateTime="2014-03-
13T00:00:00-07:00" xmlns:cdml="urn:cdml:schema:common:types" xmlns:cdml-
im="urn:cdml:schema:margin:initialMargin" xmlns:cdml-
tv="urn:cdml:schema:position:tradeValuation">
    <cdml:reportDate>2014-03-13</cdml:reportDate>
    <cdml-tv:trade>
        <cdml-tv:clearedTradeId>1329237</cdml-tv:clearedTradeId>
        <cdml-tv:CCP>CME</cdml-tv:CCP>
        <cdml-tv:clearingService>IRD</cdml-tv:clearingService>
        <cdml-tv:memberId>4Q0</cdml-tv:memberId>
        <cdml-tv:positionAccountId>AAAA</cdml-tv:positionAccountId>
        <cdml-tv:segregationAccount>C</cdml-tv:segregationAccount>
```

```
        <cdml-tv:tradeCashFlowData>
            <cdml-tv:flow settleCcy="JPY" type="NPV_ADJUSTED" amount="36243675"/>
            <cdml-tv:flow settleCcy="JPY" type="NPV_REV" amount="-36271779"/>
            <cdml-tv:flow settleCcy="JPY" type="VARIATION" amount="-28104"/>
            <cdml-tv:flow settleCcy="JPY" type="PAI" amount="72"
settleDate="2014-03-14"/>
            <cdml-tv:flow settleCcy="USD" type="VARIATION" amount="120.12"/>
            <cdml-tv:flow settleCcy="USD" type="PAI" amount="15.95"/>
            <cdml-tv:flow settleCcy="JPY" type="UPFRONT_FEE" amount="0"/>
            <cdml-tv:flow settleCcy="JPY" type="COUPON" amount="0"/>
        </cdml-tv:tradeCashFlowData>
    </cdml-tv:trade>
</cdml-tv:tradeValuationReport>
```



Type	Date	Start Date	End Date	Currency	Amount	Legal Entity	Pay/Rec	Known Dat
NPV_ADJUSTED	03/17/2014	03/17/2014	03/17/2014	JPY	36,243,675	Chicago Mercentile Exchange	REC	03/13/2014
NPV_REV	03/17/2014	03/17/2014	03/17/2014	JPY	36,271,779	Chicago Mercentile Exchange	PAY	03/13/2014
VARIATION	03/17/2014	03/17/2014	03/17/2014	JPY	28,104	Chicago Mercentile Exchange	PAY	03/13/2014
PAI	03/14/2014	03/14/2014	03/14/2014	JPY	72	Chicago Mercentile Exchange	REC	03/13/2014
UPFRONT_FEE	03/17/2014	03/17/2014	03/17/2014	JPY	0	Chicago Mercentile Exchange	REC	03/13/2014
COUPON	03/17/2014	03/17/2014	03/17/2014	JPY	0	Chicago Mercentile Exchange	REC	03/13/2014

Illustration 68: Resulting transfer trade, with overridden PAI settle date

10.3 Processing the initialMarginReport

The initial margin data is already aggregated at margin account level, so there's no need for an initial classification. Here are the main steps in IM processing

- For every `<initialMarginData>` element
  - Locate the client-facing[24] MCC (see 10.3.2 Locating contracts)
    - If none found or more than one is found, an error is logged, and nothing happens
    - If only one is found, process continues with it
      - Measures are grouped by currency. See 10.3.1 Requirement vs. native margins on how the measure or measure breakdown is selected
      - Currency set is completed in case the MCC declares more *eligible currencies* than the ones found in CDML
        - Measure amounts in those currencies will be zeros (0)
      - For every currency, a collateral exposure is located
        - If none exists, a new one is created
      - `PLMarks` are created or adjusted
  - Locate the CCP-facing MCC[25], and proceed as with the client-facing one

10.3.1 Requirement vs. native margins

Before CDML, the choice of importing the margins in native or requirement currency was driven by the CLEARING_BO_MARGIN ST configuration.

With CDML, the **configuration has to be done at MCC level**, as the CLEARING_PROCESS_FROM_CDML is the same for all report types. The additional field is *IM_IMPORT_CURRENCY*

24 *An MCC whose LE is **not** the CCP*
25 *See 10.3.4 for comments on the Intraday case*

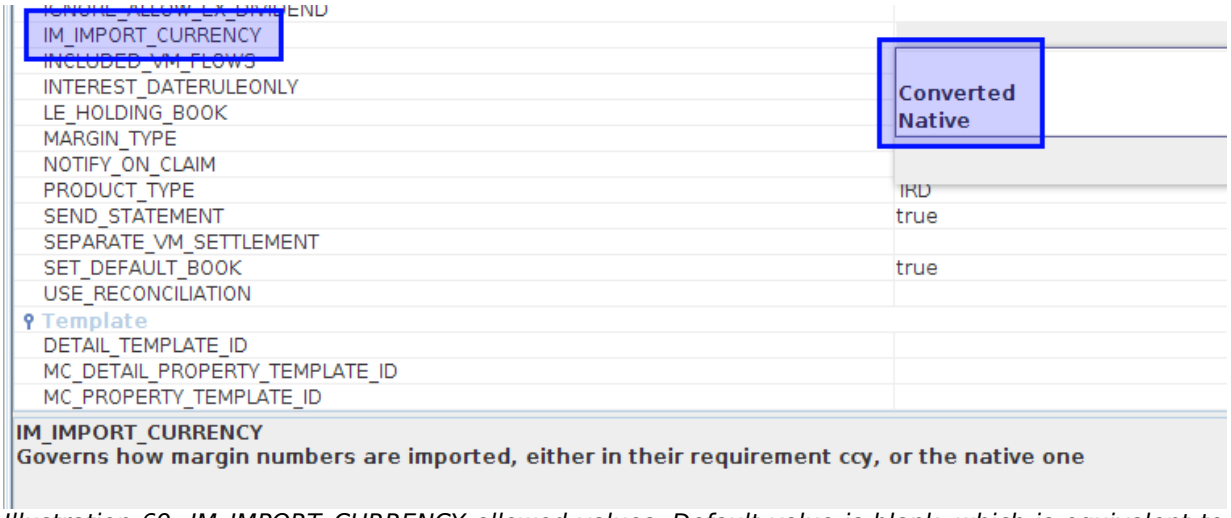
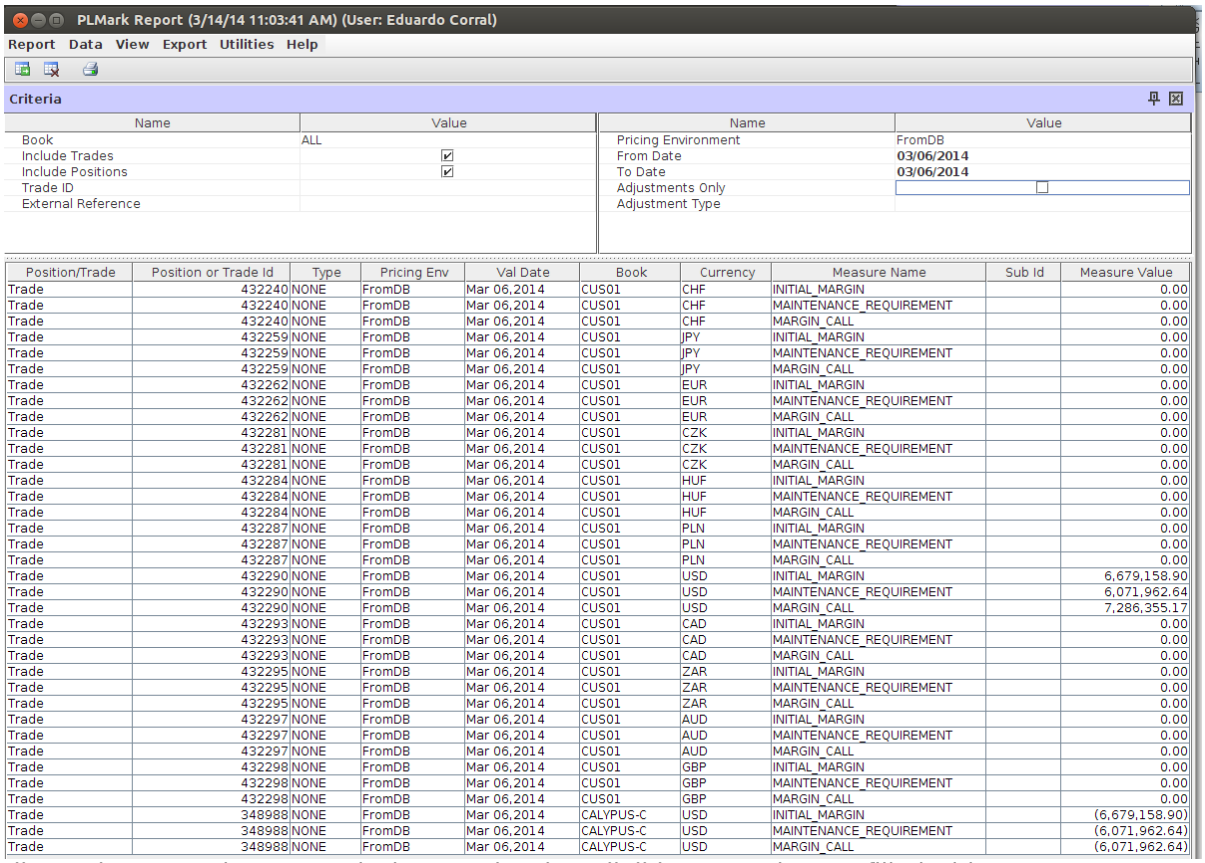


Illustration 69: IM_IMPORT_CURRENCY allowed values. Default value is blank, which is equivalent to Converted. Converted is the currency of the requirement, vs. Native, which is the activity currency

For instance, running CDML IM processing with this report

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-im:initialMarginReport modelVersion="2" version="2" generationDateTime="2014-01-28T17:33:00-08:00"
xmlns:cdml="urn:cdml:schema:common:types" xmlns:cdml-im="urn:cdml:schema:margin:initialMargin" xmlns:cdml-
tv="urn:cdml:schema:position:tradeValuation">
    <cdml:reportDate>2014-03-06</cdml:reportDate>
    <cdml-im:initialMarginData>
        <cdml-im:CCP>CME</cdml-im:CCP>
        <cdml-im:clearingService>IRD</cdml-im:clearingService>
        <cdml-im:memberId>4Q0</cdml-im:memberId>
        <cdml-im:initialMarginAccountId>AAAA</cdml-im:initialMarginAccountId>
        <cdml-im:segregationAccount>C</cdml-im:segregationAccount>
        <cdml-im:measures>
            <cdml-im:measure requirementCcy="USD" type="MAINTENANCE_REQUIREMENT" amount="6071962.6400000015">
                <cdml-im:measureBreakdown nativeCcy="JPY" amount="250135459.94" conversionFX="0.009718173"/>
                <cdml-im:measureBreakdown nativeCcy="EUR" amount="1967990.43" conversionFX="1.3668671405"/>
                <cdml-im:measureBreakdown nativeCcy="USD" amount="255594.41" conversionFX="1"/>
                <cdml-im:measureBreakdown nativeCcy="MXN" amount="98637.25" conversionFX="0.0754403832"/>
                <cdml-im:measureBreakdown nativeCcy="GBP" amount="381162.42" conversionFX="1.6583747927"/>
                <cdml-im:measureBreakdown nativeCcy="CAD" amount="62435.33" conversionFX="0.8965393581"/>
            </cdml-im:measure>
            <cdml-im:measure requirementCcy="USD" type="INITIAL_MARGIN" amount="6679158.899999999"/>
        </cdml-im:measures>
    </cdml-im:initialMarginData>
</cdml-im:initialMarginReport>
```

and the OOTB configuration (*blank/Converted*) results in USD-only marks



PLMark Report (3/14/14 11:03:41 AM) (User: Eduardo Corral)

Report Data View Export Utilities Help

Criteria

Name	Value	Name	Value
Book	ALL	Pricing Environment	FromDB
Include Trades	☑	From Date	03/06/2014
Include Positions	☑	To Date	03/06/2014
Trade ID		Adjustments Only	☐
External Reference		Adjustment Type	

Position/Trade	Position or Trade Id	Type	Pricing Env	Val Date	Book	Currency	Measure Name	Sub Id	Measure Value
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	INITIAL_MARGIN		0.00
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	MAINTENANCE_REQUIREMENT		0.00
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	MARGIN_CALL		0.00
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	INITIAL_MARGIN		0.00
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	MAINTENANCE_REQUIREMENT		0.00
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	MARGIN_CALL		0.00
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	INITIAL_MARGIN		0.00
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	MAINTENANCE_REQUIREMENT		0.00
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	MARGIN_CALL		0.00
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	INITIAL_MARGIN		0.00
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	MAINTENANCE_REQUIREMENT		0.00
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	MARGIN_CALL		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	INITIAL_MARGIN		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	MAINTENANCE_REQUIREMENT		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	MARGIN_CALL		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	INITIAL_MARGIN		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	MAINTENANCE_REQUIREMENT		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	MARGIN_CALL		0.00
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	INITIAL_MARGIN		6,679,158.90
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	MAINTENANCE_REQUIREMENT		6,071,962.64
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	MARGIN_CALL		7,286,355.17
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	INITIAL_MARGIN		0.00
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	MAINTENANCE_REQUIREMENT		0.00
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	MARGIN_CALL		0.00
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	INITIAL_MARGIN		0.00
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	MAINTENANCE_REQUIREMENT		0.00
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	MARGIN_CALL		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	INITIAL_MARGIN		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	MAINTENANCE_REQUIREMENT		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	MARGIN_CALL		0.00
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	INITIAL_MARGIN		0.00
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	MAINTENANCE_REQUIREMENT		0.00
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	MARGIN_CALL		0.00
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	INITIAL_MARGIN		(6,679,158.90)
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	MAINTENANCE_REQUIREMENT		(6,071,962.64)
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	MARGIN_CALL		(6,071,962.64)

Illustration 70: Only USD marks imported. Other eligible currencies are filled with zeros

Changing the configuration to Native

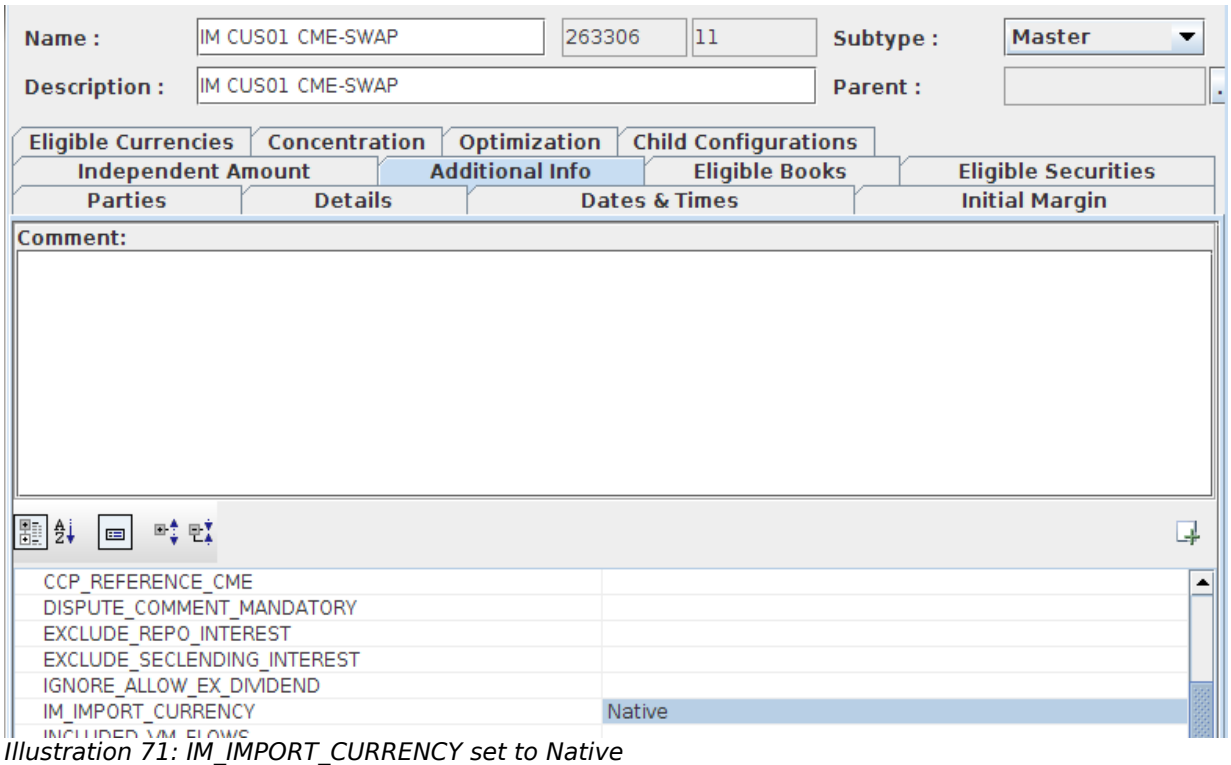
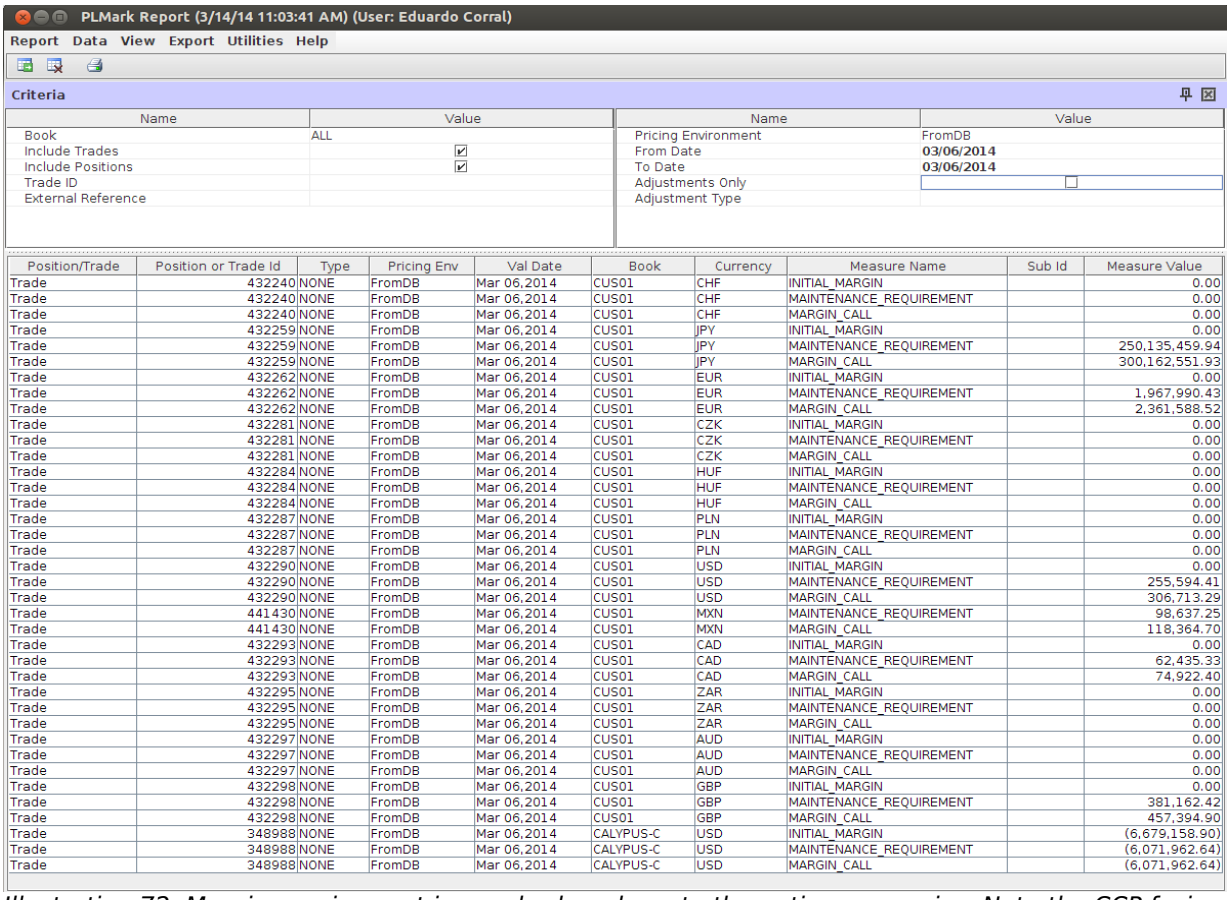


Illustration 71: IM_IMPORT_CURRENCY set to Native

makes the breakdown to be imported



Position/Trade	Position or Trade Id	Type	Pricing Env	Val Date	Book	Currency	Measure Name	Sub Id	Measure Value
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	INITIAL_MARGIN		0.00
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	MAINTENANCE_REQUIREMENT		0.00
Trade	432240	NONE	FromDB	Mar 06,2014	CUS01	CHF	MARGIN_CALL		0.00
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	INITIAL_MARGIN		0.00
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	MAINTENANCE_REQUIREMENT		250,135,459.94
Trade	432259	NONE	FromDB	Mar 06,2014	CUS01	JPY	MARGIN_CALL		300,162,551.93
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	INITIAL_MARGIN		0.00
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	MAINTENANCE_REQUIREMENT		1,967,990.43
Trade	432262	NONE	FromDB	Mar 06,2014	CUS01	EUR	MARGIN_CALL		2,361,588.52
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	INITIAL_MARGIN		0.00
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	MAINTENANCE_REQUIREMENT		0.00
Trade	432281	NONE	FromDB	Mar 06,2014	CUS01	CZK	MARGIN_CALL		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	INITIAL_MARGIN		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	MAINTENANCE_REQUIREMENT		0.00
Trade	432284	NONE	FromDB	Mar 06,2014	CUS01	HUF	MARGIN_CALL		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	INITIAL_MARGIN		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	MAINTENANCE_REQUIREMENT		0.00
Trade	432287	NONE	FromDB	Mar 06,2014	CUS01	PLN	MARGIN_CALL		0.00
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	INITIAL_MARGIN		0.00
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	MAINTENANCE_REQUIREMENT		255,594.41
Trade	432290	NONE	FromDB	Mar 06,2014	CUS01	USD	MARGIN_CALL		306,713.29
Trade	441430	NONE	FromDB	Mar 06,2014	CUS01	MXN	MAINTENANCE_REQUIREMENT		98,637.25
Trade	441430	NONE	FromDB	Mar 06,2014	CUS01	MXN	MARGIN_CALL		118,364.70
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	INITIAL_MARGIN		0.00
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	MAINTENANCE_REQUIREMENT		62,435.33
Trade	432293	NONE	FromDB	Mar 06,2014	CUS01	CAD	MARGIN_CALL		74,922.40
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	INITIAL_MARGIN		0.00
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	MAINTENANCE_REQUIREMENT		0.00
Trade	432295	NONE	FromDB	Mar 06,2014	CUS01	ZAR	MARGIN_CALL		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	INITIAL_MARGIN		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	MAINTENANCE_REQUIREMENT		0.00
Trade	432297	NONE	FromDB	Mar 06,2014	CUS01	AUD	MARGIN_CALL		0.00
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	INITIAL_MARGIN		0.00
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	MAINTENANCE_REQUIREMENT		381,162.42
Trade	432298	NONE	FromDB	Mar 06,2014	CUS01	GBP	MARGIN_CALL		457,394.90
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	INITIAL_MARGIN		(6,679,158.90)
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	MAINTENANCE_REQUIREMENT		(6,071,962.64)
Trade	348988	NONE	FromDB	Mar 06,2014	CALYPUS-C	USD	MARGIN_CALL		(6,071,962.64)

Illustration 72: Margin requirement is now broken down to the native currencies. Note the CCP-facing ones (CALYPUS-C) haven't been updated, as the CCP-facing MCC hasn't been reconfigured

Running the process the same day, with different contract configuration, is not supported. It has been done here just for the purpose of showing both results, and it is not guaranteed to work.

10.3.2 Locating contracts

Client- and CCP-facing MCCs are located in a similar way. Given an `<initialMarginData>` element, such as

```
<cdml-im:initialMarginData>
    <cdml-im:CCP>CME</cdml-im:CCP>
    <cdml-im:clearingService>IRD</cdml-im:clearingService>
    <cdml-im:memberId>4Q0</cdml-im:memberId>
    <cdml-im:initialMarginAccountId>AAAA</cdml-im:initialMarginAccountId>
    <cdml-im:segregationAccount>C</cdml-im:segregationAccount>
    ...
```

it is expected that only two contracts in the system will match such configuration: a client-facing one, and a CCP-facing one.

The **common** part of the loading retrieves all contracts

- For the given PO (`<memberId>`)
- With *CCP* contract additional field equal to the `<CCP>` element
- With *MARGIN_TYPE* additional field set to IM
- With *PRODUCT_TYPE* additional field equal to the `<clearingService>` element

In addition, when locating **CCP-facing** contracts

- Only contracts **whose LE is the CCP** are kept
- The **CCP_REFERENCE** additional field has to match the **`<segregationAccount>`** element

When locating **client-facing** contracts

- Only contracts **whose LE is not the CCP** are kept
- The **CCP_REFERENCE** additional field has to match the **`<initialMarginAccountId>`** element

The contracts found for the snippet above would look like

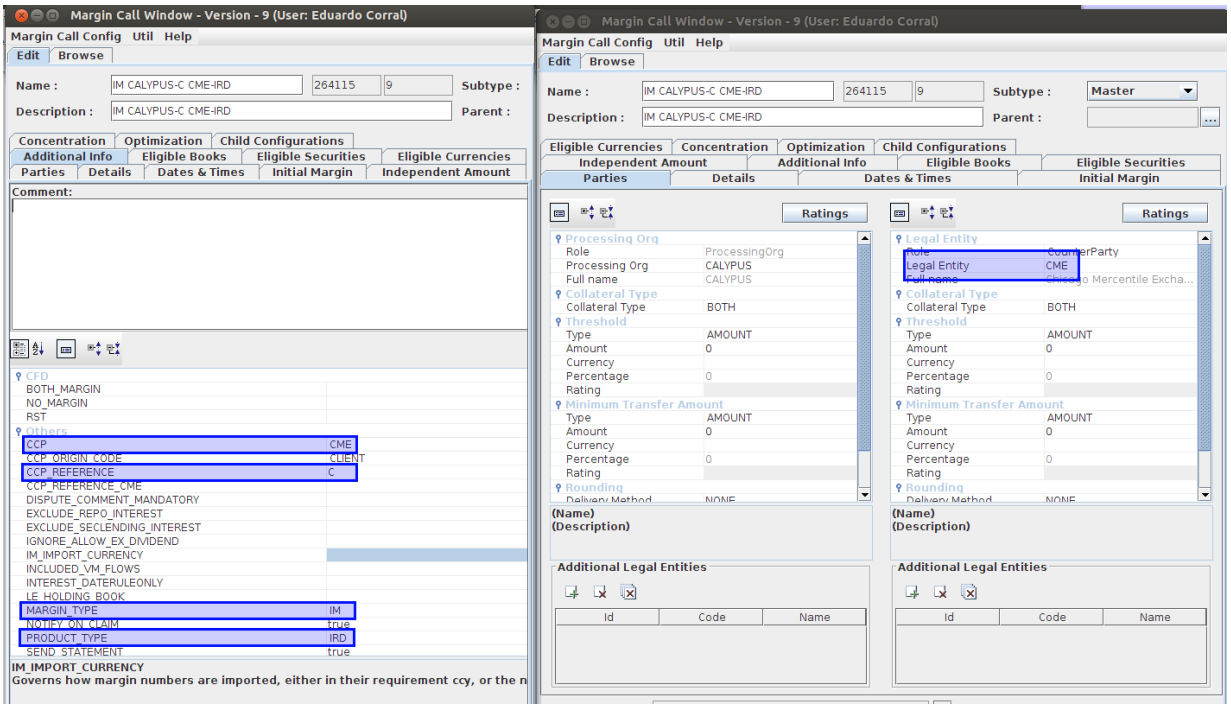


Illustration 73: CCP-facing contract. Note the CME LE, and `CCP_REFERENCE=C`

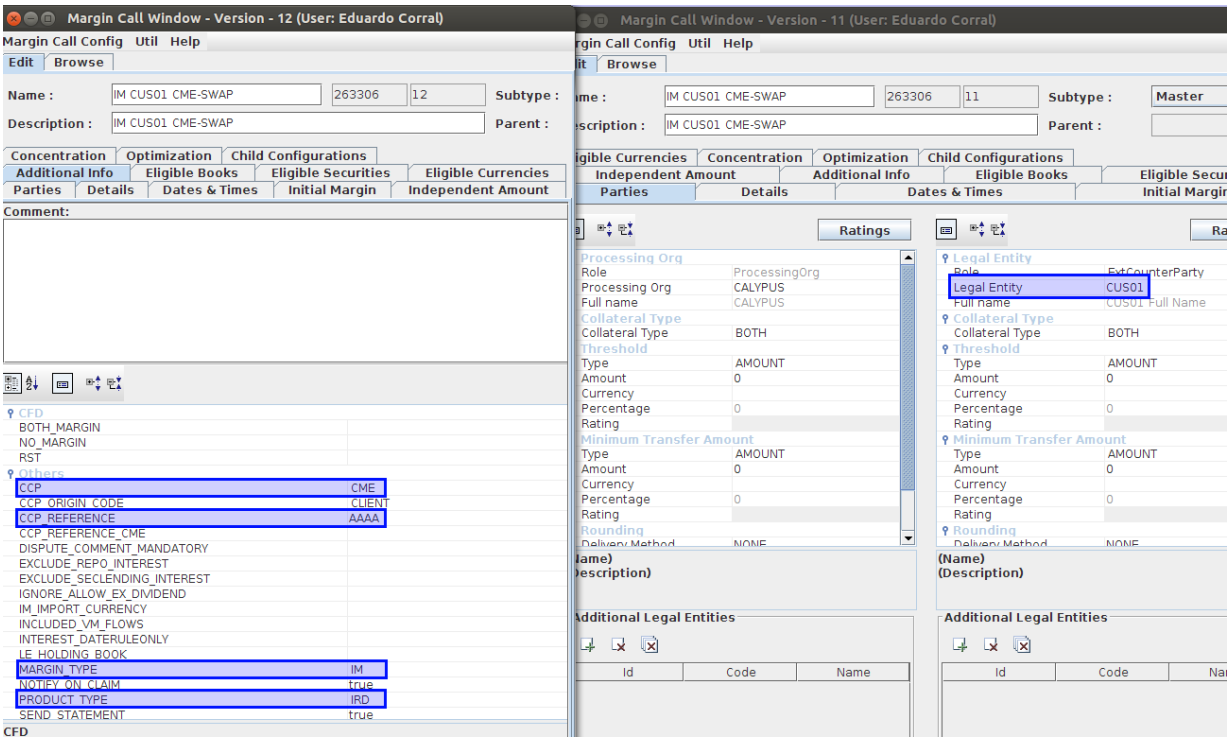


Illustration 74: Client-facing contract, with a different LE, and `CCP_REFERENCE=AAAA`

10.3.3 MARGIN_CALL measure

As seen in Illustration 70 or Illustration 72, the `MARGIN_CALL PLMark` value is being stored, but **it is not coming from CDML**. That is to honor the legacy pre-CDML clearing setup: `MARGIN_CALL` is computed as

```
MARGIN_CALL = MAINTENANCE_REQUIREMENT * MCC Credit Multiplier
```

`MARGIN_CALL` is the measure used in Collateral Valuation, and it won't be auto-computed, so it has to be recorded during CDML processing.

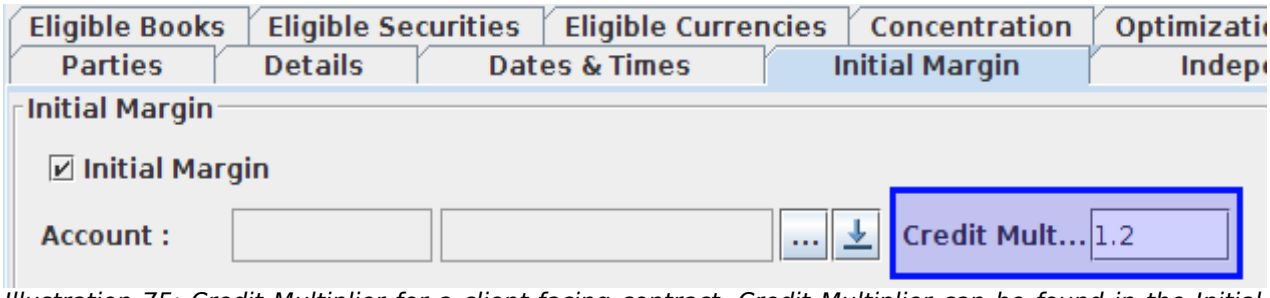


Illustration 75: Credit Multiplier for a client-facing contract. Credit Multiplier can be found in the Initial Margin MCC tab

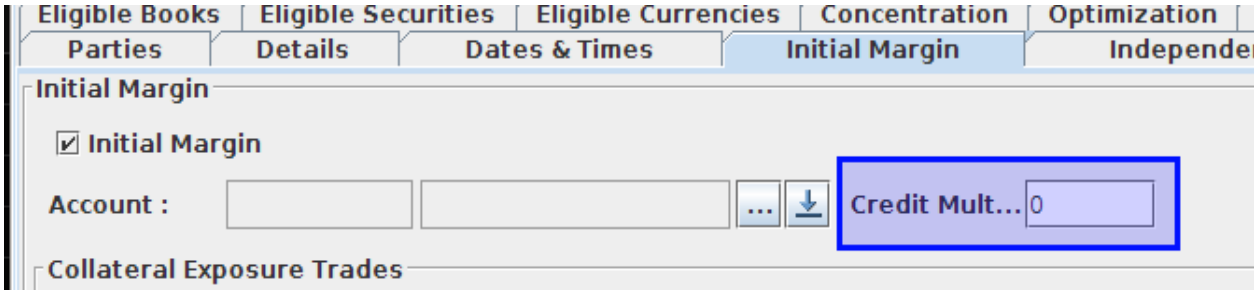


Illustration 76: CCP-facing contract multiplier. ***OOTB value is 0****, so, in terms of the* `MARGIN_CALL` *computation,* ***it's considered to be 1****: CCP-facing contracts encode margins as requested by the CCP, so it doesn't make sense to apply a multiplier, that would cause* `MAINTENANCE_REQUIREMENT` *and* `MARGIN_CALL` *to be different*

10.3.4 Intraday processing

When the `intraday` flag introduced in 7.2.1 is present in the CDML IM report, the following behavior changes occur

- The **multiplier** mechanism described in 10.3.3 **is disabled** OOTB. To force the multiplier to be applied ITD, the PO LE must be configured with the attribute **`ApplyBufferITD=true`**
- **Only client-facing exposures are created/updated**. CCP-facing contracts are ignored
- MCC ITD pricing environment is used for the `PLMark` creation, and mark values are tagged with an "`ITD`" comment

11 Appendix A: messaging (logging and TaskStation)

All CDML code, both core translation framework, processing, and everything in between, uses `MessageSink` to log information.

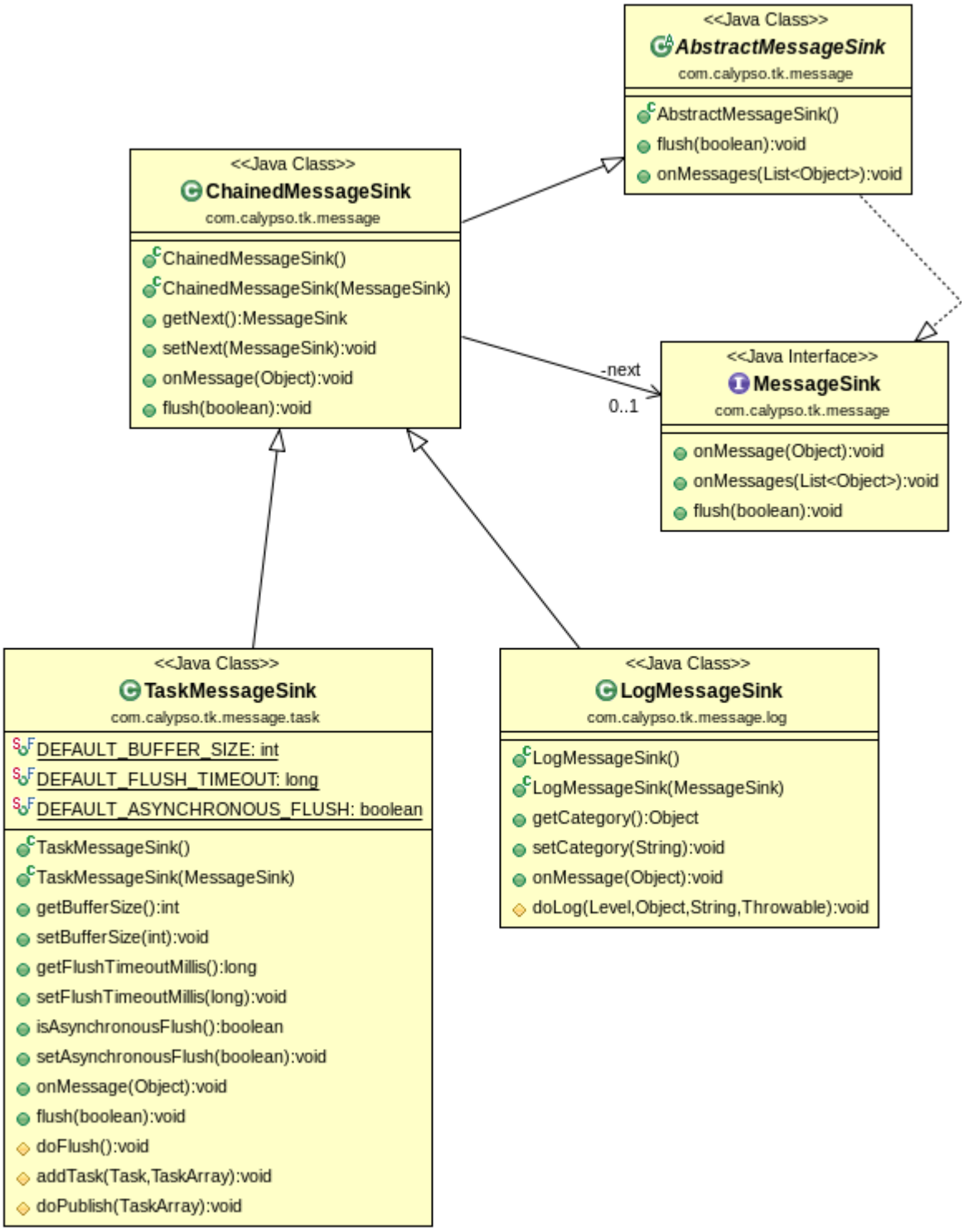


Illustration 77: `MessageSink` hierarchy, showing the two main implementations, `TaskMessageSink` and `LogMessageSink`

Logging tends to be repetitive and include sizable chunks of copy-and-paste, and useful

logging usually implies aggregating non-trivial amounts of information, in place, which distracts the reader from the actual processing code.

`MessageSink` was created to address this, and other shortcomings

- **Abstraction from underlying logging framework** (yet another): CDML core code started as a module that could be distributed *separate from Calypso*, w/o any guarantee to have access to `Log` and related classes
- **Performance**: current logging solution requires to compose the `String` argument before logging. It is preferable to pass the raw objects to be logged, and defer the composing to later stages. Sometimes that logging could even not happen (e.g. `DEBUG` not active), or it could be done in a separate thread
- **Messaging and API uniformity between log files, and Task Station**: most CDML produced messages will look the same in Task Station and in the log file, which allows better tracking of what's going behind the scenes, if the log file is ever needed. Also, API calls are the same, regardless the destination
- **Easier, cleaner logging**: by encapsulating common operations, and simplifying the use of `String` formatting

```
CDMLReportType report = cdmlBackend.loadLatestInstance(reportType, reportDate);
if (report != null) {
    doProcessSingleReport(report, sink);
} else {
    success = false;
    sink.onMessage(MessageSinkFactory.createError(MISSING_REPORT_ERROR_MESSAGE, reportType, reportDate));
}
```

Illustration 78: Example of sinking a variable error message, with two parameters, `reportType` and `reportDate`

11.1 MessageSinkConfigurable

`MessageSinkConfigurable` instances delegate their logging/messaging on the `MessageSink` they're configured with. Also, **most of them will pass such `MessageSink` to their instance variables**, if they happen to be `MessageSinkConfigurable` too.

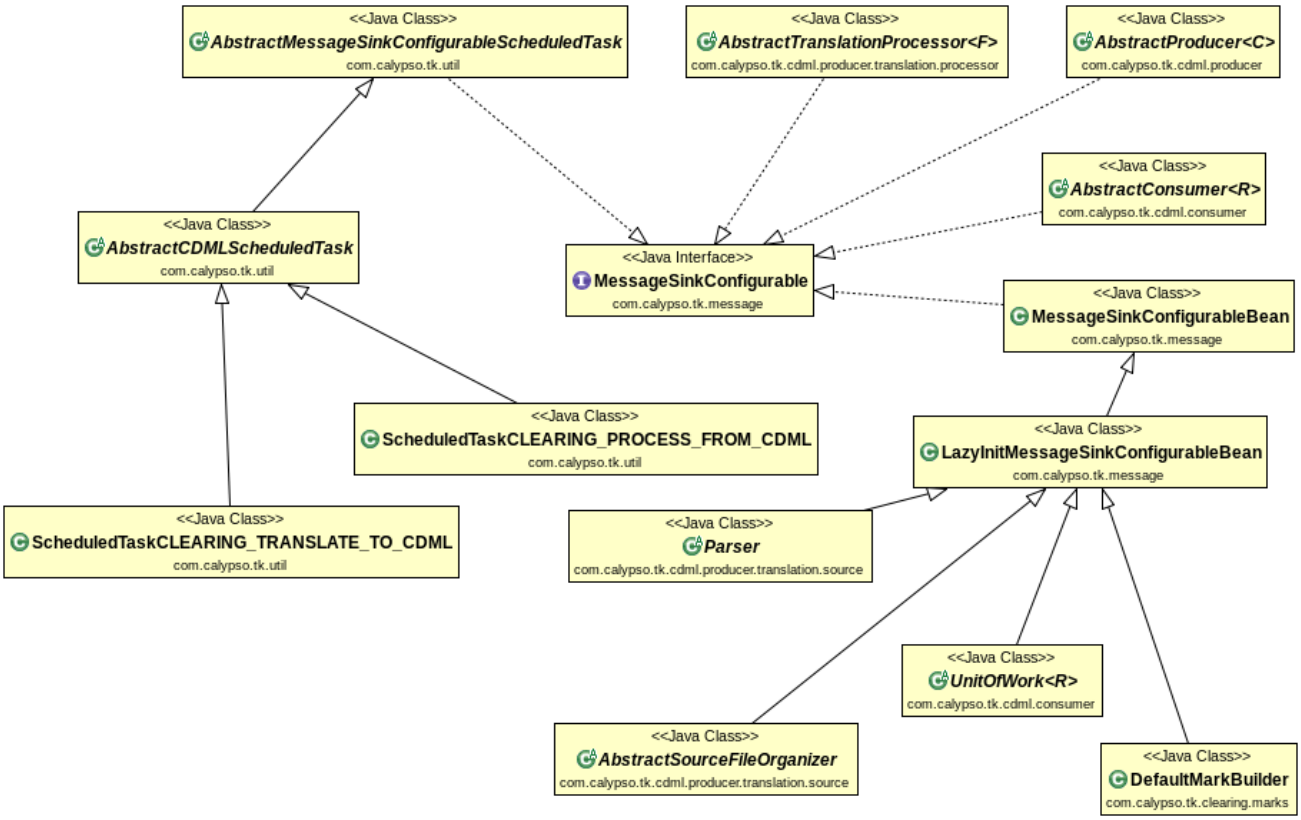


Illustration 79: `MessageSinkConfigurable` partial hierarchy

Both CDML producing and consuming scheduled tasks are `MessagingSinkConfigurable` themselves, and they pass the `MessageSink` to all downstream processing classes, thus making it the **unified sink for most CDML processing**.

11.2 CDML scheduled task MessageSink

Illustration 80: `ScheduledTaskMessageSink` hierarachy detail

As mentioned in 11.2, the CDML scheduled tasks unify all processing logging/messaging by using a common `MessageSink`.

```
/**
 * Will lazy-init if not configured previously
 *
 * @return
 */
public MessageSink getMessageSink() {
    if (messageSink == null) {
        // Chain logging and task generation
        LogMessageSink logSink = new LogMessageSink();
        logSink.setCategory(LOG_CATEGORY);
        ScheduledTaskMessageSink taskSink = new ScheduledTaskMessageSink();
        taskSink.setScheduledTask(this);
        // No debug messages in TaskStation, and minimize duplicates
        taskSink.setFilters(new LogLevelFilter(CalypsoLogMessage.Level.INFO), new LRUHashCodeFilter());
        logSink.setNext(taskSink);
        messageSink = logSink;
    }
    return messageSink;
}
```

Illustration 81: `MessageSink` that all the ST based CDML processing share

As seen in the illustration above, such sink is composed of

- A logging one: every message, regardless it's type, will be passed to the underlying Calypso Log framework
- A chained TaskStation-capable one, which will filter out debug or trace messages

12 Appendix B: CDMLViewer

The `CDMLViewer` is the evolution of the previous `ClearingDataViewer`, but simplified, due to the simple nature of CDML. **It shows all CDML instances stored in DB, for a given date range**.

The action for MainEntry/Navigator Customizer is `cdml.CDMLViewerFrame`.

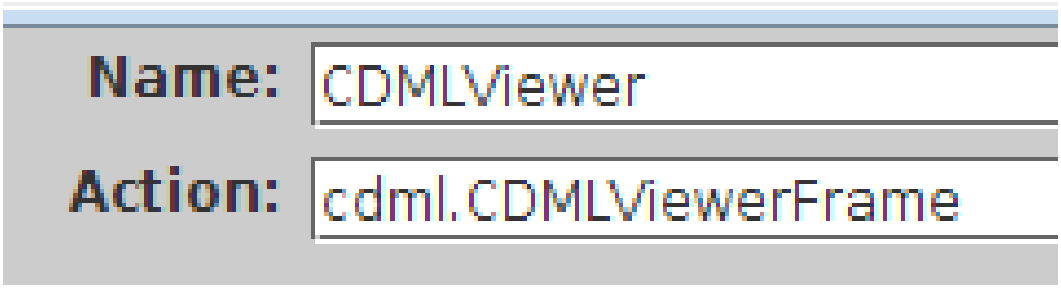


Illustration 82: Added as `CDMLViewer` to customizer

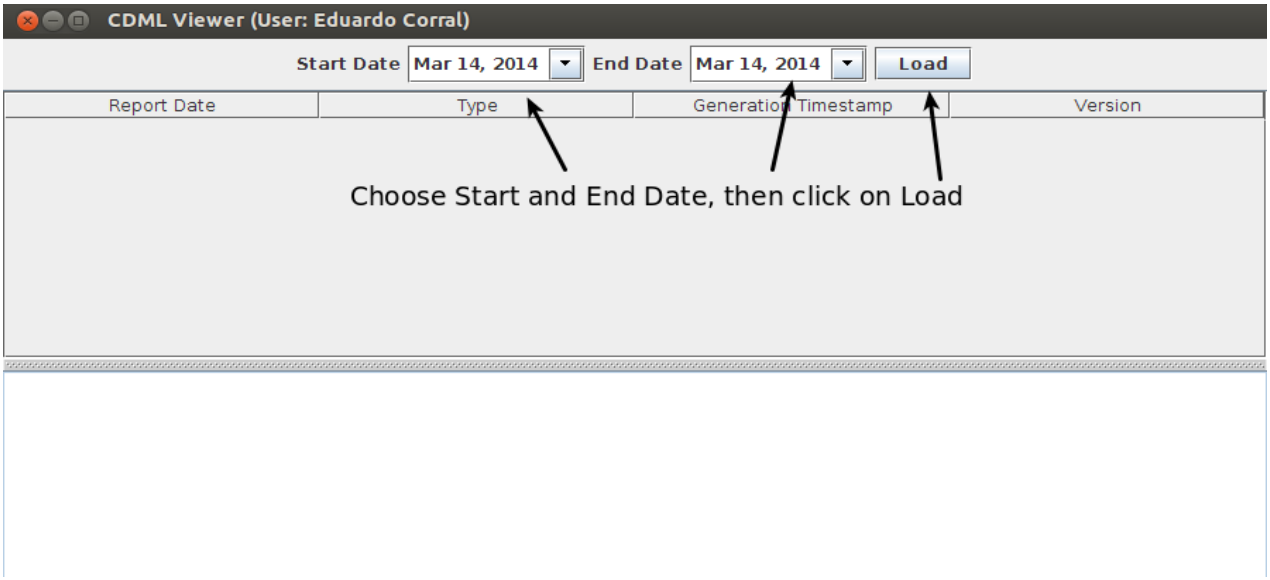
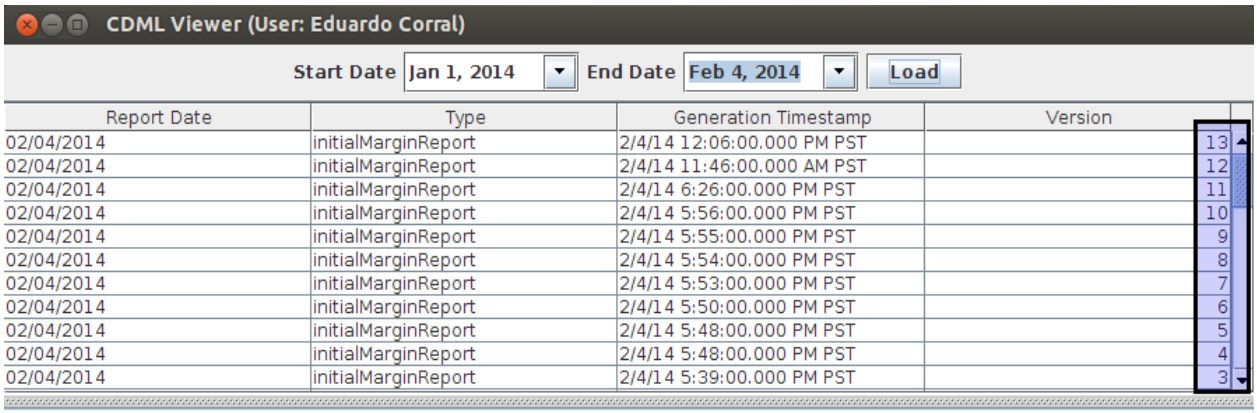
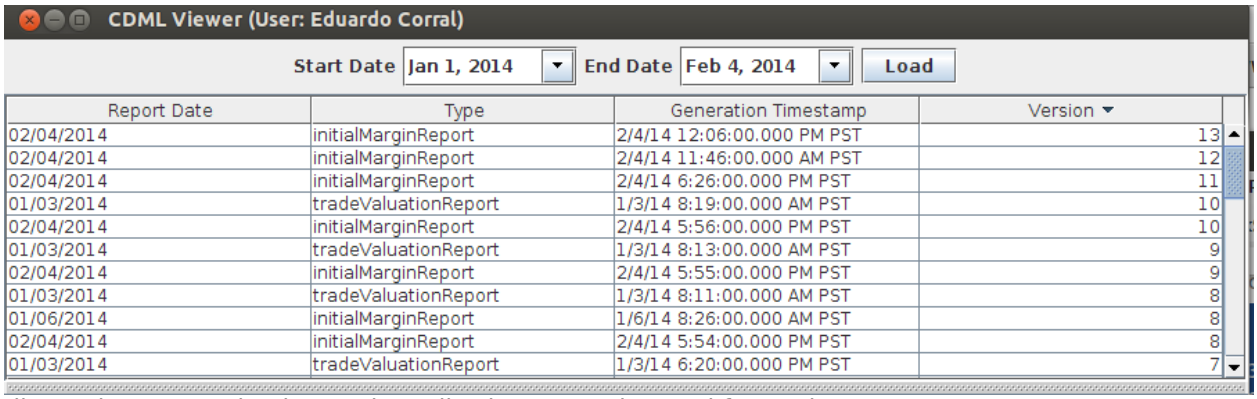


Illustration 83: Viewer loading usage



Report Date	Type	Generation Timestamp	Version
02/04/2014	initialMarginReport	2/4/14 12:06:00.000 PM PST	13
02/04/2014	initialMarginReport	2/4/14 11:46:00.000 AM PST	12
02/04/2014	initialMarginReport	2/4/14 6:26:00.000 PM PST	11
02/04/2014	initialMarginReport	2/4/14 5:56:00.000 PM PST	10
02/04/2014	initialMarginReport	2/4/14 5:55:00.000 PM PST	9
02/04/2014	initialMarginReport	2/4/14 5:54:00.000 PM PST	8
02/04/2014	initialMarginReport	2/4/14 5:53:00.000 PM PST	7
02/04/2014	initialMarginReport	2/4/14 5:50:00.000 PM PST	6
02/04/2014	initialMarginReport	2/4/14 5:48:00.000 PM PST	5
02/04/2014	initialMarginReport	2/4/14 5:48:00.000 PM PST	4
02/04/2014	initialMarginReport	2/4/14 5:39:00.000 PM PST	3

Illustration 84: CDML is never deleted: ***new versions are stored every time*** *CLEARING_TRANSLATE_TO_CDML, or other* `CDMLBackend` *save operation, are invoked.* ***Only the latest version is used by downstream processes*** *(e.g. CLEARING_PROCESS_FROM_CDML)*



Report Date	Type	Generation Timestamp	Version
02/04/2014	initialMarginReport	2/4/14 12:06:00.000 PM PST	13
02/04/2014	initialMarginReport	2/4/14 11:46:00.000 AM PST	12
02/04/2014	initialMarginReport	2/4/14 6:26:00.000 PM PST	11
01/03/2014	tradeValuationReport	1/3/14 8:19:00.000 AM PST	10
02/04/2014	initialMarginReport	2/4/14 5:56:00.000 PM PST	10
01/03/2014	tradeValuationReport	1/3/14 8:13:00.000 AM PST	9
02/04/2014	initialMarginReport	2/4/14 5:55:00.000 PM PST	9
01/03/2014	tradeValuationReport	1/3/14 8:11:00.000 AM PST	8
01/06/2014	initialMarginReport	1/6/14 8:26:00.000 AM PST	8
02/04/2014	initialMarginReport	2/4/14 5:54:00.000 PM PST	8
01/03/2014	tradeValuationReport	1/3/14 6:20:00.000 PM PST	7

Illustration 85: Sorting by version. All columns can be used for sorting

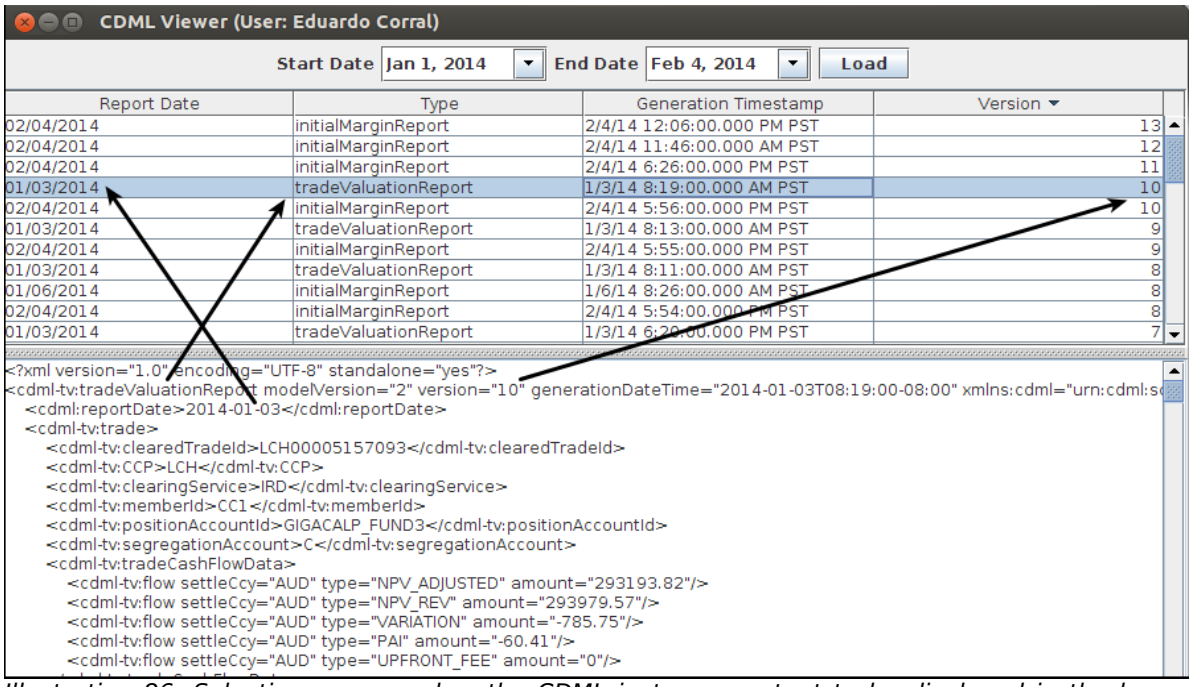


Illustration 86: Selecting a row makes the CDML instance content to be displayed in the lower panel

13 Appendix C: CDML translator distribution and artifact structure

Although not mandatory, it is highly recommended to package CDML translator distribution following the standard approach, which means

- Single module project
- One artifact per distribution[26]
- `<translator name>CDMLSchemaData.xml ExecuteSQL` file in bin/dbscripts, declaring the translator name

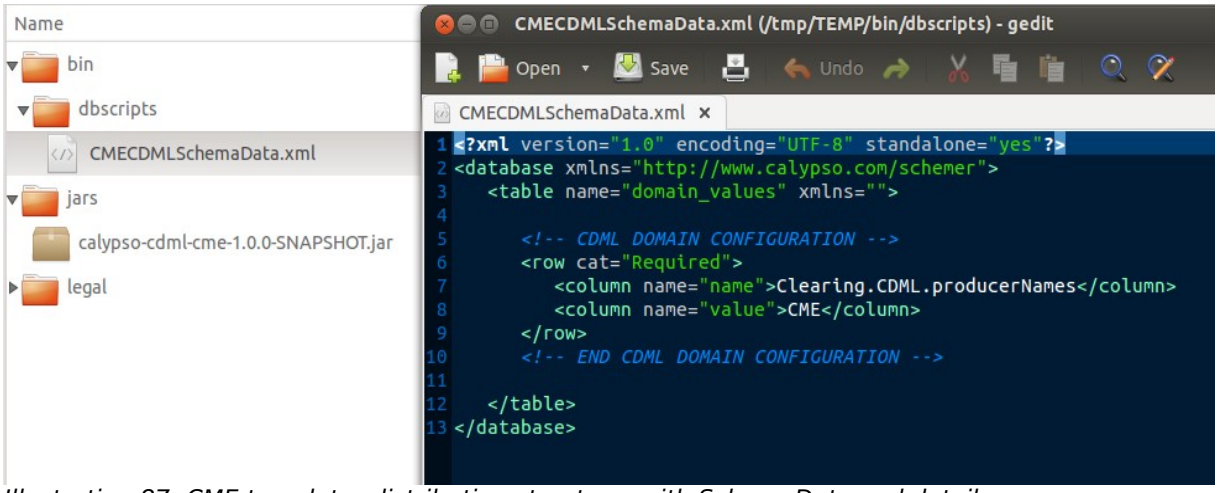


Illustration 87: CME translator distribution structure, with SchemaData.xml detail

Translator artifact (read: jar) structure is more lenient. The only true requisite is to pack the Spring context at the artifact's root

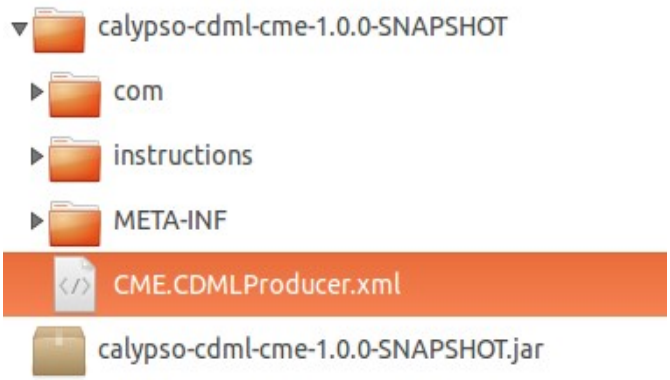


Illustration 88: Exploded artifact

26 *Not counting **approved** 3rd party dependencies*

14 Appendix D: CDML translator deploy strategies

14.1 Calypso V13

Calypso V13 instances are flexible enough to allow any deployment desired. The simplest one is to unzip the distribution proposed in 13 Appendix C: CDML translator distribution and artifact structure on the Calypso folder, and add the translator artifact to the Calypso classpath.

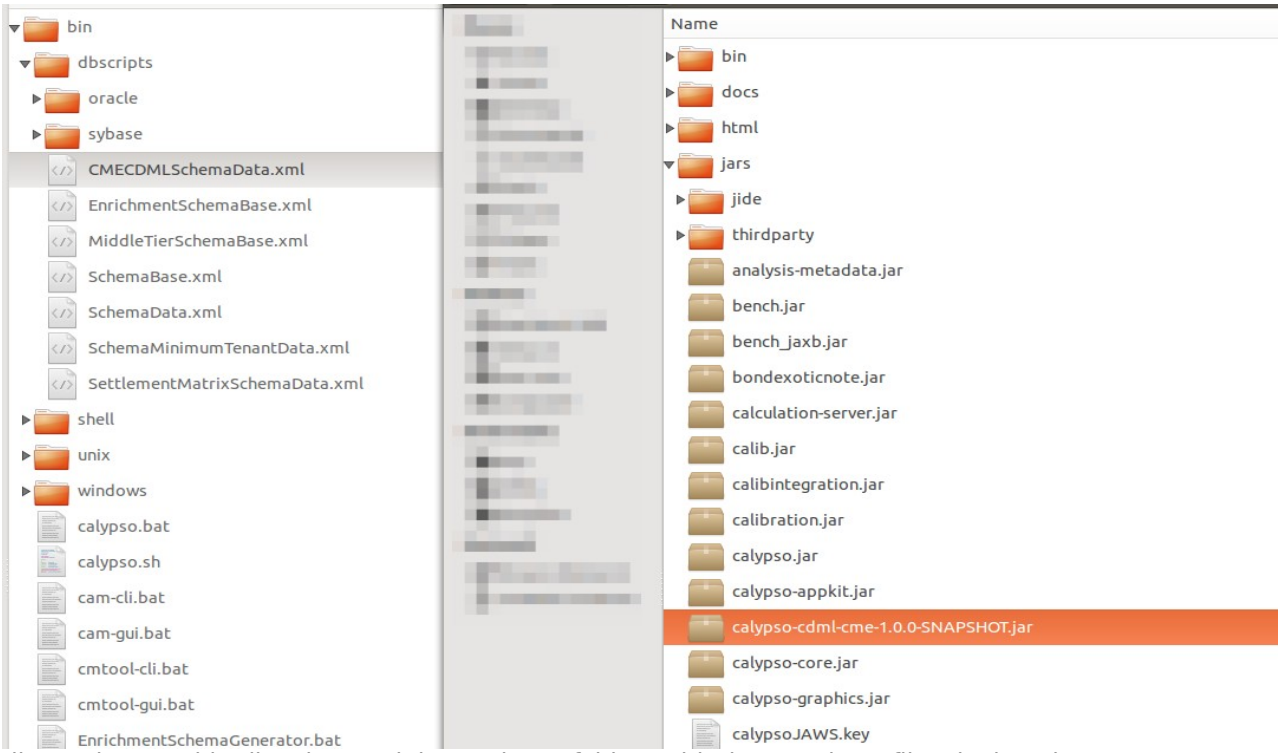


Illustration 89: /bin/dbscripts and /jars Calypso folders with the translator files deployed

```
177 :$CALYPSO_LIB/cws.jar\
178 :$CALYPSO_LIB/mbs.jar\
179 :$CALYPSO_LIB/calypso-scheduling-engine.jar\
180 :$CALYPSO_LIB/calypso-cdml-*\
181 :$GALAPAGOS\
182 :$VISOKIO\
183 :$SWIFTSIMULATOR\
```

Illustration 90: `calypso.sh` classpath script edited, with the suggested wildcard addition. This way, all translators following standard naming practices will be included

14.2 Calypso V14 and after

Calypso V14 is more strict in terms of deployment. Two approaches are available: patching the translation distribution, or using the `custom-extensions` mechanism.

14.2.1 Patching the distribution

This mechanism is the **recommended** one **when generating the distribution zip is not an issue** (e.g. with access to the *Calypso Build System*). The patching is then trivial

```
~/calypso-14.0.0.22.SP2$ ./patch.sh /tmp/calypso-cdml-cme-1.0.0-SNAPSHOT-rel.zip
Executing task: patch
[19:24:49.017]  Currently installed modules: [calypso-ers-14.0.0.22.SP2, calypso-position-
keeping-1.0.9, calypso-exchange-feed-cme-2.1.2-14.0.0.21.SP2, calypso-ateo-1.0.0-SNAPSHOT,
calypso-bloomberg-1.5.1, calypso-exchange-feed-core-2.0.8-14.0.0.21.SP2, calypso-module-
distribution-14.0.0.22.SP2, calypso-liquidity-1.2.3, calypso-datauploader-3.1.24-
14.0.0.22.SP2.HRC-SNAPSHOT, calypso-exchange-feed-lch-2.0.6-14.0.0.18, calypso-clearing-
3.2.0, calypso-collateral-1.6.1.1-14.0.0.22.SP2, calypso-cdsisdamodel-14.0.0.22.SP2]
[19:24:49.035]  Using Calypso patcher version 1.0.4
[19:24:49.371]  Patching calypso-cdml-cme-1.0.0-SNAPSHOT-rel.zip
[19:24:49.388]  Do you want to continue? [Y/n]
```

14.2.2 Using custom-extensions

Refer to V14 documentation before attempting this approach.

If the *Calypso Build System* (read: Jenkins, Nexus) is not available, it is possible to create and patch the distribution using the build system that ships with Calypso V14.

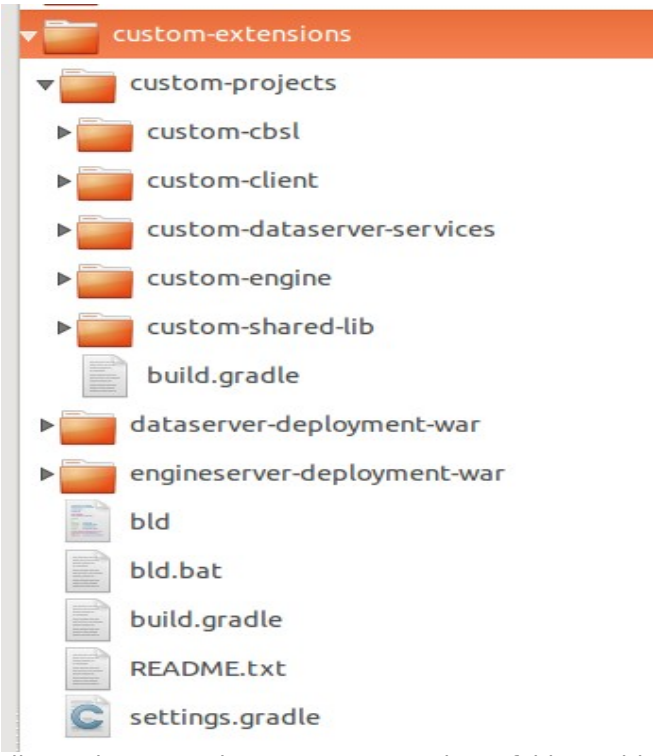


Illustration 91: The custom-extensions folder, with detail on custom-projects

Given that the CDML translators are meant to be run by the CLEARING_TRANSLATE_TO_CDML `ScheduledTask`, they can be modeled after the **custom-client sample project**.

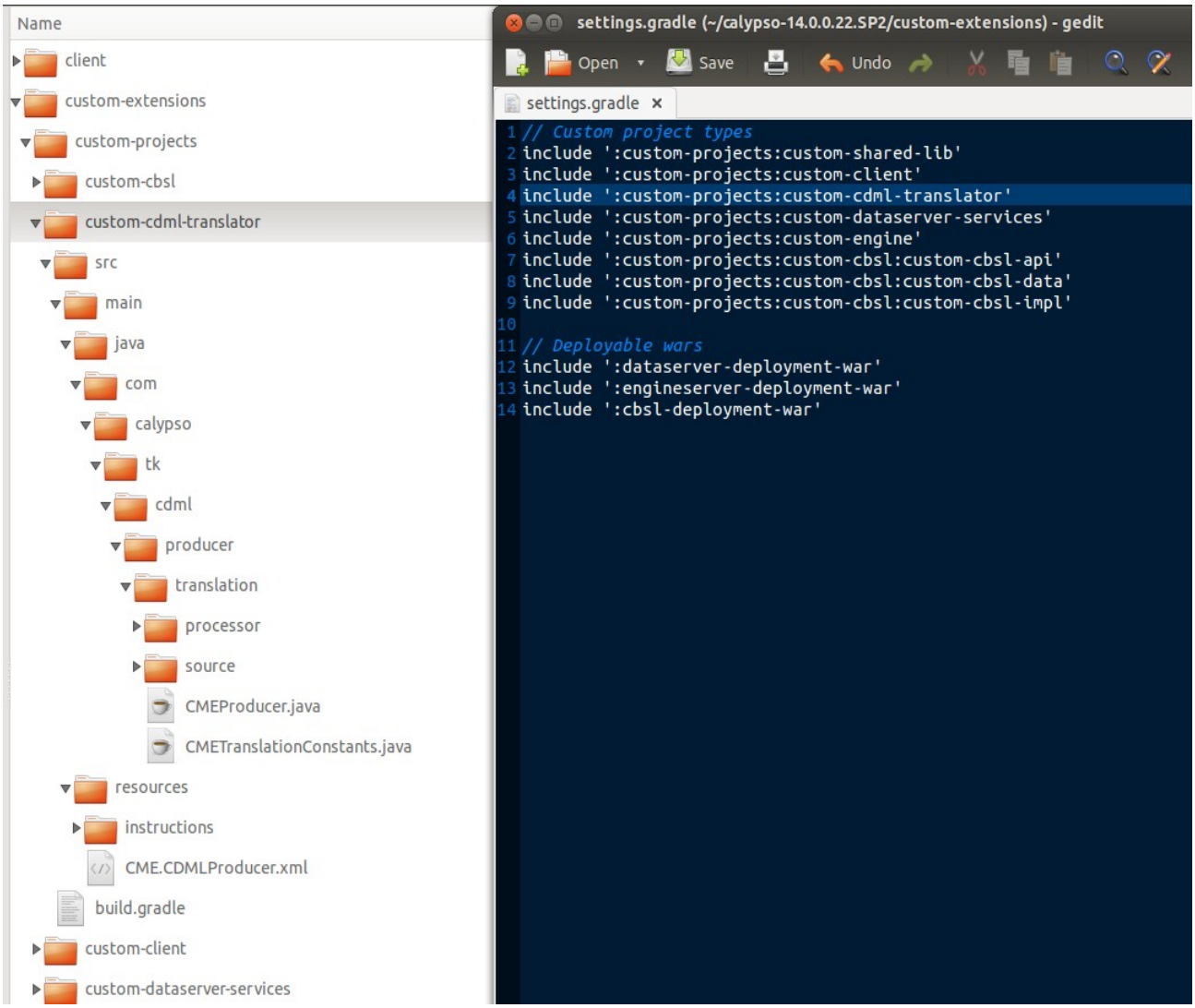


*Illustration 92: `custom-cdml-translator` project. Structure and `build.gradle` file have been copied from `custom-client`, and the CME sources over it, **as demo content**. Also, detail on the `custom-extensions/settings.gradle`, with the new project added to the build process*

The latest clearing distribution must have been already patched, either individually, or as the latest CUP, **before building the custom projects**. The clearing distribution includes the *core CDML classes*, and these need to be part of the build classpath.

For the purpose of this demo, the unneeded projects can be commented out from `settings.gradle` to speed up the build process

```
// Custom project types
include ':custom-projects:custom-shared-lib'
//include ':custom-projects:custom-client'
include ':custom-projects:custom-cdml-translator'
include ':custom-projects:custom-dataserver-services'
include ':custom-projects:custom-engine'
//include ':custom-projects:custom-cbsl:custom-cbsl-api'
```

```
//include ':custom-projects:custom-cbsl:custom-cbsl-data'
//include ':custom-projects:custom-cbsl:custom-cbsl-impl'

// Deployable wars
include ':dataserver-deployment-war'
include ':engineserver-deployment-war'
include ':cbsl-deployment-war'
```

From the custom-extensions folder, deploy is run[27]

```
:~/calypso-14.0.0.22.SP2/custom-extensions$ ./bld deploy
:unpackWars UP-TO-DATE
:custom-projects:custom-shared-lib:compileJava UP-TO-DATE
:custom-projects:custom-shared-lib:processResources UP-TO-DATE
:custom-projects:custom-shared-lib:classes UP-TO-DATE
:custom-projects:custom-shared-lib:jar UP-TO-DATE
:custom-projects:custom-cdml-translator:compileJava UP-TO-DATE
:custom-projects:custom-cdml-translator:processResources UP-TO-DATE
:custom-projects:custom-cdml-translator:classes UP-TO-DATE
:custom-projects:custom-cdml-translator:jar UP-TO-DATE
:custom-projects:custom-dataserver-services:compileJava UP-TO-DATE
:custom-projects:custom-dataserver-services:processResources UP-TO-DATE
…
SCRIPT SUCCESSFUL

BUILD SUCCESSFUL

Total time: 3 mins 20.751 secs
```

Now the client side is patched with the custom project

```
ecorral@demeter-pro:~/calypso-14.0.0.22.SP2$ ls -l client/lib/*custom-cdml*
-rw-r--r-- 1 ecorral ecorral 13849 Mar 14 20:02 client/lib/custom-cdml-translator-1.0.0-
SNAPSHOT.jar
```

27 *Truncated output, too long to display here*

15 Appendix E: CDML versioning

CDML is a **versioned specification**: as it evolves and changes, the need to identify the current version, and if a given CDML instance abides by it, will become mandatory.

The CDML version is a **short integer number**, although it can be represented in several ways, to improve readability or file naming.

The model/specification version is not to be confused with a given CDML report instance version. See 7.2.1 urn:cdml:schema:common:types for details.

15.1 Version as defined in the specification

As seen in 7.2.1, the `modelVersion` attribute declares **the specification version under which the CDML instance was created**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cdml-im:initialMarginReport modelVersion="2" version="1" generationDateTime="2014-03-
11T00:00:00-07:00" xmlns:cdml="urn:cdml:schema:common:types" xmlns:cdml-
im="urn:cdml:schema:margin:initialMargin" xmlns:cdml-
tv="urn:cdml:schema:position:tradeValuation">
    <cdml:reportDate>2014-03-13</cdml:reportDate>
    <cdml-im:initialMarginData>
...
```

The attribute is modeled after

```
<simpleType name="ReportModelVersionType">
   <restriction base="short">
      <minInclusive value="0" />
   </restriction>
</simpleType>
```

As a restricted `short` datatype, allowed values are the ones between 0 and 32767.

15.2 Version in human readable form

When displayed in a document meant for human consumption, the *"v"* character is prepended to the actual version number, and **no left padding is used**. The headers of this document show an example of this representation (*CDML v3 spec*).

15.3 Version when naming XSD and other CDML related files

For easy sorting purposes, the version number is **left padded with zeros, up to three (3) characters**, when naming the XSDs schemas, or other related documents.

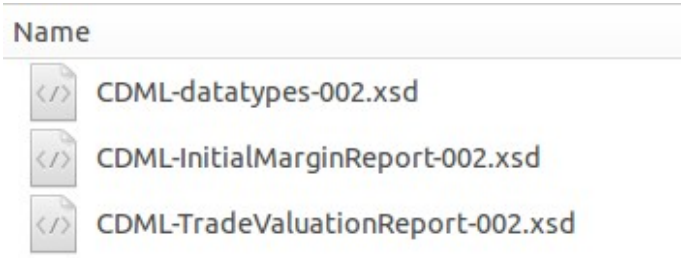


Illustration 93: CDML v2 schema files

15.4 CDML version history

`modelVersion`	Human readable form	File naming form	Version highlights	Clearing release[28]
0	v0	000	Internal development release	N/A
1	v1	001	Initial release	2.7.0/3.2.0
2	v2	002	EMIR changes	2.9.0/3.4.0
3	v3	003	Intraday support	2.11.0/3.6.0

28 *Clearing release that first included the given version*

16 Related documentation

CDML autogenerated documentation in Box	https://calypso.box.com/s/al7tz1v2z4djifstsavx
CDML Processing rules in Box	https://calypso.box.com/s/khx18p3sfew9ttj4uvdx
CDML Translators in Box	https://calypso.box.com/s/jyg5yqq367bqt4tza0ib
W3C XML Essentials	http://www.w3.org/standards/xml/core
W3C XML Schema	http://www.w3.org/standards/xml/schema
Spring 3.0.X reference	http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/
Java API for XML processing	http://docs.oracle.com/javase/tutorial/jaxp/